Universidad Tecnológica de la Mixteca

DISEÑO Y MODELADO DE UNA ARQUITECTURA HARDWARE DEL DESCRIPTOR DAISY Y SU IMPLEMENTACIÓN EN LÓGICA RECONFIGURABLE

T E S I S

QUE PARA OBTENER EL GRADO DE

MAESTRO EN ELECTRÓNICA, SISTEMAS INTELIGENTES APLICADOS

PRESENTA:

ING. ALBERTO DE JESÚS SANTOS VILLALOBOS

DIRIGIDA POR:

DR. ENRIQUE GUZMÁN RAMÍREZ

FEBREREO DE 2020 **HUAJUAPAN DE LEÓN, OAXACA**

Resumen

El presente trabajo de tesis aborda el diseño y modelado de una arquitectura hardware para el descriptor DAISY y su implementación sobre lógica reconfigurable, cuyo elemento central de procesamiento es un arreglo de compuertas programables en campo (*Field Programmable Gate Array, FPGA*).

La implementación en hardware del descriptor DAISY se realizó a partir de su modelado conceptual sobre un leguaje de alto nivel, el cual fue implementado como parte de la presente investigación con la finalidad de identificar las partes que componen al descriptor DAISY y, por lo tanto, diseñar y modelar con eficiencia la arquitectura hardware idónea para el algoritmo. Como parte del modelado conceptual, se implementaron las Memorias Asociativas Extendidas (MAE) y una interfaz gráfica de usuario (*Graphical User Interface, GUI*), cuya finalidad es de realizar la evaluación del algoritmo, orientada al reconocimiento de objetos empleando la base de datos ALOI (*Amsterdam Library of Object Image*). El objetivo de la evaluación fue determinar la configuración en los parámetros del algoritmo que ofrecen mejores resultados. Ademas, de identificar cual(es) de los operadores utilizados en las MAE muestran un alto desempeño.

Después, la arquitectura hardware del descriptor DAISY es modelada por fases mediante el lenguaje VHDL (*Very High Speed Hardware Description Language*) para la configuración de los parámetros identificados en el modelado conceptual, empleando técnicas de modelado que permiten optimizar el uso de los recursos del FPGA.

Finalmente, la implementación en hardware del descriptor DAISY es sometida a una evaluación orientada al reconocimiento de objetos para verificar su desempeño en el FPGA, empleando la base ALOI y las MAE, siendo los operadores prom y pmed los que mejor desempeño mostraron para dicha evaluación.

Los resultados obtenidos en esta evaluación muestran que la implementación en hardware del descriptor DAISY se desempeñó como se esperaba, logrando altos porcentajes de reconocimiento. Como parte de estos resultados, se incluye una comparativa de los tiempos de procesamiento y porcentajes de reconocimiento promedios, entre el modelado conceptual y la arquitectura hardware.

Abstract

The present thesis approaches the design and modeling of a hardware architecture for the DAISY descriptor and its implementation on reconfigurable logic, whose central processing element is a field programmable gate array (FPGA).

The hardware implementation of the DAISY descriptor is preceded by its conceptual modeling by a high-level language, which was implemented as part of this research in order to identify the parts that make up it and, therefore, to design and model efficiently the suitable hardware architecture. As part of conceptual modeling, extended associative memories (EAM) and a graphical user interface were implemented, whose purpose is to perform the evaluation of the algorithm, object recognition-oriented using the amsterdam library of object Image (ALOI) database. The purpose of this was to determine the configuration in the algorithm parameters that provide the best results. In addition, to identify which of the operators used in the memories show high performance.

The hardware architecture is then phased modeled using the very high-speed hardware description language, specifically for the parameters found, using modeling techniques to optimize the use of FPGA resources.

Finally, the hardware implementation of the algorithm is subjected to an object recognition-oriented assessment to verify its performance in the FPGA, using the ALOI and the EAM, being the operators who performed best for this, pmed and prom, with which the evaluation was carried out.

The results show that the hardware implementation of the DAISY descriptor performed as expected, achieving high recognition rates. As part of these results, a comparison of processing times and average recognition percentages, between conceptual modeling and hardware architecture were included.

Agradecimientos

A mi director de tesis, el Dr. Enrique Guzmán Ramírez, por creer en mi desde el primer momento, por su ayuda y paciencia a lo largo del desarrollo de este proyecto.

A mis sinodales, por contribuir en las mejoras del presente documento.

Dedicatoria

Con todo mi amor a mi esposa Mary,

A mis padres y hermanos,

Por siempre apoyarme cuando más los necesito.

Índice general

Resumen I
Abstract III
Agradecimientos V
Dedicatoria VII
Índice general IX
Índice de figuras XI
Índice de tablas XIII
Capítulo 1 Introducción 1
1.1 Planteamiento del problema 2
1.2 Justificación 3
1.3 Hipótesis 4
1.4 Objetivos 4
1.5 Descripción de la arquitectura hardware 5
1.6 Estructura del documento 6
Capítulo 2 Marco teórico 7
2.1 Introducción 7
2.2 Reconocimiento de objetos 8

2.2.1 Adquisición de la imagen 9
2.2.2 Preprocesamiento 10
2.2.3 Segmentación 10
2.3 Extracción de características 11
2.3.1 Descriptor DAISY 14
2.4 Clasificación de patrones 16
2.4.1 Memorias Asociativas 17
2.4.2 Memorias asociativas extendidas 19
2.5 Arreglo de Compuertas Programables en Campo 22
2.6 Estado del arte 24
Capítulo 3 Modelado conceptual del descriptor DAISY 28
3.1 Modelado conceptual del descriptor DAISY 28
3.1.1 Generación de los mapas de orientación 29
3.1.2 Generación de los mapas de orientación convolucionados 32
3.1.3 Generación de los vectores 39
3.1.4 Generación del vector de características 44
3.2 Modelado conceptual de las MAE 45
3.2.1 Fase de aprendizaje de una MAE 46
3.2.2 Fase de clasificación de una MAE 48
3.3 Interfaz de usuario 49
3.4 Resultados experimentales 50
3.4.3 Evaluación del modelado conceptual 52
Capítulo 4 Arquitectura hardware del descriptor DAISY 64
4.1 Diseño de la arquitectura hardware del descriptor DAISY 64
4.1.1 Generación de los mapas de orientación 65
4.1.2 Generación de los mapas de orientación convolucionados 70
4.1.3 Generación de los vectores 75
4.1.4 Generación del vector de características 84
4.2 Resultados experimentales 85
4.2.1 Evaluación de la arquitectura hardware 85
Capítulo 5 Conclusiones y trabajos futuros 100
5.1 Conclusiones 100
5.2 Trabajos futuros 102
Referencias 103

Índice de figuras

Figura 1.1 Diagrama a bloques de la arquitectura hardware del descriptor DAISY. 5
Figura 2.1 Esquema clásico de un SRO. 9
Figura 2.2 Aproximación discreta de una imagen. 10
Figura 2.3 Segmentación de objetos por color. 11
Figura 2.4 Representación de la obtención de características, a) globales y b) locales. 12
Figura 2.5 Cálculo conceptual del descriptor DAISY. 14
Figura 2.6 Descriptor DAISY. 15
Figura 2.7 Esquema general de una memoria asociativa. 18
Figura 3.1 Generación de los mapas de orientación. 29
Figura 3.2 Gradientes de la imagen original: a) gradiente en x y b) gradiente en y. 30
Figura 3.3 Máscaras: a) en x y b) en y. 30
Figura 3.4 Mapas de orientación: a) 0°, b) 60°, c) 120°, d) 180°, e) 240° y f) 300°. 32
Figura 3.5 Mapa central. 32
Figura 3.6 Proceso de convolución 2D. 35
Figura 3.7 Proceso de convolución 1D. 36
Figura 3.8 Mapas de orientación convolucionados del primer anillo: a) 0°, b) 60°, c) 120°, d) 180°, e) 240° y f) 300°. 37
Figura 3.9 Mapas de orientación convolucionados del segundo anillo: a) 0°, b) 60°, c) 120°, d) 180°, e) 240° y f) 300°. 38
Figura 3.10 Mapa central convolucionado. 38
Figura 3.11 Puntos de interés y *centros* de los anillos para el descriptor DAISY. 39
Figura 3.12 Ubicaciones de los puntos de interés para formar los vectores: a) $\mathbf{h}_{\Sigma_0}$ y b) $\mathbf{h}_{\Sigma_1}$ 42
Figura 3.13 Generación del vector h_{Σ_c}. 43

Figura 3.14 Estructura secuencial del modelado conceptual de las MAE. 45
Figura 3.15 Generación de **M** por el operador prom. 47
Figura 3.16 Obtención del índice i de la clase asociada. 48
Figura 3.17 GUI: a) Ventana principal, b) Visualización de resultados, c) Ventana de configuración de las MAE y d) Ventana para la configuración de DAISY. 49
Figura 3.18 Base de datos ALOI. 50
Figura 3.19 Objetos usados para el primer y segundo experimento. 52
Figura 3.20 Objetos usados para el tercer y cuarto experimento. 53
Figura 3.21 Objetos usados para el quinto y sexto experimento. 55
Figura 3.22 Objetos usados para el séptimo y octavo experimento. 56
Figura 3.23 Objetos usados para el noveno y décimo experimento. 59
Figura 4.1 Arquitectura general del descriptor DAISY. 65
Figura 4.2 Ventana deslizante basada en buffers de línea. 66
Figura 4.3 Funcionamiento de la VDBBL. 66
Figura 4.4 VDBBL para el obtener las derivadas de I. 67
Figura 4.5 Arquitectura hardware para obtener las derivadas: a) x y b) y. 67
Figura 4.6 Derivadas de I: a) x y b) y. 67
Figura 4.7 Arquitectura hardware para la obtención de los mapas de orientación. 68
Figura 4.8 Mapas de orientación: a) 0°, b) 60°, c) 120°, d) 180°, e) 240° y f) 300°. 69
Figura 4.9 Arquitectura hardware para la obtención del mapa central. 69
Figura 4.10 Mapa central. 69
Figura 4.11 VDBBL para acceder a los píxeles de los mapas de orientación. 71
Figura 4.12 Arquitectura hardware general para la convolución. 72
Figura 4.13 Mapas de orientación convolucionados del primer anillo: a) 0°, b) 60°, c) 120°, d) 180°, e) 240° y f) 300°. 72
Figura 4.14 VDBBL para acceder a los píxeles de los mapas de orientación convolucionados del primer anillo. 73
Figura 4.15 Mapas de orientación convolucionados del segundo anillo: a) 0°, b) 60°, c) 120°, d) 180°, e) 240° y f) 300°. 73
Figura 4.16 VDBBL para acceder a los píxeles del mapa central. 74
Figura 4.17 Mapa central convolucionado. 75
Figura 4.18 Arquitectura hardware para la generación del vector de características. 84
Figura 4.19 Objetos usados para el primer y segundo experimento. 85
Figura 4.20 Objetos usados para el tercer y cuarto experimento. 86
Figura 4.21 Objetos usados para el quinto y sexto experimento. 87
Figura 4.22 Objetos usados para el séptimo y octavo experimento. 89
Figura 4.23 Objetos usados para el noveno y décimo experimento. 91
Figura 4.24 Arquitectura hardware secuencial de las fases 1 y 2. 97
Figura 4.25 Arquitectura hardware concurrente de las fases 1 y 2. 98

Índice de tablas

Tabla 2.1 Parámetros DAISY. 16
Tabla 3.1 Orientaciones en grados y radianes de los mapas de orientación. 31
Tabla 3.2 Coordenadas polares y rectangulares de los *centros* $P_{o_j}^{k}(R_k, o_j)$ 40
Tabla 3.3 Matriz de confusión de ejemplo. 51
Tabla 3.4 Matriz de confusión del primer experimento usando el operador pmed. 52
Tabla 3.5 Resultados del primer experimento usando el operador pmed. 53
Tabla 3.6 Matriz de confusión del segundo experimento usando el operador prom. 53
Tabla 3.7 Resultados del segundo experimento usando el operador prom. 53
Tabla 3.8 Matriz de confusión del tercer experimento usando el operador pmed. 54
Tabla 3.9 Resultados del tercer experimento usando el operador pmed. 54
Tabla 3.10 Matriz de confusión del cuarto experimento usando el operador prom. 54
Tabla 3.11 Resultados del cuarto experimento usando el operador prom. 54
Tabla 3.12 Matriz de confusión del quinto experimento usando el operador pmed. 55
Tabla 3.13 Resultados del quinto experimento usando el operador pmed. 55
Tabla 3.14 Matriz de confusión del sexto experimento usando el operador prom. 56
Tabla 3.15 Resultados del sexto experimento usando el operador prom 56
Tabla 3.16 Matriz de confusión del séptimo experimento usando el operador prom. 57
Tabla 3.17 Resultados del séptimo experimento usando el operador prom. 57
Tabla 3.18 Matriz de confusión del octavo experimento usando el operador pmed. 58
Tabla 3.19 Resultados del octavo experimento usando el operador pmed. 58
Tabla 3.20 Matriz de confusión del noveno experimento usando el operador pmed. 60

Tabla 3.21 Resultados del noveno experimento usando el operador pmed. 61
Tabla 3.22 Matriz de confusión del décimo experimento usando el operador prom. 62
Tabla 3.23 Resultados del décimo experimento usando el operador prom. 63
Tabla 4.1 Aproximaciones de cos y sen en formato de punto fijo por cada o_j. 68
Tabla 4.2 Representación en punto fijo de los coeficientes del filtro gaussiano para el primer anillo. 70
Tabla 4.3 Representación en punto fijo de los coeficientes del filtro gaussiano para el segundo anillo. 71
Tabla 4.4 Aproximaciones de los coeficientes del filtro gaussiano del mapa central. 74
Tabla 4.5 Coordenadas rectangulares de los *centros* en el primer anillo. 75
Tabla 4.6 Direcciones y uso de memoria del primer anillo. 76
Tabla 4.7 Direcciones de memoria de los puntos de interés que forman al vector $\mathbf{v}_0^{P_0^0}$ 77
Tabla 4.8 Direcciones de memoria de los puntos de interés que forman al vector $\mathbf{v}_1^{P_{\pi/3}^0}$. 77
Tabla 4.9 Direcciones de memoria de los puntos de interés que forman al vector $\mathbf{v}_2^{P_{2\pi/3}^0}$ 78
Tabla 4.10 Direcciones de memoria de los puntos de interés que forman al vector $\mathbf{v}_3^{P_{\pi}^0}$ 78
Tabla 4.11 Direcciones de memoria de los puntos de interés que forman al vector $\mathbf{v}_4^{P_{4\pi/3}^0}$ 78
Tabla 4.12 Direcciones de memoria de los puntos de interés que forman al vector $\mathbf{v}_5^{P_{5\pi/3}^0}$ 79
Tabla 4.13 Coordenadas rectangulares de los *centros* en el segundo anillo. 79
Tabla 4.14 Direcciones y uso de memoria del segundo anillo. 80
Tabla 4.15 Direcciones de memoria de los puntos de interés que forman al vector $\mathbf{v}_0^{P_0^1}$ 80
Tabla 4.16 Direcciones de memoria de los *centros* que forman al vector $\mathbf{v}_1^{P_{\pi/3}^1}$ 81
Tabla 4.17 Direcciones de memoria de los *centros* que forman al vector $\mathbf{v}_2^{P_{\pi/3}^1}$ 81
Tabla 4.18 Direcciones de memoria de los *centros* que forman al vector $\mathbf{v}_3^{P_{\pi}^1}$. 81
Tabla 4.19 Direcciones de memoria de los *centros* que forman al vector $\mathbf{v}_4^{P_{4\pi/3}^1}$. 82
Tabla 4.20 Direcciones de memoria de los puntos de interés que forman al vector $\mathbf{v}_5^{P_{5\pi/3}^1}$ 82
Tabla 4.21 Direcciones de memoria de los puntos de interés que forman al vector $\mathbf{h}_{\Sigma_c}$. 83
Tabla 4.22 Matriz de confusión del primer experimento usando el operador pmed. 85
Tabla 4.23 Resultados del primer experimento usando el operador pmed. 85
Tabla 4.24 Matriz de confusión del segundo experimento usando el operador prom. 86
Tabla 4.25 Resultados del segundo experimento usando el operador prom. 86
Tabla 4.26 Matriz de confusión del tercer experimento usando el operador pmed. 86
Tabla 4.27 Resultados del tercer experimento usando el operador pmed. 87
Tabla 4.28 Matriz de confusión del cuarto experimento usando el operador prom. 87
Tabla 4.29 Resultados del cuarto experimento usando el operador prom. 87
Tabla 4.30 Matriz de confusión del quinto experimento usando el operador pmed. 88
Tabla 4.31 Resultados del quinto experimento usando el operador pmed. 88
Tabla 4.32 Matriz de confusión del sexto experimento usando el operador prom. 88
Tabla 4.33 Resultados del sexto experimento usando el operador prom. 89
Tabla 4.34 Matriz de confusión del séptimo experimento usando el operador pmed. 89

Tabla 4.35 Resultados del séptimo experimento usando el operador pmed. 90
Tabla 4.36 Matriz de confusión del octavo experimento usando el operador prom. 90
Tabla 4.37 Resultados del octavo experimento usando el operador prom. 91
Tabla 4.38 Matriz de confusión del noveno experimento usando el operador pmed. 92
Tabla 4.39 Resultados del noveno experimento usando el operador pmed. 93
Tabla 4.40 Matriz de confusión del décimo experimento usando el operador prom. 94
Tabla 4.41 Resultados del décimo experimento usando el operador prom. 95
Tabla 4.42 Porcentajes de reconocimiento promedio por operador. 95
Tabla 4.43 Resumen de utilización del dispositivo por la arquitectura hardware del descriptor DAISY. 96
Tabla 4.44 Tiempo en ms para generar el vector de características. 96
Tabla 4.45 Tiempos de procesamiento por fase del descriptor DAISY. 99

Capítulo 1 Introducción

En la actualidad los sistemas de reconocimiento de objetos (SRO) cotidianos son producto de los avances tecnológicos que se tienen. El interés por automatizar los procesos de reconocimiento de objetos ha ido fluyendo de una manera paulatina, ayudándose en la mayoría de los casos por hardware especializado.

Un SRO tiene por objetivo permitir a una máquina interactuar eficientemente con el entorno al proporcionarle ciertas habilidades que le permitan detectar y determinar la identidad de los objetos de interés [1]. El SRO tiene una amplia gama de aplicaciones en diversas áreas, tales como: detección de rostros en sistemas de vigilancia [2], sistemas para detección de frutas [3], detección de emociones [4], sistema para detección de tumores en imágenes gastroscópicas [5], robótica móvil [6], etc.

En este sentido y considerando un aspecto general, el presente trabajo de tesis tiene por objetivo abordar el problema que surge cuando una imagen contiene uno o varios objetos que son de interés, y el SRO tiene que ser capaz de identificar y clasificar solo estos objetos utilizando una base de datos, previamente construida, asignándoles las etiquetas o clases a las que pertenecen. Además, es deseable que el sistema cumpla diversas características, tales como: ser robusto a la alta variabilidad en objetos de la misma clase, sea altamente discriminante y preciso ante la presencia de objetos con diferentes tipos de forma, color y textura, tiene que ser robusto a invariancias (por ejemplo, traslación, rotación, escalado, etc.), debe presentar una baja

complejidad computacional debido a que se requiere que el SRO sea utilizado en sistemas autónomos que trabajen en tiempo real, etc.

Ahora, considerando un aspecto particular, este trabajo de tesis se centra en el estudio de la fase de extracción de características de un SRO mediante el diseño y modelado de una arquitectura hardware de un descriptor que sus autores han denominado descriptor local rápido para emparejamiento denso (DAISY) [7], y su respectiva implementación en lógica reconfigurable. La lógica reconfigurable, representada por el FPGA, tiene como principal ventaja la concurrencia, es decir, son dispositivos que son capaces de ejecutar dos o más procesos en forma simultánea. Esta característica los hace una tecnología superior a aquellas que operan con un enfoque secuencial, como los procesadores. El presente trabajo incluirá una GUI, desarrollada en un lenguaje de alto nivel, que permitirá comunicarse con la arquitectura hardware del descriptor DAISY implementada en el FPGA. La GUI tiene por objetivo permitir evaluar el desempeño de la arquitectura hardware del descriptor DAISY, mediante simples tareas de interacción, tales como, enviar una imagen al FPGA para que sea procesada por la arquitectura implementada, visualizar los resultados generados por cada una de las fases de la arquitectura hardware del descriptor DAISY, obtener el vector de características y utilizarlo en procesos de clasificación implementados en la GUI.

La base de datos ALOI fue utilizada en las pruebas aplicadas a la arquitectura hardware del descriptor DAISY.

1.1 Planteamiento del problema

Un sistema de reconocimiento de objetos tiene por objetivo permitir a una máquina interactuar eficientemente con su entorno al dotarla con habilidades que le permitan detectar y determinar la identidad de los objetos de su interés. En este sentido, de acuerdo con Tou J. T. y González R. C., el reconocimiento de objetos es definido como la categorización de datos de entrada en clases identificadas, utilizando para este propósito la extracción de características significativas o atributos de los datos extraídos de un medio ambiente que contiene detalles irrelevantes [1].

De acuerdo con lo mencionado y considerando el aspecto general, el presente trabajo de investigación se enfrenta a un problema de etiquetado que se basa en modelos de objetos conocidos. Dicho de otra manera, el problema consiste en, dada una imagen que contiene uno o más objetos de interés (y el fondo de la imagen) y un conjunto de etiquetas, una para cada modelo conocido por el sistema, el sistema debería asignar etiquetas correctas a regiones o conjunto de regiones similares en la imagen.

Ahora, el punto central de esta investigación considera únicamente la implementación de algoritmos pertenecientes a la fase de extracción de características de un sistema de reconocimiento de objetos y su utilización en aplicaciones en tiempo real sobre sistemas autónomos. Entonces, considerando el aspecto particular de esta investigación se pueden identificar dos problemas a resolver:

Por un lado, y debido a que no existe algún sistema eficaz que reconozca todo tipo de objetos en cualquier ambiente, es necesario definir algoritmos en la fase de extracción de características que ofrezcan las siguientes características:

- Robustos a la alta variabilidad de la apariencia de objetos del mismo tipo. Pueden existir objetos del mismo tipo con gran diversidad de forma, color y textura, además múltiples factores como la posición, la iluminación, las oclusiones, ruido, entre otras, pueden aumentar estas diferencias.
- Robustos a la carencia de imágenes tomadas como referencia durante la fase de entrenamiento. Los pocos datos disponibles generalmente no son suficientes para cubrir la variabilidad en apariencia. Aunado a esto, pueden existir diferencias significativas en las condiciones del entrenamiento y de la operación del sistema.
- Posean una baja complejidad computacional. Debido a que se busca sean utilizados en aplicaciones en tiempo real.

Por otro lado, y debido a que se requiere que el sistema lleve a cabo su tarea en el menor tiempo posible y sea utilizado en sistemas autónomos, se requiere diseñar y modelar una arquitectura hardware del algoritmo de extracción de características elegido que ofrezca un alto desempeño computacional y minimice los recursos requeridos para cumplir su función.

1.2 Justificación

Es común que un sistema autónomo requiera de un método que le permita percibir su entorno para poder interactuar con él. Diversos dispositivos pueden ser utilizados como elemento central de procesamiento del sistema mencionado. Una computadora personal resulta poco práctica para cumplir esta tarea, debido a que se trata de un sistema completo donde la mayor parte de sus recursos no serían utilizados. Un microcontrolador o un procesador digital de señales (DSP, *Digital Signal Processor*) pueden ser buenas alternativas, sin embargo, debido a que estos dispositivos presentan un paradigma secuencial, su operación se lleva a cabo mediante la ejecución secuencial de instrucciones, la velocidad de procesamiento del sistema se ve afectada drásticamente, esto se debe a que, si el número de instrucciones necesarias en la implementación del algoritmo es alto, el tiempo de procesamiento también crecerá.

La lógica reconfigurable, tecnología utilizada en este trabajo de tesis y cuyo principal representante es el arreglo de compuertas programable en el campo FPGA posee una estructura que le permite aplicar un paradigma concurrente en la consecución de su tarea, es decir, un FPGA es capaz de evaluar varias hipótesis en forma simultánea. Quizá esta sea la característica más importante de un FPGA ya que permite aprovechar aspectos presentes en el algoritmo a implementar, tales como la no-dependencia existente entre los elementos hardware que lo forman, diversos grados de paralelismo, etc. Es claro que explotar esta característica afecta directamente la velocidad de procesamiento del sistema.

Dentro del área del reconocimiento de objetos, la representación de los objetos juega un papel fundamental para obtener un buen desempeño en la tarea de reconocimiento. Técnicas basadas en modelos de características generan representaciones a partir de la apariencia de regiones

locales de la imagen; con respecto a métodos que obtienen representaciones globales, estos métodos son más robustos a condiciones como la orientación del objeto, el tamaño, variaciones de iluminación y al ruido presente en la imagen, y quizá más importante, estos métodos son computacionalmente menos costosos. Los descriptores SIFT (*Scale Invariant Feature Transform*) [8], HOG (*Histogram of Oriented Gradients*) [9] y SURF (*Speeded Up Robust Features*) [10], son de los métodos que generan representaciones locales más comúnmente utilizados en la etapa de extracción de características de un SRO.

El descriptor DAISY es otro método que entra en esta categoría. El criterio en el cual se apoya la elección de DAISY para este trabajo de investigación es el bajo costo computacional que posee y que representa la principal ventaja con los métodos anteriormente mencionados. Esta característica de DAISY permite generar arquitecturas hardware más simple lo que repercute en un ahorro de recursos, punto medular cuando el diseño se planea hacer en un FPGA.

1.3 Hipótesis

La implementación en hardware del algoritmo de extracción de características DAISY puede ofrecer un alto desempeño con un costo de recursos moderado en aplicaciones de reconocimiento de objetos para sistemas autónomos.

1.4 Objetivos

El presente trabajo de investigación tiene por objetivo diseñar y modelar una arquitectura hardware de un sub-sistema de extracción de características de objetos inmersos en imágenes, utilizando el descriptor DAISY, y su implementación en lógica reconfigurable.

Para cumplir con el objetivo planteado, los siguientes objetivos secundarios son necesarios:

1. Modelado conceptual del descriptor DAISY.
2. Proponer y modelar una arquitectura hardware para el descriptor DAISY.
3. Desarrollar una GUI y establecer un método que permita evaluar la arquitectura hardware propuesta.

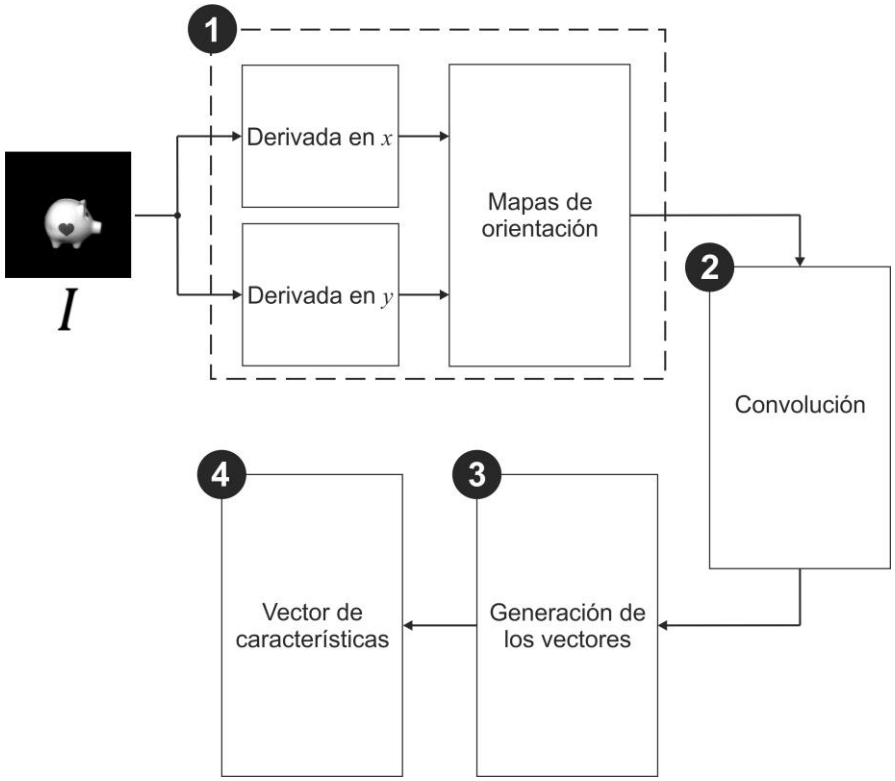


Figura 1.1 Diagrama a bloques de la arquitectura hardware del descriptor DAISY.

1.5 Descripción de la arquitectura hardware

La arquitectura hardware implementada en este trabajo de tesis está representada por la Figura 1.1, que muestra de manera general las fases que conforman a la arquitectura hardware. A continuación, se describen brevemente la función de cada una de las fases.

Fase 1. **Generación de los mapas de orientación**. En esta fase del descriptor primero son obtenidas las derivadas respecto a x y y de la imagen de entrada I y posteriormente los mapas de orientación, G_{o_j}, y el mapa central, G_c, son calculados empleando estas derivadas.

Fase 2. **Generación de los mapas de orientación convolucionados**. La tarea de esta fase es generar los mapas de orientación convolucionados, $\mathrm{G}_{o_j}^{\Sigma_k}$, a partir de la convolución de los G_{o_j}, con varios filtros gaussianos, G_{Σ_k}, donde cada Σ_k es de diferente valor. Además, se obtiene el mapa central convolucionado, $\mathrm{G}_c^{\Sigma_c}$, derivado de la convolución del G_c y el filtro gaussiano, G_{Σ_c}.

Fase 3. **Generación de los vectores**. El objetivo de esta fase es formar los vectores a partir de los $\mathrm{G}_{o_j}^{\Sigma_k}$ y del $\mathrm{G}_c^{\Sigma_c}$ con ciertos valores que contienen información relevante del objeto.

Fase 4. **Generación del vector de características**. En esta fase se forma el vector de características final del descriptor DAISY, mediante la normalización de los vectores y su concatenación.

1.6 Estructura del documento

La estructura del presente documento de investigación se distribuye en 5 capítulos, los cuales se describen brevemente a continuación.

Capítulo 1. **Introducción**. Capítulo presente, en el cual se da una breve presentación del trabajo de investigación realizado, así como de explicar por qué surge la necesidad de desarrollar la presente investigación y lo que se pretende alcanzar, mediante los objetivos planteados.

Capítulo 2. **Marco teórico**. Este capítulo aborda los temas necesarios para desarrollar la presente investigación, tales como: reconocimiento de objetos, se enfatiza la etapa de extracción de características en donde se explica formalmente el funcionamiento del descriptor DAISY. Así mismo, se aborda el tema de las MAE, el cual servirá de base para realizar su modelado conceptual. Tambien se incluye el tema de los FPGAs, para terminar, citando algunos trabajos relacionados con el presente.

Capítulo 3. **Modelado conceptual del descriptor DAISY**. En este capítulo se explica detalladamente la implementación secuencial realizada del descriptor DAISY por fases, en las cuales se incluyen resultados parciales. De igual manera, se explica el diseño de la GUI que servirá como herramienta para la evaluación del descriptor DAISY junto con las MAE, cuya implementación secuencial se describe a detalle. Al final del capítulo se incluyen los resultados experimentales aplicados empleando la base de datos ALOI.

Capítulo 4. **Arquitectura hardware del descriptor DAISY**. Este capítulo describe a detalle el diseño y modelado de la arquitectura hardware del descriptor DAISY por fases. Al final del capítulo se presentan los resultados experimentales que fueron aplicados a la implementación en hardware del algoritmo, utilizando la misma base de datos del modelado conceptual.

Capítulo 5. **Conclusiones y trabajos futuros**. En este capítulo se exponen las conclusiones que se obtuvieron durante el proceso de desarrollo de la presente investigación. Tambien se incluyen las propuestas de trabajos a futuro que den continuidad a este trabajo de tesis.

Al final del presente trabajo de tesis, se incluyen las referencias bibliográficas que se consultaron para construir la base teórica de la investigación.

Capítulo 2 Marco teórico

El presente capítulo presenta los fundamentos teóricos requeridos para el desarrollo de este trabajo de tesis. El capítulo inicia describiendo concisamente qué es el reconocimiento de objetos, los objetivos que busca cumplir y las etapas que integran a un SRO. Posteriormente, se concentra en explicar en forma precisa los métodos que son pieza clave en el desarrollo de esta investigación, el descripto DAISY, las MAE y la lógica reconfigurable. Con la finalidad de poner en contexto el trabajo presentado es este escrito, al final del capítulo se incluye el correspondiente estado del arte.

2.1 Introducción

La visión artificial o visión por computadora es una disciplina científica que intenta emular los mecanismos sensoriales de visión de un organismo biológico, con el objetivo de dotar a un sistema artificial con la capacidad de percibir, interpretar e interactuar con el entorno.

Estas tareas son desarrolladas mediante la deducción de la estructura y las propiedades del mundo tridimensional a partir de una o más imágenes bidimensionales. En este sentido, Szeliski, R. define a la visión artificial como el proceso que describe el mundo que se percibe a través de una o más imágenes mediante la reconstrucción de sus propiedades, como la forma, la iluminación y las distribuciones de color [11]. Estas propiedades se relacionan directamente con información espacial, espectral y temporal de los objetos incluidos en una imagen. La forma y la posición son propiedades espaciales, las cuales indican si el objeto está representado en una, dos o tres dimensiones. La información espectral como la frecuencia y la intensidad se relacionan con las propiedades de color y los tonos de gris de los objetos respectivamente. Finalmente, aspectos

estacionarios y dependientes del tiempo, como la presencia y/o ausencia de objetos, la aparición de eventos y el movimiento, son propiedades temporales.

Actualmente, la visión artificial se está utilizando en una amplia variedad de aplicaciones del mundo real, por ejemplo, inspección de maquinaria, reconocimiento óptico de caracteres, construcción de modelos 3D, usos médicos y biomédicos, detección de obstáculos en automóviles autónomos, vigilancia, captura de movimiento, biometría, entre muchos más.

En [12] presentan una forma de clasificar las aplicaciones de la visión artificial en función del tipo de tarea realizada. Estos tipos de tarea son, la medición o calibración, la detección de fallas, la verificación, el reconocimiento de objetos, la identificación, el análisis de localización y la guía.

De las tareas mencionadas, el reconocimiento de objetos resulta ser la más importante, incluso imprescindible para que algunas de las demás tareas cumplan su función. El reconocimiento de objetos es también parte central de la presente investigación. Razón por la cual los apartados siguientes profundizan en este tema.

2.2 Reconocimiento de objetos

El reconocimiento de objetos es un área de investigación muy activa en la última década, impulsada por el desarrollo de nuevas tecnologías, tanto en el campo de cómputo como en el de la electrónica. Es necesario enfatizar que el reconocimiento de objetos sigue siendo un gran desafío para investigadores que cultivan esta área. Dos razones son las causantes de esta situación, la primera es la gran variación que los objetos pueden presentar en su apariencia, debiéndose a causas como transformaciones visuales, diferentes poses, variaciones intrínsecas, oclusiones, ruido, etc. La segunda razón es la similitud entre clases de objetos causada por algunos patrones visuales compartidos por diferentes clases.

Buscando superar estas dificultades, el reconocimiento de objetos es el conjunto de técnicas que tienen por finalidad la identificación de un objeto inmerso en una imagen utilizando características intrínsecas a él. Estas características deben permitir discriminar el objeto en estudio de otros objetos y es deseable que sean invariantes a cambios de posición, color, iluminación y forma del objeto. Considerando que la información que caracteriza a los objetos ha sido extraída y expresada en forma conveniente, el reconocimiento de objetos puede ser formulado como un problema de clasificación, donde se interpreta nueva información contenida en imágenes con base en la experiencia obtenida previamente de imágenes en las que se conocían los contenidos. Por lo tanto, el proceso puede ser dividido en dos fases, aprendizaje e inferencia. En el aprendizaje se modela la relación entre los datos de la imagen y los objetos contenidos en ella. En la inferencia se utiliza esta relación para predecir el contenido de nuevas imágenes. En este sentido, De acuerdo con Forsyth D. A. y Ponce Jean, el reconocimiento de objetos consiste en comparar una imagen o parte de ésta con alguna información almacenada en su base de datos con la finalidad de identificarla [13].

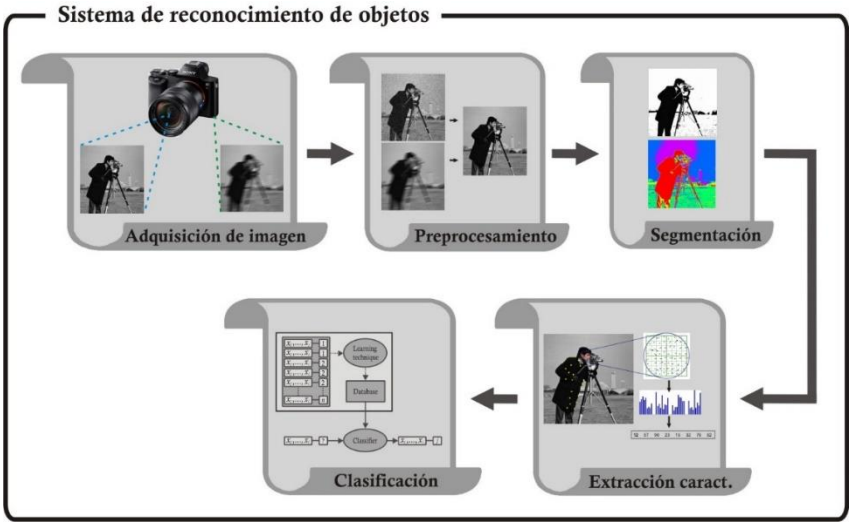


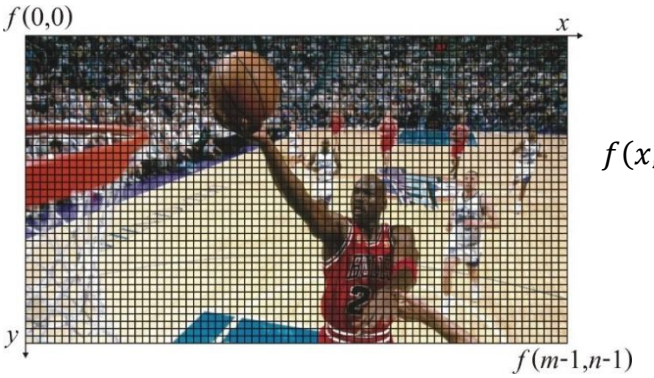
Figura 2.1 Esquema clásico de un SRO.

Por su parte, Rossius S. en [14], menciona que el procesado de la imagen que lleva a cabo un SRO está compuesto por un conjunto de bloques que cumplen tareas bien definidas y encaminadas a realizar un reconocimiento correcto y eficaz. Estas tareas están dirigidas a eliminar la mayor parte de información no necesaria para reconocer el objeto, y seleccionar únicamente aquellos parámetros que permiten una identificación e interpretación inconfundible del objeto.

El esquema clásico de un SRO es mostrado en la Figura 2.1 [15], [16], en este esquema se aprecian cinco etapas constitutivas de un SRO, adquisición de la imagen, preprocesamiento, segmentación, extracción de características y clasificación, las cuales son descritas a continuación. Cabe hacer mención que, aunque en esquemas actuales la etapa de segmentación no es necesaria, se decidió referir el esquema clásico para describir todas las etapas que en algún momento han formado parte de un SRO.

2.2.1 Adquisición de la imagen

Se trata de un proceso mediante el cual una escena del mundo real, en tres dimensiones, es discretizada y representada en dos dimensiones, normalmente denominada imagen, con la finalidad de poder ser procesada por dispositivos digitales. Este proceso puede ser realizado por una gran variedad de sistemas especializados, los más populares están basados en tecnología CCD (*Charge Coupled Devices*) o CMOS (*Complementary Metal Oxide Semiconductor*). El proceso de discretización de la escena se realiza tanto en coordenadas espaciales (x, y), denominado muestreo, como en intensidad o color, denominado cuantificación. Finalmente, la imagen resultante es representada por una función $f(x, y)$ cuyo valor es proporcional a la intensidad o al color de la escena en la coordenada espacial definida por x y y. Así, la escena es aproximada mediante una representación matricial de $m \times n$ elementos, denominados píxeles, los cuales son muestras de la escena igualmente espaciadas (ver Figura 2.2).



$$f(x,y) = \begin{bmatrix} f(0,0) & \cdots & f(0,n-1) \\ \vdots & \ddots & \vdots \\ f(m-1,0) & \cdots & f(m-1,n-1) \end{bmatrix}$$

Figura 2.2 Aproximación discreta de una imagen.

2.2.2 Preprocesamiento

La pérdida y distorsión de información en una escena son fenómenos implícitos al proceso de adquisición, esto se puede deber a diversos factores, como las condiciones del entorno, imperfecciones en el sistema de captura, el proceso de cuantificación, entre otros. Estos fenómenos tienen como consecuencia que el sistema de adquisición genere imágenes con ruido, mal enfocadas, con deficiente definición de contraste, etc.

El principal objetivo de la etapa de preprocesamiento es mejorar el aspecto de las imágenes para con esto facilitar el trabajo de etapas posteriores. Entonces, el preprocesamiento incluye aquellos métodos y técnicas que cumplen tareas como, corregir la degradación sufrida por una imagen, eliminar efectos no deseados, mejorar la calidad de la imagen, resaltar, agudizar y/o contrastar determinados aspectos de la imagen, aplicar operadores que incrementen la posibilidad de detectar detalles importantes de los objetos.

2.2.3 Segmentación

Considerando que comúnmente una imagen está formada por más de un objeto, para un SRO resulta ineficiente procesar una imagen completa. En términos generales, el objetivo de la etapa de segmentación es separar o destacar partes significativas de la imagen que tienen propiedades y/o características en común, como el valor de intensidad de los píxeles, el color, la textura, etc. Este proceso genera subimágenes que resultan más fácil de procesar y/o analizar.

Considerando las propiedades de una imagen, existen dos enfoques que agrupan las técnicas y métodos de segmentación de imágenes,

- Enfoque basado en la detección de discontinuidad. La imagen es dividida basándose en la localización de pixeles contiguos que presentan cambios bruscos de intensidad, como líneas o bordes, los cuales son considerados límites de regiones.
- Enfoque basado en la detección de semejanza. La imagen es dividida basándose en la búsqueda de zonas que contengan pixeles de valores similares.

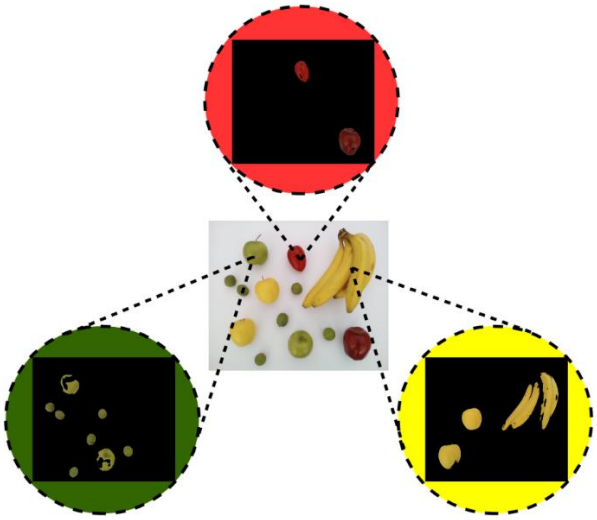


Figura 2.3 Segmentación de objetos por color.

La Figura 2.3 muestra el resultado del proceso de segmentación de una imagen en función de la propiedad de color. Las etapas de extracción de características y clasificación de un SRO son relevantes para la presente investigación. Dos razones fundamentas esto, primero, la extracción de características, y en especial la implementación en hardware del descriptor DAISY, es la parte central de este trabajo de tesis. Segundo, se desarrolló mediante un lenguaje de software un clasificador con un enfoque neuronal, las MAE, mismo que fue utilizado para determinar el desempeño de la implementación del mencionado descriptor. Por estas razones, se han dedicado las siguientes subsecciones principales para discutir a profundidad estas etapas, poniendo especial énfasis en los métodos empleados para el desarrollo de este trabajo de tesis, el descriptor DAISY y las MAE.

2.3 Extracción de características

Ofrecer una representación simple y eficaz de un objeto y que además tenga una dimensionalidad reducida es el objetivo principal de la etapa de extracción de características de un SRO. Si se considera que solo será procesada una región aislada de la imagen, la cual contiene un objeto de interés, esta representación está formada por características que proporcionan información precisa del objeto, y que suelen denominarse descriptores.

En el contexto del reconocimiento de objetos, una característica es un patrón presente en una imagen que es claramente diferente a sus vecinos cercanos.

Los descriptores son representaciones numéricas de las características visuales homogéneas del contenido de una imagen como la forma, el color, la textura, el movimiento, entre otras. Típicamente, los descriptores son agrupados en un vector, denominado vector de características o vector de descriptores, el cual es referenciado simplemente como descriptor en gran parte de la bibliografía.

Figura 2.4 Representación de la obtención de características, a) globales y b) locales.

Esencialmente y dependiendo de la estructura de imagen que se pretende describir, un vector de características o descriptor puede ser formado por características globales o características locales.

Las características globales tienen su campo de aplicación en la localización. Rara vez son utilizadas en tareas de reconocimiento de objetos debido a que el descriptor se obtiene considerando la información de toda la imagen, es decir, estos descriptores tienen por objetivo resumir toda la información contenida en la imagen, proporcionando una representación muy compacta de todos los objetos que la integran. Esto ocasiona que el descriptor sea sensible a pequeñas variaciones en la imagen provocadas por la orientación, los cambios en la iluminación o la oclusión parcial. Esto puede ocasionar un cambio importante en el descriptor obtenido y, por tanto, generar un comportamiento erróneo en el SRO. La Figura 2.4 muestra gráficamente la diferencia entre este tipo de características.

Un enfoque que basa su operación en múltiples descriptores de características locales ha sido propuesto para dar solución a este problema. Los descriptores de características locales, a diferencia de los globales, proporcionan una representación de un segmento de la imagen, en lugar de la imagen completa. En este sentido, Tuytelaars y Mikolajczyk en [17] definen una característica local como: "un segmento o patrón de imagen que difiere de su vecindad inmediata". Las características locales invariantes son consideradas una representación que permite unir eficientemente las estructuras locales de las imágenes. Esta representación se logra en dos fases, la detección de características o puntos de interés y la formación del descriptor. En la primera se genera un conjunto disperso de mediciones locales, que pretende capturar las propiedades esenciales de los objetos de las imágenes. En la segunda fase, del conjunto generado son elegidas y codificadas las mediciones locales más relevantes.

En los párrafos anteriores se encuentran los argumentos del porqué un descriptor formado por características locales es preferible en un SRO. Por lo que es necesario describir en forma más precisa las fases necesarias para obtener un descriptor de características locales.

Detección de características o puntos de interés. Esta fase consiste en identificar y extraer características sobresalientes, denominados puntos clave o de interés, de una imagen. Idealmente, un punto de interés identificado debe corresponder a objetos semánticamente significativos o a sus partes. Esta es una tarea extremadamente compleja. Por lo que un detector de características

tiene el modesto objetivo de encontrar patrones de imagen que difieren significativamente de sus vecindarios contiguos, como pueden ser bordes, esquinas, manchas, entre otros.

Es deseable que los puntos de interés detectados presentes las siguientes propiedades:

- Repetibilidad. Un alto porcentaje de los puntos de interés detectados debe ser común entre dos imágenes del mismo objeto o escena. La repetibilidad se puede lograr por invariancia o robustez.
    - Invarianza. Se presenta cuando deformaciones o transformaciones aplicadas a la imagen no afectan significativamente la detección de puntos de interés.
    - Robustez. El sistema es robusto si presentan un cierto grado de inmunidad a ruido presente en la imagen, a imágenes mal enfocadas, la perdida de información debida a un proceso de compresión, etc.
- Distinción. Los patrones de intensidad alrededor de los puntos de interés deben tener suficiente varianza para poder distinguir y combinar las características.
- Discriminación. Los puntos de interés obtenidos deben permitir discriminar objetos de diferentes clases.
- Eficiencia computacional. Los puntos de interés deben poder ser calculadas en tiempos aceptables.

Formación del descriptor. Identificados y extraídos los puntos de interés, la fase de formación del descriptor tiene por objetivo seleccionar los que se consideren relevantes y representarlos de una manera que sea invariable a transformaciones de imágenes no deseadas, como la rotación, la traslación, el tamaño, etc.

En la última década han sido ampliamente usados algoritmos que representan un objeto mediante un enfoque basado en características locales. Este enfoque calcula características locales y las codifica en un vector que comúnmente se denomina descriptor. La importancia de este enfoque radica en el hecho que las características locales pueden ser invariantes a transformaciones de escala, perspectiva y orientación, generando algoritmos que presentan un reconocimiento robusto cuando este tipo de transformaciones están presentes en los objetos.

Algoritmos que usan este enfoque han mostrado una alta eficiencia al ser utilizados para modelar objetos en un sistema de reconocimiento. Algunos de estos algoritmos son LBP (*Local Binary Patterns*) [18], SIFT (*Scale-Invariant Feature Transform*) [19], HOG (*Histogram of Oriented Gradients*) [8], SURF (*Speeded Up Robust Features*) [9] y DAISY [7].

Esta investigación tiene como principal objetivo el diseño, modelado e implementación de una arquitectura hardware del descriptor DAISY. En la siguiente sección se descrine con precisión el mencionado descriptor.

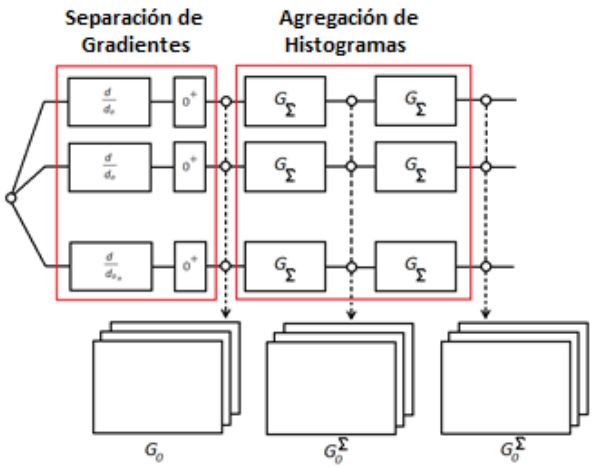


Figura 2.5 Cálculo conceptual del descriptor DAISY.

2.3.1 Descriptor DAISY

El descriptor DAISY está inspirado en los algoritmos SIFT y GLOH (*Gradient Location and Orientation Histogram*) [20]. La principal diferencia entre estos algoritmos y DAISY es que mientras los primeros calculan histogramas basados en orientaciones de gradientes y los codifican en contenedores, DAISY obtiene estos valores mediante mapas de convolución. Es decir, los descriptores locales han demostrado ser muy rentables en la correspondencia densa, pero extender su uso a todos los píxeles es una tarea de alto costo computacional. DAISY da solución a este inconveniente convolucionando mapas de orientación, que se pueden calcular de manera muy efectiva en el caso denso, para calcular los valores de los contenedores. Para dar una definición formal que permita entender con precisión el funcionamiento del descriptor DAISY, éste es dividido en 4 fases: cálculo de los mapas de orientación, cálculo de los mapas de orientación convolucionados, generación de vectores en los mapas de orientación convolucionados y definición del descriptor. Dada una imagen de entrada I, en la primera fase se calculan H mapas de orientación G_i, $1 \leq i \leq H$, uno por cada dirección cuantificada, donde $\mathrm{G}_o(u, v)$ representa el gradiente de la imagen en el punto (u, v) para la dirección o, solo si su valor es más grande que cero, en caso contrario se iguala a cero. Formalmente, los mapas de orientación son definidos como $\mathrm{G}_o = \left(\frac{\partial I}{\partial o}\right)^+$, donde o es la orientación de la derivada y $(.)^+$ es el operador tal que $(a)^+ = max(a, 0)$.

En la segunda fase, cada mapa de orientación es convolucionado varias veces con filtros gaussianos de diferente valor de desviación estándar, Σ. Este proceso genera mapas de orientación convolucionados para diferentes escalas definidos como,

$$\mathrm{G}_o^{\Sigma} = G_{\Sigma} * \left(\frac{\partial I}{\partial o}\right)^+ \tag{2.1}$$

Donde $\boldsymbol{G}_{\Sigma}$ es el filtro gaussiano. Valores diferentes en Σ se usan para controlar el tamaño de la región de agregación. Este proceso puede ser implementado eficientemente mediante el cálculo

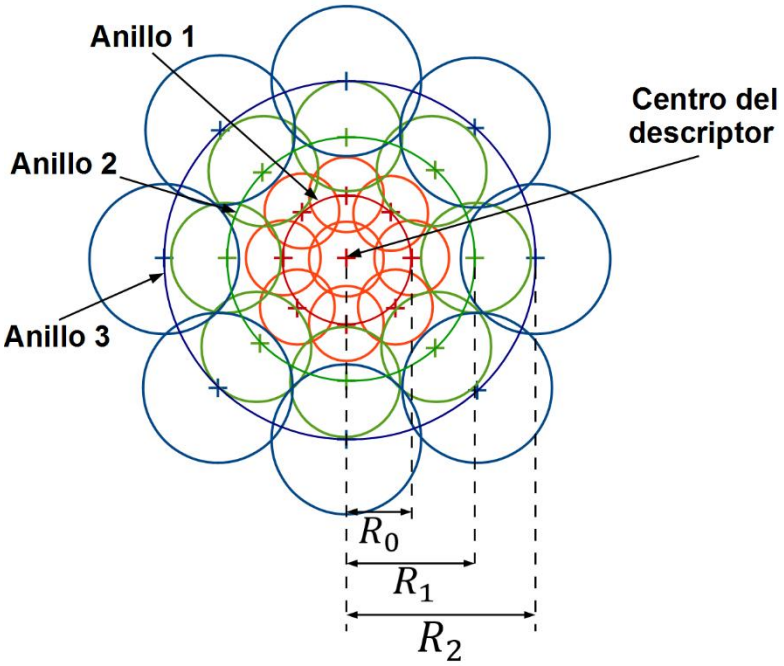


Figura 2.6 Descriptor DAISY.

recursivo de las convoluciones, como se muestra en la Figura 2.5. Un punto clave de DAISY es que los mapas de orientación convolucionados constituyen los valores de los contenedores de los histogramas del descriptor. Los mapas de orientación convolucionados se pueden calcular de manera muy efectiva en el caso denso. Esto ofrece el mismo tipo de invariancia que el histograma SIFT y GLOH, pero mucho más rápido para fines de correspondencia densa.

La Figura 2.6 muestra una representación gráfica de la operación del descriptor DAISY. Cada círculo representa una región de donde se formará el vector de características final, cuyo radio es proporcional a las desviaciones estándar de los filtros gaussianos utilizados para cada anillo y el símbolo '+' representa las ubicaciones de los *centros* de estas regiones. Al superponer las regiones, se consiguen transiciones suaves entre ellas y cierto grado de robustez rotacional. El radio de las regiones exteriores se incrementa para tener un muestreo igual al eje de rotación que es necesario para la robustez rotacional. Esta figura muestra una versión de 3 anillos del descriptor DAISY, donde cada anillo contiene 8 regiones.

De esta forma, en cada píxel de la imagen, el vector de características del descriptor DAISY es formado por los valores en los mapas de orientación convolucionados ubicados en círculos concéntricos centrados en las coordenadas de dicho pixel, y donde la cantidad de suavizado gaussiano es proporcional al radio de los círculos.

En la tercera fase se forma un vector $\mathbf{h}_{\mathbf{\Sigma}}$ a partir de los valores de las coordenadas (u, v) de cada mapa de orientación después de la convolución con el filtro gaussiano con desviación estándar, Σ:

$$\mathbf{h}_{\mathbf{\Sigma}}(u, v) = [\mathrm{G}_1^{\Sigma}(u, v), \dots, \mathrm{G}_H^{\Sigma}(u, v)]^T \quad (2.2)$$

Donde $\mathrm{G}_1^{\Sigma}, \dots, \mathrm{G}_H^{\Sigma}$ denotan los mapas de orientación convolucionados.

Al inicio de la cuarta fase, los vectores son normalizados para que sus normas sean 1, y son denotados como $\tilde{\mathbf{h}}_{\boldsymbol{\Sigma}}(u, v)$. La normalización se realiza en cada histograma de forma independiente con la finalidad de poder representar los píxeles cerca de las oclusiones de la manera más correcta posible. Si el descriptor fuera normalizado como un todo, entonces las descripciones del mismo punto que está cerca de una oclusión serán muy diferentes en dos imágenes. Finalmente, el descriptor DAISY $\boldsymbol{D}(u_o, v_o)$ para la coordenada (u_o, v_o) es definido como una concatenación de los vectores $\tilde{\mathbf{h}}$:

$$\begin{aligned}\boldsymbol{D}(u_o, v_o) = \big[&\tilde{\mathbf{h}}_{\Sigma_1}^T(u_o, v_o),\\ &\tilde{\mathbf{h}}_{\Sigma_1}^T\big(\mathrm{l}_1(u_o, v_o, R_1)\big), \dots, \tilde{\mathbf{h}}_{\Sigma_1}^T\big(\mathrm{l}_N(u_o, v_o, R_1)\big),\\ &\tilde{\mathbf{h}}_{\Sigma_2}^T\big(\mathrm{l}_1(u_o, v_o, R_2)\big), \dots, \tilde{\mathbf{h}}_{\Sigma_2}^T\big(\mathrm{l}_N(u_o, v_o, R_2)\big), \dots,\\ &\tilde{\mathbf{h}}_{\Sigma_k}^T\big(\mathrm{l}_1(u_o, v_o, R_k)\big), \dots, \tilde{\mathbf{h}}_{\Sigma_k}^T\big(\mathrm{l}_N(u_o, v_o, R_k)\big)\big]^T\end{aligned} \quad (2.3)$$

Donde $\mathrm{l}_j(u, v, R_k)$ es la ubicación con distancia $R(u, v)$ en la dirección dada por j cundo las direcciones son cuantizadas en N valores. En la tabla 2.1 se definen los parámetros que son requeridos para la configuración del descriptor DAISY.

Tabla 2.1 Parámetros DAISY.

Símbolo	Descripción (valor por defecto)
Q	Número de anillos circulares (3).
R_k	Radio del k-ésimo anillo; $k = 0, \dots, Q-1$
T	Número de histogramas por anillo (8).
H	Número de contenedores (*bins*) por cada histograma (8).
S	Número de Histogramas usados en el descriptor $= Q * T + 1$.
D_s	Tamaño del descriptor DAISY $= S * H$.

2.4 Clasificación de patrones

La clasificación de patrones es un problema que consiste en inducir, a partir de la observación de un conjunto finito de casos resueltos y con el mayor grado de exactitud posible, el valor de un caso incompleto que lo identifica dentro de una categoría fijada previamente por un supervisor. Es decir, consiste en una tarea que mediante una función f aproxima un mapeo de variables de entrada $\mathbf{x}$ a variables de salidas c; donde $\mathbf{x} \in \mathbb{R}^n$, y $c \in \mathbb{N}$. Las variables de salidas a menudo son llamadas categorías o clases, la función de mapeo predice la clase para el dato de entrada $\mathbf{x}$. Si el sistema de clasificación está formado por dos clases recibe el nombre de clasificación binaria, con más de dos clases se le llama multiclase.

Formalmente, el problema de clasificación consiste en asignar un vector $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$ a una de las N clases definidas en la aplicación. La clase verdadera se denota por c y toma valores en $\{1,2, \dots, N\}$. Se puede contemplar el clasificador como una función f que asigna etiquetas a observaciones, es decir:

$$f: (x_1, x_2, \dots, x_n) \rightarrow \{1,2, \dots, N\} \quad (2.4)$$

El objetivo es construir un clasificador que mediante una función minimice el coste total de los errores cometidos, esta función es estimada a partir de un conjunto de entrenamiento.

La clasificación de patrones puede ser desarrollada mediante los siguientes enfoques:

- Estadístico o teoría de la decisión. En este enfoque se extraen de los patrones propiedades de naturaleza cuantitativa. Este enfoque se basa en la teoría de probabilidad y estadística, supone la existencia de un conjunto de medidas numéricas con distribuciones de probabilidad conocidas y a partir de ellas se hace el reconocimiento.
- Sintáctico o estructural. Las relaciones geométricas o estructurales asociadas a la forma de los objetos en estudio representan los elementos fundamentales de este enfoque. El enfoque sintáctico busca construir una gramática que describa la estructura de los objetos pertenecientes al universo en estudio mediante la teoría de lenguajes formales.
- Lógico combinatorio. Este enfoque se basa en la idea de que la modelación del problema debe ser lo más cercana posible a la realidad de este. Es decir, los objetos en estudio y sus características deben ser modelados sin hacer suposiciones que no estén fundamentadas.
- Neuronal. Los supuestos de cómo trabaja el sistema nervioso central (SNC) biológico son las bases sobre las que cimienta su operación el enfoque neuronal. Este enfoque está formado por diversos paradigmas, siendo los más importantes las redes neuronales artificiales (RNA) y las memorias asociativas.

El último enfoque es de particular interés para el desarrollo de esta tesis, en particular las memorias asociativas.

2.4.1 Memorias Asociativas

El SNC de los seres vivos realiza tareas de reconocimiento como una actividad cotidiana y que aparentemente se logra sin esfuerzo alguno, haciéndolo además en una forma eficiente. Por ejemplo, el SNC del ser humano le permite identificar a una persona, aunque sólo observe una parte de su rostro, o que la persona se haya colocado peluca o lentes, o se haya quitado la barba o el bigote. Por su parte, el SNC de una rana le permite localizar a un insecto en pleno vuelo y calcular su trayectoria para lograr atraparlo con gran precisión. De acuerdo con el científico Donald O. Hebb, estas habilidades se relacionan con la memoria biológica y su capacidad asociativa [21]. Hebb sostuvo que la información disponible para cada neurona es local, reduciendo así el conjunto de conocimientos a procesar en cada momento y mejorando el rendimiento del sistema. Es decir, la memoria biológica reside en las conexiones sinápticas y los procesos de aprendizaje y de memorización involucraban cambios en los valores de dichas conexiones, lo que implica cambios locales progresivos en estas conexiones, en vez de cambios globales. Buscando emular este comportamiento y hacerlo disponible a un sistema artificial, se

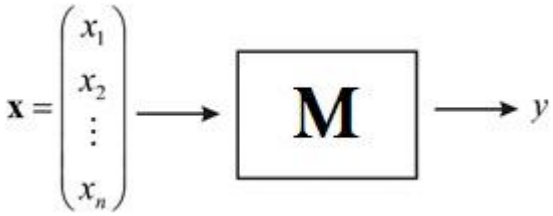


Figura 2.7 Esquema general de una memoria asociativa.

han desarrollado modelos neuronales, conocidos como memorias asociativas, que basan su operación en asociaciones de conceptos o eventos. Desde sus inicios, las memorias asociativas han demostrado ser un modelo neuronal muy eficiente, motivo por el cual han sido utilizadas en un amplio rango de aplicaciones tales como, reconocimiento de objetos, control inteligente, compresión de imágenes, cuantificación vectorial, entre otras.

Una memoria asociativa es un modelo computacional que codifica asociaciones entre patrones de entrada y patrones de salida para posteriormente almacenarla, teniendo como principal propósito utilizarla en la recuperación de patrones basándose sólo en un conocimiento parcial de su contenido. La recuperación de la información se consigue según el grado de similitud entre el patrón de entrada y los patrones memorizados. La Figura 2.7 muestra es esquema de una memoria asociativa enfocada a la clasificación de patrones. El vector columna $\mathbf{x} \in \mathbb{R}^n$ y representa al patrón de entrada, $y \in \mathbb{N}$ y representa la salida generada por la memoria, finalmente la memoria asociativa es representada por **M**.

En general, una memoria asociativa requiere de dos fases para dar cumplimento a su tarea, la fase de entrenamiento y la fase de clasificación.

El objetivo de la fase de entrenamiento es generar la memoria asociativa **M** a partir de p pares de asociaciones de vectores de entrada y vectores de salida $\{(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \dots, (\mathbf{x}^p, y^p)\}$. A esta agrupación se le denomina conjunto fundamental de asociaciones o simplemente conjunto fundamental (CF) [22]. Una representación simplificada del CF es la siguiente:

$$\{(\mathbf{x}^\mu, y^\mu) | \mu = 1,2, \dots, p\} \quad (2.5)$$

Durante el proceso de entrenamiento, la relación existente entre cada uno de los elementos del CF es codificada y almacenada en la memoria asociativa. Al concluir este proceso, la memoria asociativa ha sido creada y representada como una matriz **M** de dimensiones $p \times n$.

La fase de clasificación inicia cuando un vector de entrada, $\mathbf{x}^\mu$, es presentado a la memoria asociativa creada, **M**, y tiene por objetivo indicar a través de la salida, y^μ, a cuál clase de las aprendidas pertenece el vector de entrada.

La naturaleza del CF proporciona un importante criterio para clasificar las memorias asociativas. Si se cumple que $\mathbf{x}^\mu = y^\mu \forall \mu \in \{1,2, \dots, p\}$, se dice que la memoria es *autoasociativa*. Por otro lado, si $\exists \mu \in \{1,2, \dots, p\}$ para el que se cumple que $\mathbf{x}^\mu \neq y^\mu$ se dice que la memoria es *heteroasociativa*.

2.4.2 Memorias asociativas extendidas

Las MAE son un modelo neuronal que ha demostrado un alto desempeño al clasificar patrones con elementos de valor real; además, presenta robustez cuando clasifica vectores alterados con ruido [23]. Las MAE fueron creadas a partir de la memoria asociativa *Learmatrix*, propuesta por Steinbuch, la cual opera únicamente con patrones cuyos elementos son de valor binario [24].

2.4.2.1 Fase de aprendizaje de una MAE

Las MAE definen una función, denominada ϕ, que tiene por objetivo codificar la relación que existe entre los patrones de entrada y la clase a la que pertenecen. Esta relación es especificada por el CF, definido en la expresión 2.5. La función ϕ ubica en la i-esima fila de la memoria asociativa, representada como la matriz **M**, la información codificada de todos los vectores que pertenecen a la clase i. Para conseguirlo, la fase de entrenamiento de las MAE consiste en evaluar la función ϕ para cada clase del CF. Considerando lo mencionado, **M** puede ser estructurada de la siguiente forma:

$$\mathbf{M} = \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_i \\ \vdots \\ \phi_p \end{bmatrix} \tag{2.6}$$

La función ϕ_i es la evaluación de ϕ para todos los patrones de la clase i, donde $i = 1,2,\dots,p$. A este respecto, un importante punto es determinar la forma en que la función ϕ será evaluada. Considerando que los operadores promedio aritmético (prom) y mediana (med) frecuentemente son empleados en el análisis de señales e imágenes, en [25] se estudió el desempeño que tienen estos operadores cuando son utilizados al evaluar la función ϕ obteniendo excelentes resultados.

Considerando que el objetivo de la fase de entrenamiento es establecer la relación existente entre un vector de entrada $\mathbf{x}^\mu = \left[x_j^\mu\right]_n$ y la clase y^μ a la que pertenece, que cada clase está formada por q vectores de entrada, y que $\phi_i = \left(\phi_{i,1},\ \phi_{i,2},\dots,\ \phi_{i,n}\right)$, entonces la fase de entrenamiento de las MAE, cuando el operador prom es usado para evaluar la función ϕ_i, es definida por:

$$\phi_{i,j} = \frac{1}{q}\sum_{l=1}^{q} x_{j,l}\,, \quad j = 1,\dots,n \tag{2.7}$$

Por otro lado, la fase de entrenamiento de las MAE, cuando el operador med es usado para evaluar la función ϕ_i, es definida por:

$$\phi_{i,j} = \underset{l=1}{\overset{q}{\operatorname{med}}}\, x_{j,l}\,, \qquad j = 1,\dots,n \tag{2.8}$$

Vázquez, R. propuso añadir dos operadores a la fase de entrenamiento de las MAE, el operador de punto medio (pmed) y el operador suma (sum) [26]. Cuando el operador pmed es usado para evaluar la función ϕ_i, la fase de entrenamiento de las MAE es definida por:

$$\phi_{i,j} = \frac{\gamma_{i,j} + \lambda_{i,j}}{2}, \qquad j = 1, \dots, n \tag{2.9}$$

Usando el operador sum para evaluar la función $\phi_{i,j}$, la fase de entrenamiento de la MAE está definida por:

$$\phi_{i,j} = \gamma_{i,j} + \lambda_{i,j}, \qquad j = 1, \dots, n \tag{2.10}$$

Donde $\gamma_{i,j}$ y $\lambda_{i,j}$ representan los vectores máximo y mínimo de la clase i, respectivamente. Estos vectores se calculan aplicando las operaciones morfológicas max y min a los vectores de cada clase perteneciente al CF, siendo definidos por:

$$\gamma_{i,j} = \bigvee_{l=1}^{q} \left(x_j^{i,l} \right) \tag{2.11}$$

$$\lambda_{i,j} = \bigwedge_{l=1}^{q} \left(x_j^{i,l} \right) \tag{2.12}$$

Finalmente, La memoria asociativa **M** es obtenida después de evaluar todas las funciones ϕ_i. Para el caso donde existen p clases y los vectores a clasificar son n-dimensionales, la memoria resultante $\mathbf{M} = \left[m_{i,j} \right]_{p \times n}$ es:

$$\mathbf{M} = \begin{bmatrix} \phi_{1,1} & \phi_{1,2} & \cdots & \phi_{1,n} \\ \phi_{2,1} & \phi_{2,2} & \cdots & \phi_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{p,1} & \phi_{p,2} & \cdots & \phi_{p,n} \end{bmatrix} = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,n} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{p,1} & m_{p,2} & \cdots & m_{p,n} \end{bmatrix} \tag{2.13}$$

2.4.2.2 Fase de clasificación de una MAE

La fase de clasificación realizada por las MAE da inicio cuando un vector $\mathbf{x} \in \mathbb{R}^n$ es presentado a la memoria **M**, generada en la fase de entrenamiento. El objetivo de esta fase es determinar el índice de la clase a la que el vector presentado pertenece. Una MAE presenta un alto grado de generalización, es decir, ofrece un desempeño adecuado cuando vectores no utilizados en la fase de entrenamiento son presentados a la memoria.

Al utilizar operadores prom o pmed para construir la matriz **M**, la clase i a la cual pertenece el vector **x** presentado es determinada por,

$$i = arg \left[\bigwedge_{l=1}^{p} \bigvee_{j=1}^{n} \left| m_{lj} - x_j \right| \right] \tag{2.14}$$

De acuerdo con la expresión 2.14, la fase de clasificación es muy simple. Esta fase únicamente involucrar operaciones morfológicas, max y min, aplicadas a valores absolutos de la diferencia entre los componentes de la matriz **M** y los componentes del patrón **x** que será clasificado.

Si el operador med fue utilizado en la generación de la matriz **M**, la clase i a la cual pertenece el vector **x** presentado es definida por,

$$i = arg\left[\bigwedge_{l=1}^{p}\left|\underset{j=1}{\overset{n}{med}} m_{lj} - \underset{j=1}{\overset{n}{med}} x_j\right|\right] \tag{2.15}$$

Aunque esta forma de clasificar también es un proceso simple, tiene un grado de complejidad mayor que los operandos anteriores, debido a que es necesario obtenerse la mediana del vector presentado, involucrando un algoritmo de ordenamiento.

Finalmente, cuando se utilizó el operador sum para construir la matriz **M**, inicialmente se debe generar la matriz de recuperación $r = \left[r_{i,j}\right]_{p\times n}$. Los componentes de la matriz de recuperación pueden ser calculados de dos formas:

$$r_{ij} = x_j + \gamma_{i,j} \tag{2.16}$$

$$r_{ij} = x_j + \lambda_{i,j} \tag{2.17}$$

Entonces, la clase a la cual pertenece el vector **x** presentado está dada por:

$$i = arg\left[\bigwedge_{l=1}^{p}\bigvee_{j=1}^{n}|m_{lj} - r_{l,j}|\right] \tag{2.18}$$

La construcción de la matriz de recuperación, a pesar de requerir únicamente operaciones de suma, incrementa el grado de complejidad con respecto a operador prom.

El Teorema 1 y el Corolario 1-5 definidos en [22] gobiernan las condiciones que deben ser cumplidas para obtener un proceso de clasificación perfecto.

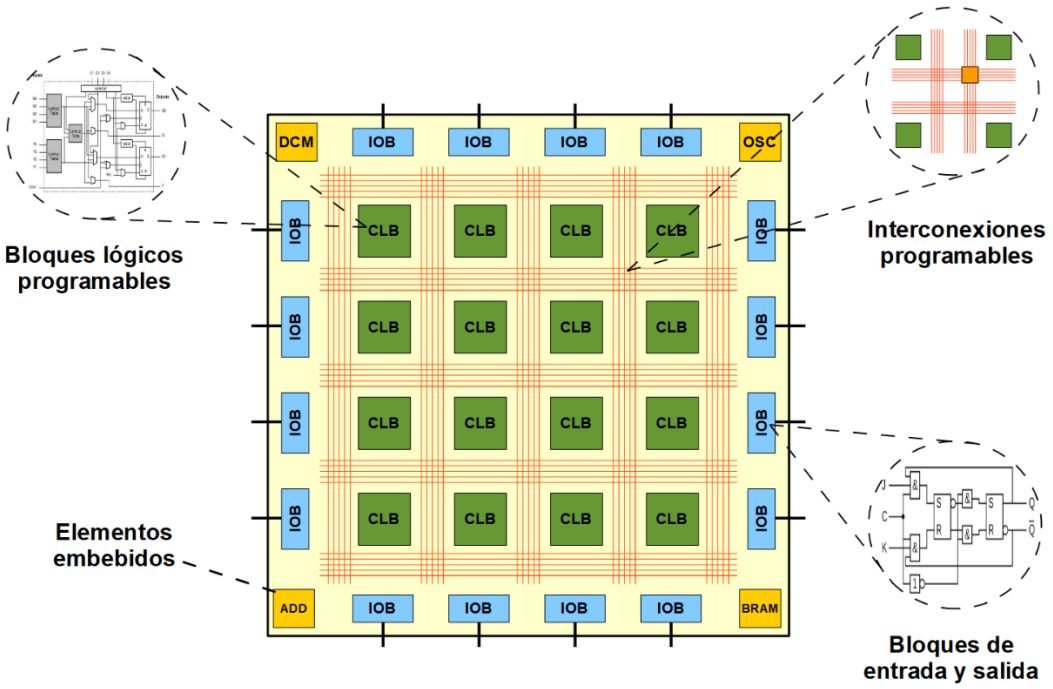


Figura 2.8. Estructura de un FPGA.

2.5 Arreglo de Compuertas Programables en Campo

Los circuitos digitales reconfigurables (CDR) son dispositivos digitales cuya función puede ser modificada utilizando únicamente una parte de los elementos que lo componen y/o cambiando la interconexión entre ellos. Dicha modificación se lleva a cabo mediante la programación del estado de un conjunto de variables binarias, a este proceso se le denomina configuración. Los CDR´s pueden ser clasificados de acuerdo con la organización de sus recursos en, CDR´s con recursos concentrados y CDR´s con recursos distribuidos.

Los CDR´s con recursos concentrados son los denominados dispositivos lógicos programables (PLD, *Programmable Logic Devices*). La arquitectura de un PLD es muy rígida, está formada por un arreglo de compuertas AND seguido de un arreglo de compuertas OR, basando su funcionamiento en sumas de productos (SOP, *Sum of Products*), y canalizando los resultados a un número limitado de biestables. Esta estructura genera retardos predecibles y frecuencias de operación típicas cercanas a los 200 MHz.

Los FPGA representan a los CDR´s con recursos distribuidos. La arquitectura de un FPGA contiene un gran número de simples bloques lógicos y abundantes recursos de interconexión, ambos distribuidos uniformemente en todo el dispositivo. Esta estructura confiere a un FPGA de una gran flexibilidad. Un FPGA utiliza las tablas de búsqueda (LUT, *Look Up Table*), también referenciadas como generadores de funciones, como principal paradigma en la implementación funciones lógicas.

La Figura 2.8 ilustra el esquema general de un FPGA, donde se puede apreciar que está integrado por cuatro elementos principales, bloques lógicos configurables (CLB, *Configurable Logic*

Block), Bloques de entrada-salida (IOB, *Input-Output Block*), recursos de interconexión y elementos embebidos de arquitectura fija.

Los CLB´s presentan una distribución matricial en el dispositivo y son los elementos de procesamiento de un FPGA al contener los recursos que son utilizados para implementar funciones lógicas. Los elementos básicos de un CLB son LUT, biestables y recursos que le brindan características adicionales. La complejidad de los CLB´s determina la granularidad del FPGA. En general, se dice que un FPGA es de granularidad fina si sus CLB´s tienen una estructura muy sencilla y aparecen en gran cantidad en el dispositivo. Por otro lado, si los CLB´s de un FPGA tienen una estructura compleja, motivo por el cual su número en el dispositivo es reducido, se trata de uno de granularidad gruesa. Un FPGA de granularidad gruesa posee la capacidad de implementar funciones complejas.

Los IOB´s forman un anillo que rodean la matriz de CLB´s. La función de los IOB´s es permitir al FPGA interactuar con el exterior, es decir, controlan el intercambio de información entre los dispositivos conectados a los pines de entrada/salida y la lógica interna del FPGA. Para cumplir con su propósito, contienen recursos que le brindan al FPGA una alta capacidad de interconexión, como buffers, latches, biestables, circuitos *by-pass*, circuitos para controlar la polaridad, resistencias pull-up y/o pull-down en la salida, etc.

La estructura de los CLB´s de un FPGA, incluso los de granularidad gruesa, es muy simple, motivo por el cual la función lógica que puede implementar un CLB es también sencilla. Cuando se requiere implementar una función compleja en un FPGA, es necesario interconectar CLB´s entre sí para conseguirlo. Los recursos de interconexión que un FPGA posee están concebidos para optimizar la interconexión entre CLB´s. Estos recursos son líneas de interconexión e interruptores programables individuales o agrupados en una matriz de interconexión. Los tipos de líneas de interconexión encontrados en un FPGA son, líneas de propósito general, líneas directas y líneas largas.

No existe un FPGA que no incluya los tres elementos mencionados. Además, cada vez es más común que funciones, como multiplicadores, memorias, administradores de reloj e incluso procesadores, sean incluidos en un FPGA como elementos de arquitectura fija. Estos son los llamados elementos embebidos de arquitectura fija. Características que cumple una función incluida como elemento de arquitectura fija en un FPGA son, ser una función comúnmente utilizada en un gran número de aplicaciones y, ser una función que requiere una gran cantidad de recursos para poder ser implementada.

Un FPGA se utiliza en aplicaciones similares a los ASICs, sin embargo, son dispositivos más lentos, tienen un mayor consumo de potencia y requieren más área de semiconductor para abarcar el mismo sistema. A pesar de esto, los FPGAs tienen las ventajas de ser reprogramables, lo que añade una enorme flexibilidad al flujo de diseño, sus costes de desarrollo y adquisición son mucho menores para pequeñas cantidades de dispositivos y el tiempo de desarrollo es también menor.

Para la implementación del sistema propuesto en esta investigación se utiliza la tarjeta de desarrollo Nexys 3, de la compañía Digilent Inc. La tarjeta Nexys tiene como elemento central de procesamiento un FPGA XC6LX16-CS324 de la familia Spartan-6, de la compañía Xilinx Inc. Este dispositivo está integrado por 2,278 *slices*, cada uno contiene 4 LUT´s de 6 entradas y 8 flip-flops (en total dispone de 18,224 flip-flops y 14,579 celdas lógicas), 32 DSP48A1 slices (cada uno contiene un multiplicador de 18×18, un sumador y un registro), 32 bloques RAM de 18 Kb (pudiendo estructurar una memoria de máximo 576 Kb), 2 módulos de gestión de reloj (cada uno contiene un administrador digital de reloj y un bucle de enganche de fase), 2 controladores de memoria (que soportan memorias DDR, DDR2, DDR3 y LPDDR) y un máximo de 232 pines de entrada salida.

2.6 Estado del arte

El presente trabajo de investigación versa sobre el diseño y modelado de una arquitectura hardware del descriptor DAISY y su implementación en un dispositivo FPGA. DAISY pertenece a la familia de descriptores que basan su operación en el cálculo de características locales. Con el objetivo de establecer una base de discusión para desarrollar una solución al problema a resolver, en este apartado se realiza una revisión de aquellos trabajos que tiene un fin similar al de esta tesis. Para ello, se analizan propuestas de arquitecturas hardware de descriptores que utilizan un enfoque basado en características locales.

En [27], Jan *et al*., presentaron la implementación del descriptor DAISY, introducido por Tola *et al*., en [7], sobre lógica reconfigurable. Los autores además propusieron una versión invariante a rotaciones del descriptor original que denominaron O-DAISY. La implementación de O-DAISY fue realizada en la tarjeta de desarrollo Xilinx Virtex-6 FPGA DSP, que contiene un FPGA LX240 de la familia Virtex-6, de la compañía Xilinx, Inc. La invarianza a la rotación se obtiene a partir de la orientación principal de los histogramas contenidos en cada uno de los anillos que posee el descriptor. Cada contenedor es alineado de forma ascendente a su orientación comenzando por la orientación principal. Este proceso se realiza tantas veces como histogramas existan en cada anillo. Finalmente, en la evaluación del rendimiento de esta propuesta se obtienen resultados que superan a SURF en el cálculo de puntos de interés y a BRIEF (*Binary Robust Independent Elementary Features*) en términos de invarianza rotacional.

Por su parte Yao *et al*., propusieron una arquitectura hardware para la detección de características optimizada de SIFT con el propósito de mejorar la velocidad de procesamiento del descriptor y minimizar el uso de recursos para la correspondencia de imágenes [28]. Su propuesta se basa en modificaciones a los siguientes tres aspectos. Primero, se utilizó el muestreo descendente en lugar del muestreo ascendente para preservar la operación de interpolación. El segundo aspecto se refiere al uso de sólo cuatro escalas con dos octavas para la comparación de imágenes con una degradación moderada del rendimiento de coincidencia. Por último, la tercera mejora es la reducción de la longitud total del vector de características a 72 elementos, a diferencia del vector de características que ofrece el algoritmo SIFT original con 128 elementos. Finalmente, la arquitectura hardware es implementada en un FPGA de la familia Virtex-5 de la compañía Xilinx, Inc. ofreciendo un desempeño superior al que muestra el algoritmo SIFT original. La última

mejora aporta un mejor desempeño del algoritmo permitiendo la detección de las características de la imagen con dimensión de 640×480 pixeles en 31 milisegundos contra los 33 milisegundos del original.

Bonato *et al*., presentaron el diseño de un sistema en chip (SoC, *System on Chip*) para la detección de características en imágenes basado en el algoritmo SIFT [29]. Para ello, aplicaron la técnica SLAM (*Simultaneous Localization And Mapping*), que permite al usuario una mayor flexibilidad para personalizar el descriptor de características de acuerdo con las necesidades de la aplicación. También se incluyen optimizaciones de hardware específicas que se consideran importantes para el diseño e integración de sistemas de control robótico. La implementación de este sistema se realizó en un kit de desarrollo Altera Stratix II S60 de la compañía Intel, en donde se incorporan cuatro cámaras con tecnología CMOS para la recopilación de las imágenes, las cuales son procesadas concurrentemente. Como resultado, el sistema es capaz de detectar características en imágenes con dimensiones de 320×420 píxeles a 30 cuadros de video por segundo (fps, *frames per second*).

Un descriptor de características similares a los que han sido citados anteriormente es el algoritmo SURF. Svab *et al*. presentaron una arquitectura hardware de SUFR que fue implementada en un FGPA XC5VFX70 de la familia Virtex-5 de la compañía Xilinx, Inc., el cual incorpora un procesador PowePC-440 [30]. En esta propuesta únicamente se implementaron las fases que demandan una gran cantidad de cálculos computacionales. Una de ellas es la etapa de detección de puntos de interés, logrando consumir el número reducido de recursos y a su vez lograr que el descriptor sea capaz de extraer los puntos de interés más importante de la imagen y generar los descriptores que son invariantes a los cambios de escala, rotación e iluminación. Finalmente, los resultados obtenidos muestran una robustez similar a la implementación del algoritmo en una GPU (*Graphics Processing Unit*), logrando obtener 10 fps en la ejecución del algoritmo a una resolución en alta definición (HD, *High Definition*) de 1024×768 píxeles. Debido a este rendimiento el sistema se vuelve idóneo para aplicaciones que trabajen en tiempo real, además de presentar un consumo de energía inferior a los 10 Watts.

Al igual que la investigación anterior, la propuesta hecha por Čížek & Faigl en [31] también la implementación en hardware del descriptor de características SURF, pero priorizando aspectos del algoritmo que son beneficiosos cuando es aplicado a técnicas de procesamiento de imágenes como SLAM. Los aspectos más importantes de la arquitectura propuesta son el rendimiento que esta ofrece y la escalabilidad de baja latencia, que la vuelve ideal para aplicaciones en donde el factor fundamental es el tiempo real. Para esto, se eligió la utilización de filtros gaussianos separables en la etapa de detección de puntos de interés. También, se realizó una optimización de la implementación sobre lógica reconfigurable para evitar los cálculos redundantes y el excesivo almacenamiento en el buffer de datos. Así como de eficientar el proceso de supresión de no máximos eliminando los accesos al sistema de memoria. Estas contribuciones hacen que la arquitectura reduzca significativamente el uso de recursos computacionales. La implementación de la propuesta se realizó en la tarjeta de desarrollo Terasic DE0-Nano que cuenta con un FPGA de la familia Cyclone V y un procesador ARM Cortex A9 de la compañía Intel. La evaluación

mostro que la arquitectura propuesta es capaz de procesar los datos de las imágenes con una velocidad de transmisión de 140 megapíxeles por segundo, lo cual es equivalente a transmisiones de video desde 640×480 en 420 fps hasta 1920×1080 en 60 fps. Además, de presentar un consumo de elementos lógicos del FPGA aproximadamente del 20 %.

En un trabajo interesante presentado por Sledevic *et al*., en [32], se realiza la combinación de los descriptores de características LBP y HOG con la identificación de movimiento en una arquitectura hardware aplicada a la detección y seguimiento de objetos en movimiento. Esta propuesta calcula y compara los vectores de características generados una sola vez cuando el objeto se ha desplazado utilizando una arquitectura para la detección de disparo único (SSD, *Single Shoot Detector*), la cual toma la decisión si se realiza el seguimiento del objeto o no. El sistema es modelado utilizando VHDL y es implementado en un FPGA de la familia Virtex-4 de la compañía Xilinx, Inc. Los resultados que se obtuvieron de la implementación de la arquitectura propuesta mostraron que el sistema es capaz de encontrar y seguir ocho objetos en movimiento de forma simultánea. Además, el seguidor diseñado basado en las características de HOG mostró robustez a la variación de luminosidad y a la oclusión parcial del objeto, mientras que el seguidor basado en las características de LBP ofrece robustez a la rotación. Siendo resultados favorecedores para aplicaciones que requieran trabajar en tiempo real a 60 fps y con una resolución de video de 640×480 píxeles.

En [33] es presentada una arquitectura hardware paralela basada en un FPGA para la detección de rostros en tiempo real utilizando el algoritmo LBP y el mapeo certero de rostros (FCM, *Face Certainty Map*). La implementación de la arquitectura propuesta fue realizada en una FPGA de la familia Virtex-5 de la compañía Xilinx, Inc. Esta arquitectura genera una pirámide de imágenes con veinte niveles de profundidad a partir de la imagen de entrada. Después, para cada una de estas subimágenes, se realiza la transformación a un patrón binario y se evalúan las características de forma paralela mediante el uso de una arquitectura de tipo ventana basada en bloques RAM. Al efectuarse la búsqueda de características entre dos subimágenes simultáneamente los recursos de enrutamiento se reducen a la mitad, ya que se comparte dicha arquitectura. Esto ocasiona que el resultado final de la implementación muestre un rendimiento en velocidad de alrededor de los 300 fps cuando se procesan imágenes con dimensiones estándar de 640×480 píxeles. Además, de reducir la tasa de falsos aceptables (FAR, *False Acceptance Rate*).

Siguiendo la misma vía de análisis de descriptores de características, en [34] es presentada una implementación basada en lógica reconfigurable aplicada al reconocimiento de objetos inmersos en imágenes. En esta propuesta se incluye un algoritmo para la detección de puntos de interés (esquinas) en la imagen llamado FAST (*Features from Accelerated Segment Test*). Para generar un vector de características binario robusto, es incorporado el descriptor de características denominado BRIEF. Finalmente, se realiza la comparación de los puntos de interés para obtener las coordenadas de los píxeles que forman el vector característico. Para implementar esta arquitectura, los autores han elegido una plataforma para desarrollo que cuenta con un SoC de la familia Zynq-700 equipado con un procesador de doble núcleo ARM Cortex-A9 y un FPGA de la familia Artix-7. Las evaluaciones experimentales muestran que la arquitectura propuesta

impacta en la reducción del uso de la memoria interna y recursos hardware en un 57% y 70%, respectivamente.

Otro descriptor perteneciente al enfoque de modelos de características locales es el presentado por Kapela *et al*., en [35] conocido como FREAK (*Fast Retina Keypoint*), el cual es incorporado a una plataforma flexible hardware-software basada en un FPGA de la familia Artix-7 y un procesador ARM Cortex-A9. En esta placa de desarrollo el sistema propuesto ejecuta la etapa de búsqueda o detección de características a partir de la imagen de entrada en el procesador. Por otro lado, la etapa de generación de los descriptores es realizada por el FPGA. Como resultado de combinar hardware con software ocasiona que el sistema se vuelva eficiente en su ejecución. Así mismo, se aprovechan las características que ofrece el FPGA de evaluar múltiples situaciones de forma simultánea. De esta forma, los cálculos más exigentes son ejecutados por el FPGA, mientras que las tareas simples son hechas por el software (procesador).

Capítulo 3 Modelado conceptual del descriptor DAISY

El modelado conceptual es una representación secuencial de un algoritmo, la cual puede ser desarrollada mediante un lenguaje de software. Mediante un modelado conceptual se puede entender el funcionamiento de un algoritmo e identificar con precisión las diferentes fases que lo integran. El diseño de una arquitectura hardware de un algoritmo puede partir de su modelado conceptual, facilitando esta tarea al posibilitar la detección de los recursos hardware involucrados en el diseño y permitiendo identificar las técnicas idóneas para su implementación en hardware. El presente capítulo describe el modelado conceptual de los algoritmos del descriptor DAISY y de las MAE. Además, se incluyen resultados experimentales que muestran el desempeño del descriptor DAISY interactuando con las MAE.

3.1 Modelado conceptual del descriptor DAISY

La implementación del modelado conceptual del descriptor DAISY inicia definiendo un conjunto finito de procesos u operaciones, organizadas de manera lógica y ordenada, que describen su comportamiento. Para cumplir esta tarea, se considera lo expuesto en la sección 2.3.1. El descriptor DAISY recibe una imagen I como parámetro de entrada y como condiciones iniciales es necesario definir los siguientes parámetros: $Q = 2, H = 6, T = 6$ y R_k, donde $k = 0,1, \dots, Q - 1$ e indica el anillo correspondiente, siempre $R_{Q-1} = R_1 = 16$ para el último anillo y los radios

de los anillos restantes se calculan con $R_k = \frac{R_{Q-1}}{Q}(k+1) = 8(k+1)$. Así, $R_0 = 8$ es el radio del segundo anillo.

Considerando lo mencionado en el párrafo anterior y que el algoritmo del descriptor DAISY tiene un grado de complejidad relativamente alto, inicialmente, y para facilitar el desarrollo del modelado conceptual de este descriptor, se definen procesos generales que especifican la forma de operar de DAISY. Una buena forma de empezar es hacer coincidir estos procesos generales con las fases de DAISY debido a que éstas indican en términos generales la secuencia ordenada de las operaciones necesarias para obtener el descriptor. Por lo tanto, los procesos generales son,

1. Calcular un mapa de orientación en cada una de las orientaciones definidas.
2. Aplicar filtros gaussianos con diferentes valores de desviación estándar a los mapas de orientaciones para obtener los mapas de orientación convolucionados.
3. Generación histogramas a partir de los mapas de orientación convolucionados.
4. Formar el descriptor mediante un proceso de concatenación de los histogramas y aplicarle un proceso de normalización.

Las siguientes subsecciones definen el conjunto de subprocesos requeridos para cumplir con cada uno de los procesos generales mencionados.

3.1.1 Generación de los mapas de orientación

La fase generación de mapas de orientación del descriptor DAISY tiene por objetivo obtener *H* mapas de orientación, los cuales son subimagenes con una orientación definida (derivadas), obtenidos a apartir de la imagen de entrada (ver Figura 3.1).

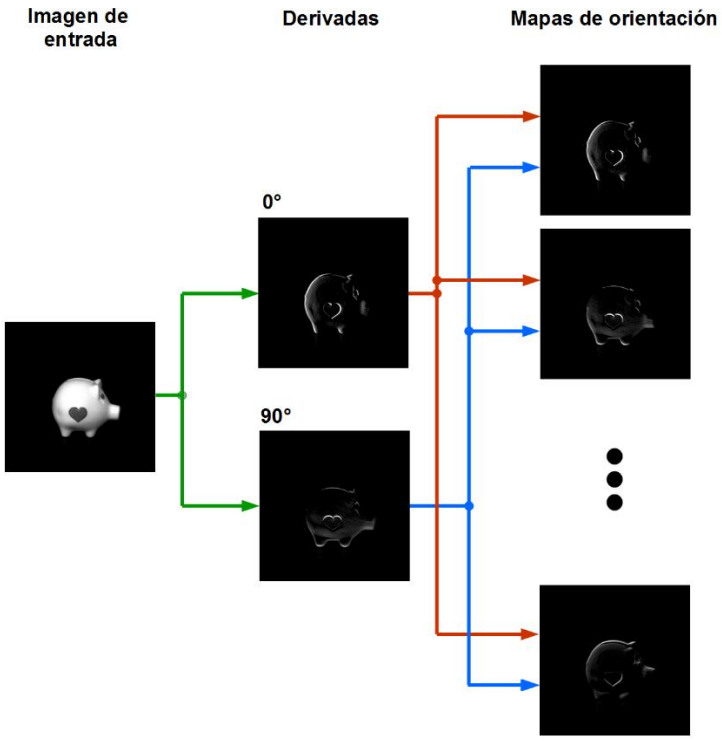


Figura 3.1 Generación de los mapas de orientación.

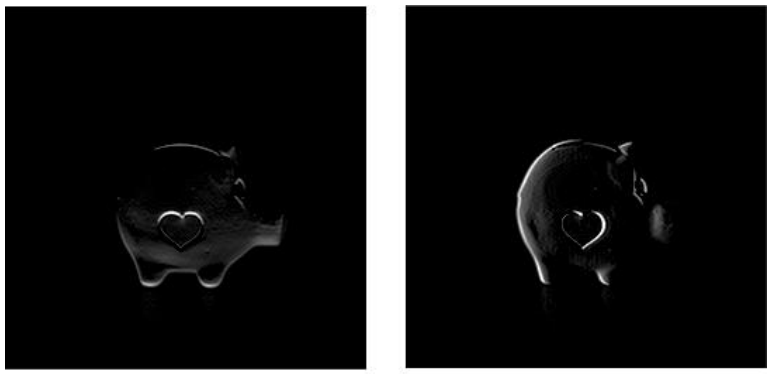


Figura 3.2 Gradientes de la imagen original: a) gradiente en x y b) gradiente en y.

Los subprocesos necesarios para obtener los mapas de orientación y la implementación de cada uno de ellos son explicados a continuación en forma conjunta.

Subproceso 1. El proceso de obtención de los mapas de orientación exige que primero sean definidas las T orientaciones donde se calculará cada uno de ellos. Considerando la primera orientación $o_0 = 0°$, las restantes son calculadas mediante $o_j = \frac{360° * j}{T}$, para $j = 0, 1, \dots, T-1$, obteniendo las siguientes orientaciones $\{0°, 60°, 120°, 180°, 240°, 300°\}$.

Subproceso 2. Posteriormente, se calculan las derivadas $\frac{\partial I}{\partial x}$ y $\frac{\partial I}{\partial y}$ de la imagen original, cuyas orientaciones son 0° y 90°, respectivamente. Para ello se utilizan las siguientes expresiones basadas en las máscaras mostradas en la Figura 3.3.

$$\frac{\partial I}{\partial x} = I(x+1, y) - I(x-1, y) \quad (3.1)$$

$$\frac{\partial I}{\partial y} = I(x, y+1) - I(x, y-1) \quad (3.2)$$

Las derivadas son obtenidas en este subproceso las cuales son ilustradas en la Figura 3.2.

Subproceso 3. En este subproceso los mapas de orientación son obtenidos. Para cumplir esta tarea es necesario expresar en radianes las orientaciones definidas, esto se logra mediante $o_j = \frac{2\pi j}{T}$. Además, todas las operaciones de los demás procesos que involucren a o_j utilizarán su representación en radianes. Estas direcciones son visualizadas en la Tabla 3.1.

0	0	0
1	0	-1
0	0	0

a)

0	-1	0
0	0	0
0	1	0

b)

Figura 3.3 Máscaras: a) en x y b) en y.

Tabla 3.1 Orientaciones en grados y radianes de los mapas de orientación.

Orientaciones (o_j)	
Grados	**Radianes**
0	0
60	$\pi/3$
120	$2\pi/3$
180	π
240	$4\pi/3$
300	$5\pi/3$

La expresión 3.1 define el cálculo de los mapas de orientación. La expresión 3.1 se evalúa para cada orientación, o_j, obteniendo un mapa de orientación por cada uno de ellos.

$$\mathrm{G}_{o_j} = cos(o_j)\frac{\partial I}{\partial x} + \sin(o_j)\frac{\partial I}{\partial y} \qquad (3.3)$$

El Algoritmo 3.1 contiene el pseudocódigo del proceso que calcula los mapas de orientación. Es necesario recalcar que de acuerdo con el descriptor DAISY únicamente deben ser considerados aquellos valores donde $cos(o_j)\frac{\partial I}{\partial x} + \sin(o_j)\frac{\partial I}{\partial y} \geq 0$.

Algoritmo 3.1 Pseudocódigo del algoritmo para calcular los $\mathbf{G_o}$.

```
1     Función mapas_orientacion
2         p ← 0;
3         pixel ← 0.0;
4         Para j ← 0 Hasta j < T Con Paso 1 Hasta
5             Para y ← 0 Hasta y < alto Con Paso 1 Hasta
6                 Para x ← 0 Hasta x < ancho Con Paso 1 Hasta
7                     Pixel←((cos(o[j])*∂I/∂x[y][x]) + (sen(o[j])*∂I/∂y[y][x]));
8                   Si pixel < 0 Entonces
9                     G_o[j][p] = 0.0;
10                  Si No Entonces
11                    G_o[j][p] = pixel;
12                  Fin Si
13                p ← p + 1;
14                Fin Para
15            Fin Para
16        Fin Para
17    Fin Función
```

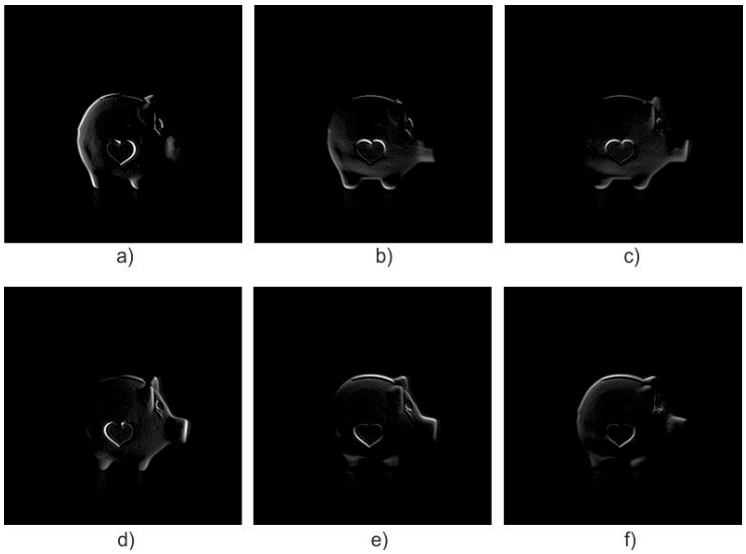


Figura 3.4 Mapas de orientación: a) 0°, b) 60°, c) 120°, d) 180°, e) 240° y f) 300°.

Figura 3.5 Mapa central.

Los seis G_o obtenidos a partir de las derivadas y las orientaciones definidas son mostrados en la Figura 3.4.

Subproceso 4. En este proceso de generación de los mapas de orientación, se debe obtener un mapa de orientación adicional, denominado mapa central G_c, que corresponde a la magnitud del gradiente y es definido por:

$$\mathrm{G}_c = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2} \qquad (3.4)$$

La Figura 3.5 ilustra el G_c obtenido en este subproceso.

3.1.2 Generación de los mapas de orientación convolucionados

El objetivo principal de este proceso es generar los $Q \times T$ mapas de orientación convolucionados, $\mathrm{G}_{o_j}^{\Sigma_k}$, a partir de la convolución de G_{o_j} con varios filtros gaussianos, G_{Σ_k}, donde cada Σ_k es de diferente valor; no está por demás recordar que $k = 0, \ldots, Q-1$ y $j = 0, \ldots, T-1$. Además, se obtiene el mapa central convolucionado, $\mathrm{G}_c^{\Sigma_c}$, al convolucionar G_c con el filtro G_{Σ_c}. Esta tarea es realizada mediante un conjunto de subprocesos que son definidos y descritos a continuación.

Subproceso 1. El algoritmo DAISY define un filtro por anillo, para este caso existen $Q = 2$ anillos, por lo tanto, también existen 2 filtros. En este subproceso se calculan los valores de Σ_0 y Σ_1, varianzas de los filtros del primer y segundo anillo respectivamente, mediante la siguiente expresión:

$$\Sigma_k = \frac{R_{Q-1}(k+1)}{2Q} \tag{3.5}$$

Siendo los valores obtenidos: $\Sigma_0 = 4$ y $\Sigma_1 = 8$.

El cálculo de los $\mathrm{G}_{o_j}^{\Sigma_k}$ se hace mediante la expresión 2.1, la cual se reescribe a continuación,

$$\mathrm{G}_{o_j}^{\Sigma_k} = G_{\Sigma_k} * \mathrm{G}_{o_j}$$

Así, $\mathrm{G}_{o_j}^{\Sigma_0} = G_{\Sigma_0} * \mathrm{G}_{o_j}$, mientras que si se cumple que $\Sigma_1 > \Sigma_0$, una alternativa para el cálculo de $\mathrm{G}_{o_j}^{\Sigma_1}$ es recalcular la varianza de G_{Σ_1} mediante $\acute{\Sigma}_1 = \sqrt{(\Sigma_1)^2 - (\Sigma_0)^2}$. Entonces el nuevo valor de $\acute{\Sigma}_1 = \sqrt{(8)^2 - (4)^2} = 6.928203 \approx 7$, y los $\mathrm{G}_{o_j}^{\Sigma_1}$ son definidos por,

$$\mathrm{G}_{o_j}^{\Sigma_1} = G_{\acute{\Sigma}_1} * \mathrm{G}_{o_j}^{\Sigma_0} \tag{3.6}$$

El valor de $\acute{\Sigma}_1$ es menor que Σ_1, lo cual repercute en un tamaño menor de $G_{\acute{\Sigma}_1}$ comparado con G_{Σ_1}, resultando en un proceso de convolución más simple.

Subproceso 2. Por simplicidad se retoma la nomenclatura anterior para la varianza, es decir $\acute{\Sigma}_1$ vuelve a ser denominada Σ_1. Por tanto, el subproceso 2 utiliza los valores de $\Sigma_0 = 4$ y $\Sigma_1 = 7$ para obtener los tamaños de G_{Σ_0} y G_{Σ_1} mediante la expresión 3.5, para la cual se debe considerar que el tamaño más pequeño de filtro que puede existir es 3.

$$t_k = \begin{cases} (5 * \Sigma_k) + 1 & \text{si residuo}\left(\frac{5 * \Sigma_k}{2}\right) = 0 \\ 5 * \Sigma_k & \text{en caso contrario} \end{cases} \tag{3.7}$$

Algoritmo 3.2 Pseudocódigo del algoritmo para calcular el tamaño del G_{Σ_k}.

```
Función tamaño_filtro (Σ)
    t ← 5*Σ;
    Si (t mod 2) Es Igual 0 Entonces
       t ← t + 1;
    Fin Si
    Si No (t < 3) Entonces
       t ← 3;
    Fin Si
Fin Función
```

Es necesario recordar que se usa el mismo filtro para calcular los $\mathrm{G}_{o_j}^{\Sigma_k}$ que pertenecen al mismo anillo. Así, se obtuvieron los tamaños $t_0 = 21$ para el primer anillo y $t_1 = 35$ para el segundo anillo.

Subproceso 3. El subproceso 3 recibe los valores de Σ_0, t_0, Σ_1 y t_1 y los emplea para calcular los coeficientes de los filtros del primer y segundo anillo, G_{Σ_0} y G_{Σ_1}, mediante la expresión 3.6.

$$sum = \sum_{x=0}^{t_k-1} G_{\Sigma_k}(x) = e^{-\frac{x^2}{2(\Sigma_k)^2}}$$
$$\tilde{G}_{\Sigma_k}(x) = \frac{G_{\Sigma_k}(x)}{sum} \quad (3.8)$$

Nuevamente, por simplicidad se reutiliza la nomenclatura anterior para los filtros gaussianos, es decir $\tilde{G}_{\Sigma_k}$ vuelve a denominarse como G_{Σ_k}. El Algoritmo 3.3 muestra el pseudocódigo que implementa al subproceso 3.

Algoritmo 3.3 Pseudocódigo del algoritmo para diseñar los filtros gaussianos unidimensionales.

```
1     Función gaussian_1d (Σ, t)
2         e ← 0;
3         sz ← 0;
4         v ← 0.0;
5         sum ← 0.0;
6         sz ← (t-1) / 2;
7         v ← Σ*Σ*2;
8         Para e ← (-1*sz) Hasta e < sz Con Paso 1 Hasta
9             G_Σ[e] ← exp(-1*((e*e) / v));
10            e ← e + 1;
11        Fin Para
12        Para e ← 0 Hasta e < t Con Paso 1 Hasta
13            sum ← sum + G_Σ[e];
14        Fin Para
15        Para e ← 0 Hasta e < t Con Paso 1 Hasta
16            G_Σ[e] ← G_Σ[e] / sum;
17        Fin Para
18    Fin Función
```

Los coeficientes de G_{Σ_0} y G_{Σ_1} obtenidos como resultado de este subproceso son los siguientes:

$$G_{\Sigma_0} = [0.00441, 0.00800, 0.01361, 0.02175, 0.03265, 0.04605, 0.06101, 0.07592, 0.08877, 0.09749, 0.10058, 0.09749, 0.08877, 0.07592, 0.60101, 0.04605, 0.03265, 0.02175, 0.01361, 0.00800, 0.00441] \quad (3.9)$$

$$G_{\Sigma_1} = [0.00298, 0.00431, 0.00608, 0.00837, 0.01126, 0.01480, 0.01898, 0.02377, 0.02908, 0.03474, 0.04053, 0.04617, 0.05137, 0.05582, 0.05922, 0.06137, 0.06210, 0.06137, 0.05922, 0.05582, 0.05137, 0.04617, 0.04053, 0.03474, 0.02908, 0.02377, 0.01898, 0.01480, 0.01126, 0.00837, 0.00608, 0.00431, 0.00298, 0.00451, 0.00791] \quad (3.10)$$

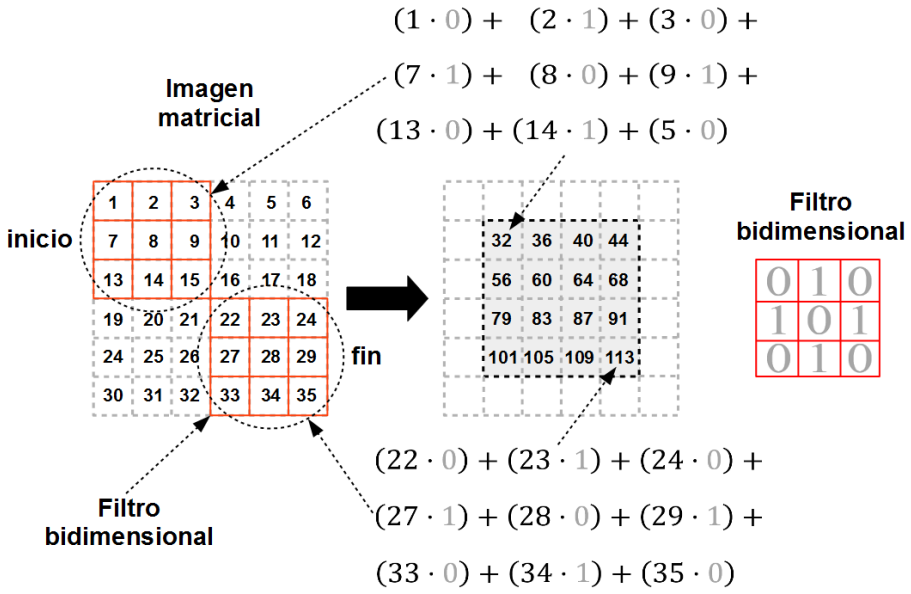


Figura 3.6 Proceso de convolución 2D.

Subproceso 4. La tarea de este subproceso es la generación de los mapas de orientación convolucionados del primer anillo, $\mathrm{G}_{o_j}^{\Sigma_0}$. Para cumplirla, se aplica la convolución bidimensional (convolución 2D) entre los G_{o_j} y el filtro gaussiano G_{Σ_0}. La convolución 2D es un proceso local en el cual una matriz de coeficientes (denominada comúnmente como filtro de convolución, máscara de filtrado o kernel) es desplazada por toda la imagen, calculando un píxel de salida por cada píxel de la imagen original. La Figura 3.6 presenta una representación parcial, gráfica y analítica, del proceso de convolución 2D. Entonces, el cálculo de los mapas de orientación convolucionados mediante de la convolución 2D es definido por

$$\begin{aligned} \mathrm{G}_{o_j}^{\Sigma_0}(x,y) &= \mathrm{G}_{o_j}(x,y) * G_{\Sigma_0}(p,q) \\ &= \sum_{p=-a}^{a} \sum_{q=-b}^{b} G_{\Sigma_0}(p+a, q+b) * \mathrm{G}_{o_j}(x-p, y-q) \end{aligned} \tag{3.11}$$

Donde, para centrar a G_{Σ_0} con el píxel de $\mathrm{G}_{o_j}^{\Sigma_0}$ a calcular, $a = \frac{(\text{dimensión en } x \text{ de } G_{\Sigma_0})-1}{2}$ y $b = \frac{(\text{dimensión en } y \text{ de } G_{\Sigma_0})-1}{2}$.

Desde una perspectiva algorítmica, el cálculo de $\mathrm{G}_{o_j}(x,y) * G_{\Sigma_0}(p,q)$ supone la realización de las siguientes operaciones:

1. Reflexión. Se rota $G_{\Sigma_0}(p,q)$ 180° teniendo como referencia su centro para obtener $G_{\Sigma_0}(-p,-q); \quad -a \le p \le a; \quad -b \le q \le b$
2. Se sobrepone $G_{\Sigma_0}(-p,-q)$ sobre la ventana de convolución de $\mathrm{G}_{o_j}^{\Sigma_0}(x,y)$.
3. Multiplica posición por posición los coeficientes de G_{Σ_0} por los pixeles de $\mathrm{G}_{o_j}^{\Sigma_0}$ dentro de la ventana de convolución obteniendo la secuencia producto

$$G_{\Sigma_0}(p,q)\mathrm{G}_{\boldsymbol{o}}(x-p,y-q);\quad -a\le p\le a,\quad -b\le q\le b$$

4. Suma todos los valores de la secuencia para obtener la respuesta en el píxel indicado.
5. Se desplaza G_{Σ_0} al siguiente píxel y se repite el proceso hasta concluir con todos los píxeles de $\mathrm{G}_{o_j}^{\Sigma_0}(x,y)$

Es necesario mencionar que para la implementación del proceso de convolución se aprovecha la propiedad de separabilidad que poseen de los filtros gaussianos, es decir, se tienen dos filtros gaussianos de tipo vector para efectuar la convolución unidimensional (convolución 1D) sobre G_{o_j}, el cual se define usando la nomenclatura de la convolución 2D como lo muestra la Ecuación 3.12:

$$\mathrm{G}_{o_j}^{\Sigma_0}(x,y)=\sum_{p=-a}^{a}\mathrm{G}_{o_j}(x+p,y)*G_{\Sigma_0}(p+a) \tag{3.12}$$

La cual representa el recorrido del G_{Σ_0} sin transponer (recorrido horizontal), y la Ecuación 3.13:

$$\mathrm{G}_{o_j}^{\Sigma_0}(x,y)=\sum_{p=-a}^{a}\mathrm{G}_{o_j}^{\Sigma_0}(x,y+p)*\left(G_{\Sigma_0}\right)'(p+a) \tag{3.13}$$

complementa la convolución 1D realizando el recorrido con el G_{Σ_0} transpuesto $\left(G_{\Sigma_0}\right)'$ (recorrido vertical). De esta manera, se reduce el tiempo del proceso de convolución, ya que para el ejemplo de convolución 2D que muestra la Figura 3.6 se necesitan 144 multiplicadores y 128 sumadores para efectuar el proceso. Mientras que para la convolución 1D que ilustra la Figura 3.7, se utilizan 96 multiplicadores y 64 sumadores.

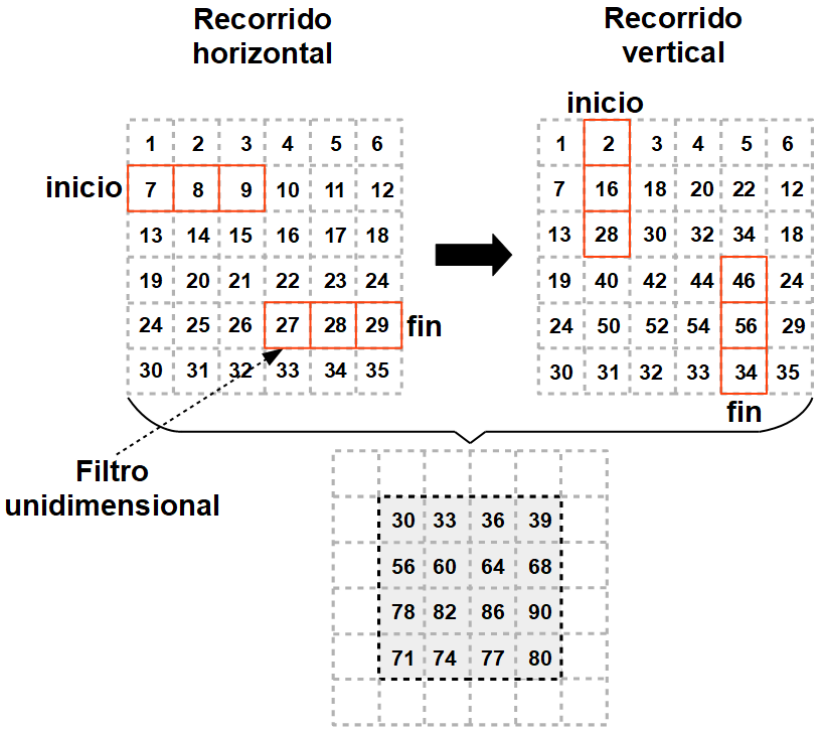


Figura 3.7 Proceso de convolución 1D.

El Algoritmo 3.4 muestra el pseudocódigo que realiza la tarea descrita.

Algoritmo 3.4 Pseudocódigo del algoritmo para la convolución de los mapas de orientación.

```
Función conv_mapas_orientacion (t)
    centro ← (t - 1) / 2; //Se calcula el centro del filtro
    Para j ← 0 Hasta j < T Con Paso 1 Hasta
        //Se realiza el recorrido horizontal
        Para y ← 0 Hasta y < alto Con Paso 1 Hasta
            Para x ← 0 Hasta x < ancho Con Paso 1 Hasta
                Para e ← 0 Hasta e < t Con Paso 1 Hasta
                    pixel ← pixel + (G_o[y][x]* G_Σ[e]);
                Fin Para
                G_o^Σ[y+centro][x+centro] ← pixel; pixel ← 0.0;
            Fin Para
        Fin Para
        //Se realiza el recorrido vertical
        Para y ← 0 Hasta y < alto Con Paso 1 Hasta
            Para x ← 0 Hasta x < ancho Con Paso 1 Hasta
                Para e ← 0 Hasta e < t Con Paso 1 Hasta
                    pixel ← pixel + (G_o^Σ[y][x]* G_Σ[e]);
                Fin Para
            G_o^Σ[y+centro][x+centro] ← pixel; pixel ← 0.0;
        Fin Para
    Fin Para
Fin Función
```

Los $G_{o_j}^{\Sigma_0}$ obtenidos por este subproceso son mostrados en la Figura 3.8.

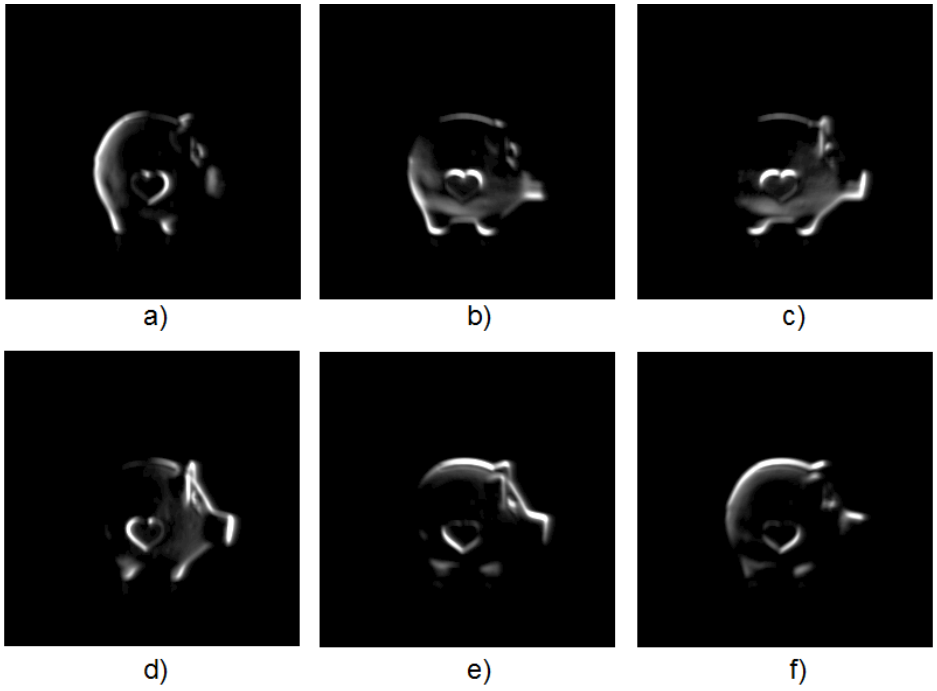


Figura 3.8 Mapas de orientación convolucionados del primer anillo: a) 0°, b) 60°, c) 120°, d) 180°, e) 240° y f) 300°.

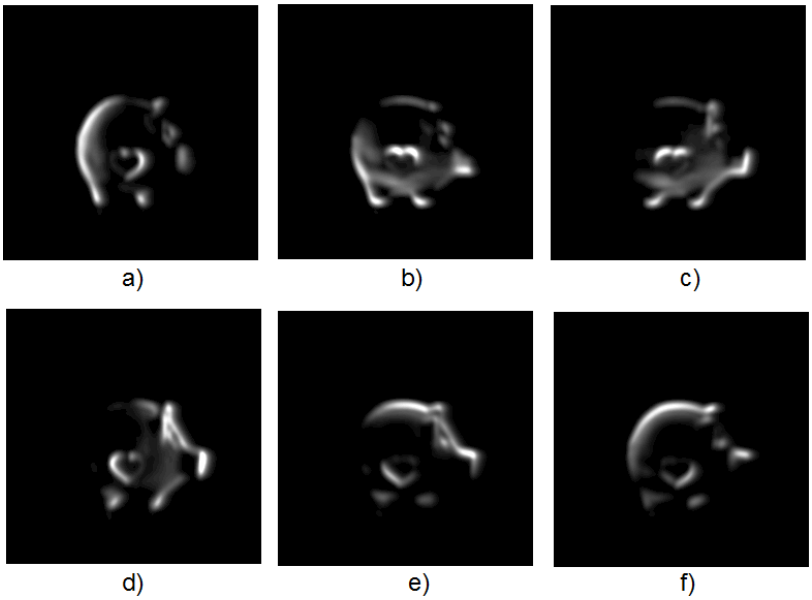


Figura 3.9 Mapas de orientación convolucionados del segundo anillo: a) 0°, b) 60°, c) 120°, d) 180°, e) 240° y f) 300°.

Subproceso 5. En este subproceso se obtienen los mapas de orientación convolucionados del segundo anillo $G_{o_j}^{\Sigma_1}$. Con este fin, se aplica la convolución 1D entre los $G_{o_j}^{\Sigma_0}$ y el filtro gaussiano G_{Σ_1}, quedando definida por las expresiones 3.10 y 3.11. Finalmente, se reutiliza el pseudocódigo 3.4 para obtener los $G_{o_j}^{\Sigma_1}$ de este subproceso (ver Figura 3.9).

Subproceso 6. Como último paso, en el subproceso 6 se realiza la convolución del mapa central, G_c con su respectivo filtro G_{Σ_c}. Para ello, se calcula el valor de Σ_c utilizando la expresión $\Sigma_c = \sqrt{(\Sigma_{c2})^2 - (\Sigma_{c1})^2}$, considerando valores para $\Sigma_{c2} = 1.6$ y $\Sigma_{c1} = 0.5$ (valores obtenidos de [36]). Seguido a esto, se calcula el tamaño de G_{Σ} utilizando la expresión 3.7, de donde se obtiene $t_c = 7$. Con estos parámetros, se diseña el G_{Σ_c} (cálculo de los coeficientes), mediante la expresión 3.8 y con ayuda del Algoritmo 3.3:

$$G_{\Sigma_c} = [0.038143, 0.112572, 0.215497, 0.267574, 0.215497, 0.112572, 0.038143] \quad (3.14)$$

Finalmente, se convoluciona el G_c con el G_{Σ_c} para obtener el mapa central convolucionado, $G_c^{\Sigma_c}$ mediante el Algoritmo 3.4. El $G_c^{\Sigma_c}$ obtenido se muestra en la Figura 3.10.

Figura 3.10 Mapa central convolucionado.

3.1.3 Generación de los vectores

El objetivo de este proceso es formar los vectores $\mathbf{h}_{\Sigma_c}$, $\mathbf{h}_{\Sigma_0}$ y $\mathbf{h}_{\Sigma_1}$ con ciertos valores denominados puntos de interés, los cuales contienen información relevante de $G_c^{\Sigma_c}$, $G_{o_j}^{\Sigma_0}$ y $G_{o_j}^{\Sigma_1}$, respectivamente. El algoritmo DAISY considera que la información relevante del objeto se encuentra en torno al origen del descriptor $cd(0,0°)$ y en torno a los T puntos $P_{o_j}^{k}(R_k, o_j)$ de las regiones que existen por cada anillo que se denominarán como *centro(s)*, cuyo radio de estas es igual a la desviación estándar, Σ. Nótese que el origen del descriptor y los *centros* están definidos por sus coordenadas polares, también es evidente que los *centros* se encuentran sobre los anillos del descriptor (véase la Figura 3.11). Entonces, h_{Σ_c} es calculado a partir de $cd(0,0°)$, mientras que $\mathbf{h}_{\Sigma_0}$ y $\mathbf{h}_{\Sigma_1}$ son calculados a partir de $P_{o_j}^{0}(R_0, o_j)$ y $P_{o_j}^{1}(R_1, o_j)$, respectivamente, existiendo T *centros* por anillo, uno en cada orientación definida ($j = 0,1,\dots,T-1$). Cabe recalcar que DAISY construye el vector de características con ciertos puntos de interés de los G_o^{Σ}, lo cual es equivalente al proceso que realiza el algoritmo SIFT para obtener sus histogramas. Es por ello, que en la presente investigación se adopta el término "vector" para referirse a los $\mathbf{h}_{\Sigma_c}$, $\mathbf{h}_{\Sigma_0}$ y $\mathbf{h}_{\Sigma_1}$, a diferencia de Tola *et al.*, en [7] que utilizan el término "histograma". Para cumplir con el objetivo de esta fase, se han definido los siguientes subprocesos.

Subproceso 1. En este subproceso se calculan las coordenadas de los T *centros* del primer anillo, $P_{o_j}^{0}(R_0, o_j)$, con respecto al centro de la imagen $c(x,y)$ y se representan en coordenadas rectangulares. Inicialmente, considerando la Tabla 3.1, se obtienen las coordenadas rectangulares de los puntos $P_{o_j}^{0}(R_0, o_j)$ haciendo uso de $x = R_0\cos(o_j)$ para obtener el valor en el eje de las abscisas y $y = R_0 sen(o_j)$ para el valor en el eje de las ordenadas.

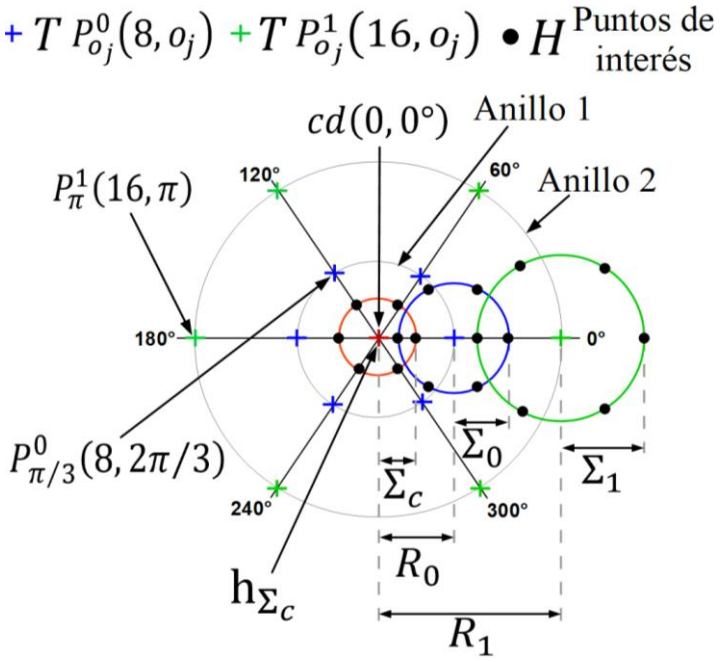


Figura 3.11 Puntos de interés y *centros* de los anillos para el descriptor DAISY.

Ahora, es necesario recordar que el origen, coordenada $(0,0)$, del descriptor se encuentra en su centro, mientras que el de la imagen (en este caso la imagen es $G_{o_j}^{\Sigma_0}$) se encuentra en la esquina superior izquierda. Por tanto, se deben determinar las coordenadas medias de la imagen mediante $cmi\_x = (ancho\_i - 1)/2$ y $cmi\_y = (alto\_i - 1)/2$, para luego sumarlas con las coordenadas (x, y), calculadas previamente a partir de las coordenadas polares de los $P_{o_j}^0(R_0, o_j)$, y de esta manera hacer coincidir el centro de la imagen con el centro del descriptor y ubicar correctamente los $P_{o_j}^0(R_0, o_j)$ sobre los $G_{o_j}^{\Sigma_0}$.

Subproceso 2. Este subproceso sigue el mismo procedimiento del subproceso 1 para calcular las coordenadas de los T *centros* del segundo anillo, $P_{o_j}^1(R_1, o_j)$. La Tabla 3.2 contiene las coordenadas polares y rectangulares de los *centros* de ambos anillos. El pseudocódigo que define las coordenadas rectangulares de todos los *centros* es mostrado en el Algoritmo 3.5.

Algoritmo 3.5 Pseudocódigo del algoritmo para calcular los puntos de interés $(\boldsymbol{u}, \boldsymbol{v})$.

```
1     Función centros
2         Para k ← 0 Hasta k < Q Con Paso 1 Hasta
3             //Centro de la imagen (x, y) – coordenadas medias
4             cmi_x ← (ancho-1) / 2; cmi_y ← (alto-1) / 2;
5             Para j ← 0 Hasta j < T Con Paso 1 Hasta
6                 //Se obtienen las coordenadas polares
7                 r ← (R(a+1)) / Q; //radio
8                 //Se convierten a coordenadas rectangulares
9                 P[j][0] ← cmi_x + (r*cos(o[j])); //x
10                P[j][1] ← cmi_y + (r*sin(o[j])); //y
11            Fin Para
12        Fin Para
13    Fin Función
```

Tabla 3.2 Coordenadas polares y rectangulares de los *centros* $P_{o_j}^k(R_k, o_j)$.

Anillo	***centros***	**Coordenadas**			
		Polares		**Rectangulares**	
		Radio (R_k)	**Ángulo (o_j)**	x	y
1	$P_0^0(8,0)$	8	0	$cmi\_x + (8)$	$cmi\_y + (0)$
	$P_{\pi/3}^0(8, \pi/3)$	8	$\pi/3$	$cmi\_x + (4)$	$cmi\_y + (6.93)$
	$P_{\pi/3}^0(8, 2\pi/3)$	8	$2\pi/3$	$cmi\_x + (-4)$	$cmi\_y + (6.93)$
	$P_{\pi}^0(8, \pi)$	8	π	$cmi\_x + (-8)$	$cmi\_y + (0)$
	$P_{\pi/3}^0(8, 4\pi/3)$	8	$4\pi/3$	$cmi\_x + (-4)$	$cmi\_y + (-6.93)$
	$P_{\pi/3}^0(8, 5\pi/3)$	8	$5\pi/3$	$cmi\_x + (4)$	$cmi\_y + (-6.93)$
2	$P_0^1(16,0)$	16	0	$cmi\_x + (16)$	$cmi\_y + (0)$
	$P_{\pi/3}^1(16, \pi/3)$	16	$\pi/3$	$cmi\_x + (8)$	$cmi\_y + (13.86)$
	$P_{\pi/3}^1(16, 2\pi/3)$	16	$2\pi/3$	$cmi\_x + (-8)$	$cmi\_y + (13.86)$
	$P_{\pi}^1(16, \pi)$	16	π	$cmi\_x + (-16)$	$cmi\_y + (0)$
	$P_{\pi/3}^1(16, 4\pi/3)$	16	$4\pi/3$	$cmi\_x + (-8)$	$cmi\_y + (-13.86)$
	$P_{\pi/3}^1(16, 5\pi/3)$	16	$5\pi/3$	$cmi\_x + (-8)$	$cmi\_y + (-13.86)$

Subproceso 3. En este subproceso se obtienen vectores en torno a cada *centro* $P_{o_j}^k(R_k, o_j)$ de los $G_{o_j}^{\Sigma_0}$ y $G_{o_j}^{\Sigma_1}$, los cuales serán utilizados para conformar a $\mathbf{h}_{\Sigma_0}$ y $\mathbf{h}_{\Sigma_1}$. El Algoritmo 3.6 muestra el pseudocódigo que define este subproceso.

Algoritmo 3.6 Pseudocódigo del algoritmo para la formación de los vectores de los anillos 1 y 2.

```
1     Función vectores_anillos (Σ)
2         Para q ← 0 Hasta q < H Con Paso 1 Hasta
3             h_Σ[q] ← G_o^Σ[P[q][1] + (Σ * sen(ô[q]))][P[q][0] + (Σ * cos(ô[q]))];
4         Fin Para
5     Fin Función
```

En cada centro, $P_{o_j}^k(R_k, o_j)$, se genera un vector de H puntos de interés, $\mathbf{v}_j^{P_{o_j}^k} = \left[b_q^{P_{o_j}^k}\right], q = 0, 1, \dots, H-1$. Tomando a $P_{o_j}^k(R_k, o_j)$ como origen, el valor de cada puntos de interés $b_q^{P_{o_j}^k}$ es igual al valor de $G_{o_j}^{\Sigma_k}$ definido por la coordenada polar $(\Sigma_k, \hat{o}_q)$, donde $\hat{o}_q$ es un vector que contiene las orientaciones en radianes de los H puntos de interés, las cuales son calculadas mediante: $\hat{o}_q = \frac{2\pi q}{T}$, obteniendo las siguientes orientaciones $\left\{0, \frac{\pi}{3}, \frac{2\pi}{3}, \pi, \frac{4\pi}{3}, \frac{5\pi}{3}\right\}$. Por lo tanto, existen H $\hat{o}_q$ por cada o_j definida. Entonces, para obtener las coordenadas de cada $b_q^{P_{o_j}^k}$, $(\Sigma_k, \hat{o}_q)$ son convertidas a coordenadas rectangulares mediante $\hat{x} = \Sigma_k \cos(\hat{o}_q)$ y $\hat{y} = \Sigma_k sen(\hat{o}_q)$ y añadidas a las coordenadas rectangulares del $P_{o_j}^k(R_k, o_j)$. Finalmente, los $\mathbf{h}_{\Sigma_k}$ son definidos por,

$$\mathbf{h}_{\Sigma_k} = \left[\mathbf{v}_0^{P_{o_0}^k}, \mathbf{v}_1^{P_{o_1}^k}, \dots, \mathbf{v}_{T-1}^{P_{o_{T-1}}^k}\right] \tag{3.15}$$

Cada $\mathbf{h}_{\Sigma_k}$ está integrado por $H \times T$ elementos.

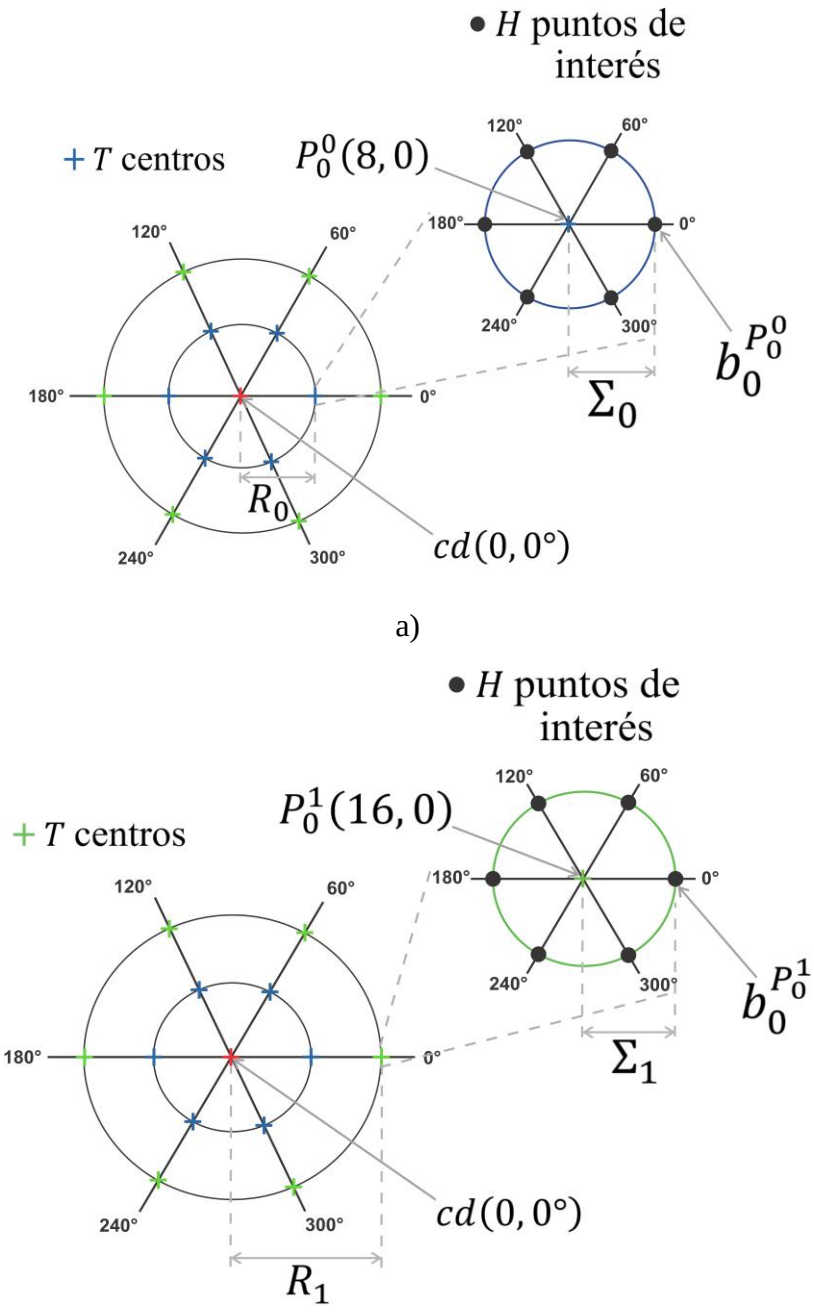


Figura 3.12 Ubicaciones de los puntos de interés para formar los vectores: a) $\mathbf{h}_{\Sigma_0}$ y b) $\mathbf{h}_{\Sigma_1}$.

La Figura 3.12 es una representación gráfica del proceso mencionado aplicado a ambos anillos. Para el caso en estudio, la dimensión de los $\mathbf{h}_{\Sigma_k}$, $H \times T = 6 \times 6 = 36$, evita que puedan mostrarse completos. Debido a esto, y únicamente con fines demostrativos, las expresiones 3.14 y 3.15 muestran los valores de algunos vectores $\mathbf{v}_j^{P_{o_j}^k}$.

$$\mathrm{v}_0^{P_{o_0}^0} = [0.209032, 0.076789, 0.208275, 0.254286, 0.179700, 0.192762] \qquad (3.16)$$

$$\mathrm{v}_0^{P_{o_0}^1} = [1.702641, 1.211104, 1.133158, 0.649545, 0.925045, 1.318509] \quad (3.17)$$

Subproceso 4. La tarea del último subproceso es obtener el vector $\mathbf{h}_{\Sigma_c}$ a partir del origen $cd(0, 0°)$ del $\mathrm{G}_c^{\Sigma_c}$. El Algoritmo 3.7 muestra el pseudocódigo que define este subproceso.

Algoritmo 3.7 Pseudocódigo para generar el vector $\mathbf{h}_{\Sigma_c}$.

```
1     Función vector_mapa_central
2         cmi_x ← 0; cmi_y ← 0;
3         //Centro de la imagen (x, y) - coordenadas medias
4         cmi_x ← (ancho-1) / 2;
5         cmi_y ← (alto-1) / 2;
6         Para q ← 0 Hasta q < H Con Paso 1 Hasta
7             h_Σc[q] ← G_c^Σ[cmi_y + (Σc*sen(o[q])][cmi_x + (Σc*cos(o[q])];
8         Fin Para
9     Fin Función
```

El vector $\mathbf{h}_{\Sigma_c}$ está formado por H puntos de interés, $b_q^{cd(0,0°)}$, que se obtienen a partir del origen $cd(0, 0°)$ del $\mathrm{G}_c^{\Sigma_c}$, el cual ya corresponde con el centro del descriptor. El valor de cada punto de interés $b_q^{cd(0,0°)}$ es igual al valor de $\mathrm{G}_c^{\Sigma_c}$ definido por la coordenada polar $(\Sigma_c, \hat{o}_q)$. Para obtener las coordenadas de cada $b_q^{cd(0,0°)}$, $(\Sigma_c, \hat{o}_q)$ son convertidas a coordenadas rectangulares mediante $x = \Sigma_c \cos(\hat{o}_q)$ y $y = \Sigma_c sen(\hat{o}_q)$ y añadidas a las coordenadas rectangulares del origen definidas por $c(cmi\_x, cmi\_y)$. La Figura 3.13 representa gráficamente de donde se obtendrán los H puntos de interés. Por otro lado, $\mathbf{h}_{\Sigma_c}$ está integrado por H puntos de interés y es definido por,

$$\mathbf{h}_{\Sigma_c} = \left[b_0^{cd(0,0°)}, b_1^{cd(0,0°)}, \ldots, b_{H-1}^{cd(0,0°)}\right] \quad (3.18)$$

El cual tiene los siguientes elementos,

$$\mathbf{h}_{\Sigma_c} = [0.705799, 0.099121, 0.01076, 1.117574, 0.014641, 0.012819] \quad (3.19)$$

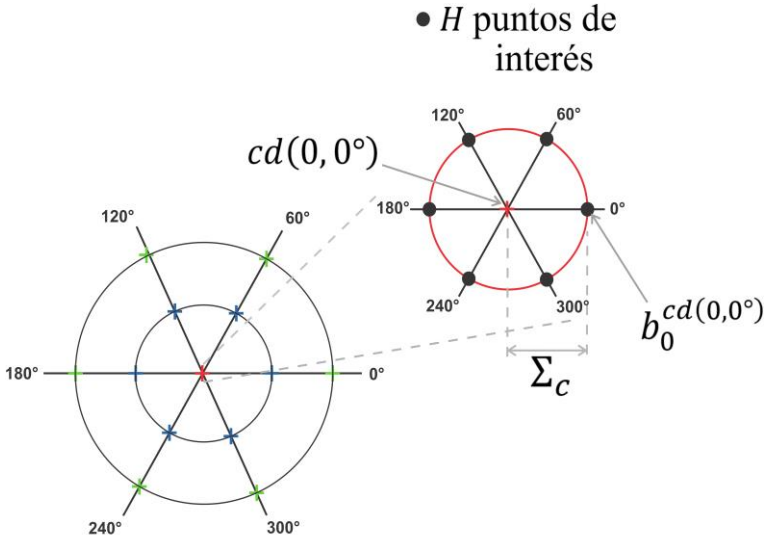


Figura 3.13 Generación del vector $\mathbf{h}_{\Sigma_c}$.

3.1.4 Generación del vector de características

El objetivo de este proceso es generar el vector de características o descriptor DAISY. Con este fin, dos subprocesos han sido definidos.

Subproceso 1. Como primer paso, los vectores que forma a $\mathbf{h}_{\Sigma_c}$, $\mathbf{h}_{\Sigma_0}$ y $\mathbf{h}_{\Sigma_1}$ deben ser normalizados en forma individual mediante,

$$\tilde{b}_q^{P_{o_j}^k} = \frac{b_q^{P_{o_j}^k}}{\max\left(\mathbf{v}_j^{P_{o_j}^k}\right)}, \text{ para obtener los } \tilde{\mathbf{v}}_j^{P_{o_j}^k} \text{ y por tanto los } \tilde{\mathbf{h}}_{\Sigma_k}$$
$$\tilde{b}_q^{cd(0,0°)} = \frac{b_q^{cd(0,0°)}}{\max\left(\mathbf{h}_{\Sigma_c}\right)}, \text{ para obtener } \tilde{\mathbf{h}}_{\Sigma_c} \tag{3.20}$$

En la ecuación 3.20 la nomenclatura $\tilde{\mathbf{h}}_{\Sigma_c}$ se usa para representar la versión normalizada de $\mathbf{h}_{\Sigma_c}$; así para todos los vectores y elementos de estos. El Algoritmo 3.8 contiene el pseudocódigo que describe el subproceso mencionado.

Algoritmo 3.8 Pseudocódigo para normalizar los vectores.

```
Función normalizacion
    Para q ← 0 Hasta q < H Con Paso 1 Hasta
        temp ← h_Σ[q];
        //Se busca el elemento mayor por vector
        Si temp > maximo Entonces
            maximo ← temp;
        Fin Si
    Fin Para
    //Se normaliza cada elemento del vector
    Para q ← 0 Hasta q < H Con Paso 1 Hasta
        h̃_Σ[q] ← h_Σ[q] / maximo;
    Fin Para
    maximo ← 0.0;
Fin Función
```

Subproceso 2. Finalmente, este subproceso da forma al descriptor DAISY, $D(u_o, v_o)$, mediante una simple operación de concatenación que une todos los elementos de $\mathbf{h}_{\Sigma_c}$, $\mathbf{h}_{\Sigma_0}$ y $\mathbf{h}_{\Sigma_1}$,

$$\begin{aligned}\boldsymbol{D}(u_o, v_o) &= \left[\tilde{\mathbf{h}}_{\Sigma_c}, \tilde{\mathrm{h}}_{\Sigma_0}, \tilde{\mathbf{h}}_{\Sigma_1}\right] \\ &= \left[\tilde{\mathbf{h}}_{\Sigma_c}, \tilde{\mathbf{v}}_j^{P_{o_j}^0}, \tilde{\mathbf{v}}_j^{P_{o_j}^1}\right] \\ &= \left[\tilde{b}_q^{cd(0,0°)}, \tilde{b}_q^{P_{o_j}^0}, \tilde{b}_q^{P_{o_j}^1}\right] \\ &= \left[\tilde{b}_0^{cd(0,0°)}, \tilde{b}_1^{cd(0,0°)}, \ldots, \tilde{b}_{H-1}^{cd(0,0°)},\right. \\ &\left.\tilde{b}_0^{P_0^0}, \tilde{b}_1^{P_0^0}, \ldots, \tilde{b}_{H-1}^{P_0^0}, \tilde{b}_0^{P_1^0}, \tilde{b}_1^{P_1^0}, \ldots, \tilde{b}_{H-1}^{P_1^0}, \ldots, \tilde{b}_{H-1}^{P_{H-1}^0},\right.\end{aligned} \tag{3.21}$$

$$\tilde{b}_0^{P_0^1}, \tilde{b}_1^{P_0^1}, \dots, \tilde{b}_{H-1}^{P_0^1}, \tilde{b}_0^{P_1^1}, \tilde{b}_1^{P_1^1}, \dots, \tilde{b}_{H-1}^{P_1^1}, \dots, \tilde{b}_{T-1}^{P_{H-1}^{Q-1}}\Big]$$

3.2 Modelado conceptual de las MAE

En esta sección se describe la implementación del modelado conceptual de las MAE, modelo que será utilizado como clasificador para realizar la evaluación del desempeño del descriptor DAISY. Esta implementación está basada en el mapa conceptual mostrado por la Figura 3.14, donde se distinguen dos fases generales:

1. Fase de aprendizaje de una MAE.
2. Fase de clasificación de una MAE.

Para cumplir con estas fases se considera lo expuesto en las subsecciones 2.4.1 y 2.4.2. Las MAE, en su fase de aprendizaje, reciben como parámetro de entrada un conjunto de descriptores $D(u_o, v_o)$ generado por DAISY y la clase a la que pertenece cada uno de ellos, estos elementos representan al conjunto fundamental, CF, el cual está representado por simplicidad mediante $\{(\mathbf{x}^\mu, y^\mu) | \mu = 1,2, \dots, p\}$, donde $\mathbf{x}^\mu$ es el vector de entrada y y^μ es la salida para indicar a que clase p pertenece el vector de entrada. Por otro lado, en la fase de clasificación se le presenta a la MAE el descriptor DAISY del que se desea conocer a cuál clase pertenece. Con la finalidad de simplificar el análisis del modelado conceptual de las MAE, y sabiendo que la expresión 3.19 define al descriptor DAISY con un tamaño $\boldsymbol{D}_s = S * H$, este descriptor será representado simplemente como $\mathbf{x} = [x_j]_{\boldsymbol{D}_s - 1}$, donde $j = 0, 1, \dots, \boldsymbol{D}_s - 1$. En las siguientes subsecciones se describen las fases mencionadas, haciendo énfasis en los subprocesos que las forman.

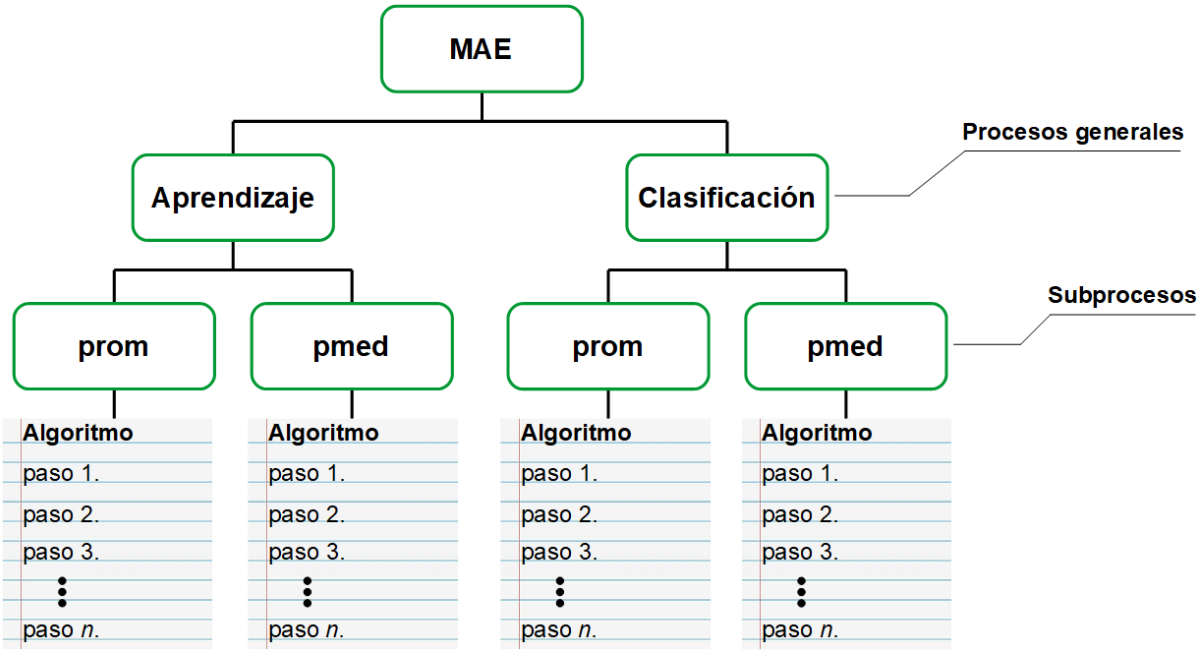


Figura 3.14 Estructura secuencial del modelado conceptual de las MAE.

3.2.1 Fase de aprendizaje de una MAE

Esta subsección tiene como objetivo describir el proceso de generación de la matriz **M**. Para ello, se han realizado las implementaciones de los operadores prom y pmed. Por tal motivo, se han definido los siguientes subprocesos para lograr el objetivo de esta subsección.

Subproceso 1. La tarea de este subproceso es calcular y almacenar el CF, que servirá para generar a **M**. El CF es obtenido usando la implementación secuencial del descriptor DAISY para calcular el correspondiente vector de características (ver Algoritmo 3.9).

Algoritmo 3.9 Pseudocódigo para obtener el CF.

```
Función lectura_patrones
    Para e ← 0 Hasta e < p Con Paso 1 Hasta
        Para l ← 0 Hasta l < q Con Paso 1 Hasta
            LeerImagen (); //Se lee la imagen
            DAISY (); //Se ejecuta el descriptor DAISY
            Para j ← 0 Hasta j < Ds Con Paso 1 Hasta
                CF[e][j] ← x[j];
            Fin Para
        Fin Para
    Fin Para
Fin Función
```

Subproceso 2. Este subproceso cumple con la tarea de definir la implementación del operador prom y utilizarlo para generar a **M**. En el pseudocódigo del Algoritmo 3.10 se puede observar la descripción de la generación de la matriz **M** cuando el operador prom es usado.

Algoritmo 3.10 Pseudocódigo del operador prom para generar a **M.**

```
Función op_prom
    Si q == 1 Entonces
        Para e ← 0 Hasta e < p Con Paso 1 Hasta
            Para j ← 0 Hasta j < Ds Con Paso 1 Hasta
                M[k][j] ← CF[k][j];
            Fin Para
        Fin Para
    Fin Si
    Si No Entonces
        Para e ← 0 Hasta e < p Con Paso 1 Hasta
            Para j ← 0 Hasta j < Ds Con Paso 1 Hasta
                Para l ← (e*q) Hasta l < (e*q) + q Con Paso 1 Hasta
                    sum ← sum + CF[l][j];
                Fin Para
                M[k][j] ← sum / q;
            Fin Para
        Fin Para
    Fin Si No
Fin Si
```

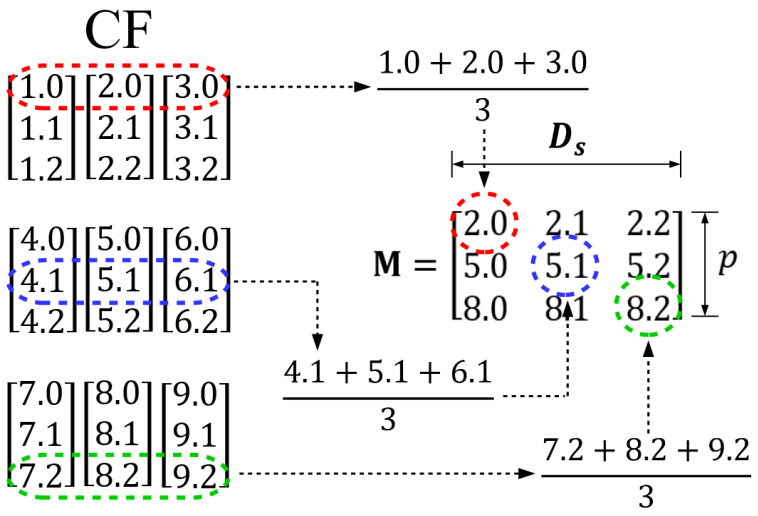


Figura 3.15 Generación de **M** por el operador prom.

En la Figura 3.15 se puede observar como a través del operador prom se genera la memoria **M**, utilizando la expresión 2.7, la cual sirve para calcular cada elemento $\phi_{i,j}$ de **M**. De esta manera se realiza este procedimiento tantas veces como clases existan para generar a **M**, cuyas dimensiones son $p \times D_s$.

Subproceso 3. Al igual que el subproceso anterior, el objetivo de este subproceso general es explicar a detalle la formación de **M** usando el operador pmed. El Algoritmo 3.11 define este procedimiento.

Algoritmo 3.11 Pseudocódigo del operador pmed para generar a **M.**

```
1     Función op_pmed
2         temp[] ← {{0.0}};
3         γ[][] ← {{0.0}};
4         λ[][] ← {{0.0}};
5         aux ← 0.0;
6         centro ← 0;
7         res ← 0;
8         e ← 0;
9         max ← 0.0;
10        min ← 2.0;
11        //Se realiza el proceso de aprendizaje
12        Para e ← 0 Hasta e < p Con Paso 1 Hasta
13             Para j ← 0 Hasta j < Ds Con Paso 1 Hasta
14                  Para l ← (e*q) Hasta l < (e*q) + q Con Paso 1 Hasta
15                       E[g] ← CF[e][j];
16                       g ← g + 1;
17                  Fin Para
18                  //Se genera el vector "gama" de máximos
19                  Para l ← 0 Hasta l < q Con Paso 1 Hasta
20                       Si (E[l] > max) Entonces
21                          max ← E[l];
22                       Fin Si
23                  Fin Para
24                  //Se genera el vector "lambda" de mínimos
```

```
25                  Para l ← 0 Hasta l < q Con Paso 1 Hasta
26                       Si (E[l] < min) Entonces
27                          min ← E[l];
28                       Fin Si
29                  Fin Para
30                  //Se genera la memoria M
31                  γ[k][j] ← max;
32                  λ[k][j] ← min;
33                  M[k][j] ← (γ[k][j] + λ[k][j]) / 2;
34              Fin Para
35          Fin Para
36      Fin Función
```

En el seudocódigo del Algoritmo 3.11, primeramente, se obtiene un vector **E** formado por el elemento j de cada patrón que corresponde a la i-ésima clase. Despues, se realiza la búsqueda del máximo y mínimo elemento del vector para generar las matrices $\boldsymbol{\gamma}$ y $\boldsymbol{\lambda}$, respectivamente. Finalmente, se realiza el promedio de las matrices para formar a **M**.

3.2.2 Fase de clasificación de una MAE

Esta fase de la MAE ejemplificada por la Figura 3.16, se encarga de generar el índice i de la clase a la cual pertenece el vector **x** cuando se le presenta a **M**. Por lo cual, el objetivo a cumplir en esta subsección es la explicación de la implementación secuencial del algoritmo usando la expresión 2.14, la cual modela el proceso de clasificación. Este algoritmo es usado para ambos casos, aquellos que usaron los operadores prom y pmed para generar a **M**. Se han definido los siguientes subprocesos, necesarios para el cumplimiento del objetivo.

Subproceso 1. En este subproceso se obtiene el valor absoluto de la diferencia entre los lj elementos de **M** y los j elementos de **x**, $|m_{lj} - x|$, esto genera una matriz **A** de $p \times \boldsymbol{D}_s$ elementos.

Subproceso 2. Posteriormente, se obtiene un vector columna **c** formado por los máximos elementos de cada vector **f** $\left(a_{f_0}, a_{f_1}, \ldots\ a_{f_{\boldsymbol{D}_s-1}}\right)$ de **A**.

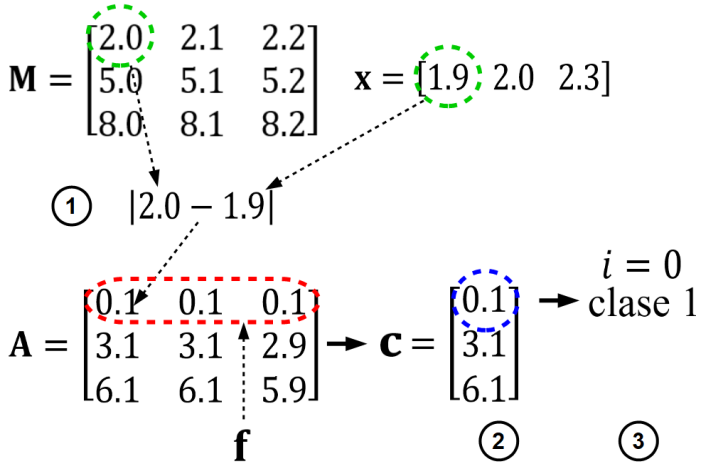


Figura 3.16 Obtención del índice $\boldsymbol{i}$ de la clase asociada.

Subproceso 3. Finalmente, se obtiene la posición del mínimo elemento de **c**, el cual corresponde a la *i*-ésima clase utilizada. Estos subprocesos se ejemplifican en la Figura 3.16. Cabe mencionar que para la presente investigación sólo se utilizaron los operadores pmed y prom, debido a que el rendimiento de las MAE con los operadores med y sum no presenta resultados satisfactorios.

3.3 Interfaz de usuario

Esta sección describe las funcionalidades con las que cuenta la GUI diseñada especialmente para la realizar las evaluaciones correspondientes al modelado secuencial del descriptor DAISY y su posterior implementación en hardware, la cual fue desarrollada sobre un lenguaje de alto nivel (ver Figura 3.17). Para la comunicación entre la GUI y la implementación en hardware del descriptor DAISY se permite mediante la utilización del dispositivo DLP-USB245M, el cual contiene una interfaz paralela para la transmisión y recepción de datos, logrando velocidades hasta de 8 millones de bits (1 Megabyte) por segundo.

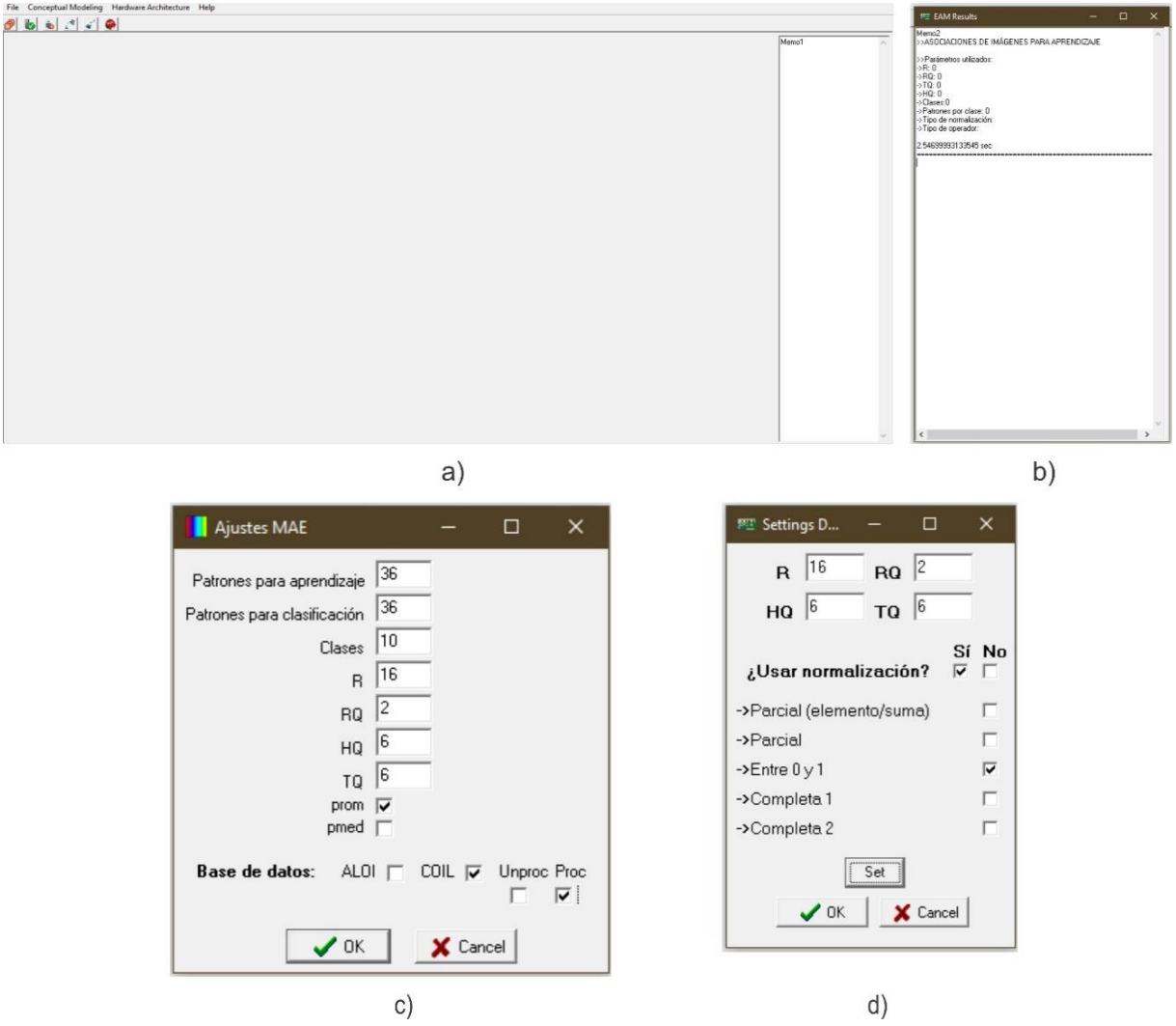


Figura 3.17 GUI: a) Ventana principal, b) Visualización de resultados, c) Ventana de configuración de las MAE y d) Ventana para la configuración de DAISY.

Esta GUI cuenta con las siguientes características.

- Carga y visualización de la(s) imagen(es) que se enviará(n) al FPGA.
- Recepción y visualización del vector de características que el descriptor DAISY genera como salida.
- Acceso a la configuración de los parámetros del descriptor DAISY.
- Acceso a la configuración del clasificador, el cual permite al usuario establecer el número de: clases a clasificar, patrones que se utilizarán en el aprendizaje y clasificación, así como también el operador a usar y la base de datos que se empleará para generar el conjunto fundamental, CF.
- Visualización de la matriz de confusión generada de la clasificación.

3.4 Resultados experimentales

En esta sección se presentan los resultados experimentales generados por el modelado conceptual del descriptor DAISY. Para ello, se empleó la base de datos ALOI en los experimentos conducidos en esta sección, la cual es comúnmente usada en la evaluación de sistemas de reconocimiento de objetos. Por otro lado, los operadores prom y pmed fueron utilizados en las MAE, debido a que presentaron un alto desempeño para la tarea de reconocimiento. Por el contrario, los operadores sum y med proporcionaron porcentajes de reconocimiento inferiores al 50%, por lo que no se incluyeron estos resultados en la presente sección y en la sección de resultados experimentales del capítulo siguiente.

ALOI introducida por Geusebroek *et al.*, en [37], la cual contiene una colección en color de mil objetos pequeños, en donde se variaron el ángulo de visión, ángulo de iluminación y la temperatura de color (cálido, neutro y frío) para cada objeto. Para esta evaluación se ha elegido el conjunto de imágenes en escala de grises de 384×288 píxeles donde el ángulo de visión del objeto varía en 72 direcciones (ver Figura 3.18).

Figura 3.18 Base de datos ALOI.

La visualización del rendimiento del descriptor DAISY interactuando con las MAE se realizará mediante la matriz de confusión. La matriz de confusión se utiliza comúnmente para presentar resultados de clasificación, la cual contiene información sobre clasificaciones reales y predichas por un sistema de clasificación. Esta información se estructura mediante columnas que contienen a las clases predichas por el clasificador y filas que contienen a las clases reales. En la diagonal de esta matriz se encuentran las predicciones exitosas y si los valores que se encuentran fuera de esta diagonal son iguales a cero, indican que se obtuvieron resultados de clasificación perfectos. Por lo contrario, si existen valores diferentes de cero fuera de la diagonal quiere decir que el clasificador confundió una clase con otra. Por lo tanto, el uso de la matriz de confusión es muy práctico para exponer los resultados obtenidos por el clasificador, debido a que toda la información que se requiere para determinar si el sistema ofrece o no un rendimiento favorable, se encuentra dispuesta en una sola tabla.

Para ello, la matriz de confusión utiliza diferentes términos que ayudan en la definición del rendimiento del clasificador, estos términos se muestran en la Tabla 3.3.

Tabla 3.3 Matriz de confusión de ejemplo.

Valores reales	**Clasificador**	
	Negativos	Positivos
Negativos	VN	FN
Positivos	FP	VP

En donde:

- VN (Verdaderos Negativos), corresponden a las asociaciones negativas clasificadas correctamente
- FN (Falsos Negativos), corresponden a las asociaciones negativas que fueron clasificadas incorrectamente.
- FP (Falsos Positivos), corresponden a las asociaciones positivas que fueron clasificadas incorrectamente.
- VP (Verdaderos Positivos), corresponden a las asociaciones positivas correctamente clasificadas.

Un aspecto importante de una matriz de confusión son sus métricas asociadas, las más importantes son mencionadas a continuación,

La exactitud es medida en proporción al total de asociaciones correctas, dada por:

$$exactitud = \frac{VP + VN}{VP + VN + FP + FN} \tag{3.22}$$

La precisión también es medida en base a la proporción de casos asociados correctamente que realmente son correctos por:

$$precision = \frac{VP}{VP + FP} \tag{3.23}$$

Por otro lado, se tiene la razón de verdaderos positivos (RVP) la cual mide los casos positivos que fueron correctamente asociados:

$$RVP = \frac{VP}{VP + FN} \tag{3.24}$$

Por último, se mide la proporción de casos negativos que fueron correctamente asociados, denominado como razón de verdaderos negativos (RVN).

$$RVN = \frac{VN}{VN + FP} \tag{3.25}$$

La tasa de error (TE) es otra métrica importante que indica la frecuencia con la que el clasificador predice erróneamente.

$$TE = \frac{FP + FN}{VP + VN + FP + FN} \tag{3.26}$$

3.4.3 Evaluación del modelado conceptual

En esta subsección se presenta el rendimiento del descriptor DAISY al ser evaluado con las MAE. Para ello, se realizaron experimentos en los cuales se usaron los operadores prom y pmed y varias configuraciones en los parámetros del descriptor DAISY.

En el primer experimento se establecen los siguientes valores en los parámetros del descriptor DAISY: $R_0 = 8$, $R_1 = 16$, $Q = 2$, $H = 6$ y $T = 6$. Siendo el operador pmed el utilizado para 3 clases de objetos que se muestran en la Figura 3.19.

En la Tabla 3.4 se observa la matriz de confusión que se generó en este experimento.

Tabla 3.4 Matriz de confusión del primer experimento usando el operador pmed.

	MAE		
Clases	Cesto1	Sombrero	Cesto2
Cesto1	20	0	0
Sombrero	0	20	0
Cesto2	0	0	20

Figura 3.19 Objetos usados para el primer y segundo experimento.

En la Tabla 3.5 se presentan los resultados del rendimiento de este experimento al aplicar las métricas antes mencionadas.

Tabla 3.5 Resultados del primer experimento usando el operador pmed.

Clases	exactitud	precisión	RVP	RVN	TE
Cesto1	1	1	1	1	0
Sombrero	1	1	1	1	0
Cesto2	!	1	1	1	0

Para el segundo experimento se empleó el operador prom, mientras que el número de clases (ver Figura 3.19) y los valores en los parámetros del descriptor DAISY se mantuvieron sin cambios. La matriz de confusión formada por este experimento se expone en la Tabla 3.6.

Tabla 3.6 Matriz de confusión del segundo experimento usando el operador prom.

	MAE		
Clases	Cesto1	Sombrero	Cesto2
Cesto1	20	0	0
Sombrero	0	20	0
Cesto2	0	0	20

En la Tabla 3.7 se presentan los resultados del rendimiento de este experimento al aplicar las métricas antes mencionadas.

Tabla 3.7 Resultados del segundo experimento usando el operador prom.

Clases	exactitud	precisión	RVP	RVN	TE
Cesto1	1	1	1	1	0
Sombrero	1	1	1	1	0
Cesto2	!	1	1	1	0

Para el tercer experimento, se mantienen los mismos valores en el operador DAISY. Se utiliza el operador pmed y se eligieron 5 clases de objetos (ver Figura 3.20).

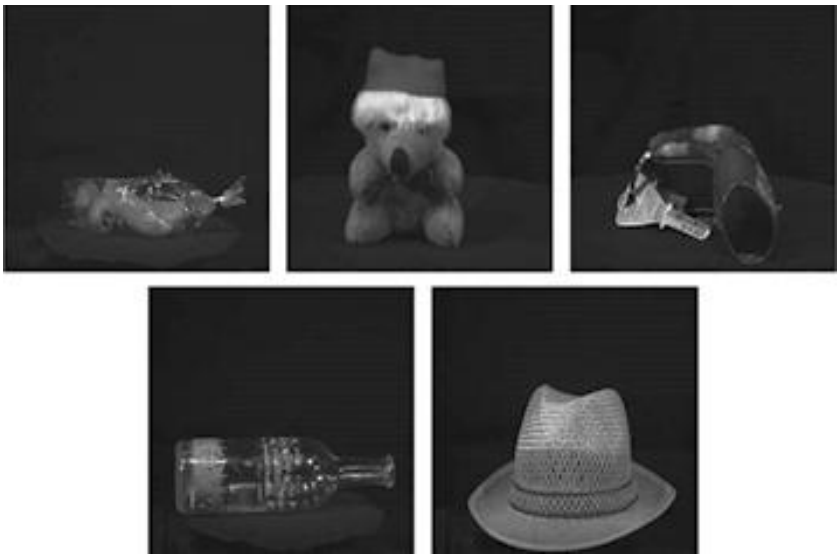


Figura 3.20 Objetos usados para el tercer y cuarto experimento.

La Tabla 3.8 presenta la matriz de confusión resultante de este experimento.

Tabla 3.8 Matriz de confusión del tercer experimento usando el operador pmed.

	MAE				
Clases	Chiles	Peluche	Llavero	Botella	Sombrero
Chiles	34	2	0	0	0
Peluche	0	35	1	0	0
Llavero	0	2	34	0	0
Botella	4	2	2	28	0
Sombrero	0	0	0	0	36

Los resultados del rendimiento de este experimento se pueden observar en la Tabla 3.9, la cual contiene las métricas asociadas a la matriz de confusión anteriormente mostrada.

Tabla 3.9 Resultados del tercer experimento usando el operador pmed.

Clases	**exactitud**	**precisión**	**RVP**	**RVN**	**TE**
Chiles	0.9667	0.8947	0.9444	0.9722	0.033333
Peluche	0.9611	0.8537	0.9722	0.9583	0.038889
Llavero	0.9722	0.9189	0.9444	0.9791	0.027778
Botella	0.9556	1	0.7777	1	0.044444
Sombrero	1	1	1	1	0

Ahora, para el cuarto experimento se ha utilizado el operado prom, mientras que la configuración de los parámetros del descriptor DAISY se mantuvo sin ninguna modificación, así como también los objetos a clasificar. Se puede consultar la Tabla 3.10 para visualizar la matriz de confusión que se obtuvo al efectuar este experimento.

Tabla 3.10 Matriz de confusión del cuarto experimento usando el operador prom.

	MAE				
Clases	Chiles	Peluche	Llavero	Botella	Sombrero
Chiles	33	3	0	0	0
Peluche	0	33	3	0	0
Llavero	0	7	29	0	0
Botella	0	4	0	32	0
Sombrero	0	1	0	0	35

Seguidamente, el rendimiento de este experimento se puede observar en la Tabla 3.11, la cual contiene los valores correspondientes a las métricas calculadas a partir de la matriz de confusión anterior.

Tabla 3.11 Resultados del cuarto experimento usando el operador prom.

Clases	**exactitud**	**precisión**	**RVP**	**RVN**	**TE**
Chiles	0.9833	1	0.9166	1	0.016667
Peluche	0.9000	0.6875	0.9166	0.8958	0.1
Llavero	0.9444	0.9063	0.8055	0.9791	0.055556
Botella	0.9778	1	0.8888	1	0.022222
Sombrero	0.9944	1	0.9722	1	0.005556

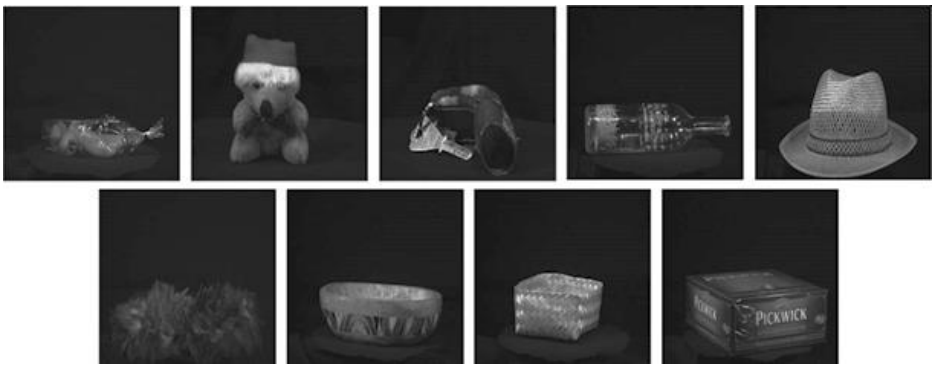


Figura 3.21 Objetos usados para el quinto y sexto experimento.

Posteriormente, para realizar el quinto experimento se ajustan los parámetros del operador DAISY a: $R_0 = 7.5$, $R_1 = 15$, $Q = 2$, $H = 8$ y $T = 8$. Además, se utiliza el operador pmed para el entrenamiento y clasificación de las MAE, así como también se eligieron 9 clases (ver Figura 3.21). Para ello, la matriz de confusión generada construida en este experimento se muestra en la Tabla 3.12.

Tabla 3.12 Matriz de confusión del quinto experimento usando el operador pmed.

					MAE				
Clases	Chiles	Peluche	Llavero	Botella	Sombrero	Plumero	Tazón	Cesto	Tea
Chiles	33	1	0	0	0	0	1	1	0
Peluche	0	34	2	0	0	0	0	0	0
Llavero	0	3	30	0	0	0	0	3	0
Botella	4	3	0	25	0	3	0	1	0
Sombrero	0	3	0	0	33	0	0	0	0
Plumero	9	2	0	0	0	25	0	0	0
Tazón	0	2	0	0	0	2	32	0	0
Cesto	0	0	6	0	0	0	0	30	0
Tea	0	2	4	0	0	0	0	0	30

De la cual se calculan las métricas correspondientes en base a la matriz de confusión, mismas que se presentan en la Tabla 3.13 para observar el rendimiento del descriptor DAISY.

Tabla 3.13 Resultados del quinto experimento usando el operador pmed.

Clases	**exactitud**	**Precisión**	**RVP**	**RVN**	**TE**
Chiles	0.9456	0.7174	0.9166	0.9496	0.054422
Peluche	0.9388	0.6800	0.9444	0.9379	0.061224
Llavero	0.9388	0.7143	0.8333	0.9534	0.061224
Botella	0.9626	1	0.6944	1	0.037415
Sombrero	0.9898	1	0.9166	1	0.010204
Plumero	0.9456	0.8065	0.6944	0.9806	0.054422
Tazón	0.9830	0.9697	0.8888	0.9961	0.017007
Cesto	0.9660	0.8824	0.8333	0.9826	0.033951
Tea	0.9815	1	0.8333	1	0.020408

De la misma forma que el experimento anterior, en este experimento se ha usado la misma configuración de los valores iniciales del operador DAISY y las mismas clases de objetos (ver Figura 3.21). A diferencia del experimento anterior, en este se utiliza el operador prom. Como resultado de este experimento, en la Tabla 3.14 se muestra la matriz de confusión.

Tabla 3.14 Matriz de confusión del sexto experimento usando el operador prom.

	MAE								
Clases	Chiles	Peluche	Llavero	Botella	Sombrero	Plumero	Tazón	Cesto	Tea
Chiles	30	1	0	0	0	4	1	0	0
Peluche	0	36	0	0	0	0	0	0	0
Llavero	0	6	25	0	0	0	0	5	0
Botella	0	3	0	26	0	0	6	1	0
Sombrero	0	5	0	0	31	0	0	0	0
Plumero	1	2	0	0	0	28	5	0	0
Tazón	0	6	0	0	0	1	29	0	0
Cesto	0	2	8	0	0	0	0	26	0
Tea	0	4	11	0	0	0	0	0	21

Partiendo de la matriz de confusión anterior, se han calculado las métricas asociadas a ella, mismas que se presentan en la Tabla 3.15. Las cuales indican el rendimiento del operador DAISY interactuando con las MAE.

Tabla 3.15 Resultados del sexto experimento usando el operador prom

Clases	**exactitud**	**precisión**	**RVP**	**RVN**	**TE**
Chiles	0.9769	0.9677	0.8333	0.9962	0.023102
Peluche	0.9043	0.5538	1	0.8913	0.09571
Llavero	0.9010	0.5682	0.6944	0.9288	0.09901
Botella	0.9670	1	0.7222	1	0.033003
Sombrero	0.9835	1	0.8611	1	0.016502
Plumero	0.9571	0.8485	0.7777	0.9812	0.042904
Tazón	0.9373	0.7073	0.8055	0.9550	0.062706
Cesto	0.9506	0.8125	0.7222	0.9791	0.049383
Tea	0.9537	1	0.5833	1	0.049505

Para el siguiente experimento, se han elegido 15 clases (ver Figura 3.22), así mismo, los parámetros en el descriptor DAISY se modificaron, quedando en: $R_0 = 8$, $R_1 = 16$, $Q = 2$, $H = 6$, $T = 6$. El operador utilizado en este experimento fue el prom. En la Tabla 3.16 se muestra la matriz de confusión obtenida del experimento.

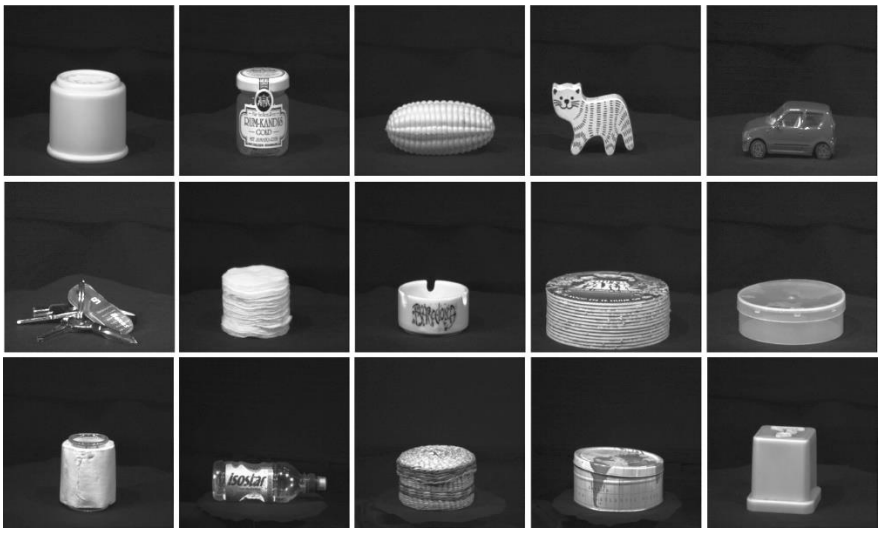


Figura 3.22 Objetos usados para el séptimo y octavo experimento.

Tabla 3.16 Matriz de confusión del séptimo experimento usando el operador prom.

								MAE						
Clases	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	10	0	0	0	0	0	7	0	0	3	0	0	0	0
3	0	0	18	0	0	0	0	1	0	0	0	0	1	0	0
4	0	0	0	17	0	0	0	1	0	2	0	0	0	0	0
5	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	18	0	0	0	0	0	0	0	2	0
7	0	0	0	0	0	0	18	0	0	2	0	0	0	0	0
8	0	0	0	0	2	1	0	13	0	0	4	0	0	0	0
9	0	0	0	0	0	0	0	2	18	0	0	0	0	0	0
10	0	0	0	8	0	0	5	0	0	7	0	0	0	0	0
11	0	0	0	0	0	0	0	6	0	0	14	0	0	0	0
12	0	1	0	0	0	0	0	5	0	0	0	13	1	0	0
13	0	1	1	0	0	0	0	7	0	0	0	0	11	0	0
14	1	0	0	0	1	0	0	0	0	0	0	0	0	18	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20

En la Tabla 3.17 se muestran los resultados de rendimiento calculados a partir de la matriz de confusión anterior.

Tabla 3.17 Resultados del séptimo experimento usando el operador prom.

Clases	**exactitud**	**precisión**	**RVP**	**RVN**	**TE**
1	0.9967	0.9524	0.9	0.99643	0.01
2	0.96	0.8333	0.9	0.98214	0.02333
3	0.99	0.9474	0.65	0.89643	0.12
4	0.9633	0.68	0.9	1	0.00667
5	0.99	0.8696	0.35	0.98571	0.05667
6	0.99	0.9474	0.9	0.99643	0.01
7	0.9767	0.7826	0.9	0.98214	0.02333
8	0.88	0.3095	0.65	0.89643	0.12
9	0.9933	1	0.9	1	0.00667
10	0.9433	0.6364	0.35	0.98571	0.05667
11	0.9567	0.6667	0.7	0.975	0.0433
12	0.9767	1	0.65	1	0.02333
13	0.9633	0.8462	0.55	0.99286	0.03667
14	0.9867	0.9	0.9	0.99286	0.01333
15	1	1	1	1	0

Para el octavo experimento, se utiliza el operador pmed y el resto de las configuraciones permanecen sin modificaciones. En la Tabla 3.18 se observa la matriz de confusión generada.

Tabla 3.18 Matriz de confusión del octavo experimento usando el operador pmed.

							MAE								
Clases	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	19	0	0	0	0	0	1	0	0	0	0	0	0	0
3	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0
7	0	0	1	0	0	0	15	1	1	0	0	2	0	0	0
8	0	0	0	0	2	0	0	18	0	0	0	0	0	0	0
9	0	0	12	0	0	0	0	2	6	0	0	0	0	0	0
10	0	0	3	0	0	0	0	0	0	17	0	0	0	0	0
11	0	2	0	0	0	0	0	5	0	0	13	0	0	0	0
12	0	0	0	0	0	0	0	1	0	0	0	19	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0
14	2	0	0	0	0	0	0	0	0	0	0	0	0	18	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20

Los resultados de rendimiento que se calcularon a partir de la matriz de confusión anterior, se pueden apreciar en la Tabla 3.19.

Tabla 3.19 Resultados del octavo experimento usando el operador pmed.

Clases	**exactitud**	**precisión**	**RVP**	**RVN**	**TE**
1	0.9933	0.9091	1	0.99286	0.00667
2	0.99	0.9048	0.95	0.99286	0.01
3	0.9233	0.4651	1	0.91786	0.07667
4	0.9033	0.4082	1	0.89643	0.09667
5	0.9933	0.9091	1	0.99286	0.00667
6	1	1	1	1	0
7	0.9333	0.9421	0.937	0.879	0.0667
8	0.9633	0.6667	0.9	0.96786	0.03667
9	0.9533	1	0.3	1	0.04667
10	0.9333	0.96	0.9	0.9875	0.0667
11	0.9767	1	0.65	1	0.02333
12	0.9967	1	0.95	1	0.00333
13	1	1	1	1	0
14	0.9933	1	0.9	1	0.00667
15	1	1	1	1	0

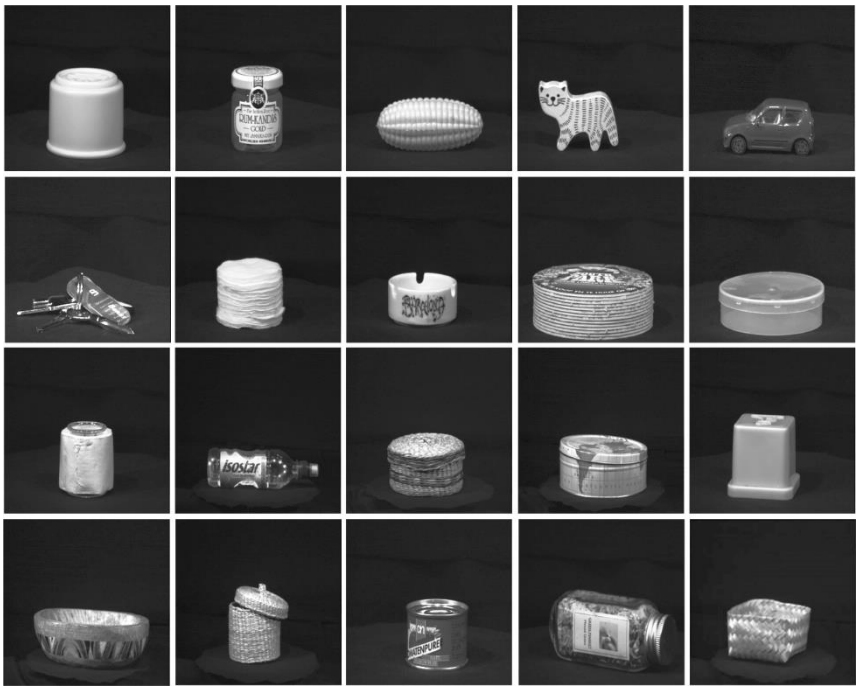


Figura 3.23 Objetos usados para el noveno y décimo experimento.

Como parte del noveno experimento realizado, se eligieron 20 clases las cuales se pueden apreciar en la Figura 3.23 y se utilizó el operador pmed. Los parámetros del descriptor DAISY se mantienen sin ninguna modificación. La matriz de confusión obtenida es mostrada por la Tabla 3.20.

Tabla 3.20 Matriz de confusión del noveno experimento usando el operador pmed.

	MAE																			
Clases	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
1	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	19	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	3	0	0	14	3	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	18	0	0	0	0	0	0	0	0	0	2	0	0
9	0	0	12	0	0	0	0	2	6	0	0	0	0	0	0	0	0	0	0	0
10	0	0	2	0	0	0	0	3	0	15	0	0	0	0	0	0	0	0	0	0
11	0	2	0	0	0	0	0	5	0	0	13	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	1	0	0	0	19	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	18	0	0	0	2	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0
16	0	2	0	0	0	0	0	0	4	0	0	0	0	0	0	16	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	13	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20

El rendimiento se puede consultar en la Tabla 3.21, la cual contiene las métricas asociadas a la matriz de confusión anterior.

Tabla 3.21 Resultados del noveno experimento usando el operador pmed.

Clases	exactitud	precisión	RVP	RVN	TE
1	1	1	1	1	0
2	0.9875	0.8261	0.95	0.98947	0.0125
3	0.9425	0.4651	1	0.93947	0.0575
4	0.9275	0.4082	1	0.92368	0.0725
5	1	1	1	1	0
6	1	1	1	1	0
7	0.95	0.873	0.8	0.97	0.05
8	0.9575	0.5455	0.9	0.96053	0.0425
9	0.965	1	0.3	1	0.035
10	0.95	0.89	0.9	0.95	0.05
11	0.95	0.5	0.65	0.96579	0.05
12	0.95	0.5	0.95	0.95	0.05
13	1	1	1	1	0
14	0.995	1	0.9	1	0.005
15	1	1	1	1	0
16	0.95	0.92	0.875	0.88	0.05
17	0.95	0.91	0.9	0.83	0.05
18	0.99	0.8333	1	0.98947	0.01
19	1	1	1	1	0
20	1	1	1	1	0

Para el último experimento realizado se emplean las mismas clases (ver Figura 3.23) y la misma configuración de los parámetros del descriptor DAISY. El operador prom es el utilizado para este experimento. La siguiente Tabla 3.22 muestra la matriz de confusión generada.

Tabla 3.22 Matriz de confusión del décimo experimento usando el operador prom.

	MAE																			
Clases	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
1	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	1
2	0	10	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	18	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	17	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	18	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	1	0	13	0	0	0	0	0	0	0	0	0	2	0	0
9	0	0	0	0	0	0	0	2	18	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	8	0	0	5	0	0	7	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	6	0	0	14	0	0	0	0	0	0	0	0	0
12	0	1	0	0	0	0	0	5	0	0	0	13	0	0	0	0	0	0	0	0
13	0	1	1	0	0	0	0	7	0	0	0	0	11	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	18	0	0	0	2	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0
16	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	14	0	0	0	0
17	0	1	0	0	0	0	0	5	0	0	0	1	0	0	0	0	13	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17

Como última parte de este experimento, se presentan los resultados del rendimiento obtenido, los cuales son mostrados en la Tabla 3.23.

Tabla 3.23 Resultados del décimo experimento usando el operador prom.

Clases	exactitud	precisión	RVP	RVN	TE
1	0.9849	1	0.7	1	0.015075
2	0.9673	0.7692	0.5	0.992063	0.032663
3	0.9925	0.9474	0.9	0.997354	0.007538
4	0.9724	0.68	0.85	0.978836	0.027638
5	1	1	1	1	0
6	0.9849	0.9375	0.75	0.997354	0.015075
7	0.9824	0.7826	0.9	0.986772	0.017588
8	0.8819	0.2453	0.65	0.89418	0.11809
9	0.995	1	0.9	1	0.005025
10	0.9573	0.6364	0.35	0.989418	0.042714
11	0.9322	0.4	0.7	0.944444	0.067839
12	0.9523	0.5	0.684211	0.965699	0.047739
13	0.9749	0.9167	0.55	0.997354	0.025126
14	0.995	1	0.9	1	0.005025
15	1	1	1	1	0
16	0.9497	0.895	0.8	1	0.050251
17	0.59523	0.67	0.5	1	0.047739
18	0.9749	0.6667	1	0.973545	0.025126
19	0.9874	0.8	1	0.986772	0.012563
20	0.9849	0.85	0.85	0.992063	0.015075

En términos generales, los experimentos realizados en esta sección han demostrado que el descriptor DAISY ofrece un desempeño excelente al interactuar con las MAE, cuyos operadores utilizados para el aprendizaje y clasificación fueron el prom y pmed. Por otro lado, se pudo determinar que las configuraciones en los parámetros del descriptor DASIY ($R_0 = 8$, $R_1 = 16$, $Q = 2$, $H = 6$, $T = 6$ y $R_0 = 7.5$, $R_1 = 15$, $Q = 2$, $H = 8$, $T = 8$) fueron los que brindaron resultados superiores a otras configuraciones.

Capítulo 4 Arquitectura hardware del descriptor DAISY

En el presente capítulo describe el diseño de la arquitectura hardware del descriptor DAISY, realizado a partir del modelado conceptual descrito en el capítulo anterior. Inicialmente, se presenta el modelado hardware de las fases del algoritmo, exponiendo su estructura y los resultados parciales obtenidos por cada una de ellas. Al final, y con ayuda del modelado conceptual de las MAE, se presentan una serie de experimentos que tienen por finalidad medir el desempeño de la arquitectura hardware del descriptor DAISY obtenida.

4.1 Diseño de la arquitectura hardware del descriptor DAISY

La implementación del descriptor DAISY sobre lógica reconfigurable inicia con la definición de procesos generales en donde cada uno de ellos representa a una fase del descriptor DAISY. Primero, recordemos que se deben establecer los parámetros de entrada y las condiciones iniciales necesarias para la implementación, donde I es la imagen que tiene una dimensión estándar de 259×259 ($ancho\_i \times alto\_i$, respectivamente) que recibe DAISY y las condiciones iniciales son: $Q = 2$, $H = 6$, $T = 6$, $R_0 = 8$ y $R_1 = 16$.

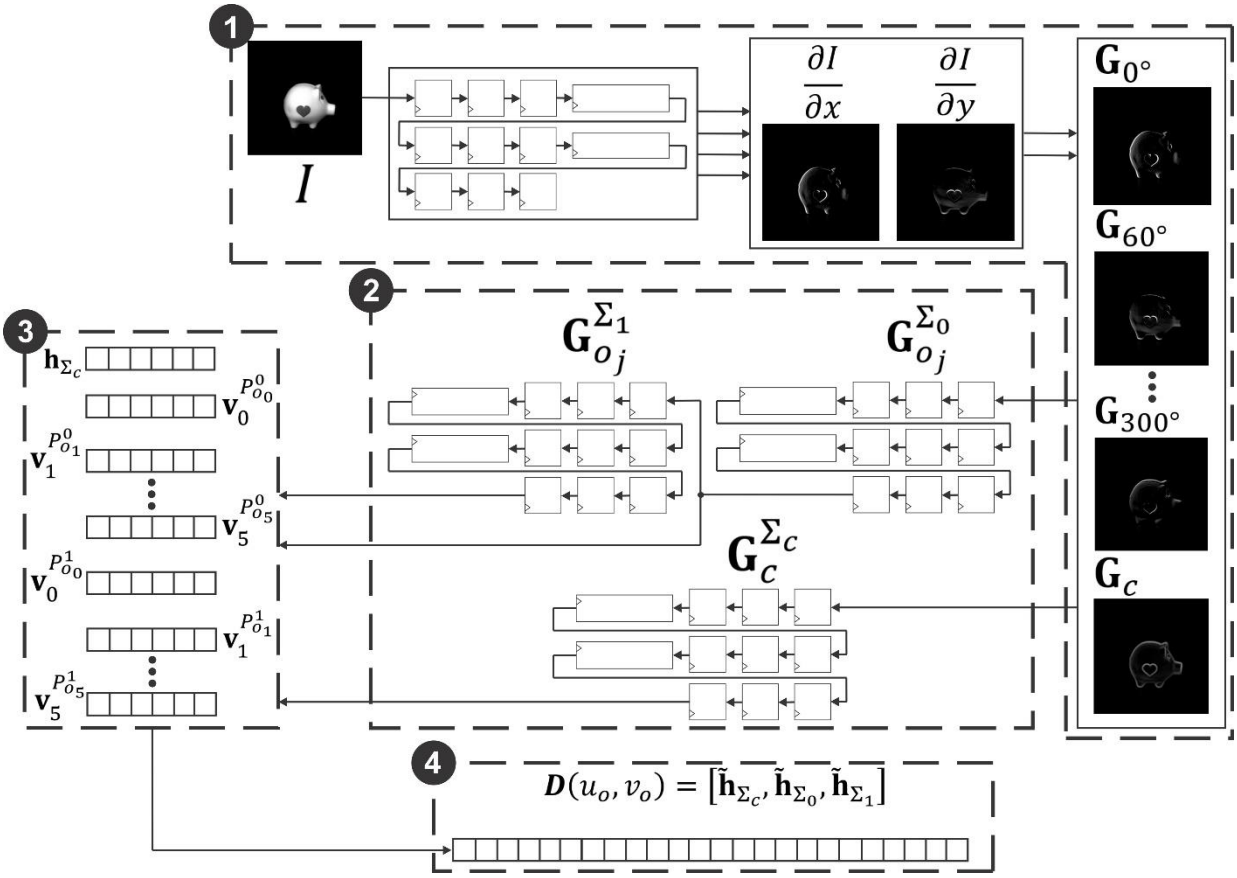


Figura 4.1 Arquitectura general del descriptor DAISY.

Ahora, en la Figura 4.1 se observa la arquitectura hardware general del descriptor DAISY, en donde cada una de las fases corresponde a un proceso general:

1. Calcular un mapa de orientación en cada una de las orientaciones definidas.
2. Aplicar filtros gaussianos con diferentes valores de desviación estándar a los mapas de orientaciones para obtener los mapas de orientación convolucionados.
3. Generación histogramas a partir de los mapas de orientación convolucionados.
4. Formar el descriptor mediante un proceso de concatenación de los histogramas y aplicarle un proceso de normalización.

En las siguientes subsecciones se muestran y describen las arquitecturas hardware que se diseñaron para cumplir con cada uno de los procesos definidos.

4.1.1 Generación de los mapas de orientación

En este primer proceso general se detalla la arquitectura hardware diseñada para la generación de los T mapas de orientación, G_o, y el mapa central, G_c. Para ello, se han definido un conjunto de subprocesos cuyas arquitecturas hardware están enfocadas a cumplir con la tarea de este proceso.

Subproceso 1. Este primer subproceso muestra el diseño de la arquitectura hardware para obtener las derivadas de I: $\frac{\partial I}{\partial x}$ y $\frac{\partial I}{\partial y}$. Con este fin, se utiliza una arquitectura de ventana deslizante basada en buffers de línea (VDBBL), la cual se muestra en la Figura 4.2.

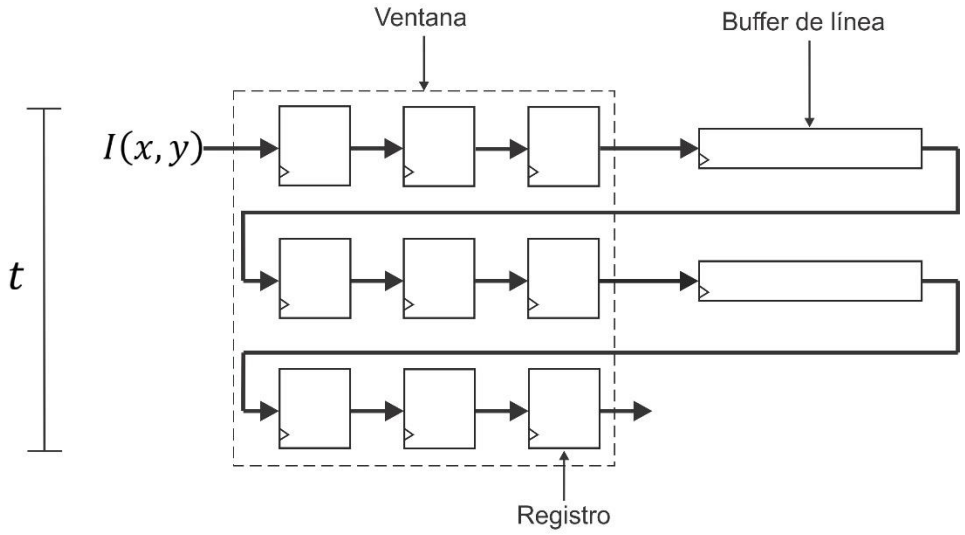


Figura 4.2 Ventana deslizante basada en buffers de línea.

Una arquitectura VDBBL está formada por un arreglo de $t \times t$ registros, donde t define al ancho y largo de la ventana, y existen $t - 1$ buffers de línea, donde cada buffer de línea está formado por $ancho\_i - t$ biestables, donde $ancho\_i$ es el ancho de la imagen I. El procesamiento de los píxeles inicia cuando el primer píxel se encuentre en la salida del último registro (a esto se le denomina saturación de la VDBBL) y finaliza cuando el último píxel se encuentre en la salida del primer registro, para el cual se acceden a los píxeles en las salidas de los registros. Cabe mencionar que no todos los píxeles serán procesados, sólo aquellos en donde la ventana se sitúe dentro de las dimensiones de I (ver Figura 4.3).

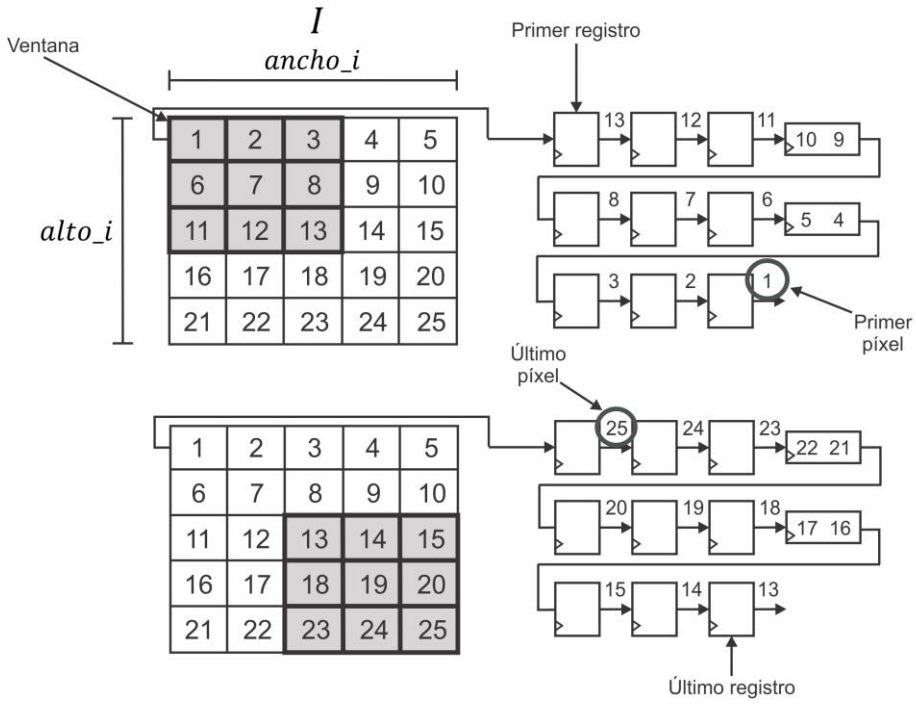


Figura 4.3 Funcionamiento de la VDBBL.

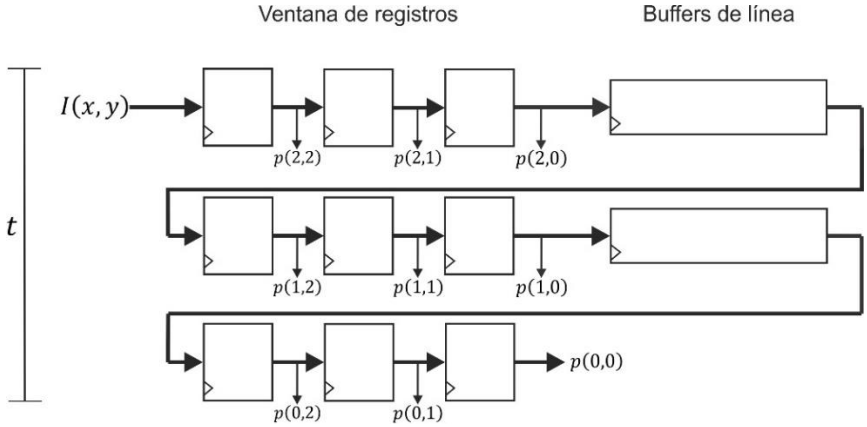


Figura 4.4 VDBBL para el obtener las derivadas de I.

En la Figura 4.4 se muestra la arquitectura VDBBL que permitirá el cálculo de la $\frac{\partial I}{\partial x}$ y $\frac{\partial I}{\partial y}$, la cual está formada por una ventana de 3×3 registros de 8 bits y $t - 1 = 3 - 1 = 2$ buffers de línea, donde cada uno almacena $ancho\_i - t = 259 - 3 = 256$ píxeles. Por otro lado, el total de píxeles necesarios para que el VDBBL se sature, se calcula con $[ancho\_i * (t - 1)] + t$, resultando en 521 píxeles. El cálculo de las derivadas de la imagen I se hace utilizando las expresiones 3.1. Considerando la arquitectura VDBBL de la Figura 4.4, esto consiste en dos simples restas, $p(1,0) - p(1,2)$ para obtener $\frac{\partial I}{\partial x}$ y $p(0,1) - p(2,1)$ para obtener $\frac{\partial I}{\partial y}$ (ver Figura 4.5), en cada desplazamiento de la ventana a través de I hasta terminar de recorrerla. La Figura 4.6 presenta los resultados obtenidos por este proceso. Cabe mencionar que las dimensiones de las derivadas son de $ancho\_i - (t - 1)$ y $alto\_i - (t - 1)$, resultando en 257×257 píxeles, siendo estas mismas dimensiones para los G_{o_j} y para el G_c.

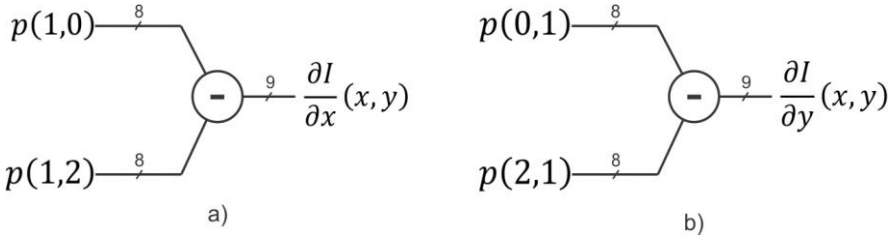


Figura 4.5 Arquitectura hardware para obtener las derivadas: a) x y b) y.

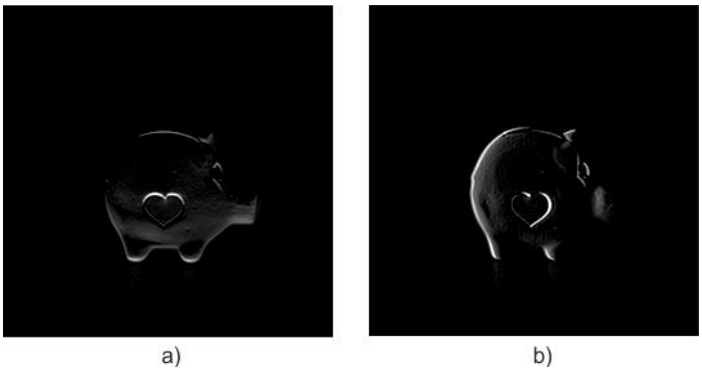


Figura 4.6 Derivadas de I: a) x y b) y.

Subproceso 2. En este subproceso los G_{o_j} son calculados haciendo uso de las orientaciones en radianes o_j, presentadas en la Tabla 3.1 (ver subsección 3.1.1). Para esta aplicación se ha utilizado la representación en formato de punto fijo con signo para aproximar los valores resultantes de $cos(o_j)$ y $sen(o_j)$ de la expresión 3.1, recordando que $j = 0, 1, \dots, T-1$. Esto se logra tomando 1 bit para representar la parte entera y 6 bits para la parte fraccionaria, más 1 bit extra para el signo situado en el bit más significativo ($Q1.1.6$). La Tabla 4.1 muestra las correspondientes aproximaciones a cada orientación.

Tabla 4.1 Aproximaciones de $\boldsymbol{cos}$ y $\boldsymbol{sen}$ en formato de punto fijo por cada $\boldsymbol{o_j}$.

Representación en punto fijo		
$\boldsymbol{o_j}$	$cos(o_j)$	$sen(o_j)$
0	01.000000	00.000000
$\pi/3$	00.100000	00.111000
$2\pi/3$	10.100000	00.111000
π	10.000000	00.000000
$4\pi/3$	10.100000	10.111000
$5\pi/3$	00.100000	10.111000

Finalmente, para obtener los G_{o_j} se ha diseñado la arquitectura hardware mostrada por la Figura 4.7. Para la cual se emplearon 12 multiplicadores de 9×8-bit, 6 sumadores de 17 bits y 6 comparadores de 17 bits. Este proceso se realiza $257 \times 257 = 66049$ veces para T mapas de orientación de forma paralela.

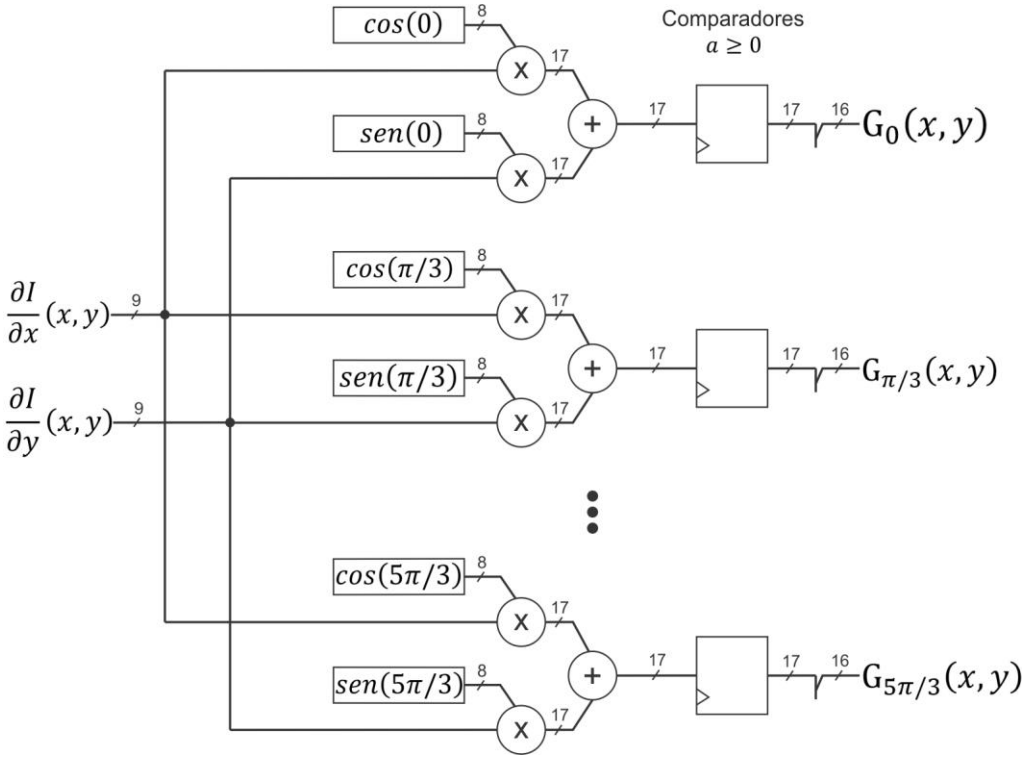


Figura 4.7 Arquitectura hardware para la obtención de los mapas de orientación.

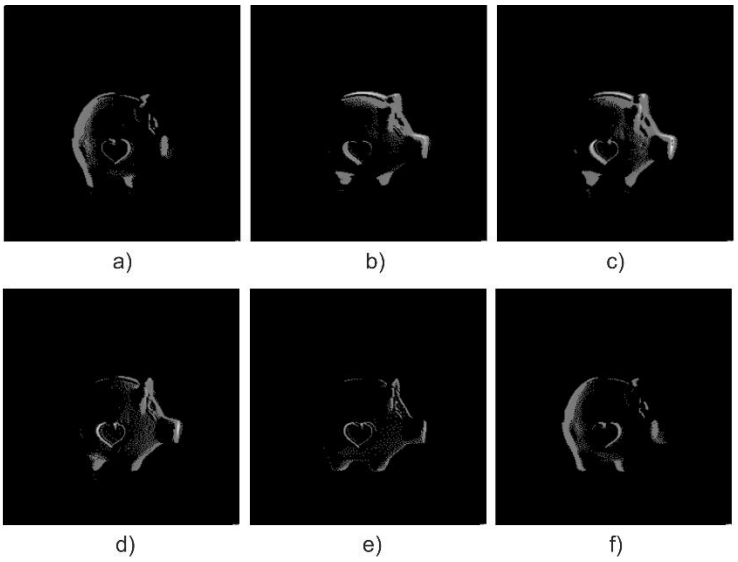


Figura 4.8 Mapas de orientación: a) 0°, b) 60°, c) 120°, d) 180°, e) 240° y f) 300°.

Los resultados obtenidos por este subproceso se presentan en la Figura 4.8. También cabe mencionar que cada píxel de los G_{o_j} está representado en formato de punto fijo sin signo $Q10.6$.

Subproceso 3. En este subproceso se calcula el G_c con la arquitectura hardware que muestra la Figura 4.9, la cual se genera utilizando la expresión 3.2 (ver subsección 3.1.1). Esta arquitectura realiza la suma de los cuadrados cada píxel de $\frac{\partial I}{\partial x}$ y $\frac{\partial I}{\partial y}$ para posteriormente obtener su raíz cuadrada. En la estructura de esta arquitectura se utilizaron 2 multiplicadores de 9×9-bit, 1 sumador de 18 bits y un IPCore para generar la raíz cuadrada en formato de punto fijo sin signo $Q10.6$. Finalmente, el G_c calculado, resultado de este proceso, es presentado por la Figura 4.10.

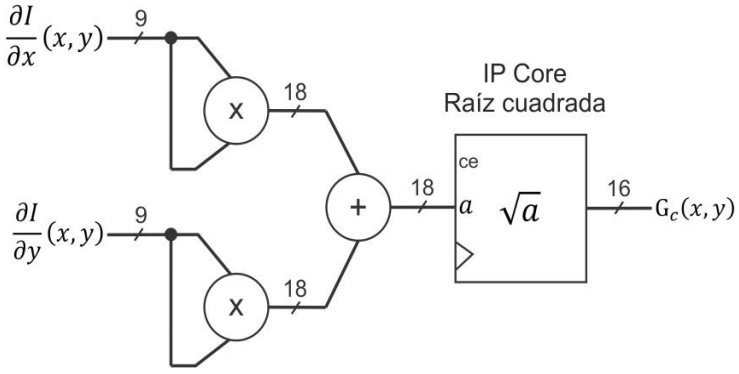


Figura 4.9 Arquitectura hardware para la obtención del mapa central.

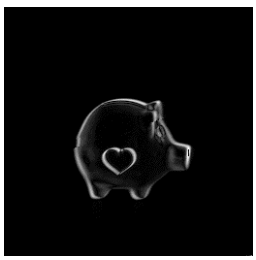


Figura 4.10 Mapa central.

4.1.2 Generación de los mapas de orientación convolucionados

En este proceso se describe las arquitecturas hardware diseñadas para obtener los $Q \times T$ mapas de orientación convolucionados $G_{o_j}^{\Sigma_k}$ $(k = 0, \dots, Q-1)$ y el mapa central convolucionado $G_c^{\Sigma_c}$. Para facilitar su explicación, este proceso ha sido dividido en subprocesos que desarrollan tareas simples y específicas.

Subproceso 1. En este subproceso, se calculan los coeficientes del G_{Σ_0} y del G_{Σ_1} con los valores de Σ_0, t_0, Σ_1 y t_1. Para ello, con la finalidad de disminuir recursos del FPGA, en este subproceso se definen tamaños para el G_{Σ_0} y el G_{Σ_1}más pequeños a los usados en el modelado conceptual. Para ello, se realizaron diversos experimentos de la implementación secuencial del descriptor DAISY y las MAE para observar el rendimiento del descriptor ante tamaños de filtros pequeños, de los cuales se determinó que los tamaños: $t_0 = 5$ y $t_1 = 7$, para el primer y segundo anillo, respectivamente, no afectan el rendimiento del descriptor DAISY que fue presentado en la sección 3.3. Ahora, se calculan los coeficientes mediante la siguiente expresión:

$$sum = G_{\Sigma_k}(x,y) = \frac{1}{2\pi(\Sigma_k)^2} e^{-\frac{x^2+y^2}{2(\Sigma_k)^2}}$$
$$\tilde{G}_{\Sigma_k}(x,y) = \frac{G_{\Sigma_k}(x,y)}{sum} \quad (4.1)$$

Donde por cuestiones de simplicidad, se regresa a la nomenclatura anterior para el G_{Σ}, es decir $\tilde{G}_{\Sigma_k}$ vuelve a denominarse como G_{Σ_k}. Siendo los coeficientes del G_{Σ_0} mostrados a continuación.

$$G_{\Sigma_0} = \begin{bmatrix} 0.035204 & 0.038664 & 0.039891 & 0.038664 & 0.035204 \\ 0.038664 & 0.042464 & 0.043812 & 0.042464 & 0.038664 \\ 0.039891 & 0.043812 & 0.045202 & 0.043812 & 0.039891 \\ 0.038664 & 0.042464 & 0.043812 & 0.042464 & 0.038664 \\ 0.035204 & 0.038664 & 0.039891 & 0.038664 & 0.035204 \end{bmatrix}$$

Posteriormente, se realiza la representación en formato de punto fijo sin signo $Q1.8$ de los coeficientes del G_{Σ_0}, la cual se puede consultar en la Tabla 4.2.

Tabla 4.2 Representación en punto fijo de los coeficientes del filtro gaussiano para el primer anillo.

0.00001001	0.00001010	0.00001010	0.00001010	0.00001001
0.00001010	0.00001011	0.00001011	0.00001011	0.00001010
0.00001010	0.00001011	0.00001100	0.00001011	0.00001010
0.00001010	0.00001011	0.00001011	0.00001011	0.00001010
0.00001001	0.00001010	0.00001010	0.00001010	0.00001001

Seguidamente, los coeficientes del G_{Σ_1} se calculan usando la expresión 4.1, los cuales se muestran a continuación.

$$G_{\Sigma_1} = \begin{bmatrix} 0.0183655 & 0.0193474 & 0.0199615 & 0.0201705 & 0.0199615 & 0.0193474 & 0.0183655 \\ 0.0193474 & 0.0203818 & 0.0210287 & 0.0212489 & 0.0210287 & 0.0203818 & 0.0193474 \\ 0.0199615 & 0.0210287 & 0.0216963 & 0.0219235 & 0.0216963 & 0.0210287 & 0.0199615 \\ 0.0201705 & 0.0212489 & 0.0219235 & 0.022153 & 0.0219235 & 0.0212489 & 0.0201705 \\ 0.0199615 & 0.0210287 & 0.0216963 & 0.0219235 & 0.0216963 & 0.0210287 & 0.0199615 \\ 0.0193474 & 0.0203818 & 0.0210287 & 0.0212489 & 0.0210287 & 0.0203818 & 0.0193474 \\ 0.0183655 & 0.0193474 & 0.0199615 & 0.0201705 & 0.0199615 & 0.0193474 & 0.0183655 \end{bmatrix}$$

La Tabla 4.3 presenta las aproximaciones de los coeficientes del G_{Σ_1}, en formato de punto fijo sin signo $Q1.8$.

Tabla 4.3 Representación en punto fijo de los coeficientes del filtro gaussiano para el segundo anillo.

0.00000101	0.00000101	0.00000101	0.00000101	0.00000101	0.00000101	0.00000101
0.00000101	0.00000101	0.00000101	0.00000101	0.00000101	0.00000101	0.00000101
0.00000101	0.00000101	0.00000110	0.00000110	0.00000110	0.00000101	0.00000101
0.00000101	0.00000101	0.00000110	0.00000110	0.00000110	0.00000101	0.00000101
0.00000101	0.00000101	0.00000110	0.00000110	0.00000110	0.00000101	0.00000101
0.00000101	0.00000101	0.00000101	0.00000101	0.00000101	0.00000101	0.00000101
0.00000101	0.00000101	0.00000101	0.00000101	0.00000101	0.00000101	0.00000101

Subproceso 2. En el subproceso 3 se realiza el diseño de la VDBBL para acceder a los píxeles de los mapas de orientación. Para ello, es necesario recordar las dimensiones de los mapas de orientación es de 257×257 píxeles. Entonces, se construye una ventana usando el tamaño del filtro antes determinado ($t_0 = 5$), quedando de 5×5 registros de 16 bits y 4 buffers de línea, donde cada uno almacena $ancho\_i - t_0 = 257 - 5 = 252$ píxeles por línea (ver Figura 4.11). La saturación de la VDBBL se logra con $[ancho\_i * (t_0 - 1)] + t_0 = [257 * (5 - 1)] + 5 = 1033$ píxeles.

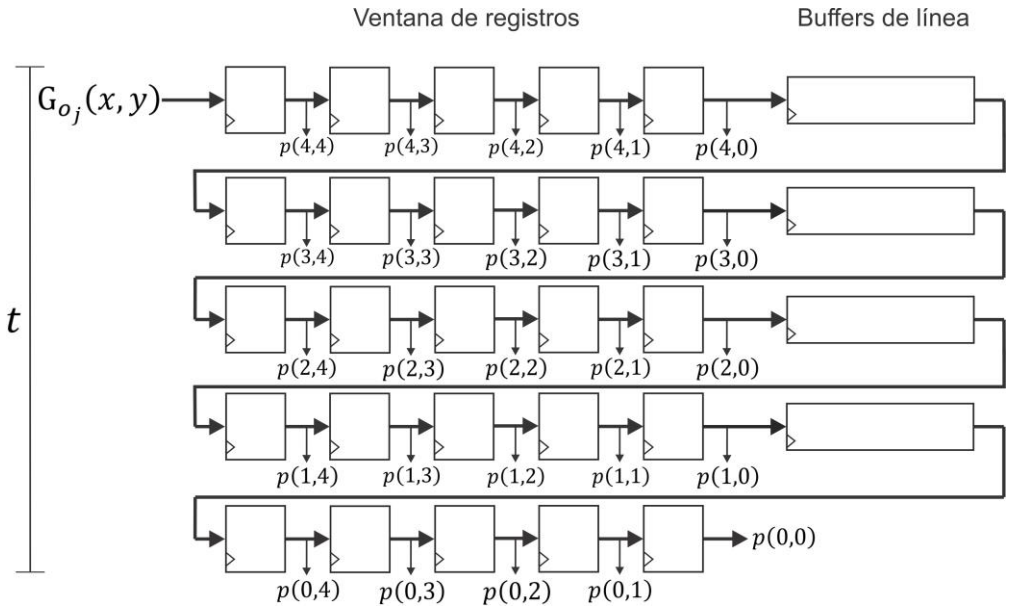


Figura 4.11 VDBBL para acceder a los píxeles de los mapas de orientación.

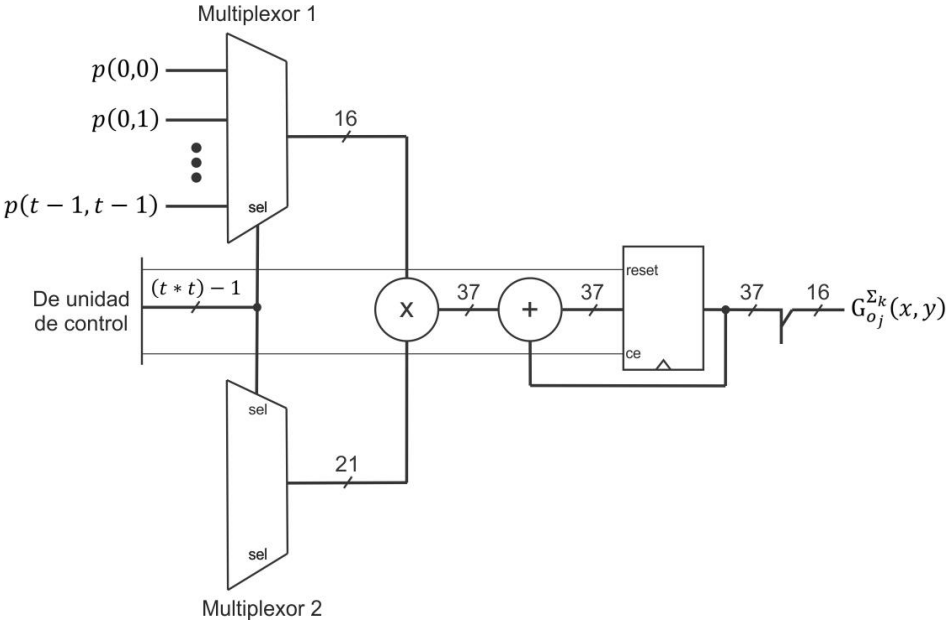


Figura 4.12 Arquitectura hardware para el cálculo de sumatorias de productos.

Subproceso 3. Para la convolución se ha diseñado una arquitectura hardware (véase la Figura 4.12) formada por 2 multiplexores, 1 multiplicador de 21×16-bit, 1 sumador de 37 bits y 1 registro de 37 bits. Esta arquitectura recibe los píxeles provenientes de la VDBBL en el multiplexor 1, mientras que los coeficientes del filtro se encuentran en el multiplexor 2, donde el selector de salida va de 0 a $(t * t) - 1$. Despues, se calcula la sumatoria de los productos entre los píxeles de los mapas de orientación y los coeficientes de los filtros $t * t$ veces por cada píxel para obtener el píxel perteneciente a los $G_{o_j}^{\Sigma_k}$, el cual se trunca tomando 10 bits para la parte entera y 6 bits para la parte fraccionaría (formato de punto fijo sin signo $Q10.6$). La Figura 4.13 muestra los $G_{o_j}^{\Sigma_0}$ obtenidos a partir de las arquitecturas antes descritas, cuyas dimensiones son de $ancho\_i - (t_0 - 1)$ y $alto\_i - (t_0 - 1)$, resultando en 253×253 píxeles, mismas que servirán para diseñar el VDBBL para el segundo anillo.

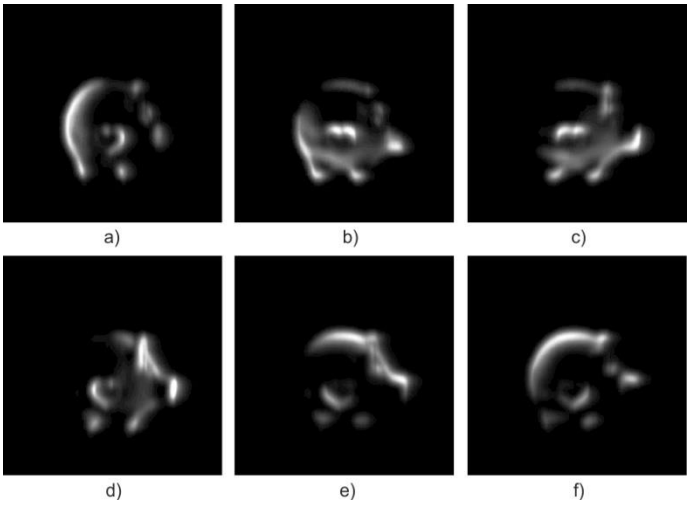


Figura 4.13 Mapas de orientación convolucionados del primer anillo: a) 0°, b) 60°, c) 120°, d) 180°, e) 240° y f) 300°.

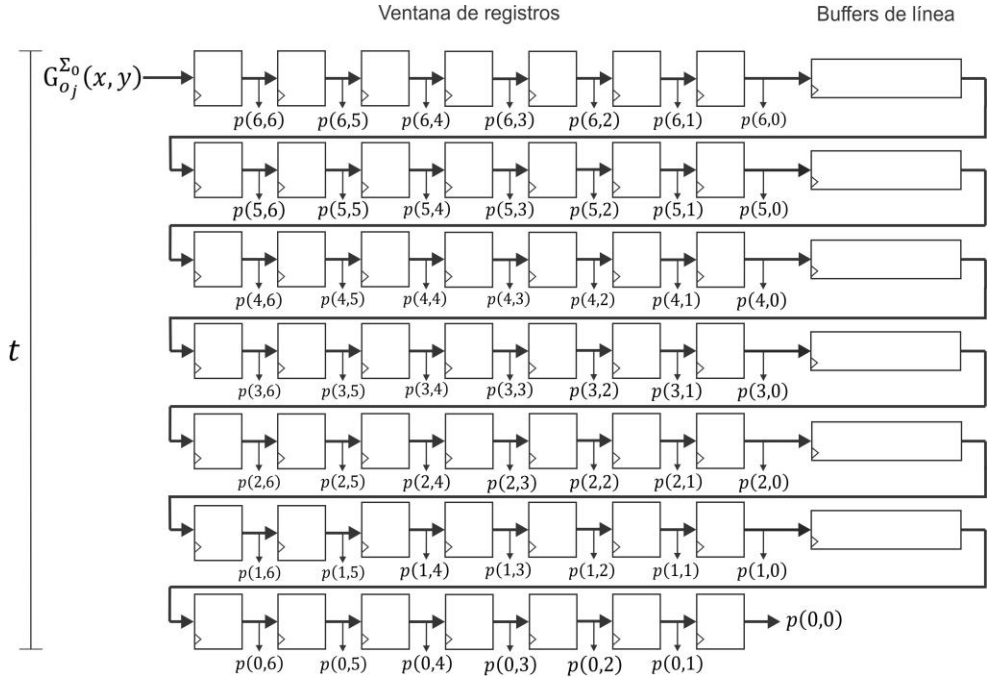


Figura 4.14 VDBBL para acceder a los píxeles de los mapas de orientación convolucionados del primer anillo.

Subproceso 4. En este subproceso se muestra el diseño de la VDBBL para acceder a los píxeles de los mapas de orientación del segundo anillo (ver Figura 4.14). El cual contiene una ventana de 7×7 registros de 16 bits y 6 buffers de línea, donde cada uno almacena $ancho\_i - t_1 = 253 - 7 = 246$ píxeles por línea. La VDBBL se satura con $[ancho\_i * (t_1 - 1)] + t_1 = [253 * (7 - 1)] + 7 = 1525$ píxeles. Finalmente, se obtienen los $G_{o_j}^{\Sigma_1}$ empleando la VDBBL antes presentada y la arquitectura hardware para la convolución mostrada en la Figura 4.12, pero en este caso el selector que controla a los multiplexores va de 0 a $(t_1 * t_1) - 1 = (7 * 7) - 1 = 48$. En la Figura 4.15 se observan los $G_{o_j}^{\Sigma_1}$ obtenidos, con unas dimensiones de $ancho\_i - (t_1 - 1)$ y $alto\_i - (t_1 - 1)$, resultando en 247×247 píxeles.

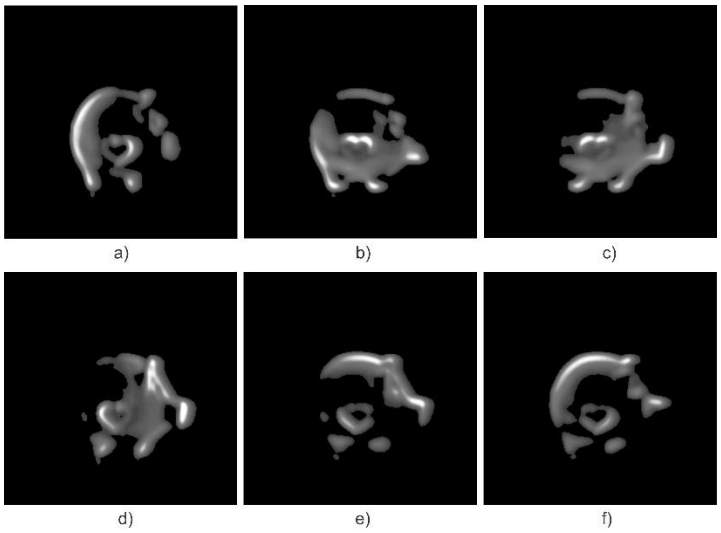


Figura 4.15 Mapas de orientación convolucionados del segundo anillo: a) 0°, b) 60°, c) 120°, d) 180°, e) 240° y f) 300°.

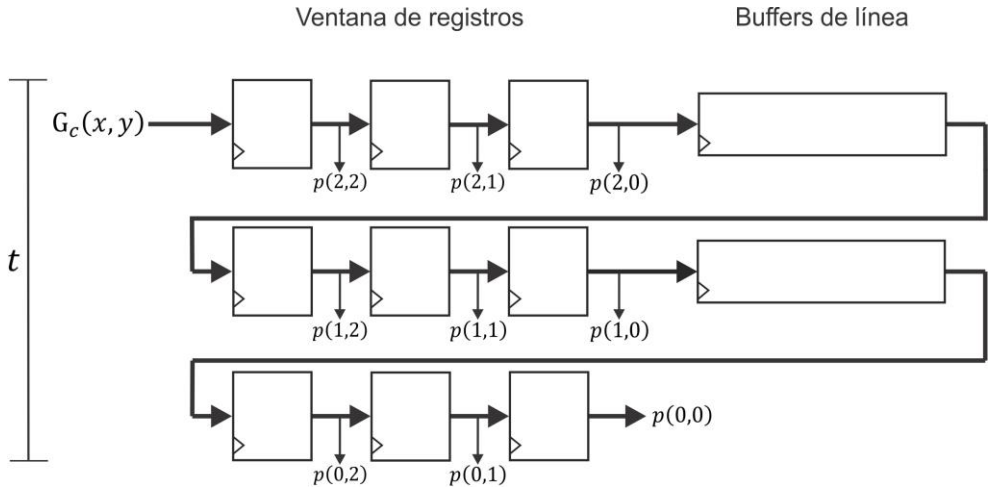


Figura 4.16 VDBBL para acceder a los píxeles del mapa central.

Subproceso 5. Con la finalidad de disminuir recursos del FPGA, en este subproceso se determina un tamaño del G_{Σ_c} menor al utilizado en el modelado conceptual que no afecta el rendimiento del descriptor DAISY. Después de haberse realizado diversos experimentos en la implementación secuencial de este descriptor, se determinó que el tamaño $t_c = 3$ es el adecuado para dicho filtro. Seguidamente, con los valores de Σ_c y t_c se pueden calcular los coeficientes del G_{Σ_c} con ayuda de la expresión 4.1, los cuales se muestran a continuación.

$$G_{\Sigma_c} = \begin{bmatrix} 0.095165 & 0.118158 & 0.095165 \\ 0.118158 & 0.146707 & 0.118158 \\ 0.095165 & 0.118158 & 0.095165 \end{bmatrix} \quad (4.2)$$

Siendo las aproximaciones de estos coeficientes representados en punto fijo sin signo $Q1.7$, las que se muestran en la Tabla 4.4.

Tabla 4.4 Aproximaciones de los coeficientes del filtro gaussiano del mapa central.

0.0010011	0.0001111	0.0010011
0.0001111	0.0010011	0.0001111
0.0010011	0.0001111	0.0010011

Subproceso 6. La Figura 4.16 muestra la estructura de la VDBBL utilizada para acceder a los píxeles del G_c y posteriormente realizar la convolución con el filtro gaussiano de la expresión 4.2. En la definición de la estructura de la VDBBL se retoman las dimensiones del G_c (257×257 píxeles) que servirán para calcular su saturación con: $[ancho\_i * (t_c - 1)] + t_c = [257 * (3 - 1)] + 3 = 517$ píxeles. Esta arquitectura emplea una ventana de 3×3 registros de 16 bits y 2 buffers de línea, en donde cada uno almacena $ancho\_i - t_c = 257 - 3 = 254$ píxeles por línea.

Subproceso 7. En este último subproceso, se convoluciona el G_c con el filtro gaussiano usando la VDBBL antes mostrada y la arquitectura hardware para la convolución. Para este caso solo se modifica el conteo que realiza el selector que controla a los multiplexores, el cual va de 0 a $(t_c * t_c) - 1 = (3 * 3) - 1 = 8$.

Figura 4.17 Mapa central convolucionado.

Las dimensiones del $G_c^{\Sigma_c}$ obtenido son de $ancho\_i - (t_c - 1)$ y $alto\_i - (t_c - 1)$, resultando en 255×255 píxeles. El $G_c^{\Sigma_c}$ obtenido es presentado por la Figura 4.17.

4.1.3 Generación de los vectores

El objetivo de este proceso es la generación de los vectores $\mathbf{h}_{\Sigma_c}$, $\mathbf{h}_{\Sigma_0}$ y $\mathbf{h}_{\Sigma_1}$ utilizando lógica reconfigurable. Para ello, se presentan y describen como se calculan las direcciones de localidades, las cuales servirán para acceder a la memoria externa donde se encuentran almacenados el $G_c^{\Sigma_c}$, los $G_{o_j}^{\Sigma_0}$ y $G_{o_j}^{\Sigma_1}$, para obtener los puntos de interés que conformarán a los vectores. Por otro lado, dado que se requiere el almacenamiento de los vectores $\mathbf{h}_{\Sigma_c}$, $\mathbf{h}_{\Sigma_0}$ y $\mathbf{h}_{\Sigma_1}$ para su posterior procesamiento, se ha diseñado una memoria RAM que contiene 78 localidades de profundidad, donde cada localidad es de 24 bits, 78×24 bits. Para cumplir con el objetivo, se han definido los siguientes subprocesos para cumplir con dicho objetivo.

Subproceso 1. En este subproceso se calculan las coordenadas rectangulares de los *centros* $P_{o_j}^0(8, o_j)$ del primer anillo ubicados sobre los $G_{o_j}^{\Sigma_0}$, de los cuales se obtendrán los puntos de interés que formarán a los vectores. Primeramente, se obtienen las cmi_x y cmi_y (coordenadas medias de la imagen) que para este caso, se calculan utilizando las dimensiones de los $G_{o_j}^{\Sigma_0}$ (253×253 píxeles), empleando $cmi\_x = (ancho\_i - 1)/2$ y $cmi\_y = (alto\_i - 1)/2$, quedando en $cmi\_x = 253/2 = 126.5 \approx 127$ y $cmi\_y = 253/2 = 126.5 \approx 127$, respectivamente. Después, usando las coordenadas rectangulares de la Tabla 3.2 presentada en la subsección 3.1.3, se calculan las coordenadas rectangulares de los *T centros*, las cuales se pueden consultar en la Tabla 4.5.

Tabla 4.5 Coordenadas rectangulares de los *centros* en el primer anillo.

centros	**Coordenadas rectangulares**	
	x	y
$P_0^0(8, 0)$	135	127
$P_{\pi/3}^0(8, \pi/3)$	131	134
$P_{2\pi/3}^0(8, 2\pi/3)$	123	134
$P_{\pi}^0(8, \pi)$	119	127
$P_{4\pi/3}^0(8, 4\pi/3)$	123	120
$P_{5\pi/3}^0(8, 5\pi/3)$	131	120

Subproceso 2. Ahora, se calculan las coordenadas rectangulares de los puntos de interés que formarán al $\mathbf{h}_{\Sigma_0}$ en el vecindario de cada uno de los *centros* antes calculados. Para lograr esto se usan las siguientes expresiones: $\hat{x} = x + [\Sigma_0 \cos(\hat{o}_q)]$ y $\hat{y} = y + [\Sigma_0 \text{sen}(\hat{o}_q)]$, recordando que $\Sigma_0 = 4$ y que $\hat{o}_q$ $(q = 0, 1, \ldots, H-1)$ es un vector que contiene las orientaciones en radianes de los H puntos de interés que contiene cada vector $\mathbf{v}_j^{P_{o_j}^0}$. Después de haber calculado las coordenadas rectangulares de los puntos de interés, se muestra cómo se obtienen las direcciones de localidades en memoria de dichas coordenadas. Se parte de que los $G_{o_j}^{\Sigma_0}$ están almacenados en la memoria externa y sabiendo que la cantidad de memoria que ocupa cada mapa está definida por $ancho\_i * alto\_i = 253 * 253 = 64009$ píxeles, en donde cada píxel corresponde una localidad de memoria, cuyas direcciones se observan en la Tabla 4.6.

Tabla 4.6 Direcciones y uso de memoria del primer anillo.

Intervalo de direcciones	Uso de la memoria
0x0A1220 - 0x0B0C29	$G_0^{\Sigma_0}$
0x0B2390 - 0x0C1D99	$G_{\pi/3}^{\Sigma_0}$
0x0C3500 - 0x0D2F09	$G_{2\pi/3}^{\Sigma_0}$
0x0D4670 - 0x0E4079	$G_{\pi}^{\Sigma_0}$
0x0E57E0 - 0x0F51E9	$G_{4\pi/3}^{\Sigma_0}$
0x0F6950 - 0x106359	$G_{5\pi/3}^{\Sigma_0}$

Con este fin, se aplica la siguiente expresión:

$$pos = pos + 1 \quad hasta\ que\ (r = \hat{x}\ \text{y}\ s = \hat{y}) \tag{4.3}$$

Donde $r = 0, 1, 2, \ldots, ancho\_i - 1$ y $s = 0, 1, 2, \ldots, alto\_i - 1$. En el Algoritmo 4.1 se observa el pseudocódigo en donde la expresión anterior determina la posición de cada punto de interés en base a su correspondiente coordenada rectangular.

Algoritmo 4.1 Pseudocódigo para obtener las direcciones de localidades de los puntos de interés.

```
1     Función direcciones (x̂, ŷ, ancho_i, alto_i, dir_ini)
2         Para s ← 0 Hasta s < alto_i Con Paso 1 Hasta
3             Para r ← 0 Hasta r < ancho_i Con Paso 1 Hasta
4                 pos ← pos + 1;
5                 Si (r == x̂ && s == ŷ) Entonces
6                     pos ← pos + dir_ini;
7                     break;
8                 Fin Si
9             Fin Para
10        Fin Para
11    Fin Función
```

El pseudocódigo del Algoritmo 4.1 muestra cómo se obtiene la posición del punto de interés correspondiente a su coordenada rectangular $(\hat{x}, \hat{y})$, la cual se obtiene respecto a las dimensiones de la imagen $(ancho\_i \times alto\_i)$, en donde el contador pos incrementa en una unidad en cada

iteración de n y m hasta que coincidan con las coordenadas rectangulares $(\hat{x}, \hat{y})$. Finalmente, la dirección se obtiene adicionando la dirección inicial de cada imagen dir_ini (de acuerdo con la Tabla 4.6) al contador pos. En las tablas siguientes se pueden visualizar las coordenadas rectangulares de los puntos de interés, así como de las direcciones de las localidades que los contienen.

La Tabla 4.7 presenta las coordenadas rectangulares de los puntos de interés que se encuentran en la vecindad del *centro* $P_0^0(8, 0)$, así como también las direcciones correspondientes.

Tabla 4.7 Direcciones de memoria de los puntos de interés que forman al vector $\mathbf{v}_0^{P_0^0}$.

Puntos de interés	$\hat{x}$	$\hat{y}$	Direcciones
$b_0^{P_0^0}$	139	127	0x0B0BB3
$b_1^{P_0^0}$	137	130	0x0B0BB6
$b_2^{P_0^0}$	133	130	0x0B0BB2
$b_3^{P_0^0}$	131	127	0x0B0BB0
$b_4^{P_0^0}$	133	124	0x0B0BB2
$b_5^{P_0^0}$	137	124	0x0B0BB6

De la misma forma, la Tabla 4.8 contiene las coordenadas rectangulares de los puntos de interés que se encuentran en la vecindad del *centro* $P_{\pi/3}^0(8, 0)$, así como también las direcciones correspondientes.

Tabla 4.8 Direcciones de memoria de los puntos de interés que forman al vector $\mathbf{v}_1^{P_{\pi/3}^0}$.

Puntos de interés	$\hat{x}$	$\hat{y}$	Direcciones
$b_0^{P_{\pi/3}^0}$	135	134	0x0C1D24
$b_1^{P_{\pi/3}^0}$	133	137	0x0C1D22
$b_2^{P_{\pi/3}^0}$	129	137	0x0C1D1E
$b_3^{P_{\pi/3}^0}$	127	134	0x0C1D1C
$b_4^{P_{\pi/3}^0}$	129	131	0x0C1D1E
$b_5^{P_{\pi/3}^0}$	133	131	0x0C1D22

Así mismo, las coordenadas rectangulares de los puntos de interés obtenidos en torno al *centro* $P_{2\pi/3}^0(8, 2\pi/3)$, se observan en la Tabla 4.9. También se incluyen las direcciones correspondientes.

Tabla 4.9 Direcciones de memoria de los puntos de interés que forman al vector $\mathbf{v}_2^{P_{2\pi/3}^0}$.

Puntos de interés	$\hat{x}$	$\hat{y}$	Direcciones
$b_0^{P_{2\pi/3}^0}$	127	134	0x0D2E8C
$b_1^{P_{2\pi/3}^0}$	125	137	0x0D2E8A
$b_2^{P_{2\pi/3}^0}$	121	137	0x0D2E86
$b_3^{P_{2\pi/3}^0}$	119	134	0x0D2E84
$b_4^{P_{2\pi/3}^0}$	121	131	0x0D2E86
$b_5^{P_{2\pi/3}^0}$	125	131	0x0D2E8A

Seguidamente, en la Tabla 4.10 se visualizan las ubicaciones en coordenadas rectangulares y las direcciones de localidades, obtenidas para los puntos de interés situados alrededor del *centro* $P_\pi^0(8,\pi)$.

Tabla 4.10 Direcciones de memoria de los puntos de interés que forman al vector $\mathbf{v}_3^{P_\pi^0}$.

Puntos de interés	$\hat{x}$	$\hat{y}$	Direcciones
$b_0^{P_\pi^0}$	123	127	0x0E3FF8
$b_1^{P_\pi^0}$	121	130	0x0E3FF6
$b_2^{P_\pi^0}$	117	130	0x0E3FF2
$b_3^{P_\pi^0}$	115	127	0x0E3FF0
$b_4^{P_\pi^0}$	117	124	0x0E3FF2
$b_5^{P_\pi^0}$	121	124	0x0E3FF6

En la Tabla 4.11, se pueden consultar las coordenadas rectangulares y las direcciones de localidades de los puntos de interés para el *centro* $P_{4\pi/3}^0(8,4\pi/3)$.

Tabla 4.11 Direcciones de memoria de los puntos de interés que forman al vector $\mathbf{v}_4^{P_{4\pi/3}^0}$.

Puntos de interés	$\hat{x}$	$\hat{y}$	Direcciones
$b_0^{P_{4\pi/3}^0}$	127	120	0x0F516C
$b_1^{P_{4\pi/3}^0}$	125	123	0x0F516A
$b_2^{P_{4\pi/3}^0}$	121	123	0x0F5166
$b_3^{P_{4\pi/3}^0}$	119	120	0x0F5164
$b_4^{P_{4\pi/3}^0}$	121	117	0x0F5166
$b_5^{P_{4\pi/3}^0}$	125	117	0x0F516A

Por último, en torno al *centro* $P^0_{5\pi/3}(8, 5\pi/3)$ se obtuvieron las siguientes coordenadas rectangulares y sus respectivas direcciones de localidades correspondientes a los puntos de interés, mostradas por la Tabla 4.12.

Tabla 4.12 Direcciones de memoria de los puntos de interés que forman al vector $\mathbf{v}_5^{P^0_{5\pi/3}}$.

Puntos de interés	$\hat{x}$	$\hat{y}$	Direcciones
$b_0^{P^0_{5\pi/3}}$	135	120	0x1062E4
$b_1^{P^0_{5\pi/3}}$	133	123	0x1062E2
$b_2^{P^0_{5\pi/3}}$	129	123	0x1062DE
$b_3^{P^0_{5\pi/3}}$	127	120	0x1062DC
$b_4^{P^0_{5\pi/3}}$	129	117	0x1062DE
$b_5^{P^0_{5\pi/3}}$	133	117	0x01062E2

Por lo tanto, el $\mathbf{h}_{\Sigma_0}$ se estructura de la siguiente manera: $\mathbf{h}_{\Sigma_0} = \left[\mathbf{v}_0^{P^0_0}, \mathbf{v}_1^{P^0_{\pi/3}}, \dots, \mathbf{v}_{T-1}^{P^0_{5\pi/3}}\right]$, siendo la longitud de este de $H \times T = 6 \times 6 = 36$ elementos, lo cual impide que se presente completo. Debido a esto, y únicamente con fines demostrativos, la expresión 4.4 muestra los valores del vector $\mathbf{v}_0^{P^0_0}$.

$$\mathbf{v}_0^{P^0_0} = [2.07421875, 0.41796875, 0.05859375, 0.96484375, 1.13671875, 0] \quad (4.4)$$

Subproceso 3. En este subproceso se calculan las ubicaciones de los *centros* $P^1_{o_j}(16, o_j)$ sobre los $\mathrm{G}^{\Sigma_1}_{o_j}$. Para ello, se utilizan las dimensiones de los $\mathrm{G}^{\Sigma_1}_{o_j}$ (247×247 píxeles) para calcular las coordenadas medias de la imagen, por lo tanto $cmi\_x = 247/2 = 123.5 \approx 124$ y $cmi\_y = 247/2 = 123.5 \approx 124$, respectivamente. Después, usando las coordenadas rectangulares de la Tabla 3.2 presentada en la subsección 3.1.3, se calculan las coordenadas rectangulares de los T *centros* $P^1_{o_j}(16, o_j)$, las cuales se pueden consultar en la Tabla 4.13.

Tabla 4.13 Coordenadas rectangulares de los *centros* en el segundo anillo.

centros	**Coordenadas rectangulares**	
	x	y
$P^1_0(16, 0)$	140	124
$P^1_{\pi/3}(16, \pi/3)$	132	138
$P^1_{2\pi/3}(16, 2\pi/3)$	116	138
$P^1_{\pi}(16, \pi)$	108	124
$P^1_{4\pi/3}(16, 4\pi/3)$	116	110
$P^1_{5\pi/3}(16, 5\pi/3)$	132	110

Subproceso 4. Ahora, se calculan las coordenadas rectangulares puntos de interés que formarán al $\mathbf{h}_{\Sigma_1}$ en el vecindario de cada uno de los *centros* $P^1_{o_j}(16, o_j)$ antes calculados. Para lograr esto,

se calculan mediante $\hat{x} = x + [\Sigma_1 \cos(\hat{o}_q)]$ y $\hat{y} = y + [\Sigma_1 \text{sen}(\hat{o}_q)]$, recordando que $\Sigma_1 = 7$. Usando estas coordenadas rectangulares se obtienen las direcciones de las localidades donde se encuentran los puntos de interés en la memoria externa. Para eso, se parte de que los $G_{o_j}^{\Sigma_1}$ están almacenados en la memoria externa y sabiendo que la cantidad de memoria que ocupa cada imagen está definida por $ancho\_i * alto\_i = 247 * 247 = 61009$ píxeles, en donde cada píxel se almacena en una localidad de memoria. Estas direcciones se observan en la Tabla 4.14.

Tabla 4.14 Direcciones y uso de memoria del segundo anillo.

Intervalo de direcciones	Uso de la memoria
0x107AC0 - 0x116911	$G_0^{\Sigma_1}$
0x118C30 - 0x127A81	$G_{\pi/3}^{\Sigma_1}$
0x129DA0 - 0x138BF1	$G_{2\pi/3}^{\Sigma_1}$
0x13AF10 - 0x149D61	$G_{\pi}^{\Sigma_1}$
0x14C080 - 0x15AED1	$G_{4\pi/3}^{\Sigma_1}$
0x15D1F0 - 0x16C041	$G_{5\pi/3}^{\Sigma_1}$

Seguidamente, se reutiliza el pseudocódigo del Algoritmo 4.1 para obtener las direcciones de localidades de los puntos de interés. Estas direcciones se pueden consultar en las tablas que a continuación se presentan, en donde se incluyen las coordenadas rectangulares calculadas, así como también las direcciones de memoria.

La Tabla 4.15 presenta las coordenadas rectangulares de los puntos de interés que se encuentran en la vecindad del *centro* $P_0^1(16, 0)$, así como también los valores de las direcciones de las localidades correspondientes.

Tabla 4.15 Direcciones de memoria de los puntos de interés que forman al vector $\mathbf{v}_0^{P_0^1}$.

Puntos de interés	$\hat{x}$	$\hat{y}$	Direcciones
$b_0^{P_0^0}$	147	124	0x1168AE
$b_1^{P_0^0}$	144	130	0x1168AB
$b_2^{P_0^0}$	137	130	0x1168A4
$b_3^{P_0^0}$	133	124	0x1168A0
$b_4^{P_0^0}$	137	118	0x1168A4
$b_5^{P_0^0}$	144	118	0x1168AB

En la Tabla 4.16, se exponen las direcciones de los puntos de interés que formarán al vector $\mathbf{v}_0^{P_{\pi/3}^1}$, así como las coordenadas rectangulares correspondientes.

Tabla 4.16 Direcciones de memoria de los *centros* que forman al vector $\mathbf{v}_0^{P_{\pi/3}^1}$.

Puntos de interés	$\hat{x}$	$\hat{y}$	Direcciones
$b_0^{P_{\pi/3}^0}$	139	138	0x127A16
$b_1^{P_{\pi/3}^0}$	136	144	0x127A13
$b_2^{P_{\pi/3}^0}$	129	144	0x127A0C
$b_3^{P_{\pi/3}^0}$	125	138	0x127A08
$b_4^{P_{\pi/3}^0}$	129	132	0x127A0C
$b_5^{P_{\pi/3}^0}$	136	132	0x127A13

Seguidamente, se pueden observar las direcciones que servirán para obtener los puntos de interés que forman el vector $\mathbf{v}_0^{P_{2\pi/3}^1}$, en la Tabla 4.17.

Tabla 4.17 Direcciones de memoria de los *centros* que forman al vector $\mathbf{v}_0^{P_{2\pi/3}^1}$.

Puntos de interés	$\hat{x}$	$\hat{y}$	Direcciones
$b_0^{P_{2\pi/3}^0}$	123	138	0x138B76
$b_1^{P_{2\pi/3}^0}$	120	144	0x138B73
$b_2^{P_{2\pi/3}^0}$	116	144	0x138B6C
$b_3^{P_{2\pi/3}^0}$	109	138	0x138B68
$b_4^{P_{2\pi/3}^0}$	113	132	0x138B6C
$b_5^{P_{2\pi/3}^0}$	120	132	0x138B73

De la misma manera, son expuestas por la Tabla 4.18 las coordenadas rectangulares de los puntos de interés correspondientes al vector $\mathbf{v}_0^{P_\pi^1}$, que sirvieron para obtener las direcciones donde estos se ubican en la memoria externa.

Tabla 4.18 Direcciones de memoria de los *centros* que forman al vector $\mathbf{v}_0^{P_\pi^1}$.

Puntos de interés	$\hat{x}$	$\hat{y}$	Direcciones
$b_0^{P_\pi^0}$	115	124	0x149CDE
$b_1^{P_\pi^0}$	112	130	0x149CDB
$b_2^{P_\pi^0}$	105	130	0x149CD4
$b_3^{P_\pi^0}$	101	124	0x149CD0
$b_4^{P_\pi^0}$	105	118	0x149CD4
$b_5^{P_\pi^0}$	112	118	0x149CDB

Para formar el vector $\mathbf{v}_0^{P_{4\pi/3}^1}$, la Tabla 4.19 contiene las direcciones de las localidades en memoria donde se encuentran sus respectivos puntos de interés.

Tabla 4.19 Direcciones de memoria de los *centros* que forman al vector $\mathbf{v}_0^{P_{4\pi/3}^1}$.

Puntos de interés	$\hat{x}$	$\hat{y}$	Direcciones
$b_0^{P_{4\pi/3}^0}$	123	110	0x15AE56
$b_1^{P_{4\pi/3}^0}$	120	116	0x15AE53
$b_2^{P_{4\pi/3}^0}$	113	116	0x15AE4C
$b_3^{P_{4\pi/3}^0}$	109	110	0x15AE48
$b_4^{P_{4\pi/3}^0}$	113	104	0x15AE4C
$b_5^{P_{4\pi/3}^0}$	120	104	0x15AE53

Por último, la Tabla 4.20 expone las direcciones de cada punto de interés perteneciente al vector $\mathbf{v}_0^{P_{5\pi/3}^1}$, que se utilizarán para acceder a la memoria externa.

Tabla 4.20 Direcciones de memoria de los puntos de interés que forman al vector $\mathbf{v}_0^{P_{5\pi/3}^1}$.

Puntos de interés	$\hat{x}$	$\hat{y}$	Direcciones
$b_0^{P_{5\pi/3}^0}$	139	110	0x16BFD6
$b_1^{P_{5\pi/3}^0}$	136	116	0x16BFD3
$b_2^{P_{5\pi/3}^0}$	129	116	0x16BFCC
$b_3^{P_{5\pi/3}^0}$	125	110	0x16BFC8
$b_4^{P_{5\pi/3}^0}$	129	104	0x16BFCC
$b_5^{P_{5\pi/3}^0}$	136	104	0x16BFD3

De la misma manera, partiendo de la estructura del vector $\mathbf{h}_{\Sigma_1} = \left[\mathbf{v}_0^{P_0^1}, \mathbf{v}_1^{P_{\pi/3}^1}, \ldots, \mathbf{v}_{T-1}^{P_{5\pi/3}^1}\right]$ y dado que su longitud es de $H \times T = 6 \times 6 = 36$ elementos, la expresión 4.5 muestra solamente el vector $\mathbf{v}_0^{P_0^1}$.

$$\mathrm{v}_0^{P_0^1} = [5.31640625, 0, 6.734375, 2.06640625, 0, 8.39453125] \tag{4.5}$$

Subproceso 5. Ahora en este subproceso se obtienen las coordenadas medias del mapa central convolucionado $\mathrm{G}_c^{\Sigma_c}$, utilizando sus dimensiones (255×255 píxeles), aplicando: $cmi\_x = ancho\_i/2$ y $cmi\_y = alto\_i/2$, las cuales son $cmi\_x = ancho\_i/2 = 255/2 = 127.5 \approx 128$ y $cmi\_y = alto\_i/2 = 255/2 = 127.5 \approx 128$, respectivamente. Después, se calculan las coordenadas rectangulares de los H puntos de interés que formarán al vector $\mathbf{h}_{\Sigma_c}$, cuyas ubicaciones están en torno al $cd(0, 0°)$ del $\mathrm{G}_c^{\Sigma_c}$. Para ello, se aplican las expresiones: $x =$

$cmi\_x + [\Sigma_c * cos(\hat{o}_q)]$ y $y = cmi\_y + [\Sigma_c * sen(\hat{o}_q)]$, recordando que $\Sigma_c = 1.5$. Finalmente, se obtienen las direcciones de localidades correspondientes a los puntos de interés. Para lograr esto, se reutiliza el Algoritmo 4.1 para obtener sus posiciones utilizando sus coordenadas rectangulares. Partiendo de la dirección de lectura del $G_c^{\Sigma_c}$ en: 0x900B0. Estas direcciones son mostradas en la Tabla 4.21.

Tabla 4.21 Direcciones de memoria de los puntos de interés que forman al vector $\mathbf{h}_{\Sigma_c}$.

Puntos de interés	x	y	**Direcciones**
$b_0^{cd(0,0°)}$	130	128	0x9FE35
$b_1^{cd(0,0°)}$	129	129	0x9FE34
$b_2^{cd(0,0°)}$	127	129	0x9FE32
$b_3^{cd(0,0°)}$	127	128	0x9FE31
$b_4^{cd(0,0°)}$	127	127	0x9FE30
$b_5^{cd(0,0°)}$	129	127	0x9FE34

Siendo la expresión 4.8 la que muestra el $\mathbf{h}_{\Sigma_c}$ generado.

$$h_{\Sigma_c} = [3.671875, 12.8671875, 12.671875, 17.390625, 13.22265625, 0] \quad (4.6)$$

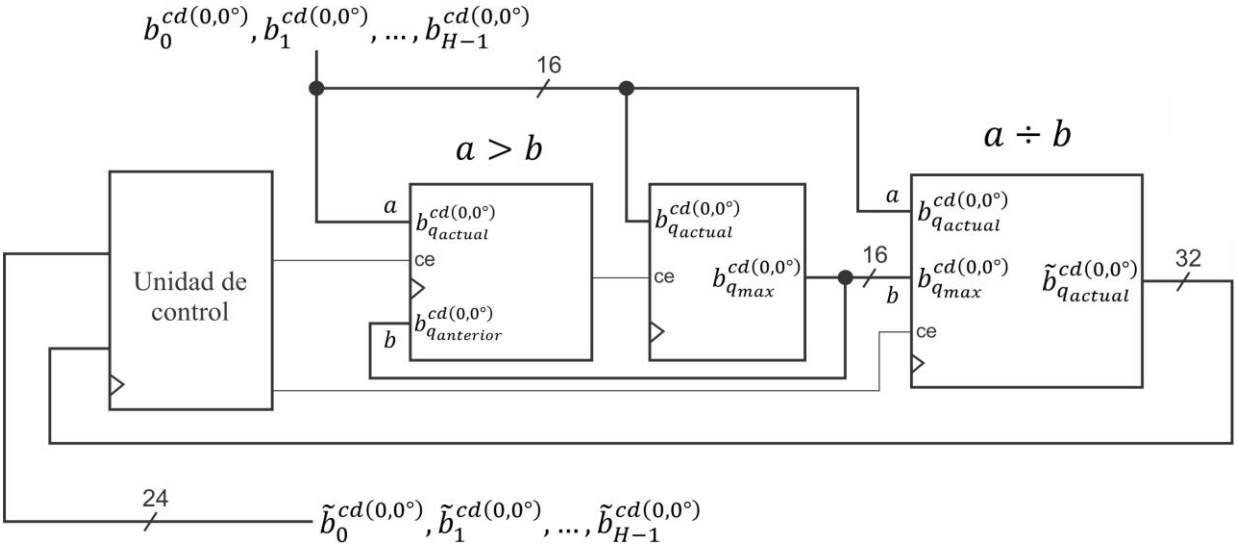


Figura 4.18 Arquitectura hardware para la generación del vector de características.

4.1.4 Generación del vector de características

En este proceso se genera el vector de características que da como resultado final el descriptor DAISY, para lo cual a continuación se detalla la arquitectura hardware diseñada para cumplir con dicha tarea. Para ejemplificar el funcionamiento de la arquitectura, se describe la generación del vector $\mathbf{h}_{\Sigma_c}$, el cual es parte del vector de características final. Primeramente, se realiza la búsqueda del *bin* que tenga el mayor valor en el $\mathbf{h}_{\Sigma_c}$, lo cual se logra utilizando un comparador y un registro, ambos de 16 bits, donde el comparador estará habilitando al registro para que almacene el *bin* cada que se cumpla la condición $b_{q_{actual}}^{cd(0,0°)} > b_{q_{anterior}}^{cd(0,0°)}$. Una vez encontrado el *bin* mayor, la unidad de control habilita el módulo para la división (el cual se ha diseñado utilizando la herramienta *LogiCORE IP Divider Generator v3.0* de la compañía Xilinx, Inc) en donde cada *bin* ingresa a este módulo para aplicar dicha operación, al cual le toma 43 ciclos de reloj para realizarla. Finalizada la operación, el resultado ingresa a la unidad de control para tomar los bits correspondientes a la parte entera y fraccionaría, obteniendo así el *bin* normalizado $\tilde{b}_q^{cd(0,0°)}$ con el formato de punto fijo sin signo $Q8.16$. Estos procedimientos se realizan de la misma manera H veces para cada vector $\mathbf{v}_j^{P_{o_j}^k}$, formando así el vector de características final $\boldsymbol{D}(u_o, v_o)$ basándose en la expresión 3.19 presentada en la subsección 3.1.4. Estos procedimientos quedan ilustrados mediante la Figura 4.18.

Figura 4.19 Objetos usados para el primer y segundo experimento.

4.2 Resultados experimentales

El objetivo de esta sección es presentar los resultados experimentales generados por la arquitectura hardware del descriptor DAISY sobre lógica reconfigurable con ayuda del modelado conceptual de las MAE. Para cumplir con el objetivo, se han realizado diversos experimentos para el clasificado de objetos utilizando la base de datos ALOI. Para exponer estos resultados se usó la matriz de confusión que permite determinar el rendimiento de la arquitectura hardware. Para ello, la subsección siguiente muestra y describe estos experimentos.

4.2.1 Evaluación de la arquitectura hardware

Como parte de esta evaluación, se han dispuesto los siguientes experimentos en donde la configuración en los parámetros del descriptor DAISY: $R_0 = 8, R_0 = 16, Q = 2, H = 6$ y $T = 6$, siempre será la misma.

Para el primer experimento se han elegido al azar 3 clases de objetos los cuales se pueden observar en la Figura 4.19. Para el aprendizaje y clasificación de las MAE se utilizó el operador pmed. En la Tabla 4.22 se puede consultar la matriz de confusión generada para el primer experimento.

Tabla 4.22 Matriz de confusión del primer experimento usando el operador pmed.

	MAE		
Clases	Cesto1	Sombrero	Cesto2
Cesto1	20	0	0
Sombrero	0	19	1
Cesto2	0	0	20

Los resultados del rendimiento de este experimento se pueden observar en la Tabla 4.23, la cual contiene las métricas asociadas a la matriz de confusión anteriormente mostrada.

Tabla 4.23 Resultados del primer experimento usando el operador pmed.

Clases	**exactitud**	**precisión**	**RVP**	**RVN**	**TE**
Cesto1	1	1	1	1	0
Sombrero	0.9833	1	0.95	1	0.016667
Cesto2	0.9833	0.9524	1	0.975	0.016667

En el segundo experimento se emplearon las mismas clases que el experimento anterior (ver Figura 4.19), pero a diferencia del mismo se empleó el operador prom para el aprendizaje y clasificación de las MAE. En la Tabla 4.24 se puede consultar la matriz de confusión generada en este experimento.

Tabla 4.24 Matriz de confusión del segundo experimento usando el operador prom.

	MAE		
Clases	Cesto1	Sombrero	Cesto2
Cesto1	20	0	0
Sombrero	0	20	0
Cesto2	0	0	20

A continuación, en la Tabla 4.25 se presentan los resultados de rendimiento por clase, la cual muestra las métricas calculadas asociadas a la matriz de confusión anterior.

Tabla 4.25 Resultados del segundo experimento usando el operador prom.

Clases	**exactitud**	**precisión**	**RVP**	**RVN**	**TE**
Cesto1	1	1	1	1	0
Sombrero	1	1	1	1	0
Cesto2	1	1	1	1	0

Para el tercer experimento se emplearon 5 clases de objetos que se pueden apreciar en la Figura 4.20, las cuales fueron elegidas al azar. Siendo el operador pmed el utilizado para este experimento. A continuación, en la Tabla 4.26 se puede consultar la matriz de confusión que se generó tras realizar este tercer experimento.

Tabla 4.26 Matriz de confusión del tercer experimento usando el operador pmed.

	MAE				
Clases	Chiles	Peluche	Llavero	Botella	Sombrero
Chiles	18	0	0	2	0
Peluche	0	19	0	1	0
Llavero	0	0	20	0	0
Botella	0	0	0	20	0
Sombrero	0	0	0	1	19

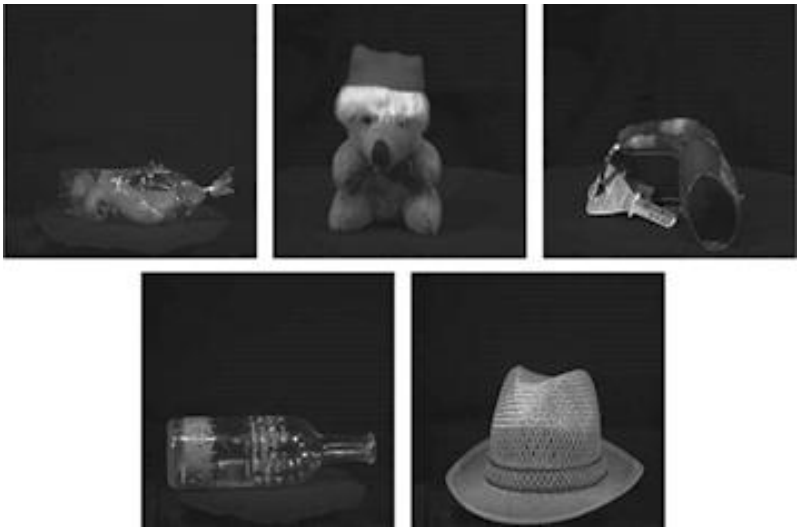


Figura 4.20 Objetos usados para el tercer y cuarto experimento.

Ahora, en Tabla 4.27 se pueden observar los resultados del rendimiento calculados a partir de la matriz de confusión anterior.

Tabla 4.27 Resultados del tercer experimento usando el operador pmed.

Clases	exactitud	precisión	RVP	RVN	TE
Chiles	0.98	1	0.9	1	0.02
Peluche	0.99	1	0.95	1	0.01
Llavero	1	1	1	1	0
Botella	0.96	0.8333	1	0.95	0.04
Sombrero	0.99	1	0.95	1	0.01

El cuarto experimento se realizó con el operador prom, usando las mismas clases que el tercer experimento (ver Figura 4.20). La matriz de confusión generada tras realizarse este experimento se puede consultar en la Tabla 4.28.

Tabla 4.28 Matriz de confusión del cuarto experimento usando el operador prom.

	MAE				
Clases	Chiles	Peluche	Llavero	Botella	Sombrero
Chiles	20	0	0	0	0
Peluche	0	20	0	0	0
Llavero	0	0	20	0	0
Botella	2	0	0	18	0
Sombrero	0	0	0	0	20

Seguidamente, la Tabla 4.29 expone el rendimiento por clase que ofrecen las MAE al interactuar con el descriptor DAISY. La cual contiene las métricas que se derivan de la matriz de confusión de la Tabla 4.28.

Tabla 4.29 Resultados del cuarto experimento usando el operador prom.

Clases	exactitud	precisión	RVP	RVN	TE
Chiles	0.98	0.9091	1	0.975	0.02
Peluche	1	1	1	1	0
Llavero	1	1	1	1	0
Botella	0.98	1	0.9	1	0.02
Sombrero	1	1	1	1	0

Para ejecutar el quinto experimento, se han elegido 9 clases de objetos al azar que se pueden observar en la Figura 4.21. Por otro lado, el operador pmed fue el empleado para el aprendizaje y clasificación de las MAE.

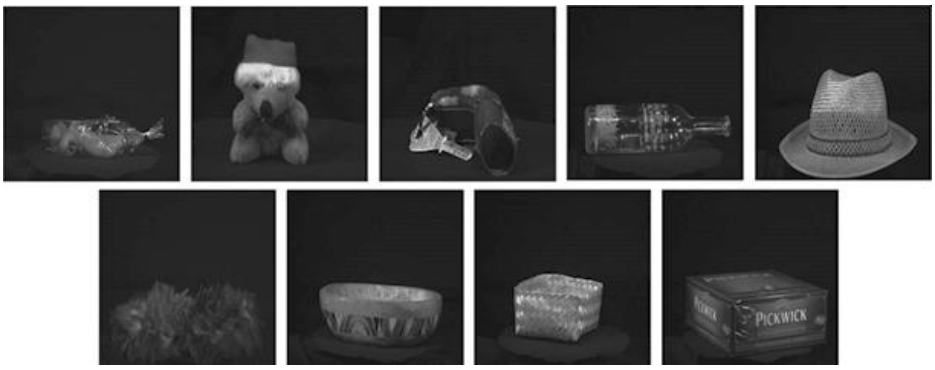


Figura 4.21 Objetos usados para el quinto y sexto experimento.

Siendo la matriz de confusión que se generó la que se presenta en la Tabla 4.30.

Tabla 4.30 Matriz de confusión del quinto experimento usando el operador pmed.

	MAE								
Clases	Chiles	Peluche	Llavero	Botella	Sombrero	Plumero	Tazón	Cesto	Tea
Chiles	19	0	0	0	0	0	1	0	0
Peluche	0	18	0	0	0	2	0	0	0
Llavero	0	0	20	0	0	0	0	0	0
Botella	1	0	0	18	0	1	0	0	0
Sombrero	0	0	0	0	20	0	0	0	0
Plumero	0	0	0	0	0	20	0	0	0
Tazón	0	0	0	0	0	0	19	1	0
Cesto	0	0	0	0	0	0	0	20	0
Tea	0	0	0	0	2	0	1	0	17

De la matriz de confusión anterior, se calculan las métricas correspondientes para determinar el rendimiento del descriptor DAISY interactuando con las MAE, las cuales se presentan en la Tabla 4.31.

Tabla 4.31 Resultados del quinto experimento usando el operador pmed.

Clases	**exactitud**	**precisión**	**RVP**	**RVN**	**TE**
Chiles	0.9877	0.95	0.95	0.9930	0.01227
Peluche	0.9877	1	0.9	1	0.01227
Llavero	1	1	1	1	0
Botella	0.9877	1	0.9	1	0.01227
Sombrero	0.9877	0.9091	1	0.9860	0.01227
Plumero	0.9816	0.8696	1	0.9790	0.018405
Tazón	0.9816	0.9048	0.95	0.9860	0.018405
Cesto	0.9944	0.9524	1	1	0.005556
Tea	0.9833	1	0.85	1	0.018405

En el sexto experimento, se emplearon las mismas clases que el quinto experimento (ver Figura 4.21), pero en este caso se ha utilizado el operador prom. De este experimento se ha generado la matriz de confusión que se muestra en la Tabla 4.32.

Tabla 4.32 Matriz de confusión del sexto experimento usando el operador prom.

	MAE								
Clases	Chiles	Peluche	Llavero	Botella	Sombrero	Plumero	Tazón	Cesto	Tea
Chiles	19	0	0	0	0	1	0	0	0
Peluche	0	20	0	0	0	0	0	0	0
Llavero	0	0	19	1	0	0	0	0	0
Botella	2	0	0	18	0	0	0	0	0
Sombrero	0	0	0	0	20	0	0	0	0
Plumero	0	0	1	0	0	19	0	0	0
Tazón	0	0	0	0	0	0	19	0	1
Cesto	0	0	0	0	0	0	0	20	0
Tea	0	0	0	0	0	0	1	0	18

En la Tabla 4.33 se muestran los datos correspondientes al rendimiento resultante en este experimento que se derivaron de la matriz de confusión anterior.

Tabla 4.33 Resultados del sexto experimento usando el operador prom.

Clases	exactitud	precisión	RVP	RVN	TE
Chiles	0.9814	0.9048	0.95	0.9858	0.018634
Peluche	1	1	1	1	0
Llavero	0.9876	0.95	0.95	0.9929	0.012422
Botella	0.9814	0.9474	0.9	0.9929	0.018634
Sombrero	1	1	1	1	0
Plumero	0.9876	0.95	0.95	0.9929	0.012422
Tazón	0.9944	0.9524	1	0.9937	0.006211
Cesto	0.9888	1	0.9	1	0.005587
Tea	0.9888	1	0.9	1	0.012422

Para el sexto experimento, se han empleado 15 clases (ver Figura 4.22) y el operador pmed. La Tabla 4.34 muestra la matriz de confusión generada.

Tabla 4.34 Matriz de confusión del séptimo experimento usando el operador pmed.

								MAE							
Clases	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	17	0	0	0	1	1	1	0	0	0	0	0	0	0	0
2	0	18	0	0	0	0	0	1	0	0	1	0	0	0	0
3	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	18	0	2	0	0	0	0	0	0	0	0
6	0	0	0	0	0	19	0	0	0	0	0	0	1	0	0
7	0	0	4	3	0	0	13	0	0	0	0	0	0	0	0
8	0	0	0	0	2	0	0	15	0	1	0	1	0	1	0
9	0	0	12	0	0	0	0	2	6	0	0	0	0	0	0
10	0	0	2	4	0	0	0	0	0	14	0	0	0	0	0
11	0	2	0	0	0	0	0	3	0	0	15	0	0	0	0
12	0	0	0	0	0	0	1	1	0	0	0	18	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0
14	2	0	0	0	0	0	0	0	0	0	0	0	0	18	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20

Figura 4.22 Objetos usados para el séptimo y octavo experimento.

En la Tabla 4.35 se pueden consultar los resultados de rendimiento que se calcularon a partir de la matriz de confusión anterior.

Tabla 4.35 Resultados del séptimo experimento usando el operador pmed.

Clases	Exactitud	precisión	RVP	RVN	TE
1	0.9833	0.8947	0.85	0.992857	0.016667
2	0.9867	0.9	0.9	0.992857	0.013333
3	0.9233	0.4651	1	0.917857	0.076667
4	0.9033	0.4082	1	0.896429	0.096667
5	0.9833	0.8571	0.9	0.989286	0.016667
6	0.9933	0.95	0.95	0.996429	0.006667
7	0.92	0.88	0.9	0.985714	0.08
8	0.96	0.6818	0.75	0.975	0.04
9	0.9533	1	0.3	1	0.046667
10	0.93	0.911	0.89	0.996429	0.07
11	0.98	0.9375	0.75	0.996429	0.02
12	0.99	0.9474	0.9	0.996429	0.01
13	0.9967	0.9524	1	0.996429	0.003333
14	0.99	0.9474	0.9	0.996429	0.01
15	1	1	1	1	0

En el octavo experimento, se mantuvo el mismo número de clases (ver Figura 4.22), pero en este caso se utilizó el operador prom. La matriz de confusión generada es mostrada por la Tabla 4.36.

Tabla 4.36 Matriz de confusión del octavo experimento usando el operador prom.

					MAE										
Clases	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	17	0	0	0	1	1	1	0	0	0	0	0	0	0	0
2	0	18	0	0	0	0	0	1	0	0	1	0	0	0	0
3	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	18	0	2	0	0	0	0	0	0	0	0
6	0	0	0	0	0	19	0	0	0	0	0	0	1	0	0
7	0	0	3	0	0	0	17	0	0	0	0	0	0	0	0
8	0	0	0	0	2	0	0	15	0	1	0	1	0	1	0
9	0	0	9	0	0	0	0	0	11	0	0	0	0	0	0
10	0	0	2	4	0	0	0	0	0	14	0	0	0	0	0
11	0	2	0	0	0	0	0	3	0	0	15	0	0	0	0
12	0	0	0	0	0	0	1	1	0	0	0	18	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0
14	2	0	0	0	0	0	0	0	0	0	0	0	0	18	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20

Seguidamente, se muestra en la Tabla 4.37 las métricas asociadas a la matriz de confusión anterior, las cuales corresponden al rendimiento que se obtuvo por cada clase.

Tabla 4.37 Resultados del octavo experimento usando el operador prom.

Clases	exactitud	precisión	RVP	RVN	TE
1	0.9833	0.8947	0.85	0.992857	0.016667
2	0.9867	0.9	0.9	0.992857	0.013333
3	0.9533	0.5882	1	0.95	0.046667
4	0.9867	0.8333	1	0.985714	0.013333
5	0.9833	0.8571	0.9	0.989285	0.016667
6	0.9933	0.95	0.95	0.996429	0.006667
7	0.9767	0.8095	0.85	0.985714	0.023333
8	0.9667	0.75	0.75	0.982143	0.033333
9	0.97	1	0.55	1	0.03
10	0.9767	0.9333	0.7	0.996429	0.023333
11	0.98	0.9375	0.75	0.996429	0.02
12	0.99	0.9474	0.9	0.996429	0.01
13	0.9967	0.9524	1	0.996429	0.003333
14	0.99	0.9474	0.9	0.996429	0.01
15	1	1	1	1	0

Como parte del noveno experimento, solo se han cambiado el número de clases para este caso, el cual se eligieron 20 (ver Figura 4.23) y se ha utilizado el operador pmed. Seguidamente, es mostrada la matriz de confusión que se generó en este experimento, la cual se muestra en la Tabla 4.38.

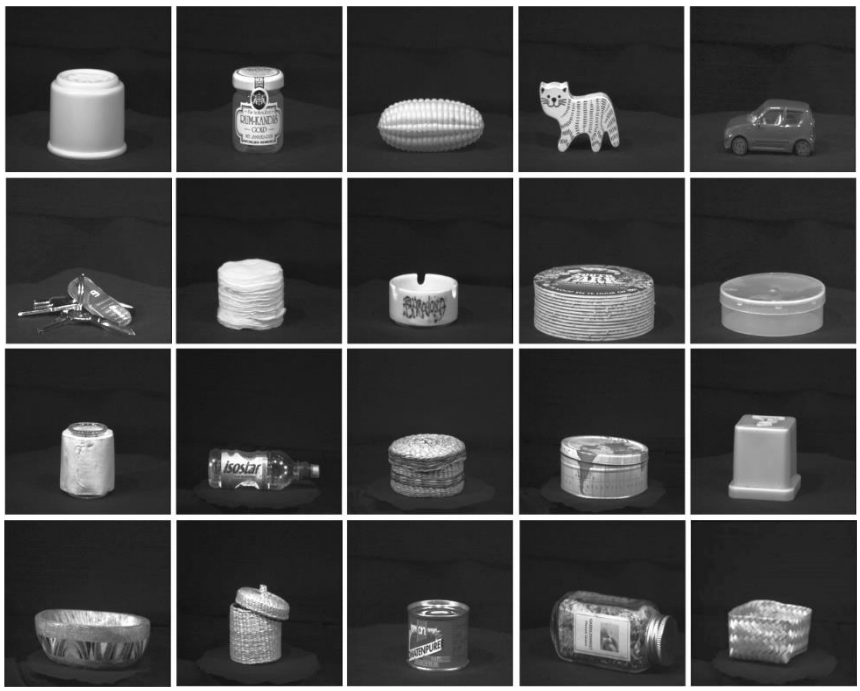


Figura 4.23 Objetos usados para el noveno y décimo experimento.

Tabla 4.38 Matriz de confusión del noveno experimento usando el operador pmed.

	MAE																			
Clases	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
1	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	1
2	0	10	0	0	0	0	0	7	0	0	3	0	0	0	0	0	0	0	0	0
3	0	0	16	0	0	0	0	1	0	0	0	0	1	0	0	0	2	0	0	0
4	0	0	0	17	0	0	0	1	0	2	0	0	0	0	0	0	0	0	0	0
5	0	0	3	0	14	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0	5	0
7	0	0	0	0	0	0	15	0	0	2	0	0	2	0	1	0	0	0	0	0
8	0	0	0	0	0	1	0	13	0	0	4	0	0	0	0	0	0	2	0	0
9	0	0	0	0	0	0	0	2	18	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	3	0	0	5	0	0	12	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	6	0	0	14	0	0	0	0	0	0	0	0	0
12	0	1	0	0	0	0	0	5	0	0	0	13	0	0	0	0	0	0	0	0
13	0	1	1	0	0	0	0	7	0	0	0	0	11	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	18	0	0	0	0	0	2
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	18	2	0	0	0	0
16	0	0	0	0	0	0	0	4	0	0	14	0	0	0	0	2	0	0	0	0
17	0	1	0	0	0	0	0	5	0	0	0	5	0	0	0	0	5	0	0	0
18	3	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	15	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	1
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	17

Siendo los resultados de rendimiento correspondientes a cada clase utilizada los que se observan en la Tabla 4.39.

Tabla 4.39 Resultados del noveno experimento usando el operador pmed.

Clases	exactitud	precisión	RVP	RVN	TE
1	0.9775	0.8235	0.7	0.9921	0.0225
2	0.9675	0.7692	0.5	0.9921	0.0325
3	0.98	0.8	0.8	0.98	0.02
4	0.985	0.85	0.85	0.9921	0.015
5	0.9725	0.7368	0.7	0.9868	0.0275
6	0.985	0.9375	0.75	0.9973	0.015
7	0.975	0.75	0.75	0.9868	0.025
8	0.8875	0.2549	0.65	0.9	0.1125
9	0.995	1	0.9	1	0.005
10	0.9625	0.6316	0.6	0.9815	0.0375
11	0.9325	0.4	0.7	0.9447	0.0675
12	0.9675	0.65	0.6842	0.9716	0.0325
13	0.97	0.7857	0.55	0.9921	0.03
14	0.995	1	0.9	1	0.005
15	0.9925	0.9474	0.9	0.9973	0.007463
16	0.95	0.5	0.1	0.9947	0.005
17	0.955	0.7143	0.238	0.9947	0.045
18	0.9625	0.6	0.75	0.97	0.0375
19	0.9850	0.7917	0.95	0.9868	0.015
20	0.9825	0.8095	0.85	0.9894	0.0175

Por último, se presenta el décimo experimento realizado el cual utiliza el operador prom y las mismas clases que el experimento anterior (ver Figura 4.23). La Tabla 4.40 expone la matriz de confusión que se obtuvo al finalizar este experimento.

Tabla 4.40 Matriz de confusión del décimo experimento usando el operador prom.

	MAE																			
Clases	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
1	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	1
2	2	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	16	0	0	0	0	1	0	0	0	0	1	0	0	0	2	0	0	0
4	0	0	0	17	0	0	0	1	0	2	0	0	0	0	0	0	0	0	0	0
5	0	0	3	0	14	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0	5	0
7	0	0	0	0	0	0	15	0	0	2	0	0	2	0	1	0	0	0	0	0
8	0	0	0	0	0	1	0	13	0	0	4	0	0	0	0	0	0	2	0	0
9	0	0	0	0	0	0	0	2	18	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	3	0	0	5	0	0	12	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	6	0	0	14	0	0	0	0	0	0	0	0	0
12	0	1	0	0	0	0	0	5	0	0	0	13	0	0	0	0	0	0	0	0
13	0	1	1	0	0	0	0	7	0	0	0	0	11	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	18	0	0	0	0	0	2
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	18	2	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0
18	3	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	15	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	1
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	17

Donde, el rendimiento por clase se puede consultar en la Tabla 4.41.

Tabla 4.41 Resultados del décimo experimento usando el operador prom.

Clases	exactitud	precisión	RVP	RVN	TE
1	0.9724	0.7368	0.7	0.9868	0.027569
2	0.99	0.9	0.9	0.9947	0.010025
3	0.9799	0.8	0.8	0.9894	0.02005
4	0.985	0.85	0.85	0.9920	0.015038
5	0.985	1	0.7	1	0.015038
6	0.985	0.9375	0.75	0.9973	0.015038
7	0.9749	0.75	0.75	0.9868	0.025063
8	0.9273	0.3714	0.65	0.9419	0.072682
9	0.995	1	0.9	1	0.002013
10	0.9624	0.6316	0.6	0.9815	0.037594
11	0.9749	0.7778	0.7	0.9894	0.025063
12	0.9799	0.8667	0.6842	0.9947	0.02005
13	0.9699	0.7857	0.55	0.9920	0.030075
14	0.995	1	0.9	1	0.005013
15	0.9925	0.9474	0.9	0.9973	0.007481
16	0.995	0.9091	1	0.9947	0.005013
17	0.995	0.9091	1	0.9947	0.005013
18	0.9624	0.6	0.75	0.9736	0.037594
19	0.95	0.79	0.95	0.98	0.013018
20	0.9822	0.8095	0.85	0.9894	0.017544

Los porcentajes promedios de reconocimiento correspondientes a cada experimento realizado en esta sección se muestran a continuación en la Tabla 4.42. En la cual también son incluidos los porcentajes promedios de reconocimiento del modelado conceptual, con la finalidad de realizar una comparativa entre ellos y determinar el desempeño de la arquitectura hardware implementada.

Tabla 4.42 Porcentajes de reconocimiento promedio por operador.

Clases	Modelado conceptual		Arquitectura hardware	
	pmed	**prom**	**pmed**	**prom**
3	100%	100%	99%	100%
5	98%	99%	97%	96%
9	96%	98%	95%	96%
15	78%	79%	76%	77%
20	74%	71%	71%	70%

Ahora, en la Tabla 4.43 se muestra el uso de recursos de lógica reconfigurable por la implementación en hardware del descriptor DAISY, así como también se indica la frecuencia máxima a la que opera el descriptor y el tiempo por cada ciclo de reloj.

Tabla 4.43 Resumen de utilización del dispositivo por la arquitectura hardware del descriptor DAISY.

Recursos	Uso		Frecuencia máxima de operación (MHz)
	Usado - Total	Porcentaje	
Slices Registers	2481 de 18224	13	27.482
Slices LUTs	4235 de 9112	46	
- Usados como lógica	2484 de 9112	27	
- Usados como memoria	1632 de 2176	75	
RAM	1 de 64	1	

Así mismo, en la Tabla 4.44 se muestran los tiempos en milisegundos que le toma al modelado conceptual y a la arquitectura hardware del descriptor DAISY generar el vector de características final.

Tabla 4.44 Tiempo en ms para generar el vector de características.

Vector de características	
Modelado conceptual	Arquitectura hardware
232.54	774.74

En términos generales, de los experimentos realizados a la implementación en hardware del descriptor DAISY se obtuvieron altos porcentajes de reconocimiento al interactuar con las MAE, empleando los operadores pmed y prom. Lo que significa que a pesar de haber elegido utilizar la representación en formato de punto fijo para aproximar los datos del algoritmo, el desempeño de este no se vio afectado como bien se observa en la Tabla 4.42, en donde se aprecian que los porcentajes promedios de reconocimiento entre el modelado conceptual y de la arquitectura hardware, son muy similares. Por otro lado, el uso de los recursos del FPGA alcanzó aproximadamente el 46%, debido a que los recursos para la implementación del algoritmo son limitados, ya que se utilizó un FPGA de gama baja. Por último, el tiempo de procesamiento que requiere el descriptor DAISY para generar el vector de características en el FPGA sufrió un incremento en comparación con el tiempo obtenido en el modelado conceptual (ver Tabla 4.44), debido a los procesos de convolución iterativos que se realizaron en la fase 2 del algoritmo. Es por este último punto que se realizó un diseño de una arquitectura hardware para disminuir el tiempo de procesamiento del algoritmo en el FPGA, la cual será comparada con la implementación en hardware diseñada en este trabajo de investigación.

La Figura 4.24 muestra el diseño de la arquitectura hardware de las fases 1 y 2 del descriptor DAISY, la cual se orientó para un FPGA de gama baja que se utilizó en la presente investigación. En la cual se puede observar que en la fase 1 se obtienen los mapas de orientación secuencialmente y en la fase 2 se realizan convoluciones secuenciales para obtener los mapas de orientación convolucionados para el primer y segundo anillo, para lo cual se utilizan filtros gaussianos de 5×5 y 7×7, respectivamente, empleando una arquitectura hardware iterativa para el cálculo de sumas de productos, requiriendo 1 multiplicador, 1 sumador y 1 registro que funciona como acumulador para la sumatoria. Con ello, se logra un uso de recursos del FPGA necesarios, sin embargo, el tiempo de procesamiento total del algoritmo se ve afectado como lo indica la Tabla 4.44.

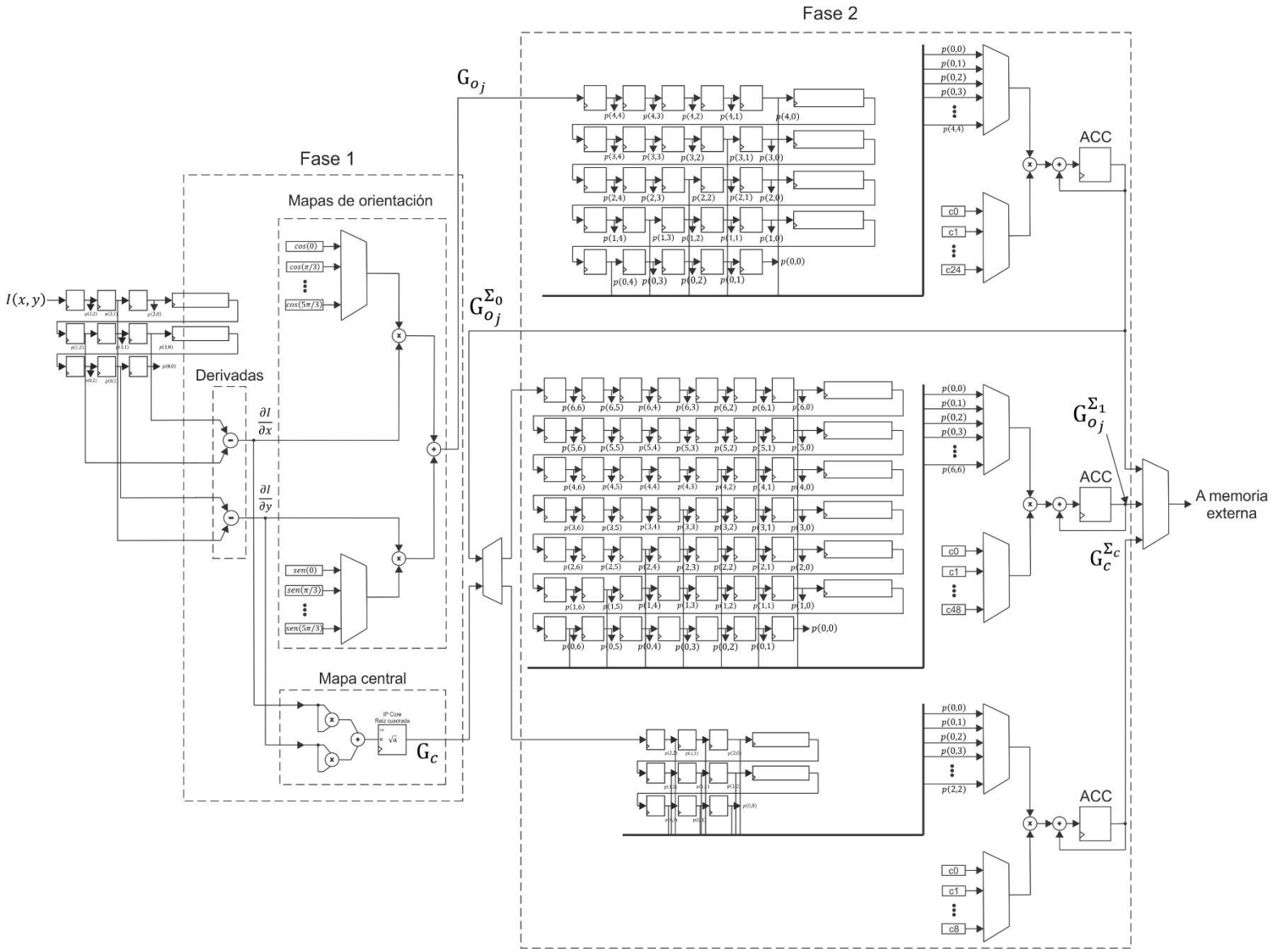


Figura 4.24 Arquitectura hardware secuencial de las fases 1 y 2.

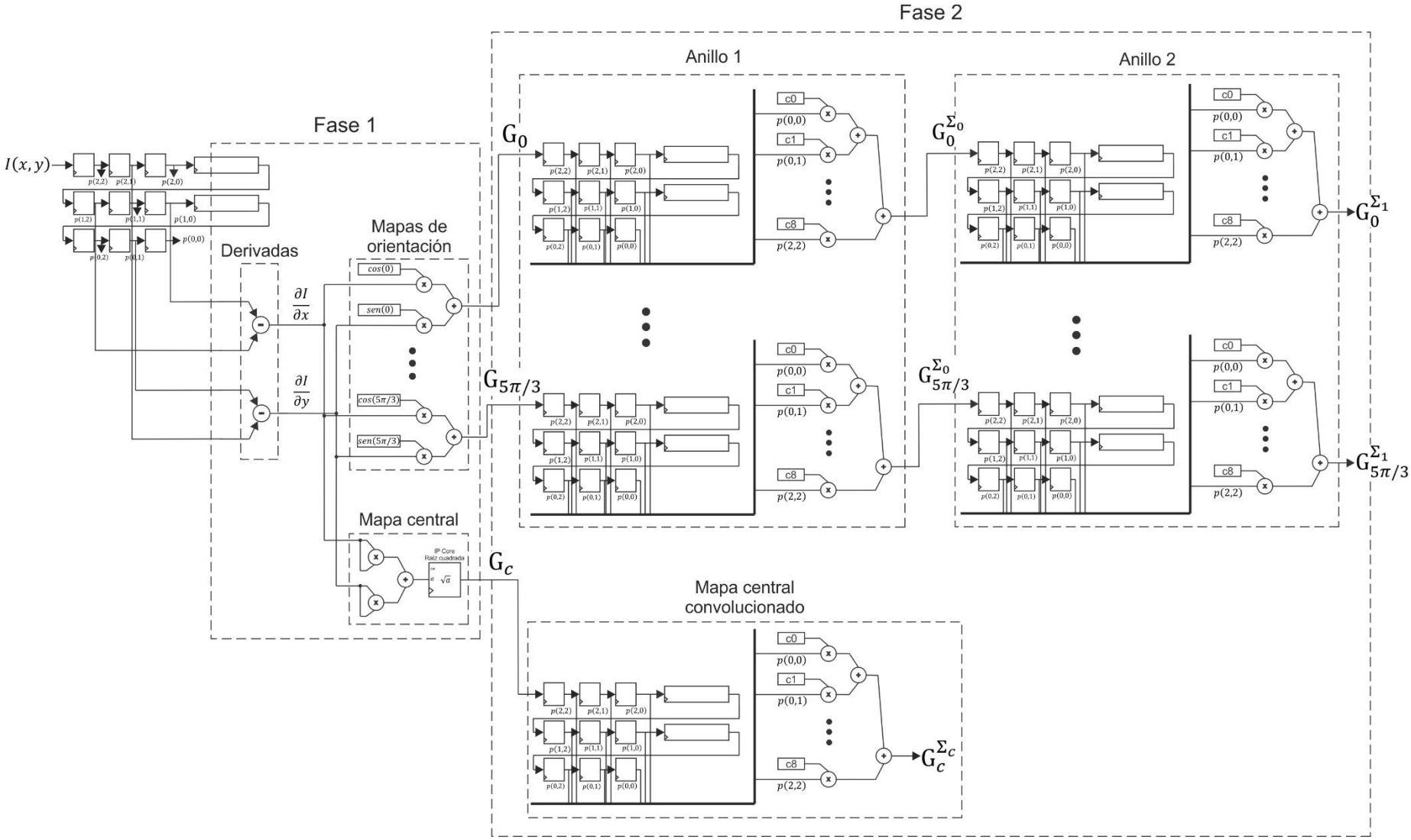


Figura 4.25 Arquitectura hardware concurrente de las fases 1 y 2.

Para esto, en la Figura 4.25 se muestra el diseño de una arquitectura hardware concurrente para dichas fases, orientada para un FPGA de gama alta que posea un mayor número de recursos. Por un lado, en la fase 1 se calculan de forma paralela las derivadas de la imagen de entrada y a su vez, cada mapa de orientación y el mapa central. Por otro lado, en la fase 2, tanto para el anillo 1 como para anillo 2 y para el mapa central, se sustituyeron los filtros gaussianos de 5×5 y 7×7 por filtros pequeños de 3×3 para ambos anillos. Ademas, se reemplazó la arquitectura hardware iterativa por una de árbol para el cálculo paralelo de sumatorias de productos, en donde ésta última utiliza 25 multiplicadores y 24 sumadores por cada mapa del anillo 1, y 49 multiplicadores y 48 sumadores por cada mapa del anillo 2. De la misma manera, para convolucionar el mapa central se emplearon 9 multiplicadores y 8 sumadores. Lo anterior ocasiona que ahora el uso de los recursos del FPGA se vea afectado, obtenido una mejora considerable en el tiempo de procesamiento del descriptor DAISY, como lo muestra la Tabla 4.45, en donde se realiza una comparativa de los tiempos de procesamiento por fase y el tiempo total entre las arquitecturas antes descritas, los cuales están expresado en el orden de los milisegundos.

Tabla 4.45 Tiempos de procesamiento por fase del descriptor DAISY.

<table>
<tr><th rowspan="2">Fase</th><th colspan="4">Arquitectura hardware</th></tr>
<tr><th colspan="2">Secuencial</th><th colspan="2">Concurrente</th></tr>
<tr><td>1</td><td>35.483</td><td rowspan="4">774.931</td><td>5.07</td><td rowspan="4">9.28256</td></tr>
<tr><td>2</td><td>739.351</td><td>4.136</td></tr>
<tr><td>3</td><td>0.022</td><td>0.001560</td></tr>
<tr><td>4</td><td>0.075</td><td>0.075</td></tr>
</table>

Con esta comparativa realizada, se determina que la arquitectura hardware de tipo concurrente se convierte en una opción para mejorar los tiempos de procesamiento del algoritmo, especialmente para la fase 2, ya que, a diferencia de las otras fases, en ella se realizan cálculos con más desgaste computacional.

Capítulo 5 Conclusiones y trabajos futuros

En este capítulo se presentan las conclusiones que se originaron después de haber observado y analizado el desempeño de la arquitectura hardware para el descriptor DAISY implementada sobre lógica reconfigurable. Además, se exponen las conclusiones correspondientes al rendimiento que ofreció la implementación secuencial del descriptor DAISY. Por otro lado, también se mencionan los posibles trabajos que en un futuro darán continuidad a la presente investigación.

5.1 Conclusiones

Los resultados obtenidos por la implementación del algoritmo DAISY sobre lógica reconfigurable permiten asegurar que este descriptor es una atractiva alternativa para ser utilizado en la fase de extracción de características de un sistema de reconocimiento que trabaje en sistemas autónomos. Esta aseveración se fundamenta, por una parte, en que dicha implementación presentó un alto porcentaje de reconocimiento, superior al 96% para 5 clases o menos y superior al 95% para 9 clases. Por otro lado, el porcentaje de recursos requeridos por la implementación, aproximadamente fue del 46%, el cual parece alto, pero hay que considerar que la implementación fue realizada en un FPGA de gama baja.

Por una parte, a partir del modelado conceptual del descriptor DAISY se determinaron las siguientes conclusiones:

- El funcionamiento del descriptor DAISY se pudo analizar mediante su modelado conceptual, permitiendo identificar con precisión qué tipo de recursos hardware son los que necesita cada una de las fases que lo conforman. Esto facilita la implementación sobre lógica reconfigurable, ya que permite identificar qué tipo de técnicas son las idóneas para realizar de manera eficiente el diseño y modelado de la arquitectura hardware adecuada y su posterior implementación sobre lógica reconfigurable.
- Los resultados obtenidos de los diferentes experimentos realizados al modelado conceptual del descriptor DAISY presentaron altos porcentajes de reconocimiento, por ejemplo, usando 3 clases se obtuvo 100% de reconocimiento empleando ambos operadores (pmed y prom). Para el caso de uso de 5 clases se obtuvo un 98% de reconocimiento empleando el operador pmed y aproximadamente 99% de reconocimiento cuando se utilizó el operador prom. Así mismo, para 9 clases se obtuvo por arriba del 96% de reconocimiento utilizando el operador pmed y cerca del 98% para el caso de uso del operador prom. Las configuraciones en los parámetros del descriptor DAISY que propiciaron estos resultados fueron: $R_0 = 8$, $R_1 = 16$, $Q = 2$, $H = 6, T = 6$ y $R_0 = 7.5$, $R_1 = 15$, $Q = 2$, $H = 8, T = 8$.
- Se determinó que las MAE ofrecen un alto desempeño, ya que, por un lado, poseen una baja complejidad computacional y, por otro lado, son robustas cuando clasifican patrones alterados con ruido, donde sus elementos son de valor real volviéndolas una interesante opción para este tipo de aplicaciones.

Por su parte, respecto a la implementación en hardware del descriptor DAISY se resaltan las siguientes conclusiones:

- Como resultado de los diferentes experimentos aplicados a la implementación, se obtuvieron altos porcentajes de reconocimiento, siendo aproximadamente el 99% de reconocimiento para 3 clases utilizando el operador pmed y 100% empleando el operador prom. Para el uso de 5 clases presentó por encima del 97% de reconocimiento con el operador pmed y superior al 96% con el operador prom. Empleando 9 clases se obtuvo por arriba del 95% de reconocimiento al utilizar el operador pmed y un 96% de reconocimiento empleando el operador prom.
- El uso de la ventana deslizante basada en buffers de línea ofrece una gran facilidad y rapidez para realizar el proceso de convolución, ya que con ella se acceden a los píxeles que se encuentran dentro de las dimensiones de la ventana deslizante de forma simultánea en cada ciclo de reloj. Lo cual repercute en el consumo de recursos del FPGA, debido a que este tipo de arquitectura se construye puramente con registros, a diferencia de otras arquitecturas que emplean bloques de memoria RAM para construir los buffers de línea, lo que incrementa el consumo de recursos del FPGA.
- La generación de los vectores que forman al vector de características, se basa solamente en seleccionar los puntos de interés que se encuentren en sus respectivas orientaciones.

Esto representa una forma alternativa de calcular los histogramas como lo hacen otros algoritmos, debido a que no posee complejidad computacional, ya que para esta implementación solamente se emplearon accesos a memoria para obtener los valores de los puntos de interés.

- Se eligió usar la representación en formato de punto fijo debido a que el ahorro de recursos es mayor al que se pueda conseguir si se utiliza punto flotante. Sin embargo, emplear punto fijo para la representación de los datos del algoritmo afecta en la resolución de estos mismos, lo que significa que no se tendrá la máxima resolución del dato como se tiene si se utiliza punto flotante, esto debido a que la resolución se limita en cierta cantidad de bits para aproximar la parte entera y fraccionaria del dato. No obstante, lo anterior no asegura que debido a eso se obtengan resultados de reconocimientos bajos, por lo contrario, se comprobó que la implementación en hardware del algoritmo a pesar de utilizar punto fijo para aproximar los datos ofreció altos porcentajes de reconocimiento.

5.2 Trabajos futuros

- Un posible trabajo que dé continuidad al realizado en esta investigación, consiste en someter a diversas evaluaciones a la implementación en hardware del descriptor DAISY con otro tipo de clasificadores o redes neuronales y/u otras bases de datos para determinar su mejor desempeño.
- Otro trabajo que se propone, es dotar de más robustez al descriptor DAISY, como, por ejemplo, que sea invariante a la rotación del objeto, lo cual implicaría realizar la rotación del descriptor, lo cual significa cambiar las orientaciones de los *bins* de cada vector existente en cada anillo.
- Se propone un trabajo que realice pruebas a la implementación en hardware del descriptor DAISY con el propósito de determinar su rendimiento, empleando imágenes con objetos sin segmentar en la fase de clasificación, cuyo entrenamiento de la red neuronal y/o clasificador emplearía las imágenes con los objetos segmentados.

Referencias

[1] Tou, J.T. & González, R.C. (1974). Pattern Recognition Principles. Addison-Wesley Publishing Co, Inc. USA.

[2] Sosa, J. T. R., Ojeda, M. A., & Del Rosario, L. R. (2011). Teoría de la mente, reconocimiento facial y procesamiento emocional en la esquizofrenia. *Revista de psiquiatría y salud mental*, *4*(1), 28-37.

[3] Pencue, E. L., & León-Téllez, J. (2003). Detección y clasificación de defectos en frutas mediante el procesamiento digital de imágenes. *Revista Colombiana de Física*, *35*(1), 148-151.

[4] Valderrama, C. E., & Ulloa, G. (2012). Análisis espectral de parámetros fisiológicos para la detección de emociones. *Sistemas & Telemática*, *10*(20), 27-49.

[5] Volcanes, R. A., Lameda, C., & Lameda, O. (2008). Sistema para detección de tumores en imágenes gastroscópicas utilizando técnica de encadenamiento difuso de pirámide y redes neuronales. *Revista Ingeniería UC*, *15*(2).

[6] Contreras, S., & De la Rosa, F. (2016). Aplicación de deep learning en robótica móvil para exploración y reconocimiento de objetos basados en imágenes. In *Computing Conference (CCC), 2016 IEEE 11th Colombian* (pp. 1-8). IEEE.

[7] Tola, E., Lepetit, V., & Fua, P. (2008, June). A fast local descriptor for dense matching. In Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on (pp. 1-8). IEEE.

[8] Dalal N, Triggs B. (2005). Histograms of Oriented Gradients for Human Detection. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1, 886-893.

[9] Bay H, Tuytelaars T, Van Gool L. (2008). SURF: Speeded up robust features. Proceedings of the European Conference on Computer Vision, 110, 346–359.

[10] Szeliski, R. (2010). Computer vision: algorithms and applications. Springer Science & Business Media.

[11] Sobrado, E.A. (2003). Sistema de visión artificial para el reconocimiento y manipulación de objetos utilizando un brazo robot. Tesis de Maestría, Pontificia Univ. Católica del Perú, Lima, Perú.

[12] Sossa, J. H. (2006). Rasgos descriptores para el reconocimiento de objetos. Instituto Politécnico Nacional, México.

[13] Forsyth, D.A., and Ponce, J. Computer Vision: A Modern Approach. Prentice Hall. USA. 2011.

[14] Rossius S. (2012). Reconocimiento de objetos mediante WebCam en tiempo real. In Escuela Politécnica de Valencia, 2012Ingeniería Técnica de Telecomunicaciones (pp. 6). IEEE.

[15] Gómez F. W. (2015). Reconocimiento de objetos en fotografías. In CINVESTAV-LTI Top Tamaulipas, 2015.

[16] Gonzalez, R. C., & Woods, R. E. (2006). Digital image processing. Prentice-Hall, USA.

[17] Tuytelaars, T., & Mikolajczyk, K. (2008). Local invariant feature detectors: a survey. Foundations and trends® in computer graphics and vision, 3(3), 177-280.

[18] Ojala, T., Pietikäinen, M., & Harwood, D. (1996). A comparative study of texture measures with classification based on featured distributions. Pattern recognition, 29(1), 51-59.

[19] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. International journal of computer vision, 60(2), 91-110.

[20] Mikolajczyk, K. & Schmid C. (2005). A performance evaluation of local descriptors. IEEE transactions on pattern analysis and machine intelligence, 27(10), 1615-1630.

[21] Hebb D. O. (1949). *The Organization of Behavior: A Neuropsychological Theory*. John Wiley and Sons, New York. ISBN-13: 978-0805843002.

[22] Yáñez, C. y Díaz de León J. L. (2003). *Introducción a las Memorias Asociativas*. Centro de Investigación en Computación, IPN. México. ISBN 970-36-0116-2.

[23] Sossa, H., Barrón, R., Vázquez, A.: Real-valued Patterns Classification based on Extended Associative Memory. In: Fifth Mexican Int. Conf. on Computer Science, ENC 2004, IEEE Computer Society, pp. 213-219. (2004).

Referencias

[24] Steinbuch, K.: Die Lernmatrix, Kybernetik. 1(1), 26-45 (1961).

[25] Barron, R.: Associative Memories and Morphological Neural Networks for Patterns Recall (In Spanish). PhD dissertation, Center for Computing Research - National Polytechnic Institute. (2006).

[26] Vázquez, R. A. (2005). Reconocimiento de objetos en presencia de traslapes por medio de memorias asociativas y descripciones invariantes. Tesis de Maestría, Centro de Investigación en Computación – Instituto Politécnico Nacional.

[27] Fischer, J., Ruppel, A., Weisshardt, F., & Verl, A. (2011). A rotation invariant feature descriptor O-DAISY and its FPGA implementation. In Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on (pp. 2365-2370). IEEE.

[28] Yao, L., Feng, H., Zhu, Y., Jiang, Z., Zhao, D., & Feng, W. (2009, December). An architecture of optimised SIFT feature detection for an FPGA implementation of an image matcher. In *2009 International Conference on Field-Programmable Technology* (pp. 30-37). IEEE.

[29] Bonato, V., Marques, E., & Constantinides, G. A. (2008). A parallel hardware architecture for scale and rotation invariant feature detection. *IEEE transactions on circuits and systems for video technology*, *18*(12), 1703-1712.

[30] Svab, J., Krajnik, T., Faigl, J., & Preucil, L. (2009, November). FPGA based speeded up robust features. In *2009 IEEE International Conference on Technologies for Practical Robot Applications* (pp. 35-41). IEEE.

[31] Čížek, P., & Faigl, J. (2017). Real-Time FPGA-Based Detection of Speeded-Up Robust Features Using Separable Convolution. *IEEE Transactions on Industrial Informatics*, *14*(3), 1155-1163.

[32] Sledeviè, T., Serackis, A., & Plonis, D. (2018, November). FPGA-Based Selected Object Tracking Using LBP, HOG and Motion Detection. In *2018 IEEE 6th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)* (pp. 1-5). IEEE.

[33] Jin, S., Kim, D., Nguyen, T. T., Jun, B., Kim, D., & Jeon, J. W. (2009, July). An FPGA-based parallel hardware architecture for real-time face detection using a face certainty map. In *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors* (pp. 61-66). IEEE.

[34] Heo, H., Lee, J. Y., Lee, K. Y., & Lee, C. H. (2013, October). FPGA based implementation of FAST and BRIEF algorithm for object recognition. In *2013 IEEE International Conference of IEEE Region 10 (TENCON 2013)* (pp. 1-4). IEEE.

[35] Kapela, R., Gugala, K., Sniatala, P., Swietlicka, A., & Kolanowski, K. (2015). Embedded platform for local image descriptor-based object detection. *Applied Mathematics and Computation*, *267*, 419-426.

[36] https://www.epfl.ch/labs/cvlab/software/descriptors-and-keypoints/daisy/

[37] Geusebroek, J. M., Burghouts, G. J., & Smeulders, A. W. (2005). The Amsterdam library of object images. *International Journal of Computer Vision*, *61*(1), 103-112.