UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Reconocimiento de objetos en video
Autores: Bernarda Albanesi y Nadia Funes
Director: Prof. Lic. Laura Lanzarini
Codirector: Prof. Lic. Franco Chichizola
Carrera: Licenciatura en Sistemas

Resumen

El reconocimiento de objetos en video sigue siendo un tema de actualidad y ha motivado el desarrollo de diversos métodos informáticos para tratar de resolverlo. Uno de los enfoques más utilizados radica en la caracterización de los pixeles más relevantes de la imagen. Si bien en esta dirección existen distintas soluciones, en todos los casos se busca expresar la información referida a la vecindad de dichos puntos de la forma más invariante posible. El método SIFT (Scale-Invariant Feature Transform) es, sin duda, uno de los más utilizados. Sin embargo, su costo computacional es alto si se piensa en su aplicación en tareas tales como tracking de video.

El objetivo central de esta tesina es proponer una implementación paralela del método de SIFT sobre una arquitectura paralela de memoria compartida, aprovechando el amplio desarrollo de los multicores. Este tipo de máquina permite obtener menores tiempos de respuesta para una aplicación al dividir el trabajo entre los cores que lo forman. Para lograr tiempos de respuesta acordes a los requeridos para el procesamiento de videos en tiempo real se utiliza un esquema *Bag of Task* para lograr de esta manera un balance de carga que asegure un buen rendimiento de la aplicación. El método aquí propuesto es aplicado al reconocimiento de objetos en video en tiempo real.

Palabras Claves

Tracking de Video, Descriptores SIFT, Arquitectura Multicore.

Trabajos Realizados

Se ha realizado un completo relevamiento de la literatura en lo que respecta a la construcción de descriptores característicos asociados a distintas regiones de una imagen. Se ha seleccionado SIFT por su invarianza a la escala, rotación, cambios de iluminación y puntos de vista, oclusión y ruido.
Con el objetivo de reducir el tiempo de cálculo se han estudiado conceptos básicos de paralelismo, librería OpenMP y arquitectura multicore.

Conclusiones

Se ha desarrollado una versión paralela del método SIFT aplicable a tracking de video. Se analizaron tres alternativas: particionamiento de frames, división por puntos claves y distribución de frames. Se seleccionó la última por su buen desempeño en el procesamiento de videos en tonos de grises. Los resultados de esta investigación fueron publicados en el VIII Workshop de Computación Gráfica, Imágenes y Visualización realizado en el marco del XVI Congreso Argentino de Ciencias de la Computación (Bs.As. - Octubre/2010).

Trabajos Futuros

Actualmente se está trabajando en la caracterización de videos a partir de sus objetos más relevantes. Esto permitiría clasificarlos automáticamente por distintos criterios. Además, a través de este enfoque sería factible el uso y aplicación de técnicas de Minería de Textos para recuperar y catalogar dichos videos automáticamente.
Otra línea de trabajo futura es profundizar en el estudio de técnicas de tracking de video.

Fecha de la presentación: Diciembre 2010

Reconocimiento de objetos en video

Autores: Bernarda Albanesi y Nadia Funes

Directora : Prof. Lic. Laura Lanzarini
Codirector : Prof. Lic. Franco Chichizola

Tesina de Licenciatura en Sistemas
Facultad de Informática
Universidad Nacional de La Plata

2010

0.1. Agradecimientos

Quisiéramos agradecer a nuestra familia y amigos todo el apoyo incondicional que nos han brindado en esta etapa tan importante de nuestras vidas. No queremos dejar de mencionar a la Lic. Laura Lanzarini y el Lic. Franco Chichizola que han contribuido con la elaboración de esta tesina.

0.2. Resumen

El reconocimiento de objetos en video sigue siendo un tema de actualidad y ha motivado el desarrollo de diversos métodos informáticos para tratar de resolverlo.
Uno de los enfoques más utilizados radica en la caracterización de los pixels más relavantes de la imagen. Si bien en esta dirección existen distintas soluciones, en todos los casos se busca expresar la información referida a la vecindad de dichos puntos de la forma más invariante posible. El método SIFT (Scale-Invariant Feature Transform) es, sin duda, uno de los más utilizados. Sin embargo, su costo computacional es alto si se piensa en su aplicación en tareas tales como tracking de video.

El objetivo central de esta tesina es proponer una implementación paralela del método de SIFT sobre una arquitectura paralela de memoria compartida, aprovechando el amplio desarrollo de los multicores. Este tipo de máquina permite obtener menores tiempos de respuesta para una aplicación al dividir el trabajo entre los cores que lo forman. Para lograr tiempos de respuesta acordes a los requeridos para el procesamiento de videos en tiempo real se utiliza un esquema *Bag of Task* para lograr de esta manera un balance de carga que asegure un buen rendimiento de la aplicación.

El método aquí propuesto ha sido aplicado al reconocimiento de objetos en video en tiempo real con un buen desempeño en el procesamiento de videos en tonos de grises. Los resultados de esta investigación fueron publicados en el VIII Workshop de Computación Gráfica, Imágenes y Visualización realizado en el marco del XVI Congreso Argentino de Ciencias de la Computación bajo el título *"Reconocimiento de objetos en video utilizando SIFT Paralelo"*.

Palabras Claves: Tracking de Video, Descriptores SIFT, Arquitectura Multicore.

Índice general

Capítulo 1

Técnicas básicas de procesamiento de imágenes

1.1. Introducción

La historia del procesamiento digital de imágenes (PDI) está directamente ligada con el desarrollo y evolución de las computadoras. Su progreso ha ido de la mano con el desarrollo de las tecnologías de hardware, ya que requiere un alto poder y recursos computacionales para almacenar y procesar las imágenes. De igual manera el desarrollo de los lenguajes de programación y los sistemas operativos han hecho posible el crecimiento continuo de aplicaciones relacionadas al procesamiento de imágenes, tales como: imágenes médicas, satelitales, astronómicas, geográficas, arqueológicas, biológicas, aplicaciones industriales, entre otras [38] [35].

El interés en el procesamiento digital de imágenes (PDI) se basa esencialmente en dos aspectos, en mejorar la información contenida en una imagen para la interpretación humana y en el tratamiento de los datos de una escena para la percepción autónoma por una máquina. Es un tema muy amplio, en el que se incluyen aspectos de física, matemáticas, ingeniería eléctrica, computación. Estudia los fundamentos conceptuales de la adquisición y despliegue de imágenes y con detalle los fundamentos teóricos y algorítmicos del procesamiento como tal. Tiene además, como objetivo mejorar el aspecto de las imágenes y hacer más evidentes en ellas ciertos detalles que se desean hacer notar.

La Visión Artificial (o visión computacional) puede ser definida como los procesos de obtención, caracterización e interpretación de información de imágenes tomadas de un mundo tridimensional. Estos procesos pueden ser subdivididos en seis áreas principales y están agrupados de acuerdo a la complicación y delicadeza que lleva su implementación. Consideramos tres niveles de procesamiento: visión de bajo, medio y alto nivel. La figura 1.1 muestra un detalle de los procesos involucrados en cada uno de ellos:

1. La captura o adquisición es el proceso a través del cual se obtiene una imagen digital utilizando un dispositivo de captura como una cámara digital, video cámara, escáner, satélite, etc.

2. El pre procesamiento incluye técnicas tales como la reducción del ruido, realce del con-

Procesos del PDI	**Nivel de Visión**
1. Captura o adquisición	Bajo
2. Preprocesamiento	
3. Segmentación	Medio
4. Descripción	
5. Reconocimiento	
6. Interpretación	Alto

Figura 1.1: Niveles de procesamiento

Figura 1.2: Efecto que el ruido produce sobre una imagen. (a) Imagen original. (b) Imagen (a) con ruido

traste, realce de ciertos detalles, o características de la imagen.

3. La segmentación es el proceso que divide una imagen en objetos que sean de interés de estudio.

4. La descripción es el proceso que obtiene características convenientes para diferenciar un tipo de objeto de otro, por ejemplo: la forma, el tamaño, área, etc.

5. El reconocimiento es el proceso que identifica los objetos, como por ejemplo: una llave, un tornillo, moneda, coche, etc.

6. La interpretación es el proceso que asocia un significado a un conjunto de objetos reconocidos (llaves, tornillos, herramientas, etc.) y trata de emular la cognición.

1.2. Ruidos

Todas las imágenes tienen cierta cantidad de información no deseada llamada ruido, que las contamina. Esto se puede deber a la cámara, escáner o al medio de transmisión de la señal. Generalmente el ruido se manifiesta como píxeles aislados que toman un nivel de gris diferente al de sus vecinos y aparece como pequeñas y aleatorias variaciones en el brillo y en el color. La figura 1.2 ejemplifica esta situación.

Las imágenes pueden contener ruido provocado por fuentes ruidosas, como por ejemplo los sensores ópticos eléctricos o los mecanismos de apertura en cámaras fotográficas [25]. También el canal físico, por el cual en ocasiones se transmite la imagen, puede ser una fuente de ruido.

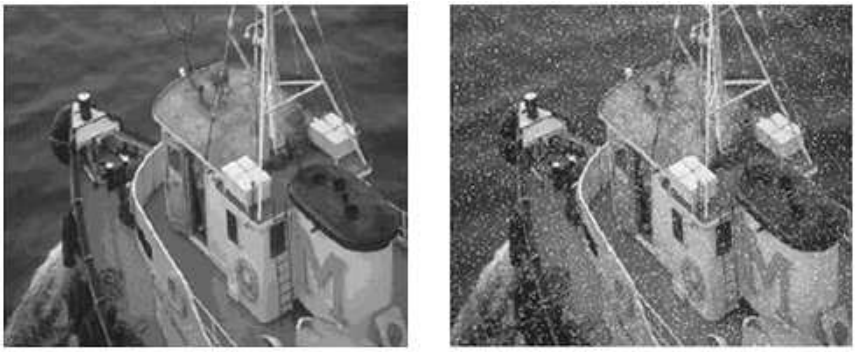

Figura 1.3: Ruido sal y pimienta. (a) Imagen original. (b) Imagen ruidosa

1.2.1. Tipos de ruidos

Existen diferentes tipos de ruidos que se diferencian en el aspecto y el comportamiento. Los mismos se detallaran a continuación.

Ruido impulsional (Sal y Pimienta)

Cuando se produce este tipo de ruido, el valor que toma el píxel no tiene relación con el valor ideal, sino que toma valores muy altos o bajos (puntos blancos y/o negros) causados por una saturación del sensor o por un valor mínimo captado, si se ha perdido la señal en ese punto.

Una imagen que contiene ruido sal y pimienta tendrá píxeles oscuros en las regiones brillantes y píxeles brillantes en las regiones oscuras. Generalmente este tipo de ruido sólo afectará a un pequeño número de píxeles de la imagen, como se puede observar en la imagen de la derecha de la figura 1.3.b) [38].

Ruido gaussiano

Este tipo de ruido produce pequeñas variaciones en la imagen. Generalmente se debe a diferentes ganancias en la cámara, ruido en los digitalizadores, perturbaciones en la transmisión. Se considera que el valor final del píxel sería el valor ideal más una cantidad correspondiente al error que puede describirse como una variable aleatoria gaussiana [38]. La figura 1.4.b) ejemplifica este tipo de ruido.

Ruido Multiplicativo

En este tipo de ruido la imagen obtenida es el resultado de la multiplicación de dos señales. El ruido que afecta a la imagen sigue una distribución uniforme. La probabilidad de tomar cualquier valor de gris dentro de un intervalo definido es constante. La figura 1.5.b) muestra un ejemplo de este tipo de ruido.

Figura 1.4: Ruido gaussiano. (a) Imagen original, (b) Imagen ruidosa

Figura 1.5: Ruido multiplicativo. (a) Imagen original. (b) Imagen con ruido multiplicativo

1.2.2. Eliminación

En PDI es importante poder eliminar o disminuir el ruido en las imágenes para lograr mejorar la visión o el procesamiento.

Los algoritmos de filtrado permiten reducir el ruido y/o efectos espurios que pueden presentarse en una imagen a consecuencia del proceso de captura, digitalización y transmisión. Su utilización es normalmente necesaria antes de la aplicación de un detector de bordes. No obstante las técnicas de reducción de ruido presentan varios inconvenientes, ya que se puede observar pérdida de resolución de la imagen y/o reducción del contraste.

1.3. Filtros

El filtrado es un tipo de operación que altera el valor de un píxel en función de los valores de los pixels que se encuentran a su alrededor, es por ello que a este tipo de procesamiento de la imagen también se le denomina procesamiento basado en la vecindad u operación de vecindad.

Los filtros modifican la imagen original con el objetivo de hacerla más luminosa, u oscura, o borrosa, o resaltada, o más definida, etc. El efecto dependerá del procesamiento que se esté realizando sobre dicha imagen.

Los filtros se encuentran divididos en: espaciales y frecuenciales.

El término "espacial" se utiliza para distinguir que la alteración del píxel se realiza dependiendo de los valores de los píxeles del entorno sin realizar ninguna modificación previa de sus valores, lo cual no ocurre con el filtrado frecuencial, que requiere de la aplicación de la transformada de Fourier.

Dada una imagen de entrada $f(x, y)$ se le aplica un operador T y el resultado es una nueva imagen $g(x, y)$ cuyos pixeles fueron alterados. Esto se expresa en la ecuación (1.1).

$$g(x, y) = T\{f(x, y)\} \tag{1.1}$$

Estos filtros pueden ser lineales y no lineales. En el primer caso el operador depende de forma lineal de los pixeles del entorno. En el segundo caso, también se basan en operaciones realizadas sobre los pixels del vecindario, con la diferencia de que dichas operaciones no son lineales.

En esta sección sólo se describirán los filtros espaciales dado que fueron utilizados para el desarrollo de la tesina.

1.3.1. Filtrado espacial lineal

La aplicación de un filtro a una imagen de entrada $f(x, y)$ requiere de la definición de una matriz de dimensión mxn, denominada máscara, la cual contiene los coeficientes del filtro. La figura 1.6 ejemplifica una máscara de 3x3.

El proceso de aplicar un filtro consiste en ir desplazando el centro de la máscara a lo largo de la imagen inicial $f(x, y)$, desde la esquina superior izquierda hasta la esquina inferior derecha,

w_1 (x-1, y-1)	w_2 (x, y-1)	w_3 (x+1,y-1)
w_4 (x-1,y)	w_5 (x,y)	w_6 (x+1,y)
w_7 (x-1,y+1)	w_8 (x,y+1)	w_9 (x+1, y+1)

Figura 1.6: Máscara de 3x3

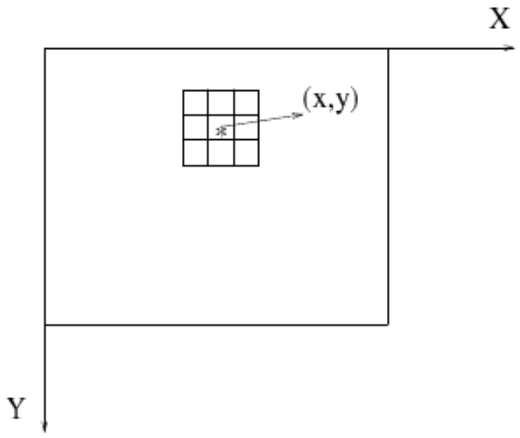

Figura 1.7: Mascara de 3x3 aplicada a una imagen

aplicando el operador T en cada punto, para luego obtener el valor de la imagen de salida $g(x, y)$ en cada punto. La figura 1.7 indica las direcciones en las cuales se desplaza el centro (X, Y) del filtro y la ecuación (1.2) la manera de calcularlo para una máscara de 3x3.

$$\begin{aligned} g(x,y) &= T\{f(x,y)\} \\ &= w1f(x-1,y-1) + w2f(x,y-1) + w3f(x+1,y-1) + \\ &\quad w4f(x-1,y) + w5f(x,y) + w6f(x+1,y) + \\ &\quad w7f(x-1,y+1) + w8f(x,y+1) + w9f(x+1,y+1) \end{aligned} \tag{1.2}$$

Uno de los inconvenientes que este tipo de filtro enfrenta es el de cómo tratar el vecindario de los píxeles que constituyen el contorno de la imagen de entrada. Por ejemplo, cuando se pretende aplicar la máscara al píxel $f(1, 1)$ se desconocen algunos valores de f. Para resolver esto, se agregan píxeles de relleno a la matriz que representa a la imagen de entrada $f(x, y)$. De esta forma, cuando se procesen los pixeles que están en el contorno, sus vecinos poseerán algún valor [34][35].

Filtro promedio

Aplicando este filtro, para cada pixel de la imagen de entrada $f(x, y)$, se calcula el promedio entre su valor y los valores de los pixels que lo rodean. Intervienen en el promedio los valores

Figura 1.8: Filtro promedio: (a) Imagen original, (b) Imagen filtrada

de la imagen original que quedan cubiertos por la ventana de mxn utilizada como máscara. El valor del pixel (x, y) en la imagen de salida g contendrá este promedio. La ecuación (1.3) indica la manera de calcular el filtro promedio para el punto (x, y) siendo nm el número de píxels en la ventana W de dimensión mxn. Nótese que es equivalente a lo expresado por la ecuación (1.2) dándole valor 1 a todos los elementos de la máscara.

$$g(x, y) = \frac{1}{nm} \sum_{(x,y)\in W} f(x, y) \tag{1.3}$$

En la figura 1.8 se muestra el resultado de aplicar el filtro promedio de tamaño 5.

Filtro Gaussiano

La ecuación (1.4) representa la manera de obtener cada valor de la máscara del filtro gaussiano siendo s y t la distancia al pixel central correspondiente a la columna y a la fila respectivamente.

$$G(s, t) = \frac{1}{2\pi\sigma^2} e^{-\frac{s^2+t^2}{\sigma^2}} \tag{1.4}$$

El término σ influye en el tamaño de la máscara, esto es, cuanto mayor sea σ, mayor es el tamaño de la máscara. Es importante considerar que un filtro con estas características otorga mayor importancia al pixel central que a los que se encuentran a su alrededor. Sin embargo, el peso del pixel central dependerá de la "amplitud" definida por σ. Por lo tanto, la imagen de salida resultará más o menos borroneada de acuerdo al σ utilizado.

La figura 1.9 muestra el resultado de aplicar el filtro Gaussiano.

1.3.2. Filtrado espacial no lineal

Cuando se utilizan filtros lineales, el valor de cada pixel de la imagen transformada luego de aplicar dicho filtro, se basa únicamente en utilizar una máscara como si se tratase de una matriz

Figura 1.9: Filtro Gaussiano : (a) Imagen original, (b) Imagen filtrada.

de pesos. Es decir que el nuevo valor se simplemente una combinación lineal de los valores originales ponderados según los coeficientes de la máscara.

Con los filtros no lineales, en general las máscaras no se utilizan. En su lugar, el nuevo valor de cada pixel es el resultado de aplicar un procedimiento distinto del promedio pesado de los valores originales. Por ejemplo, podría utilizarse una medida estadística (distinta de la media) basada sobre una vecindad de NxN.

A continuación se describen algunos filtros no lineales.

Filtro de la media armónica

El filtro de la media armónica se define como la división entre el tamaño de la ventana S que determina la vecindad de pixels a utilizar y la suma de la inversa de los pixeles de dicha vecindad.

Su cálculo se indica en la ecuación (1.5) siendo $f(x, y)$ los valores de la imagen original y $\hat{f}(x, y)$ los de la imagen resultante de aplicar el filtro. El tamaño de la ventana S es de mxn.

$$\hat{f}(x,y) = mn * \left(\sum_{(s,t)\in S_{(x,y)}} \frac{1}{f(s,t)} \right)^{-1} \tag{1.5}$$

Es adecuado para aplicarlo a imágenes con ruido gaussiano preservando los detalles de la imagen y para imágenes con ruido tipo sal, pero falla para imágenes con ruido pimienta.

La figura 1.10 muestra el resultado de aplicar el filtro a una imagen con ruido gaussiano.

Filtro de la media contra-armónica

El filtro de la media contra-armónica se define por la ecuación (1.6) donde Q es el factor de orden del filtro. Dependiendo de este valor, el filtro elimina en las imágenes el ruido sal o el ruido pimienta. Esto es, si Q es menor a 0 se elimina el ruido sal, si Q es mayor a 0 se elimina el ruido pimienta. Además si Q es igual a 0 el filtro es igual al de media aritmética y si el factor Q igual a -1 el filtro coincide con el de media armónica.

Figura 1.10: Filtro de la media armónica : (a) Imagen original, (b) Imagen con ruido gaussiano, (c) Imagen (b) filtrada.

$$\hat{f}(x,y) = \frac{\sum_{(s,t)\in S_{(x,y)}} f(s,t)^{Q+1}}{\sum_{(s,t)\in S_{(x,y)}} f(s,t)^{Q}} \tag{1.6}$$

Filtro de la media geométrica

El filtro de la media geométrica se define como el producto de los valores de los pixeles dentro de una ventana S elevado a la potencia $\frac{1}{mn}$, siendo mn el número de pixeles en la ventana de dimensiones mxn. Esto se muestra en la ecuación (1.7).

$$\hat{f}(x,y) = \left[\Pi_{(s,t)\in S_{(x,y)}} f(s,t)\right]^{\frac{1}{mn}} \tag{1.7}$$

Este filtro es adecuado para eliminar el ruido gaussiano, pero falla con el ruido impulsional.

Filtro de máximo

Se denomina filtro de máximo al que elige el pixel con el máximo valor de una vecindad S, como se indica en la ecuación (1.8). Se utiliza en imágenes que contienen el ruido pimienta, el cual se manifiesta con los pixeles negros. El efecto de este filtro es aclarar la imagen.

$$\hat{f}(x,y) = max_{(s,t)\in S_{(x,y)}}\{f(s,t)\} \tag{1.8}$$

En la figura 1.11 puede verse el funcionamiento del filtro

La figura 1.12 muestra el resultado de aplicar el filtro a una imagen con ruido gaussiano.

Filtro de mínimo

En este filtro se elige el pixel con valor mínimo de una vecindad S. Es utilizado en imágenes con ruido sal, el cual se manifiesta con pixeles blancos. El efecto del filtro es oscurecer la imagen. La ecuación de dicho filtro se puede expresar como en (1.9)

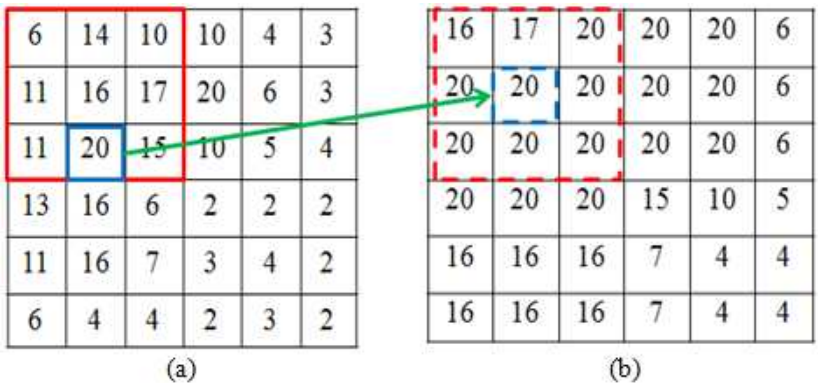

Figura 1.11: Aplicación de un filtro de máximo utilizando una ventana S de 3x3. (a) valores imagen de entrada. (b) valores imagen de salida con filtro de máximo

Figura 1.12: Filtro de máximo. (a) Imagen original, (b) Imagen con ruido gaussiano, (c) Imagen (b) filtrada.

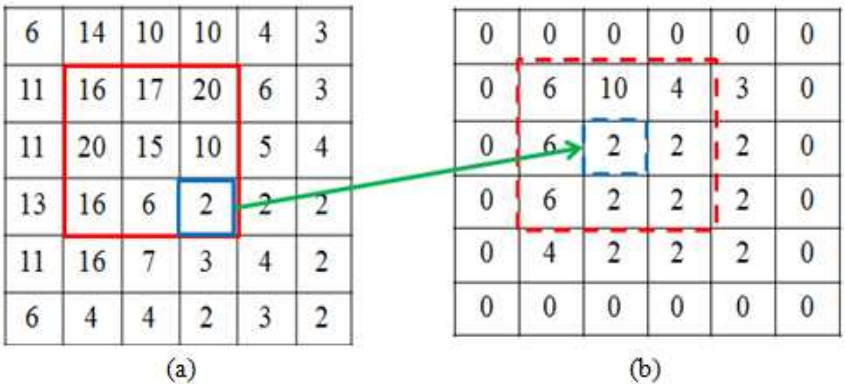

Figura 1.13: Aplicación de un filtro de minimo utilizando una ventana S de 3x3. (a) valores imagen de entrada. (b) valores imagen de salida con filtro de mínimo

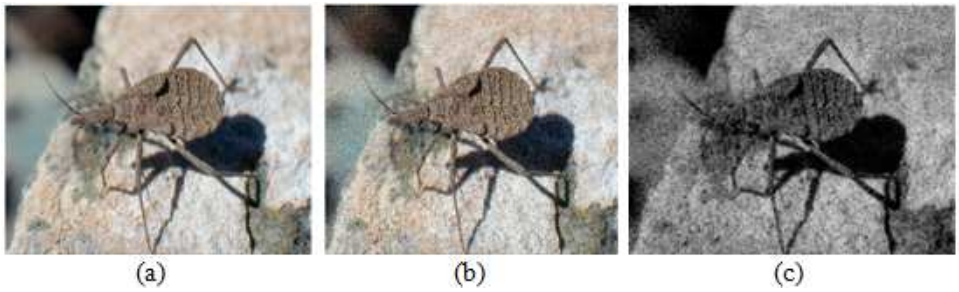

Figura 1.14: Aplicación de un filtro de minimo utilizando una ventana S de 3x3. (a) Imagen original, (b) Imagen con ruido gaussiano, (c) Imagen resultante al aplicar el filtro mínimo a la imagen (b)

$$\hat{f}(x,y) = min_{(s,t)\in S_{(x,y)}}\{f(s,t)\} \tag{1.9}$$

En la figura 1.13 se ve el funcionamiento del filtro de mínimo.

La figura 1.14 muestra el resultado de aplicar el filtro a una imagen con ruido gaussiano.

Filtro de punto medio

El filtro de punto medio se basa en la elección del valor promedio entre el pixel máximo y el mínimo de una ventana S. La ecuación (1.10) indica la manera de calcularlo. Es útil aplicarlo a imágenes con ruido gaussiano o uniforme.

$$\hat{f}(x,y) = \frac{1}{2}\left[max_{(s,t)\in S_{(x,y)}}\{f(s,t)\} + min_{(s,t)\in S_{(x,y)}}\{f(s,t)\}\right] \tag{1.10}$$

La figura 1.15 muestra el resultado de aplicar el filtro a una imagen con ruido gaussiano.

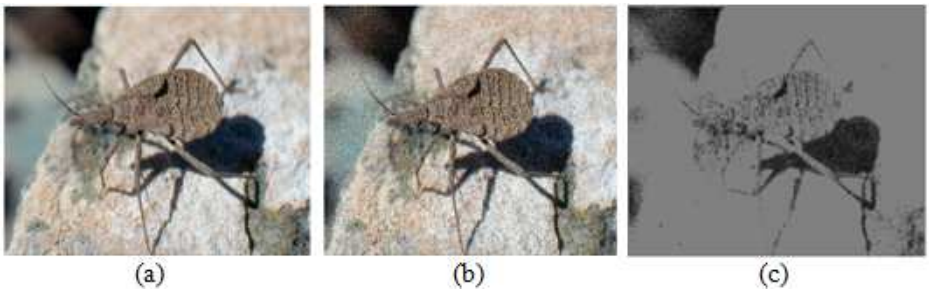

Figura 1.15: Aplicación de un filtro de minimo utilizando una ventana S de 3x3. (a) Imagen original, (b) Imagen con ruido gaussiano, (c) Imagen resultante al aplicar el filtro de punto medio a la imagen (b)

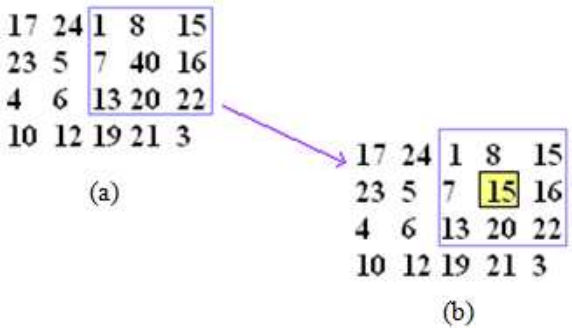

Figura 1.16: Aplicación de un filtro de mediana en el punto central de la vecindad S marcada. (a) Ventana con los valores de la imagen original, (b) Pixel central de la ventana reemplazado por la mediana de los 9 valores.

Filtro de la mediana

El filtro de la mediana calcula el pixel de la imagen transformada como la mediana de los pixels de la imagen original dentro de una vecindad S. Recuérdese que la mediana es una medida estadística que divide los valores en dos partes iguales de manera que todos los valores de la primera mitad son inferiores a su valor y los de la segunda mitad son superiores. Para hallar el valor de la mediana, se ordenan de menor a mayor los valores de los pixels de la vecindad S y luego se toma como mediada el valor central. Si la cantidad de pixels es par, se promedian los dos valores centrales.

La figura 1.16 muestra un ejemplo de lo mencionado anteriormente. Los valores ordenados de la ventana de 3x3 indicada en la figura 1.16.a) son: 1 7 8 13 15 16 20 22 40. El pixel central resulta ser el 15, con lo cual el resultado se puede observar en la figura 1.16.b).

Este filtro es adecuado para aplicarlo a imágenes con ruido impulsional. La figura 1.17 muestra el resultado de aplicar el filtro de la mediana a una imagen con ruido gaussiano.

Figura 1.17: (a) Imagen original. (b) Imagen con ruido gaussiano. (c) Imagen resultante al aplicar el filtro de la mediana a la imagen (b).

1.4. Realce

El realce de imagen consiste en aplicar a una imagen una técnica o conjunto de ellas cuyo objetivo sea resaltar alguna característica de dicha imagen que resulte de interés [12].

Se consideran operaciones de realce, entre otras:

- La reducción del ruido de fondo en la captación de una imagen. En este caso, lo que se pretende resaltar de la imagen es el contenido de la misma, eliminando la textura que pueda estar presente en el fondo, esto es, tratar de conseguir que el fondo presente una intensidad lo más constante posible.
- El ajuste de intensidad y/o contraste. Una vez más, se pretende resaltar el contenido de la imagen con respecto al fondo.
- El realce de bordes. En este caso se pretende enfatizar las transiciones presentes en la imagen (las fronteras entre dos estructuras diferentes; por ejemplo, en imagen radiográfica, la separación entre tejido óseo y tejido no óseo).

Las técnicas de realce pueden tener dos posibles destinatarios:

- Un observador humano, de forma que el objetivo del realce sea hacer la representación de la imagen más agradable, o en general, más apta para la interpretación humana.
- Una máquina (un ordenador o procesador dedicado), de forma que el objetivo del realce sea preparar a la imagen para su posterior procesado. En este segundo caso, el resultado del realce no tiene por qué ser una imagen interpretable por un ser humano.

Las técnicas de realce de imagen se han clasificado tradicionalmente acorde con las siguientes categorías:

- Operaciones punto a punto: en este caso, el valor de la imagen realzada en un píxel es función del valor de su píxel correspondiente en la imagen original. Ejemplo de estas técnicas son las operaciones de ajuste de contraste.
- Operaciones espaciales: en esta segunda categoría se engloban las técnicas de tipo convolutivo, es decir las técnicas en las que el valor de un píxel en la imagen realzada es función del

valor de su píxel correspondiente en la imagen original, y también, de los píxeles vecinos a este en la imagen original. Las operaciones espaciales se suelen llevar a la práctica a través de máscaras de convolución.

- Operaciones en el dominio transformado: típicamente son un caso particular de las anteriores, pero se suelen clasificar en grupos distintos por claridad. Las operaciones en un dominio transformado, consisten, como indica su nombre, en operaciones de realce, no en el dominio espacial original, sino en otros dominios que se consideren más cómodos o útiles para llevar a cabo operaciones de realce. Los dominios típicamente empleados son el dominio de la frecuencia (transformada de Fourier y otras) y el dominio logarítmico (filtrado homomórfico).
- Operaciones para imágenes en color: aquí se incluye la extensión multiespectral de las técnicas anteriores, así como el procesado en pseudo color [11].

1.4.1. Modelado de Histogramas

Los histogramas constituyen la base de varias técnicas de procesamiento en el dominio espacial. La manipulación de los histogramas es usada de manera eficiente en el realce o mejoramiento de la calidad de una imagen. La información estadística obtenida a partir de los histogramas se utiliza en diversas aplicaciones como compresión y segmentación de imágenes [34].

El histograma de una imagen permite representar, para cada nivel de gris, la cantidad de pixels que poseen el mismo valor.

En general se representa como un gráfico de barras en el que el eje de las abscisas son los distintos niveles de grises de la imagen y el de las ordenadas la frecuencia relativa con la que cada nivel aparece en la imagen.

El histograma proporciona información sobre el brillo y el contraste de la imagen y puede ser utilizado para ajustar estos parámetros, eliminar ciertas tonalidades molestas, etc.

La mejora de la calidad de una imagen se basa en modificar el histograma de la imagen original para obtener, a partir del histograma transformado, una nueva imagen presumiblemente mejor. La figura 1.18.(a) representa el histograma de la figura 1.18.(b).

1.4.2. Ecualización de histograma

Consiste en una expansión del histograma de la imagen, dotando al mismo de mayor linealidad y haciendo que éste ocupe el ancho del espectro de tonalidades grises por completo, ello implica las mejoras en la imagen que se explican a continuación:

- Una mayor utilización de los recursos disponibles: al ecualizar el histograma, se ve como los tonos que antes estaban más agrupados, se separan, ocupando todo el rango de grises, por lo que la imagen se enriquece al tener niveles de gris más distintos entre sí, mejorando, por tanto, la apariencia visual de la imagen.
- Un aumento del contraste: esta ventaja es consecuencia del punto anterior, ya que si se hace que el histograma de la imagen ocupe todo el rango de grises, se aumenta la

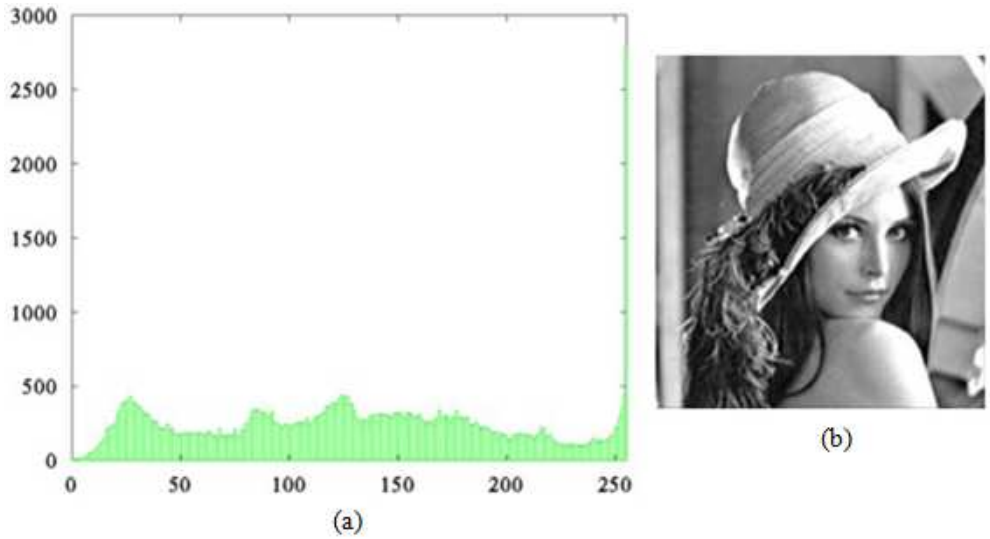

Figura 1.18: (a) Histograma de la imagen representada en (b)

distancia entre el tono más claro y el más oscuro, convirtiendo a éstos, en blanco y negro y consecuentemente aumentando el contraste de la imagen.

- Constituye una regulación óptima y automática del contraste de la imagen. Evitando los ajustes manuales con los que no se consigue un equilibrio óptimo entre el blanco y el negro.

A su vez, aparecen algunos inconvenientes que surgen a la hora de ecualizar la imagen, algunos de ellos se detallan a continuación:

- Pérdida de información: puede ocurrir que a algunos pixeles que en la imagen original tenían distintos niveles de gris se les asigne, tras la ecualización global, al mismo nivel de gris. Por otro lado, hay casos en los que dos niveles de gris muy próximos se separen, dejando huecos en el histograma.
- En ocasiones, las bandas horizontales, fruto de una deficiente digitalización pueden resultar intensificadas, resaltando aún más este error indeseado.

La figura 1.19 muestra una imagen de prueba a la cual se calcula su histograma. La figura 1.20 muestra la ecualización del histograma anterior y el resultado de esta en la imagen de prueba.

Como consecuencia, una imagen ecualizada, permite que ciertos detalles sean visibles en regiones oscuras o brillantes [35].

1.5. Conclusiones

Este primer capítulo ha tenido como objetivo introducir al lector en las técnicas básicas del procesamiento de imágenes relacionadas con el tema de esta tesina.

Se han definido los distintos tipos de ruidos que pueden presentarse en una imagen y se ha tratado el tema de filtros espaciales lineales como una herramienta para reducir su efecto. Los

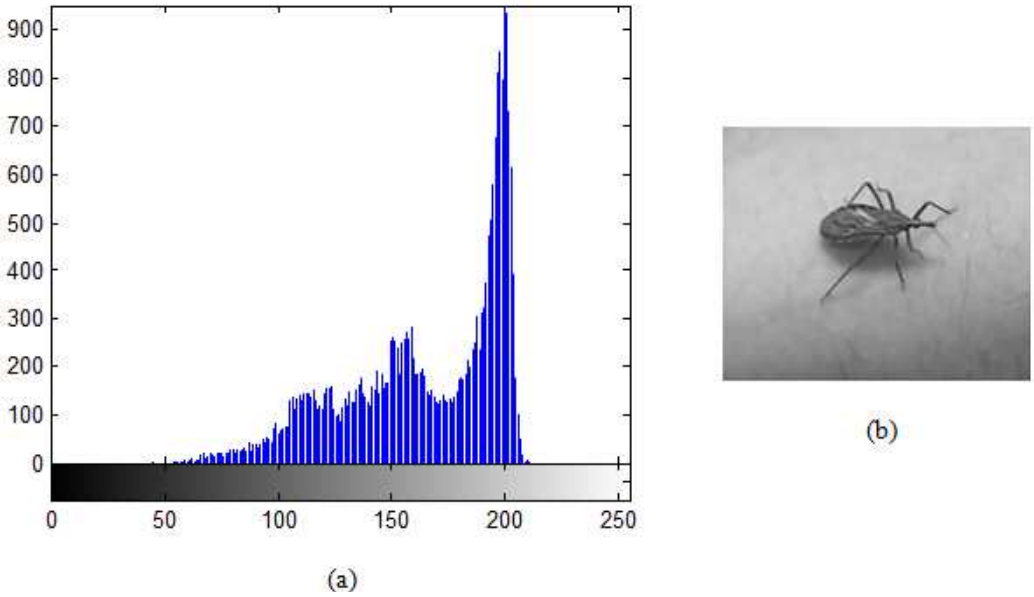

Figura 1.19: (a) Histograma de la figura (b)

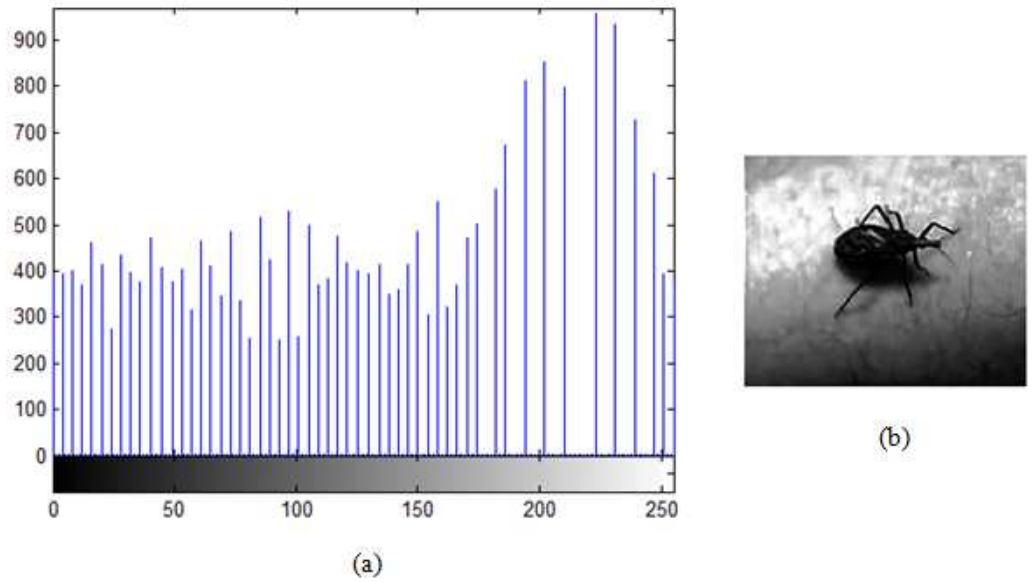

Figura 1.20: (a) Ecualización del histograma de la figura 1.19. (b) Imagen realzada mediante la ecualización del histograma

filtros de frecuencia se encuentran fuera del alcance de esta tesina y por tal motivo su descripción ha sido omitida.

Se ha introducido el concepto de histograma como una herramienta para realzar la imagen. En este caso en particular, no es otra cosa que la distribución de frecuencia de los distintos niveles de grises que posee la imagen. Diferentes operaciones realizadas sobre el histograma permiten mejorar por ejemplo el brillo y el contraste de la imagen.

Sin embargo el histograma es una herramienta gráfica estadística que permite representar la distribución de frecuencia de los valores de una variable aleatoria. En el capítulo 4 se utilizará un histograma en este contexto.

En el siguiente capítulo se mencionan algunos filtros que permiten la extracción de bordes y esquinas de las imágenes. Su comprensión es necesaria para analizar las diferencias de acción entre distintos métodos de extracción de características.

Capítulo 2

Métodos para detección de bordes y esquinas

En este capítulo se explicaran varias técnicas para la detección de los tipos básicos de discontinuidades, como bordes y esquinas.

Los operadores estudiados para detección de bordes pertenecen a la categoría de filtros lineales, es decir que utilizan una máscara que se convoluciona con la imagen original para obtener el resultado buscado.

Se encuentran divididos entre los que utilizan la derivada primera discreta calculada sobre los niveles de grises de la imagen y los que utilizan la derivada segunda. La elección del operador depende de la velocidad de cambio entre dichos niveles de grises. Cuando el cambio es abrupto, se recomienda el uso de los operadores basados en la derivada primera; cuando el cambio es más suave (involucra un mayor número de pixels) es conveniente utilizar los operadores basados en la derivada segunda.

2.1. Detección de bordes

En el área de procesamiento de imágenes, la detección de los bordes de una imagen es de suma importancia y utilidad, pues facilita muchas tareas, entre ellas, el reconocimiento de objetos, la segmentación de regiones, etc.

Los bordes de una imagen digital se pueden definir como transiciones entre dos regiones de niveles de gris significativamente distintos. Suministran una valiosa información sobre las fronteras de los objetos y son muy útiles para su reconocimiento.

2.1.1. Operadores de gradiente

Cuando se trabaja con una imagen continua, $f(x, y)$ su derivada alcanza un máximo local en la dirección del borde. Por lo tanto, una técnica para detectarlo es medir el gradiente de f con respecto a r en una dirección θ (ver figura 2.1); es decir

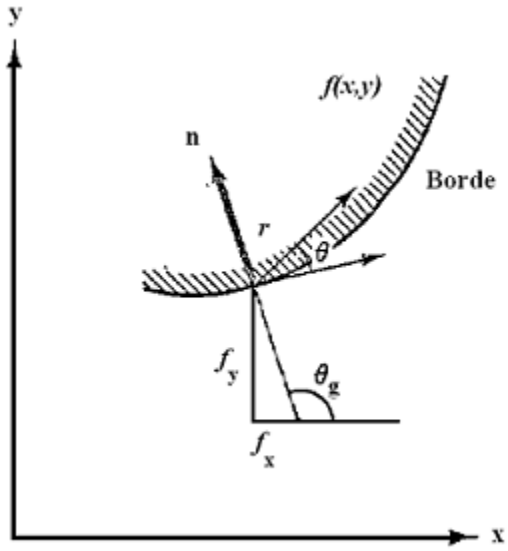

Figura 2.1: Gradiente de $f(x, y)$ en la dirección r

$$\nabla f = \frac{\partial f}{\partial r} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial r} = f_x cos(\theta) + f_y sin(\theta) \tag{2.1}$$

El valor máximo de $\frac{\partial f}{\partial r}$ se obtienen cuando $\frac{\partial}{\partial \theta}\frac{\partial f}{\partial r} = 0$. Esto implica que

$$-f_x sin(\theta_g) + f_y cos(\theta_g) = 0 \Rightarrow \theta_g = tan^{-1}\left(\frac{f_y}{f_x}\right) \tag{2.2}$$

$$mag(\nabla f) = \sqrt{(f_x^2 + f_y^2)} \tag{2.3}$$

donde θ_g es la dirección del eje.

Cono ya se dijo anteriormente, los métodos basados en el gradiente son más adecuados cuando la transición en los niveles de gris de los píxeles del entorno es brusca, similar a una función escalón. Sin embargo, estos operadores son pocos sensibles a los cambios graduales en los niveles de gris.

A continuación se describen brevemente algunos de los operadores basados en gradiente más utilizados.

Operador Sobel

El operador Sobel aplicado sobre una imagen digital en escala de grises, calcula el gradiente de la intensidad de brillo de cada punto (píxel) dando la dirección del mayor incremento posible (de negro a blanco). El resultado muestra la magnitud de cambio de la imagen en cada punto analizado, informando que puntos representan bordes en la imagen y la orientación a la que tienden los mismos.

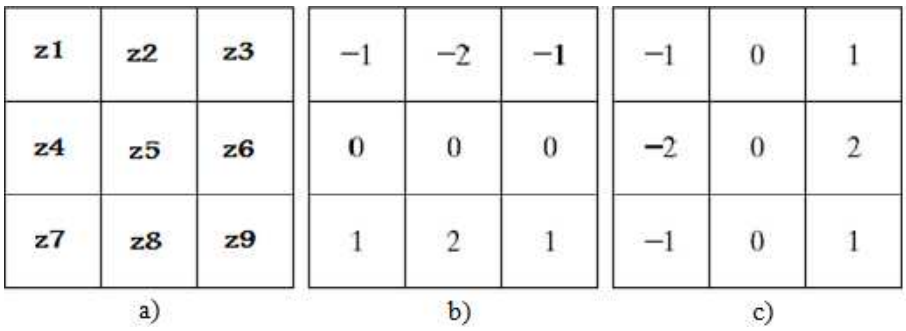

Figura 2.2: Filtro de Sobel. (a) Vecindario de la imagen. (b) Filtrado horizontal: (c) Filtrado vertical

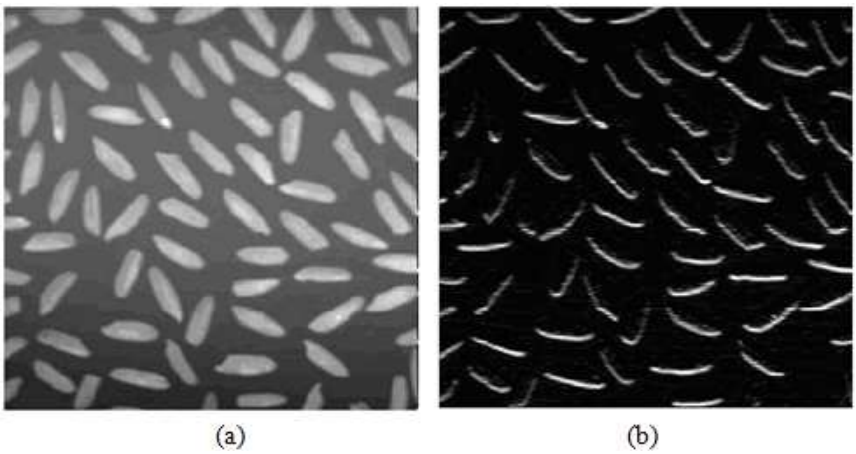

Figura 2.3: Filtro de Sobel. (a) Imagen original. (b) Imagen filtrada

Las máscaras que el operador utiliza para el filtrado horizontal y vertical están dadas por las siguientes fórmulas $G_x = (z7+2z8+z9)-(z1+2z2+z3)$ y $G_y = (z3+2z6+z9)-(z1+2z4+z7)$ respectivamente. En la figura 2.2 se pueden observar los filtros mencionados. Nótese que en las máscaras de Sobel, tienen más peso los píxeles situados en posición vertical y horizontal con respecto al píxel estudiado que los situados en la diagonal. Este operador es menos sensible al ruido [1].

En la figura 2.3 se muestra un ejemplo de la aplicación del filtro de Sobel a una imagen.

Operador Prewitt

El operador Prewitt otorga el mismo peso a los píxeles contiguos en vertical y horizontal.

Las máscaras que el operador utiliza para el filtrado horizontal y vertical están dadas por las siguientes fórmulas: $G_x = (z7+z8+z9)-(z1+z2+z3)$ y $G_y = (z3+z6+z9)-(z1+z4+z7)$ respectivamente. En la figura 2.4 se pueden observar los filtros mencionados. En la figura 2.5 se muestra un ejemplo de la aplicación del filtro de Prewitt a la misma imagen de la figura 2.3.(a).

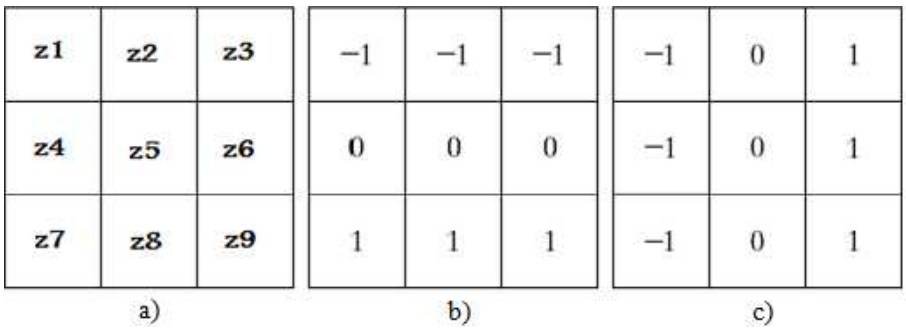

Figura 2.4: Filtro de Prewitt. (a) Vecindario de la imagen. (b) Filtrado horizontal: (c) Filtrado vertical

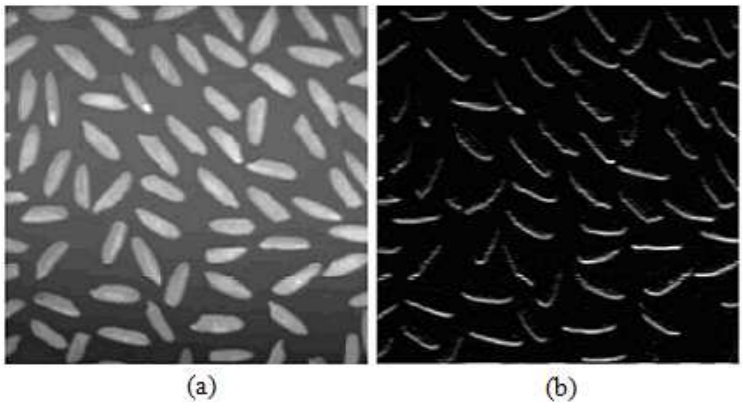

Figura 2.5: Filtro de Prewitt. (a) Imagen original. (b) Imagen filtrada

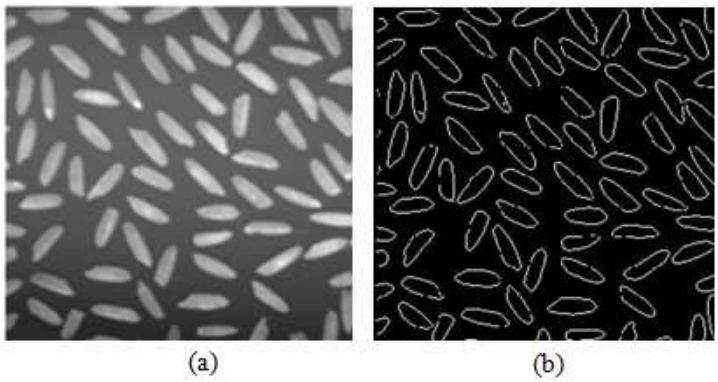

Figura 2.6: Filtro de Canny. (a) Imagen original. (b) Imagen filtrada

Operador Canny

El operador Canny es usado para detectar todos los bordes existentes en una imagen [10]. Está considerado uno de los mejores métodos de detección de bordes, basado en la primera derivada, que utiliza máscaras de convolución.

El algoritmo se basa en los siguientes criterios:

- Criterio de detección: expresa el hecho de evitar la eliminación de bordes importantes y no suministrar falsos bordes.
- Criterio de localización: establece que la distancia entre la posición real y la localizada del borde se debe minimizar.
- Criterio de una respuesta: que integre las respuestas múltiples correspondientes a un único borde.

Por otro lado, el algoritmo consiste en tres grandes pasos:

- Obtención del gradiente: en este paso se calcula la magnitud y orientación del vector gradiente en cada píxel.
- Supresión no máxima: en este paso se logra el adelgazamiento del ancho de los bordes, obtenidos con el gradiente, hasta lograr bordes de un píxel de ancho.
- Histéresis de umbral: en este paso se aplica una función de histéresis basada en dos umbrales; con este proceso se pretende reducir la posibilidad de aparición de contornos falsos.

En la figura 2.6 se muestra un ejemplo de la aplicación del filtro de Canny.

2.1.2. Operador Laplaciano

El operador Laplaciano se caracteriza por ser lineal e independiente de la dirección de las discontinuidades de la imagen. Por lo tanto es invariante a la rotación, por esta razón se dice que es isotrópico [2].

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

a) b)

Figura 2.7: Filtro Laplaciano. (a) Máscaras de 3x3 de 4 vecinos, (b) de 8 vecinos.

Consiste en la utilización de una máscara la cual se aplica para destacar sectores de las imágenes donde se encuentran cambios significativos en la intensidad. Un uso común de este filtro es la detección de bordes. La ecuación, para una función en dos dimensiones, está dada por (2.4).

$$\Delta f(x,y) = \nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2} \tag{2.4}$$

Los términos derivativos se calculan como se muestra en las ecuaciones (2.5) y (2.6)

$$\frac{\partial^2 f(x,y)}{\partial x^2} = f(x+1,y) + f(x-1,y) - 2f(x,y) \tag{2.5}$$

$$\frac{\partial^2 f(x,y)}{\partial y^2} = f(x,y+1) + f(x,y-1) - 2f(x,y) \tag{2.6}$$

La ecuación (2.7) muestra el resultado de reemplazar (2.5) y (2.6) en (2.4).

$$\nabla^2 f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y) \tag{2.7}$$

dando lugar a la máscara que se observa en la figura 2.7.a) conocida como la laplaciana de los cuatro vecinos. Otra máscara muy utilizada es la que se observa en la figura 2.7.b); se la denomina de 8 vecinos y no es separable en sumas de derivadas segundas.

Este filtro resulta una herramienta sumamente útil para detectar bordes en situaciones donde el cambio de intensidad no es inmediato sino que requiere de varios pixels. En estos casos, la derivada segunda tomará valor cero al cruzar el punto medio de la línea de borde. Es decir que la línea de borde pude ser localizada detectando el cruce por cero de las diferencias de las derivadas de segundo orden de la imagen.

La figura 2.8 ejemplica la aplicación del filtro laplaciano.

2.1.3. LoG - Operador Laplaciano de la gaussiana

El filtro laplaciano descripto anteriormente puede generar una cantidad excesiva de bordes al ser aplicado sobre imágenes con ruido. Por ese motivo, resulta conveniente suavizar previamente la imagen mediante su convolución con un filtro gaussiano con desviación típica σ [2].

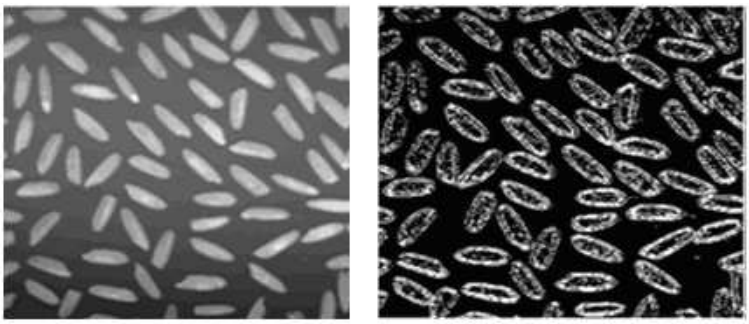

Figura 2.8: Filtro Laplaciano. (a) Imagen original, (b) Imagen filtrada.

$$\frac{1}{273}\begin{pmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{pmatrix}$$

Figura 2.9: Máscara de 5x5 para el filtro gaussiano definido en la ecuación (2.8) con $\sigma = 1$

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.8}$$

La figura 2.9 ejemplifica una máscara de 5x5 para el filtro gaussiano, definido en la ecuación (2.8), utilizando $\sigma = 1$.

Realizar primero la convolución del filtro gaussiano sobre la imagen original y luego aplicar el filtro laplaciano es equivalente a calcular la derivada segunda del filtro gaussiano y luego convolucionar la expresión resultante sobre la imagen. Esto es lo que se representa en la ecuación (2.9). A la expresión $\Delta G_\sigma(x,y)$ se lo denomina laplaciano de la gaussina y se lo suele denotar como LoG.

$$\Delta[G_\sigma(x,y) * f(x,y)] = [\Delta G_\sigma(x,y)] * f(x,y) = LoG * f(x,y) \tag{2.9}$$

La ecuación (2.10) representa el operador o kernel de convolución a aplicar

$$LoG = \Delta G_\sigma(x,y) = \frac{\partial^2}{\partial x^2} G_\sigma(x,y) + \frac{\partial^2}{\partial y^2} G_\sigma(x,y) = \frac{x^2+y^2-2\sigma^2}{2\pi\sigma^6} \tag{2.10}$$

La figura 2.10 ejemplifica una máscara de 5x5 para el filtro Laplaciana de la Gaussiana (LoG) definido en la ecuación (2.10).

En la figura 2.11 muestra el resultado de aplicar el filtro Laplaciana de la Gaussiana.

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 1 & 2 & -16 & 2 & 1 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Figura 2.10: Máscara de 5x5 para el filtro *LoG* definido en la ecuación ((2.10)).

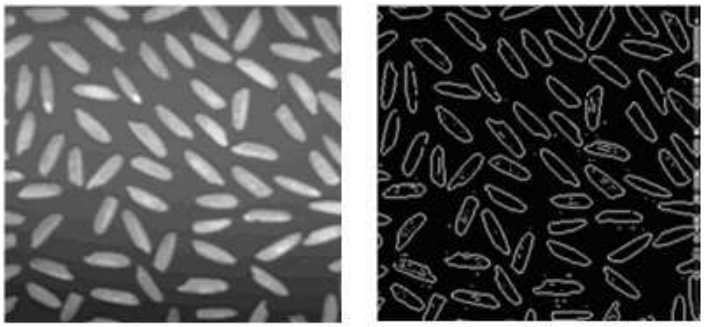

Figura 2.11: (a) Imagen original, (b) Imagen resultante al aplicar el filtro LoG

2.1.4. DoG - Diferencia de Gaussianas

La función gaussiana en 2-D tiene la siguiente fórmula

$$G(x,y) = ke^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.11}$$

donde k es la altura de la función y σ es la desviación estándar.

El operador *DoG* es un filtro pasa banda, en el dominio del espacio, construido a partir de dos filtros Gaussianos simples [24]. Estos dos filtros deben tener varianzas diferentes. Al restar las dos imágenes obtenidas de aplicar cada filtro por separado, se obtiene una imagen que contiene un rango de frecuencia que interesa detectar de la siguiente manera:

$$DoG(x,y) = k_1 e^{-\frac{x^2+y^2}{2\sigma_1^2}} - k_2 e^{-\frac{x^2+y^2}{2\sigma_2^2}} \tag{2.12}$$

Una aproximación de este filtro, está dada por la ecuación (2.13)

$$h(x,y) = e^{-\frac{x^2+y^2}{2\sigma_1^2}} - e^{-\frac{x^2+y^2}{2\sigma_2^2}} \tag{2.13}$$

Este operador hace la detección de bordes usando el algoritmo llamado "Diferencia de Gaussianas", que ejecuta dos desenfoques Gaussianos en la imagen, con diferentes radios y resta las dos versiones para obtener el resultado final. En la figura 2.12 se muestra un ejemplo de ello. Es

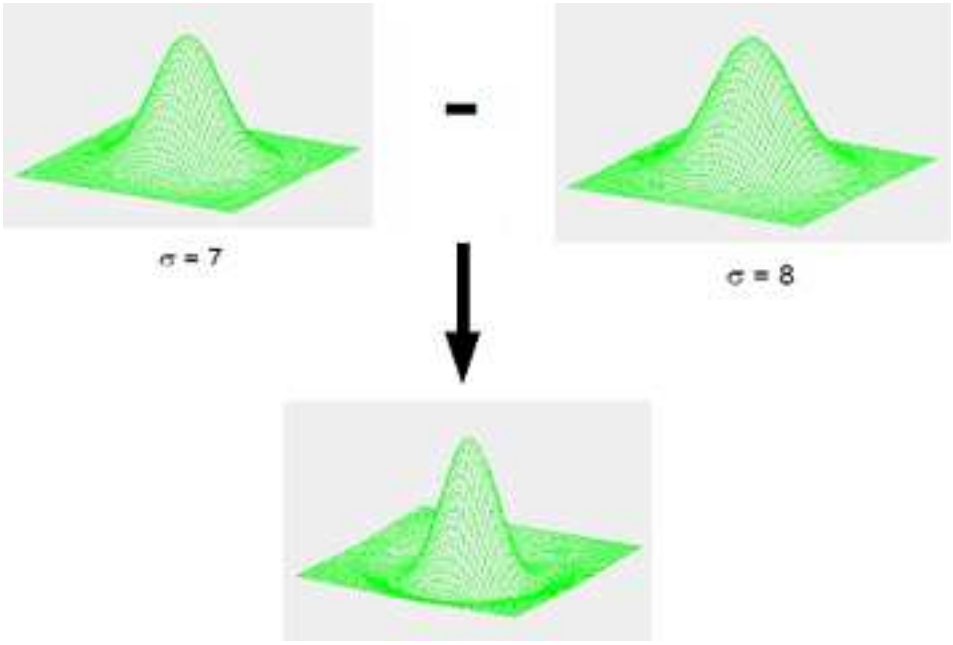

Figura 2.12: Diferencia de Gaussianas (DoG)

un algoritmo ampliamente usado en el campo de la visión artificial, por lo que existen numerosos métodos eficientes para realizar desenfoques Gaussianos en forma rápida.

2.2. Detección de Esquinas

Las esquinas son lugares de la imagen donde, claramente, hay al menos dos orientaciones dominantes distintas del gradiente.

Las esquinas son seleccionadas en puntos donde los contornos tienen una curvatura significativa, la cual es calculada teniendo en cuenta una región que define la relevancia de la esquina. A continuación se explican algunos métodos de detección de esquinas distinguiendo los que se basan en la curvatura, en los puntos de interés y en el gradiente.

2.2.1. Métodos basados en la curvatura

Método Beaudet

Este método fue propuesto en 1978 y se basa en la curvatura de la imagen. Propone un operador invariante a la rotación, que calcula el determinante de una matriz y queda definido por:

$$DET = f_{xx}f_{yy} - f_{xy}^2 \tag{2.14}$$

donde f_{xy} es la derivada parcial de la imagen con respecto al eje x y después con respecto al eje y, y f_{xx} y f_{yy} son la segunda derivada con respecto a x e y respectivamente. Este operador presenta características de curvaturas interesantes. La detección de esquinas se realiza por umbralización

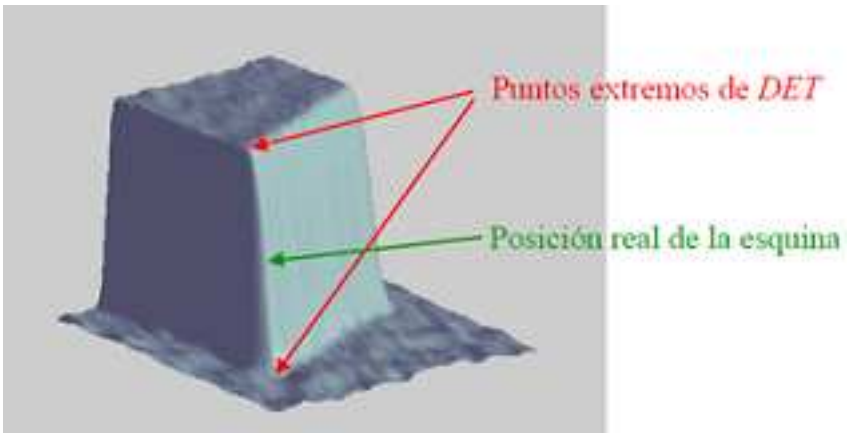

Figura 2.13: Método Beaudet

de los valores extremos de este operador. Se corresponde con el determinante de la matriz Hessiana y se relaciona con la curvatura gaussiana. En la figura 2.13 se muestra este método [33].

Método Dreschler y Nagel

Fue propuesto por Dreschler y Nagel en 1982 [46]. Detecta extremos basándose en la aplicación del concepto de curvatura gaussiana principal de una superficie. Para localizar el punto esquina realiza los siguientes pasos:

1. Calcula la curvatura gaussiana de acuerdo a la ecuacion (2.15).
2. Identifica el máximo positivo (punto elíptico) y mínimo negativo (punto hiperbólico) de la curvatura.
3. El punto esquina se encuentra en la intersección de la línea que une a los puntos elíptico e hiperbólico, con la curva K igual a 0. Esto puede apreciarse en la figura 2.14

$$K = k_{max}k_{min} = \frac{f_{xx}f_{yy} - f_{xy}^2}{(1 + f_x^2 + f_y^2)^2} \tag{2.15}$$

2.2.2. Métodos basados en puntos de interés

SUSAN

Este método fue propuesto por Smith y Brady en 1994 [40]. Determina para cada punto (núcleo) un círculo y una región con intensidad similar dentro de dicho círculo (USAN-"Univalue Segment Assimilating Nucleus").

La existencia de esquinas se determinan en función del área y del centro de gravedad del USAN.

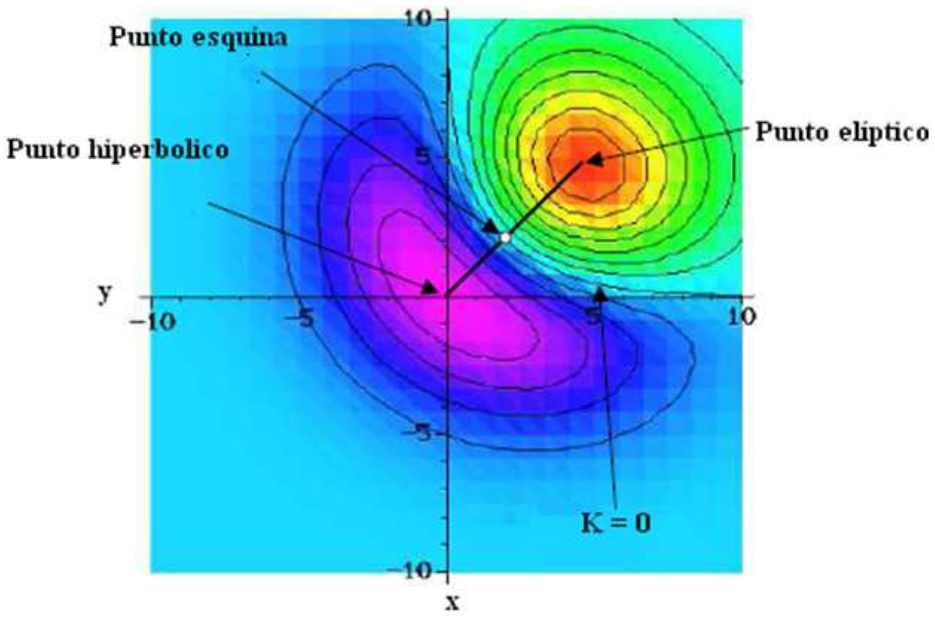

Figura 2.14: Ubicación del punto esquina del detector de Dreschler y Nagel

Si se considera la figura 2.15 se aprecia un rectángulo oscuro sobre un fondo blanco, y máscaras circulares (a, b, c, d) las cuales tienen un pixel central llamado núcleo.

El área de la máscara se puede definir como aquella superficie cuyos pixeles tienen el mismo o similar brillo a los del núcleo. Esta área, la cual contiene mucha información sobre la estructura de la imagen, es máxima cuando el núcleo se encuentra en una región plana de la superficie de la imagen. En la figura 2.16 se puede apreciar lo anteriormente mencionado.

Este enfoque tiene muchas diferencias con el resto de los detectores de esquinas, dado que no utiliza derivadas de las imágenes y no requiere de la reducción de ruido.

En la figura 2.17 se muestra un ejemplo de aplicación del operador de SUSAN.

Detector de Harris-Plessey

Busca entornos locales a cada punto (ventanas) donde, claramente, haya dos "modos de variación" bien distinguidos en la distribución del gradiente.

Su formula está dada por:

$$H = \frac{\langle I_x^2 \rangle + \langle I_y^2 \rangle}{\langle I_x^2 \rangle \langle I_y^2 \rangle - \langle I_x I_y \rangle^2} \tag{2.16}$$

Cuanto menor es H, más carácter de esquina se le concede al píxel.

Aplica supresión de no máximos para evitar respuestas múltiples alrededor de la misma esquina.

Para evitar la introducción de un umbral, se seleccionan los N píxeles con más carácter de esquina [16] [17].

En la figura 2.18 se muestra un ejemplo de aplicación del filtro de Harris-Pressey.

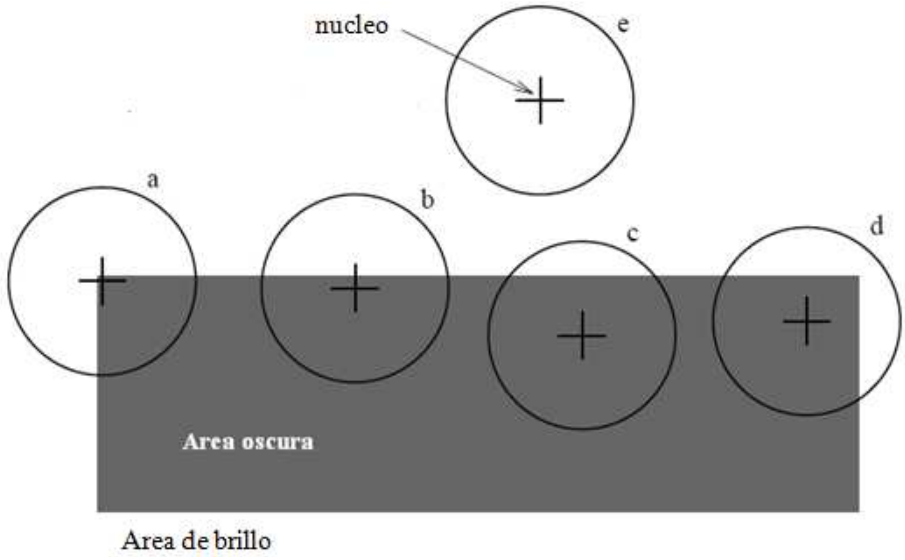

Figura 2.15: Mascaras circulares ubicadas en diferentes partes de una imagen

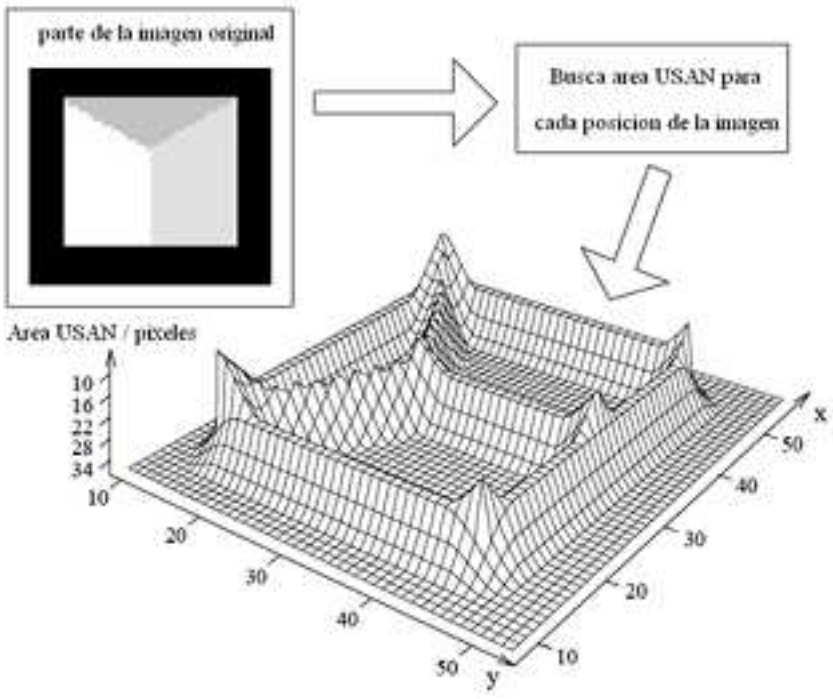

Figura 2.16: Gráfico en tres dimensiones del área USAN, considerando una pequeña parte de una imagen de prueba.

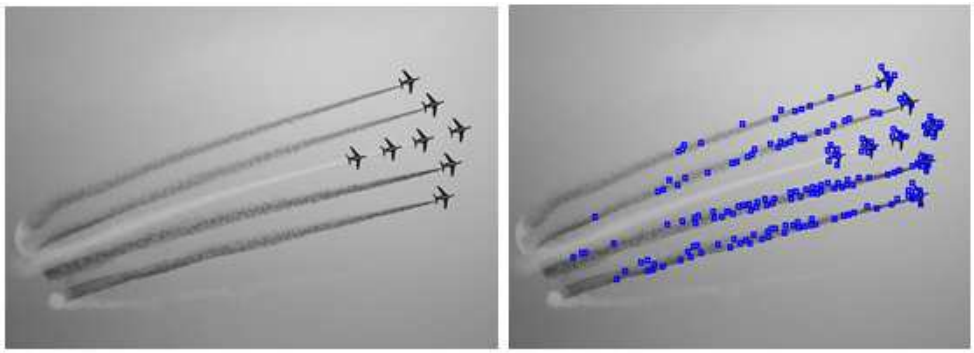

Figura 2.17: (a) Imagen original. (b) Esquinas detectadas por SUSAN

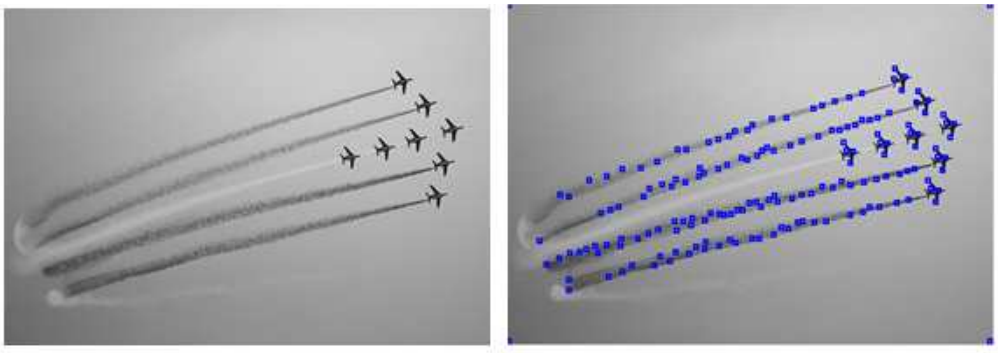

Figura 2.18: (a) Imagen original. (b) Esquinas detectadas por Harris

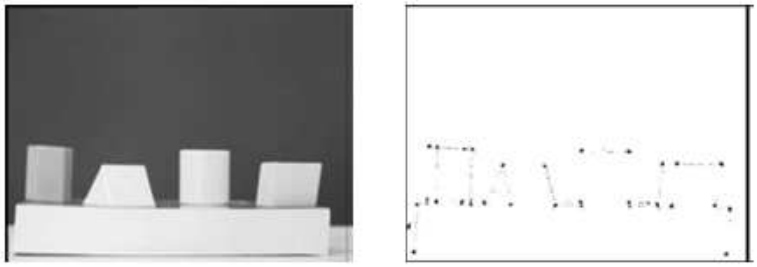

Figura 2.19: (a) Imagen. (b) Esquinas detectadas por Kitchen y Rosenfeld

2.2.3. Método basado en el gradiente

Kitchen y Rosenfeld

Es uno de los primeros detectores de esquinas que se ha desarrollado. Cambia la dirección del gradiente a lo largo del borde, y la multiplica por la magnitud del gradiente. Está definido como:

$$K = \frac{f_{xx}f_y^2 + f_{yy}f_x^2 - 2f_{xy}f_xf_y}{(f_x^2 + f_y^2)^(3/2)} \geq U_1 \tag{2.17}$$

siendo f_x, f_y las primeras derivadas y f_{xx},f_{xy}, f_{yy} las segundas derivadas. U_1 es un umbral para determinar si es una esquina [23].

Para los puntos que son máximos en la dirección del gradiente, determina los máximos locales de la expresión anterior. La figura 2.19 muestra una imagen y las esquinas detectadas por Kitchen y Rosenfeld [43].

2.3. Conclusiones

En lo que se refiere a los detectores de bordes descriptos en este capítulo puede decirse que, el detector Prewitt es ligeramente más simple de implementar computacionalmente que el detector Sobel, pero tiende a producir resultados ruidosos. Se puede observar que el coeficiente con valor 2 en el detector Sobel provee un suavizado.

En cuanto al detector Canny, tiene como principal ventaja su gran adaptabilidad para poder ser aplicado a diversos tipos de imágenes, además de no disminuir su performance ante la presencia de ruido en la imagen original. Sin embargo, algunas de las desventajas que se pueden identificar al implementar este algoritmo se encuentran en el suavizamiento, puesto que, si se aumenta el σ de la máscara se logra reducir el ruido pero se difuminan los bordes y se pierde calidad al momento de calcular la orientación.

Con respecto al detector Laplaciano, dado que es una derivada de segundo orden, es inaceptablemente sensible al ruido. La magnitud del Laplaciano produce bordes dobles, lo cual es un efecto indeseado ya que dificulta la segmentación. Finalmente, el Laplaciano es incapaz de detectar la dirección de los bordes (no es isotrópico).

En cuanto al operador Laplaciano de la gaussiana, el filtro obtenido resulta bastante costoso en tiempo. La convolución del filtro gaussiano con el laplaciano reduce el efecto del ruido al suavizar la imagen. Utilizando este operador se detectan bordes en todas las direcciones. Se puede trabajar a diferentes escalas al variar el valor de la desviación estándar de la Gaussiana. Cuanto mayor es esta desviación, habrá un menor número de pasos por cero. Resumiendo, el propósito de la Gaussiana en el operador *LoG* es suavizar la imagen (reducir el ruido), mientras que el propósito del operador Laplaciano es proveer de información sobre los cruces por cero y establecer la localización de los bordes.

Por otra parte, en cuanto a los detectores de esquinas, puede concluirse que, el detector SUSAN es preciso y rápido pero sin embargo es poco robusto; en la localización de esquinas resulta ser inadecuado para imágenes borroneadas (blurred), pero bueno para el resto de los casos.

El detector de esquina Harris-Plessey es ampliamente utilizado gracias a su tasa de detección mejorada. Sin embargo, es computacionalmente exigente, no es invariante a la rotación y es sensible al ruido dado que se basa en la información del gradiente.

El detector de Dreschler y Nagel posee algunos incovenientes: la distribución de los puntos elípticos, hiperbólicos y parabólicos depende del ángulo de la esquina y del difuminado, el ángulo de la esquina depende del punto de vista; y el difuminado depende del enfoque. Además es muy sensible al ruido dado que se basa en segundas derivadas.

El detector de Kitchen presenta la desventaja de que la dirección del gradiente en la cercanía de la esquina no es continua, por lo que podría estar mal definida.

Finalmente, la función *DoG* resulta ser una aproximación del *LoG*, aunque mucho más eficiente. Las limitaciones comunes de estos operadores están dadas porque los máximos locales que pueden ser detectados en las cercanías de los contornos o líneas rectas donde el cambio se produce en una sola dirección.

Capítulo 3

Métodos para la generación de descriptores

3.1. Introducción

La correspondencia entre imágenes constituye un aspecto fundamental de muchos problemas en visión de computadoras, incluyendo reconocimiento de objetos o escenas, reconstrucción de estructuras 3D a partir de múltiples imágenes, correspondencia estéreo y seguimiento de objetos en movimiento.

Para poder lograr la correspondencia anteriormente mencionada, se deben determinar aquellas características distintivas de las imágenes. Estas se representan por descriptores, calculados sobre pequeñas regiones de interés de una imagen [29].

Se desea que los descriptores sean altamente distintivos e invariantes a cambios de iluminación, puntos de vistas, rotación, etc; para poder lograr una correspondencia exitosa. Dicha correspondencia se puede comparar con los buscadores de contenido textual.

Este capítulo describe algunos de los métodos para la detección de los puntos característicos de una imagen y la menera de construir los descriptores correspondientes.

3.2. SIFT (Scale Invariant Features Transform)

SIFT es uno de los métodos más utilizados [26]. El algoritmo toma una imagen y la transforma en una colección de vectores de características locales. Cada uno de estos vectores, es distintivo e invariante a cualquier escala, rotación y traslación de la imagen [8].

En la puesta en práctica, estas características pueden ser usadas para encontrar objetos distintivos en imágenes diferentes.

Consta de cuatro etapas principales:

1. Selección de extremos en el espacio-escala

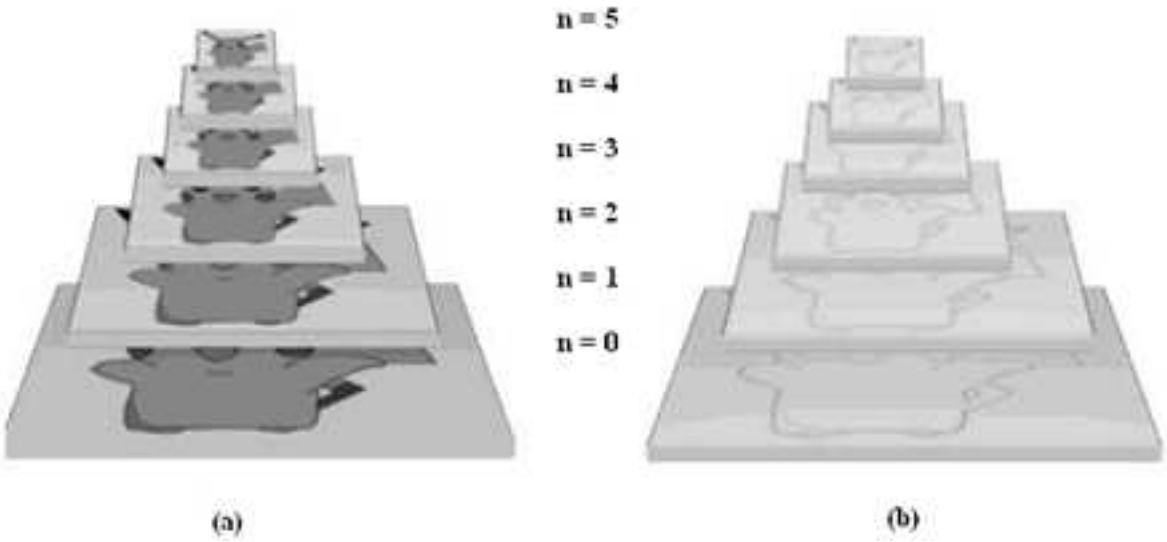

Figura 3.1: (a) Pirámide Gaussiana. (b) Pirámide Laplaciana

2. Identificación de los puntos de interés
3. Asignación de las orientaciones
4. Construcción de los descriptores.

En la primera etapa, los potenciales puntos claves se identifican a través del procesamiento de la imagen en su posición y escala. Esto es implementado eficientemente mediante la construcción de una pirámide Gaussiana y la posterior construcción de la pirámide Laplaciana (DoG), generada mediante la sustracción de las imágenes de la pirámide Gaussiana. Para cada imagen de la pirámide DoG, se compara cada pixel con sus ocho vecinos más los 9 vecinos de las imágenes anterior y posterior, obteniendo de esta manera los extremos locales (aquellos que sean más grandes o más pequeños que el resto). En la figura 3.1 se observan las pirámides mencionadas.

En la segunda etapa, se descartan aquellos puntos claves que tengan bajo contraste y/o aquellos que se encuentren cercanos al borde.

En la tercera etapa para cada punto clave se calcula la orientación dominante. Puede suceder que un mismo punto tenga más de una orientación.

Por último, la cuarta etapa construye para cada punto clave un descriptor de características, el cual se utiliza para la correspondencia entre imágenes [22].

3.3. PCA-SIFT

El algoritmo de PCA-SIFT se basa en el código original del descriptor SIFT, modificando solo la cuarta etapa del mismo, que es la generación de descriptores de puntos claves.

Acepta la misma entrada que el descriptor SIFT: la ubicación del sub-píxel, la escala, y las orientaciones dominantes del punto clave.

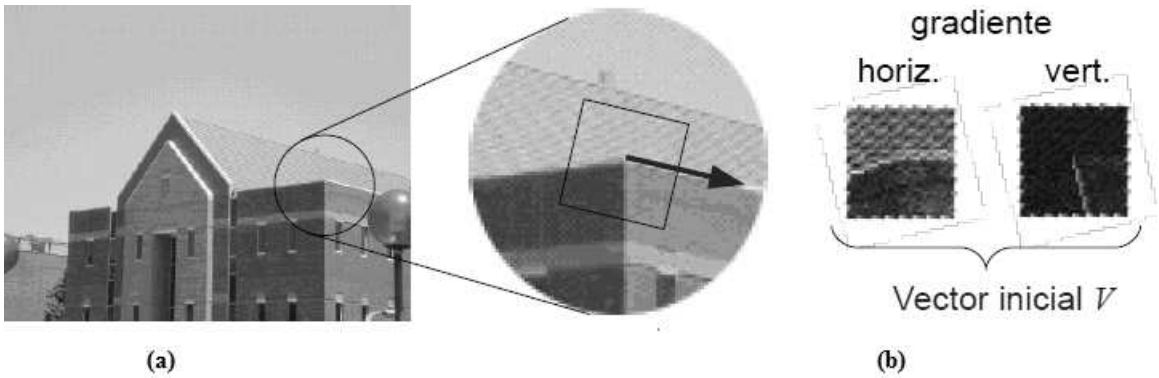

Figura 3.2: (a) Orientación del punto. (b) Gradientes horizontales y verticales

Luego, para cada punto clave, se extrae un área de 41x41 en la escala dada, centrada en dicho punto y girada para alinear su orientación dominante a su dirección canónica. Se codifican los gradientes en las direcciones x e y por separado. En la figura 3.2 se muestra lo mencionado anteriormente.

El resultado del algoritmo es, por cada punto clave, un vector de dimensión 3042 que luego se reduce a dimensión 32 utilizando PCA (Principal Component Analysis). Esta es una técnica estándar para la reducción de dimensionalidad y se ha aplicado a una amplia variedad de problemas de visión por computadora, incluyendo la selección de características, reconocimiento de objetos y reconocimiento de rostros [22] [21] [45].

3.4. SURF (Speeded Up Robust Features)

Es un detector y descriptor de imagen robusto, presentado por primera vez por Herbert Bay en el año 2006 [6], que puede ser utilizado en tareas de visión por computadora, como reconocimiento de objetos o de reconstrucción 3D.

En parte está inspirado por el descriptor SIFT. La versión estándar del SURF es varias veces más rápido que SIFT y según sus autores, es más robusto que SIFT frente a las diferentes transformaciones de la imagen.

Describe una distribución de las respuestas Haar wavelet en el vecindario del punto de interés. Los espacios de escalas son implementados como una pirámide, las imágenes son suavizadas con un filtro gaussiano y luego submuestreadas para alcanzar un nivel más alto de la pirámide. En la implementación de SIFT se resta esta pirámide para conseguir la pirámide de diferencias gaussianas (DOG) en la cual se encuentran los puntos claves.

Debido al uso de los filtros e imágenes integrales, en SURF no se aplica iterativamente el mismo filtro a la salida de una capa previamente filtrada, sino que se aplica el filtro, de tamaño variante, a la imagen original. Por lo tanto, el espacio de la escala se analiza por el aumento de escala del tamaño filtro en lugar de reducir iterativamente el tamaño de la imagen. Esto puede verse en la figura 3.3.

Para los descriptores se utilizan 64 dimensiones, reduciendo el tiempo de cómputo de la función

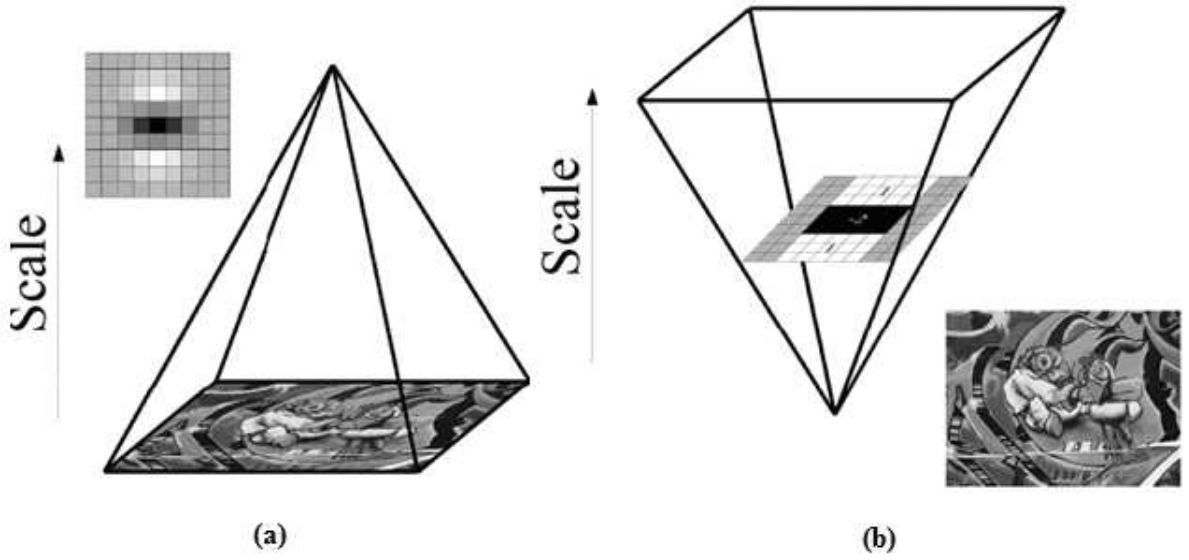

Figura 3.3: (a) SIFT. (b) SURF

y el aumento al mismo tiempo de la robustez. También constituyen un paso de indexación nuevo basado en el signo de la laplaciana, lo que aumenta no sólo la velocidad de la correspondencia, sino también la solidez de los descriptores.

El propósito del descriptor SURF se basa en propiedades similares a las del descriptor SIFT, pero con más complejidad. El primer paso consiste en fijar orientaciones basadas en la información de una región circular alrededor del punto de interés. Luego, se construye una región cuadrada alineada con la orientación seleccionada, y se extrae el descriptor SURF de la misma. SURF es, hasta cierto punto, similar en su concepción a SIFT, ya que ambos se centran en la distribución espacial de la información del gradiente.

SURF tiene varios tipos de descriptores de longitud variable. El uso regular de SURF es de longitud 64. En SURF-128 la longitud del descriptor se ha duplicado, y U-SURF (donde la invarianza de la rotación de los puntos de interés ha quedado fuera), la longitud del descriptor es 64.

3.5. RIFT (Rotation Invariant Feature Transform)

El método RIFT es una generalización de SIFT que hace énfasis en la invarianza a la rotación. Este enfoque permite encontrar partes expresivas y geométricamente invariantes en el modelado de objetos 3D. Se basa en la identificación de grupos de regiones locales afines (características de la imagen que tienen un aspecto característico y forma elíptica) que permanecen aproximadamente rígidas en una serie de puntos de vista de un objeto, y a través de varias instancias de la misma clase de objeto. Estos grupos, denominados partes afines semilocales, son aprendidos mediante la búsqueda de correspondencia entre pares de imágenes de entrada no segmentadas y desordenadas, seguido por una validación contra imágenes adicionales. Se procura identificar grupos de regiones afines locales vecinas cuya apariencia y configuración espacial permanecen estables en múltiples instancias.

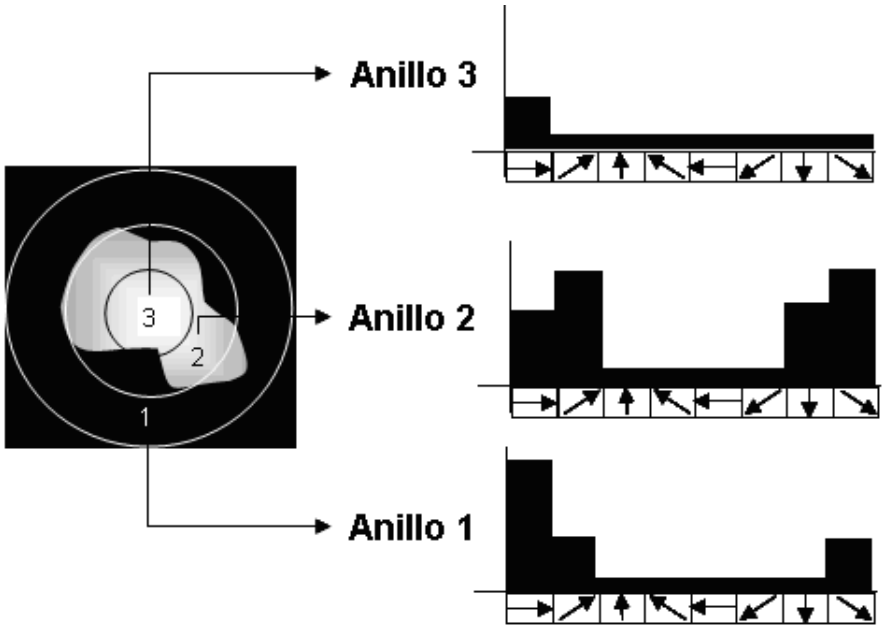

Figura 3.4: Descriptores del método RIFT

A diferencia de SIFT, la orientación está representada por 8 intervalos distintos, por lo que el descriptor final tiene un total de 24 entradas. Esto se presenta en la figura 3.4.

Este enfoque ha sido aplicado al reconocimiento de mariposas en imágenes naturales [44].

3.6. G-RIF (Generalized Robust Invariant Feature)

Es un descriptor que combina la información perceptiva con la codificación espacial. Fue propuesto como un método basado en el contexto para el reconocimiento de objetos. Inicialmente se desarrolló un modelo computacionalmente eficiente basado en los últimos descubrimientos de los mecanismos neuronales y cognitivos de los sistemas visuales humanos. Este enfoque se compone de la detección, descripción y clasificación de la parte visual. El primer paso es la detección de la parte visual (orientación espacial y bordes).

Para que el reconocimiento de objetos sea exitoso, las partes visuales deben cumplir algunos requisitos. En primer lugar, la información del fondo debería evitarse mientras sea posible. En segundo lugar, deben ser perceptualmente significativas. Por último, deben ser resistentes a los efectos de las distorsiones geométricas y fotométricas.

La descripción de las piezas visuales es muy importante para la identificación o clasificación de objetos. Dos factores importantes en el diseño del descriptor son la selectividad y la invarianza. La selectividad significa que partes diferentes o clases de partes pueden ser discriminadas fuertemente. La invarianza significa que partes iguales o clases de partes pueden ser detectadas a pesar de las distorsiones fotométricas y geométricas. Los descriptores deben satisfacer ambas propiedades.

Se encontró que la forma óptima del detector de parte visual es una combinación de un detector

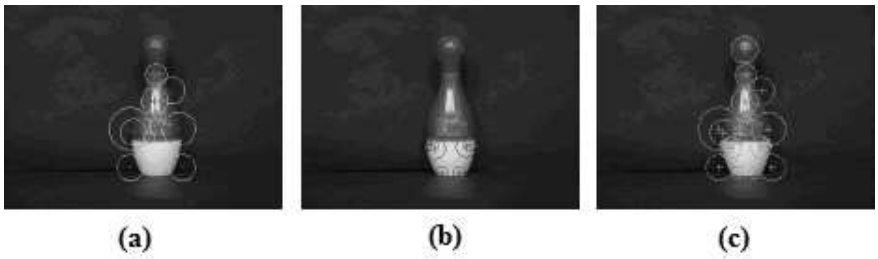

Figura 3.5: Descriptores del método RIFT

de simetría radial y de un detector de esquinas, como se muestra en la figura 3.5. El resultado final presenta un descriptor de contexto general que engloba la orientación de bordes, la densidad de bordes y la información de tonalidad en una forma unificada [42].

3.7. Conclusiones

La correspondencia de características locales se ha convertido en el método más usado para comparar imágenes. Varios métodos han sido propuestos. SIFT, con su alta precisión y relativamente bajo tiempo de cómputo, se ha convertido en el estándar. Se han realizado algunos intentos de mejoras al algoritmo, como PCA-SIFT. El más reciente es SURF.

Los algoritmos de características locales han demostrado ser una buena opción para el análisis de correspondencias de imágenes en plataformas móviles, ya que tanto oclusiones u objetos perdidos pueden ser manejados. En particular, la aplicación de SIFT en imágenes panorámicas ha dado buenos resultados en ambientes interiores y también en cierta medida en ambientes al aire libre. Sin embargo, los ambientes al aire libre son muy diferentes de los ambientes interiores. Hay un número de factores que pueden alterar la apariencia de una escena al aire libre: las condiciones de iluminación, sombras, cambios estacionales, etc. Todos estos aspectos hacen que la correspondencia entre imágenes se haga difícil.

Si bien los detectores para encontrar los puntos de interés de una imagen en los métodos SIFT y SURF, trabajan de manera diferente, la salida, en ambos casos, es la representación del vecindario alrededor de un punto de interés como un vector descriptor. Estos descriptores son los que luego se comparan con los extraídos en otra imagen para verificar si existen correspondencias. SIFT usa un descriptor de longitud 128 y dependiendo de la aplicación existen diferentes estrategias de comparación. El método común, propuesto por Lowe, consiste en calcular el vecino más cercano a una característica y a continuación comprobar si el segundo vecino más cercano se aleja más que un cierto umbral. Otra estrategia es considerar sólo el vecino más cercano si la distancia es menor que un umbral, o calcular solo el vecino más cercano utilizando una aproximación árbol k-dimensional.

SURF utiliza el mismo esquema de correspondencia que SIFT, pero con algunas mejoras. Esto incluye la señal del Laplaciano, es decir, permite una rápida distinción entre características claras en fondo oscuro y características oscuras en fondo claro. Por otro lado presenta cierta inestabilidad a la rotación y cambios de iluminación.

En comparación con la representación estándar, PCA-SIFT es más distintivo y más compacto, con lo cual lleva a lograr mejoras significativas en la exactitud de la correspondencia (y la veloci-

dad), tanto para condiciones controladas como para las del mundo real. Sin embargo presenta baja performance para imágenes borrosas y/o de poca resolución.

El método RIFT es conveniente para modelar una amplia gama de transformaciones en 3D, incluyendo el cambio de punto de vista y las deformaciones no rígidas.

El método G-RIF muestra una mejor performance en el reconocimiento de objetos en entornos severos (poca iluminación y/o bajo contraste) que el resto de los métodos.

Finalmente, SIFT demuestra ser la mejor opción dado su fuerte invarianza a los cambios de escala y rotación y a su invarianza parcial frente a cambios de puntos de vista e iluminación.

Capítulo 4

SIFT (Scale Invariant Features Transform)

4.1. Introducción

Este capítulo presenta una descripción detallada del método SIFT (Scale Invariant Feature Transform) el cual fue brevemente introducido en el capítulo anterior.

SIFT extrae características invariantes distintivas de las imágenes utilizadas para el reconocimiento de objetos en diferentes vistas de los mismos. Fue publicado por David Lowe en 1999 [26] [27] y patentado en 2004.

Las características que extrae el método SIFT son invariantes a la escala y rotación de la imagen. Además son robustas a cambios en la iluminación, oclusión parcial, ruido y pequeños cambios en el punto de vista. Son altamente distintivas, fácil de extraer y permiten la correcta identificación de objetos con baja probabilidad de error [18].

4.2. Descripción general del algoritmo

El algoritmo analiza una imagen a través del espacio escala utilizando el método piramidal con sucesivos filtros Gaussianos, formando una pirámide Gaussiana. Luego, construye una pirámide Laplaciana formada por las diferencias de Gaussianas (DoG). Esta última se utiliza para la detección de máximos y mínimos, los cuales son los potenciales puntos claves. Del conjunto de puntos claves obtenidos, se descartan aquellos que sean inestables y para los que resulten estables se genera un descriptor. Los descriptores representan la información local de la imagen alrededor del punto clave de tal forma que es invariante a los cambios en escala, rotación y pequeñas variaciones de iluminación, lo que significa que las mismas características van a ser detectadas en la imagen aún si los objetos han sido rotados o son vistos desde una perspectiva diferente o ha cambiado la distancia [27] [15] [20].

A continuación se detallan las etapas principales usadas para generar el conjunto de características de las imágenes.

4.3. Detección de puntos extremos

El primer paso del algoritmo consiste en la detección de los puntos claves. Para ello se identifican las posiciones y las escalas que pueden ser asignadas repetidamente bajo diferentes puntos de vista del mismo objeto.

La detección de posiciones invariantes a los cambios de escala de la imagen puede lograrse mediante la búsqueda de características estables entre todas las escalas posibles usando una función de escala continua conocida como espacio escala. El espacio escala de una imagen está definido como una función $L(x, y, \sigma)$, que se produce mediante la convolución de una Gaussiana de escala variable, $G(x, y, \sigma)$, con una imagen de entrada, $I(x, y)$. Esto se observa en la ecuación (4.1)

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{4.1}$$

donde $*$ es la operación de convolución en x e y y $G(x, y, \sigma)$ se define como en (4.2)

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma 2} e^{(} - \frac{x^2 + y^2}{2\sigma^2} \tag{4.2}$$

Para detectar eficientemente la ubicación de puntos estables en el espacio escala se calculan los puntos extremos en la función diferencia de Gaussianas convolucionada con la imagen, $D(x, y, \sigma)$, la cual puede ser calculada a partir de las diferencias entre dos escalas vecinas separadas por un factor constante k. La función de detección queda como en la ecuación (4.3)

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma) \tag{4.3}$$

donde L es una imagen suavizada con un filtro gaussiano.

Para cada octava del espacio escala, la imagen inicial es convolucionada repetidamente con Gaussianas, para producir el conjunto de imágenes del espacio escala como se muestra a la izquierda de la figura 4.1. Las imágenes Gaussianas adyacentes son restadas para producir las imágenes de diferencias de Gaussianas como se muestra a la derecha de dicha figura 4.1. Luego de cada octava, la imagen Gaussiana es sub-muestreada a la cuarta parte y el proceso se repite nuevamente [37] [32].

Un enfoque eficiente para construir $D(x, y, \sigma)$ se muestra en la figura 4.1. La imagen inicial es incrementalmente convolucionada con Gaussianas para producir imágenes separadas por un factor constante k en el espacio escala. Cada octava del espacio escala se divide (es decir, se duplica σ) en un numero entero, s, de intervalos, tal que $k = 2^{1/s}$

Se producen $s + 3$ imágenes en la pila de imágenes suavizadas para cada octava, por lo que la detección extrema final abarca una octava completa. Las escalas de las imágenes adyacentes son restadas para obtener las imágenes de diferencias de Gaussianas. Una vez que ha sido procesada una octava completa, la siguiente comienza con la imagen Gaussiana que se encuentra en la mitad de la octava anterior reduciendo su tamaño a la cuarta parte; para lo cual, sólo se toman las filas y columnas pares. La exactitud del muestreo en relación a σ no es diferente para el inicio de la octava anterior, mientras que el cálculo se reduce considerablemente.

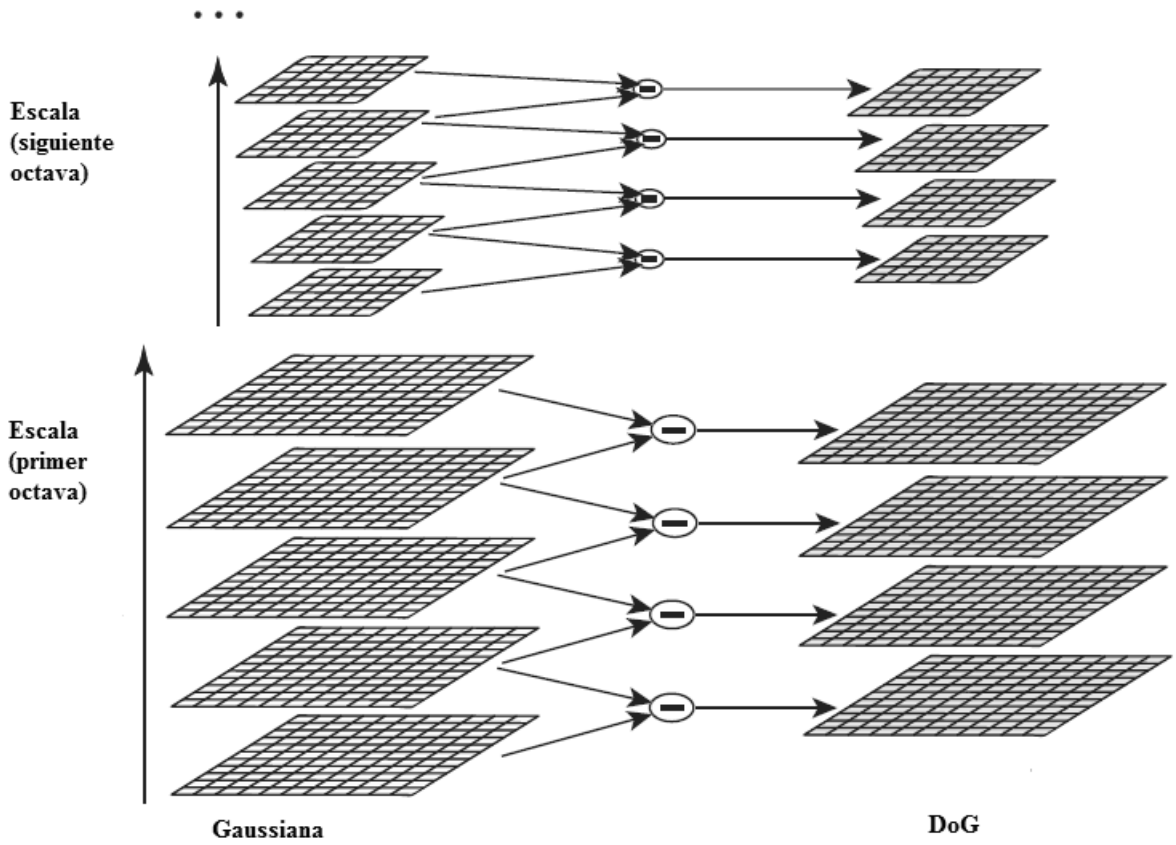

Figura 4.1: Pirámide Gaussiana y Diferencias de Gaussianas (DoG) del espacio escala

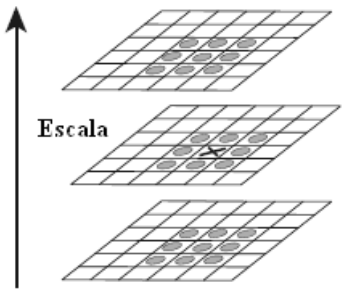

Figura 4.2: Detección de máximos y mínimos

4.3.1. Detección de extremo local

Para detectar el máximo y mínimo local de $D(x, y, \sigma$, cada punto es comparado con sus ocho vecinos en la imagen actual y nueve vecinos en la escala anterior y posterior, como se muestra en la figura 4.2. Este es seleccionado solo si es más grande o más pequeño que todos sus vecinos. El costo de este chequeo es razonablemente bajo ya que antes de realizarlo se descartan los puntos de muestreo con bajo contraste y cercanos al borde, reduciendo la cantidad de puntos a analizar.

4.4. Localización exacta de puntos claves

Una vez que un punto clave candidato se ha encontrado mediante la comparación de un píxel con sus vecinos, el siguiente paso es realizar un ajuste detallado de los datos de los vecinos para la posición, escala y proporción de las curvaturas principales. Esta información permite rechazar los puntos con bajo contraste (sensibles al ruido) o que están mal localizados a lo largo de un borde.

La implementación inicial de este enfoque simplemente localiza puntos claves en la posición y escala del punto central de la muestra. Sin embargo, Brown ha desarrollado un método [7] para el ajuste de una función cuadrática 3D a los puntos de muestreo local para determinar la posición interpolada del máximo, y sus experimentos demostraron que esto proporciona una mejora sustancial en la correspondencia y la estabilidad. Su método utiliza el desarrollo de Taylor (hasta los términos cuadráticos) de la función del espacio escala, $D(x, y, \sigma)$, modificado de manera que el origen está en el punto de muestreo (4.4)

$$D(x) = D + \frac{\partial D^T}{\partial x}x + \frac{1}{2}x^T\frac{\partial^2 D}{\partial x^2}x \tag{4.4}$$

donde D y sus derivadas son evaluadas en el punto de muestra y $x = (x, y, \sigma)^T$ es el desplazamiento desde ese punto. La posición del extremo, $\hat{x}$, se determina tomando la derivada de esta función con respecto a x e igualando a cero, obteniendo (4.5)

$$\hat{x} = -\frac{\partial^2 D^{-1}}{\partial x^2}\frac{\partial D}{\partial x} \tag{4.5}$$

Según lo sugerido por Brown, el Hessiano y derivada de D se aproximan utilizando las diferencias

Figura 4.3: Imagen original

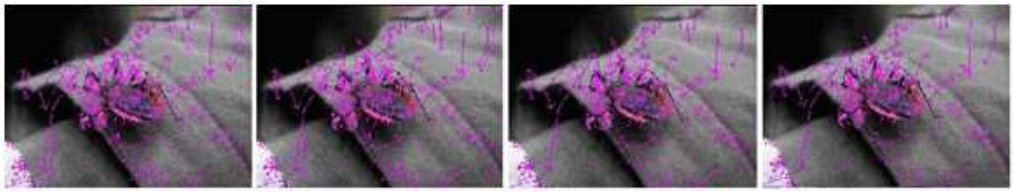

Figura 4.4: (a) Todos los puntos claves detectados. (b) Puntos claves luego de descartar los de bajo contraste. (c) Puntos claves luego de descartar los cercanos al borde. (d) Puntos claves luego de descartar los de bajo contraste y cercanos al borde

de puntos vecinos de muestreo. El sistema lineal resultante de 3x3 se puede resolver con un costo mínimo. Si el desplazamiento $hatx$ es mayor que 0.5 en cualquier dimensión entonces significa que el extremo se encuentra más cercano a un punto de muestreo diferente. El desplazamiento final $\hat{x}$ es agregado a la posición de ese punto de muestra para la interpolación estimada de la posición del extremo. El valor de la función en el extremo, $D(\hat{x})$, es útil para rechazar extremos con bajo contraste. Esto se obtiene sustituyendo las ecuaciones (4.4) y (4.5), quedando (4.6)

$$D(\hat{x}) = D + \frac{1}{2}\frac{\partial D^T}{\partial x}\hat{x} \tag{4.6}$$

La figura 4.5 muestra los efectos de la selección de puntos claves en una imagen de un insecto. Se utiliza una imagen de 320 x 240 pixeles y se muestran los puntos claves como vectores dando la posición, la escala, y la orientación de cada punto clave. La figura 4.3.(a) muestra la imagen original de la que se obtienen los puntos claves. La figura 4.4.(a) muestra los 350 puntos claves máximos y mínimos detectados de la función de diferencias de gaussianas. En la figura 4.4.(b) se muestran los 328 puntos claves restantes removiendo aquellos con bajo contraste. La figura 4.4.(c) muestra los 300 puntos donde los cercanos al borde fueron descartados. La figura 4.4.(d) muestra los 286 puntos resultantes de haber eliminado los de bajo contraste y los cercanos al borde.

4.5. Eliminación de bordes

Para la estabilidad no es suficiente con rechazar los puntos con bajo contraste. La función de diferencia de Gaussianas tiene una respuesta firme a lo largo de los bordes, incluso si la posición a lo largo del borde no está bien determinada y por lo tanto inestable a pequeñas cantidades de ruido. Un pico mal definido en la función de diferencia de gaussianas tendrá una gran curvatura principal a través del borde, pero una pequeña en la dirección perpendicular. Las curvaturas principales pueden ser calculadas desde una matriz Hessiana, H, de 2x2 calculada en la posición y escala del punto clave como se muestra en (4.7):

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{4.7}$$

Las derivadas son estimadas tomando las diferencias entre puntos vecinos. Los valores propios de H son proporcionales a las curvaturas principales. Sea α el valor propio con la mayor magnitud y β el más pequeño, se puede calcular la suma de los valores propios a partir de la traza de H (4.8) y el producto a partir de los determinantes (4.9):

$$Tr(H) = D_{xx} + D_{yy} = \alpha + \beta \tag{4.8}$$

$$Det(H) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta \tag{4.9}$$

En el caso de que el determinante sea negativo, lo cual es poco probable, las curvaturas tienen signos diferentes de modo que el punto se descarta por no ser un extremo. Sea r la relación entre el valor propio de la magnitud más grande y la más pequeña, de modo que $\alpha = r\beta$, tomando las ecuaciones (4.8) y (4.9) se llega a la ecuación (4.10)

$$\frac{Tr(H)^2}{Det(H)} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r} \tag{4.10}$$

La ecuación (4.10) solo depende de la relación de los valores propios en lugar de sus valores individuales. La cantidad $(r+1)^2/r$ es mínima cuando los dos valores propios son iguales y se incrementa con r. Por lo tanto, para comprobar que la relación de las curvaturas principales está por debajo de cierto umbral, r , sólo es preciso verificar que se cumpla la ecuación (11)

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r} \tag{4.11}$$

Se utiliza $r = 10$, que elimina puntos claves que tienen una relación entre las curvaturas principales mayor que 10. La transición de la figura 4.4.(b) a la 4.4.(c) muestra los efectos de esta operación [36] [27].

4.6. Asignación de la orientación

En base a las propiedades de la imagen local se asigna una orientación a cada punto clave de manera que el descriptor correspondiente pueda ser representado en relación a dicha orientación. De esta forma, el descriptor será invariante a la rotación de la imagen.

La escala del punto clave es usada para seleccionar la imagen suavizada Gaussiana, L, cuya escala esté más cerca, para que todos los cálculos sean realizados de manera invariante a la escala. Para cada muestra de imagen, $L(x, y)$, en esa escala, la magnitud del gradiente, $m(x, y)$, y la orientación $\theta(x, y)$ se calculan usando diferencia de pixeles como se muestra en las ecuaciones (4.12) y (4.13) respectivamente:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \tag{4.12}$$

$$\theta(x,y) = tan^{-1}\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right) \tag{4.13}$$

Luego se considera una región alrededor del punto clave y se construye un histograma con las orientaciones de los puntos de dicha región. El histograma tiene 36 niveles de discretización con el objetivo de cubrir todas las posibles orientaciones (360 grados).

Los picos en el histograma corresponden a las direcciones dominantes de los gradientes locales. Se detecta el pico más alto junto con los que estén dentro del 80 % del pico antes mencionado y se usa para crear un punto clave con esa orientación. Por lo tanto, para posiciones con múltiples picos de magnitud similar, habrá múltiples puntos claves creados en la misma posición y escala, pero con orientaciones diferentes [27].

4.7. Descriptor de la imagen local

Las operaciones anteriores han asignado una posición, una escala y una orientación a cada punto clave de la imagen. El próximo paso es calcular un descriptor para la región de imagen local que sea altamente distintivo, tan invariante como sea posible a las variaciones restantes, tales como cambios en la iluminación o punto de vista 3D.

4.7.1. Representación del descriptor

La figura 4.5 ilustra el cálculo del descriptor de puntos claves. Primero las magnitudes del gradiente de la imagen y orientaciones son probadas alrededor de la posición del punto clave, usando la escala del mismo para seleccionar el nivel de borroneado Gaussiano para la imagen. A fin de lograr la invariancia de orientación, las coordenadas del descriptor y las orientaciones del gradiente se rotan en relación con la orientación del punto clave. Por eficiencia, los gradientes son precalculados para todos los niveles de la pirámide. Estos están ilustrados con pequeñas flechas en cada posición de la figura 4.5.(a).

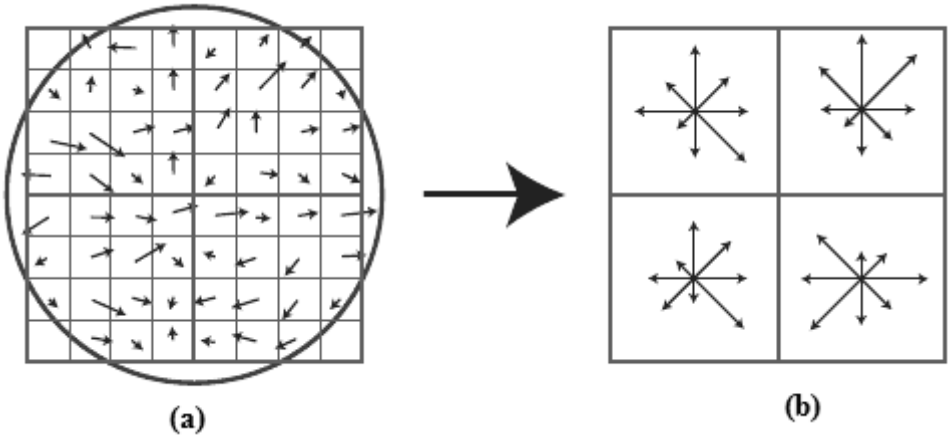

Figura 4.5: (a) gradientes de la imagen, (b) descriptor del punto clave

Un descriptor se crea mediante el cálculo de la magnitud del gradiente y la orientación en cada punto de muestreo de la imagen en una región alrededor de la ubicación del punto clave, como se muestra en la figura 4.5.(a). Estos son ponderados por una ventana Gaussiana, indicada por el círculo superpuesto. Estas muestras se acumulan en los histogramas de orientación que resumen el contenido sobre subregiones de 4x4, como se muestra en la figura 4.5.b), con la longitud de cada flecha correspondiente a la suma de las magnitudes del gradiente cerca de esa dirección dentro la región.

Una función de ponderación de Gauss se utiliza para asignar un peso a la magnitud de cada punto de muestreo. El propósito de esta ventana Gaussiana es evitar los cambios bruscos en el descriptor con pequeños cambios en la posición de la ventana y dar menos énfasis a los gradientes que son lejanos al centro del descriptor, ya que estos son los que más probabilidad tienen de ser erróneos. Esto permite cambios importantes en las posiciones de gradientes creando histogramas de orientación sobre regiones de muestra de 4x4. En la figura 4.5.(b) se observan ocho direcciones para cada histograma de orientación, con la longitud de cada flecha correspondiente a la magnitud de la entrada del histograma.

Cada entrada en un nivel de discretización se multiplica por un peso de $1-d$ para cada dimensión, donde d es la distancia de la muestra desde el valor central del nivel de discretización medida en unidades del espaciado de niveles del histograma.

El descriptor está formado por un vector que contiene los valores de todas las entradas del histograma de orientación, que corresponden a las longitudes de las flechas de la figura 4.5.(b).

Se utilizan $4 * 4 * 8 = 128$ elementos del vector de características para cada punto clave. Finalmente, el vector de características es modificado para reducir los efectos del cambio de iluminación. Primero, el vector se normaliza a la longitud unidad. Luego se realiza un cambio de contraste de la imagen multiplicando el valor de cada pixel por una constante. Nótese que los gradientes serán multiplicados por la misma constante, por lo que este cambio de contraste será cancelado por la normalización del vector. También se produce un cambio de brillo añadiendo una constante a cada píxel de la imagen. Esto tampoco afecta los valores del gradiente, ya que son calculados a partir de diferencias de píxeles. Por lo tanto el descriptor es invariante a los cambios en la iluminación [28].

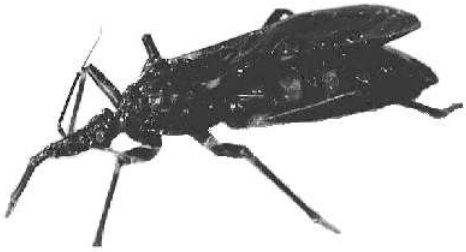

Figura 4.6: Objeto a buscar

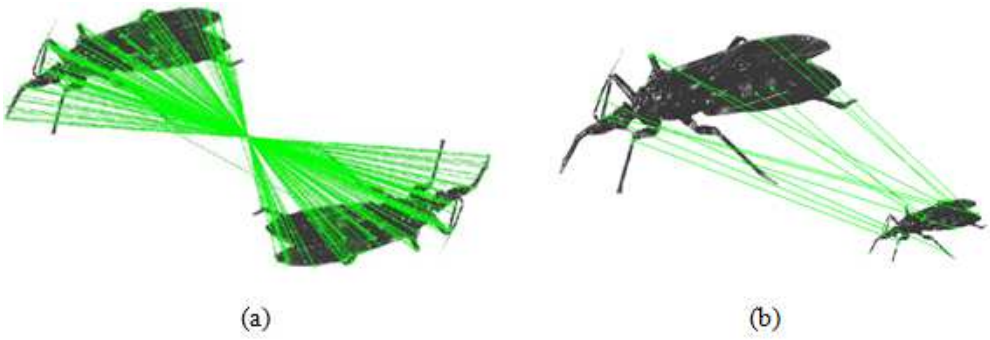

Figura 4.7: (a) Correspondencias entre una imagen y la misma rotada. (b) Correspondencias entre una imagen y la misma con escala menor.

Sin embargo, los cambios de iluminación no lineales también pueden ocurrir debido a la saturación de la cámara o a los cambios de iluminación que afectan a las superficies 3D con diferentes orientaciones en diferentes cantidades. Estos efectos pueden causar un gran cambio en las magnitudes relativas de algunos gradientes, pero tienen menos probabilidades de afectar a las orientaciones del gradiente. Por lo tanto, se reduce la influencia de las grandes magnitudes de gradientes haciendo una umbralización de los valores del vector de características para que no supere el valor de 0.2, y luego se realiza una renormalización [27].

4.8. Pruebas de correspondencias

En esta sección se muestran algunas pruebas realizadas con diferentes imágenes. La figura 4.6 muestra el objeto (modelo) a buscar. La figura 4.7.(a) muestra las correspondencias entre la imagen del objeto a buscar y la misma imagen rotada. La figura 4.7.(b) muestra las correspondencias entre la imagen del objeto a buscar y la misma imagen en una escala reducida.

La figura 4.8 muestra las correspondencias entre imágenes que se encuentran en diferentes puntos de vista. La figura 4.9 muestra las correspondencias entre un logo y el mismo en una pagina web.

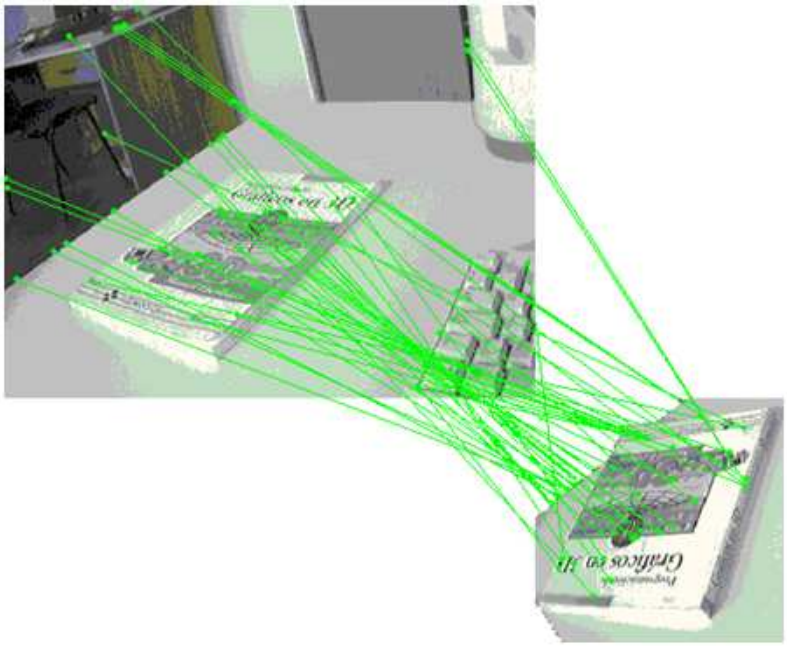

Figura 4.8: Correspondencia entre imágenes con diferentes puntos de vista

Figura 4.9: Correspondencias entre un logo y el mismo en una pagina web.

4.9. Conclusiones

Los descriptores SIFT detallados en este capítulo son especialmente útiles debido a su carácter distintivo, lo que permite la correspondencia de objetos en diferentes imágenes.

Los descriptores han demostrado ser invariantes a la rotación de la imagen, la escala y robustos a la distorsión afín, ruido y cambios en la iluminación.

El método SIFT es uno de los más utilizados, sin embargo, su costo computacional es alto si se piensa en su aplicación en tareas tales como tracking de video.

En el siguiente capítulo se exponen conceptos básicos de paralelismo, para luego proponer una implementación paralela del algoritmo que permita reducir los tiempos de ejecución.

Capítulo 5

Conceptos de Paralelismo

5.1. Introducción

Tradicionalmente los programas de computadoras se han desarrollado para el cálculo secuencial. Esto es, funcionan en una computadora con un única CPU, un problema es dividido en un conjunto de instrucciones, las instrucciones son ejecutadas secuencialmente y solo una instrucción es ejecutada por vez.

La computación paralela surgió como la alternativa válida y natural para cubrir la insistente demanda de mayor desempeño (performance). Su incorporación como paradigma de computación no fue tan natural. La mayoría de los estudios teóricos y prácticos sobre computación paralela, generalmente están enfocados a analizar diferentes arquitecturas y lenguajes de programación o herramientas paralelas reales. El diseño de programas paralelos y su eficiencia están fuertemente relacionados a la arquitectura paralela subyacente y al modelo de programación aplicado. El cálculo en paralelo consiste en usar múltiples recursos simultáneamente para resolver un problema dado. Hace uso de una computadora con varias CPU's, el problema es dividido en partes relativamente independientes, cada parte es dividida en un conjunto de instrucciones, las instrucciones son ejecutadas secuencialmente y las partes son resueltas simultáneamente.

Cada vez mas, el procesamiento paralelo esta siendo visto como el único método con una buena relación costo/beneficio para una rápida solución de problemas con gran carga de datos y computacionalmente complejos. Para obtener eficiencia, un programa paralelo debe ser adecuado a las características de la máquina paralela real. La aparición de computadoras paralelas de bajo costo, tales como computadoras de escritorio multiprocesador y clusters de estaciones de trabajo o computadoras personales, ha hecho que estos métodos de paralelismo sean de aplicación general.

5.2. Conceptos Generales de Sistemas Paralelos

La mayor parte de los sistemas hasta ahora han sido sistemas monoprocesador; es decir han tenido una sola CPU. Sin embargo, la tendencia actual es hacia los sistemas multiprocesador. Tales sistemas tienen más de un elemento de procesamiento los cuales comparten el bus del

computador, el reloj y a veces la memoria y los dispositivos periféricos. Estos sistemas están fuertemente acoplados y hay varias razones para construir este tipo de sistemas. Una de sus ventajas es el aumento en el rendimiento, al incrementar el número de procesadores, se espera realizar más trabajo en un tiempo más corto. Sin embargo, la proporción de aumento de la velocidad con n procesadores generalmente no es n, sino menor. Cuando varios procesadores cooperan para llevar a cabo una tarea, se incurre en cierto overhead para mantener todos los componentes funcionando correctamente. Este costo extra, reduce la ganancia que cabría esperar de los procesadores adicionales. De forma análoga, un grupo de n programadores que trabajan en colaboración no produce n veces la cantidad de trabajo que realiza uno sólo.

Los multiprocesadores también pueden ahorrar dinero en comparación con varios sistemas monoprocesador porque los procesadores pueden compartir periféricos, gabinetes y fuentes de potencia. Si varios programas deben operar con el mismo conjunto de datos, es más económico guardar esos datos en un disco y hacer que todos los procesadores los compartan, en vez de tener muchos computadores con discos locales y hacer copias de los datos.

Otra razón para tener sistemas multiprocesador es que mejoran la confiabilidad. Si es posible distribuir las funciones correctamente entre varios procesadores, el fallo de un procesador no detendrá el sistema, sólo lo hará más lento. Si se tienen 10 procesadores y uno falla, cada uno de los nueve procesadores restantes deberá asumir una porción del trabajo del que falló. Así el sistema operará al 90

5.3. Arquitecturas

El hecho de que los principales proveedores de microprocesadores han cambiado la estrategia en el diseño de tecnología con mayor rendimiento, presenta nuevas oportunidades para los desarrolladores de software. Las plataformas precedentes se basan en un modelo de programación secuencial. Los sistemas operativos y otras aplicaciones simulan ambientes multitarea tomando ventaja de la velocidad del procesador para poder atender varias tareas, asignando determinado tiempo de procesador a cada una de ellas. Como resultado la tecnología multithreads dio buen resultado para poder aprovechar los tiempos de espera del procesador.

Un cluster consiste en un tipo de arquitectura de procesamiento paralelo distribuido, compuesto por un conjunto de computadoras que trabajan cooperativamente como un único e integrado recurso de cómputo. En general, se integran con componentes de bajo costo y se conectan por redes de área local. La figura 5.1 muestra la arquitectura de un clúster.

La agregación permite alcanzar un poder de cómputo muy superior a las de los supercomputadores específicos. Los clústeres son de alto rendimiento, con escalabilidad incremental y balance de carga.

Los componentes de un clúster pueden incluir:

- Nodos: son los procesadores. Pueden ejecutar en diferentes sistemas operativos y/o tener diferentes características (clusters heterogéneos).
- Software de base: el sistema operativo
- Comunicaciones: redes de alta velocidad, interface y software para comunicaciones

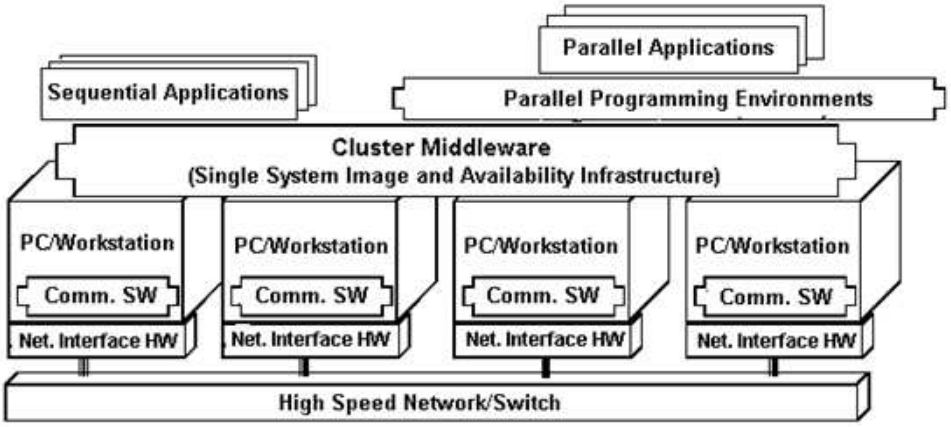

Figura 5.1: Arquitectura de un cluster

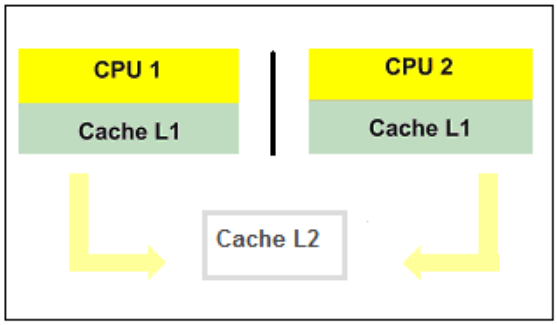

Figura 5.2: Arquitectura multicore

- Middleware: gestores de ejecución, monitores de recursos, etc.
- Entornos de programación
- Bibliotecas y herramientas de desarrollo

Desde los orígenes del primer procesador se ha buscado incrementar la capacidad de cómputo y el rendimiento de los mismos aumentando la potencia de cómputo y al mismo tiempo disminuyendo el tamaño de los circuitos integrados, pero ese aumento tiene un límite.

La arquitectura multicore consiste en un conjunto de núcleos de procesamiento integrados en un único chip. Cada procesador contiene varios núcleos que pueden ejecutar instrucciones de forma simultánea. Normalmente, cada uno de los núcleos (core) posee su propio nivel L1 de cache y de a pares comparten el nivel L2 de cache (incluido generalmente en la placa madre) como lo muestra la figura 5.2.

Con la nueva arquitectura multinúcleo se cuenta ahora con plataformas realmente paralelas en un mismo procesador, de tal manera que los desarrolladores de software cuentan con mayores capacidades y mayor versatilidad en el desarrollo y diseño de aplicaciones. De igual forma este cambio tecnológico presenta nuevos retos en el desarrollo de aplicaciones, ya que se deben tomar en cuenta ciertas consideraciones y afrontar nuevos problemas cuando se pretenden desarrollar aplicaciones paralelas.

Las aplicaciones que sacan más provecho de estos procesadores multinúcleo son aquellas que pueden generar muchos hilos de ejecución como las aplicaciones de audio/video, cálculo científico, juegos, tratamiento de gráficos en 3D, entre otras. Pero de todas maneras siempre hay aplicaciones que no se pueden dividir en hilos de ejecución, por lo que no aprovechan por completo estos procesadores. En esos casos, pueden ejecutar varias de estas aplicaciones al mismo tiempo.

Además esos procesadores multinúcleo pueden conectarse entre sí por medio de una red de interconexión para lograr una arquitectura paralela más potente. Con esto se forma una arquitectuta que combina memoria compartida y distribuida llamada cluster de mulcticores

5.4. Modelos

Existen dos formas primarias para el intercambio de datos entre tareas paralelas, ya sea accediendo a un espacio de datos compartidos o bien intercambiando mensajes.

5.4.1. Modelo de espacio de direcciones compartido

El termino espacio de direcciones compartido de una plataforma paralela es históricamente usado para arquitecturas en las cuales existe un espacio de datos común, accesible para todos los procesadores. Los procesadores interactúan modificando los objetos de datos almacenados en ese espacio común. La memoria en plataformas de espacio de direcciones compartida puede ser local (exclusiva para un procesador) o global (común a todos los procesadores). Si el tiempo tomado por un procesador para acceder a alguna palabra de memoria en el sistema (global o local) es idéntico, la plataforma es clasificada como un multicomputador de acceso de memoria uniforme (UMA). Por otro lado, si el tiempo tomado para acceder a ciertas palabras de memoria es mas largo que para otras, la plataforma es llamada un multicomputador de acceso de memoria no uniforme (NUMA). Las figuras 5.3.(a) y 5.3.(b) ilustran plataformas UMA, mientras que la figura 5.3.(c) ilustra una plataforma NUMA. Un caso interesante es ilustrado en figura 5.3.b). Aquí, es mas rápido acceder a una palabra de memoria en cache que en una posición de memoria determinada. Sin embargo, se clasifica como una arquitectura UMA. La razón de esto es que todos los microprocesadores actuales tienen jerarquías de cache. En consecuencia, un monoprocesador no podría ser considerado como UMA, si se tiene en cuenta los tiempos de acceso a cache. Por esta razón, se definen las arquitecturas NUMA y UMA en términos de tiempos de acceso a memoria y no a tiempos de acceso a cache.

La presencia de un espacio de direcciones de memoria global hace que la programación sea más fácil. Todas las interacciones de solo lectura son invisibles al programador, ya que son codificadas de la misma forma que un programa serie. Esto facilita enormemente la carga de la escritura de programas paralelos. Las interacciones de lectura/escritura son más difíciles de programar que las de solo lectura, dado que requieren exclusión mutua para los accesos concurrentes. Los paradigmas de programación de memoria compartida tales como hilos (POSIX) y directivas (OpenMP) soportan sincronización usando cerramientos (locks) y otros mecanismos relacionados [3].

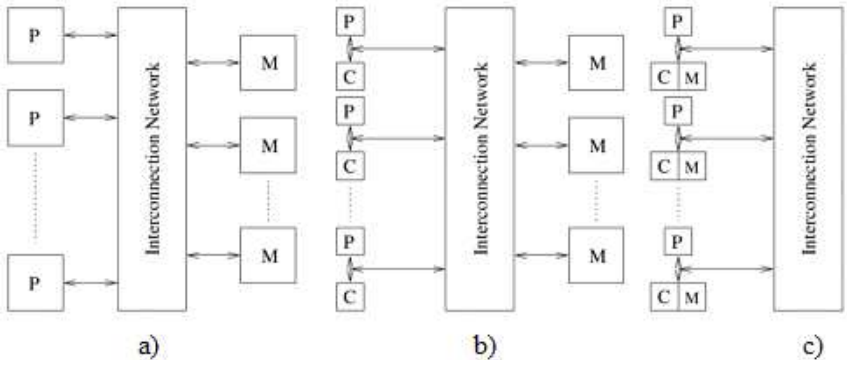

Figura 5.3: (a) Computadora de acceso del espacio de direcciones compartido y acceso de memoria uniforme. –(b) Computadora de espacio de direcciones compartido y acceso a memoria, con caché y la memoria. –(c) Computadora de espacio de direcciones compartido y acceso a memoria no uniforme

5.4.2. Modelo de pasaje de mensajes

La vista lógica de una maquina de una plataforma de pasaje de mensajes consiste de p nodos de procesamiento, cada uno con su propio espacio de direcciones exclusivo. Estos pueden ser procesadores simples o multiprocesadores con memoria compartida.

En estas plataformas, las interacciones entre los procesos corriendo sobre diferentes nodos deben ser realizadas a través de mensajes, de ahí el nombre. Ese intercambio de mensajes es usado para transferir datos, trabajo y sincronizar acciones entre procesos. En su forma más general, los paradigmas de pasaje de mensajes soportan la ejecución de un programa diferente en cada uno de los p nodos.

Dado que las interacciones son realizadas a través del envío y recepción de mensajes, las operaciones básicas de este paradigma son enviar y recibir (las llamadas correspondientes pueden variar dependiendo de las APIs pero la semántica es prácticamente idéntica). Además, dado que los procesos deben conocer con quien se establece la comunicación, tiene que haber un mecanismo para identificar únicamente a cada uno de los múltiples procesos de ejecución de un programa paralelo. Existe otra función que suele ser necesitada para completar el conjunto básico de operaciones de pasaje de mensajes para conocer el número de procesos que participan en el sistema paralelo. Con esas cuatro operaciones básicas, es posible escribir cualquier programa de pasaje de mensajes. Diferentes APIs de pasaje de mensajes, como son MPI (Interfaz de Pasaje de Mensajes) y PVM (Maquina Virtual Paralela), soportan esas operaciones básicas y una variedad de funcionalidades de más alto nivel bajo diferentes nombres de funciones.

Es fácil simular una arquitectura de pasaje de mensajes conteniendo p nodos sobre una computadora con memoria compartida con un número idéntico de nodos. Asumiendo nodos monoprocesador, esto puede ser realizado particionando el espacio de direcciones compartido dentro de p partes disjuntas y asignando una partición exclusivamente a cada procesador.

Un procesador puede entonces enviar o recibir mensajes, escribiendo o leyendo de otra partición del procesador mientras usa las primitivas de sincronización adecuadas para informar a su compañero de comunicación cuando se haya terminado de leer o escribir los datos. Sin embargo, la simulación de arquitectura de espacio de direcciones compartido sobre una computadora de

pasaje de mensajes es costosa, ya que para acceder a la memoria de otro nodo es necesario enviar y recibir mensajes [3].

5.5. Paradigmas

Existen tres patrones básicos de interacción de procesos, ellos son: productor/consumidor, cliente/servidor y pares interactivos.
Estos tres patrones pueden ser combinados de diferentes formas. Esta sección describe muchos de estos patrones de interacción de procesos e ilustra su uso. Cada patrón es un paradigma (modelo) para interacción de procesos. En particular, cada paradigma tiene una estructura determinada que puede ser usada para resolver diferentes tipos de problemas [4]. Los paradigmas que se van a describir son:

- *Manager/Workers*, el cual representa una implementación distribuida de una bolsa de tareas (*Bag of Task*);
- *Probe/Echo*, los cuales difunden y reúnen la información en árboles y grafos;
- Algoritmos *Broadcast*, los cuales son usados para descentralizar la toma de decisiones;
- Algoritmos *Pipeline*, en los cuales la información fluye desde un proceso hacia otro, usando una interacción de envío y luego recepción;
- Servidor Replicados, los cuales manejan múltiples instancias de recursos como por ejemplo archivos;
- Algoritmos *HeartBeat*, en los cuales los procesos periódicamente intercambian información usando una interacción de envío y luego de recepción;

5.5.1. Manager/Workers

La idea es que varios procesos trabajadores comparten una bolsa que contiene tareas independientes. Cada trabajador repetidamente remueve una tarea de la bolsa, la ejecuta, y posiblemente genera una o más tareas nuevas que pone dentro de la bolsa. Los beneficios de este enfoque son la facilidad de cambiar el número de trabajadores y asegurar que cada uno realice la misma cantidad de trabajo (Balance de Carga).

Bag of tasks

En este paradigma cada tarea es una unidad independiente de trabajo. Las tareas son ubicadas en un bag que es compartido por 2 o mas procesos trabajadores. Cada trabajador ejecuta el código de la figura 5.4.

Este enfoque puede ser usado para implementar paralelismo recursivo, en cuyo caso las tareas son las llamadas recursivas. También puede ser usado para resolver problemas iterativos teniendo un número fijo de tareas independientes.

```
while siempre do
  tomar una tarea de la bolsa
  if (no hay mas tareas) then
    break; # sale del while
  end if
  ejecuta la tarea, posiblemente generando nuevas tareas;
end while
```

Figura 5.4: Código básico compartido por los procesos trabajadores

Este paradigma tiene muchas ventajas. Primero, es bastante fácil su uso, sólo se tiene que definir la representación para una tarea, implementar el *"bag"*, programar el código para ejecutar una tarea, y descifrar como determinar la finalización.

Segundo, los programas que usan este paradigma son escalables, lo que significa que uno puede usar cualquier número de procesadores simplemente variando el número de *workers*. Finalmente, se logra un buen balance de carga.

Si las tareas toman diferentes cantidades de tiempo para ejecutar, algunos workers ejecutarán probablemente más tareas que otros, pero el tiempo total de cada uno será parejo.

5.5.2. Algoritmos de Probe/Echo

Los árboles y los grafos se utilizan en muchas aplicaciones, tales como búsquedas en la Web, bases de datos, juegos y sistemas expertos. Son especialmente importantes en cómputo distribuido, ya que muchos de ellos se estructuran en forma de grafo, donde los procesos son los nodos y los canales de comunicación son los arcos.

Un *probe* es un mensaje enviado por un nodo a su sucesor; un *echo* es la respuesta subsecuente. Puesto que los procesos se ejecutan concurrentemente, las comunicaciones se realizan en paralelo a todos los sucesores.

5.5.3. Algoritmos Broadcast

En este caso cada proceso esta directamente conectado a cada uno de los restantes. De hecho, estas redes de comunicación frecuentemente proveen una primitiva de comunicación llamada *broadcast*, la cual trasmite un mensaje desde un procesador a todos los demás. Ya sea que se cuente con un hardware de comunicación o no, la transmisión de mensajes ofrece una técnica de programación útil.
Se puede usar para difundir información, por ejemplo, para intercambiar el estado del procesador en redes de área local. También es posible usarlo para resolver problemas de sincronización.

5.5.4. Productores y consumidores (pipelines)

Un proceso filtro recibe datos de un puerto de entrada, los procesa, y envía los resultados a un puerto de salida. Una *pipeline* es una colección lineal de procesos filtros.

Cualquier número de procesos workers pueden utilizarse para resolver un problema de computación paralela. Existen distintas estructuras pipeline según lo que se necesite resolver.

5.5.5. Servidores replicados

Un servidor, es un proceso que maneja algún tipo de recurso. Este podría ser replicado cuando existen múltiples instancias diferentes de un recurso; cada servidor podría entonces manejar una de esas instancias. La replicación puede ser usada para brindar a los clientes la ilusión que existe un único recurso, cuando en realidad hay muchos.

5.5.6. Algoritmos Heartbeat

Así como el paradigma *Bag of Tasks* es útil para un número fijo de tareas, el paradigma *heartbeat* es útil para aplicaciones iterativas paralelas que manejen muchos datos. En particular, puede ser usado cuando el dato es dividido a lo largo de los procesos, donde cada uno es responsable de actualizar una parte determinada, y los nuevos valores de los datos dependen de los valores en poder de los vecinos inmediatos.

5.6. Herramientas

Tradicionalmente, la programación paralela se ha venido utilizando para la resolución de problemas con alto costo computacional [13] que estaban en la frontera de los problemas científicos abordables por sistemas computacionales. Esta programación era realizada por personal especializado (informáticos o científicos), que lo aprendía para resolver los problemas en los que trabajaban. Para la resolución eficiente de estos problemas es necesaria la utilización de entornos paralelos y también la optimización o el rediseño de los algoritmos implementados. En cuanto a entornos de programación, se han generado herramientas que facilitan la programación en los distintos sistemas como Posix threads (Pthreads) [30] y OpenMP para procesamiento multithreads en entornos de memoria compartida, MPI para entornos de pasaje de mensajes [41], UPC para memoria virtual compartida [14] o CUDA para procesadores básicos [31].

5.6.1. POSIX

Una ventaja de la mayoría de las implementaciones de UNIX y DCE consiste en que conforman un estándar recientemente ratificado POSIX que permite a sus programas ser portables entre ellos. El estándar POSIX, conocido comúnmente como Pthreads, es soportado por la mayoría de los sistemas operativos basados en UNIX. Un servidor de red tiene que aceptar llamadas simultáneas de muchos clientes, proveyendo a cada uno un tiempo de respuesta razonable. Un multiprocesador controla un número de programas que corren sobre varias CPUs inmediatamente, combinando los resultados cuando todos son hechos.
Con la programación de *threads*, múltiples tareas pueden correr concurrentemente en el mismo programa. Ellos pueden compartir una sola CPU o aprovechan múltiples de ellas cuando están disponibles. Proporcionan un modo de dividir las tareas de un programa mientras comparten datos. Los *threads* son vistos cada vez más a menudo como un nuevo modelo de programación.

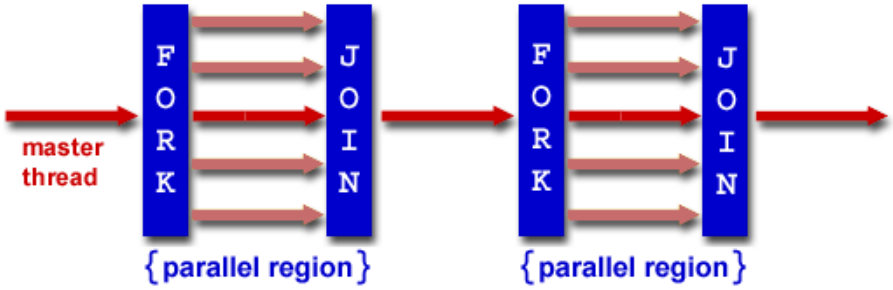

Figura 5.5: Modelo de programación de OpenMP

5.6.2. OpenMP

OpenMP es una interfaz de programación de aplicaciones (API) para la programación multiproceso de memoria compartida en múltiples plataformas. Permite añadir concurrencia a los programas escritos en C, C++ y Fortran sobre la base del modelo de ejecución fork-join como se muestra en la figura 5.5. Se compone de un conjunto de directivas de compilador, rutinas de librería y variables de entorno que influencian el comportamiento en tiempo de ejecución. Definido conjuntamente por un grupo de proveedores de hardware y de software, OpenMP es un modelo de programación portable y escalable que proporciona a los programadores una interfaz simple y flexible para el desarrollo de aplicaciones paralelas para las plataformas que van desde las computadoras de escritorio hasta las supercomputadoras. Una aplicación construida con un modelo de programación paralela híbrido se puede ejecutar en un clúster de computadoras utilizando OpenMP y MPI, o más transparentemente a través de las extensiones de OpenMP para los sistemas de memoria distribuida. La mayoría de las construcciones en OpenMP son directivas de compilación o pragmas. Como las construcciones son directivas, un programa en OpenMP puede ser compilado por compiladores que no soportan OpenMP. La parte central de OpenMP es la paralelización de lazos. Los threads se comunican utilizando variables compartidas. El uso inadecuado de variables compartidas origina barreras críticas, para esto se utiliza la sincronización para protegerse de los conflictos de datos. OpenMP se basa en la existencia de múltiples threads en el paradigma de memoria compartida [5].

Todos los programas OpenMP comienzan como un único proceso: el hilo principal. El hilo principal se ejecuta de forma secuencial hasta que encuentra la primera región con un bloque paralelo. En ese punto el hilo principal crea un grupo de hilos paralelos (*Fork*). Las sentencias en el programa que están encerradas en el constructor de la región, son ejecutadas en paralelo entre los hilos del grupo. Cuando todos los hilos completan las sentencias de la región en paralelo, se sincronizan y terminan, quedando solo el hilo principal (*Join*). Todo hilo tiene un contexto de ejecución que consiste del espacio de direcciones que el *thread* puede accesar. El contexto de ejecución incluye variables estáticas, estructuras dinámicamente asignadas, y variables en el *"run-time stack"*.

No se especifica nada acerca de la entrada/salida paralela. Esto es particularmente importante si varios hilos intentan escribir / leer desde el mismo archivo. Si todos los hilos llevan a cabo la entrada/salida en un archivo diferente, los problemas no son tan significativos. Es responsabilidad del programador asegurar que la entrada/salida se lleve a cabo correctamente en un programa multithreads.

Las iteraciones de un bucle pueden variar mucho en duración. Se puede usar una cláusula schedule para indicar como las iteraciones se asignan a los hilos. En el scheduler estático todas las iteraciones se asignan a los hijos antes de ejecutarla. En el scheduler dinámico las iteraciones se asignan a medida que los hilos están libres (sin trabajo). En el scheduler guided cthread toma iteraciones dinámicamente y progresivamente va tomando menos iteraciones. El scheduler runtime se escoge en tiempo de ejecución basado en el valor de la variable omp-shedule.

5.6.3. MPI

MPI (*"Message Passing Interface"*, Interfaz para Pasaje de Mensajes) es un estándar que define la sintaxis y la semántica de las funciones contenidas en una biblioteca de pasaje de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores. Su principal característica es que no precisa de memoria compartida, por lo que es muy importante en la programación de sistemas distribuidos. Los elementos principales que intervienen en el pasaje de mensajes son el proceso que envía, el que recibe y el mensaje. Dependiendo de si el proceso que envía el mensaje espera a que el mensaje sea recibido, se puede hablar de paso de mensajes síncrono o asíncrono. En el paso de mensajes asíncrono, el proceso que envía, no espera a que el mensaje sea recibido, y continúa su ejecución, siendo posible que vuelva a generar un nuevo mensaje y a enviarlo antes de que se haya recibido el anterior. Por este motivo se suelen emplear buffers, en los que se almacenan los mensajes a espera de que un proceso los reciba. Generalmente empleando este sistema, el proceso que envía mensajes sólo se bloquea o para, cuando finaliza su ejecución, o si el buffer está lleno. En el paso de mensajes síncrono, el proceso que envía el mensaje espera a que un proceso lo reciba para continuar su ejecución. MPI es un protocolo de comunicación entre computadoras, el cual define el estándar para la comunicación entre los nodos que ejecutan un programa en un sistema de memoria distribuida. Las implementaciones en MPI consisten en un conjunto de bibliotecas de rutinas que pueden ser utilizadas en programas escritos en los lenguajes de programación C, C++, Fortran y Ada. La ventaja de MPI sobre otras bibliotecas de paso de mensajes, es que los programas que utilizan la biblioteca son portables dado que MPI ha sido implementado para casi toda arquitectura de memoria distribuida, y rápidos, porque cada implementación de la biblioteca ha sido optimizada para el hardware en la cual se ejecuta.

5.6.4. UPC

Unified Parallel C (UPC) es una extensión del lenguaje de programación C diseñado para computación de alto rendimiento en máquinas paralelas, incluyendo tanto aquellas con un espacio de memoria global (SMP y NUMA) como aquellas con un espacio de memoria distribuido, como los clusters. El programador observa un único espacio de memoria compartida donde las variables se pueden leer y escribir desde cualquier procesador, pero cada una reside físicamente en un único procesador, con el que se dice que tiene afinidad. UPC usa un modelo de computación SPMD (*Single Program Múltiple Data*) donde la cantidad de procesos se puede indicar tanto en tiempo de compilación como al inicio de la ejecución del programa.

5.6.5. CUDA

CUDA (*Compute Unified Device Architecture*) intenta aprovechar el gran paralelismo, y el alto ancho de banda de la memoria en las GPUs en aplicaciones con un gran costo de cómputo frente a realizar numerosos accesos a memoria principal, lo que podría actuar de cuello de botella. El modelo de programación de CUDA está diseñado para que se creen aplicaciones que de forma transparente escalen su paralelismo para poder incrementar el número de núcleos computacionales. Este diseño contiene tres puntos claves, que son la jerarquía de grupos de hilos, las memorias compartidas y las barreras de sincronización. Un multiprocesador contiene ocho procesadores escalares, dos unidades especiales para funciones trascendentales, una unidad multithread de instrucciones y una memoria compartida. El multiprocesador crea y maneja los hilos sin ningún tipo de overhead por la planificación, lo cual unido a una rápida sincronización por barreras y una creación de hilos muy ligera, consigue que se pueda utilizar CUDA en problemas de muy baja granularidad, incluso asignando un hilo para un elemento por ejemplo de una imagen (un pixel).

5.7. Métricas de performance

Existen algunos factores intuitivos que pueden ser utilizados para evaluar la performance: el tiempo de ejecución que es una medida tradicionalmente utilizada para evaluar la eficiencia computacional de un programa y la utilización de los recursos disponibles.

5.7.1. Speedup

El speedup mide la mejora de rendimiento de una aplicación al aumentar la cantidad de procesadores. Se define el *speedup absoluto* como $S_N = T_0/T_N$, siendo T_0 el tiempo del algoritmo serial más rápido que resuelve el problema y T_N el tiempo del algoritmo paralelo ejecutado sobre N procesadores.

La situación ideal es lograr un speedup lineal, es decir al utilizar N procesadores obtener una mejora de factor N. No siempre es posible alcanzar el speedup ideal, lo más habitual es obtener speedup sublineal. En casos muy específicos es posible obtener valores de speedup superlineal.

5.7.2. Eficiencia

Mide la fracción de tiempo promedio durante el cual un elemento de procesamiento es empleado, se define como la proporción de speedup con el número de elementos de procesamiento. En un sistema paralelo ideal, el speedup es igual a N y la eficiencia es igual a uno. En la práctica, el speedup es menor que N y la eficiencia se encuentra entre cero y uno, dependiendo de la eficiencia con la que se utilizan los elementos de procesamiento [3].

Se define eficiencia como: $E = T/N * T_N$. Es decir $E = S_N/N$ correspondiendo a un valor normalizado de speedup respecto a la cantidad de procesadores utilizados. Valores de eficiencia cercanos a uno identifican situaciones casi ideales de mejora del desempeño.

5.7.3. Escalabilidad

Generalmente, los programas están diseñados y probados para pequeños problemas y con poca cantidad de procesadores. Sin embargo, los problemas reales que estos programas están destinados a resolver son mucho más grandes, y las máquinas contienen un mayor número de procesadores. Considerando que el desarrollo del código se ha simplificado mediante el uso de versiones a escala reducida de la máquina y el problema, su rendimiento y corrección (de programas) es mucho más difícil de establecer sobre la base de los sistemas de escala reducida.

Un sistema paralelo escalable se define como aquel en que la eficiencia puede mantenerse constante cuando el número de procesadores es mayor, siempre que el tamaño del problema sea también mayor.

Para diferentes sistemas paralelos, el tamaño del problema debe aumentar a un ritmo diferente, a fin de mantener un rendimiento igual al incrementar el número de procesadores [3].

Es la capacidad de agregar procesadores para obtener mejor rendimiento en la ejecución de aplicaciones paralelas. Constituye una de las principales características deseables de los algoritmos paralelos y distribuidos. Esta métrica permite determinar la habilidad para que el sistema se siga comportando correctamente a pesar de que diferentes aspectos del sistema crezcan en tamaño u otras características (aumento de capacidad de cómputo, de capacidad de red, de número de clientes del sistema, etc).

5.7.4. Balance de carga

El objetivo primario del paralelismo es reducir el tiempo de ejecución y hacer uso eficiente de los recursos. La manera de asignar o mapear procesos lógicos a procesadores físicos es fundamental ya que el uso desigual (desbalance) de los procesadores puede degradar fuertemente la eficiencia del procesamiento paralelo. La descomposición del problema debe ser equilibrada, todos los procesos deben estar trabajando siempre, es fundamental para la eficiencia paralela.

Las maneras de equilibrar la carga son:

- Distribución equitativa del trabajo entre los procesos:
  - Realizar la descomposición de dominios en partes iguales.
  - Dividir los bucles en tramos iguales.
  - En computadoras heterogéneas, realizar medidas de velocidad para decidir la descomposición del problema.
- Asignación dinámica del trabajo:
  - La distribución equitativa no siempre resuelve el problema del balance de carga.
  - El trabajo es asignado a los procesos a medida que terminan las asignaciones anteriores.

5.8. Conclusiones

La computación paralela es el uso simultáneo de múltiples recursos de cómputo para resolver un problema computacional. Para ser ejecutado mediante múltiples CPUs un problema se divide en partes discretas que pueden ser resueltas al mismo tiempo, donde cada parte se subdivide en una serie de instrucciones.
Por lo tanto, la computación paralela es una evolución de la informática en serie que intenta simular lo que ha sido siempre el estado de cosas en el mundo real (muchos eventos interrelacionados sucediendo al mismo tiempo).
Los algoritmos paralelos son importantes porque es más rápido tratar grandes tareas de cómputo mediante la paralelización que mediante técnicas secuenciales. Esta es la forma en que se trabaja en el desarrollo de los procesadores modernos, ya que es más difícil incrementar la capacidad de procesamiento con un único procesador que aumentar su capacidad de cómputo mediante la inclusión de unidades en paralelo, logrando así la ejecución de varios flujos de instrucciones dentro del procesador.

Los algoritmos paralelos también necesitan optimizar la comunicación entre diferentes unidades de procesamiento. Esto se consigue mediante la aplicación de dos paradigmas de programación y diseño de procesadores distintos: memoria compartida o paso de mensajes.
La técnica memoria compartida necesita del uso de bloqueos en los datos para impedir que se modifique simultáneamente por dos procesadores, por lo que se produce un costo extra en ciclos de CPU desperdiciados y ciclos de bus. También obliga a serializar alguna parte del algoritmo.
La técnica paso de mensajes usa canales y mensajes pero esta comunicación añade un costo al bus, memoria adicional para las colas y los mensajes y latencia en el mensaje.

La computación paralela se ha convertido en el paradigma dominante en la arquitectura de computadores, principalmente en los procesadores multicores. Estos pueden ofrecer importantes beneficios de rendimiento para el software multithreads mediante la adición de capacidad de procesamiento con una latencia mínima, dada la proximidad de los procesadores.

Utilizando los conceptos mencionados recientemente, en el siguiente capítulo se propone una implementación paralela del algoritmo SIFT el cual aprovecha las ventajas de los procesadores multicores y permite llegar a un rendimiento cercano al óptimo.

Capítulo 6

Solución Paralela

6.1. Introducción

Como se mencionó en el capítulo 4, el método SIFT es, sin duda, uno de los más utilizados. Sin embargo, su costo computacional es alto si se piensa en su aplicación en tareas tales como tracking de video.

Este capítulo propone una implementación paralela del método de SIFT sobre una arquitectura paralela de memoria compartida, aprovechando el amplio desarrollo de los multicores. Este tipo de máquina permite obtener menores tiempos de respuesta para una aplicación al dividir el trabajo entre los cores que lo forman. Para lograr tiempos de respuesta acordes a los requeridos para el procesamiento de videos en tiempo real se utiliza un esquema Bag of Task para lograr de esta manera un balance de carga que asegure un buen rendimiento de la aplicación.

6.2. Formas de Paralelización

Se plantearon diferentes formas de paralizar el algoritmo con el objetivo de lograr mejores tiempos de respuesta. A continuación se detallan las tres alternativas consideradas.

6.2.1. Particionamiento de Frames

La primer alternativa considerada fue paralelizar únicamente la etapa referida a la construcción de la pirámide Gaussiana.

Para ello, cada frame a procesar se particiona en bloques y estos se asignan a diferentes threads. A modo de ejemplo, en la figura 6.1 *d1, d2, d3* y *d4* son las particiones considerando un sistema con 4 threads. El proceso comienza con la imagen que se encuentra en la base de la pirámide y unicamente cuando todos los threads hayan terminado con su trabajo es posible generar la siguiente, dado que cada imagen depende enteramente de su predecesora. El proceso se repite tantas veces como octavas tenga la imagen en cuestión.

Una vez construida la pirámide Gaussiana, se continua con el método de detección de puntos

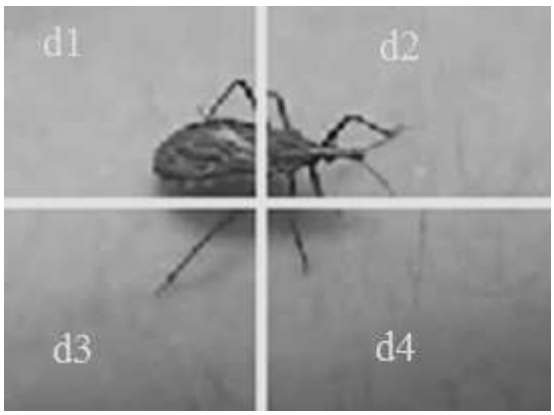

Figura 6.1: Particionamiento de frames

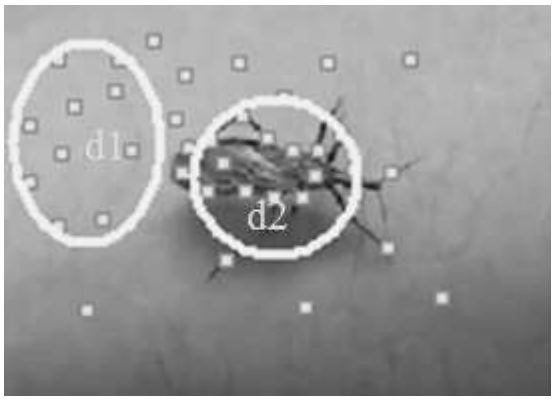

Figura 6.2: División por puntos claves

claves y la posterior generación de los descriptores correspondientes en forma secuencial.

Uno de los inconvenientes que presenta este método es la necesidad de sincronización entre los distintos threads dada la dependencia entre frames existente.

6.2.2. División por Puntos Claves

En este caso se realiza de manera secuencial la detección de puntos claves, esto implica, la construcción de las pirámides Gaussianas y Laplacianas y la posterior detección de extremos. Una vez detectados los puntos, debido a que el procesamiento requerido es idéntico para todos ellos, se distribuyen equitativamente y en forma estática a los distintos threads. En el ejemplo de la figura 6.2 se observa este tipo de distribución donde *d1* y *d2* son subconjuntos de puntos claves asignados a diferentes procesos. Se generan en forma paralela cada uno de los descriptores pertenecientes a los puntos claves. Al finalizar se comparan los vectores para encontrar las correspondencias en las imágenes.

La deficiencia de este método radica en que la mayor parte del procesamiento que se realiza en cada frame es secuencial; lo cual no permite aprovechar eficientemente la arquitectura paralela sobre la que se realiza el trabajo.

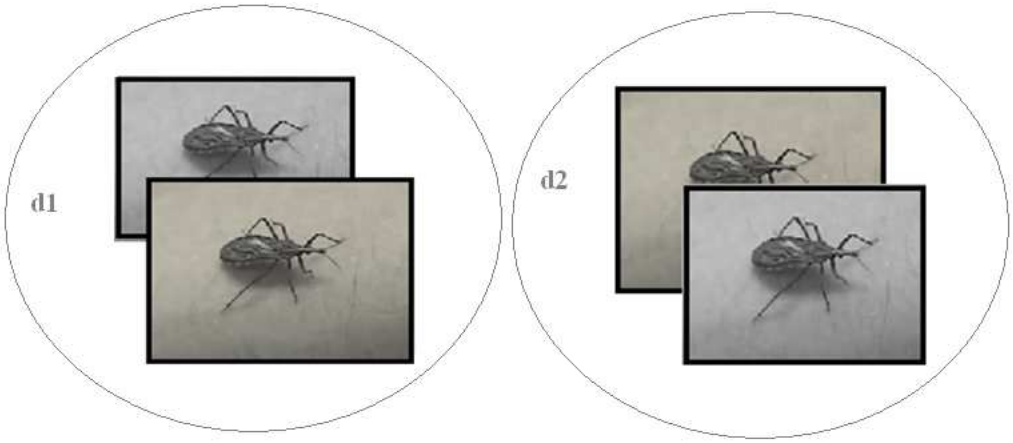

Figura 6.3: Distribución por frames

6.2.3. Distribución de Frames

Esta tercer alternativa busca aprovechar la independencia que existe entre los distintos frames que componen el video. De esta manera, se calcula secuencialmente el vector de características de la imagen que se desea buscar en el video por medio de la función SIFT. Luego, se distribuyen las imágenes que forman el video entre los procesos generados. Es decir que, cada proceso resuelve conjuntos de frames.

Se generan concurrentemente los vectores de características de los frames del video, comparando cada vector con los de la imagen a buscar y obteniendo las correspondencias existentes. En la figura 6.3 se muestra un ejemplo de este tipo de distribución, donde *d1* y *d2* son conjuntos de frames asignados a diferentes procesos.

Dado que el procesamiento que se realiza en cada frame varía de acuerdo a los datos de la imagen y no del tamaño, los frames se distribuyen entre los threads a medida que estos demandan trabajo.

6.2.4. Justificación de la técnica elegida

De acuerdo a lo visto en la sección 6.2.1, 6.2.2 y 6.2.3 se puede concluir:

- En la técnica de particionamiento de frames, descripta en la sección 6.2.1, el método genera un overhead importante por la necesidad de sincronización.
- En la técnica de división por puntos claves, detallada en la sección 6.2.2, no existe el problema mencionado en el item anterior, pero existe un overhead muy importante dado que el mayor procesamiento es secuencial.
- En la técnica mencionada en la sección 6.2.3 se observa que no existen problemas de sincronización entre los threads, dado que cada uno de los frames es independiente del resto. Tampoco se observa un uso ineficiente de la arquitectura paralela originada por el exceso de código secuencial dado que la única parte que se lleva a cabo de esa manera es el cálculo de los descriptores para la imagen a buscar.

Se eligió la forma de distribución por frames ya que presenta ventajas sobre las dos anteriores. Además, a diferencia de la alternativa descripta en la sección 6.2.1, no necesita realizar trabajo extra para la comunicación ya que los frames son independientes. Por lo tanto, la única parte secuencial es el procesamiento de la imagen a buscar, el resto se realiza en paralelo.

6.3. Experimentación

El desempeño de la alternativa de paralelización propuesta fue comparada contra la solución secuencial. A continuación se describen los algoritmos medidos y las pruebas realizadas.

6.3.1. Algoritmo secuencial

Como ya se ha mencionada anteriormente, la extracción de características utilizando los descriptores SIFT es un proceso costoso en tiempo al aplicarlo para reconocer objetos en los diferentes frames de un video. Esto dificulta su aplicación en procesos que requieren obtener respuesta en tiempo real.

Los siguientes cuatro pasos resumen brevemente el funcionamiento del proceso secuencial para determinar las características SIFT:

- En primer lugar se calcula la ubicación de potenciales puntos de interés dentro de la imagen, los que corresponden a los puntos extremos calculados a partir de subconjuntos de planos de diferencias de filtros gaussianos (DoG) aplicados sobre la imagen a distintas escalas.
- Luego, se descartan los puntos de interés que poseen bajo contraste.
- En tercer lugar, se calculan las orientaciones de los puntos de interés relevantes.
- Utilizando las orientaciones anteriores, se analiza el entorno de cada punto y se determina el vector de características correspondientes.

Como resultado de este proceso se obtiene un conjunto de vectores de características de longitud 128 que pueden ser comparados con los correspondientes a otra imagen del mismo objeto con diferente escala, orientación y/o punto de vista. Dicha comparación puede realizarse directamente a través de una medida de distancia estableciendo un umbral de similitud [27].

La necesidad de reducir el tiempo de cómputo se puede resolver incrementando la potencia de cómputo de las máquinas. Se ha llegado a un punto en el cual escalar la velocidad de los procesadores convencionales (un núcleo), trae aparejado problemas de consumo de energía y generación de calor [19].

Esta limitación ha llevado al desarrollo y utilización masiva de arquitecturas paralelas de memoria compartida, como lo son los actuales multicores. Un procesador multicore surge a partir de integrar dos o más núcleos computacionales dentro de un mismo “chip”. Este tipo de máquina permite incrementar el rendimiento de una aplicación al dividir el trabajo de cómputo entre los núcleos disponibles [39].

Dada la necesidad de lograr tiempos de respuesta acordes a los requeridos para el procesamiento de videos en tiempo real, sumado a la disponibilidad de estas arquitecturas paralelas, se desarrolla una versión paralela del algoritmo.

6.3.2. Algoritmo paralelo

Para este tipo de aplicación en la que se debe realizar un procesamiento costoso sobre un conjunto de frames de forma independiente, resulta claro que la forma más simple y eficiente de paralelizar es distribuir las imágenes que forman el video entre los threads (o hilos) generados.

Un punto a tener en cuenta es la forma en que se distribuye el trabajo entre los hilos dado que el procesamiento a realizar en cada frame depende de los datos (distribución de los *pixeles*) y no sólo del tamaño de la imagen.

Para obtener un buen rendimiento del algoritmo se debe balancear la carga de trabajo entre los diferentes threads. Para cumplir este objetivo se debe pensar en un esquema de distribución Bag of Task en el cual cada hilo se encarga de sacar una nueva tarea a realizar (frame a procesar) de un repositorio común a todos ellos [3].

Lo interesante de esta solución paralela es que no debe hacer ningún procesamiento extra al realizado por el algoritmo secuencial. Excepto aquellas acciones implícitas del compilador como son la creación o activación de los hilos y la sincronización para acceder al repositorio de tareas.

Para implementar el algoritmo paralelo se usó OpenMP sobre el lenguaje C, cuyo funcionamiento fue explicado en el capítulo 5.

Básicamente, un programa en OpenMP comienza con la ejecución de un hilo principal, el cual al llegar a un Bloque Paralelo genera o activa una cantidad previamente definida de threads para trabajar en paralelo. Al terminar todos los threads se destruyen o desactivan continuando únicamente el hilo principal hasta que se llegue a otro *Bloque Paralelo* o al final del programa.

Dentro de los *Bloques Paralelos* se pueden usar directivas que permiten especificar concurrencia entre iteraciones y tareas (constructores de trabajo compartido). Uno de estos constructores es la directiva *For* que permite distribuir las iteraciones entre los threads que se están ejecutando, especificando la forma de realizar el *Scheduling*. Para esto existen dos formas de asignar iteraciones a threads: el schedule estático y el schedule dinámico. En la asignación de schedule estático las iteraciones se asignan de manera estática a los threads en round robin, antes de comenzar su ejecución. En la asignación de schedule dinámico cada thread se le van asignando iteraciones a resolver cada vez que esta sin trabajo [3].

A continuación se muestra el pseudocódigo de la solución paralela. El hilo principal calcula el vector de características de la imagen que se desea buscar en el video (*Imagen a buscar*) por medio de la función *SIFT* y lo almacena en la variable compartida *descriptor*. Al llegar a la directiva ***#pragma omp parallel for***, genera el conjunto de threads para realizar en forma concurrente las iteraciones del *for i*. Los hilos sólo comparten la variable *descriptor*, el resto son variables privadas (o locales) a cada uno. En cada iteración se calculan los descriptores del frame correspondiente (*SIFT*). Luego, cada descriptor es comparado con los de la imagen a buscar para determinar si la distancia cumple con el umbral establecido y en caso de ser así obtiene la posición dentro del frame donde se ubica el objeto (para esto utiliza la función *MatchingSIFT*).

```
main() {
    descriptor = SIFT (Imagen a buscar)
    #pragma omp parallel for default (private) shared (descriptor)
    schedule (dynamic,1)
    for (i = 0; i ¡N; i++) {
        descriptorFrame = SIFT (Framei)
        (posición, puntaje) = MatchingSIFT (descriptor, descriptorFrame)
    }
}
```

6.3.3. Pruebas realizadas

Arquitectura utilizada

Para realizar las pruebas se ha usado un multicore Dell Poweredge 1950, cuyas características son las siguientes: 2 procesadores quad core Intel Xeón e5410 de 2.33 GHz, 4 Gb de memoria RAM (compartida entre ambos procesadores) y memoria cache L2 de 6Mb entre cada par de núcleos de los procesadores.

Diferentes pruebas

Para poder analizar el comportamiento del algoritmo paralelo se utiliza un video en tonos de gris cuyos frames tienen un tamaño de 480x640. Las pruebas realizadas varían en la cantidad N de frames utilizados (N = 250, 500, 1000, 1250, 1500, 1750, 2000, 2250 y 2500). Para cada N se ejecuta el algoritmo con schedule estático y dinámico. En cada caso se hicieron 12 corridas del algoritmo para obtener un tiempo paralelo promedio. En todas las pruebas se utilizaron 8 threads (1 por cada núcleo de la arquitectura).

6.4. Resultados

Para evaluar el comportamiento de los algoritmos desarrollados se analiza el speedup y la eficiencia, en este caso sobre arquitecturas homogéneas dado que todos los núcleos son iguales [9] [4]. Estas métricas fueron explicadas en el capitulo 5. En las tablas 6.1 hasta 6.10 se muestran los resultados obtenidos en las pruebas realizadas. Para cada cantidad de frames (N) se detalla el tiempo secuencial de la ejecución más rápida en segundos (TS), el tiempo paralelo promedio en segundos (TP) de las 12 ejecuciones para schedule estático y dinámico con los diferentes tamaños de bloques, el speedup y la eficiencia.

En la tabla 6.11 se visualizan los mejores tiempos de procesamiento promedio alcanzados en todas las pruebas realizadas. En estos resultados se puede observar que la distribución dinamica de los frames a procesar entre los procesos es más adecuada que la estática. Esto se debe al hecho de que la cantidad de trabajo que se debe realizar en cada imagen (frame) varía de acuerdo a

N	Schedule	Tam.Bloque	TS	TP	Speedup	Eficiencia
250	Estático	1	277.82	39.16	7.09	0.89
250	Dinámico	1	277.82	38.05	7.30	0.91
250	Estático	50	277.82	58.51	4.75	0.59
250	Dinámico	50	277.82	58.87	4.72	0.59
250	Estático	100	277.82	113.05	2.46	0.31
250	Dinámico	100	277.82	113.5	2.45	0.31
250	Estático	150	277.82	167.63	1.66	0.21
250	Dinámico	150	277.82	168.73	1.65	0.21
250	Estático	200	277.82	223.95	1.24	0.16
250	Dinámico	200	277.82	222.91	1.25	0.16
250	Estático	250	277.82	280.49	0.99	0.12
250	Dinámico	250	277.82	281.72	0.99	0.12

Tabla 6.1: Resultados de las 12 ejecuciones para N=250

N	Schedule	Tam.Bloque	TS	TP	Speedup	Eficiencia
500	Estático	1	552.83	75.71	7.30	0.91
500	Dinámico	1	552.83	73.82	7.49	0.94
500	Estático	50	552.83	115.78	4.77	0.60
500	Dinámico	50	552.83	114.57	4.83	0.60
500	Estático	100	552.83	114.66	4.82	0.60
500	Dinámico	100	552.83	115.57	4.78	0.60
500	Estático	150	552.83	169.81	3.26	0.41
500	Dinámico	150	552.83	169.94	3.25	0.41
500	Estático	200	552.83	226.53	2.44	0.31
500	Dinámico	200	552.83	226.17	2.44	0.31
500	Estático	250	552.83	281.17	1.97	0.25
500	Dinámico	250	552.83	282.47	1.96	0.24

Tabla 6.2: Resultados de las 12 ejecuciones para N=500

N	Schedule	Tam.Bloque	TS	TP	Speedup	Eficiencia
750	Estático	1	832.63	112.48	7.40	0.93
750	Dinámico	1	832.63	109.81	7.58	0.95
750	Estático	50	832.63	116.64	7.14	0.89
750	Dinámico	50	832.63	117.17	7.11	0.89
750	Estático	100	832.63	116.74	7.13	0.89
750	Dinámico	100	832.63	117.07	7.11	0.89
750	Estático	150	832.63	171.64	4.85	0.61
750	Dinámico	150	832.63	170.74	4.88	0.61
750	Estático	200	832.63	226.57	3.67	0.46
750	Dinámico	200	832.63	225.56	3.69	0.46
750	Estático	250	832.63	281.80	2.95	0.37
750	Dinámico	250	832.63	282.32	2.95	0.37

Tabla 6.3: Resultados de las 12 ejecuciones para N=750

N	Schedule	Tam.Bloque	TS	TP	Speedup	Eficiencia
1000	Estático	1	1103.67	148.08	7.45	0.93
1000	Dinámico	1	1103.67	145.40	7.58	0.95
1000	Estático	50	1103.67	173.31	7.14	0.80
1000	Dinámico	50	1103.67	173.17	7.11	0.80
1000	Estático	100	1103.67	228.17	7.13	0.60
1000	Dinámico	100	1103.67	227.85	7.11	0.61
1000	Estático	150	1103.67	172.54	4.85	0.80
1000	Dinámico	150	1103.67	172.87	4.88	0.80
1000	Estático	200	1103.67	228.99	3.67	0.60
1000	Dinámico	200	1103.67	228.31	3.69	0.60
1000	Estático	250	1103.67	282.57	2.95	0.49
1000	Dinámico	250	1103.67	283.06	2.95	0.49

Tabla 6.4: Resultados de las 12 ejecuciones para N=1000

N	Schedule	Tam.Bloque	TS	TP	Speedup	Eficiencia
1250	Estático	1	1390.55	196.86	7.06	0.88
1250	Dinámico	1	1390.55	181.24	7.67	0.96
1250	Estático	50	1390.55	227.33	6.12	0.76
1250	Dinámico	50	1390.55	227.00	6.13	0.77
1250	Estático	100	1390.55	228.95	6.07	0.76
1250	Dinámico	100	1390.55	229.54	6.06	0.76
1250	Estático	150	1390.55	226.32	6.14	0.77
1250	Dinámico	150	1390.55	227.61	6.11	0.76
1250	Estático	200	1390.55	230.48	6.03	0.75
1250	Dinámico	200	1390.55	230.64	6.03	0.75
1250	Estático	250	1390.55	285.07	4.88	0.61
1250	Dinámico	250	1390.55	285.27	4.87	0.61

Tabla 6.5: Resultados de las 12 ejecuciones para N=1250

N	Schedule	Tam.Bloque	TS	TP	Speedup	Eficiencia
1500	Estático	1	1655.61	223.19	7.42	0.93
1500	Dinámico	1	1655.61	216.90	7.63	0.95
1500	Estático	50	1655.61	230.65	7.18	0.90
1500	Dinámico	50	1655.61	230.67	7.18	0.90
1500	Estático	100	1655.61	230.95	7.17	0.90
1500	Dinámico	100	1655.61	230.68	7.18	0.90
1500	Estático	150	1655.61	341.36	4.85	0.61
1500	Dinámico	150	1655.61	341.56	4.85	0.61
1500	Estático	200	1655.61	230.77	7.17	0.90
1500	Dinámico	200	1655.61	233.18	7.10	0.89
1500	Estático	250	1655.61	286.74	5.77	0.72
1500	Dinámico	250	1655.61	287.19	5.76	0.72

Tabla 6.6: Resultados de las 12 ejecuciones para N=1500

N	Schedule	Tam.Bloque	TS	TP	Speedup	Eficiencia
1750	Estático	1	1938.21	260.86	7.43	0.93
1750	Dinámico	1	1938.21	252.82	7.67	0.96
1750	Estático	50	1938.21	292.64	6.62	0.83
1750	Dinámico	50	1938.21	286.84	6.76	0.84
1750	Estático	100	1938.21	340.44	5.69	0.71
1750	Dinámico	100	1938.21	337.94	5.74	0.72
1750	Estático	150	1938.21	345.79	5.61	0.70
1750	Dinámico	150	1938.21	342.23	5.66	0.71
1750	Estático	200	1938.21	395.06	4.91	0.61
1750	Dinámico	200	1938.21	397.43	4.88	0.61
1750	Estático	250	1938.21	288.00	6.73	0.84
1750	Dinámico	250	1938.21	287.03	6.75	0.84

Tabla 6.7: Resultados de las 12 ejecuciones para N=1750

N	Schedule	Tam.Bloque	TS	TP	Speedup	Eficiencia
2000	Estático	1	2226.80	293.42	7.59	0.95
2000	Dinámico	1	2226.80	288.78	7.71	0.96
2000	Estático	50	2226.80	294.65	7.56	0.94
2000	Dinámico	50	2226.80	289.31	7.70	0.96
2000	Estático	100	2226.80	343.75	6.48	0.81
2000	Dinámico	100	2226.80	343.21	6.49	0.81
2000	Estático	150	2226.80	347.33	6.41	0.80
2000	Dinámico	150	2226.80	344.92	6.46	0.81
2000	Estático	200	2226.80	455.92	4.88	0.61
2000	Dinámico	200	2226.80	455.48	4.89	0.61
2000	Estático	250	2226.80	288.05	7.73	0.97
2000	Dinámico	250	2226.80	287.96	7.73	0.97

Tabla 6.8: Resultados de las 12 ejecuciones para N=2000

N	Schedule	Tam.Bloque	TS	TP	Speedup	Eficiencia
2250	Estático	1	2490.00	330.40	7.54	0.94
2250	Dinámico	1	2490.00	324.43	7.68	0.96
2250	Estático	50	2490.00	348.35	7.15	0.89
2250	Dinámico	50	2490.00	344.80	7.22	0.90
2250	Estático	100	2490.00	350.57	7.10	0.89
2250	Dinámico	100	2490.00	346.33	7.19	0.90
2250	Estático	150	2490.00	347.70	7.16	0.90
2250	Dinámico	150	2490.00	347.89	7.16	0.89
2250	Estático	200	2490.00	456.30	5.46	0.68
2250	Dinámico	200	2490.00	456.59	5.45	0.68
2250	Estático	250	2490.00	566.29	4.40	0.55
2250	Dinámico	250	2490.00	562.91	4.42	0.55

Tabla 6.9: Resultados de las 12 ejecuciones para N=2250

N	Schedule	Tam.Bloque	TS	TP	Speedup	Eficiencia
2500	Estático	1	2780.49	369.59	7.52	0.94
2500	Dinámico	1	2780.49	360.68	7.71	0.96
2500	Estático	50	2780.49	406.16	6.85	0.86
2500	Dinámico	50	2780.49	400.42	6.94	0.87
2500	Estático	100	2780.49	458.86	6.06	0.76
2500	Dinámico	100	2780.49	455.89	6.10	0.76
2500	Estático	150	2780.49	452.94	6.14	0.77
2500	Dinámico	150	2780.49	453.44	6.13	0.77
2500	Estático	200	2780.49	457.78	6.07	0.76
2500	Dinámico	200	2780.49	457.01	6.08	0.76
2500	Estático	250	2780.49	570.07	4.88	0.61
2500	Dinámico	250	2780.49	569.95	4.88	0.61

Tabla 6.10: Resultados de las 12 ejecuciones para N=2500

N	Schedule	Tam.Bloque	TS	TP	SpeedUp	Eficiencia
250	Dinámico	1	277.82	38.05	7.30	0.91
500	Dinámico	1	552.83	73.82	7.49	0.94
750	Estático	1	832.63	109.48	7.58	0.95
1000	Estático	1	1103.67	145.40	7.59	0.95
1250	Dinámico	1	1390.55	181.24	7.67	0.96
1500	Dinámico	1	1655.61	216.90	7.63	0.95
1750	Dinámico	1	1938.21	252.82	7.67	0.96
2000	Estático	1	2226.80	288.78	7.71	0.96
2250	Estático	1	2490.00	324.43	7.67	0.96
2500	Estático	1	2780.49	360.68	7.71	0.96

Tabla 6.11: Resumen con el mejor tiempo paralelo en cada valor de N

los datos de la misma y no al tamaño, por lo tanto, no se puede saber de antemano cuanto procesamiento se debe realizar en cada uno de los frames para distribuirlos equitativamente (en cuanto al trabajo a realizar) entre los diferentes threads.

En la figura 6.4 se observa la eficiencia lograda por el algoritmo que usa distribución dinámica en los distintos tamaños de bloque probados. Por razones de legibilidad se muestran los resultados para una única cantidad de frames (N = 2500), pero el comportamiento es homogéneo para el resto de las pruebas. En el gráfico puede verse como decrece la eficiencia del algoritmo a medida que aumenta el tamaño de bloque. Esto se debe a que al repartir el trabajo en menos porciones de mayor tamaño, la cantidad de procesamiento que se debe realizar en cada uno de los bloques es más heterogénea. El tamaño de bloque ideal en las pruebas realizadas en 1. El overhead generado por tener que sincronizarse para buscar más trabajo en un repositorio común por cada frame a procesar, es ocultado por el balance de carga de trabajo conseguido.

En la figura 6.5 se muestra el comportamiento del algoritmo con distribución dinámica y tamaño de bloque igual a 1 para las diferentes cantidades de frames (N). Para ver de forma clara las diferencias se grafica la eficiencia en el rango [0.9 - 1]. Puede observarse que el algoritmo paralelo logra un rendimiento cercano al óptimo *(eficiencia = 1)*. La eficiencia crece al incrementar la cantidad de frames utilizados tendiendo a estacionarse a partir de las 1200 imágenes. Este comportamiento es el esperado de acuerdo a la Ley de Amdahl [9] [3].

Por último, es importante determinar la mejora obtenida por el algoritmo paralelo en cuanto a la cantidad de frames que se pueden procesar por segundo *(Tasa de Procesamiento)*, pensando en el tratamiento en tiempo real del video. Para calcular este valor se utiliza la ecuacion 6.1

$$TasadeProcesamiento = N/TP \tag{6.1}$$

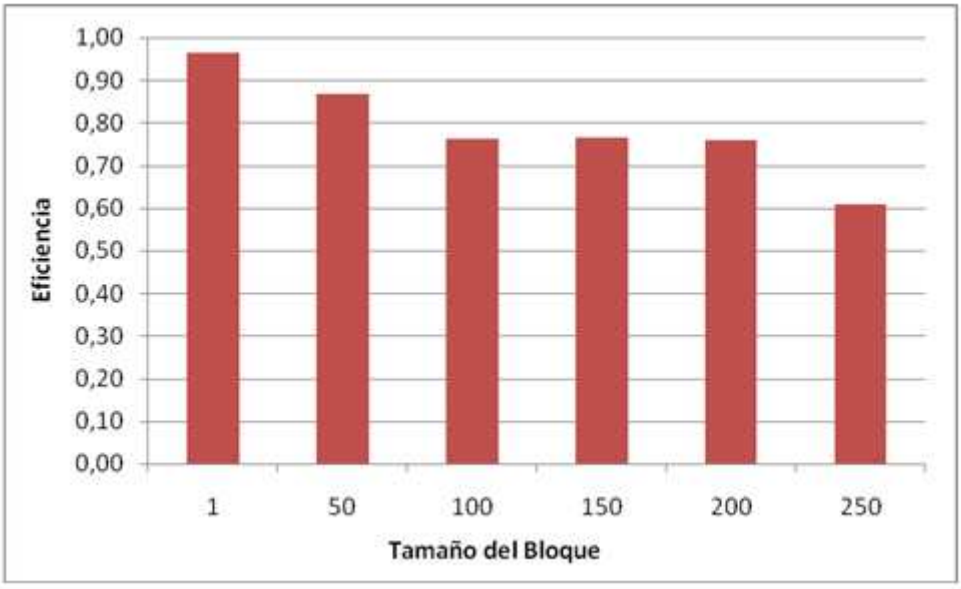

Figura 6.4: Eficiencia conseguida con diferente tamaño de bloque para N=2500 y distribución dinámica

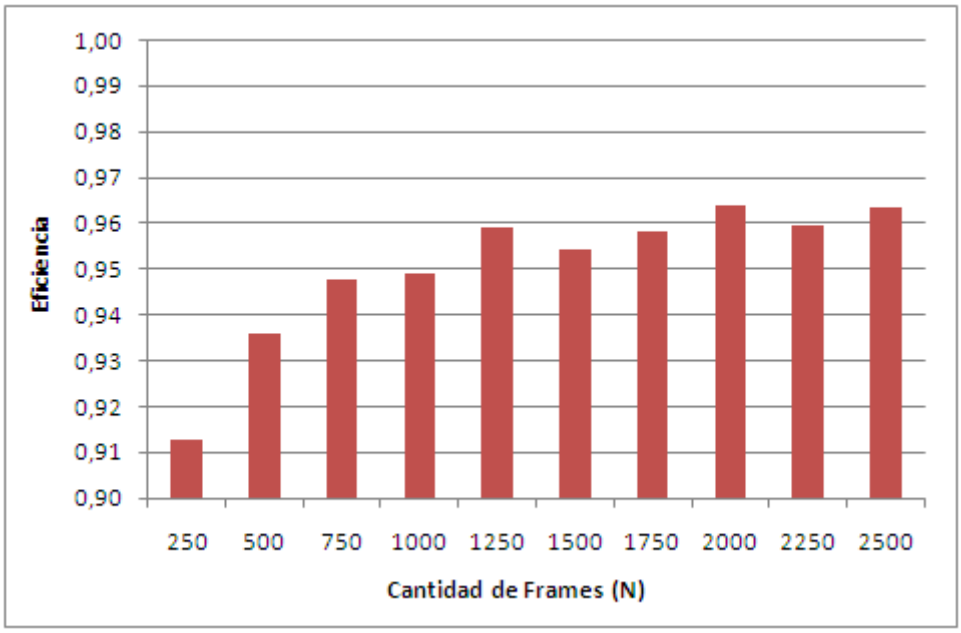

Figura 6.5: Eficiencia conseguida con distribución dinámica y tamaño de bloque 1 para las diferentes cantidades de frames (N) probadas

N	Tasa de Procesamiento Secuencial	Tasa de Procesamiento Paralelo
250	0.90	6.57
500	0.90	6.77
750	0.90	6.83
1000	0.91	6.88
1250	0.90	6.90
1500	0.91	6.92
1750	0.90	6.92
2000	0.90	6.93
2250	0.90	6.94
2500	0.90	6.93

Tabla 6.12: Tasa de procesamiento para el algoritmo secuencial y el paralelo para las diferentes valores de N

En la tabla 6.12 se presenta la tasa de procesamiento para el algoritmo secuencial y paralelo con los diferentes valores de N. Los mismos datos son mostrados en el gráfico de la figura 6.6. Como puede observarse, la cantidad de frames que se pueden procesar por segundo pasa de ser menor a 1, al usar el algoritmo secuencial, a casi 7 en el algoritmo paralelo.

6.5. Conclusiones

Se ha presentado una versión paralela del método SIFT que permite su aplicación para realizar tracking de video. Los resultados obtenidos al procesar un video en tonos de grises han resultado satisfactorios.

El algoritmo paralelo no obtiene una eficiencia óptima, pero si muy cercana a ella, debido al segmento de código que se debe resolver en forma secuencial, al overhead generado al crear/activar y sincronizar los threads y al tamaño de las cache compartidas por cada par de cores el cual es lo suficientemente grande como para trabajar con una imagen (algoritmo secuencial), pero no para procesar dos frames al mismo tiempo (dos hilos del algoritmo paralelo), lo que genera más fallos de cache.

Esto afecta en mayor grado a medida que se aumenta el tamaño de los frames. El algoritmo permite mejorar la cantidad de frames analizados por segundo *(tasa de procesamiento)* permitiendo el procesamiento en tiempo real de videos.

En el siguiente y último capítulo se presentan las conclusiones finales de esta tesina.

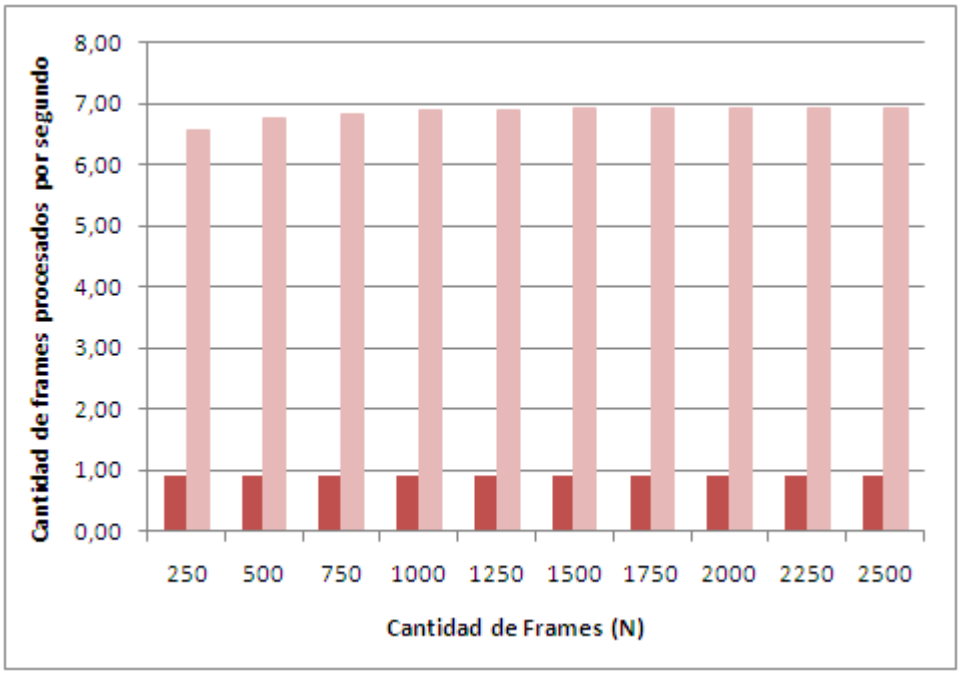

Figura 6.6: Tasa de procesamiento para el algoritmo secuencial y el paralelo para las diferentes valores de N

Capítulo 7

Conclusiones generales

El reconocimiento de objetos en video es una tarea de sumo interés en distintas áreas tales como edición de video, seguridad, entretenimientos y hasta control de accesos.

La correspondencia de características locales se ha convertido en el método más usado para comparar imágenes. Existen métodos que permiten caracterizar una imagen a través de la representación vectorial de sus pixeles más relevantes entre los cuales SIFT es uno de los más utilizados.

El algoritmo SIFT toma una imagen y la transforma en una colección de vectores de características locales. Cada uno de estos vectores, es distintivo e invariante a cualquier escala, rotación y traslación de la imagen. Las mismas pueden ser usadas para encontrar objetos distintivos en imágenes diferentes.

Se han desarrollado distintas variantes de este algoritmo como PCA-SIFT, RIFT, G-RIFT y SURF, descriptas en el capítulo 3.

La extracción de características utilizando los descriptores SIFT es un proceso costoso en tiempo al aplicarlo para reconocer objetos en los diferentes frames de un video. Esto dificulta su aplicación en procesos que requieren obtener respuesta en tiempo real.

Se ha presentado una versión paralela del método SIFT que permite su aplicación para realizar tracking de video. Para esto se analizaron tres alternativas, división de frames, división por puntos claves y distribución de frames. Las pruebas determinaron que la mejor alternativa es la distribución de frames ya que no necesita realizar trabajo extra para la comunicación debido a que los frames son independientes y la única parte secuencial que se realiza es el procesamiento del objeto a buscar. Los resultados obtenidos al procesar un video en tonos de grises han resultado satisfactorios como puede verse en el capitulo 6. El algoritmo paralelo no obtiene una eficiencia óptima, pero si muy cercana a ella.

Actualmente se está trabajando en incrementar el tamaño de los frames con el objetivo de producir una reducción importante en la eficiencia. Esto lleva a analizar una solución de grano fino para poder realizar la paralelización a nivel de obtención de puntos de interés en cada frame. Al distribuir el procesamiento de cada imagen entre los distintos cores, se reduce la cantidad de datos que debe almacenar cada hilo, decrementando los fallos de cache.

Poder representar objetos de un video a través de un conjunto de vectores característicos no sólo se identifica a dichos objetos sino también al video que los contiene. Este enfoque podría permitir caracterizar videos a partir de sus objetos más relevantes haciendo factible el uso de técnicas de Minería de Textos para recuperar y catalogar dichos videos automáticamente. Este último enfoque es una de las líneas actuales de investigación.

Indice de figuras

Indice de tablas

Bibliografía

[1] Ray A. K. Acharya, T. *Image processing: principles and applications.* John Wiley and Sons, 2005.

[2] Sánchez L. Fernández R. Á. Mostaza J. C. Alegre, E. *Procesamiento Digital de Imagen: fundamentos y prácticas con Matlab.* Universidad de Leon, 2003.

[3] George Karypis Vipin Kumar Ananth Grama, Anshul Gupta. *Introduction to Parallel Computing, Second Edition.* Addison Wesley, 2003.

[4] Gregory R. Andrews. *Foundations of Multithreaded, Parallel and Distributed Programming.* Addison Wesley, 1999.

[5] Blaise Barney. Openmp. *Livermore Computing*, 2010.

[6] Tuytelaars T y Gool V Bay H. Speeded up robust features. *Department of Information Technology and Electrical Engineering*, 2005.

[7] M. Brown and D.G. Lowe. Invariant features from interest point groups. *British Machine. Vision Conference, Cardiff, Wales*, pages 656–665, 2002.

[8] Philippe Salembier B.S. Manjunath and Thomas Sikora. *Introduction to MPEG-7: Multimedia Content Description Interface.* Wiley and Sons, 2002.

[9] Leopold C. *Parallel and Distributed Computing. A survey of Models, Paradigms, and Approaches.* Wiley, New York, 2001.

[10] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[11] Sebastián Fondra Claudia Russo, Nicolás Alonso. Compresion de imagenes, datos y video. *CACIC 2009, Universidad de Jujuy, Argentina*, pages 16–19, 2009.

[12] Arturo de la Escalera Hueso. *Visión por computador: fundamentos y métodos.* Prentice Hall, 2001.

[13] Ian; Fox Geoffrey; Gropp William; Kennedy Ken; Torczon Linda y White Andy Dongarra, Jack; Foster. *Sourcebook of Parallel Computing.* Morgan Kaufmann Publishers, 2003.

[14] William; Sterling Thomas y Yelick Katherine El-Ghazawi, Tarek; Carlson. *UPC: Distributed Shared Memory Programming.* John Wiley and Sons, 2005.

[15] J. Forsyth, D.A.and Ponce. *Computer Vision – A Modern Approach.* Prentice Hall, 2003.

[16] Parks H. Corners detectors. *McGill University, Canada*, 2005.

[17] C. Harris and M.J. Stephens. A combined corner and edge detector. *In Alvey Vision Conference*, page 147–152, 1988.

[18] D.G. Helmer, S.and Lowe. Object class recognition with many local features. *Workshop on Generative Model Based Vision 2004 (GMBV)*, 2004.

[19] Smolic A. Froehlich B. Wiegand T. Heymann S., Müller K. Sift implementation and optimization for general-purpose gpu. *In Proc. of the WSCG'07, Plzen, Czech Republic*, 2007.

[20] D.G. Iryna Gordon, I.andLowe. Modelling, recognition and tracking with invariant image features. *International Symposium on Mixed and Augmented Reality (ISMAR), Arlington, VA*, pages 110–119, 2004.

[21] I. T. Joliffe. *Principal Component Analysis*. Springer-Verlag, 1986.

[22] Y. Ke and R. Sukthankar. Pca-sift: a more distinctive representation for local image descriptors. *IEEE Conference on Computer Vision and Pattern Recognition*, II:506–513, 2004.

[23] L. y A. Rosenfeld Kitchen. Gray level corner detection. *Pattern Recognition Letters*, pages 85–102, 1982.

[24] Dr. García Reyes C. Lic. Martínez R. *Estado del arte sobre los métodos de detección de características locales*. Cuba, 2010.

[25] George C. Stockman Linda G. Shapiro. *Computer Vision*. Prentice Hall, 2001.

[26] David G. Lowe. Object recognition from local scale-invariant features. *International Conference on Computer Vision, Corfu, Greece*, pages 1150–1157, 1999.

[27] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, pages 91–110, 2004.

[28] C.A Mikolajczyk, K.; Schmid. Performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2005.

[29] K. Mikolajczyk and C. Schmid. Scale and afine invariant interest point detectors. *International Journal of Computer Vision*, page 63–86, 2004.

[30] Dick y Farrel Jacqueline Proulx Nichols, Bradford; Buttlar. *Pthreads programming: A Posix Standard for Better Multiprocessing*. O'Reilly, 1996.

[31] Ian; Garland Michael y Skadron Kevin Nickolls, John; Buck. *Scalable parallel programming with CUDA*. Queue, 2008.

[32] Witkin. A. P. Scale-space filtering. *In International Joint Conference On Artificial Intelligence, Karlsruhe, Germany*, 1983.

[33] Beaudet P. R. Rotationally invariant image operators. *International Joint Conference on Pattern Recognition*, 579:579–583, 1978.

[34] Richard Woods Rafael Gonzalez. *Tratamiento Digital de Imágenes*. Addison Wesley, 1996.

[35] Steven Eddins Rafael Gonzalez, Richard Woods. *Digital Image Processing using Matlab.* Pearson Prentice Hall, 2004.

[36] A. M. Romero and M. Cazorla. Local feature view clustering for 3d object recognition. *IEEE Conference on Computer Vision and Pattern Recognition, Kauai, Hawaii*, pages 682–688, 2001.

[37] A. M. Romero and M. Cazorla. Comparativa de detectores de características visuales y su aplicación al slam. *X Workshop de agentes físicos*, 2009.

[38] Iván Danilo García Santillán. *Vision artificial y Procesamiento Digital de Imágenes usando matlab.* Ibarra - Ecuador, 2008.

[39] Mallick A. Siddha S., Pallipadi V. Process scheduling challenges in the era of multicore processors. *Intel Technology Journal*, 11, 2007.

[40] S.M. Smith and J.M. Brady. Susan - a new approach to low level image processing. *Int. Journal of Computer Vision*, pages 45–78, 1997.

[41] William Snir, Marc y Gropp. *MPI. The Complete Reference. 2nd edition.* The MIT Press, 1998.

[42] In So Kweon Sungho Kim, Kuk-Jin Yoon. Object recognition using a generalized robust invariant feature and gestalt's law of proximity and similarity. *Conference on Computer Vision and Pattern Recognition*, 2006.

[43] D. Suter. Assessing the performance of corner detectors for point feature tracking applications. *Dept. of Electrical and Computer Systems Engineering Monash University, Australia*, 2002.

[44] Jean Ponce Svetlana Lazebnik, Cordelia Schmid. Semi local affine parts for object recognition. *British Machine Vision Conference*, 2:959–968, 2004.

[45] N. Trendafilov, I. T. Jolliffe, and M. Uddin. A modified principal component technique based on the lasso. *Journal of Computational and Graphical Statistics*, 12:531–547, 2003.

[46] Dreschler L. y Nagel H. On the selection of critical points and local curvature extrema of region boundaries for interframe matching. *In Proc. Of the International conference on Pattern Recognition*, pages 542–544, 1982.