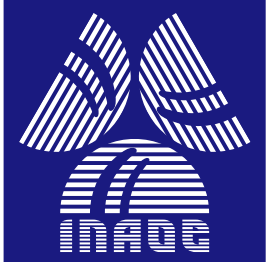


Arquitectura hardware para la detección de puntos de interés del SIFT

por

Leonardo Chang Fernández

Tesis sometida como requisito parcial para obtener el grado de

MAESTRO EN CIENCIAS
EN LA ESPECIALIDAD DE
CIENCIAS COMPUTACIONALES

en el

Instituto Nacional de Astrofísica, Óptica y Electrónica
Tonantzintla, Puebla
Noviembre 2010

Supervisada por:

Dr. Luis Enrique Sucar Succar
Dr. Miguel Octavio Arias Estrada
Dr. José Hernández Palancar

A mis padres y hermana
este minúsculo paso
en mi camino
a ser como ustedes.

Agradecimientos

Agradezco a mis asesores Dr. José Hernández Palancar, Dr. Luis Enrique Sucar Succar y Dr. Miguel Octavio Arias Estrada, por todo el apoyo, guía y consejos brindados para el desarrollo de esta tesis.

A mis sinodales Dra. Claudia Feregrino Uribe, Dr. René Cumplido Parra y Dr. Carlos Alberto Reyes García por sus observaciones y sugerencias que ayudaron a mejorar la calidad de esta tesis.

Al CENATAV, por formarme como profesional e investigador.

A mis profesores de la maestría, por todos los conocimientos que me proporcionaron en este último año.

A mis padres, por toda la sabiduría, educación, consejos y cariño que me han brindado.

A mi hermana, por ser mi mejor amiga y cómplice incondicional.

A Marieta, por todo el amor y cariño que me entrega.

A toda mi familia y amigos, por hacer mi vida muy feliz.

A todos mis compañeros de trabajo del CENATAV, en especial a Airel y Raudel, con quienes he compartido la mayor parte del tiempo de esta tesis.

A todos mis compañeros de maestría del INAOE.

Al INAOE y al CONACyT (a través de la beca No. 240251) por brindarme los medios materiales y económicos para realizar esta investigación.

Leonardo Chang Fernández.
Tonantzintla, Puebla. 19 de Noviembre de 2010.

Resumen

El uso de características locales en imágenes se ha vuelto muy popular gracias a sus resultados prometedores. Estos han demostrado considerables beneficios en una gran variedad de aplicaciones tales como reconocimiento de objetos, recuperación de imágenes, navegación de robots, construcción de panoramas y otros. Varios algoritmos se han desarrollado en esta área. Uno de los más populares y que mejores resultados ha mostrado es el SIFT. Este algoritmo permite encontrar estructuras locales que estarán presentes en distintas vistas de la imagen. Además, permite obtener una descripción de dichas estructuras invariante a transformaciones en la imagen como traslación, rotación, escala y deformaciones afines. Sin embargo, entre sus principales desventajas se encuentra su alto costo computacional. Por esto surge la necesidad de buscar alternativas para la aceleración del mismo. Con ese fin, en este trabajo se propone el diseño e implementación de una arquitectura hardware eficiente basada en FPGAs (del inglés, *Field Programmable Gate Array*) para la detección de puntos de interés del algoritmo SIFT.

Con el objetivo de sacar el máximo provecho del paralelismo en la detección de puntos de interés del algoritmo SIFT y minimizar el área del dispositivo ocupada por su implementación en hardware, se reformuló parte del algoritmo. Las principales características de dicha reformulación son la explotación del paralelismo de datos, la explotación de la propiedad de separabilidad del *kernel* de convolución Gaussiano y el entrelazado del procesamiento de las octavas. El principal aporte de la arquitectura tunelizada aquí propuesta y la principal diferencia con el resto de las arquitecturas reportadas en la literatura, radica en que a medida que aumenta el número de octavas a procesar, la cantidad de área del dispositivo ocupada se mantiene casi constante, solamente aumentando en el número de bloques de memoria necesarios para almacenar las nuevas octavas y la lógica necesaria para controlar el entrelazado de más octavas.

Las pruebas y experimentos realizados a la arquitectura, soportan la aportación antes mencionada, así como la exactitud, repetitividad y distintividad de los resultados obtenidos. También se realizan pruebas de implementación relacionadas con el área del dispositivo ocupada, restricciones de tiempo, entre otras. La arquitectura presentada en este trabajo logra detectar los puntos de interés en una imagen de 320×240 en 1.1 ms, lo que representa una aceleración de 250x con respecto a una implementación en software.

Abstract

The use of local features in images has become very popular thanks to its promising results. These have shown significant benefits in a variety of applications such as object recognition, image retrieval, robot navigation, panorama stitching, and others. Several algorithms have been developed in this area. One of the most popular and widely used is the SIFT method. This algorithm finds local structures that are present in different views of the image. It also allows a description of these structures invariant to image transformations such as translation, rotation, scale and affine transformations. However, its main disadvantage is its high computational cost. This arises the need to seek alternatives to its acceleration. To that end, this paper proposes a design and implementation of an efficient hardware architecture based on FPGAs (Field Programmable Gate Array) for the candidate keypoints detection stage of the SIFT algorithm.

In order to take full advantage of the parallelism in the candidate keypoints detection stage and to minimize the silicon area occupied by its implementation in hardware, part of the algorithm was reformulated. The main characteristics of this reformulation are the exploitation of data parallelism, the exploitation of the separability property of the Gaussian kernel and the octaves processing interleaving. The main contribution of the proposed pipelined architecture and the main difference with the rest of the architectures reported in the literature, is that while increasing the number of octaves to be processed, the amount of occupied area of the device remains almost constant, only increased in the number of blocks of memory needed to store the new octaves and in the logic needed to control the interleaving of more octaves.

The tests and experiments conducted to the architecture evidenced the contribution mentioned above, as well as accuracy, repeatability and distinctiveness of the extracted features. Tests are also related to device area occupation, timing constraints, among others. The architecture presented in this work is able to detect candidate keypoints in an image of 320×240 in 1.1 milliseconds, which represents a speedup of 250x with respect to a software implementation.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En el campo de la visión por computadora, es necesario extraer características de las imágenes que puedan ser utilizadas en aplicaciones como reconocimiento de objetos, recuperación de imágenes, navegación de robots, construcción de panoramas, reconocimiento de rostros, etcétera. Estas características deben ser invariantes a transformaciones en la imagen como traslación, rotación, escala y deformaciones afines. Se necesita, además, que el proceso de extracción de características sea repetitivo y preciso, de modo tal que las mismas características sean extraídas en distintas imágenes que contengan el mismo objeto, y que las mismas sean distintivas, o sea, que las diferentes características extraídas puedan distinguirse unas de otras.

En la década pasada, se lograron avances significativos en esta dirección con el desarrollo de las características locales invariantes. Uno de los métodos más usados y que ha mostrado mejores resultados en esta área es el SIFT (del inglés, *Scale Invariant Feature Transform*) propuesto por David Lowe. Este método fue inicialmente presentado a la comunidad de visión por computadoras en un artículo a finales de la década pasada en 1999 [18] y más recientemente, con algunas mejoras al algoritmo, en el 2004 [19]. Una de las principales desventajas de este algoritmo es que, en aras de lograr su robustez, es resultado de procesos iterativos complejos y tardados, lo que representa un alto costo computacional.

Por otra parte, una técnica que ha sido muy utilizada en los últimos años con el fin de acelerar diversas tareas de cómputo, es el uso de los FPGAs (del inglés, *Field*

Programmable Gate Array). Los FPGAs son dispositivos revolucionarios que combinan los beneficios del hardware y el software. En estos dispositivos se puede implementar circuitos, brindando grandes beneficios en cuanto a energía, área y desempeño en comparación con el software. Además, pueden ser reconfigurados de manera simple y barata para implementar un amplio rango de tareas.

En este trabajo de tesis, con el objetivo de acelerar la extracción de características SIFT, se propone una arquitectura hardware para FPGAs que implementa una de las etapas de mayor costo computacional de este algoritmo: la detección de puntos de interés. La arquitectura propuesta, se muestra superior en cuanto a la eficiencia en el uso de los recursos del FPGA y en tiempos de procesamiento, con respecto a las otras arquitecturas reportadas en la literatura con el mismo fin.

1.1. Descripción del problema

El método de características locales SIFT, tiene como una de sus principales desventajas un alto costo computacional, resultado de procesos complejos para obtener invariabilidad ante cambios de rotación, escala, traslación y pequeñas transformaciones afines y cambios de iluminación, siendo una de las etapas más costosas la detección de puntos de interés. Para una imagen de 1024×768 píxeles, una implementación software [31] de este algoritmo, en una PC de prestaciones considerables (CPU P4 3.0 GHz, RAM 2 Gb), toma alrededor de 3 segundos en extraer aproximadamente 1200 características.

Existen varios escenarios (por ejemplo: seguimiento de objetos, recuperación de imágenes en línea, navegación de robots, etc.) que requieren que estas características locales sean extraídas y comparadas en tiempo real (aproximadamente 30 cuadros por segundo) e incluso usando imágenes de alta resolución (más de 2 megapíxeles). Muy pocos sistemas en la actualidad, corriendo sobre computadoras personales, logran tales tasas de procesamiento, y los que alcanzan dicha velocidad, procesan imágenes de baja resolución o reducen el número de octavas y escalas procesadas sacrificando la robustez del algoritmo. Por lo tanto, es necesaria una implementación de dicho algoritmo que logre trabajar en

tiempo real.

1.2. Objetivos

Muchas tareas dentro del campo de aplicación del SIFT necesitan que estas características sean extraídas en tiempo real o en imágenes de alta resolución, por lo que su alto costo computacional es una de las grandes limitantes del SIFT en este tipo aplicaciones. Por lo tanto, dada la necesidad de tener implementaciones o variantes más rápidas de este algoritmo, la investigación se centra en desarrollar un algoritmo paralelo eficiente implementado en hardware que minimice el costo computacional de la detección de puntos de interés del SIFT.

Objetivo general

Teniendo en cuenta las ventajas de los FPGAs en la aceleración de tareas de cómputo, el objetivo de este trabajo es diseñar e implementar una arquitectura hardware eficiente para la detección de puntos de interés del algoritmo SIFT, que permita reducir en al menos un orden de magnitud los tiempos de procesamiento respecto a las implementaciones software reportadas en la literatura.

Objetivos específicos

Como objetivos específicos de este trabajo se encuentran los siguientes:

1. Identificar procesos de mayor potencial para el paralelismo en la etapa de detección de puntos de interés del SIFT, que lleven a una aceleración considerable de esta etapa usando FPGAs.

2. Diseñar una arquitectura hardware eficiente para la detección de puntos de interés del SIFT.

3. Obtener una implementación eficiente de dicha arquitectura mediante el uso de un lenguaje o herramienta de descripción de hardware.

1.3. Contribuciones

En este trabajo de tesis se propone una arquitectura hardware para la etapa de detección de puntos de interés del método SIFT. Esta parte del algoritmo fue reformulada en aras de explotar al máximo el grado de paralelismo de la misma, minimizar el área del dispositivo ocupada por su implementación en hardware y de esa manera sacar el mayor provecho posible de una implementación en FPGA de esta etapa del SIFT.

La principal contribución de la arquitectura y de la reformulación del algoritmo aquí propuesta, radica en que, a medida que aumenta el número de octavas a procesar, la cantidad de área del dispositivo ocupada se mantiene casi constante, solamente aumentando en el número de bloques de memoria necesarios para almacenar las nuevas octavas y la lógica necesaria para controlar el entrelazado de más octavas. Este fenómeno se debe a que todas las octavas para una misma escala, sin importar la cantidad, siempre serán procesadas en el mismo bloque de convolución. Este elemento resulta de gran importancia ya que la tendencia en el campo de la visión por computadoras es a trabajar con imágenes de mayor tamaño, y el número de octavas es función de las dimensiones de la imagen. Por lo tanto, para imágenes de mayor resolución y por ende para un mayor número de octavas, la lógica necesaria para procesar estas imágenes de mayor resolución será la misma.

La contribución antes mencionada fue soportada por las pruebas y experimentos realizados a la arquitectura, que mostraron cuantitativamente la ventaja introducida con el entrelazado del procesamiento de las octavas, obteniéndose ahorros en el uso de área del dispositivo por encima del 50 % y con tendencia al aumento mientras más octavas fuesen procesadas. Además, se reportaron pequeñas diferencias en la generación, por parte de la arquitectura propuesta, del espacio-escala de diferencias de Gaussianas en comparación con una implementación software y resultados idénticos en la detección de extremos en dicho espacio. También, se presentaron pequeñas disminuciones en la repetitividad y distintividad de las características SIFT detectadas a partir de la arquitectura en variaciones en la imágenes como punto de vista, rotación y escalamiento, compresión JPEG y

desenfoque. La arquitectura propuesta superó los indicadores de tiempo y eficiencia en el uso del área del dispositivo en casi todos los parámetros en comparación con los trabajos relacionados.

La arquitectura presentada en este trabajo, logra detectar los puntos de interés en una imagen a razón de un píxel cada dos ciclos de reloj. Implementada en un Xilinx Virtex II Pro, con una configuración de tres octavas y seis escalas, y una restricción de reloj de 145 MHz, una imagen de 320×240 píxeles es procesada en 1.1 ms, lo que representa una aceleración de 250x (dos órdenes de magnitud) con respecto a la implementación software de Vedaldi.

1.4. Estructura de la tesis

Este documento de tesis consta de 8 capítulos. En el Capítulo 2 se describe el algoritmo SIFT y se detalla a profundidad la etapa de detección de puntos de interés, la cual es de mayor interés en este trabajo. En el Capítulo 3 se explican las ventajas del uso de los FPGAs en la aceleración de tareas de cómputo, así como su relación con el procesamiento paralelo. En el Capítulo 4 se discuten los trabajos presentados en la literatura hasta la actualidad con el fin de acelerar el SIFT usando FPGAs. En el Capítulo 5 se presentan las reformulaciones realizadas a la primera etapa del SIFT con el objetivo de obtener el máximo desempeño en una implementación hardware de esta etapa del algoritmo. El Capítulo 6 detalla la arquitectura hardware que implementa el algoritmo obtenido en el Capítulo 5. Las pruebas realizadas a la arquitectura propuesta y los principales resultados obtenidos son discutidos en el Capítulo 7. Finalmente, en el Capítulo 8 se analizan las conclusiones generales de la tesis y se perfilan futuras líneas de investigación.

Capítulo 2

El método de características locales SIFT

2.1. Características locales

Los métodos basados en comparaciones de imágenes completas o ventanas dentro de ellas son apropiados para aprender o describir la estructura global de los objetos, pero no pueden lidiar con problemas de oclusión parcial, fuertes cambios de pose o perspectiva, ni con objetos no rígidos.

En la década pasada, se tuvieron avances significativos en la solución de estos problemas con el desarrollo de las características locales invariantes. El uso de estas características permite encontrar estructuras locales que estarán presentes en distintas vistas de la imagen. Además, permite obtener una descripción de dichas estructuras invariante a transformaciones en la imagen como traslación, rotación, escala y deformaciones afines. El uso de estas características ha demostrado buenos resultados en una gran variedad de aplicaciones como reconocimiento de objetos [19], recuperación de imágenes [25], navegación de robots [21], construcción de panoramas [5] y reconocimiento de rostros [8], entre otros. Un estudio y comparación de los principales métodos de extracción de características locales es presentado en [29].

El propósito de las características locales es brindar una representación que permita hallar correspondencias entre imágenes de manera eficiente y eficaz. Para cumplir este objetivo, el extractor de características debe cumplir dos aspectos importantes:

- El proceso de extracción de características debe ser repetitivo y preciso, de modo tal que las mismas características sean extraídas en distintas imágenes conteniendo el mismo objeto.

- Las características deben ser distintivas, o sea, que las diferentes características extraídas puedan distinguirse unas de otras.

A su vez, se requiere también un número suficiente de características que cubra el objeto completo de modo tal que pueda ser reconocido aún bajo oclusión parcial.

2.2. SIFT

Uno de los métodos de características locales más popular y ampliamente utilizado es el SIFT [19]. Sus descriptores han mostrado mejores resultados que otros descriptores locales [20]. Este método trata de detectar puntos claves que sean similares en cada una de las imágenes y describir estos puntos a través de un vector el cual es independiente del tamaño de la imagen y su orientación. Por lo tanto, puntos claves correspondientes en diferentes vistas de un mismo objeto deben tener similares vectores de descripción. De ser satisfactorio este proceso, usando un simple algoritmo para comparar dichos descriptores extraídos de cada imagen, se podrán obtener los puntos claves correspondientes en cada imagen. En la Figura 2.1 se muestra un ejemplo típico del uso del SIFT para localizar una imagen de prueba dentro de una escena.

Lowe dividió su método en cuatro grandes etapas de cómputo:

1. Detección de puntos de interés
2. Localización de puntos claves
3. Asignación de orientación
4. Descripción de puntos claves

En las próximas secciones se describirán cada una de estas etapas.

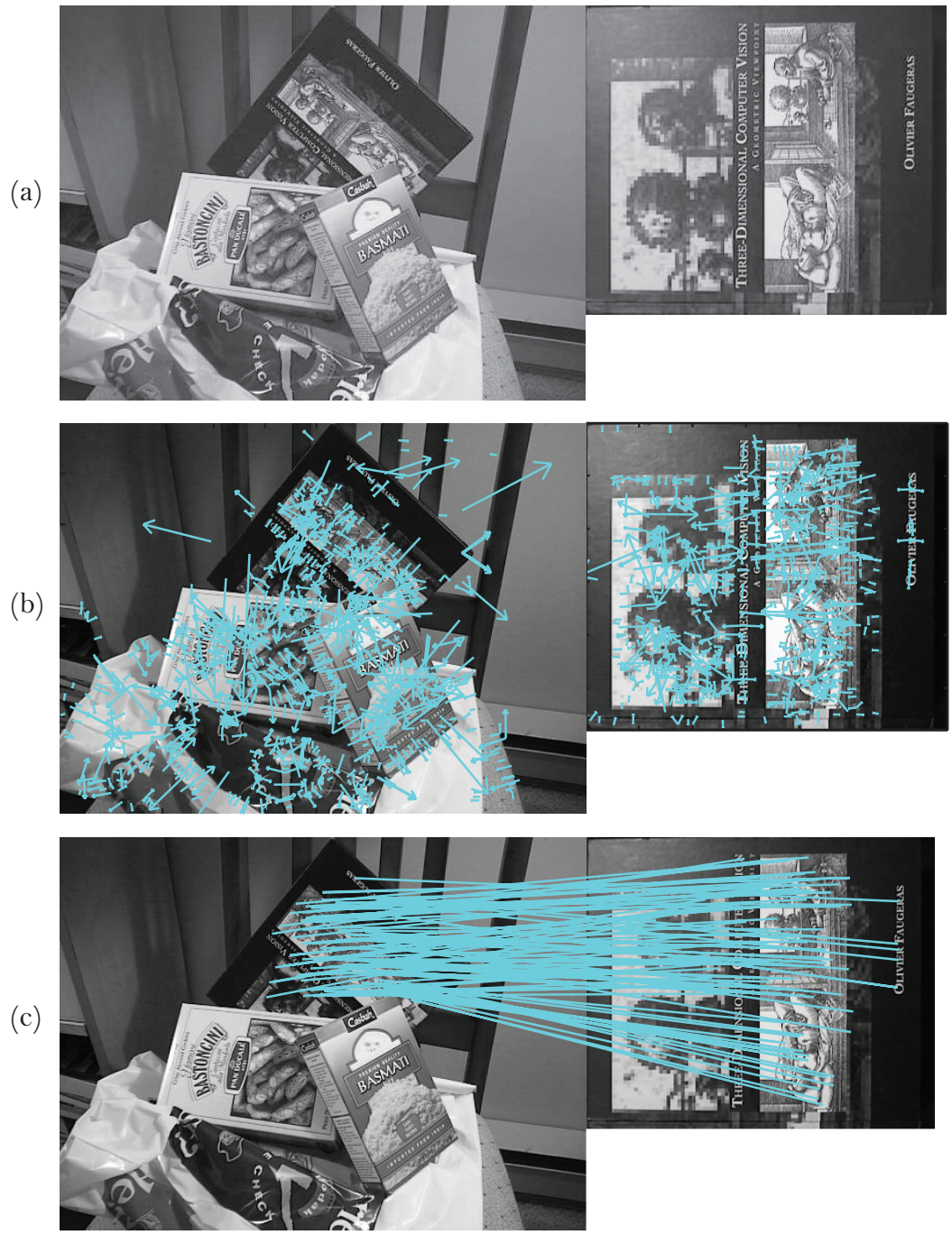


Figura 2.1: Ejemplo del uso del SIFT para localizar un objeto dentro de una escena, (a) muestra el objeto (derecha) y la escena (izquierda). De ambas imágenes son extraídos sus puntos claves y sus respectivos descriptores usando el algoritmo SIFT, estos puntos se muestran en (b). Luego, son halladas las semejanzas entre los descriptores para localizar los puntos correspondientes, los cuales se señalan en (c).

2.2.1. Detección de puntos de interés

En la primera etapa del método se busca a través de todas las escalas y localizaciones de la imagen para encontrar potenciales puntos de interés que sean invariantes a la escala y a la rotación. Para esto, la imagen es convolucionada con filtros Gaussianos a diferentes escalas y luego se hayan las diferencias entre las sucesivas imágenes filtradas. Finalmente, los máximos y mínimos en las Diferencias de Gaussianas (DoG) en las diferentes escalas son marcados como puntos de interés. Esta etapa es descrita en detalles en la Sección 2.4.

2.2.2. Localización de puntos claves

En esta etapa, para cada punto de interés se comprueba si es un candidato razonable para ser seleccionado como punto clave. Esta prueba es importante ya que algunos puntos podrían no presentar buen contraste. Para determinar esto, cada pico es modelado como una función cuadrática de 3 dimensiones, la cual es descrita en detalle en [19]. En la Figura 2.2 se muestran algunos ejemplos de picos que indican candidatos razonables y no razonables.

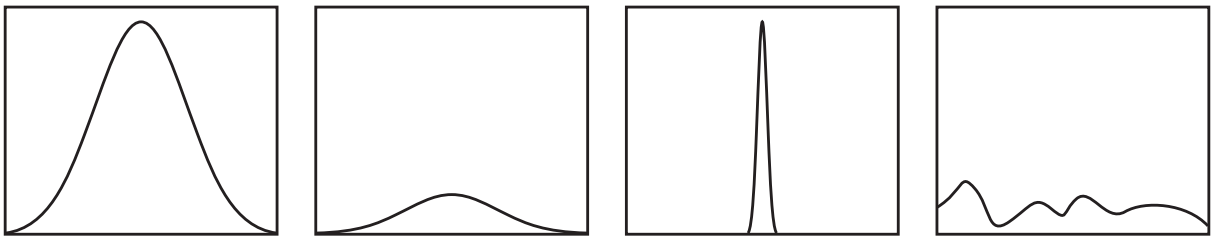
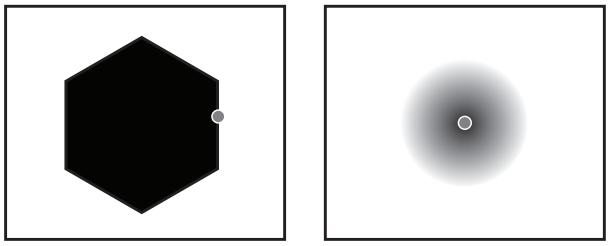


Figura 2.2: (a) Muestra un ejemplo de un pico adecuado en el espacio-escala, sin embargo (b), (c) y (d) muestran ejemplos de picos no adecuados.

También, es esencial que un punto clave no se encuentre sobre un borde de la imagen. Sobre un borde un pico está bien definido en una dirección pero no en la dirección perpendicular. Por lo tanto, cualquier ruido en la imagen podría provocar que el punto se mueva a lo largo del borde (Ver Figura 2.3). Es posible determinar los picos que se encuentran sobre bordes calculando la matriz Hessiana en la localización y escala del punto, lo cual es discutido en detalle en [19].



(a) (b)

Figura 2.3: Imágenes de ejemplos que muestran que un punto clave sobre un borde no es deseable. En (a) se muestra un punto clave (círculo gris) sobre un borde, cualquier ruido en la imagen podría provocar su desplazamiento a través del borde. En (b) sin embargo se muestra un punto clave mucho más deseable ya que es estable en ambas direcciones.

2.2.3. Asignación de orientación

Para lograr invarianza a la rotación, en esta etapa, se determina la orientación para cada característica extraída. Esto se lleva a cabo a partir de los gradientes de la imagen en una vecindad alrededor del punto, generándose un histograma de los ángulos del gradiente. El valor añadido a cada *bin* está dado por la magnitud del gradiente pesada por una ventana Gaussiana centrada en el punto clave. Un pico en el histograma determina la orientación de dicho punto clave. En caso de que se encuentre más de un pico significativo en el histograma, se generan varias copias del punto clave asignándole a cada uno la orientación determinada por cada pico. Por lo tanto, es posible que existan varios puntos claves en la misma posición en la imagen pero con distintas orientaciones.

2.2.4. Descripción de puntos claves

En la última etapa del SIFT se genera para cada punto clave un descriptor, que consiste en un vector normalizado de 128 dimensiones. En esta etapa se tiene como entrada una lista de puntos clave descritos cada uno en términos de localización, escala y orientación. Esto permite construir un descriptor con sistema de coordenadas locales alrededor del punto, que debe ser similar en diferentes vistas del mismo punto.

Lowe para crear su descriptor propone obtener un arreglo de 4×4 histogramas de 8 *bins*. Estos histogramas se calculan a partir de los valores de orientación y magnitud en una

región de 16×16 píxeles alrededor del punto, de modo tal que cada histograma se forma de una subregión de 4×4. La Figura 2.4 muestra los valores de orientación y magnitud del gradiente alrededor del punto. En la parte derecha se muestran los histogramas para cada subregión donde la longitud de cada flecha corresponde a la suma de magnitudes del gradiente cerca de esa orientación.

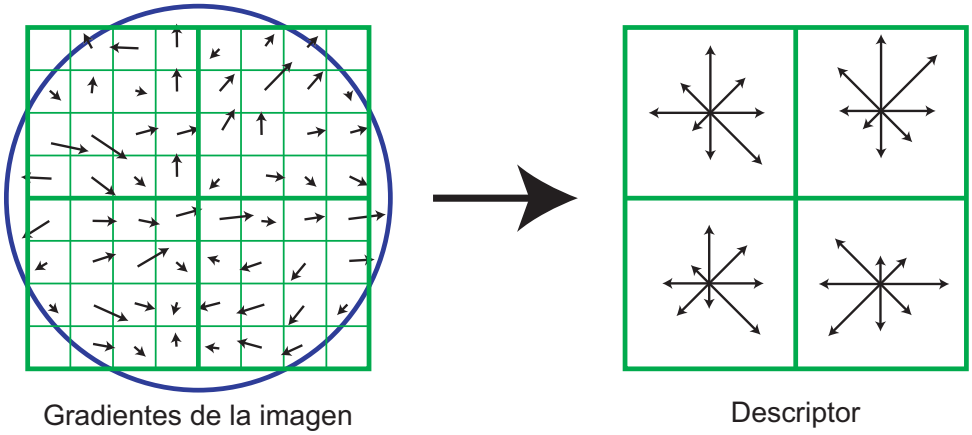


Figura 2.4: Esta figura, con propósitos de explicación, muestra un descriptor de 2×2 calculado a partir de una región de 8×8 píxeles pero Lowe propone utilizar descriptores de 4×4 en regiones de 16×16 píxeles alrededor del punto.

El descriptor consiste en un vector que es resultado de la concatenación de estos histogramas. Dado que son $4 \times 4 = 16$ histogramas de 8 *bins* cada uno, el vector resultante es de tamaño 128. Este vector es normalizado en aras de lograr invariabilidad a cambios de iluminación.

2.3. Análisis de tiempos de ejecución

Con el fin de lograr su invariabilidad ante cambios de escala y rotación, y como resultado de complejos e iterativos procesos, la extracción de características SIFT resulta una tarea de alto costo computacional.

La Tabla 2.1 muestra los tiempos de ejecución para cada una de las etapas del SIFT descritas en la sección anterior. Estos tiempos fueron registrados para una imagen de tamaño 1024×768 píxeles. Se utilizó la implementación software brindada en [31]. Los

experimentos fueron desarrollados en una PC con procesador P4 a 3.0 GHz y 2 Gb de RAM.

Tabla 2.1: Costo computacional para cada etapa del SIFT.

Etapa	Tiempo(ms)	Porcentage
(1) Detección de puntos de interés	1391	44.83 %
(2) Localización de puntos claves	97	3.13 %
(3) Asignación de orientación	341	10.99 %
(4) Descripción de puntos claves	1274	41.05 %
(*) Todas las etapas	3103	100.0 %

Como se puede apreciar en la Tabla 2.1 el tiempo total de ejecución fue de 3 segundos. La etapa de detección de puntos de interés resultó ser la más costosa ocupando casi el 45 % del procesamiento total. El alto costo computacional de esta etapa se debe a la gran cantidad de convoluciones que son producidas para generar el espacio-escala de diferencias Gaussianas, resultando en un alto número de operaciones de multiplicación-acumulación (MAC) de números flotantes. La cantidad de operaciones MAC para una imagen de tamaño $M \times N$ para obtener los puntos de interés del SIFT utilizando O octavas y S escalas está dada por:

$$\omega = \sum_{i=0}^{O-1} \frac{MN}{4^i} k^2 S,$$

donde k es el ancho del kernel de convolución Gaussiano.

Por ejemplo, para la configuración usada para medir los tiempos anteriores ($M = 1024$, $N = 768$, $O = 4$, $S = 6$ y $k = 7$) la cantidad de operaciones MAC es 307 077 120.

La etapa de descripción de los puntos claves resultó ser la segunda de mayor costo computacional, con más del 40 % del procesamiento total. En esta etapa del algoritmo, como se describió en la Sección 2.2.4, para cada punto clave, se genera un descriptor a partir de la orientación y magnitud del gradiente de sus vecinos. El cálculo de la orientación del gradiente implica operaciones trigonométricas. En hardware, para lograr un resultado por ciclo de reloj, con este tipo de operaciones se necesita gran cantidad de área del dispositivo. Otras soluciones que ocupan menos área del dispositivo, toman varios ciclos de reloj[30].

Las etapas de detección de puntos candidatos y de descripción de los puntos claves tienen costos computacionales muy similares, pero la primera presenta mayor grado de paralelismo y potencial para una aceleración mediante hardware. Por esos motivos, en este trabajo, con el objetivo de obtener la mayor aceleración posible del algoritmo SIFT mediante la aceleración de una de sus partes, se seleccionó la etapa de detección de puntos de interés.

2.4. Detección de puntos de interés del SIFT

En esta tesis se presenta una arquitectura hardware para la etapa de detección de puntos de interés del algoritmo SIFT. En esta sección se describe en detalle el funcionamiento de dicha etapa del algoritmo y sus bases teóricas.

En aras de detectar puntos de interés invariantes a la escala, Lowe propuso usar los extremos del espacio-escala en la función de diferencias de Gaussianas (DoG) convolucionada con la imagen. Para obtener la DoG es necesario realizar varias convoluciones con Gaussianas, lo que representa un alto costo computacional.

Para una imagen $I(x, y)$, los puntos claves del SIFT son obtenidos a partir de su espacio-escala Gaussiano, $L(x, y, \sigma)$, el que se obtiene a partir de la convolución de $I(x, y)$ con una Gaussiana de escala variable:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y),$$

donde $*$ es el operador de convolución en x y y, y $G(x, y, \sigma)$ es el filtro o ventana Gaussiana dado por:

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

Este espacio-escala se construye generando una serie de imágenes filtradas a valores discretos de σ, donde su dominio es dividido en intervalos logarítmicos organizados en O octavas y cada octava es luego dividida en S subniveles. El valor de σ para una octava o y un subnivel s está dado por:

$$\sigma(o, s) = \sigma_0 2^{o+s/S}, \quad o \in o_{\text{mín}} + [0, ..., O-1], \quad s \in [0, ..., S-1],$$

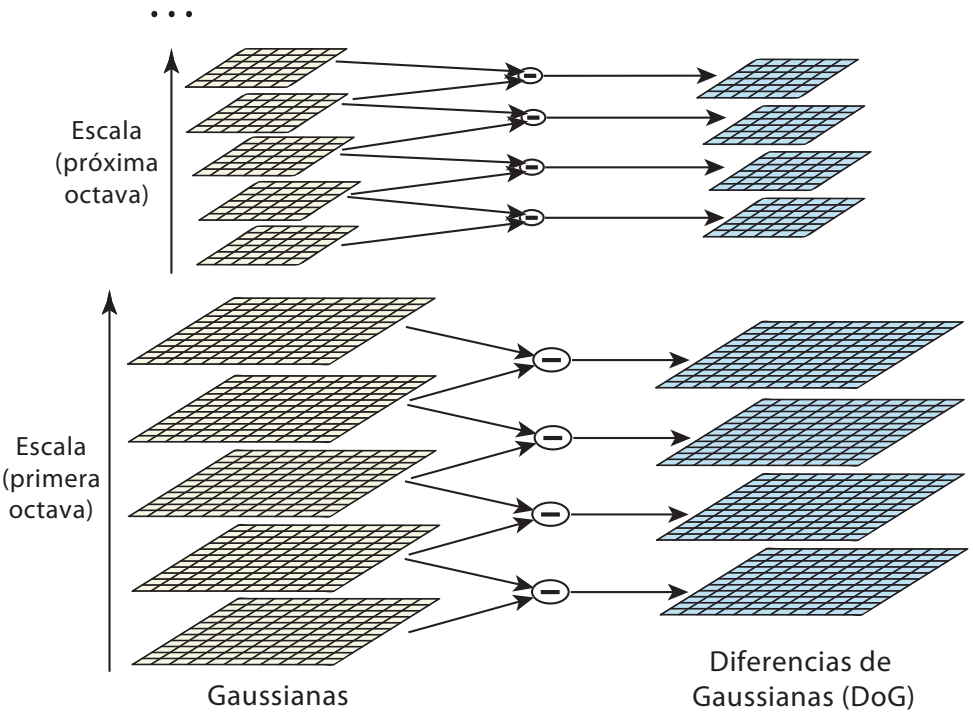


Figura 2.5: Para cada octava, la imagen inicial es repetidamente convolucionada con Gaussianas para producir el conjunto de imágenes del espacio-escala como se muestra a la izquierda. Las imágenes adyacentes son substraídas para producir las imágenes de diferencias de Gaussianas mostradas a la derecha. Luego de cada octava, la imagen Gaussiana es escalada a la mitad y el proceso es repetido.

donde σ_0 es la escala base, por ejemplo σ_0=1.6. Luego de calculada cada octava, la imagen pasada hacia la próxima octava es sub-escalada por un factor de 2.

Con el fin de detectar la posición de puntos claves estables en el espacio-escala, Lowe propuso usar los extremos del espacio-escala de diferencias Gaussianas, $D(x, y, o, s)$, calculado a partir de la diferencia de imágenes de escalas adyacentes:

$$D(x, y, o, s) = L(x, y, \sigma(o, s + 1)) - L(x, y, \sigma(o, s)).$$

En la Figura 2.5 se describe este proceso.

En aras de detectar los máximos y mínimos locales en $D(x, y, \sigma)$, cada píxel en las imágenes DoG es comparado con sus ocho vecinos en la misma imagen, más los nueve vecinos correspondientes en las escalas adyacentes (ver Figura 2.6). Si el valor del píxel es mayor o menor que sus 26 vecinos es seleccionado como un punto candidato.

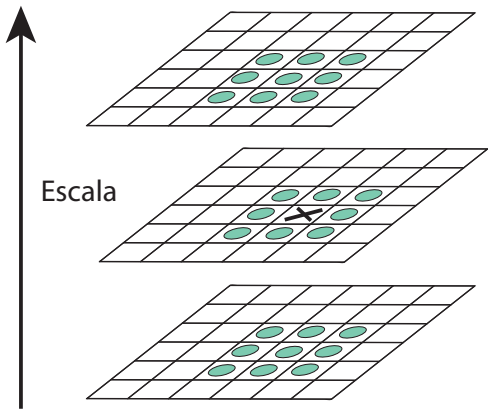


Figura 2.6: Un píxel (marcado con X) es seleccionado como punto de interés si es un máximo o un mínimo respecto a sus 26 vecinos (marcados con círculos) en su escala y en las escalas adyacentes.

$Im(x, y)$: Imagen de entrada

```
1  foreach octava o do
       // construir espacio-escala
2      foreach escala s do
3          L(x, y, o, s) = I(x, y) * G(x, y, σ(o, s));
4          I(x, y) = L(x, y, o, s);
5      end
       // obtener espacio-scala DoG
6      foreach escala s, 1 ≤ s ≤ S − 1 do
7          D(x, y, o, s) = L(x, y, o, s) − L(x, y, o, s − 1);
8      end
       // encontrar extremos locales en el espacio-escala DoG
9      foreach escala s, 2 ≤ s ≤ S − 2 do
10         foreach píxel (x, y) en D(x, y, o, s) do
11             if (x, y) es un extremo local then
12                 (x, y) es un punto de interés;
13             end
14         end
15     end
16     I(x, y) =HalveSize(L(x, y, o, S − 2));
17 end
```

Algoritmo 2.1: Detección de puntos de interés del SIFT.

Los pasos propuestos por Lowe para detectar los extremos en el espacio-escala para una imagen $I(x, y)$ son resumidos en el Algoritmo 2.1.

2.5. Conclusiones del capítulo

En los últimos años, las características locales han demostrado ser muy efectivas en la búsqueda de características o rasgos correspondientes entre diferentes vistas de un escenario. Estas han demostrado considerables resultados en una gran variedad de aplicaciones como reconocimiento de objetos, recuperación de imágenes, navegación de robots, construcción de panoramas, reconocimiento de rostros, etcétera.

La idea tradicional de estos métodos es primero detectar estructuras o puntos significativos en la imagen y obtener una descripción discriminante de estas estructuras a partir de sus alrededores, la cual será utilizada luego para la comparación usando una medida de similitud entre estos descriptores. Un detector de puntos de interés es diseñado para encontrar el mismo punto en diferentes imágenes incluso si el punto está en distintas posiciones y escalas.

El SIFT es un método de características locales que ha recibido mucha atención en los últimos años. Las características extraídas por el SIFT son en gran medida invariantes a la escala, rotación, cambios de iluminación, ruido y pequeños cambios de pose o perspectiva. Los descriptores basados en SIFT han mostrado mejores resultados que otros descriptores locales. Al mismo tiempo, en aras de lograr su robustez y como resultado de complejos procesos, la extracción de las características SIFT es una tarea de alto costo computacional. Existen varios escenarios (e.g. reconocimiento de objetos en línea) que requieren que estas características locales sean extraídas y comparadas en tiempo real e incluso usando imágenes de alta resolución. De ahí que en los últimos años se han evidenciado varios intentos de acelerar el SIFT. En el Capítulo 4 se discuten algunos trabajos reportados en la literatura que comparten el mismo objetivo de acelerar la ejecución del SIFT, algunos usando GPUs (del inglés, *Graphics Processing Unit*) o modificaciones del algoritmo, y otros mediante el uso de FPGAs.

Capítulo 3

Diseño de algoritmos sobre FPGAs

En el mundo de las computadoras y la electrónica, se está acostumbrado a dos maneras de realizar tareas de cómputo: mediante hardware o software. El hardware, como por ejemplo los dispositivos ASICs (*Application-Specific Integrated Circuits*), brinda recursos altamente optimizados para realizar tareas a gran velocidad, pero éste está permanentemente configurado para realizar una única tarea e implica un alto costo de desarrollo. Por otro lado, el software brinda la flexibilidad de poder modificar las tareas y de llevar a cabo varias de ellas. Sin embargo, es varios órdenes de magnitud más lento en ciertas aplicaciones que las implementaciones ASICs en términos de desempeño, eficiencia en el uso de área de silicio y energía.

Los FPGAs son dispositivos revolucionarios que combinan los beneficios del hardware y el software [13]. Los FPGAs también implementan circuitos, brindando grandes beneficios en cuanto a energía, área y desempeño en comparación con el software. Además, pueden ser reconfigurados de manera simple y barata para implementar un amplio rango de tareas. Estos sistemas pueden ser cientos de veces más rápidos que los basados en microprocesadores y a diferencia de los ASICs, estos no están permanentemente programados luego del proceso de fabricación. Esto significa que un sistema basado en FPGA puede ser programado y reprogramado varias veces.

Los FPGAs poseen casi todos los beneficios del software en cuanto a la flexibilidad y los modelos de desarrollo, además de casi todos los beneficios en cuanto a eficiencia que provee el hardware, pero todo esto implica algunas desventajas.

Comparado con un microprocesador, estos dispositivos pueden llegar a ser varios órdenes de magnitud más rápidos y más eficientes en el uso de energía, pero crear programas o diseños eficientes para estos dispositivos es mucho más complejo. De manera general, los FPGAs son útiles en operaciones que procesan grandes flujos de datos, como procesamiento de señales, en redes, etc.

Comparado con los ASICs, estos dispositivos pueden ser de 5 a 25 veces peores en términos de área, retardo y desempeño. Sin embargo, mientras un diseño de ASIC puede tomar de meses a años de desarrollo con un alto costo financiero, un diseño para FPGA puede solamente tomar días con costos muy inferiores.

Para sistemas que no requieran del mayor rendimiento o del más bajo consumo de energía que se pueda alcanzar, la simplicidad del proceso de desarrollo y la posibilidad de corregir fallas y agregar funcionalidades que brindan los FPGAs, hacen de ellos una alternativa altamente competitiva. Por lo tanto, para muchas tareas los FPGAs son la decisión ideal.

3.1. FPGAs

El concepto original de FPGA fue desarrollado por Ross Freeman, uno de los fundadores de Xilinx[1], en 1984. La idea de Ross Freeman fue que en vez de usar un procesador genérico y escribir software para que corriese sobre él, valdría la pena personalizar chips electrónicos para realizar tareas específicas programando directamente sobre estos chips. El nombre genérico de este tipo de chip es Dispositivo Lógico Programable (PLD) de los cuales los FPGAs son un subconjunto.

Un dispositivo FPGA es un arreglo de unidades de procesamiento de *bits* cuya interconexión y funcionalidad se puede programar después de su fabricación [9].

La mayoría de los FPGAs tradicionales están formados a partir de un bloque lógico, duplicado cientos a miles de veces. Un bloque lógico es básicamente una pequeña tabla de verdad (LUT), un biestable tipo D y un multiplexor 2-a-1. La LUT es como una

[1]Una de las dos compañías líderes mundiales en la fabricación de FPGAs.

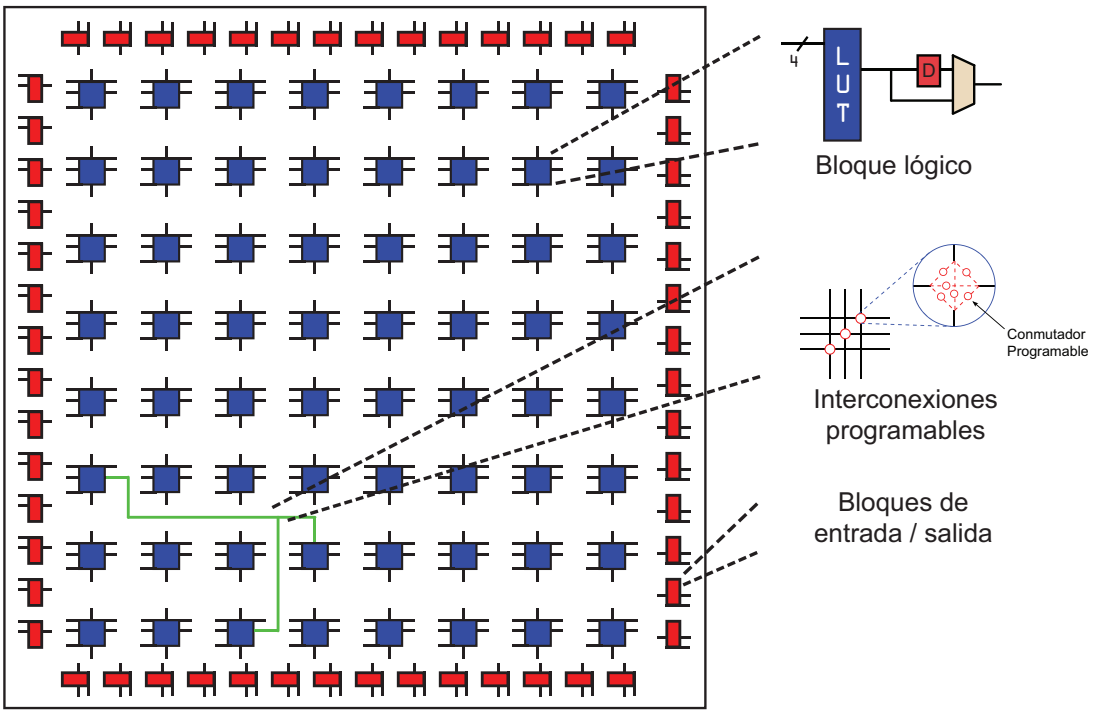


Figura 3.1: La arquitectura FPGA más común consiste en una serie de bloques lógicos, interconexiones programables y bloques de entrada / salida. En general, todos los canales de enrutamiento tienen el mismo ancho (número de interconexiones programables). Varios bloques de entrada / salida son posicionados a lo largo de una fila o una columna de la matriz. Cada bloque lógico puede ser conectado a otros a través de estos recursos de interconexión para crear funciones lógicas más complejas. Los cables de interconexión también llegan hasta los bordes del dispositivo, donde los bloques de entrada / salida son implementados y conectados a los pines del FPGA.

pequeña RAM y generalmente en estos dispositivos tiene 4 entradas, por lo que se puede implementar cualquier compuerta lógica de hasta 4 entradas. Por ejemplo, una compuerta AND con 3 entradas cuyo resultado luego pasa por una compuerta OR junto con otra entrada $((A{\cdot}B{\cdot}C){+}D)$, cabría en una de estas LUT. En un FPGA cada bloque lógico puede ser conectado a otros a través de recursos de interconexión (cables alrededor de los bloques lógicos). Un bloque por sí solo no puede hacer mucho, pero varios de ellos interconectados pueden crear complejas funciones lógicas. Los cables de interconexión también llegan hasta los bordes del dispositivo, donde los bloques de entrada / salida son implementados y conectados a los pines del FPGA. En adición a los medios de interconexión de propósito general, los FPGAs tienen líneas dedicadas muy rápidas entre los bloques vecinos. El tipo

más común de este tipo de líneas son las *carry chains*, las que permiten crear funciones aritméticas (como contadores y sumadores) eficientemente (mínima cantidad de bloques y alta velocidad en las operaciones) [9]. Un diagrama de esta estructura se muestra en la Figura 3.1.

3.2. FPGAs y su relación con el procesamiento paralelo

La mayoría de los procesadores de propósito general (GPPs) cuentan con relojes hasta 50 veces más rápidos que los de un FPGA. La diferencia radica en que un GPP convencional sólo puede procesar a lo sumo una instrucción por ciclo de reloj. Por el contrario, un FPGA puede ser configurado como varios procesadores capaces de funcionar en paralelo. Algunos FPGAs pueden contener hasta millones de bloques lógicos, por lo que para una sencilla tarea como la comparación de una cadena de texto se podría contar con un FPGA con miles de procesadores trabajando simultáneamente. Una configuración como ésta sin lugar a dudas haría despreciable una desventaja de velocidad de reloj de 50 o 100 veces. Por lo tanto, para alcanzar tales provechos es indispensable explotar el paralelismo de los algoritmos. En esta sección, dada su importancia, se analizan algunos conceptos básicos del paralelismo.

El paralelismo se puede dividir en dos grandes grupos: el paralelismo temporal y el paralelismo espacial [11]. El paralelismo temporal o *pipeline* se refiere a ejecutar una tarea como una cascada de sub-tareas, donde existe una unidad de procesamiento para realizar cada sub-tarea. Estas unidades trabajan al mismo tiempo de modo solapado, como una línea de producción de una fábrica. El paralelismo espacial hace mención a la ejecución simultánea de tareas por varias unidades de procesamiento, donde estas unidades ejecutan la misma tarea o diferentes tareas en un instante dado.

En un FPGA se pueden explotar ambos tipos de paralelismo, veamos el siguiente ejemplo de un filtro espacial: $y_i = \sum_{k=0}^{3} w_k x_{i-k}$. En la Figura 3.2 se puede observar el paralelismo espacial en los diferentes multiplicadores o sumadores, donde simultáneamen-

te una misma operación es realizada por distintas unidades de procesamiento. En este ejemplo también está presente el *pipeline*, que es una técnica muy usada en el diseño de arquitecturas para este tipo de dispositivos, logrando retornar un resultado por ciclo de reloj luego de una latencia inicial. En este ejemplo notamos como para obtener un resultado todas sus sub-tareas no son realizadas concurrentemente, sino desplazadas en el tiempo, manteniéndose todas las unidades de procesamiento en funcionamiento generando sub-resultados de distintas entradas.

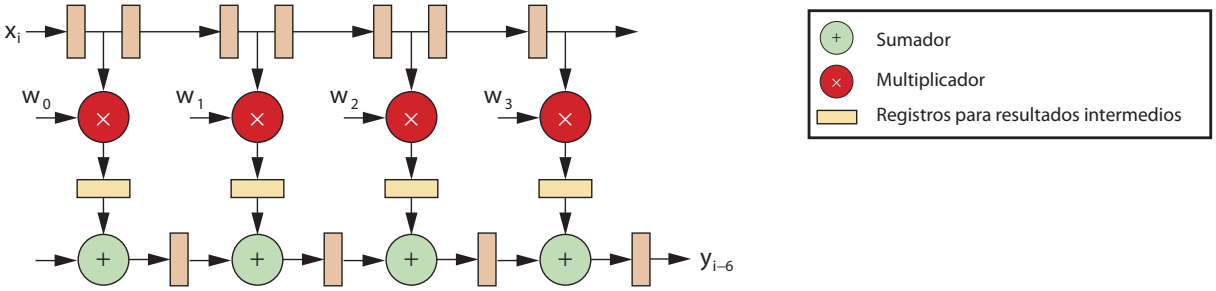


Figura 3.2: Ejemplo de filtro espacial donde se evidencian ambos tipos de paralelismo: espacial y temporal. El paralelismo espacial se puede observar en los multiplicadores y sumadores, donde simultáneamente una misma operación es realizada por distintas unidades de procesamiento. El paralelismo temporal se evidencia mediante el tunelizado, donde para obtener un resultado todas sus sub-tareas no son realizadas concurrentemente, sino desplazadas en el tiempo, manteniéndose todas las unidades de procesamiento en funcionamiento generando sub-resultados de distintas entradas, como en una línea de producción de una fábrica.

En un algoritmo paralelo una propiedad significativa es el tipo de modelo de memoria empleado. Existen fundamentalmente dos esquemas de memoria: compartida y distribuida. En un modelo de memoria compartida cada unidad de procesamiento tiene acceso directo a un espacio de memoria global, mientras que en el modelo de memoria distribuida cada procesador tiene acceso directo a su correspondiente espacio de memoria local. Esta diferencia se puede apreciar en la Figura 3.3.

Para la descomposición del problema existen dos modelos fundamentalmente, estos son el paralelismo de datos y el paralelismo de tareas. El paralelismo de datos se refiere a la ejecución de la misma operación sobre varios subconjuntos de datos al mismo tiempo, mientras que el paralelismo de tareas se refiere a la ejecución concurrente de varios proce-

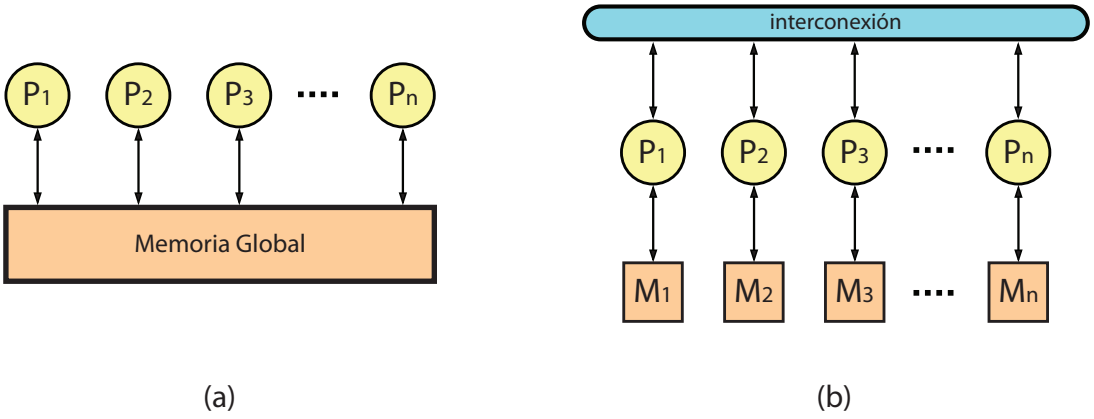


Figura 3.3: Diferencias entre modelo de (a) memoria compartida y (b) de memoria distribuida.

sos. Ambos modelos se pueden juntar para alcanzar un mayor rendimiento. Las diferencias fundamentales entre estos modelos y el procesamiento secuencial pueden ser observadas en la Figura 3.4. En los FPGAs, el uso del paralelismo de datos es el más sencillo, sólo requiere de replicar una unidad de procesamiento y darle como entrada a cada réplica una partición del conjunto de datos de entrada y tener a la salida un bloque que recolecte los resultados parciales y devuelva entonces el resultado final. El paralelismo de tareas generalmente se ve reflejado en el uso de diseños tunelizados o simplemente diferentes bloques de procesamiento realizando distintas tareas al mismo tiempo. El uso masivo del paralelismo en los FPGAs generalmente está acotado por el área del dispositivo ocupada, el tamaño del bus de datos del mismo y el consumo de corriente del sistema.

En la Figura 3.4 (a) se muestra un flujo de procesamiento secuencial donde las operaciones son ejecutadas una a la vez sobre sus datos correspondientes. En la Figura 3.4 (b) se muestran estas mismas operaciones usando paralelismo de tareas, donde, asumiendo que existen 3 unidades de procesamiento, las 3 operaciones son realizadas al mismo tiempo sobre su conjunto de datos correspondiente. En la Figura 3.4 (c) se muestra la operación Op_1 siendo ejecutada usando paralelismo de datos donde cada procesador aplica la misma operación sobre una partición del conjunto de datos.

Para evaluar la calidad de un sistema paralelo se utilizan las medidas: aceleración, eficiencia y escalabilidad. Para calcular la aceleración (Sp) tomemos t_1 como el tiempo

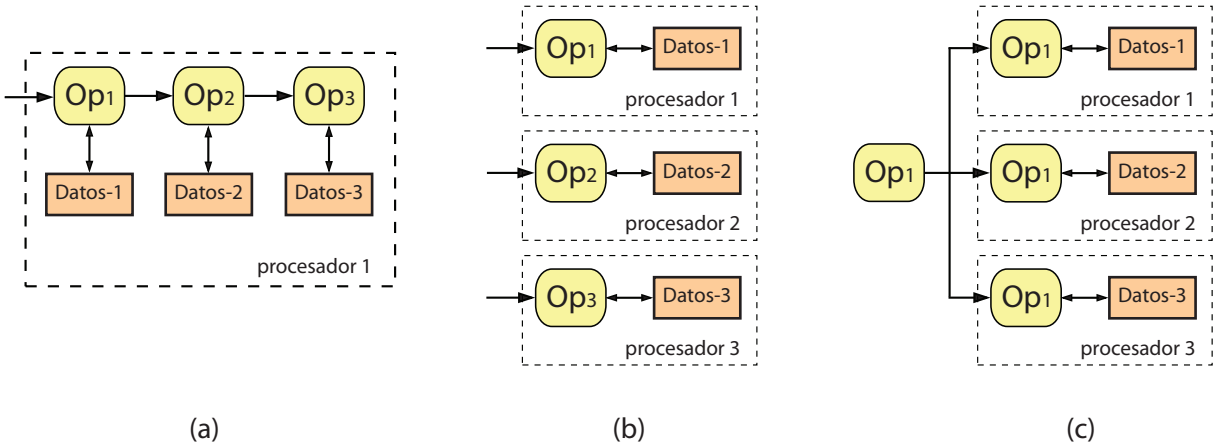


Figura 3.4: Diferencias entre (a) procesamiento secuencial, donde las operaciones son ejecutadas una a la vez, (b) paralelismo de tareas, donde las operaciones son realizadas al mismo tiempo y (c) paralelismo de datos, donde una tarea es realizada por distintos procesadores sobre distintas particiones de los datos.

total que se toma el algoritmo secuencial más rápido en un sistema de un solo procesador y t_p como el tiempo que se toma el algoritmo paralelo corriendo en una máquina paralela de p procesadores. Sp es la relación entre estos tiempos: $Sp = t_1/t_p$. Idealmente, un sistema paralelo debiera tener Sp lineal. Generalmente, para calcular la aceleración de un sistema FPGA se usa la relación entre el tiempo que se toma el algoritmo secuencial más rápido en un sistema de un solo procesador y el tiempo que se toma el sistema FPGA.

Nótese que Sp solo mide la efectividad en la explotación del paralelismo, sin tomar en cuenta la eficiencia de este proceso. Esta eficiencia (Ef), en un sistema de p procesadores, está dada por: $Ef = Sp/p$. En un sistema sobre FPGA esta medida de la forma anterior no cobra sentido, midiéndose entonces la relación entre la aceleración alcanzada y el área del dispositivo empleada.

La medida de escalabilidad (Sc) evalúa el comportamiento del sistema con respecto a un aumento en el tamaño del dispositivo y en el tamaño del problema a resolver.

3.3. Conclusiones del capítulo

Los FPGAs son dispositivos semiconductores que contienen bloques de lógica cuya interconexión y funcionalidad se puede programar después de su fabricación. Esto nos

permite su utilización en la construcción de circuitos a la medida, desde algunos tan sencillos como compuertas lógicas hasta complejos sistemas combinacionales.

El uso de los FPGAs para realizar tareas de cómputo se encuentra en un estado intermedio entre el uso del software o el hardware. Al implementar circuitos, brindan grandes beneficios en cuanto a energía, área y desempeño en comparación con el software, además de poder ser configurados, después de su fabricación, de manera simple para implementar un amplio rango de tareas.

Las aplicaciones de FPGAs incluyen procesamiento de señales digitales, sistemas aeroespaciales y de defensa, creación de prototipos de ASIC, imágenes médicas, visión por computadoras, reconocimiento de voz, criptografía, bioinformática y una gama cada vez mayor de otras áreas. Su popularidad y amplio uso en estas áreas se debe principalmente al empleo del paralelismo a gran escala que ofrece su arquitectura. Estos dispositivos permiten explotar el paralelismo tanto a nivel de instrucciones, como de datos.

Los elementos antes mencionados, hacen de estos dispositivos una buena elección a la hora de acelerar tareas de análisis de imágenes y extracción de características como el SIFT, donde se tienen altos grados de paralelismo y grandes flujos de datos. Este trabajo tiene como objetivo diseñar e implementar una arquitectura hardware para FPGAs que desarrolle la etapa de detección de puntos de interés del algoritmo SIFT. Algunos trabajos reportados en la literatura con este mismo fin son discutidos en el próximo capítulo.

Capítulo 4

Aceleración del SIFT

4.1. Introducción

En los últimos años, como resultado de la popularidad alcanzada por el SIFT como método de características locales, y dado que su alto costo computacional es una de sus principales desventajas que lo hace no viable para muchas aplicaciones en línea o en tiempo real, la comunidad científica ha dedicado cierta atención a obtener implementaciones más rápidas de dicho algoritmo. Una de las principales vías para esto ha sido el uso de unidades de procesado de gráficos (GPU, del inglés, *Graphics Processing Unit*). Ejemplos de trabajos de este tipo son [27][28][17][15]. Algunos trabajos en la literatura también se han enfocado en acelerar dicho algoritmo pero mediante aproximaciones o modificaciones del mismo en software. Los ejemplos más significativos son [16][12][2]. También, debido al amplio uso y resultados positivos de los FPGAs como medio para acelerar diversas tareas de cómputo, varios investigadores han comenzado a enfocarse en obtener sistemas basados en FPGAs para la extracción en tiempo real de características SIFT. Los principales trabajos que usan este tipo de dispositivos para acelerar el SIFT son [26][22][7][3][23]. En este capítulo, debido a que resultan de mayor interés para el objetivo de esta tesis, se discuten cada uno estos trabajos, resaltando sus ventajas y desventajas, así como analizando el tipo de arquitectura hardware que es propuesta en cada uno de ellos.

4.2. Sistemas basados en FPGAs para la aceleración del SIFT

El primer trabajo reportado en la literatura en el área de extracción de características invariantes a la escala y a la rotación usando FPGA fue el trabajo de Se y colaboradores en el 2004 [26]. En este trabajo, con el objetivo de acelerar el SIFT con respecto a implementaciones software, se presenta una implementación en FPGA del algoritmo usando punto fijo[1]. Esta implementación fue desarrollada basada en una implementación software que empleaba punto flotante. Los autores mencionan además, que varias de las rutinas de la versión software fueron modificadas para hacer la implementación hardware más eficiente. Para poder implementar la mayor parte del algoritmo se usó la herramienta de alto nivel Xilin System Generator, ya que los autores plantean que usar herramientas de descripción de hardware de bajo nivel como VHDL o Verilog sería muy costoso en cuanto a tiempo de desarrollo. No obstante, VHDL fue usado para implementar procesos de bajo nivel como el acceso directo a memoria (DMA, del inglés, *Direct Memory Access*) y otras rutinas de acceso a memoria. En este trabajo fue utilizado un FPGA Virtex II de Xilinx. El tiempo de ejecución del SIFT para una imagen de 640×480 fue reducido a 60 ms en comparación con 600 ms requeridos por un procesador Pentium III a 700 MHz. En este trabajo solo se brindan los elementos antes mencionados, no se dan ningún tipo de detalles acerca de las modificaciones realizadas al algoritmo, ni especificaciones de la arquitectura. Tampoco se ofrecen detalles en cuanto al uso de los recursos del FPGA o consumo de potencia. No se realiza ningún análisis de la exactitud de la implementación hardware usando punto fijo con respecto a la implementación software.

En [22] se presenta una implementación parcial del SIFT para la calibración estéreo en línea. Son implementadas algunas de las partes más costosas del SIFT: la generación del espacio-escala de diferencias Gaussianas y el filtro Sobel. Estas partes del algoritmo fueron implementadas en un FPGA Virtex II de Xilinx y el resto del algoritmo se ejecuta

[1]Una representación de números de punto fijo es un tipo de datos real que tiene una cantidad fija de dígitos después del punto decimal, útil para representar los valores fraccionarios, por lo general en base 2 o base 10. Por lo general, esta representación es más eficiente en cuanto a rendimiento y uso de recursos de hardware que la de punto flotante.

en software en una computadora personal. En la Figura 4.1, se muestra un diagrama de su arquitectura para la obtención de una octava. En este trabajo los autores proponen una arquitectura tunelizada donde los bloques de convolución son conectados en cascada para reducir los errores introducidos por tener un *kernel* muy pequeño en comparación con su desviación estándar, y poder utilizar entonces un *kernel* de tamaño fijo. En la Figura 4.1, se puede apreciar que para obtener cada imagen-escala en una octava es utilizado un bloque de convolución diferente. Para la convolución se aprovecha la propiedad de separabilidad del *kernel* Gaussiano y se sustituyen las multiplicaciones usando una LUT, la manera de hacer esto es descrita en [1][10]. Esta técnica, a pesar de reemplazar las multiplicaciones, implica un compromiso entre precisión y el tamaño de la LUT, ya que éste depende del ancho del *kernel* de convolución y de la cantidad de bits utilizados para representar al mismo. Los autores plantean que su sistema logra trabajar a 60 Hz y que reducen el tiempo de extracción de características entre un 50 y un 70 %, pero no se brinda ninguna información acerca de la resolución de la imagen de entrada. En este trabajo tampoco se ofrecen detalles del uso del área del dispositivo FPGA ni se brinda un análisis de cómo la sustitución de las multiplicaciones y demás detalles de su arquitectura influyen en la exactitud del resultado final.

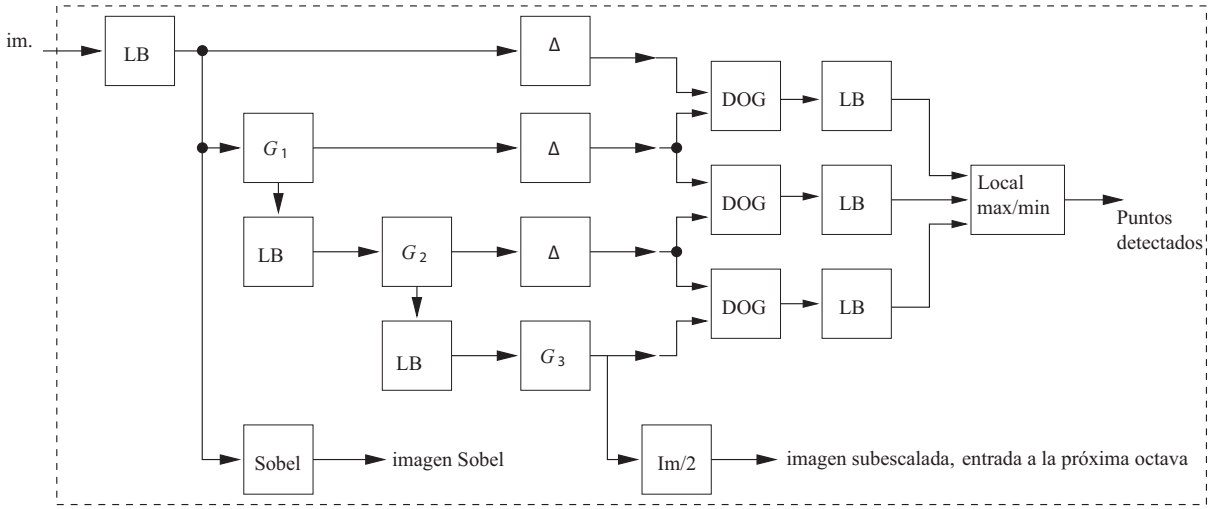


Figura 4.1: Arquitectura tunelizada para la obtención de una octava, propuesta en [22]. im. = flujo de la imagen de entrada, LB = *buffer* de línea, Δ = retraso, G_s = bloque de convolución para la escala s.

Otra implementación parcial del SIFT basada en FPGA es presentada en [7]. En este trabajo, Chati y colaboradores, presentan un co-diseño hardware / software para la detección de puntos claves del SIFT, llevando a hardware las partes de mayor grado de paralelismo. Ellos plantean usar una gran matriz o ventana de desplazamiento para lograr producir todas las escalas al mismo tiempo, sin embargo, no brindan ninguna información acerca del funcionamiento de este método. En este trabajo se mencionan algunas adecuaciones al algoritmo para su funcionamiento en hardware, pero no se brindan detalles de la arquitectura del sistema. Tampoco mencionan detalles acerca de los recursos del dispositivo utilizados ni análisis de otra índole. El dispositivo utilizado fue un FPGA Xilinx Virtex II Pro donde el sistema logra procesar imágenes de tamaño 320×240 en 0.8 ms.

La implementación en FPGA más completa del SIFT reportada en la literatura hasta la fecha es la de Bonato y colaboradores [3][4]. Esta implementación usa una estrategia de co-diseño hardware / software, donde, exceptuando la parte de generación de los descriptores, que es ejecutada en un procesador software NIOS-II, el resto de las etapas del SIFT son implementadas en hardware. La arquitectura propuesta en [3][4] está formada por tres bloques en hardware, uno para la generación del espacio-escala de diferencias Gaussianas, otro para el cálculo de la orientación y magnitud y otro para la localización de los puntos claves. El bloque de generación del espacio-escala de DoG recibe la imagen de entrada proveniente de la cámara y su resultado es enviado a los otros dos bloques de hardware. Además, dicha arquitectura, cuenta con un bloque software el que se encarga de la generación de los descriptores para cada punto clave. Los autores plantean que la generación de los descriptores es desarrollada en software ya que el tipo de cálculo que es desarrollado en esta etapa es más factible su realización en un procesador software, además, que es más fácil de implementar en software que en hardware y que le brinda mayor flexibilidad para la modificación del descriptor de acuerdo a la aplicación final. La implementación del bloque de generación del espacio-escala de DoG considera las propiedades de separabilidad y simetría del *kernel* Gaussiano. Además, ahorran cuatro multiplicadores normalizando el *kernel* de convolución de manera que siempre tome valores de 0 o 1 en primera y última

posición, evitando las multiplicaciones en estos puntos. Esto trae como inconveniente que se vean obligados a trabajar con valores de punto fijo, pues para determinados valores de σ si estos valores son normalizados de esta manera y luego aproximados a enteros, todos los elementos tendrían el mismo valor. Un diagrama de la arquitectura propuesta para la generación del espacio-escala de DoG es mostrado en la Figura 4.2. El sistema propuesto implementa 18 bloques de convolución con Gaussianas, uno para cada imagen-escala, y el mismo sigue una configuración de tres octavas y seis escalas. Otra adecuación en esta arquitectura con el objetivo de ahorrar área del dispositivo es que se representan las imágenes de DoG con 5 bits sin signo. El uso de un formato sin signo afecta la cantidad de puntos detectados, el que se reduce aproximadamente a la mitad, ya que solo se consideran los puntos que sean máximos locales, no tomándose en cuenta los mínimos. Según los autores, esta disminución en la cantidad de puntos no es considerado un problema para su aplicación de SLAM (del inglés, *Simultaneous Localization And Mapping*) donde sólo unas decenas de éstos son necesarias. Este sistema, implementado en un FPGA Altera Stratix II EP2S60F672C3, con un procesador soft NIOS-II corriendo a 100 MHz, requiere 33 ms para extraer las características SIFT de una imagen de 320×240 píxeles, donde el cuello de botella de la arquitectura se encuentra en la generación de descriptores llevada a cabo en el NIOS-II.

En [23], Qiu y colaboradores, presentan una arquitectura para la generación del espacio-escala de DoG. Este trabajo logra superar a [3] y [24] en cuanto al uso de recursos del dispositivo. Este sistema logra obtener el espacio-escala de DoG para imágenes de entrada de tamaño 320×240 en 12 ms. Para esto, aprovechan las propiedades de separabilidad del *kernel* Gaussiano, realizando la convolución separable según [24] (otro trabajo de los mismos autores). Además, se hace uso de la propiedad asociativa de la convolución, donde el resultado de una convolución puede ser equivalente a dos sucesivas convoluciones, donde la suma de los cuadrados de los radios de los *kernels* de convolución de estos últimos es igual al cuadrado del radio del primero ($R_0^2 = R_1^2 + R_2^2$). Esto les permite dividir una convolución en dos usando kernels más pequeños. Según los autores, las ventajas de usar esta técnica está dada por la posibilidad de reutilizar resultados in-

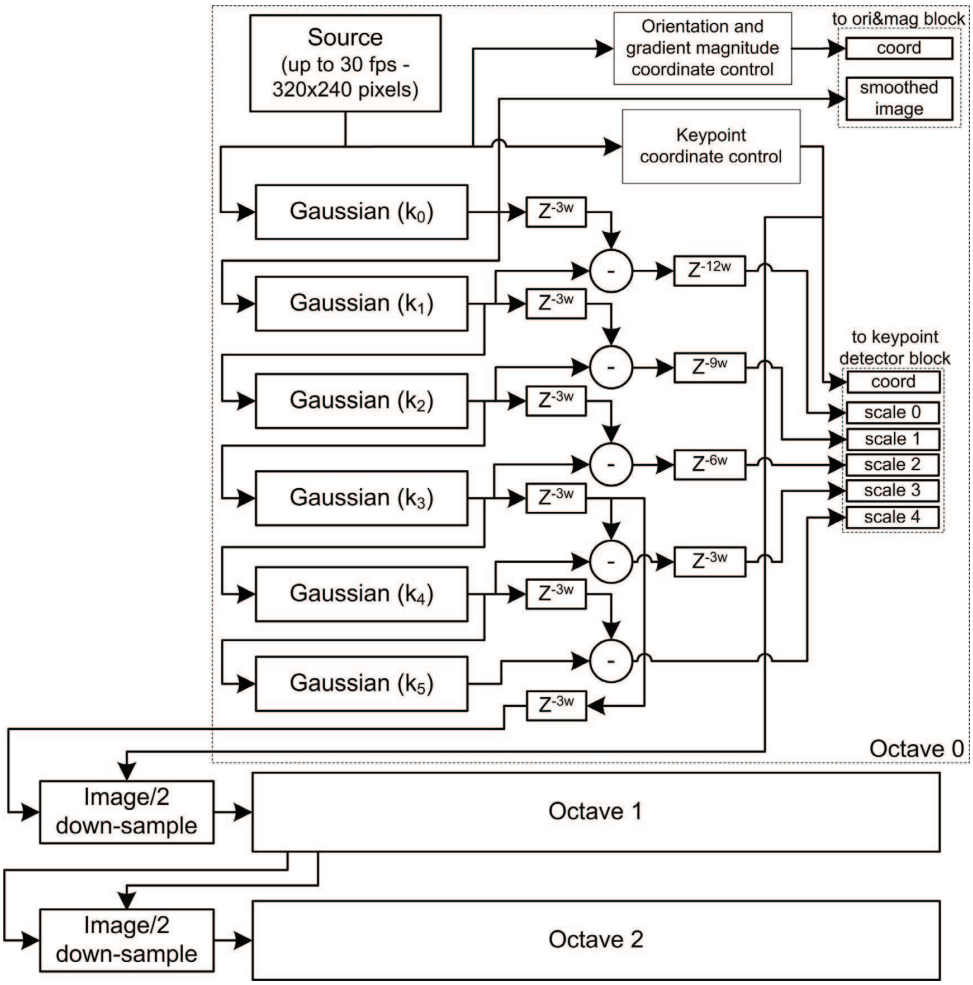


Figura 4.2: Arquitectura tunelizada para la obtención del espacio-escala de DoG, propuesta en [3][4]. Son implementados 18 bloques de convolución, uno para cada imagen-escala. El sistema propuesto tiene una configuración de tres octavas y seis escalas.

termedios, ahorro de recursos de hardware y conveniencias dadas por el orden en que se realizan las convoluciones. Plantean, además, que teóricamente, esto les brinda un ahorro de hasta el 17.8 % de costo de recursos de hardware. En esta arquitectura, los autores proponen utilizar solamente cinco bloques de convolución, en los cuales, luego de siete iteraciones, logran obtener el espacio-escala de diferencias Gaussianas para un sistema de cinco octavas y seis escalas ($O = 5, S = 6$). En la Figura 4.3 se muestra un diagrama de la arquitectura propuesta en [23]. Este esquema tiene como desventaja, que a pesar de que sólo utilizan cinco bloques de convolución (lo que implica un ahorro en el uso de área

del dispositivo), se deben realizar siete iteraciones para obtener todo el espacio-escala de DoG. En este trabajo, se plantea lograr no tan sólo mejoras en cuanto a área del FPGA con respecto a [3][4], sino además, en tiempo, aunque para esto toman en cuenta la restricción brindada por el sistema completo y no solamente la de esta etapa del algoritmo, la cual en [3][4] es más eficiente en tiempo que la arquitectura propuesta en el trabajo de Qiu y colaboradores.

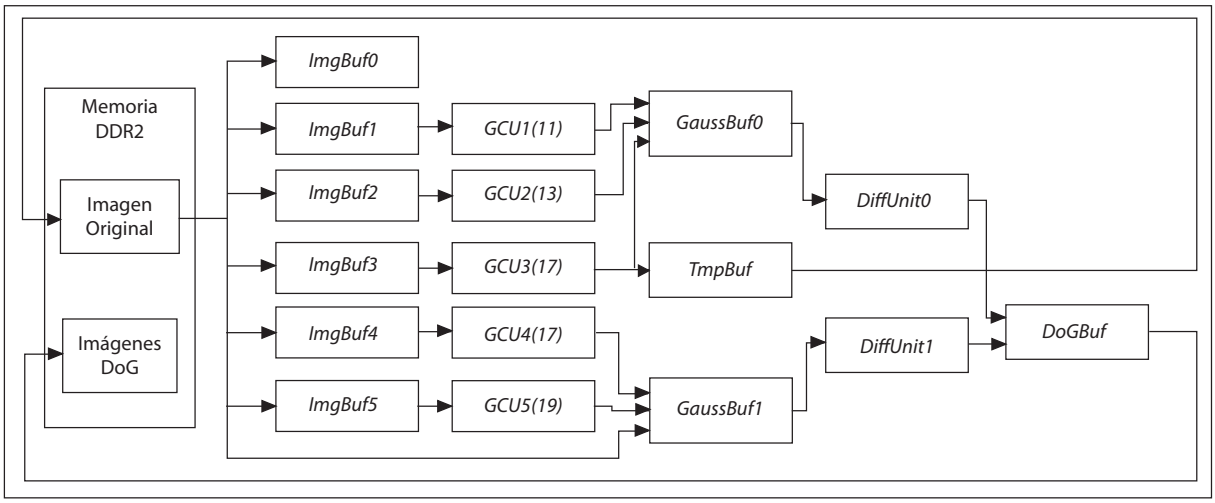


Figura 4.3: Arquitectura hardware para la obtención del espacio-escala de DoG, propuesta en [23]. Los bloques GCU representan los bloques de convolución. Los números entre paréntesis indican el ancho del *kernel* de convolución de cada bloque. El sistema propuesto tiene una configuración de cinco octavas y seis escalas.

La Tabla 4.1 muestra un resumen y comparación de los trabajos antes mencionados, así como las principales diferencias de la arquitectura propuesta en esta tesis con estos trabajos reportados en la literatura.

4.3. Conclusiones del capítulo

El algoritmo SIFT ha recibido gran aceptación y popularidad en los últimos años, esto se ha debido principalmente a su robustez ante cambios de escala, rotación, punto de vista e iluminación. Sin embargo, una de sus mayores desventajas es el alto costo computacional provocado por los complejos procesos que son llevados a cabo en aras de obtener invariabilidad ante estos cambios. Con el fin de darle solución a este problema,

varios investigadores se han dedicado a lograr implementaciones más rápidas de dicho algoritmo.

En este capítulo se discutieron los principales trabajos reportados en la literatura con el objetivo de acelerar el algoritmo SIFT usando FPGA, y específicamente los enfocados en la aceleración de la etapa de detección de puntos de interés de dicho algoritmo.

El primer trabajo se presentó en el 2004 y hasta la fecha se han obtenido avances en cuanto a ahorro de recursos del FPGA y reducciones considerables de tiempo de ejecución en comparación con implementaciones software. No obstante, todavía no se han explotado en su totalidad las posibilidades de lograr una máxima aceleración mediante hardware de dicho algoritmo, usando la mínima cantidad de recursos del dispositivo.

En este trabajo de tesis, también se presenta una arquitectura hardware con el propósito de acelerar la detección de puntos de interés del SIFT, la cual es descrita en los próximos capítulos. La principal diferencia de la arquitectura propuesta en este trabajo con las anteriores reportadas en la literatura, radica en un uso más eficiente de los recursos del FPGA mediante el entrelazado del procesamiento de las octavas, mientras se logra obtener un resultado cada dos ciclos de reloj, implicando una considerable aceleración con respecto a las implementaciones software existentes y muchas de las implementaciones hardware discutidas en este capítulo. Además, la arquitectura aquí presentada, logra mayores tasas de ahorro de recursos del FPGA mientras mayor número de octavas se procesen. Esto último implica una gran ventaja ya que el número de octavas es función de las dimensiones de la imagen, y la tendencia en la visión por computadoras es a trabajar cada vez más con imágenes de mayor resolución.

Tabla 4.1: Resumen y comparación de los trabajos relacionados.

Trabajos relacionados	**Tamaño de la imagen**	**Octavas**	**Escalas**	**Velocidad del proceso**	**Máx. frecuencia reloj**	**Potencia computacional (Mpixeles/s)**	**Registros**	**LUTs**
Se et al 2004 [26]	VGA	N/A	N/A	16 fps	N/A	N/A	N/A	N/A
Pettersson et al 2005 [22]	N/A	4	3	60 fps	54.0 MHz	N/A	N/A	N/A
Chati et al 2007 [7]	QVGA	N/A	N/A	1250 fps	N/A	N/A	N/A	N/A
Bonato et al 2008 [3]	QVGA	3	6	1940 fps	149.0 MHz	149.0	7256	15137
Qiu et al 2009 [24]	VGA	N/A	6	16 fps	82.0 MHz	5.1	6333	5825
Qiu et al 2010 [23]	QVGA	6	6	81 fps	95.0 MHz	15.3	6120	5011
Arquitectura propuesta	QVGA	3	6	900 fps	145.1 MHz	72.6	5676	5554

Capítulo 5

Algoritmo paralelo para la detección de puntos de interés del SIFT

Para realizar una tarea de cómputo determinada es común contar con diferentes vías o algoritmos. La selección final generalmente está dada por la aplicación y el dispositivo hardware a utilizar. Usualmente, el algoritmo óptimo para un FPGA difiere del algoritmo óptimo para un GPP o una computadora secuencial.

A pesar de que las especificaciones y configuraciones de sistemas en FPGA se asemejen a programas en lenguajes de alto nivel, éstos especifican hardware y no software. Debido a que el hecho de que una técnica de cómputo sea buena en software no implica necesariamente que sea buena en hardware, una reformulación del algoritmo en software puede muchas veces significar una mejora substancial en el desempeño del mismo en hardware [14].

En esta sección se describen las características del algoritmo paralelo propuesto para la detección de puntos de interés del SIFT. Este algoritmo es una reformulación del algoritmo presentado por Lowe [19] para este fin. El mismo tiene como objetivo obtener el máximo desempeño en una implementación hardware de esta etapa del algoritmo. Estas reestructuraciones están principalmente enfocadas a sacar el máximo provecho del paralelismo en este proceso, a la vez que tratan de minimizar el área del dispositivo ocupada.

5.1. Consideraciones generales del algoritmo

Con el objetivo de obtener un algoritmo que permita hacer un uso más eficiente de los recursos del FPGA, se tuvo en cuenta la potencialidad de la explotación del paralelismo de datos, de la propiedad de separabilidad del kernel Gaussiano y el entrelazado en el procesamiento de las octavas. En esta sección se detallan estos elementos que forman la base del algoritmo propuesto para la etapa de detección de puntos candidatos.

5.1.1. Explotando el paralelismo de datos

Sea I una imagen bidimensional y G una máscara de convolución de tamaño impar $k \times k$, entonces la convolución de I y G está dada por:

$$f(x,y) = \sum_{-i}^{i} \sum_{-j}^{j} I(i,j)G(x-i, y-j), \tag{5.1}$$

donde $i, j = \left\lfloor \frac{k}{2} \right\rfloor$.

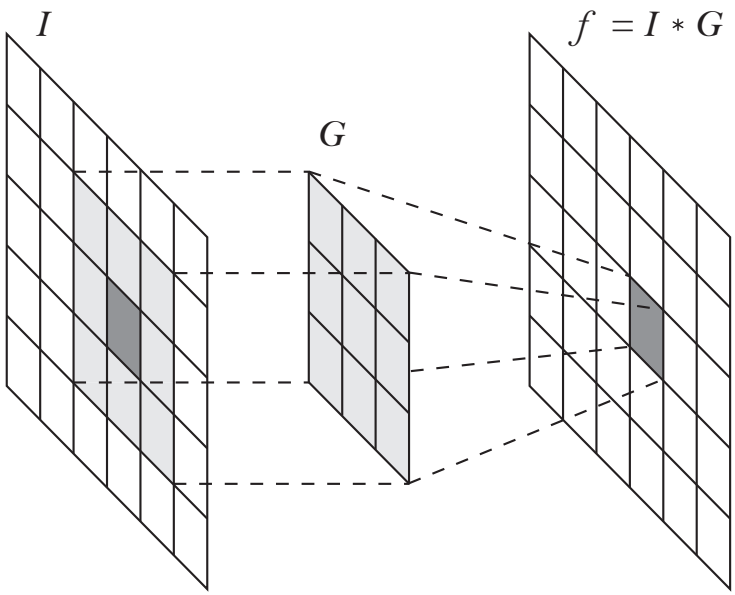


Figura 5.1: En la convolución 2D, el resultado de un píxel sólo depende de una vecindad, del mismo tamaño que la ventana de convolución, alrededor del píxel en la imagen de entrada. En esta figura para una ventana de convolución G de tamaño 3×3 el resultado sólo depende de una región de igual tamaño en la imagen de entrada I.

Como se puede observar en la Ecuación 5.1, para el cálculo de $f(x_1, y_1)$ sólo una vecindad de tamaño $k \times k$ de centro (x_1, y_1) es necesaria. Esto también se muestra de manera gráfica en la Figura 5.1. De igual manera, para determinar si un punto es un punto de interés solo se necesita una vecindad de tamaño 3×3 en la imagen de diferencia de Gaussianas y en las adyacentes.

Estas características de la convolución en 2D y de la detección de extremos en el espacio-escala brindan un alto potencial para el paralelismo de datos, específicamente del tipo SPMD (del inglés, *Single Process, Multiple Data*, que se refiere a un solo proceso, múltiples datos). Un ejemplo de uso del paralelismo del tipo SPMD en esta tarea se muestra en la Figura 5.2, donde la imagen es dividida en P particiones con un solapamiento de $k-1$ líneas y todas las particiones son procesadas simultáneamente por P procesadores distintos. Esto implica una mejora en el tiempo de procesamiento de P veces, pero también un aumento en el uso de área del dispositivo del mismo factor. Por lo tanto, se debe encontrar el equilibrio adecuado entre aceleración deseada y uso del área del dispositivo en dependencia de la aplicación.

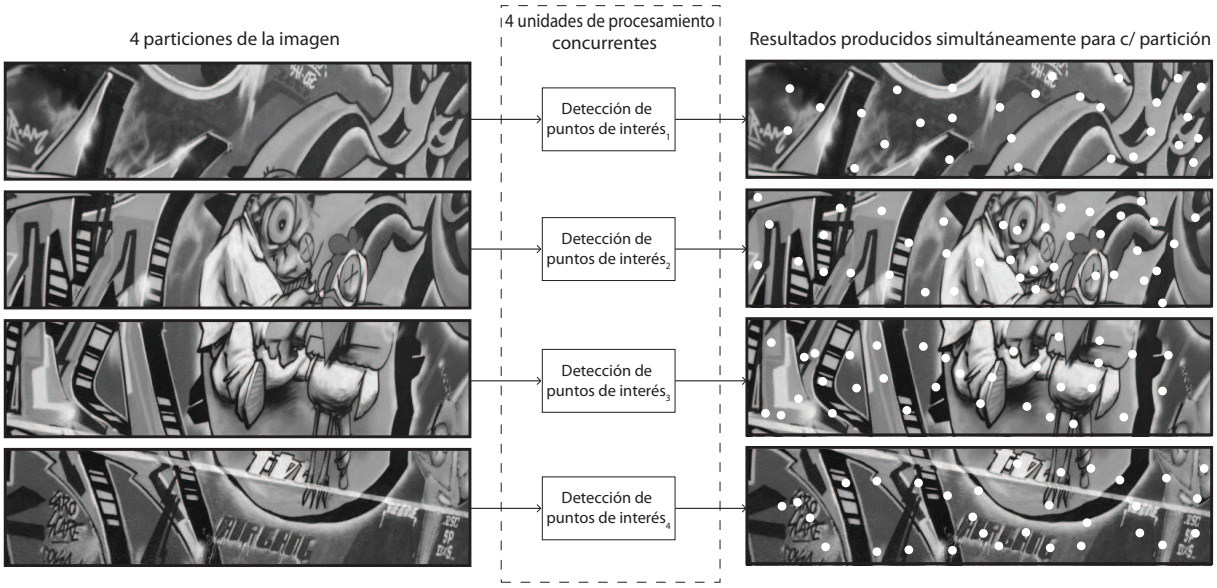


Figura 5.2: En este ejemplo se tienen cuatro particiones de la imagen, las cuales son procesadas independiente y simultáneamente por distintas unidades de procesamiento. Haciendo uso del paralelismo del tipo SPMD, en este ejemplo se logra obtener una aceleración de $4x$, aunque también un aumento del mismo factor en el uso de área del dispositivo.

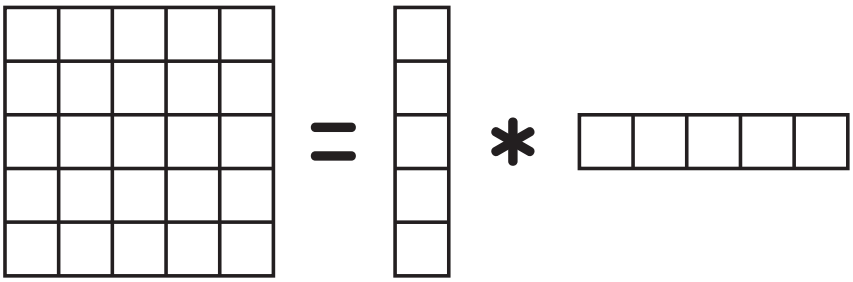


Figura 5.3: Una matriz de $M \times N$ es separable si puede ser descompuesta en dos matrices de $M \times 1$ y $1 \times N$.

5.1.2. Explotando la propiedad de separabilidad del *kernel* Gaussiano

Con el objetivo de disminuir el número de operaciones aritméticas, la propiedad de separabilidad de la Gaussiana es aprovechada. Un filtro 2D es separable si puede ser dividido en dos señales de 1D: una proyección vertical y otra horizontal (Ver Figura 5.3). El filtro Gaussiano puede ser separado del siguiente modo:

$$G(x, y, \sigma) = h(x, \sigma) * v(y, \sigma),$$

donde

$$h(x, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/2\sigma^2}, \text{y } v(y, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-y^2/2\sigma^2}.$$

Además, la convolución cumple la propiedad de asociatividad:

$$I(x, y) * (h(x, \sigma) * v(y, \sigma)) = (I(x, y) * h(x, \sigma)) * v(y, \sigma).$$

Por lo tanto, la convolución 2D de una imagen con un filtro Gaussiano puede ser llevada a cabo primero convolucionando la imagen con $h(x, \sigma)$ en la dirección horizontal y luego con $v(y, \sigma)$ en la dirección vertical o viceversa. La convolución en 1D para obtener un valor de la salida necesita k operaciones MAC (multiplicación - acumulación) en comparación con las k^2 operaciones MAC requeridas por la variante 2D. Por lo tanto, la ventaja computacional de la convolución separable sobre la no-separable es $k^2/2k$. Si se tuviera una ventana de convolución de tamaño 7×7 el uso de esta técnica representaría una reducción de la cantidad de operaciones MAC por un factor de 49/14=3.5, lo que

podría significar una reducción de hasta 3.5 veces del uso del área del dispositivo para estas operaciones.

Como desventaja, la convolución 2D separable requiere almacenamiento adicional para mantener los cálculos intermedios. Es decir, si se realiza primero la convolución 1D vertical, se deben almacenar temporalmente estos resultados con el fin de utilizarlos para la convolución horizontal posteriormente.

5.1.3. Entrelazando el procesamiento de octavas

Luego de procesar cada octava, la imagen es subescalada por un factor de dos, tomando cada segundo píxel en cada fila y en cada columna, o sea $I_o(x, y) = I_{o-1}(2x, 2y)$. Luego de escalar una imagen a la mitad, el número total de píxeles es reducido en cuatro. En hardware, para reducir la cantidad de datos, su frecuencia de muestreo es reducida por el mismo factor. Si luego de procesar cada octava la cantidad de datos es reducida por un factor de cuatro, el período de muestro τ de una octava o está dado por

$$\tau(o) = \tau_0 4^o, \tag{5.2}$$

donde τ_0 es el período de muestreo de la primera octava. Por lo tanto, luego de subescalar, hay un gran porciento de tiempo de procesamiento inactivo con respecto al período de procesamiento de la primera octava. Esta gran cantidad de tiempo de procesamiento inactivo se debe a los altos períodos de muestreo en las últimas octavas debido al pequeño tamaño de las imágenes con respecto a la imagen original. El mismo está dado por

$$\hat{i} = \tau(o) - 1,$$

y pueden ser identificados en la Figura 5.4 a) como los flancos de subida no marcados en negritas en cada una de las octavas.

Aprovechar estos intervalos de inactividad podría hacer posible el procesar las O octavas de una escala en un solo bloque de convolución. Esto se logra entrelazando el procesamiento de las O convoluciones de modo tal que para todas las octavas en un tiempo

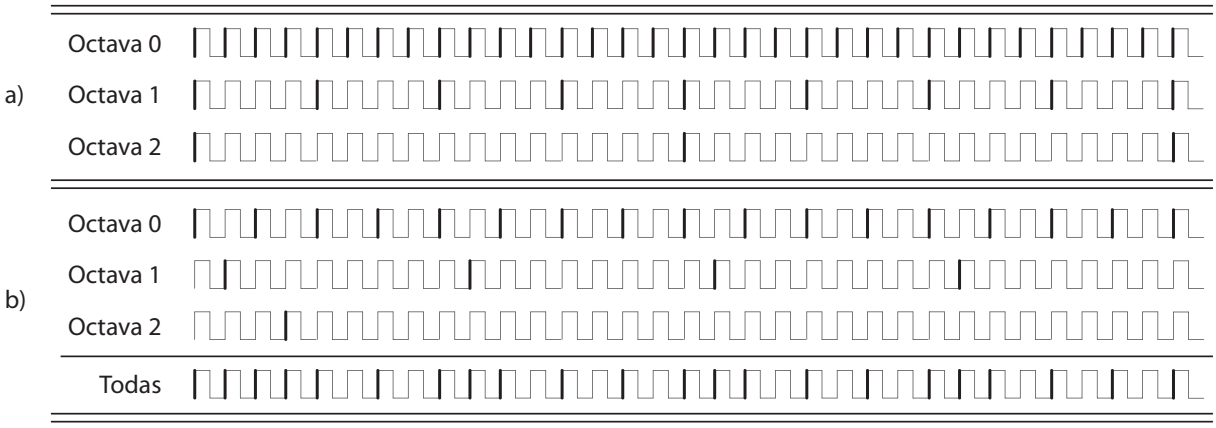


Figura 5.4: Los flancos de subida no marcados en negritas en a) indican la cantidad de tiempo de procesamiento que no es utilizado en obtener un nuevo resultado. En b) se muestra una manera de aprovechar estos tiempos, donde el período de muestreo de cada octava es duplicado, haciendo posible entrelazar el procesamiento de todas las octavas en un solo bloque de convolución.

t dado, el número de elementos procesados $p(o,t)$ cumpla que

$$p(o,t) = \left\lfloor \frac{t + \varepsilon(o)}{\tau(o)} \right\rfloor, \tag{5.3}$$

donde $\varepsilon(o)$ es la latencia de la octava o en la línea de entrelazado. Una manera de realizar este procedimiento puede ser observado en la Figura 5.4 b).

Para entrelazar el procesamiento de las O octavas se asume que el período de muestro de la primera octava es mayor o igual a dos ciclos de reloj.

El entrelazar el procesamiento de las octavas trae como inconveniente que deban agregarse funciones extras para controlar el orden de entrelazado. Además se introduce una restricción en la velocidad del sistema de la mitad de su velocidad máxima alcanzable, si se prescindiese de esta característica.

5.2. Detección de puntos de interés del SIFT

En esta sección se detalla el algoritmo propuesto para la detección de puntos de interés del SIFT. Este algoritmo es una reformulación del algoritmo original propuesto por Lowe [19]. Estas reestructuraciones tienen como objetivo sacar el máximo provecho del paralelismo en este proceso, a la vez que tratan de minimizar el área del dispositivo ocupada.

Por lo tanto, el algoritmo propuesto constituye una optimización del algoritmo original para su implementación en hardware.

Según se detalló en la Sección 5.1.1, para la detección de puntos candidatos, la imagen puede ser particionada y cada partición procesada por separado sin afectar el resultado. Para cada partición $I_p(x, y)$ de la imagen, los puntos claves del SIFT de la misma son obtenidos a partir de su espacio-escala Gaussiano, $L_p(x, y, \sigma)$, el que se obtiene a partir de la convolución de $I_p(x, y)$ con una Gaussiana de escala variable:

$$L_p(x, y, \sigma) = I_p(x, y) * G(x, y, \sigma),$$

haciendo uso de la propiedad de separabilidad del *kernel* de convolución Gaussiano y según se explicó en la Sección 5.1.2 ésta queda expresada de la siguiente forma:

$$L_p(x, y, \sigma) = (I_p(x, y) * h(x, \sigma)) * v(y, \sigma).$$

Como se detalló en la Sección 2.4, este espacio-escala se construye generando una serie de imágenes filtradas a valores discretos de σ, donde su dominio es dividido en intervalos logarítmicos organizados en O octavas y cada octava es luego dividida en S subniveles. Por lo tanto, para obtener un resultado de una posición en la imagen a una determinada escala es necesario haber obtenido el valor de esa misma posición en la escala anterior, y así sucesivamente. Gracias a que el valor de un resultado en la convolución sólo depende de una pequeña región, todas las convoluciones son desarrolladas concurrentemente, existiendo una latencia, relativamente pequeña, con respecto al tamaño de la imagen, en el dato de entrada con respecto a la escala anterior. De igual manera, las diferencias entre escalas adyacentes para formar el espacio-escala de diferencias de Gaussianas son realizadas concurrentemente a la par que se va obteniendo el espacio-escala Gaussiano.

En aras de detectar los extremos locales en el espacio-escala de DoG, cada píxel en las imágenes DoG es comparado con sus ocho vecinos en la misma imagen, más los nueve correspondientes vecinos en las escalas adyacentes. Esto implica que una misma vecindad de 3×3 en una escala determinada es usada en tres momentos, al procesar la propia escala y las adyacentes, comparándose en cada caso con valores distintos. Una manera

eficiente y equivalente para obtener los máximos y mínimos locales que permita reutilizar resultados parciales se describe a continuación.

Para todas las imágenes adyacentes se obtiene el mínimo y el máximo punto a punto:

$$Min_1(x, y, o, s) = \text{mín}\,(D(x, y, o, s), D(x, y, o, s + 1)),$$
$$Max_1(x, y, o, s) = \text{máx}\,(D(x, y, o, s), D(x, y, o, s + 1)).$$

Luego el proceso se repite sobre las imágenes obtenidas en el paso anterior:

$$Min_2(x, y, o, s) = \text{mín}\,(Min_1(x, y, o, s), Min_1(x, y, o, s + 1)),$$
$$Max_2(x, y, o, s) = \text{máx}\,(Max_1(x, y, o, s), Max_1(x, y, o, s + 1)).$$

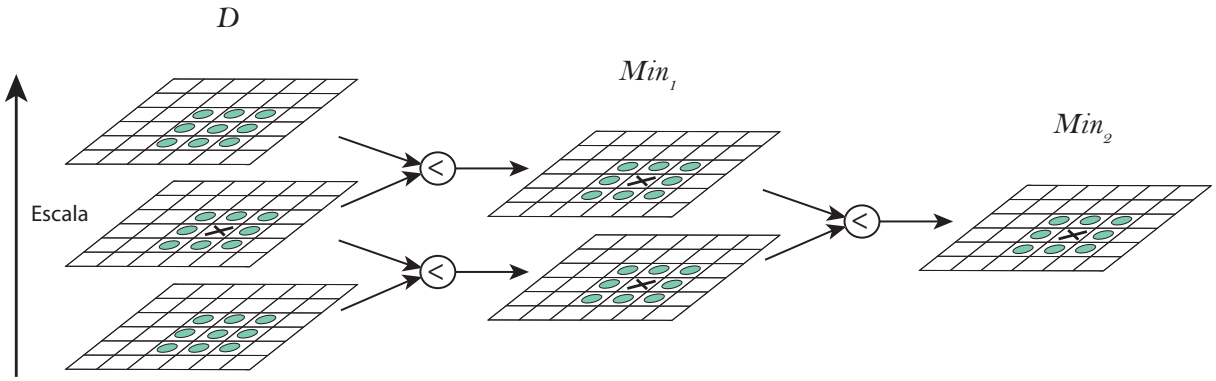


Figura 5.5: Un píxel (marcado con X en D) es seleccionado como punto de interés si es un mínimo en Min_2 respecto a sus 9 vecinos (marcados con círculos). Min_2 es el mínimo de segundo orden entre escalas adyacentes en el espacio-escala de diferencias Gaussianas. De igual manera esta figura aplica para los máximos.

Con este procedimiento se logra obtener imágenes indicando los valores mínimos y máximos a lo largo de tres imágenes adyacentes. Para comprobar si un píxel es un punto de interés es necesario comprobar que sea un máximo o un mínimo local en $Min_2(x, y, o, s)$ o en $Max_2(x, y, o, s)$ respectivamente. La Figura 5.5 muestra un diagrama para este procedimiento. Además, se debe comprobar que su valor sea igual al píxel correspondiente en la DoG, y que a pesar de ser un extremo local no es igual a su correspondiente en cualquiera de las escalas adyacentes. Para esto a $Min_2(x, y, o, s)$ y $Max_2(x, y, o, s)$ se les agrega una bandera $\beta(x, y, o, s)$ indicando estos fenómenos.

El método paralelo propuesto para la detección de puntos de interés del SIFT recientemente descrito es resumido en el Algoritmo 5.1.

```
1  forall particiones p de la imagen in parallel
2      foreach ciclo de reloj clk do
3          o = SelectOctaveInInterleaving(clk);
4          do in parallel
               // construir espacio-escala DoG
5              forall escalas s, 2 ≤ s ≤ S in parallel
6                  L_p(x, y, o, s) = L_p(x, y, o, s − 1) ∗ G(x, y, σ(o, s));
7                  D_p(x, y, o, s − 1) = L_p(x, y, o, s) − L_p(x, y, o, s − 1);
8              end
               // detectar extremos en el espacio-escala DoG
9              forall escalas s, 1 ≤ s ≤ S − 2 in parallel
10                 Min_1(x, y, o, s) = Min(D_p(x, y, o, s), D_p(x, y, o, s + 1));
11                 Max_1(x, y, o, s) = Max(D_p(x, y, o, s), D_p(x, y, o, s + 1));
12                 Min_2(x, y, o, s) = Min(Min_1(x, y, o, s), Min_1(x, y, o, s + 1));
13                 Max_2(x, y, o, s) = Min(Max_1(x, y, o, s), Max_1(x, y, o, s + 1));
14                 if β(x, y, o, s) then
15                     if IsLocalMin(Min_2(x, y, o, s)) then
16                         I_p(x, y) es un punto de interés;
17                     end
18                     else if IsLocalMin(Max_2(x, y, o, s)) then
19                         I_p(x, y) es un punto de interés;
20                     end
21                 end
22             end
23         end
24     end
25 end
```

Algoritmo 5.1: Algoritmo propuesto para la detección de puntos de interés del SIFT.

5.3. Conclusiones del capítulo

Cuando se desea obtener una implementación en hardware de un algoritmo, una reformulación del mismo puede muchas veces significar una mejora substancial en su desempeño en hardware. Estas reestructuraciones deben estar enfocadas a sacar el máximo provecho del paralelismo del algoritmo, a la vez que traten de minimizar el área del dispositi-

vo ocupada. En este capítulo se describieron las características de las reestructuraciones propuestas al algoritmo para la detección de puntos de interés del SIFT con el fin de implementarlo en hardware.

Dadas las características del tipo de procesamiento realizado en este algoritmo se tuvo en cuenta la potencialidad de la explotación del paralelismo de datos, del cual se puede hacer uso particionando la imagen de entrada y procesando cada partición por separado. El uso de esta técnica implica una mejora en el tiempo de procesamiento de P veces, donde P es el número de particiones de la imagen de entrada, pero también un aumento en el uso de área del dispositivo del mismo factor.

Con el fin de disminuir la cantidad de operaciones de multiplicación-acumulación y gracias a la propiedad de separabilidad del kernel Gaussiano se propuso utilizar la convolución separable, la cual brinda una ventaja computacional de $k^2/2k$.

Analizando los intervalos de procesamiento inactivos que existirían en el resto de las octavas con respecto a la primera, se planteó en esta sección la posibilidad de entrelazar el procesamiento de las mismas, lo que permitiría realizar todas las operaciones correspondientes a una misma escala en una sola unidad de procesamiento. El uso del entrelazado tiene como desventaja que se añade complejidad a la hora de controlar el orden del mismo, además de introducir una restricción en la velocidad del sistema de la mitad de su velocidad máxima alcanzable si no se usase esta técnica.

Capítulo 6

Arquitectura hardware para la detección de puntos de interés del SIFT

En el capítulo anterior se propuso una reformulación al algoritmo para la detección de puntos de interés del SIFT propuesto por Lowe [19]. Esta reformulación está enfocada en sacar el máximo provecho del paralelismo de este algoritmo y a minimizar el área de dispositivo ocupada por una implementación del mismo. En este capítulo se propone una arquitectura hardware para la detección de puntos de interés del SIFT. Dicha arquitectura implementa el algoritmo paralelo propuesto en el capítulo anterior.

La arquitectura propuesta usa elementos antes discutidos en el Capítulo 5 como la explotación del paralelismo de datos, la explotación de la propiedad de separabilidad del *kernel* Gaussiano y el entrelazado del procesamiento de las octavas. El uso de estos elementos contribuye a un mejor uso del área del dispositivo ya que provee una manera eficiente de realizar este proceso.

En este capítulo se describen cada una de las partes que forman la arquitectura, las cuales también son ilustradas mediante esquemáticos, señalándose sus correspondencias con el algoritmo paralelo propuesto en el capítulo anterior.

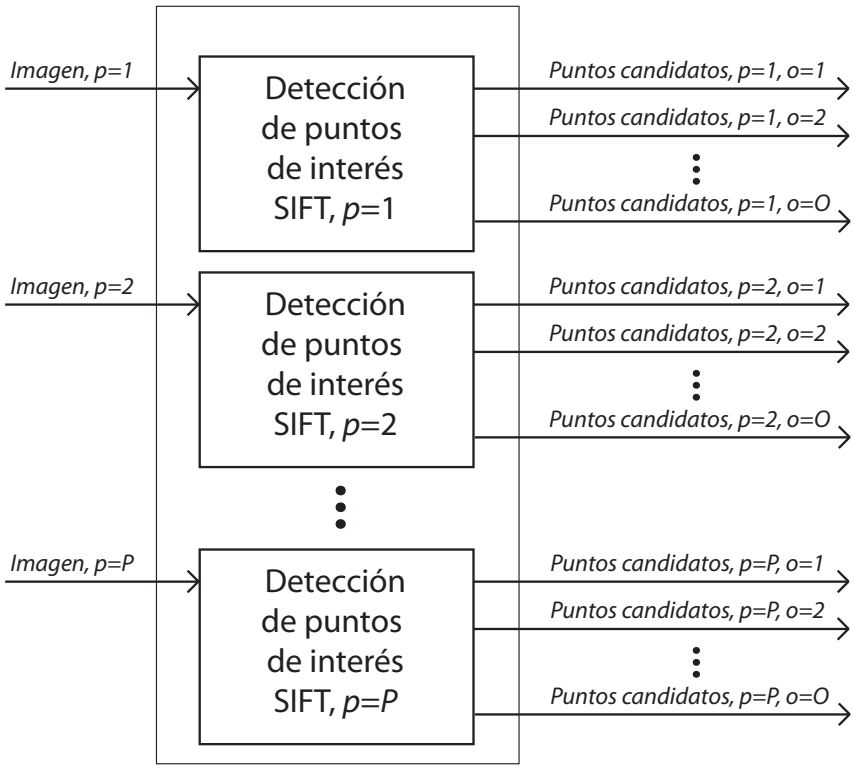


Figura 6.1: La arquitectura propuesta en este trabajo para la detección de puntos de interés del SIFT recibe como entrada P particiones de una imagen en escala de grises y devuelve como salida un vector para cada octava o en cada partición p indicando si cada punto es o no, un punto de interés.

6.1. Arquitectura hardware propuesta

En la Figura 6.1 se muestra un diagrama al más alto nivel de la arquitectura propuesta en este trabajo para la detección de puntos de interés de SIFT. Esta arquitectura recibe como entrada P particiones de una imagen en escala de grises en forma de un vector que es resultado de la concatenación de las columnas de la imagen, donde cada valor indica la intensidad de un píxel. Como salida se devuelve un vector con la misma estructura indicando si cada píxel en cada octava es o no, un punto de interés en cada partición.

Para la detección de puntos de interés del SIFT, la arquitectura se dividió en dos grandes partes: i) la generación del espacio-escala de DoG y ii) la detección de extremos en el espacio-escala de DoG (Ver Figura 6.2). La imagen de entrada es procesada por el bloque de generación del espacio-escala de DoG, el cual devuelve $O \cdot (S-1)$ imágenes que

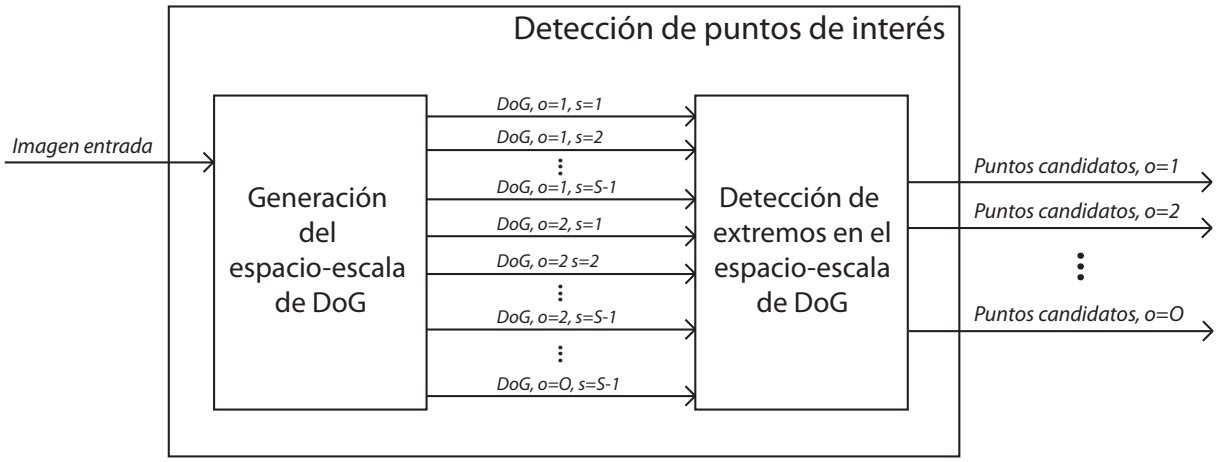


Figura 6.2: Cada bloque en la Figura 6.1 está formado por dos grandes bloques: uno para la generación del espacio-escala de DoG y el otro para la detección de extremos en dicho espacio. El primer bloque recibe la imagen, generando el espacio-escala de DoG, el cual sirve como entrada al segundo bloque, el cual extrae los puntos de interés.

conforman el espacio-escala de DoG. Estas imágenes son pasadas al bloque de detección de extremos que determina cuales puntos de la imagen son considerados como puntos de interés.

6.1.1. Generación del espacio-escala de DoG

En las arquitecturas propuestas en [22] y [3], con el objetivo de generar el espacio-escala de DoG, se usa un bloque de convolución para cada operación de este tipo que se realiza, dividiendo por octavas el procesamiento, por lo que se necesitan $O \cdot S$ bloques de convolución. En la arquitectura aquí presentada, proponemos usar solamente S bloques de convolución para realizar las $O \cdot S$ convoluciones, dividiendo el procesamiento en escalas y manteniendo el mismo rendimiento del sistema. Esto se logra mediante el entrelazado del procesamiento de las octavas como se detalló en la Sección 5.1.3.

Un diagrama del bloque de generación del espacio-escala de DoG se muestra en la Figura 6.3. Este diagrama muestra un sistema de cuatro octavas y cinco escalas ($O = 4, S = 5$). Esta arquitectura puede ser generalizada para cualquier configuración de estos parámetros.

La arquitectura propuesta está formada, en mayor medida, por bloques de cálculo

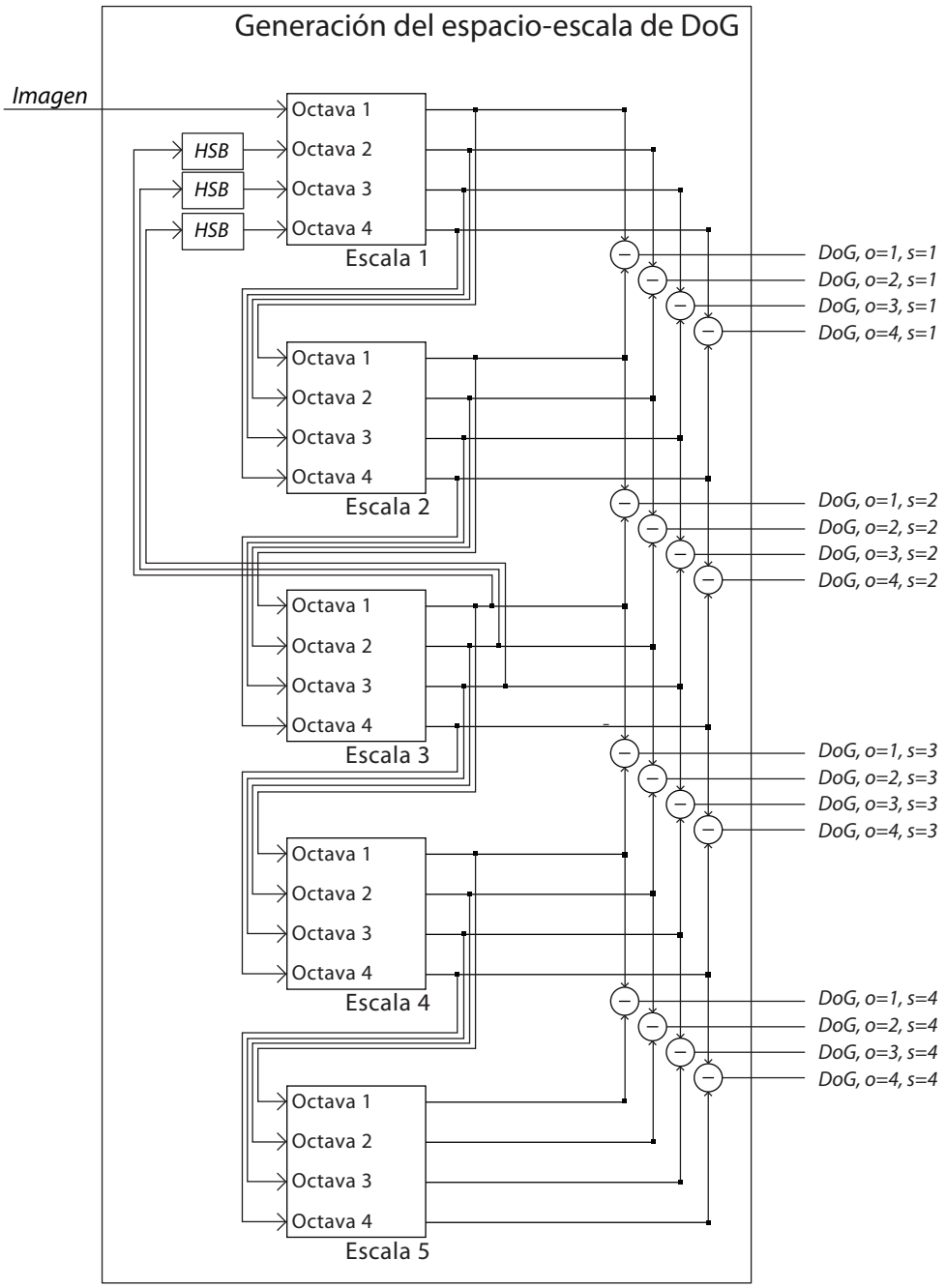


Figura 6.3: Esquema a alto nivel de la arquitectura para la generación del espacio-escala de DoG. Se muestran las conexiones en cascada entre los bloques procesadores de escalas, donde cada uno de estos procesa O octavas. Como salida de este bloque se obtiene el espacio-escala de DoG el cual sirve como entrada al bloque de detección de extremos en dicho espacio.

de escala (SCB). Un solo bloque SCB realiza las O operaciones de filtrado Gaussiano para una escala dada siguiendo el procedimiento de entrelazado descrito en la Sección 5.1.3. Por lo tanto, cada SCB tiene O puertos de entrada y O de salida, uno para cada octava respectivamente, donde el período de muestreo para cada octava está definido por la Ecuación 5.2. Los bloques SCB están conectados en cascada con el fin de usar un *kernel* de convolución de tamaño fijo y así evitar las convoluciones con *kernels* de gran tamaño. Esta conexión en cascada puede ser observada en la Figura 6.3.

Un bloque SCB, para realizar el filtrado Gaussiano, aprovecha la propiedad de separabilidad del kernel Gaussiano como se describe en la Sección 5.1.2. Aprovechando dicha propiedad, este bloque realiza dicho filtrado primero en la dirección horizontal y luego en la vertical, lo que se puede observar en la Figura 6.4.

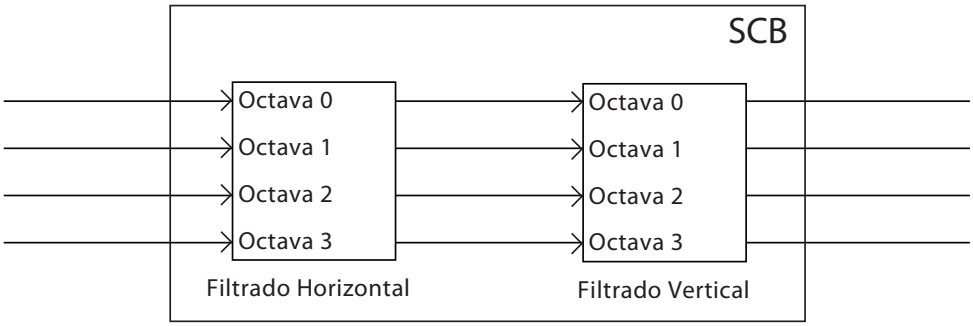


Figura 6.4: La convolución 2D es realizada por dos bloques de convolución 1D, pasando primero por un filtrado horizontal y luego otro vertical.

La organización interna del bloque de filtrado horizontal se detalla en la Figura 6.5. Cada señal de entrada es desplazada a través de $k-1$ registros, donde k es la cantidad de coeficientes del *kernel* de convolución. Las k señales correspondientes a las O octavas son multiplexadas con el objetivo de controlar el orden de procesamiento de las octavas y lograr el entrelazado de las mismas. La lógica de los multiplexores en un instante t está determinada por el bloque M, el que implementa la función $m(t)$ y cumple la condición establecida en la Ecuación 5.3. El orden de entrelazado se define a continuación:

$$m(t) = \begin{cases} o_1 & \text{if } t \mod \tau(o_1) \equiv \varepsilon(o_1) \\ o_2 & \text{if } t \mod \tau(o_2) \equiv \varepsilon(o_2) \\ \vdots & \vdots \\ o_O & \text{if } t \mod \tau(o_O) \equiv \varepsilon(o_O). \end{cases}$$

donde $\varepsilon(o)$ es la latencia de la octava o en la línea de entrelazado.

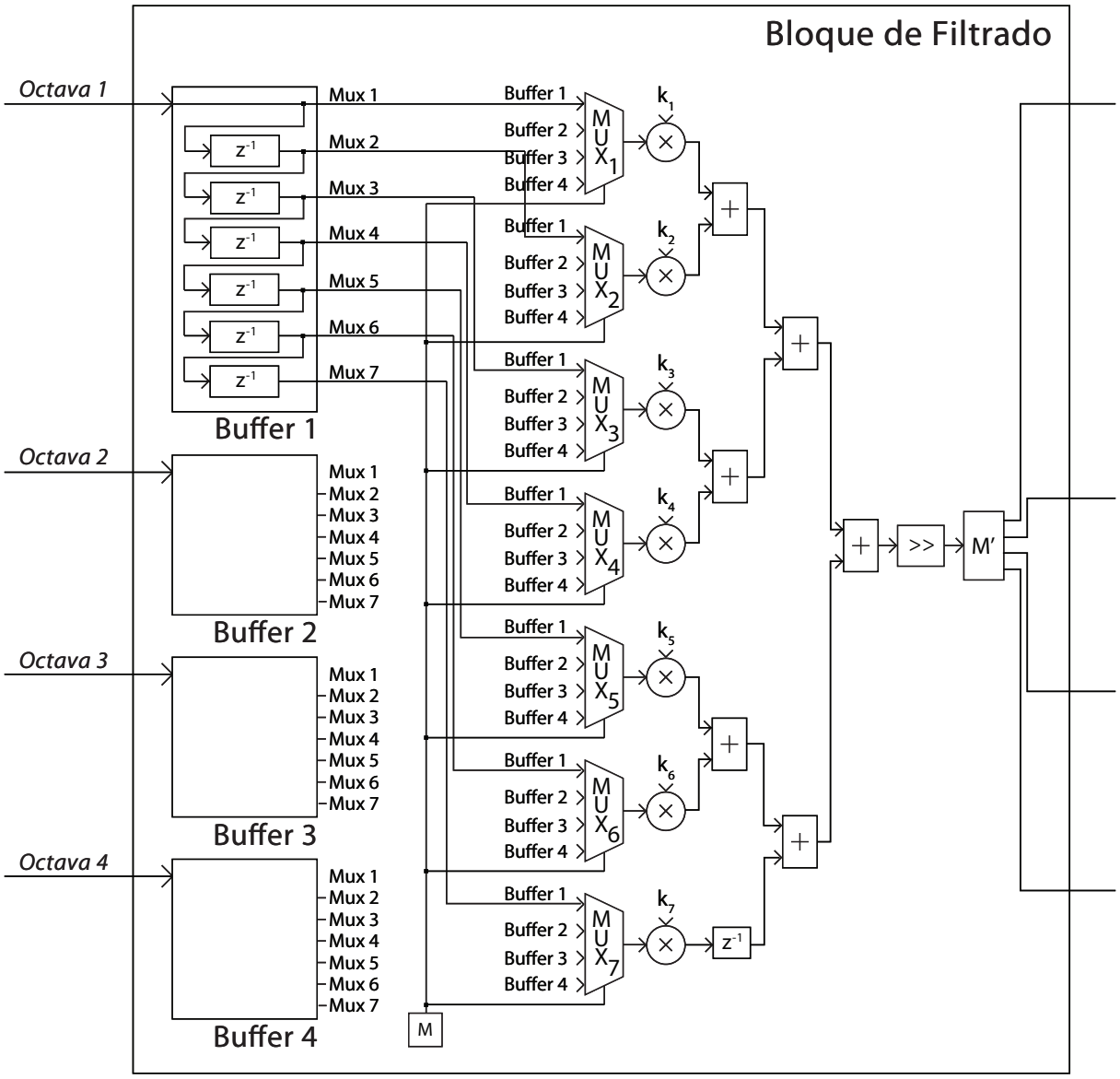


Figura 6.5: Estructura interna de un bloque de convolución 1D. Se puede apreciar como todas las octavas son procesadas en el mismo bloque. La lógica de entrelazado de dicho procesamiento está dada por el bloque M y los multiplexores a los que controla.

La estructura del bloque de filtrado vertical es la misma que la del horizontal, con la diferencia de que cada *buffer* almacena las k últimas líneas de la imagen en vez de los k últimos píxeles. La estructura interna de estos bloques se muestra en la Figura 6.6. Para almacenar estos valores se utiliza un bloque de RAM para almacenar cada línea. Por lo tanto, esta parte del diseño utilizará $k-1$ bloques de RAM por cada octava en cada bloque SCB, de ahí que la cantidad de bloques de RAM utilizados para la generación del espacio-escala de DoG está dada por:

$$\#bloquesRAM = (k-1) \cdot O \cdot S.$$

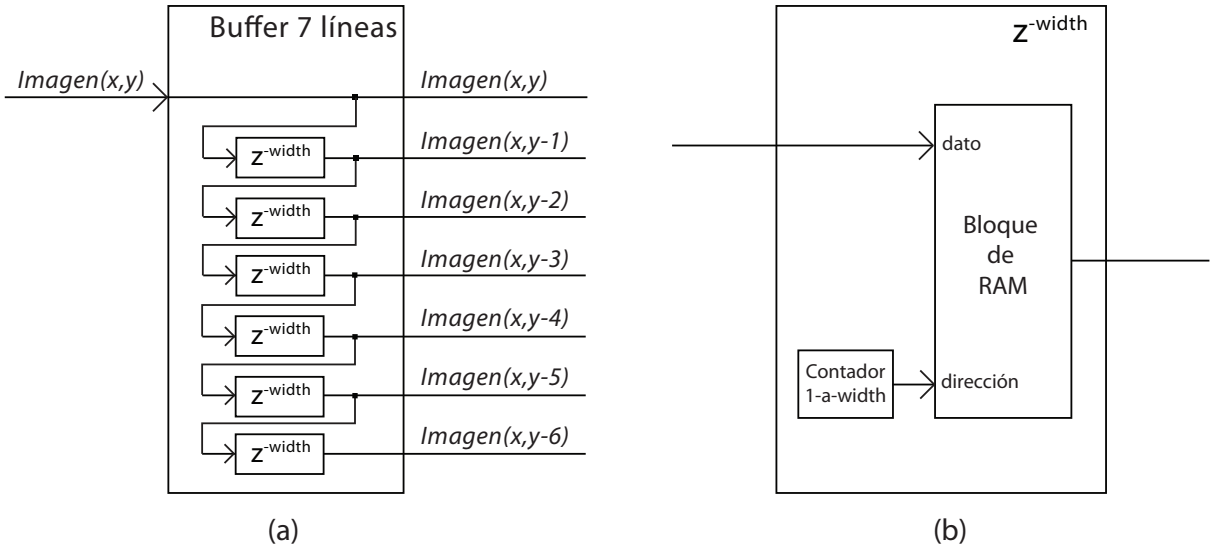


Figura 6.6: En (a) se muestra la estructura interna de un *buffer* de siete líneas, donde seis *buffers* de una línea son conectados en cascada. En (b) se muestra la estructura interna de cada uno de estos *buffers* de una línea, los cuales utilizan los bloques de RAM del dispositivo para almacenar una línea.

Un bloque de convolución 1D ocupa k multiplicadores y $k-1$ sumadores, cantidad a la que llamaremos $r(k)$. Entonces, la cantidad de recursos aritméticos de estos tipos utilizada por la arquitectura está dada por:

$$\#multiplicadores\_sumadores = 2r(k) \cdot S.$$

Como se puede observar, esta cantidad solamente depende del tamaño del *kernel* de

convolución, de la cantidad de escalas y no del número de octavas.

El bloque HSB en la Figura 6.3 realiza el subescalamiento de la imagen. Con este fin, se usa un registro de desplazamiento direccionable y un contador.

Con el fin de evitar trabajar con valores de punto fijo, los coeficientes del *kernel* de convolución son multiplicados por una constante. Luego, los resultados filtrados son normalizados dividiéndolos por esta misma constante. Preferiblemente esta constante debe ser una potencia de dos, para así poder reemplazar la operación de división por un simple corrimiento.

6.1.2. Detección de extremos en el espacio-escala de DoG

El bloque de procesamiento para la detección de extremos recibe como entrada el espacio-escala de diferencias de Gaussianas. Este bloque implementa el algoritmo para este propósito detallado en la Sección 5.2. Un diagrama a alto nivel de este bloque para un sistema con un espacio-escala de DoG de cuatro octavas y cuatro escalas se muestra en la Figura 6.7. Esta arquitectura también puede ser generalizada para cualquier configuración de estos parámetros.

Para cada octava, todas sus imágenes de diferencias de Gaussianas son pasadas a un bloque *esExtremo*, el cual determina cuales son los puntos que serán considerados como de interés. La salida de este bloque es un vector de 1 bit que indica para cada punto si éste es considerado o no como de interés. La estructura interna de un bloque *esExtremo* se detalla en la Figura 6.8.

Como se explicó en la Sección 5.1.1, el procedimiento diseñado para la detección de extremos tiene como objetivo reutilizar los cálculos intermedios por más de una escala, resultando en un ahorro de recursos del dispositivo. Como se puede observar en la Figura 6.8, el cálculo de los mínimos y máximos a través de tres imágenes adyacentes fue dividido en dos etapas para de este modo reutilizar los valores de mínimos y máximos en las imágenes en común. Los bloques encerrados en líneas discontinuas implementan la bandera β, la cual indica si cada valor mínimo o máximo es igual al píxel correspondiente en la DoG y si no es igual a su correspondiente en cualquiera de las escalas adyacentes. Los

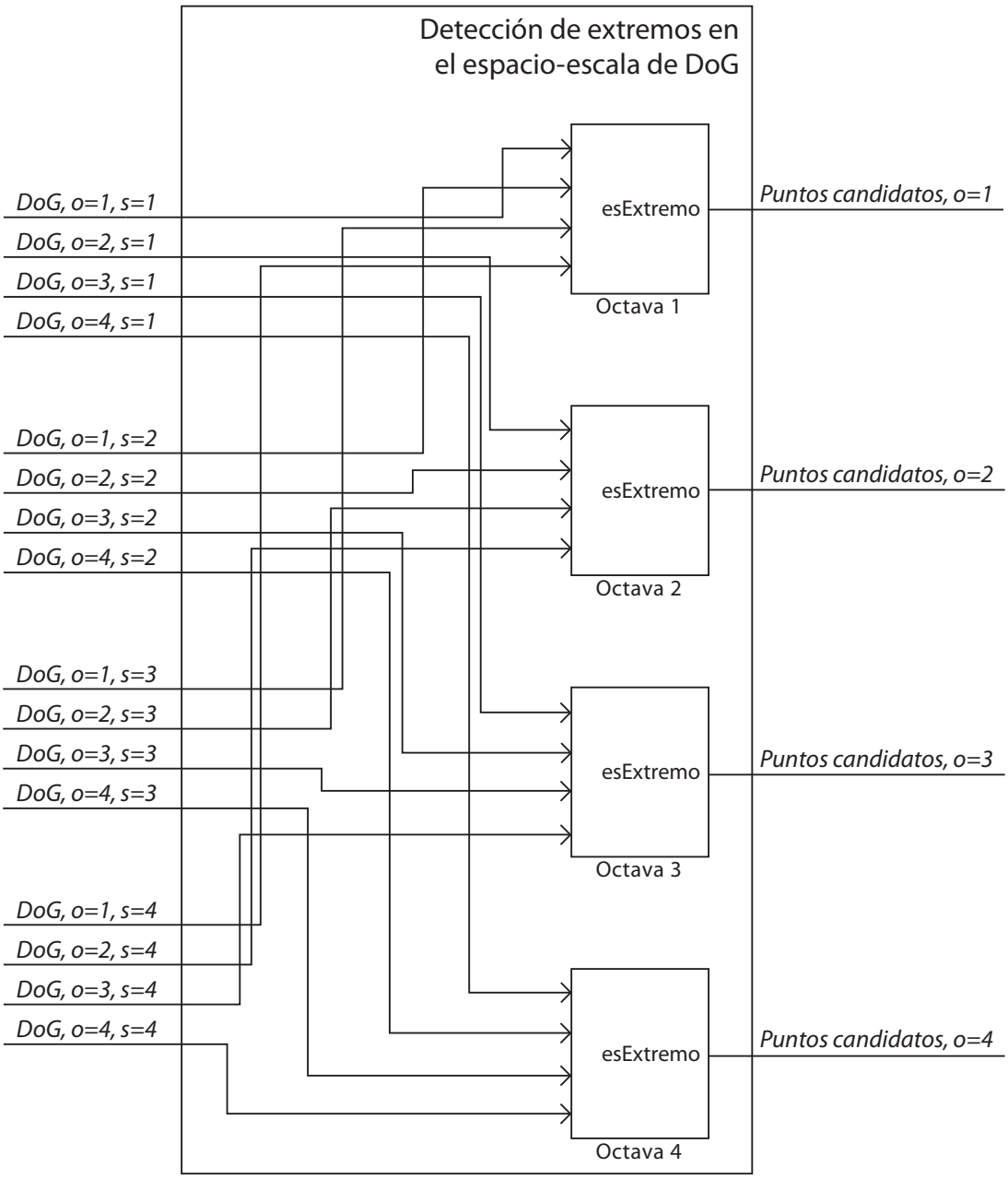


Figura 6.7: Diagrama a alto nivel de la arquitectura para la detección de extremos en el espacio-escala de DoG. Cada bloque *esExtremo* recibe todas las imágenes de DoG de una octava, para la cual, este bloque determina los puntos de interés.

bloques *esMínLocal* y *esMáxLocal* determinan si cada píxel es un extremo local en una vecindad de 3×3, tomando en cuenta además el valor de la bandera β. Para que un punto sea considerado como de interés basta con que sea un mínimo o un máximo en cualquiera

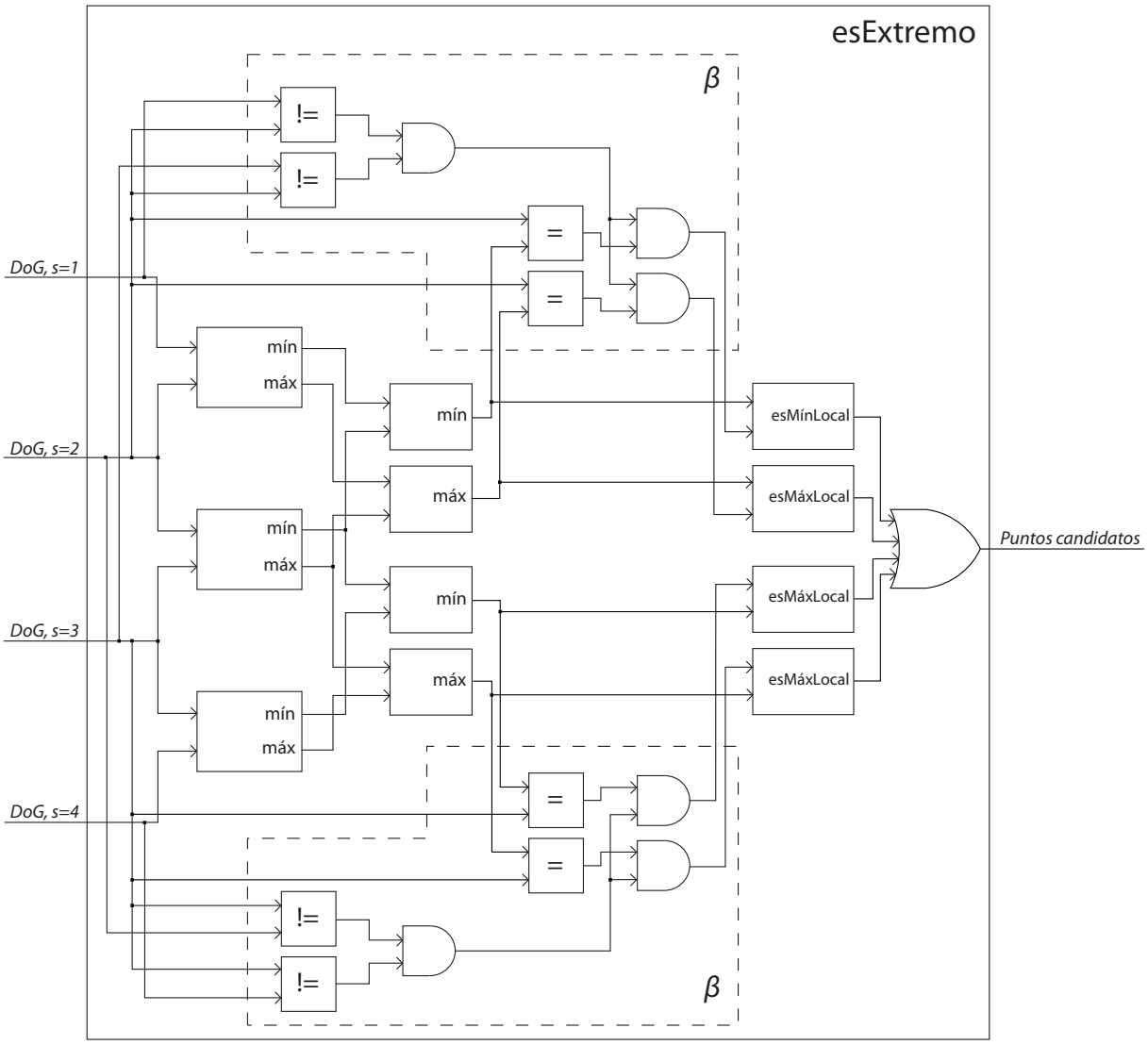


Figura 6.8: Estructura interna de un bloque *esExtremo*. Primeramente, este bloque obtiene los máximos y mínimos para cada tres imágenes adyacentes, esto lo realiza en dos etapas con el fin de reutilizar resultados intermedios. Los bloques encerrados en líneas discontinuas implementan la bandera β, que sirve como entrada a los bloques *esMínLocal* y *esMáxLocal* los que determinan los puntos de interés.

de las escalas, por lo que es usada una compuerta OR a la salida.

6.2. Conclusiones del capítulo

En este capítulo se propuso una arquitectura hardware para la detección de puntos de interés del SIFT. Esta arquitectura implementa el algoritmo paralelo propuesto en el

capítulo anterior. La arquitectura propuesta permite procesar P particiones de la imagen de manera concurrente. La arquitectura está dividida en dos grandes partes: i) la generación del espacio-escala de diferencias Gaussianas y ii) la detección de extremos en dicho espacio. Dichas partes fueron descritas en el presente capítulo, mostrándose mediante diagramas los detalles y estructura de las mismas y sus correspondencias con el algoritmo paralelo propuesto en el capítulo anterior.

El principal aporte de dicha arquitectura y del algoritmo que implementa, radica en que a medida que aumenta el número de octavas a procesar, la cantidad de área del dispositivo ocupada se mantiene casi constante, solamente aumentando en el número de bloques de memoria necesarios para almacenar las nuevas octavas y la lógica necesaria para controlar el entrelazado de más octavas. Este fenómeno se debe a que todas las octavas para una misma escala, sin importar la cantidad, siempre serán procesadas en el mismo bloque de convolución.

En el próximo capítulo se detallan los experimentos que muestran la aportación antes mencionada, así como pruebas de implementación relacionadas con el área del dispositivo ocupada, restricciones de tiempo, correctez, repetitividad y distintividad de los resultados obtenidos, entre otras.

Capítulo 7

Experimentos y resultados

En este capítulo se detallan las pruebas realizadas a la arquitectura propuesta para la detección de puntos de interés del SIFT y se analizan los resultados obtenidos. Las pruebas realizadas a la arquitectura se centran en medir la confiabilidad y exactitud de los resultados obtenidos, así como la manera en que se afecta la repetitividad y distintividad de las características SIFT extraídas. También, se analiza la eficiencia en el uso del área del dispositivo y la aceleración obtenida con respecto a implementaciones en software. Además, se comparan los resultados obtenidos con otras arquitecturas para este mismo fin reportadas en la literatura.

7.1. Plataforma de experimentación

Con el objetivo de comprobar la exactitud y confiabilidad de la arquitectura propuesta se implementó un sistema híbrido donde la etapa de detección de puntos candidatos es realizada por la arquitectura propuesta. El resto de las etapas del algoritmo son realizadas por la implementación software de Vedaldi [31] disponible en Internet. La Figura 7.1 muestra un diagrama a alto nivel de este sistema híbrido.

La arquitectura propuesta fue modelada y simulada usando Xilinx System Generator 10.1 + Simulink. Como se muestra en la Figura 7.1, los resultados de la primera etapa del SIFT obtenidos de la simulación de la arquitectura son pasados al *workspace* de Matlab, de donde la implementación software toma los valores necesarios para llevar a cabo el resto de las etapas del algoritmo. La comparación de los resultados obtenidos por esta

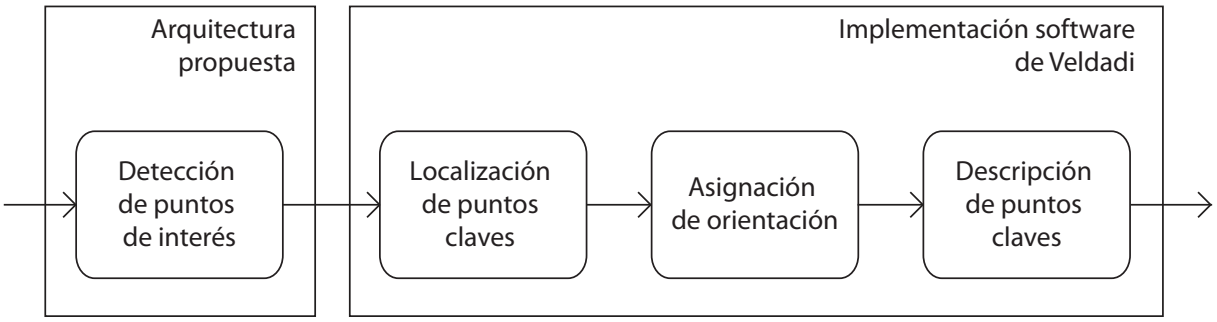


Figura 7.1: Plataforma de experimentación. La etapa de detección de puntos de interés del SIFT es llevada a cabo por la arquitectura propuesta. Los resultados obtenidos son pasados a una implementación software la que se encarga del resto de las etapas.

implementación híbrida y por la implementación software nos proveerá de una base para determinar la calidad de los resultados producidos por la arquitectura propuesta.

7.2. Pruebas de exactitud

Para comprobar la exactitud de la arquitectura propuesta se compararon los resultados obtenidos por la implementación híbrida y la implementación software de Vedaldi en un conjunto de 38 imágenes. Las imágenes fueron tomadas del sitio web de Krystian Mikolajczyk[1]. Estas imágenes fueron capturadas con el objetivo de probar algoritmos de extracción de características locales y son usadas en [20] para comparar los métodos de este tipo en el estado del arte. El tamaño de las imágenes es 800×600 píxeles. Los experimentos y análisis realizados se centran solamente en la generación del espacio-escala de DoG, pues para la detección de extremos en dicho espacio los resultados son idénticos a la implementación software.

Como medida de evaluación se usó el Error Cuadrado Medio (MSE, del inglés, *Mean Squared Error*). El MSE es una manera de cuantificar la diferencia entre un resultado obtenido y el resultado esperado o verdadero. Éste mide el promedio del cuadrado del error, donde este error es la cantidad en la que el resultado obtenido difiere del valor verdadero. En este trabajo es usado para medir la diferencia entre los valores obtenidos por la arquitectura propuesta y la implementación software. El MSE es definido en la

[1]Disponible en http://lear.inrialpes.fr/people/Mikolajczyk/

Ecuación 7.1.

$$MSE = \frac{\sum_{M,N} [I_{sw}(m,n) - I_{hw}(m,n)]^2}{MN} \tag{7.1}$$

donde $I_{sw}(m,n)$ y $I_{hw}(m,n)$ son los valores de intensidad del píxel (m,n) en las imágenes de tamaño $M \times N$ generadas por la implementación software y la implementación híbrida respectivamente.

Para estas pruebas se tomó en cuenta el espacio-escala Gaussiano generado por cada una de las implementaciones para una configuración de seis octavas y cinco escalas (O=6, S=5). Valores más pequeños de MSE indicarán que los resultados obtenidos por nuestra arquitectura son más similares a los obtenidos por la implementación software, o sea, que tienen un mayor grado de exactitud. Los resultados de esta prueba son resumidos en la Figura 7.2.

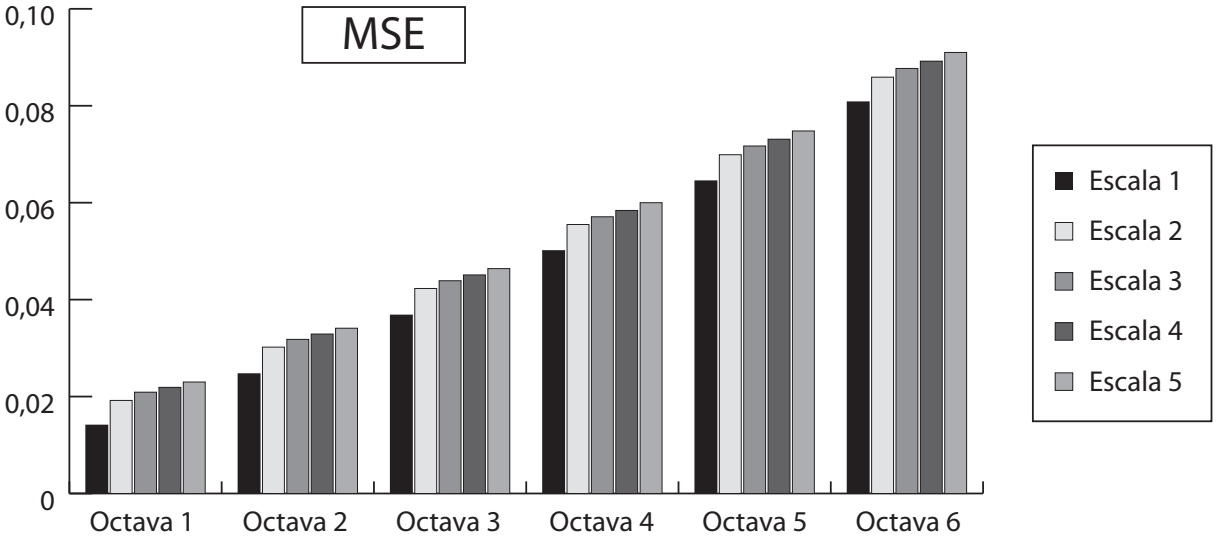


Figura 7.2: Valores de MSE para cada octava y escala. Los errores de aproximación cometidos en el proceso de convolución son propagados según el orden de dependencia entre las imágenes en el espacio-escala, de ahí que el MSE aumente en ese mismo orden.

Cada barra en la Figura 7.2 representa el MSE promedio de las 38 imágenes de prueba en una octava y escala específica de los espacios-escala Gaussianos generados por ambas implementaciones. En esta figura, se puede apreciar como el error dentro de una octava va aumentando en cada escala, así como también aumenta de octava en octava. Esto se

debe a que, como se explicó en la Sección 2.4, cada imagen-escala depende de la anterior y la primera imagen en cada octava de la antepenúltima en la octava anterior. Por lo tanto, el error obtenido en el cálculo de cada imagen es propagado a las siguientes. Estas dependencias entre las imágenes pueden ser observadas en la Figura 6.3.

El error inicial y que es luego incrementado a lo largo de cada escala y octava está provocado por las aproximaciones realizadas al kernel de convolución Gaussiano con el fin de realizar multiplicaciones con valores enteros y no con números flotantes. Este proceso es detallado en la Sección 6.1.1.

El MSE brinda una medida cuantitativa de cuanto afectan las aproximaciones al espacio-escala de Gaussianas obtenido, pero no brinda ninguna información acerca de cuanto esas aproximaciones afectan el resultado final del algoritmo, que es la detección de puntos claves. Con este objetivo, sobre este mismo conjunto de imágenes se extrajeron los puntos claves SIFT usando la implementación híbrida y la implementación software antes descritas. Para cada imagen se hallaron las correspondencias entre los puntos claves detectados por ambas implementaciones. Un ejemplo de esto se muestra en la Figura 7.3.

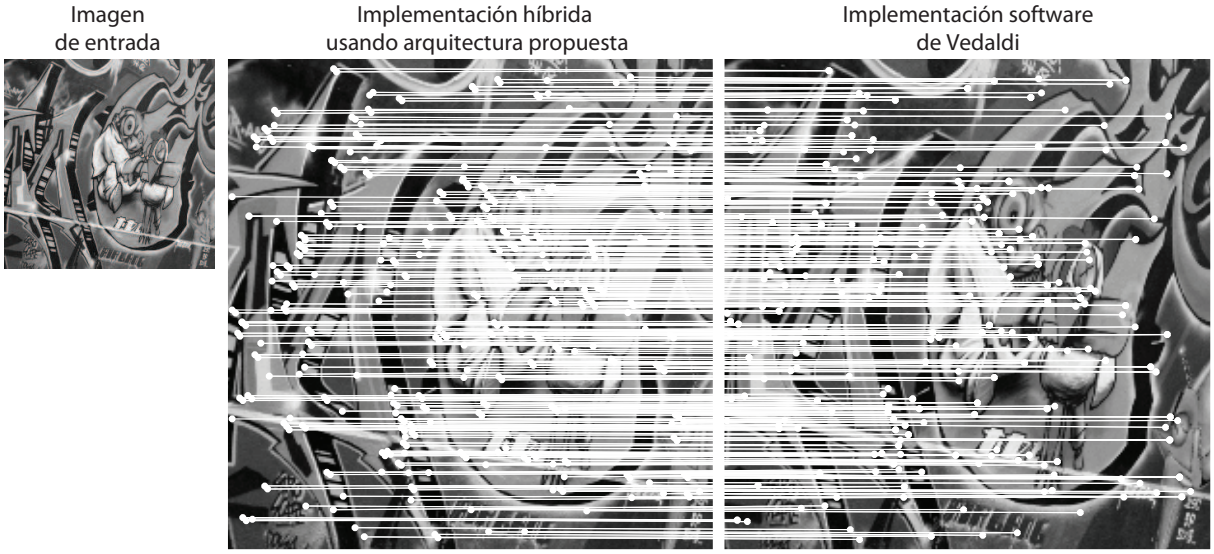


Figura 7.3: De la imagen de entrada son extraídas las características SIFT por la implementación híbrida y la software independientemente. Luego a partir de sus correspondencias se calcula el error cometido en la detección.

La Figura 7.4 muestra la variación promedio en cada una de las 38 imágenes pa-

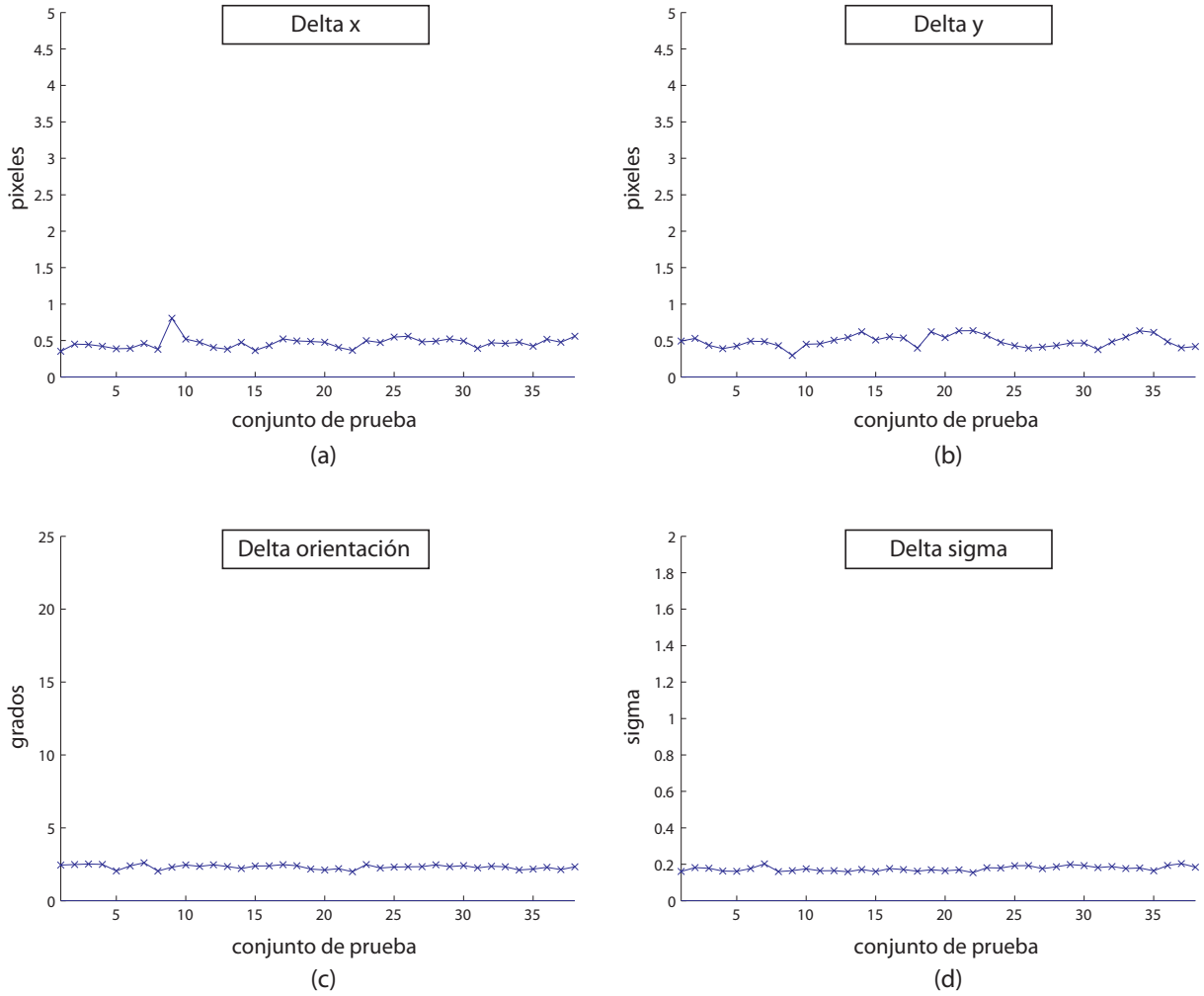


Figura 7.4: Errores cometidos en la detección con respecto a la implementación software. En (a) y (b) se muestran los errores en localización y en (c) y (d) los errores en orientación y escala respectivamente.

ra los resultados obtenidos con ambas implementaciones. Se midieron variaciones en las coordenadas, la escala y la orientación del gradiente de los puntos detectados. Se puede observar que las variaciones, en cuanto a las coordenadas de los puntos detectados para estas imágenes, como promedio no superaron un píxel, aunque las mayores variaciones presentadas fueron de cuatro píxeles. En cuanto a la orientación del gradiente de cada punto la variación también fue pequeña, su media no superó los 2,5°, las mayores variaciones fueron de 10,0°. La variación promedio de σ fue de 0,2, lo que también representa

una pequeña diferencia.

7.3. Pruebas de repetitividad y distintividad

En la sección anterior se presentaron resultados que muestran la exactitud de la arquitectura propuesta desde un punto de vista más teórico enfocándose en el error cometido en la generación del espacio-escala Gaussiano y en la detección de los puntos claves. En esta sección se sigue una experimentación más enfocada al uso de estas características en una aplicación. En una aplicación real, no sólo se necesita que los puntos sean detectados con la mayor exactitud, sino que también sean repetitivos y distintivos, o sea, que un mismo punto pueda ser detectado en diferentes vistas de la escena u objeto y que además éste pueda diferenciarse del resto.

Con este fin, se comprobaron las correspondencias entre los puntos SIFT detectados en diferentes imágenes de una misma escena. La Figura 7.5 muestra ejemplos de las imágenes utilizadas para evaluar la repetitividad y distintividad de la arquitectura propuesta. Se evaluaron cuatro cambios diferentes en las condiciones de las imágenes: cambios de punto de vista o perspectiva (Figura 7.5 (a)), cambios de escala y rotación (Figura 7.5 (b)), compresión JPEG (Figura 7.5 (c)) y desenfoque de la imagen (Figura 7.5 (d)). En las imágenes con cambios de punto de vista la posición de la cámara varía desde una posición frontal hasta una lateral con una desviación de 60 grados. Las imágenes con cambios de escala y rotación fueron obtenidas variando el *zoom* y la inclinación de la cámara. La secuencia de variaciones en la compresión JPEG fue obtenida con un software estándar modificando el parámetro de calidad de la imagen. Las imágenes desenfocadas fueron obtenidas variando el foco de la cámara. Estas imágenes también fueron obtenidas del sitio web de Krystian Mikolajczyk, las mismas fueron capturadas con el objetivo de realizar este mismo tipo de pruebas.

Para medir la repetitividad y distintividad de los puntos claves usamos la tasa de correspondencias. Ésta se calcula como la relación entre el número de correspondencias correctas entre dos imágenes y el menor número de puntos detectados en este par de

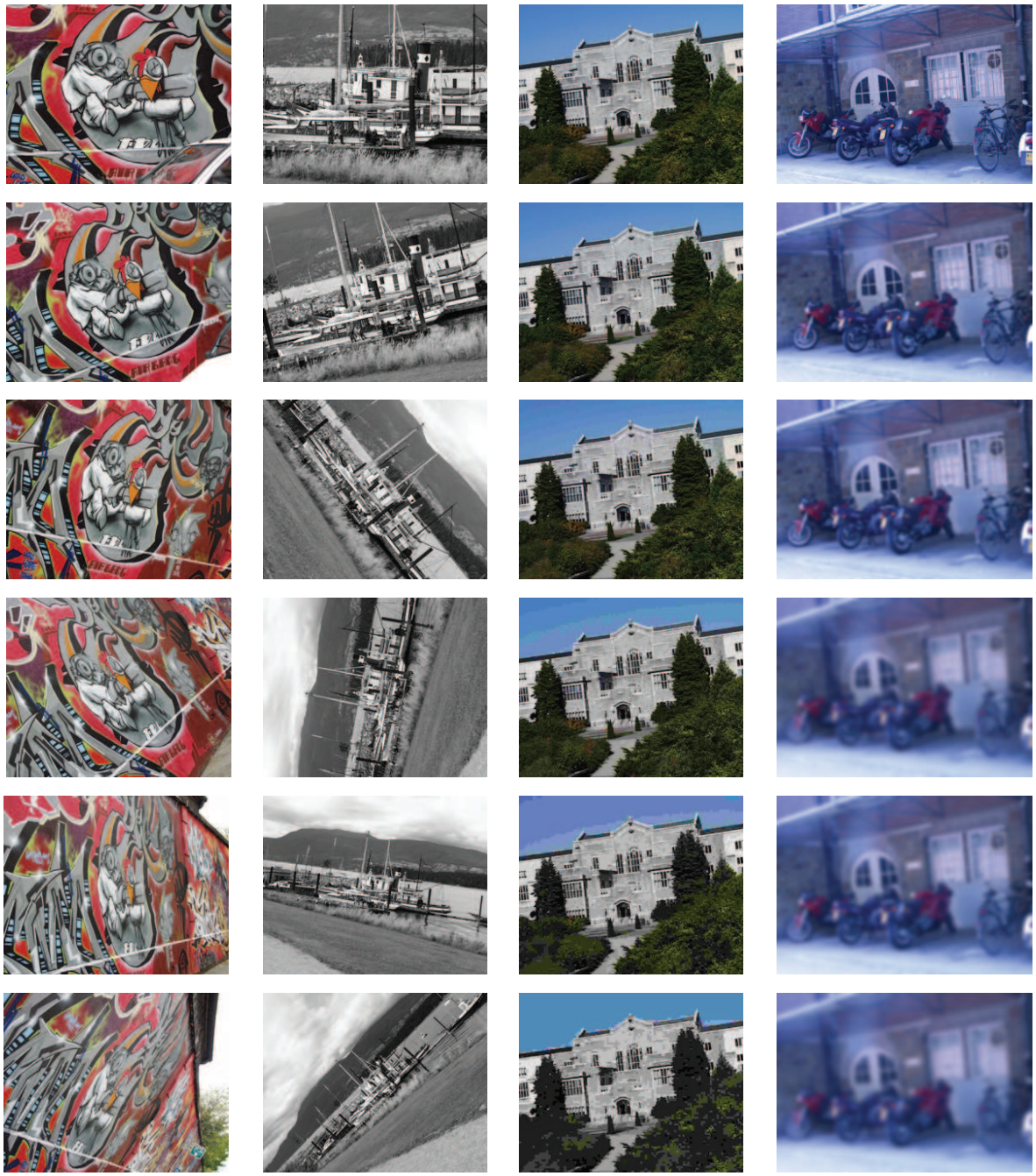


Figura 7.5: Conjunto de prueba. En (a) variaciones de punto de vista, en (b) cambios de escala y rotación, en (c) variaciones en la compresión JPEG y en (d) variaciones de desenfoque. En las pruebas realizadas, para cada uno de estos conjuntos, la primera imagen es tomada como imagen de referencia.

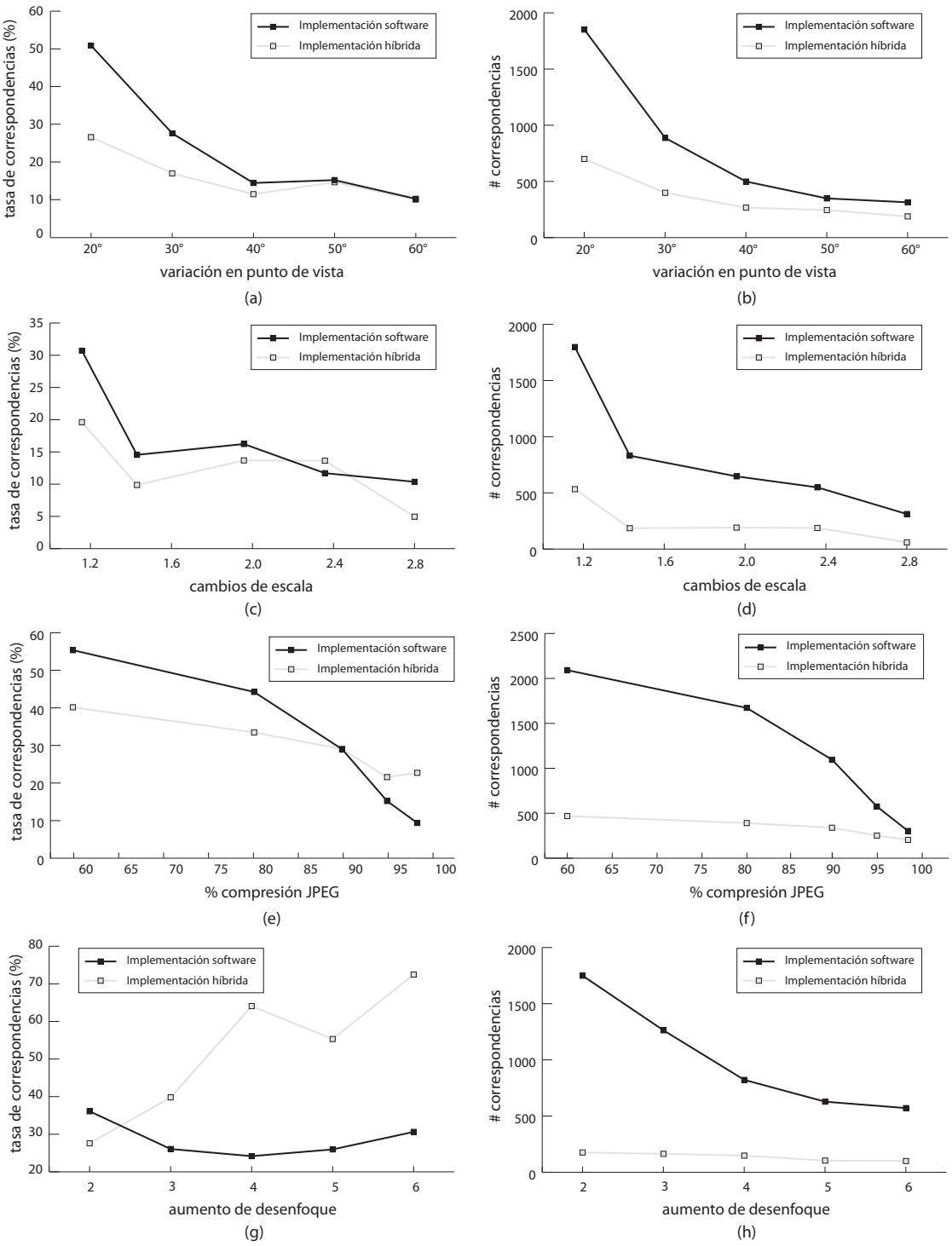


Figura 7.6: Variaciones en cuanto a tasas de correspondencias y número de correspondencias para cada conjunto de imágenes.

imágenes:

$$tasa\_correspondencia(I, I') = \frac{\#correspondecias\_correctas(I, I')}{\text{mín}(\#puntosSIFT(I), \#puntosSIFT(I'))}. \qquad (7.2)$$

Se desea que la arquitectura propuesta presente una alta tasa de correspondencias pero también un alto número de correspondencias. Además, que esté lo más cerca posible de los valores de estas medidas obtenidos por la implementación software.

Los resultados de estas pruebas se muestran en la Figura 7.6. Esta medida fue calculada, para cada una de las variaciones antes mencionadas, entre una imagen de referencia (primera imagen en cada columna de la Figura 7.5) y el resto de las imágenes en el conjunto. Un resultado ideal sería una línea horizontal en el 100 %.

Como se puede observar en la Figura 7.6, para las tres primeras variaciones la tasa de correspondencias de la arquitectura propuesta descendió con respecto a la implementación software, principalmente en las imágenes de menores variaciones, con tasas más similares en las imágenes con mayores variaciones. A pesar de estos descensos, la cantidad de correspondencias encontradas siempre estuvieron cercanas a las doscientas, lo que representa una buena cifra para muchas aplicaciones.

Las líneas que describen los resultados obtenidos por la arquitectura propuesta en todos los casos presentan menor pendiente que las obtenidas para la implementación software, lo que indica que a pesar de obtener menores tasas de correspondencias en las imágenes con menores variaciones, sus resultados fueron más estables a lo largo de todas las imágenes. Incluso para el conjunto de datos de imágenes desenfocadas los resultados mostraron una pendiente positiva.

Las diferencias en las tasas de correspondencias y cantidad de correspondencias obtenidas por la implementación híbrida con respecto a la implementación software, muestran los efectos en la repetitividad y distintividad ocasionados por los errores generados los cuales fueron discutidos en la sección anterior. No obstante, el descenso de estos valores no fue muy drástico.

7.4. Pruebas de eficiencia en el uso de área del dispositivo

En el Capítulo 5 se plantea que al introducir el entrelazado del procesamiento de las octavas, para una misma escala, todas sus octavas podrían ser calculadas en la misma unidad de procesamiento, por lo tanto, esto traería una gran ventaja en cuanto al consumo de área del dispositivo. Luego, en el Capítulo 6 se describe la arquitectura que logra llevar a cabo esta idea y se evidencia de mejor manera el ahorro en recursos del FPGA que ésta implica. En esta sección se presentan un conjunto de experimentos realizados con el fin de mostrar la validez de esta contribución.

Con este objetivo, se rediseñó la arquitectura propuesta manteniendo las mismas características exceptuando el entrelazado del procesamiento de las octavas. En este nuevo diseño se agregó un bloque de convolución para cada imagen en cada escala y en cada octava. El diseño resultante es muy similar al propuesto por Bonato y colaboradores en [3]. Su diagrama a alto nivel se mantiene igual al mostrado en la Figura 6.3. Internamente sólo se modificó el bloque de filtrado. La Figura 7.7 muestra el resultado de dicha modificación, donde cada imagen es procesada por un bloque de convolución, esta modificación puede ser contrastada con la Figura 6.5, en la cual, todas las imágenes de una misma escala son procesadas por el mismo bloque de convolución.

Para mostrar las ventajas del entrelazado de las octavas, se obtuvieron varias implementaciones de ambas arquitecturas para diferentes configuraciones de sus parámetros, donde el número de octavas variaba entre tres y siete ($O = [3, 7]$) y la cantidad de escalas ($S = 5$) y las dimensiones de la imagen ($M = 512, N = 512$) se mantuvieron constantes. Luego, estas implementaciones fueron sintetizadas para un dispositivo Xilinx Virtex II Pro (XC2VP30-5FF1152), con el objetivo de obtener la cantidad de recursos del dispositivo ocupados por cada una de estas arquitecturas para distintas cantidades de octavas y poder obtener una medida cuantitativa de la ventaja del uso del entrelazado de las octavas en cuanto a uso de área del dispositivo.

La Figura 7.8 muestra un resumen de estas comparaciones. En las Figuras 7.8 (a) y

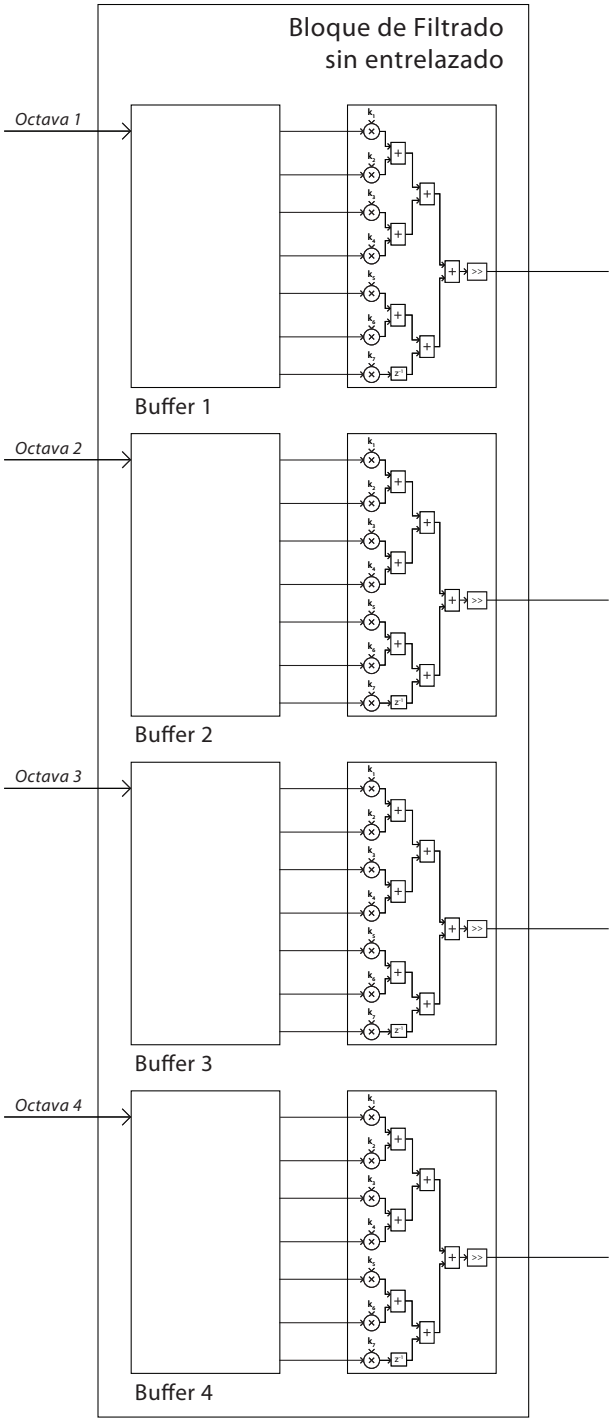


Figura 7.7: Bloque de filtrado de la arquitectura propuesta sin utilizar el entrelazado de octavas. Nótese que se utiliza un bloque de convolución para cada imagen, a diferencia del entrelazado que utiliza un solo bloque para todas las imágenes de una misma escala.

(c) se muestran la cantidad de registros y tablas de verdad ocupadas por cada una de las arquitecturas. Se puede notar fácilmente la reducción en uso de área del FPGA que implica el entrelazado de octavas, además de eso, se puede apreciar su menor tendencia al crecimiento ya que la recta que la describe tiene una menor pendiente. Las Figuras 7.8 (b) y (d) muestran el porciento de ahorro de registros y tablas de verdad, respectivamente, que provee el uso de esta técnica. Estos valores se encuentran casi todos por encima del 50 % y con una apreciable tendencia a crecer a medida que aumenta el número de octavas procesadas.

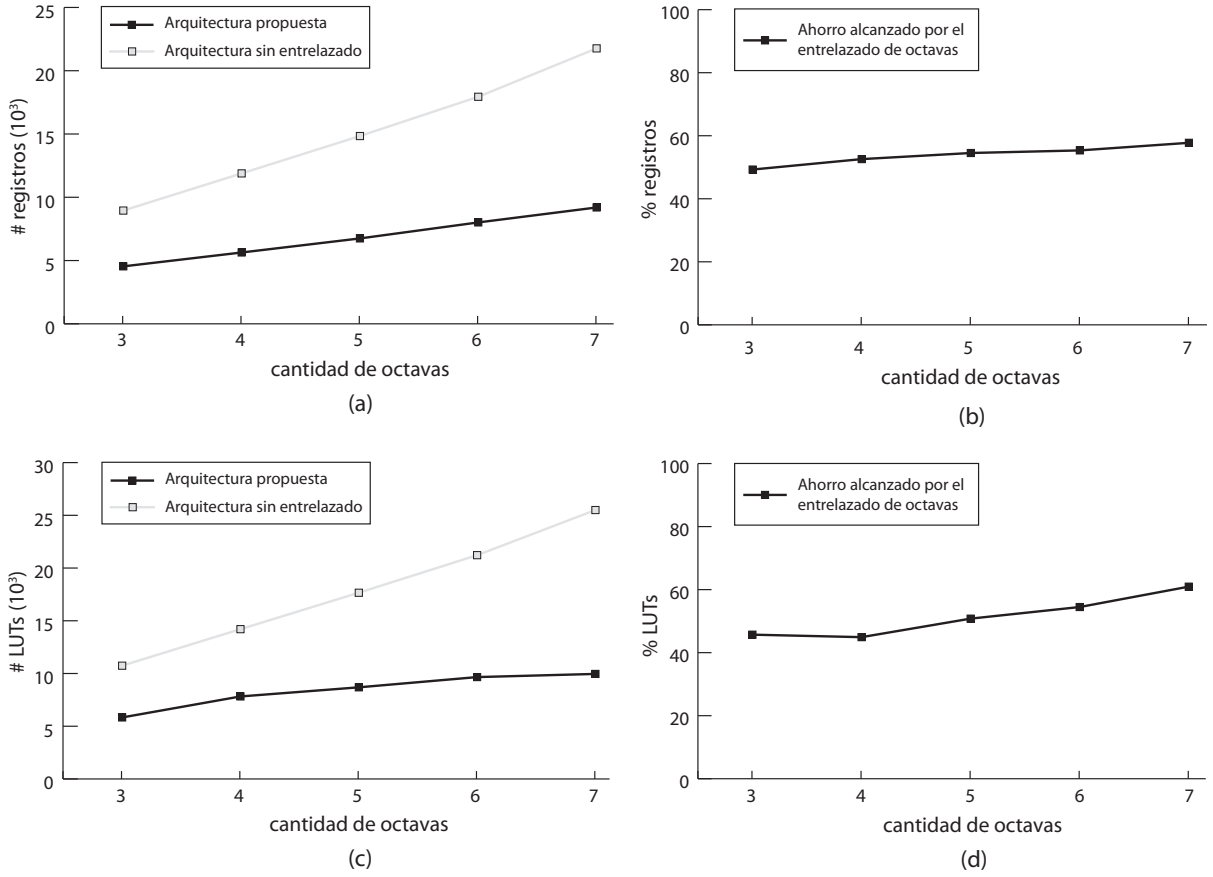


Figura 7.8: Ventajas en cuanto a área del dispositivo del uso del entrelazado del procesamiento de octavas. En (a) y (c) diferencias entre la arquitectura propuesta usando esta técnica y prescindiendo de ella. En (b) y (d) porcientos de ahorro en el uso de recursos de hardware.

7.5. Comparación con otras arquitecturas

En esta sección se comparan los resultados obtenidos por la arquitectura propuesta con los trabajos relacionados [3][24][23]. Con este fin, se sintetizó la arquitectura propuesta en un Xilinx Virtex II Pro (XC2VP30-5FF1152) con una configuración lo más cercana a la de estos trabajos ($M = 320, N = 240, O = 3, S = 6, k = 7$). Los resultados de la síntesis para esta configuración en este dispositivo son resumidos en la Tabla 7.1.

Tabla 7.1: Resultados de la síntesis de hardware de la arquitectura propuesta para una configuración de $M = 320, N = 240, O = 3, S = 6, k = 7$ en un Xilinx Virtex II Pro (XC2VP30-5FF1152).

Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	5,676	27,392	20 %
Number of 4 input LUTs	5,554	27,392	20 %
Logic Distribution			
Number of occupied Slices	4,393	13,696	32 %
Number of Slices containing only related logic	4,393	4,393	100 %
Number of Slices containing unrelated logic	0	4,393	0 %
Total Number of 4 input LUTs	6,699	27,392	24 %
Number used as logic	5,154		
Number used as a route-thru	1,145		
Number used as Shift registers	400		
Number of bonded IOBs	153	644	23 %
Number of RAMB16s	108	136	79 %
Number of BUFGMUXs	1	16	6 %

Luego de este proceso de síntesis también se determinó que la implementación podría funcionar a una frecuencia máxima de 145.122 MHz. Por lo tanto, dado que la arquitectura devuelve un resultado cada dos ciclos de reloj, para procesar una imagen de 320×240 píxeles el sistema se tomará 1,1 milisegundos, lo que permitirá procesar una secuencia de video a 900 cuadros por segundo (fps).

En la Tabla 7.2 se brinda una comparación de estos resultados con los obtenidos por otras arquitecturas reportadas en la literatura para la etapa de detección de puntos de interés del SIFT.

Como se puede observar en la Tabla 7.2, la máxima frecuencia a la que podría trabajar

Tabla 7.2: Comparación de la síntesis de hardware con los trabajos relacionados.

Parámetros de comparación	**Arquitectura propuesta**	**Bonato et al 2008 [3]**	**Qiu et al 2009 [24]**	**Qiu et al 2010 [23]**
Tamaño de la imagen	QVGA	QVGA	VGA	QVGA
Máx. frecuencia reloj	145.122 MHz	149.0 MHz	82.0 MHz	95.0 MHz
Potencia computacional (millones píxeles/seg)	72.6	149.0	5.1	15.3
Velocidad del proceso	900 fps	1940 fps	16 fps	81 fps
Registros	5676	7256	6333	6120
LUTs	5554	15137	5825	5011

el sistema es superior al resto de los trabajos a excepción del trabajo de Bonato et al, que tienen frecuencias máximas muy similares. Esta máxima frecuencia, combinada con el hecho de que el sistema devuelve un resultado cada dos ciclos de reloj le permite tener una gran velocidad de procesamiento de 900 fps, muy superior a [24][23]. El trabajo de Bonato et al para esta etapa del algoritmo logra devolver un resultado por ciclo de reloj, por lo que esta parte funcionando por separado logrará tener el doble de velocidad que nuestro sistema, aunque su arquitectura general tiene una restricción de 30 fps, la cual es introducida en otra etapa del algoritmo. La arquitectura propuesta en este trabajo logra la mitad de la potencia computacional que el trabajo de Bonato et al, esto se debe a que para lograr el entrelazado de las octavas es necesario dividir en dos la frecuencia de muestro, no obstante, se sacrifica la mitad de la potencia computacional para obtener una ventaja de casi tres veces en cuanto a área del dispositivo, que es el factor crítico. Este trabajo supera en varias veces la potencia computacional presentada por los trabajos de Qiu y colaborades. La arquitectura propuesta también ocupa menos área del dispositivo que las restantes, exceptuando la cantidad de LUTs en comparación con el trabajo de [23] donde la diferencia es muy pequeña. En adición, como se ha discutido en otras secciones, a medida que se procesen un mayor número de octavas, la tasa de aumento en el uso de área del dispositivo de nuestra arquitectura será menor, por lo que para mayor cantidad de octavas esta ventaja en el uso de los recursos de hardware de la arquitectura propuesta se hará mucho mayor.

7.6. Conclusiones del capítulo

En este capítulo se presentaron un conjunto de pruebas y experimentos realizados a la arquitectura aquí propuesta. El primer objetivo de estas pruebas era comprobar cuán semejantes eran los resultados obtenidos por ella en comparación con una implementación software. Se evidenciaron bajas tasas de error al comparar los espacios-escala Gaussianos generados por la arquitectura y una implementación software, además de errores promedio inferiores a un píxel en la localización de los puntos candidatos.

Estas primeras pruebas brindaban una medida del error cometido en los cálculos, pero no brindaban ninguna información acerca de cómo estos afectaban a las características SIFT en su uso en alguna aplicación. Con este objetivo, se llevaron a cabo un conjunto de pruebas para comprobar la variación en cuanto a repetitividad y distintividad de las características SIFT detectadas a partir de la arquitectura. Se tomaron en cuenta variaciones en la imágenes como punto de vista, rotación y escalamiento, compresión JPEG y desenfoque. Las diferencias en cuanto a tasas de correspondencias fueron pequeñas, detectándose un número suficiente de correspondencias de características entre las imágenes.

En capítulos anteriores se había mencionado la ventaja que implica la introducción del entrelazado de las octavas en cuanto a ahorro de área del dispositivo. También fueron llevados a cabo un conjunto de pruebas que mostraron cuantitativamente esta ventaja, obteniéndose ahorros en el uso de área del dispositivo por encima del 50 % y con tendencia al aumento mientras más octavas fuesen procesadas.

Finalmente, se compararon los resultados obtenidos por la arquitectura propuesta con el resto de las arquitecturas presentadas en la literatura con el mismo fin. La arquitectura propuesta mostró mejores indicadores de tiempo y eficiencia en el uso del área del dispositivo que el resto en casi todos los parámetros.

Capítulo 8

Conclusiones y trabajo futuro

8.1. Conclusiones

En la presente tesis se propuso una arquitectura hardware para la detección de puntos de interés del algoritmo SIFT.

Con el objetivo de sacar el máximo provecho del paralelismo de esta etapa del algoritmo y minimizar el área del dispositivo ocupada por su implementación en hardware, se reformuló parte del algoritmo. Dadas las características del tipo de procesamiento realizado en este algoritmo se tuvo en cuenta la potencialidad de la explotación del paralelismo de datos. Con el fin de disminuir la cantidad de operaciones de multiplicación-acumulación y gracias a la propiedad de separabilidad del kernel Gaussiano se propuso utilizar la convolución separable. Además, se planteó la posibilidad de entrelazar el procesamiento de las octavas, lo que permitiría realizar todas las operaciones de convolución correspondientes a una misma escala en una sola unidad de procesamiento.

El principal aporte de dicha arquitectura y del algoritmo que implementa, radica en que a medida que aumenta el número de octavas a procesar, la cantidad de área del dispositivo ocupada se mantiene casi constante, solamente aumentando en el número de bloques de memoria necesarios para almacenar las nuevas octavas y la lógica necesaria para controlar el entrelazado de más octavas. Este fenómeno se debe a que todas las octavas para una misma escala, sin importar la cantidad, siempre serán procesadas en el mismo bloque de convolución.

Las pruebas y experimentos realizados a la arquitectura aquí propuesta tenían como primer objetivo comprobar cuán semejantes eran los resultados obtenidos en comparación con una implementación software. Se reportaron bajas tasas de error en la generación de los espacios-escala Gaussianos, además de errores promedio menores de un píxel en la localización de los puntos candidatos. También, se llevaron a cabo un conjunto de pruebas para comprobar la variación en cuanto a repetitividad y distintividad de las características SIFT detectadas a partir de la arquitectura. Se tomaron en cuenta variaciones en la imágenes como punto de vista, rotación y escalamiento, compresión JPEG y desenfoque. Las diferencias en cuanto a tasas de correspondencias fueron pequeñas, detectándose un número suficiente de correspondencias de características entre las imágenes. Además, fueron llevados a cabo un conjunto de pruebas que mostraron cuantitativamente la ventaja introducida con el entrelazado del procesamiento de las octavas, obteniéndose ahorros en el uso de área del dispositivo por encima del 50 % y con tendencia al aumento mientras más octavas fuesen procesadas. Finalmente, se compararon los resultados obtenidos por la arquitectura propuesta con el resto de las arquitecturas presentadas en la literatura con el mismo fin. La arquitectura propuesta mostró mejores indicadores de tiempo y eficiencia en el uso del área del dispositivo que el resto en casi todos los paráme-tros. La arquitectura presentada en este trabajo, logra detectar los puntos de interés en una imagen a razón de un píxel cada dos ciclos de reloj. Implementada en un Xilinx Virtex II Pro, con una configuración de tres octavas y seis escalas, y una restricción de reloj de 145 MHz, una imagen de 320×240 es procesada en 1.1 ms, lo que representa una aceleración de 250x (dos órdenes de magnitud) con respecto a la implementación software de Vedaldi.

Con el desarrollo de esta arquitectura y los resultados obtenidos se considera que se han alcanzado los objetivos planteados al inicio de la tesis.

Los primeros resultados de este trabajo fueron presentados en el XIV Congreso Iberoamericano de Reconocimiento de Patrones, CIARP'09. Estos resultados fueron reportados en [6].

8.2. Trabajo futuro

Con base en los resultados obtenidos en esta tesis, existen algunas ideas que se pueden seguir como posibles trabajos futuros. Primeramente, implementar en hardware el resto de las etapas del SIFT o la etapa de generación de los descriptores que es la segunda de mayor costo computacional, para de esta manera obtener una mayor aceleración del algoritmo en general. También, resulta de interés explorar la posibilidad de acelerar mediante hardware otras variaciones del SIFT, que desde su concepción en software ya fueron pensados con el objetivo de acelerar este algoritmo, ya sea mediante aproximaciones o mediante sustitución de operaciones por otras equivalentes de menor costo computacional, como por ejemplo el SURF (del inglés, *Speeded Up Robust Features*)[2].

Anexos

A. Glosario de Términos

FPGA	*Field Programmable Gate Array*
SIFT	*Scale Invariant Features Transform*, nombre dado a un algoritmo de extracción de características locales
DoG	Diferencias de Gaussianas
espacio-escala	Espacio para la representación multi-escala de imágenes
octava	Se refiere a un intervalo del espacio-escala
escala	Se refiere a un subnivel dentro de una octava
bin	Un intervalo de un histograma
MAC	Multiplicación - Acumulación
convolución	Es una operación matemática sobre dos imágenes para formar una tercera, la cual es vista como una modificación de una de las imágenes originales
GPU	Unidades de procesado de gráficos (del inglés, *Graphics Processing Unit*)
ASIC	Circuitos integrados de aplicación específica (del inglés, *Application-Specific Integrated Circuits*)

PLD	Dispositivo lógico programable (del inglés, *Programmable Logic Device*)
LUT	Tabla de verdad (del inglés, *Look Up Table*)
RAM	Memoria de acceso aleatorio (del inglés, *Random Access Memory*)
GPP	Procesador de propósito específico o CPU (del inglés, *General Purpose Processor*)
VHDL	*VHSIC hardware description language*, nombre dado a un lenguaje de descripción de hardware
Verilog	Nombre dado a un lenguaje de descripción de hardware
DMA	Acceso directo a memoria (del inglés, *Direct Memory Access*
SLAM	Localización y mapeo simultáneo (del inglés, *Simultaneous Localization And Mapping*)
SPMD	Se refiere al paralelismo del tipo un solo proceso, múltiples datos (del inglés, *Single Process, Multiple Data*
1D	Se refiere a una dimensión
2D	Se refiere a dos dimensiones
MSE	Error cuadrado medio (del inglés, *Mean Squared Error*)

B. Trabajos publicados

Los primeros resultados de este trabajo fueron presentados en el XIV Congreso Iberoamericano de Reconocimiento de Patrones, CIARP'09:

- Leonardo Chang and José Hernández-Palancar. **A Hardware Architecture for SIFT Candidate Keypoints Detection.** In *CIARP '09: Proceedings of the 14th Iberoamerican Conference on Pattern Recognition*, pages 95–102, Berlin, Heidelberg, 2009. Springer-Verlag.

Referencias

[1] Atmel. 3x3 convolver with run-time reconfigurable vector multiplier in Atmel at6000 FPGAs. *[Online]. Available: http://atmel.com/dyn/resources/prod documents/DOC0764.PDF*, 1999.

[2] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-Up Robust Features (SURF). *Comput. Vis. Image Underst.*, 110(3):346–359, 2008.

[3] V. Bonato, E. Marques, and G. A. Constantinides. A Parallel Hardware Architecture for Scale and Rotation Invariant Feature Detection. *Circuits and Systems for Video Technology, IEEE Transactions on*, 18(12):1703–1712, 2008.

[4] Vanderlei Bonato, Eduardo Marques, and George A. Constantinides. A Parallel Hardware Architecture for Image Feature Detection. In *ARC '08: Proceedings of the 4th international workshop on Reconfigurable Computing*, pages 137–148, Berlin, Heidelberg, 2008. Springer-Verlag.

[5] Matthew Brown and David G. Lowe. Automatic panoramic image stitching using invariant features. *Int. J. Comput. Vision*, 74(1):59–73, 2007.

[6] Leonardo Chang and José Hernández-Palancar. A Hardware Architecture for SIFT Candidate Keypoints Detection. In *CIARP '09: Proceedings of the 14th Iberoamerican Conference on Pattern Recognition*, pages 95–102, Berlin, Heidelberg, 2009. Springer-Verlag.

[7] H. Djakou Chati, F. Muhlbauer, T. Braun, C. Bobda, and K. Berns. Hardware/Software co-design of a key point detector on FPGA. In *FCCM '07: Proceedings*

of the 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pages 355–356, Washington, DC, USA, 2007. IEEE Computer Society.

[8] Claudia Cruz, Luis Enrique Sucar, and Eduardo F. Morales. Real-time face recognition for human-robot interaction. In *8th IEEE International Conference on Automatic Face and Gesture Recognition, FG 2008*, pages 1–6. IEEE, 2008.

[9] André DeHon. The Density Advantage of Configurable Computing. *Computer*, 33(4):41–49, 2000.

[10] J. Evans. Efficient FIR filter architectures suitable for FPGA implementation. Online, 1994.

[11] Alex A. Freitas and S. H. Lavington. *Mining Very Large Databases with Parallel Processing*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[12] Michael Grabner, Helmut Grabner, and Horst Bischof. Fast Approximated SIFT. In P. J. Narayanan, Shree K. Nayar, and Heung-Yeung Shum, editors, *ACCV (1)*, volume 3851 of *Lecture Notes in Computer Science*, pages 918–927. Springer, 2006.

[13] Scott Hauck and André DeHon. *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation (Systems on Silicon)*. Morgan Kaufmann, November 2007.

[14] Martin C. Herbordt, Tom VanCourt, Yongfeng Gu, Bharat Sukhwani, Al Conti, Josh Model, and Doug DiSabello. Achieving High Performance with FPGA-Based Computing. *Computer*, 40(3):50–57, 2007.

[15] S. Heymann, B. Fröhlich, Fakultät Medien, K. Müller, and T. Wiegand. SIFT implementation and optimization for general-purpose GPU. In *In WSCG '07*, 2007.

[16] Yan Ke and R. Sukthankar. PCA-SIFT: a more distinctive representation for local image descriptors. In *2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 506–513, 2004.

[17] Marc Lalonde, David Byrns, Langis Gagnon, Normand Teasdale, and Denis Laurendeau. Real-time eye blink detection with GPU-based SIFT tracking. In *CRV '07: Proceedings of the Fourth Canadian Conference on Computer and Robot Vision*, pages 481–487, Washington, DC, USA, 2007. IEEE Computer Society.

[18] David G. Lowe. Object Recognition from Local Scale-Invariant Features. In *ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2*, page 1150, Washington, DC, USA, 1999. IEEE Computer Society.

[19] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[20] Krystian Mikolajczyk and Cordelia Schmid. A Performance Evaluation of Local Descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(10):1615–1630, 2005.

[21] A. C. Murillo, José Jesús Guerrero, and Carlos Sagüés. SURF features for efficient robot localization with omnidirectional images. In *ICRA*, pages 3901–3907, 2007.

[22] Niklas Pettersson and Lars Petersson. Online stereo calibration using FPGAs. In *Intelligent Vehicles Symposium, 2005. Proceedings. IEEE*, pages 55–60, 2005.

[23] J.B. Qiu, T. Huang, and T. Ikenaga. A 7-Round Parallel Hardware-Saving Accelerator for Gaussian and DoG Pyramid Construction Part of SIFT. In *ACCV09*, pages III: 75–84, 2010.

[24] Jingbang Qiu, Tianci Huang, and Takeshi Ikenaga. 1D-based 2D Gaussian Convolution Unit Based Hardware Accelerator for Gaussian & DoG Pyramid Construction in SIFT. *Proceedings of the IEICE General Conference*, 2009(2):178, 2009-03-04.

[25] Cordelia Schmid and Roger Mohr. Local grayvalue invariants for image retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(5):530–535, 1997.

[26] Stephen Se, Ho kong Ng, Piotr Jasiobedzki, and Tai jing Moyung. Vision based modeling and localization for planetary exploration rovers. In *55th International Astronautical Congress 2004*, 2004.

[27] Sudipta Sinha, Jan-Michael Frahm, Marc Pollefeys, and Yakup Genc. Feature tracking and matching in video using programmable graphics hardware. *Machine Vision and Applications*.

[28] Sudipta Sinha, Jan-michael Frahm, Marc Pollefeys, and Yakup Genc. GPU-based Video Feature Tracking and Matching. Technical report, 2006.

[29] Tinne Tuytelaars and Krystian Mikolajczyk. Local Invariant Feature Detectors: A Survey. *Foundations and Trends in Computer Graphics and Vision*, 3(3):177–280, 2007.

[30] S. Vadlamani and W. Mahmoud. Comparison of CORDIC algorithm implementations on FPGA families. In *System Theory, 2002. Proceedings of the Thirty Fourth Southeastern Symposium on*, pages 192–196, 2002.

[31] A. Vedaldi. An open implementation of the SIFT detector and descriptor. Technical Report 070012, UCLA CSD, 2007.