SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN PROCESADOR DIGITAL DE SEÑALES

ARLES FELIPE GARCIA MAYA
JUAN CAMILO MORENO RUIZ

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
FACULTAD DE INGENIERÍAS: ELÉCTRICA, ELECTRÓNICA, FÍSICA Y CIENCIAS DE LA COMPUTACIÓN
PROGRAMA DE INGENIERÍA ELECTRÓNICA
PEREIRA, COLOMBIA
MAYO DE 2014

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN PROCESADOR DIGITAL DE SEÑALES

ARLES FELIPE GARCIA MAYA
JUAN CAMILO MORENO RUIZ

PROYECTO DE GRADO PARA OPTAR AL TÍTULO DE INGENIEROS ELECTRÓNICOS

DIRECTOR
MSc. EDWIN ANDRÉS QUINTERO SALAZAR

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
FACULTAD DE INGENIERÍAS: ELÉCTRICA, ELECTRÓNICA, FÍSICA Y CIENCIAS DE LA COMPUTACIÓN
PROGRAMA DE INGENIERÍA ELECTRÓNICA
PEREIRA, COLOMBIA
MAYO DE 2014

Nota de aceptación

_____
_____
_____
_____
_____
_____
_____
_____
_____

_____
Firma del Director

_____
Firma del jurado

Pereira, Mayo de 2014

Este proyecto de grado está dedicado a todas aquellas personas que entre familiares, amigos y profesores, creyeron en nosotros y pusieron su granito de arena en nuestra formación y educación como personas, y sobre todo como ingenieros de este país.

Agradecimientos

Son muchas las personas que durante el transcurrir de nuestras vidas han estado a nuestro lado día a día, que han puesto mucho esfuerzo, trabajo, energía, tiempo y dedicación para que podamos ir alcanzando y cumpliendo nuestros más grandes sueños. De este modo, es muy importante para nosotros demostrar que toda la fe y recursos que han depositado en nosotros, han rendido frutos; y por lo tanto, darles nuestros más sinceros agradecimientos, es lo menos que podemos hacer para intentar retribuir algo de lo que con tanta voluntad y dedicación nos han brindado.

Nuestros padres son los primeros a los cuales debemos agradecerles, ya que simple y sencillamente ellos son los artífices de que hoy por hoy estemos aquí, poniendo en práctica todas sus enseñanzas y directrices que, a través de más de 20 años nos han ofrecido a diario.

A nuestros profesores del colegio, que gracias a ellos aprendimos las herramientas básicas, las cuales, con el paso del tiempo nos han permitido adquirir y desarrollar nuevas habilidades y competencias, que hoy están siendo puestas a prueba. También y con gran orgullo, agradecer enormemente a nuestros profesores de ingeniería, que sin importar su nivel educativo y ocupaciones, dedicaron y pusieron gran empeño en transmitir de la mejor manera, sus conocimientos y habilidades hacia nosotros, para que sean utilizadas de la mejor manera y con el objetivo de desarrollar la ciencia y tecnología en este país.

Y finalmente pero no menos importante, agradecerle enormemente a nuestro director de proyecto el MSc Edwin Andrés Quintero, quien nos ha guiado a través de este camino bastante sinuoso, proporcionado sus amplios conocimientos y tiempo para lograr el buen término de este proyecto.

Resumen

Los sistemas de seguridad para el control de acceso a lugares restringidos se han convertido en estos últimos años en un tema de bastante importancia, no solo para las grandes empresas, sino también para la gente del común, que quiere mantener a salvo y bajo su protección sus bienes más preciados de manos inescrupulosas; problema que se ha visto durante toda la historia de la humanidad.

Debido a lo anterior, en este proyecto se decidió desarrollar en MatLab® e implementar en el ADSP-BF533 EZ-Kit Lite® de Analog Devices un sistema de reconocimiento de iris que hace uso de la Transformada Wavelet de Haar 2D y las herramientas del procesamiento digital de imágenes como los filtros espaciales, detector de bordes y análisis de envolvente convexa, para segmentar, normalizar, codificar y validar la imagen del iris de un sujeto de prueba tomado de la base datos CASIA Iris Databa V4.0 contra el sujeto previamente registrado en el sistema.

Los algoritmos creados para MatLab® están desarrollados utilizando solo funciones lógicas y matemáticas básicas, es decir, sin el uso de toolbox especializados en el procesamiento digital de imágenes; esto, con el objetivo de poder desarrollar primeramente en MatLab® y posteriormente implementar los mismos algoritmos en el DSP. Igualmente esto se realizó con el objetivo de finalmente evaluar el rendimiento del sistema en ambas plataformas y obtener un punto de comparación entre ambos sistemas.

Posteriormente se procedió a realizar un conjunto de pruebas tanto en MatLab® como en el ADSP-BF533 EZ-Kit Lite®, utilizando el método de validación cruzada Leave One Out para determinar el porcentaje de aceptabilidad y confiabilidad del sistema de reconocimiento de iris; las cuales arrojaron un mejor resultado en ambas pruebas para el sistema desarrollado en MatLab®; sin embargo el sistema desarrollado en el DSP ofrece ventajas significativas de implementación real al ser este un sistema embebido especialmente diseñado para el procesamiento digital de imágenes.

Introducción

Desde tiempos inmemorables se han desarrollado distintos mecanismos para mantener protegidos lugares de accesos no autorizados, utilizando desde un simple alambre para cerrar el lugar, pasando por las más recientes y muy populares cerraduras que utilizan llaves o claves mecánicas, llegando hasta las modernas que ofrecen control de acceso usando tarjetas identificación por radiofrecuencia, o utilizando contraseñas con diferentes métodos y técnicas de cifrado digital.

El desarrollo de nuevos, mejores, rápidos y sobre todo más confiables sistemas de seguridad es una necesidad creciente en un mundo donde se hace cada día más importante garantizar y verificar completamente la identidad de una persona; todo esto con el fin minimizar las posibilidades de intento de suplantación de identidad y/o acceso de personal no autorizado a sitios sensibles, tanto para una empresa, como para las personas del común.

De este modo, y respondiendo a esta necesidad se hace necesario, importante y muy pertinente que desde el campo de la ingeniería electrónica; la cual ofrece el uso de las diferentes herramientas del procesamiento digital de imágenes como lo son los filtros espaciales, las transformada Wavelet de Haar 2D y los algoritmos de detección de bordes y de patrones geométricos; se desarrolle un sistema de reconocimiento de iris que ofrezca mejorar algunas de las características de este tipo de sistemas. Así, este proyecto plantea el desarrollo y simulación en el software matemático MatLab® y su posterior implementación en un DSP ADSP-BF533 de Analog Devices, de un sistema de reconocimiento de iris, el cual, hace uso de los iris de 200 personas diferentes pertenecientes a la base de datos CASIA Iris Databas,

Los últimos desarrollos de sistemas en el mercado se basan en las nuevas técnicas de biometría, que utilizan el procesamiento de imágenes en conjunto con análisis matemáticos y estadísticos para el reconocimiento de diferentes patrones del cuerpo humano como la huella dactilar o rasgos faciales de cada persona para validar su identidad; de este modo, el desarrollo de un sistema de reconocimiento de iris simulado en MatLab® e implementado en un procesador digital de señales; el cual, utiliza las herramientas existentes en reconocimiento de iris y procesamiento digital de imágenes como filtros y transformadas espaciales, ofrecería una solución a la creciente necesidad, tanto de las empresas como de la gente del común, de dispositivos de control de acceso mucho más confiables, seguros y eficientes que los disponibles actualmente. Así, desarrollando y simulando en MatLab® con más de 3000 imágenes y 200 usuarios diferentes aportados por la base de datos de iris CASIA Iris Database, y su posterior implementación en un DSP ADSP-BF533 de Analog Devices, arroja una alta confiabilidad en el control de acceso y un bajo tiempo de verificación de identidad; todo esto gracias al uso de poderosas herramientas como la Transformada Wavelet de Haar 2D, filtros espaciales tipo Pillbox y la técnica de detección de bordes Sobel.

Así, el desarrollo de este sistema de reconocimiento de iris ofrece una solución de alta tecnología y confiabilidad al usar las herramientas del procesamiento digital de imágenes en conjunto con sistemas embebidos como lo son los procesadores digitales de señales, los cuales brindan un alto rendimiento en este tipo de aplicaciones con imágenes.

Índice General

1. PRELIMINARES……..17
1.1 Definición del Problema. 18
1.2 Justificación. 20
1.3 Objetivos. 21
1.3.1 Objetivo General. 21
1.3.2 Objetivos Específicos. 21
1.4 Marco de Antecedentes. 22

2. Procesamiento Digital de Imágenes 26
2.1 Concepto de Imagen. 27
2.2 Espacio de Color 27
2.3 Transformada Wavelet. 29
2.3.1 Introducción 29
2.3.2 Transformada de Wavelet de Haar 30
2.3.3 Transformada de Wavelet vs Transformada de Fourier 31
2.3.4 Transformada Wavelet de Haar 2D 36
2.4 Filtros Espaciales. 37
2.4.1 Filtro Gaussiano 38
2.4.2 Filtro de Promedio Circular. 40
2.5 Operador Sobel 43
2.6 Envolvente Convexa 46

3. Procesador Digital de Señales ADSP-BF533 48
3.1 Descripción General [21] 49
3.2 Arquitectura del *ADSP-BF533 EZ-Kit Lite*® 50
3.2.1 Características Generales 50
3.2.2 Procesador ADSP-BF533……..51
3.2.3 Núcleo de Procesador ADSP-BF533. 53
3.2.4 Arquitectura de Memoria. 55

4. Sistema de Reconocimiento de Iris. 57
4.1 Descripción General 58
4.2 Diagrama de Bloques General del Sistema de Reconocimiento de Iris. 58
4.2.1 Registro de Usuario 60
4.2.2 Validación de Usuario. 65
4.3 Segmentado del Iris 67

4.4 Normalización del Iris 72
4.5 Análisis de Identidad....74
4.6 Cálculo Umbral Distancia de Hamming (UmbralH)....77
4.7 Base de Datos de Pruebas CASIA Iris Database V4.0....79

5. Implementación en MatLab®....84
5.1 Descripción General....85
5.2 Registro De Usuario....85
5.3 Validación Usuario....98

6. Implementacion en el ADSP-BF533 EZ-KIT Lite®....104
6.1 Administración de Memoria....105
6.2 Implementación del Sistema de Reconocimiento de Iris....109
6.2.1 Registro Usuario....109
6.2.2 Validación....121

7. Resultados....124
7.1 Metodología de las Pruebas....125
7.2 Resultados Pruebas del Sistema de Reconocimiento de Iris Implementado en MatLab®....127
7.3 Resultados Pruebas del Sistema de Reconocimiento de Iris Implementado en el ADSP-BF533 EZ-Kit Lite®.....130

8. Conclusiones....134

Bibliografía....137

Anexos....140

Índice de Figuras

FIGURA 1. IMAGEN CLÁSICA DE *LENA* USADA PARA EJEMPLOS EN EL PROCESAMIENTO DIGITAL DE IMÁGENES..... 27
FIGURA 2. MODELO DE COLOR RGB..... 28
FIGURA 3. COMPARACIÓN DE LOS MODELOS DE COLOR CMYK Y RGB..... 28
FIGURA 4. DIAGRAMA DE BLOQUES DE LA TRANSFORMADA DE WAVELET DE 4° NIVEL. DONDE S_{J-N} REPRESENTAN LA SEÑAL PROMEDIO Y D_{J-N} REPRESENTAN LA SEÑAL DE DETALLE..... 31
FIGURA 5. SEÑAL ORIGINAL. 32
FIGURA 6. SEÑAL ALTERADA. 33
FIGURA 7. TRANSFORMADA WAVELET DE HAAR DE LA SEÑAL ORIGINAL. PRIMERA GRÁFICA. SEÑAL ORIGINAL. SEGUNDA GRAFICA. APROXIMACIÓN DE PRIMER NIVEL. TERCERA GRÁFICA. DETALLES DE PRIMER NIVEL. 33
FIGURA 8. TRANSFORMADA WAVELET DE HAAR DE LA SEÑAL ALTERADA. PRIMERA GRÁFICA. SEÑAL ALTERADA. SEGUNDA GRAFICA. APROXIMACIÓN DE PRIMER NIVEL. TERCERA GRÁFICA. DETALLES DE PRIMER NIVEL. 34
FIGURA 9. TRANSFORMADA DE FOURIER DE LA SEÑAL ORIGINAL. 34
FIGURA 10. TRANSFORMADA DE FOURIER DE LA SEÑAL ALTERADA..... 35
FIGURA 11. DIFERENCIA ENTRE LA TRANSFORMADA DE WAVELET DE LA SEÑAL ORIGINAL Y LA TRANSFORMADA DE WAVELET DE LA SEÑAL ALTERADA. 35
FIGURA 12. DIFERENCIA ENTRE LA TRANSFORMADA DE FOURIER DE LA SEÑAL ORIGINAL Y LA TRANSFORMADA DE WAVELET DE LA SEÑAL ALTERADA. 36
FIGURA 13. DIAGRAMA DE BLOQUES DE LA TRANSFORMADA DE WAVELET 2D..... 37
FIGURA 14. TRANSFORMADA DE WAVELET DE PRIMER NIVEL DE UNA IMAGEN. SUP. IZQ. IMAGEN SUB-MUESTREADA. SUP. DER. DERIVADA VERTICAL INF IZQ. DERIVADA HORIZONTAL INF DER. GRADIENTE. 37
FIGURA 15. REPRESENTACIÓN ESPACIAL DEL FILTRO GAUSSIANO PREDETERMINADO DE MATLAB®. 39
FIGURA 16. IMAGEN ORIGINAL ANTES DE SER FILTRADA. 39
FIGURA 17. IMAGEN FILTRADA POR EL FILTRO GAUSSIANO PREDETERMINADO DE MATLAB®. 40
FIGURA 18. PSF DEL FILTRO DE PROMEDIO CIRCULAR..... 41
FIGURA 19. REPRESENTACIÓN ESPACIAL DEL FILTRO GAUSSIANO PREDETERMINADO DE MATLAB®. 42
FIGURA 20. IMAGEN ORIGINAL ANTES DE SER FILTRADA. 42
FIGURA 21. IMAGEN FILTRADA POR EL FILTRO DE PROMEDIO CIRCULAR PREDETERMINADO DE MATLAB®..... 43

FIGURA 22. IMAGEN ORIGINAL. 44
FIGURA 23. DERIVADA HORIZONTAL Gx. 45
FIGURA 24. DERIVADA HORIZONTAL Gy. 45
FIGURA 25. MAGNITUD DEL GRADIENTE. 46
FIGURA 26. DIRECCIÓN DEL GRADIENTE. 46
FIGURA 27. ENVOLVENTE CONVEXA DE UN GRUPO DE PUNTOS 47
FIGURA 28. TARJETA DE DESARROLLO *ADSP-BF533 EZ-KIT LITE*® DE ANALOG DEVICES. 49
FIGURA 29. ARQUITECTURA DEL ADSP-BF533 EZ-KIT LITE®. 52
FIGURA 30. DIAGRAMA DE BLOQUE DEL ADSP-BF533. 53
FIGURA 31. DIAGRAMA DE BLOQUES DEL NÚCLEO DEL ADSP-BF533 54
FIGURA 32. ARQUITECTURA DE MEMORIA DEL PROCESADOR ADSP-BF533. 56
FIGURA 33. DIAGRAMA DE FLUJO GENERAL DEL SISTEMA DE RECONOCIMIENTO DE IRIS. 59
FIGURA 34. DIAGRAMA DE FLUJO DE LA IMPLEMENTACIÓN DE LA TRANSFORMADA WAVELET DE HAAR 2D. 62
FIGURA 35. HISTOGRAMA 95 VECTORES DIFERENTES DE COEFICIENTES DE LA TRANSFORMADA WAVELET DE HAAR 2D. 63
FIGURA 36. DIAGRAMA DE FLUJO MÓDULO "SEGMENTADO DEL IRIS". 68
FIGURA 37. GRÁFICA DE LA CAMPANA DEL FILTRO DE PROMEDIO CIRCULAR IMPLEMENTADO (PILLBOX). 70
FIGURA 38. BÚSQUEDA DE LA PUPILA. 71
FIGURA 39. ALGORITMO DE BÚSQUEDA DEL BORDE INFERIOR DEL IRIS. 73
FIGURA 40. CAMBIO DE COORDENADAS POLARES A COORDENADAS RECTANGULARES DE LA REGIÓN DEL IRIS. DONDE N Y M ES EL TAMAÑO DE LA IMAGEN DE SALIDA. 73
FIGURA 41. DIAGRAMA DE FLUJO DEL MÓDULO DE NORMALIZACIÓN 75
FIGURA 42. DIAGRAMA DE FLUJO DEL MÓDULO DE ANÁLISIS DE IDENTIDAD. 76
FIGURA 43. HISTOGRAMA DE LAS DISTANCIAS DE HAMMING PARA LAS PRUEBAS DE USUARIOS REGISTRADOS EN EL SISTEMA (VERDE) Y DE LOS USUARIOS NO REGISTRADOS EN EL SISTEMA (AZUL). 78
FIGURA 44. EJEMPLO IMAGEN CASIA IRIS INTERVAL. 80
FIGURA 45. EJEMPLO IMAGEN CASIA IRIS LAMP.. 80
FIGURA 46. EJEMPLO IMAGEN CASIA IRIS TWINS.. 80
FIGURA 47. EJEMPLO IMAGEN CASIA IRIS DISTANCE. 81
FIGURA 48. EJEMPLO IMAGEN CASIA IRIS THOUSAND.. 81
FIGURA 49. EJEMPLO IMAGEN CASIA IRIS SYN.. 82
FIGURA 50. IMAGEN ORIGINAL CARGADA DE LA BASE DE DATOS CASIA IRIS DATABASE 4.0. 86
FIGURA 51. IMAGEN DESPUÉS DEL RECORTE 1. 87

FIGURA 52. IMAGEN FILTRADA POR FILTRO DE PROMEDIO CIRCULAR 1 DEL BANCO DE FILTROS 1. 87
FIGURA 53. IMAGEN FILTRADA POR FILTRO DE PROMEDIO CIRCULAR 2 DEL BANCO DE FILTROS 1. 88
FIGURA 54. IMAGEN DESPUÉS DEL DETECTOR DE BORDES. 88
FIGURA 55. IMAGEN DE LA SEGMENTACIÓN DE LA PUPILA. 89
FIGURA 56. IMAGEN ORIGINAL DESPUÉS DEL RECORTE 2. 90
FIGURA 57. IMAGEN FILTRADA POR FILTRO DE PROMEDIO CIRCULAR 1 DEL BANCO DE FILTROS 2. 90
FIGURA 58. IMAGEN FILTRADA POR FILTRO DE PROMEDIO CIRCULAR 2 DEL BANCO DE FILTROS 2. 91
FIGURA 59. IMAGEN DESPUÉS DEL DETECTOR DE BORDES 2. 91
FIGURA 60. IMAGEN DE LA SEGMENTACIÓN DEL IRIS FINAL. 92
FIGURA 61. IMAGEN NORMALIZADA DEL IRIS. 93
FIGURA 62. IMAGEN NORMALIZADA DESPUÉS DEL RECORTE 3. 94
FIGURA 63. TRANSFORMADA WAVELET DE HAAR DE 4° NIVEL. 94
FIGURA 64. COEFICIENTES DE 4° NIVEL LA TRANSFORMADA WAVELET DE HAAR 2D. 95
FIGURA 65. HISTOGRAMA DE LOS COEFICIENTES DE LA TRANSFORMADA WAVELET DE HAAR 2D DE UNA DE LAS 19 IMÁGENES DEL USUARIO A REGISTRAR. 96
FIGURA 66. HISTOGRAMA DE LOS COEFICIENTES CUANTIFICADOS DE LA TRANSFORMADA WAVELET DE HAAR 2D DE UNA DE LAS 19 IMÁGENES DEL USUARIO A REGISTRAR. ..96
FIGURA 67. IMAGEN DEL IRIS DEL USUARIO A VALIDAR. 98
FIGURA 68. SEGMENTACIÓN Y NORMALIZACIÓN DEL IRIS DEL USUARIO A VALIDAR. 99
FIGURA 69. TRANSFORMADA WAVELET DE HAAR 2D DEL USUARIO A VALIDAR. 99
FIGURA 70. COEFICIENTES DE 4° NIVEL DE LA TRANSFORMADA WAVELET DE HAAR 2D DEL USUARIO A VALIDAR. 100
FIGURA 71. HISTOGRAMA DE LOS COEFICIENTES DE LA TRANSFORMADA WAVELET DE HAAR 2D DEL USUARIO DE PRUEBA. 101
FIGURA 72.HISTOGRAMA DE LOS COEFICIENTES CUANTIFICADOS DE LA TRANSFORMADA WAVELET DE HAAR 2D DEL USUARIO DE PRUEBA. 101
FIGURA 73. ENTORNO DE DESARROLLO VISUALDSP++ 5.0. 108
FIGURA 74. CARGA DE LA IMAGEN DEL USUARIO A REGISTRAR AL DSP USANDO EL IMAGEVIEWER DEL VISUALDSP++. 110
FIGURA 75. IMAGEN DEL IRIS CARGADA AL DSP USANDO EL IMAGEVIEWER DEL VISUALDSP++®. 110
FIGURA 76. IMAGEN RESULTANTE DESPUÉS DEL RECORTE 1. 111
FIGURA 77. IMAGEN RESULTANTE DESPUÉS DEL PRIMER FILTRO PILLBOX DEL PRIMER BANCO DE FILTROS. 112

FIGURA 78. IMAGEN RESULTANTE DESPUÉS DEL SEGUNDO FILTRO PILLBOX DEL PRIMER BANCO DE FILTROS. 113
FIGURA 79. IMAGEN RESULTANTE DEL DETECTOR DE BORDES PARA EL SEGMENTADO DE LA PUPILA. 114
FIGURA 88. GRÁFICO DE LAS FALSAS ACEPTACIONES DEL SISTEMA IMPLEMENTADO EN MATLAB® DISCRIMINADO POR MÉTODOS DE ANÁLISIS DE IDENTIDAD DISTANCIA DE HAMMING (AZUL) Y DISTANCIA EUCLIDIANA (NARANJA) Y RESULTADO FINAL DEL SISTEMA (GRIS). 128
FIGURA 89. GRÁFICO DE LOS FALSOS RECHAZOS DEL SISTEMA IMPLEMENTADO EN MATLAB® DISCRIMINADO POR MÉTODOS DE ANÁLISIS DE IDENTIDAD DISTANCIA DE HAMMING (AZUL) Y DISTANCIA EUCLIDIANA (NARANJA) Y RESULTADO FINAL DEL SISTEMA (GRIS). 130
FIGURA 90. GRÁFICO DE LOS FALSOS RECHAZOS DEL SISTEMA IMPLEMENTADO EN EL ADSP-BF533 EZ-KIT LITE®, DISCRIMINADO POR MÉTODOS DE ANÁLISIS DE IDENTIDAD DE DISTANCIA DE HAMMING (AZUL) Y DISTANCIA EUCLIDIANA (NARANJA) Y RESULTADO FINAL DEL SISTEMA (GRIS). 131
FIGURA 91. GRÁFICO DE LAS FALSAS ACEPTACIONES DEL SISTEMA IMPLEMENTADO EN EL ADSP-BF533 EZ-KIT LITE® DISCRIMINADO POR MÉTODOS DE ANÁLISIS DE IDENTIDAD DISTANCIA DE HAMMING (AZUL) Y DISTANCIA EUCLIDIANA (NARANJA) Y RESULTADO FINAL DEL SISTEMA (GRIS). 133

Índice de Tablas

TABLA 1. MÁSCARA DEL FILTRO GAUSSIANO PREDETERMINADO DE MATLAB®. TAMAÑO 3X3. 38
TABLA 2. MÁSCARA DEL FILTRO DE PROMEDIO CIRCULAR PREDETERMINADO DE MATLAB®. TAMAÑO 11X11. 41
TABLA 3. MÁSCARA DEL FILTRO DE PROMEDIO CIRCULAR. 69
TABLA 4. RESULTADOS DEL ANÁLISIS DE CONFIABILIDAD PARA LOS UMBRALES EXTRAÍDOS DEL HISTOGRAMA DE DISTANCIAS DE HAMMING. 79
TABLA 5. CARACTERÍSTICAS TÉCNICAS DE LA BASE DE DATOS CASIA IRIS DATABASE V4.0 DE BIT. 83
TABLA 6. EJEMPLO DEL MAPA DE MEMORIA USADA DEL SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN DSP. 107

Índice de Anexos

ANEXO 1. TABLA DE MAPA DE MEMORIA USADA 141
ANEXO 2. REGISTRO_DB.C 142
ANEXO 3. CARACTERIZACION.C 147
ANEXO 4. CIRCULO.C 154
ANEXO 5. CIRCUNCENTRO.C 162
ANEXO 6. ENCONTRARCENTROPUPILA.C 166
ANEXO 7. ENCONTRARIRIS.C 172
ANEXO 8. ESTADISTICA.C 176
ANEXO 9. ETIQUETAS.H 183
ANEXO 10. GENERALES.C 188
ANEXO 11. GESTIONDEARCHIVOS.C 194
ANEXO 12. GRADIENTE.C 201
ANEXO 13. INITIALIZATION.C 206
ANEXO 14. NORMALIZAR.C 210
ANEXO 15. PILLBOX.C 218
ANEXO 16. SLAVE_GLOBAL.H 223
ANEXO 17. WAVELETHAAR2D.C 224
ANEXO 18. VALIDACION.C 232
ANEXO 19. VERIFICACION.C 237

1
PRELIMINARES

1.1 Definición del Problema.

Los sistemas de seguridad para el control de acceso a lugares restringidos se han convertido en estos últimos años en un tema de bastante importancia, no solo para las grandes empresas, sino también para la gente del común, que quiere mantener a salvo y bajo su protección sus bienes más preciados de manos inescrupulosas.

Durante toda la historia de la humanidad se han desarrollado distintos mecanismos para mantener protegidos estos lugares de accesos no autorizados, utilizando desde un simple alambre para cerrar el lugar, pasando por las muy populares cerraduras que utilizan llaves o claves mecánicas, llegando hasta las actuales cerraduras que ofrecen control de acceso usando tarjetas identificación por radiofrecuencia, o utilizando contraseñas con diferentes métodos y técnicas de cifrado digital.

Los últimos sistemas en el mercado se basan en las nuevas técnicas de biometría, que utilizan el procesamiento de imágenes en conjunto con análisis matemáticos y estadísticos para el reconocimiento de diferentes patrones del cuerpo humano como la huella dactilar o rasgos faciales de cada persona para validar su identidad.

Uno de los sistemas biométricos más usados actualmente es la huella dactilar, en la cual se encontró que en [1], se presentó un diseño completo de un sistema de adquisición y de reconocimiento de huellas dactilares.

Recientemente y gracias a los buenos resultados obtenidos con técnicas de biometría como las mencionadas anteriormente, se ha impulsado el desarrollo de diferentes procesos, métodos y algoritmos en diferentes plataformas para el reconocimiento de iris; en el cual se pueden destacar algunas como:

En el campo del desarrollo de software e implementación en sistemas embebidos, se halló que en [2] presentaron un algoritmo de reconocimiento de iris usando un procesador digital de señales (DSP "*Digital Signal Processor*"), y utilizando la transformada rápida de Fourier para imágenes 2D. Además realizaron pruebas en diferentes bases de datos especializadas como el CASIA Iris Database y el Iris Challenge Evaluation, arrojando buenos resultados en el algoritmo de reconocimiento. Utilizando también un DSP, en [3] presentaron un algoritmo de reconocimiento de iris basado en extraer las características de textura del iris, obteniendo así, una alta precisión en la correcta identificación del iris. Por otro lado, y usando arreglos de compuertas lógicas programables (FPGA "*Field Programmable Gate Array"*), se encontró que en [4] desarrollaron un procesador para el reconocimiento biométrico de iris, optimizando los tiempos de procesamiento y confiabilidad del sistema.

En la parte de desarrollo de software para reconocimiento de iris, se encuentra que en el artículo [5] se presenta un algoritmo que consiste en digitalizar la imagen, pre procesarla y finalmente codificarla; conjuntamente propusieron un algoritmo para la localización del iris y un nuevo método de implementación de este.

Por último, se halló una patente [6] en la cual se propone un sistema de seguridad de autenticación por iris para cerraduras de puertas utilizando principalmente un DSP y una serie de módulos acoplados a ella.

Con el desarrollo de este proyecto, se podría ofrecer una solución de alta tecnología y fiabilidad al problema de reconocimiento de identidad, presente en el campo del control de acceso; utilizando un sistema embebido y de funcionamiento en tiempo real como lo es un DSP, además como beneficio secundario, se podrían fijar las bases del desarrollo de algoritmos y métodos de reconocimiento de iris implementados sobre este tipo de hardware; inclusive, se crearía la oportunidad para el desarrollo de un prototipo funcional de control de acceso biométrico utilizando lo aquí planteado.

1.2 Justificación.

El desarrollo de este sistema ofrecería utilizar las herramientas existentes en reconocimiento de iris y procesamiento digital de imágenes en crear un sistema que ofrezca una solución a la creciente necesidad, tanto de las empresas como de la gente del común, de dispositivos de control de acceso mucho más confiables, seguros y eficientes que los disponibles actualmente.

El desarrollo e implementación de este sistema, podría incentivar el campo de la biometría aplicada en el país, y no únicamente en el área de reconocimiento de iris, sino también en campos como las huellas dactilares, geometría de la palma de la mano o patrones faciales. También puede impulsar nuevos proyectos que apliquen sistemas de reconocimiento de iris para otras funciones como patrón en registros de usuarios en grandes bases de datos y verificación de estos.

La creación de este sistema podría ser un punto de referencia para el desarrollo de futuro sistema completo de una cerradura biométrica comercial basada en los parámetros aquí mencionados; incluso, puede ser tomado como producto base de un portafolio de productos y servicios para una nueva empresa de seguridad biométrica.

La implementación de este sistema embebido como lo es un DSP daría ventajas significativas sobre otras plataformas de desarrollo como lo son los computadores personales y los micros controladores, ya que un DSP tiene superioridades significativas en tamaño, portabilidad, lenguaje de programación y velocidad de procesamiento de imágenes comparado con estas otras plataformas. Además un DSP ofrecería la posibilidad de realizar un procesamiento de la aplicación totalmente autónomo y en tiempo real.

1.3 Objetivos.

1.3.1 Objetivo General.

Desarrollar un sistema de reconocimiento de iris usando un DSP ADSP-BF533 de Analog Devices.

1.3.2 Objetivos Específicos.

1. Desarrollar y simular en MatLab® un software de reconocimiento de iris que realice el filtrado, segmentación del iris, verificación y la validación de la identidad.

2. Implementar y probar el software desarrollado en el DSP ADSP-BF533.

3. Realizar pruebas estadísticas de confiabilidad del sistema.

1.4 Marco de Antecedentes.

Los sistemas de reconocimiento de iris han sido tratados desde hace ya bastante tiempo; pero es en los últimos años que han venido cobrando importancia en el campo de la seguridad, ya que ofrecen altos niveles de confiabilidad a la hora de autenticar la identidad de un usuario.

Como es mencionado en [7] *"la primera implementación de sistemas de control de acceso basadas en biometría de iris fue basada en la patente de J. Daugman's U.S Patent 5 291 560…"* De este modo se puede deducir que J Daugman's se puede considerar como el pionero en este campo, y que todos los desarrollos actuales de este tipo de sistemas y algoritmos están basado en su trabajo.

Actualmente existen diferentes variantes de los algoritmos de reconocimiento de iris, los cuales son desarrollados en diferentes plataformas, desde las tradicionales arquitecturas de computadores personales (PC, "*Personal Computer*"), hasta los actuales procesadores de varios núcleos [7].

En esta sección, se presentarán los artículos más relevantes encontrados en referencia a sistemas de reconocimiento de iris implementados en DSP y FPGA.

- **Estructuras Básicas de los Sistemas y Algoritmo de Reconocimiento de Iris.**

En la parte de software, en [2] se presenta un algoritmo de reconocimiento de iris el cual tiene dos etapas principales. La primera es la localización del iris en la imagen, en la cual se realiza primero la normalización de la imagen, luego la eliminación de las pestañas y párpados, después se localiza el iris y por último, se mejora el contraste de la imagen.

La segunda etapa, la cual es de comparación, se realiza inicialmente la extracción de la región de interés, después la alineación de la imagen, seguidamente el cálculo de la correlación entre las imágenes, luego se realiza el análisis del umbral para la aceptación o rechazo de la imagen y por último se entrega el resultado final de la comparación.

En [3] se presenta un algoritmo implementado en un DSP para el reconocimiento de iris usando las características de textura del iris. Este algoritmo inicialmente localiza los límites superiores e inferiores del iris en la imagen a analizar, luego elimina el ruido, las pestañas y los párpados que interfieren con el iris, a continuación normaliza la imagen, más tarde extrae las características de textura del iris usando Filtro Logarítmico de Gabor, después compara las características extraídas anteriormente con una plantilla predefinida usando "Hamming Distance" y finalmente evalúa el resultado de la comparación.

En los sistemas embebidos como la FPGA, se encuentra que [4] presenta un sistema de reconocimiento de iris que realiza inicialmente la adquisición de la

imagen mediante una cámara de alta definición (HD,"High Definition") con LED's infrarrojos, después la extracción del iris de la imagen, seguidamente se realiza el mejoramiento de la imagen del iris extraída mediante filtros digitales, luego se procede a la transformación de la imagen utilizando filtros de Gabor a un vector binario que representa las características del iris para luego realizar la comparación del vector de características de la plantilla con el vector creado del iris a verificar y, finalmente hacer el análisis del resultado de la diferencia entre los vectores para la autenticación del usuario.

En [5] se presenta una arquitectura para un sistema de reconocimiento de iris basado en DSP realizando primero la adquisición de la imagen mediante una cámara con sensor de carga acoplada (CCD*,"Charge-Coupled Device*") con diodos emisores de luz (LED, *"Light Emitting Diode*") infrarrojos, luego la digitalización de la imagen usando un SAA7115H Video Process Chip, seguidamente se realiza el pre-procesamiento, codificación y reconocimiento de la imagen en un DSP, para luego hacer la conexión entre la DSP y el computador, para así, terminar el proceso de verificación de la imagen.

En la patente descrita en [6] se describe el funcionamiento de una cerradura controlada por biometría de iris, la cual fue implementada en un DSP. Este desarrollo inicialmente realiza la creación del código de iris a comparar con el iris del usuario a autenticar, más tarde procede a la comparación entre el iris del usuario y el iris almacenado en la base de datos para finalmente, realizar la autenticación del iris del usuario autorizando o no, la apertura de la cerradura.

- **Comparación entre los Sistemas y Algoritmos.**

Plataformas de Implementación.

Generalmente este tipo de aplicaciones se desarrollan en computadores de propósito general, [7] pero con los recientes avances en tecnología de sistemas embebidos como DSP's y FPGA's; los cuales ofrecen mejor rendimiento para este tipo de tareas de procesamiento de imágenes; las implementaciones aquí mencionadas de los sistemas y algoritmos de reconocimiento de iris se han venido realizado en este tipo de dispositivos; así que se tiene que, [2], [3] y [5] desarrollaron su sistema en Procesadores Digitales de Señales (DSP), mientras que [4] fue implementado en una FPGA.

Adquisición de las Imágenes.

Se encontró que entre los diferentes sistemas y algoritmos de biometría de iris, la manera como obtienen las imágenes son básicamente dos. Por un lado, los sistemas [4], [5] y [6] adquirieron sus imágenes usando cámaras con sensores CCD, mientras que los algoritmos [2] y [3], se limitaron a utilizar las bases de datos *"CASIA Iris Imagen Data Base 1.0*", *"CASIA Iris Imagen Data Base 2.0"* y la *"ICE 2005 Database"* para su pruebas.

Procesamiento de la Imagen.

Existen diferentes técnicas y métodos para los diferentes procesos que debe sufrir una imagen antes de su comparación con la plantilla.

Segmentación del Iris.

Se encuentra que lo primero a realizar es hallar los límites del iris (borde entre la pupila y el iris, y la esclerótica y el iris).

En [5] para hallar los límites del iris realizan el siguiente proceso: *"…el límite inferior es localizado mediante la umbralización de las imágenes y la cadena de código de Freeman. El Operador Canny y la Transformada circular de Hough es usada para localizar el límite superior…*".

En [2] primero pasan la imagen a escala de grises, luego "…*el centro de la pupila es estimado como el centro de gravedad de la imagen binaria…*", y luego mediante el cálculo de elipses, usando parámetros de estimación, los limites inferiores y superiores del iris son hallados.

En [4] y [6] se propone un bloque funcional de segmentación del iris donde se hallaban los bordes superiores e inferiores del iris de la imagen a procesar.

Y en [3] se utiliza un algoritmo donde, primero la imagen es digitalizada y luego umbralizada, un "Operador Morfológico" es aplicado para eliminar el brillo e interferencias de las pestañas y párpados. Después de esto, es aplicado un filtro gaussiano, el cual genera una serie de círculos en la imagen binaria, los cuales son usados para hallar el límite inferior del iris. Este mismo procedimiento es aplicado para hallar el límite superior del iris.

Métodos de Análisis y Comparación del Iris.

Después de realizar la extracción del iris de la imagen original, los sistemas y algoritmos proceden a realizar una serie de operaciones especializadas con la imagen del iris para realiza la autenticación de este. A continuación se mostrarán los diferentes métodos y técnicas usadas.

En [2] se presenta el diseño de un algoritmo para el reconocimiento de iris usando solo los componentes de la fase de la transformada de Fourier Discreta en dos dimensiones de una imagen, además propone la codificación de la base de datos de las imágenes de los iris, usando solo las componentes de fase de la Transformada de Fourier Discreta en 2D de la imagen procesada "*2D Fourier Phase Code (FPC)"*; esto con el fin de reducir el tamaño de la base de datos y mejorar el nivel de seguridad de la misma.

En [3], mediante la realización de filtros de Gabor, se extrae la información frecuencial que representa las características de textura de la imagen. Este tipo de

extracción de características (textura) no depende de factores como la iluminación y contrastes en la imagen.

Para la comparación de las características obtenidas con las características de la plantilla, se utiliza igualmente el proceso de distancia de Hamming, pero en este caso se incluyen plantillas de enmascaramiento en el proceso de comparación para filtrar zonas de ruido en el iris.

En [4] el algoritmo inicialmente, realiza un cambio de coordenadas en la imagen, llevando esta a coordenadas polares, luego segmenta la imagen, cada una de estas es filtrada mediante filtro de Gabor en direcciones 0, π/4, π/2 y 3π/4. Después de esto cada sector es integrado y es comparado con un umbral determinado empíricamente y dependiendo de cada una de estas comparaciones se generan vectores binarios que corresponden al vector de características del iris procesado. La comparación entre los dos vectores se realiza mediante el proceso de distancias de Hamming, el cual consiste en realizar la sumatoria de las operaciones XOR de las componentes de los dos vectores uno a uno. Este valor será el evaluado según el criterio umbral para verificar la autenticidad del iris.

El método usado en [5] fue propuesto pensando en la simplicidad, esto es, el no uso de operadores complejos para el análisis de la imagen, como es la transformada de Hough; a cambio de esto, propone una técnica bastante sencilla pero efectiva de análisis de la imagen por el histograma en escala de grises; donde se realiza un algoritmo que recorre la imagen y analiza los valores binarios de cada uno de los pixeles.

El sistema presentado en [6] no describe a profundidad el método de verificación, pero aclara que recibe la imagen del iris, la analiza y da un resultado válido o no

2
Procesamiento Digital de Imágenes

2.1 Concepto de Imagen

Una imagen puede tener diferentes interpretaciones y/o definiciones dependiendo del campo desde el cual se esté analizando, esto se puede evidencia ya que se puede encontrar una definición desde el campo de la neurociencia, pasando por las matemáticas hasta llegar a su definición filosófica. Sin embargo para efectos de este proyecto, se tratará la imagen desde el punto de vista de la electrónica y del procesamiento digital de imágenes, de este modo la definición más general de una imagen se toma como: "*conjunto de datos bidimensionales de valores finitos, los cuales corresponden a una aproximación de la representación visual de un objeto ya sea real o imaginario".*

Más específicamente, y teniendo en cuenta que las imágenes del iris fueron extraídos de la base de datos CASIA Iris Database 4.0 fueron tomadas por una cámara digital, se puede definir una imagen como *"conjunto de valores digitales bidimensionales resultantes de la conversión análogo/digital de una señal del bidimensional de frecuencia e intensidad lumínica, los cuales una son aproximación de la representación visual de un objeto real".* En la Figura 1 se puede observar la imagen digital de un iris.

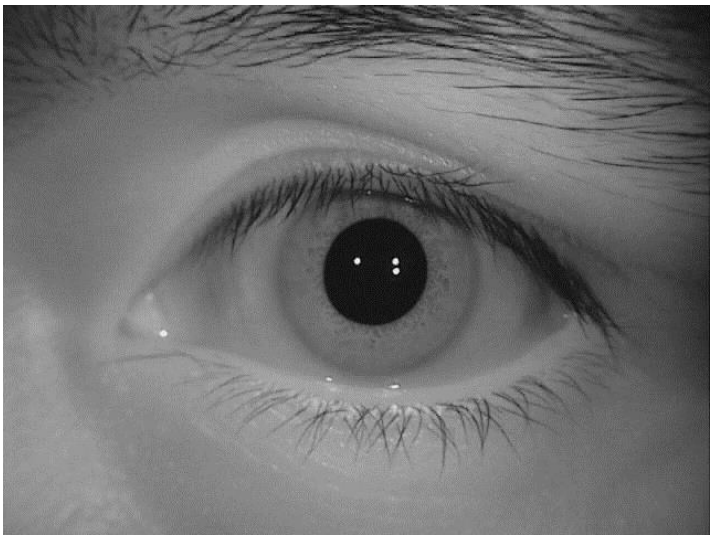

Figura 1. Imagen de un Iris de la Base de Datos Iris Casia Database V4.0.

2.2 Espacio de Color

Para realizar una descripción lo más sencilla pero completa posible acerca de espacio de color se definirán unos conceptos algo básicos pero muy necesarios.

2.2.1 Modelo de Color

Un modelo de color se puede definir como modelo matemático, el cual, haciendo uso de varios colores primarios (primarias) permite la representación numérica del color. Un ejemplo de modelo de color es el RGB visto en la Figura 2.

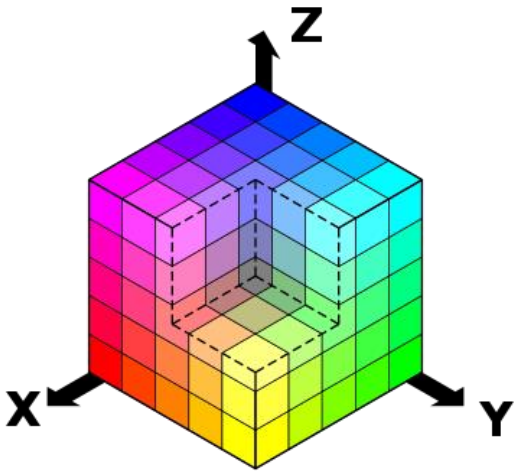

Figura 2. Modelo de Color RGB. [9]

2.2.2 Espacio de Color de Referencia

Es aquel espacio de color que mejor representan la gama de colores que el ser humano puede ver, de este modo los espacios de color de referencia son CIELAB o CIEXYZ; comparados ambos en la Figura 3.

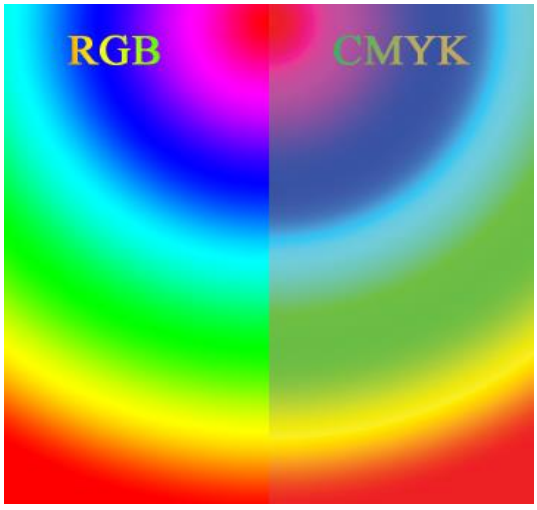

Figura 3. Comparación de los modelos de Color CMYK y RGB. [10].

De esta manera, teniendo un espacio de color de referencia y un modelo de color, se puede generar una relación entre ambos mediante una función de mapeo llamada *gama de color*. Así, con esta *gama de color* y el modelo de color, se puede definir un nuevo espacio de color. De forma más sencilla, se puede describir el espacio de color como un modelo matemático, creado a partir de un modelo de color y la función de mapeo Gama de Color. En la Figura 3 se puede observar la comparación entre los espacios de color RGB y CMYK.

2.3 Transformada Wavelet

2.3.1 Introducción

La Transformada de Wavelet es una poderosa herramienta que al igual que la Transformada de Fourier utilizar un conjunto de funciones ortogonales entre sí para crear una base orto-normal; las cuales son utilizadas para representar una señal. Del mismo modo que la Transformada de Fourier, la Transformada Wavelet tiene sus aplicaciones con señales continuas y señales discretas. En el caso más general se encuentra la Transformada Wavelet Continua, la cual se define matemáticamente como lo indica [11]:

La Transformada Wavelet Continua en se define en la ecuación (1) como:

$$\boldsymbol{W_x(a,b) = \langle x, \psi_{a,b} \rangle = \int_{-\infty}^{\infty} x(t)\bar{\psi}_{a,b}(t)\delta t = \frac{1}{\sqrt{a}}\int_{-\infty}^{\infty} x(t)\bar{\psi}_{a,b}\left(\frac{t-b}{a}\right)\delta t} \quad (1)$$

"Donde el operador $\langle , \rangle$ *significa el producto escalar en* R^2 *y se define como* $\langle f, g \rangle := \int f(t)\bar{g}(t)dt$*; y el símbolo "¯" significa el conjugado complejo. La Transformada Wavelet Continua mide la variación de* ***x*** *en una vecindad de* ***b*** *cuyo tamaño es proporcional* ***a****"* [11] y la Transformada Inversa Wavelet Continua se define en la ecuación (2) como:

$$x(t) = \frac{1}{C_\psi}\int_0^{\infty}\int_{-\infty}^{\infty} W_x(a,b)\psi_{a,b}(t)\frac{\delta a \delta b}{a^2} \quad (2)$$

Donde:
Se tiene la ecuación (3) es la Condición de Admisibilidad:

$$C_\psi = \int_0^{\infty} \frac{\left|\hat{\psi}(w)\right|^2}{|w|}\delta w < \infty \quad (3)$$

Y en la ecuación (4) se tiene la Wavelet de Dilatación-Traslación:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right) \quad (4)$$

Donde b ∈ R y es un parámetro de traslación y a ∈ R^+ (a ≠ 0).

De este mismo modo, existen diferentes versiones de la Transformada Wavelet en el tiempo discreto; siendo algunas de las más conocidas como la Transformada de Haar, la Transformada de Daubechies (Daubechies Wavelets) y la Transformada Wavelet de Doble Árbol Compleja (The Dual-Tree Complex Wavelet Transform, por su nombre en inglés). Sin embargo, siendo este proyecto una aplicación de técnicas de procesamiento digital de imágenes, y en razón de disminuir la complejidad de su implementación en un DSP, se hará énfasis en la Transformada de Wavelet Discreta y en especial la Transformada de Haar.

2.3.2 Transformada de Wavelet de Haar [12]

La Transformada Wavelet de Haar fue desarrollada por el matemático Alfred Haar en el año 1909 y pertenece al conjunto de Wavelet Discretas y es la más simple y sencilla de todas.

Esta transformada funciona con conjuntos de datos de tamaño 2^n a la entrada, para entregar a su salida dos vectores de tamaño $\frac{2^n}{2}$ cada uno. El primer vector es el promedio de la señal original (A^1) y el segundo vector representa los detalles de la señal original (D^1). De este modo se puede observar la Transformada de Haar de primer nivel cuenta con dos vectores, uno de promedio de la señal y otro de detalles de la señal original. En la ecuación (5) se tiene el Primer Nivel de la Transformada de Haar.

$$f = A^1 + D^1 \quad (5)$$

Donde f es la función original de tamaño 2^n.

De este modo en [12] se define que: "*La Transformada de Haar de nivel m es sencillamente una transformación tal que a cada señal f le hace corresponder la señal consistente en los coeficientes de las tendencias de nivel m, junto con las fluctuaciones de nivel m e inferior ordenadas, es decir:*

$$f \xrightarrow{Hm} (a^m | d^m | d^{m-1} | \dots | d^1)“$$

Básicamente lo que realiza esta transformada *es: “una ‘compresión’ de la señal original en la primera mitad, multiplicada por el factor de* $\sqrt{2}$*, mientras que la segunda mitad contiene las fluctuaciones, las cuales serán ‘pequeñas’ si la señal original no tiene cambios ‘bruscos’”.*

También en [12] se menciona que la Transformada Wavelet de Haar es una transformada lineal y ortogonal ya que la matriz de transformación forma una base ortonormal.

En la Figura 4, se puede observar más fácilmente, el diagrama de bloques de una Transformada Wavelet, donde, como se mencionó anteriormente, a la entrada de cada bloque se tiene un conjunto de datos de tamaño 2^n y a la salida se obtienen dos vectores de la mitad del tamaño del vector de entrada.

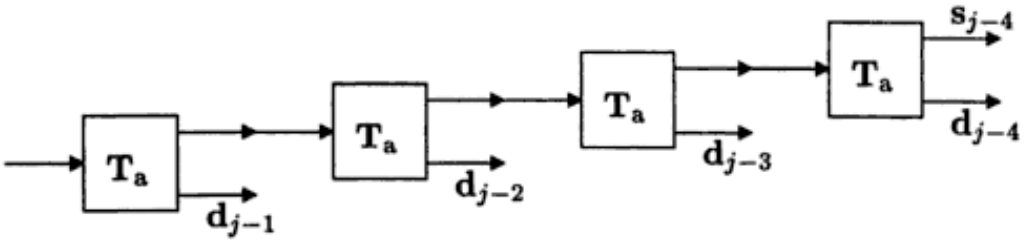

Figura 4. Diagrama de Bloques de la Transformada de Wavelet de 4° nivel. Donde S_{j-n} representan la señal promedio y d_{j-n} representan la señal de detalle. [13].

2.3.3 Transformada de Wavelet vs Transformada de Fourier [14]

Como se menciona en [14], la Transformada de Wavelet representa una serie de ventajas sobre la Transformada de Fourier; siendo una de ellas que la Transformada Wavelet es una transformada local. Para explicar fácil y rápidamente este concepto se utilizará el ejemplo propuesto en [14].

“Supóngase que se tiene un conjunto de 100 datos que representan una señal (Figura 1.5 (a)), de los cuales cuatro valores fueron alterados intencionalmente (Figura 1.5 (b)); posteriormente se le aplica a ambas señales la Transformada de Wavelet, obteniéndose las gráficas de la Figura 1.6 (a) correspondiente a la transformada de la señal original, y la gráfica de la Figura 1.6 (b) correspondiente a

la transformada de la señal alterada. En la Figura 1.7 se representa la diferencia entre las Transformada de Wavelet de la señal original y de la señal alterada.

De este este mismo modo se procedió a aplicar la Transformada Discreta de Fourier a ambas señales, siendo la Figura 1.8(a) la DFT de la señal original y la Figura 1.8 (b) la DFT de la señal alterada. "

Para ofrecer una mejor compresión sobre lo planteado en el ejemplo, se procedió a implementar en MatLab® las gráficas propuestas en [14] y se obtuvieron los siguientes resultados.

En las Figuras 5 y 6 se puede observar la señal original y la señal alterada respectivamente, ambas, de una un tamaño de 100 datos.

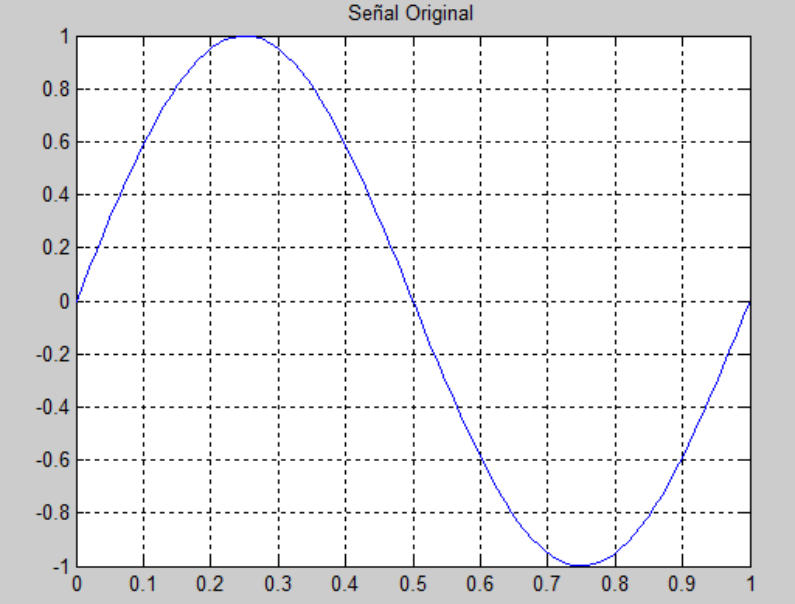

Figura 5. Señal Original.

Posteriormente se procedió a aplicarles a ambas señales la Transformada Wavelet de Haar, obteniendo las Figuras 7 y 8, transformada de la señal original y transformada de la señal alterada respectivamente.

Consecutivamente se procedió a aplicar la Transformada de Fourier tanto a la señal original como a la alterada, obteniéndose las Figuras 9 y 10 respectivamente.

Por último se procedió a realizar la comparación entre las diferencias entre transformadas. El la Figura 11, se puede observar la diferencia entre la Transformada de Wavelet de la señal original y la señal alterada.

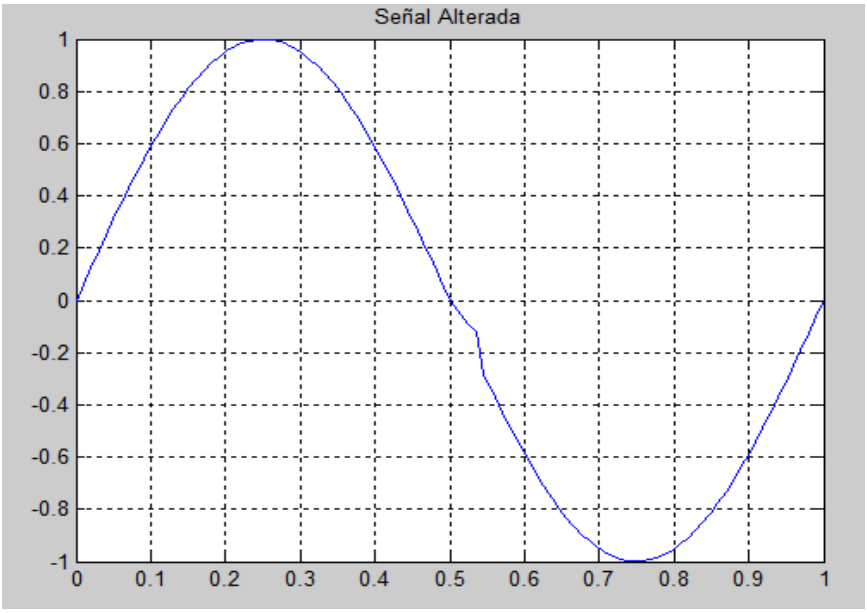

Figura 6. Señal Alterada.

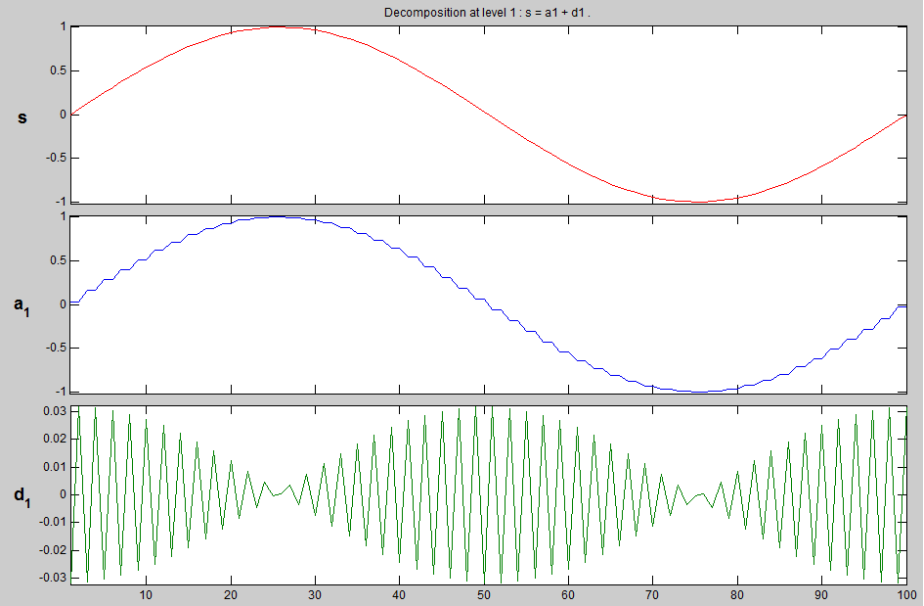

Figura 7. Transformada Wavelet de Haar de la Señal Original. Primera Gráfica. Señal Original. Segunda Grafica. Aproximación de Primer Nivel. Tercera Gráfica. Detalles de Primer Nivel.

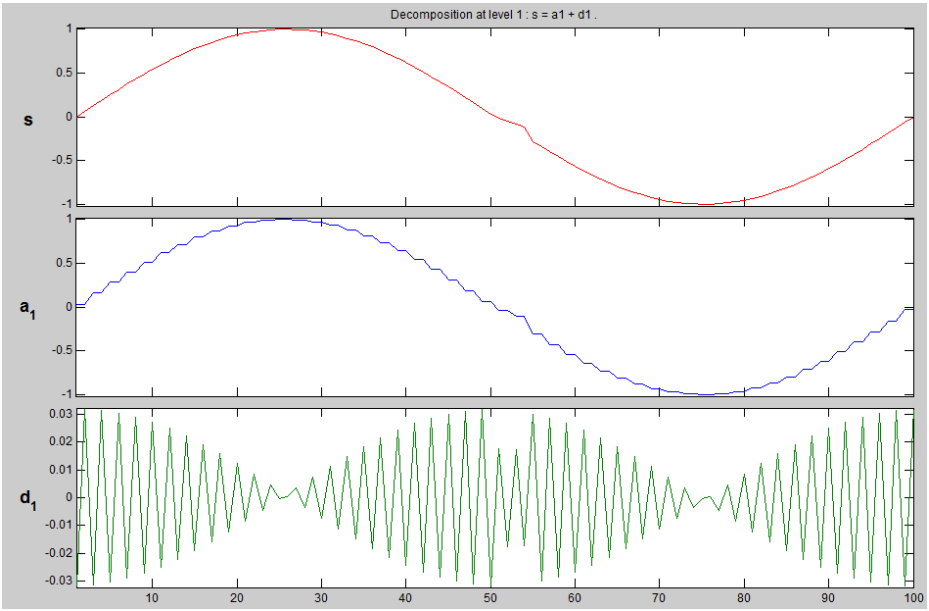

Figura 8. Transformada Wavelet de Haar de la Señal Alterada. Primera Gráfica. Señal Alterada. Segunda Grafica. Aproximación de Primer Nivel. Tercera Gráfica. Detalles de Primer Nivel.

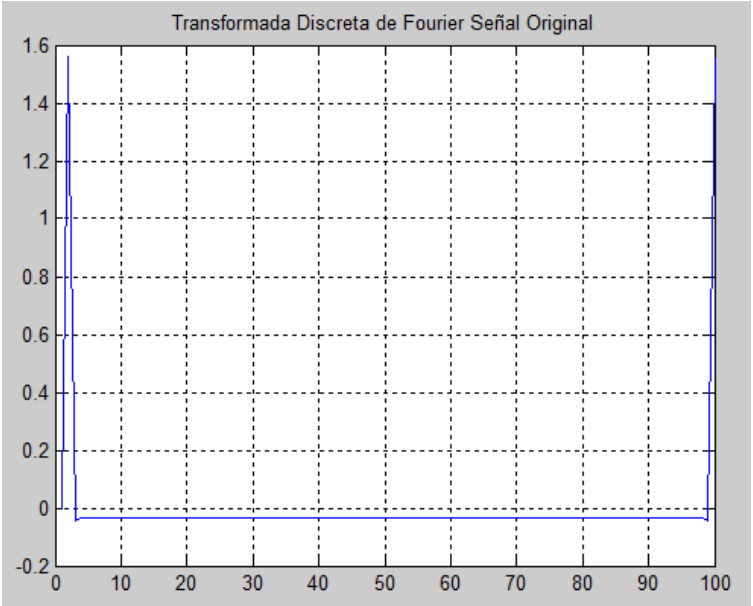

Figura 9. Transformada de Fourier de la Señal Original.

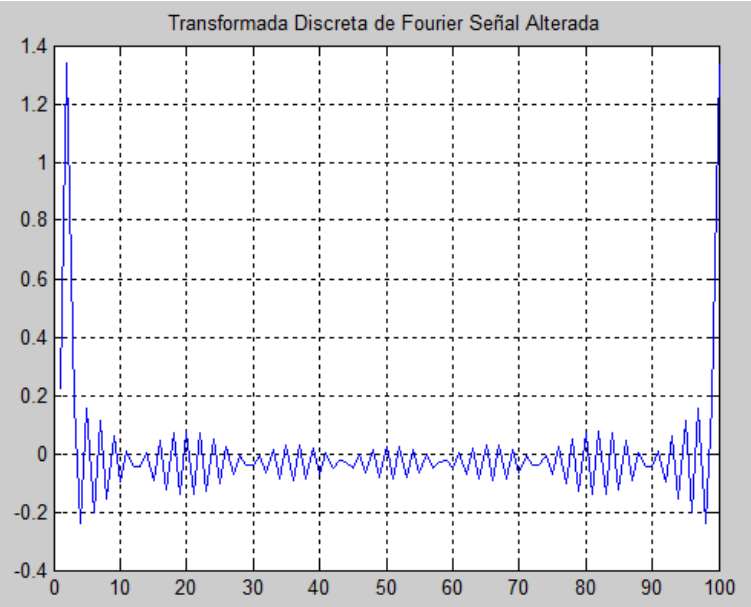

Figura 10. Transformada de Fourier de la Señal Alterada.

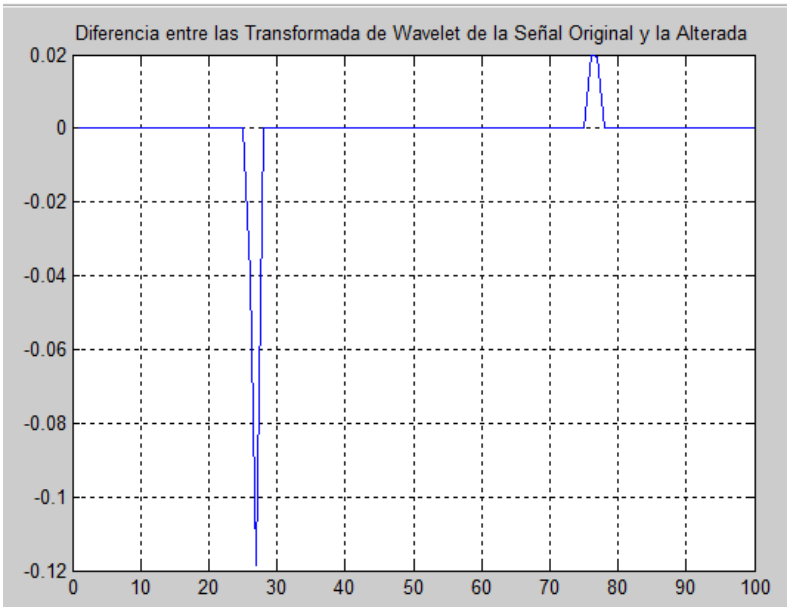

Figura 11. Diferencia entre la Transformada de Wavelet de la Señal Original y la Transformada de Wavelet de la Señal Alterada.

Y en la Figura 12, se puede observar la diferencia entre la Transformada de Fourier de la señal original y la señal alterada.

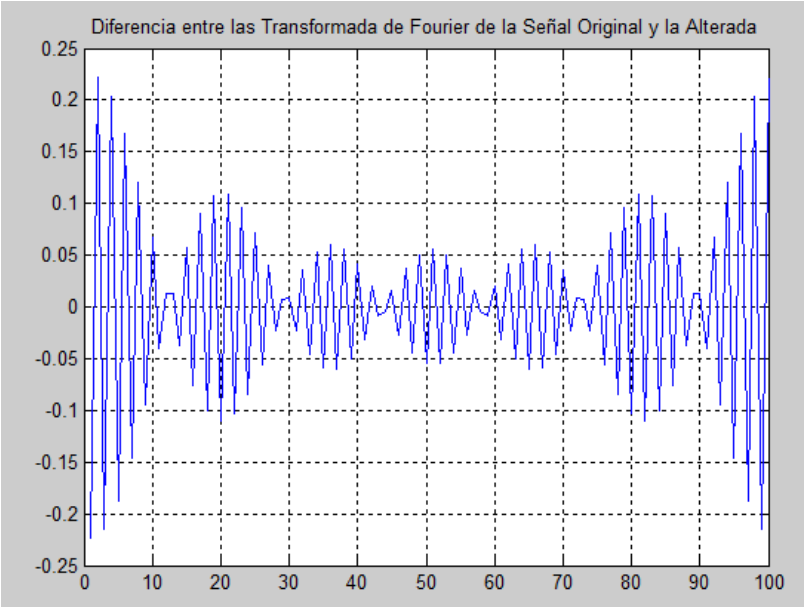

Figura 12. Diferencia entre la Transformada de Fourier de la Señal Original y la Transformada de Wavelet de la Señal Alterada.

Se puede observar evidentemente una menor inmunidad al ruido por parte de la Transformada de Fourier, ya que, como se vio en el ejemplo, solo se cambiaran 4 de 100 datos, que representan el 4% de la información contenida en la señal, su representación en el dominio de la frecuencia varia demasiado, mientras que con la Transformada de Wavelet, este cambio solo se ve reflejado de manera local, justo donde los datos fueron modificados y no en la representación completa de la señal.

2.3.4 Transformada Wavelet de Haar 2D

Al igual que la Transformada Discreta de Fourier en 1D se puede expandir para su uso en 2D mediante la técnica de dividir al aplicar la transformada primero a las filas y luego a las columnas; la Transformada de Wavelet puede ser implementada para imágenes de este mismo modo, obviamente teniendo en cuenta los algunos cambios en el algoritmo de cálculo, sin embargo, fundamentalmente es la misma idea. En la Figura 13 se muestra el diagrama de bloques para la implementación de la Transformada Wavelet 2D.

El resultado de aplicar la Transformada de Wavelet a una imagen de tamaño nxn, es un conjunto de 4 imágenes más pequeñas de tamaño $\frac{n}{2} x \frac{n}{2}$ las cuales representan la imagen promedio, la derivada vertical, la derivada horizontal y la gradiente de la imagen como lo muestra la Figura 14.

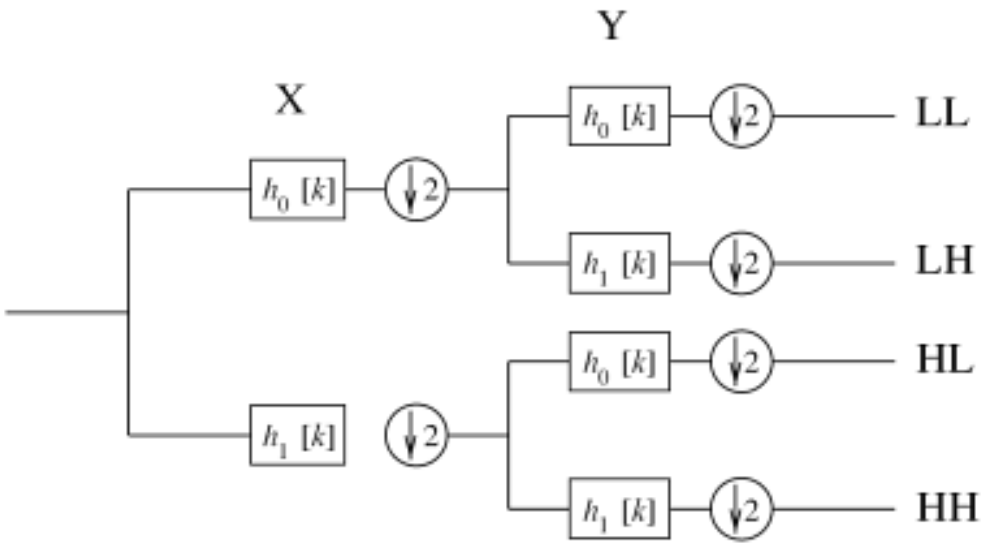

Figura 13. Diagrama de Bloques de la Transformada de Wavelet 2D. [15].

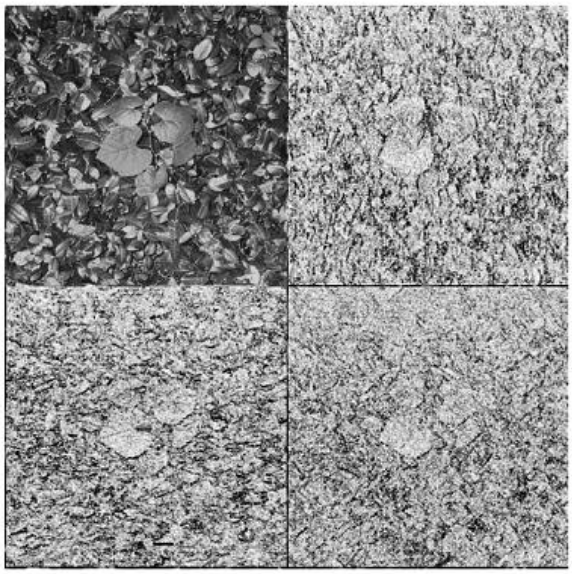

Figura 14. Transformada de Wavelet de Primer Nivel de una Imagen. [15] Sup. Izq. Imagen Sub-muestreada. Sup. Der. Derivada Vertical Inf Izq. Derivada Horizontal Inf Der. Gradiente.

2.4 Filtros Espaciales

Los filtros espaciales son una de las más útiles herramientas en el procesamiento digital de imágenes, ya que ellos permiten eliminar o resaltar características

específicas en una imagen. Existen diferentes tipos de filtros espaciales; algunos suprimen o resaltan una frecuencia espacial específica, otros se encargan de eliminar algún tipo de ruido y otros resaltan algunas formas geométricas en la imagen. Se denominan filtros espaciales ya que estos son aplicados directamente sobre la imagen y no sobre una transformada espacial de la misma.

Básicamente los filtros espaciales son matrices $n * n$ (kernel) las cuales contiene los coeficientes del filtro. La máscara del filtro es aplicada a cada pixel de la imagen original para obtener la imagen final de salida. Descrito matemáticamente en la ecuación (6), se tiene que siendo la máscara $h(k,l)$, $x(m,n)$ la imagen original y $y(m,n)$ como la imagen filtrada, se tiene que:

$$y(m,n) = \sum_{k=-a}^{a} \sum_{l=-b}^{b} h(k,l) * x(x+k, y+l) \quad (6)$$

2.4.1 Filtro Gaussiano

Matemáticamente se puede definir un filtro gaussiano como se describe en la ecuación (7):

$$g(x,y) = \frac{1}{2\pi\sigma^2} * e^{\frac{x^2+y^2}{2\sigma^2}} \quad (7)$$

Donde σ es la desviación estándar del filtro.

Este filtro se puede describir sencillamente como lo menciona [16] *"El calificativo de 'gaussiano' le viene de la definición del filtro, que sigue la forma de campana de Gauss. El máximo de la distribución coincide con la media y, de manera simétrica, a la derecha e izquierda de la media se colocan los valores según la desviación estándar de la distribución."*

Tomando como ejemplo, en la Tabla 1 se pueden observar los valores de la máscara del filtro gaussiano predeterminado por MatLab® obtenido con la función *fspecial*, de un tamaño de 3x3 y de una desviación estándar de 0.5.

0,01134374	0,08381951	0,01134374
0,08381951	0,61934703	0,08381951
0,01134374	0,08381951	0,01134374

Tabla 1. Máscara del Filtro Gaussiano Predeterminado de MatLab®. Tamaño 3x3.

En la Figura 15 se puede observar la representación gráfica del filtro gaussiano de la Tabla:

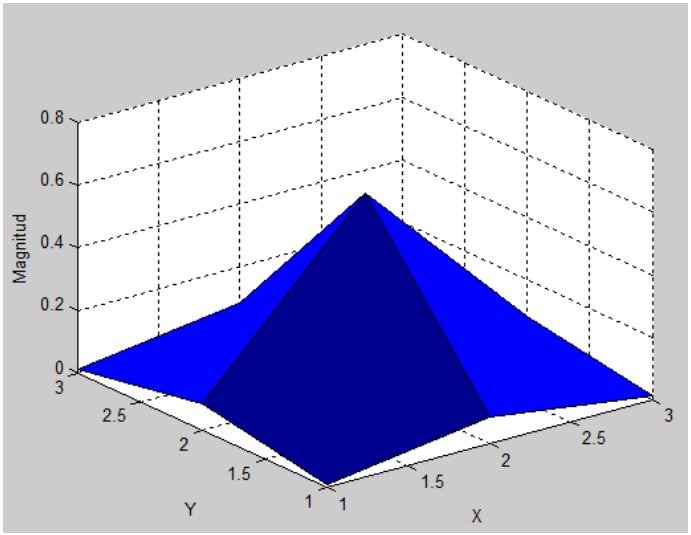

Figura 15. Representación Espacial del Filtro Gaussiano predeterminado de MatLab®.

Ahora, para evidenciar los efectos del filtrado, se mostrarán en la Figura 17 los efectos del filtro gaussiano predeterminado de MatLab® sobre la imagen original de la Figura 16

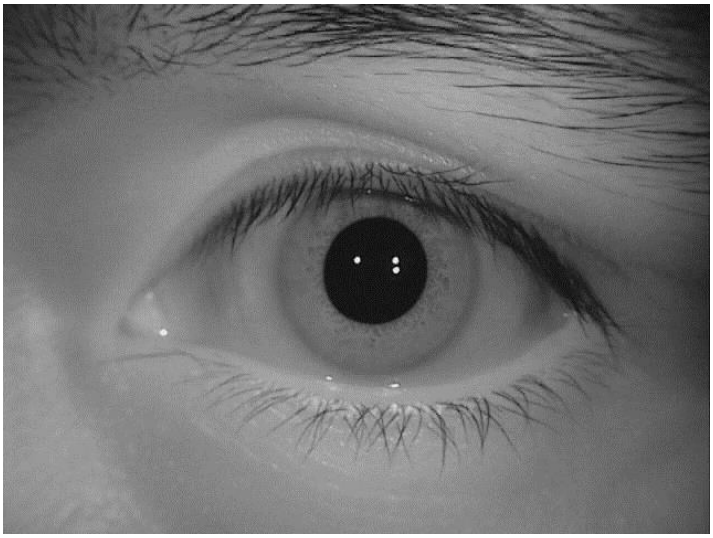

Figura 16. Imagen original antes de ser filtrada.

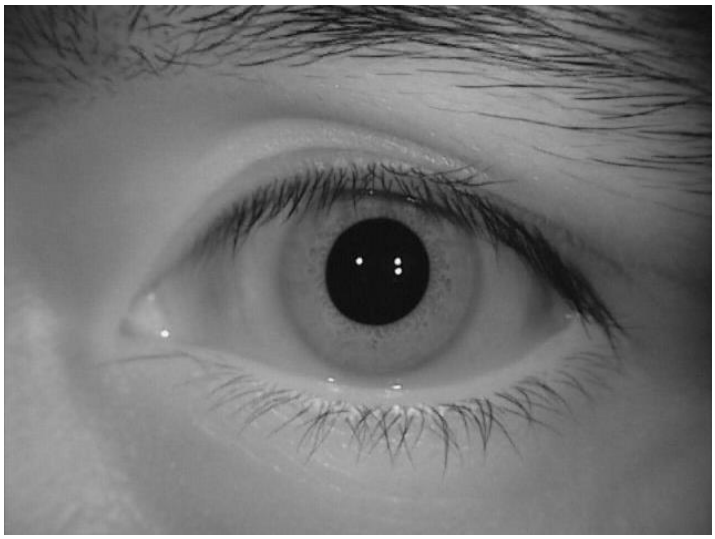

Figura 17. Imagen filtrada por el Filtro Gaussiano predeterminado de MatLab®.

2.4.2 Filtro de Promedio Circular.

Es un tipo de filtro espacial, que al igual que los filtros gaussianos son filtros de suavizado, los cuales generan una imagen *borrosa,* en la cual sus bordes son *difuminados*; siendo este el efecto en el dominio del espacio. En el dominio frecuencial, atenúa las altas frecuencias, es decir, es un filtro pasa bajas; así, su uso es muy extensivo en el mejoramiento del ruido aleatorio en las imágenes al eliminar pixeles que se alejan demasiado del promedio de sus vecinos. La forma circular es dada por el usuario para resaltar diferentes características en la imagen y es totalmente manipulable para efectuar filtrados resaltando más una dirección que otra. Estos filtros son conocidos también por su nombre en inglés como *Pillbox,* como lo muestra la Figura 18, en la cual se puede ver la *Point Spread Function (*Función de Dispersión Puntual, *PSF* por sus siglas en inglés*)* del filtro*.*

Como ejemplo, en la Tabla 2 se pueden observar los valores de la máscara del filtro de promedio circular (llamado *Disk* en MatLab®) predeterminado por MatLab® obtenido con la función *fspecial*, de un tamaño de 11x11.

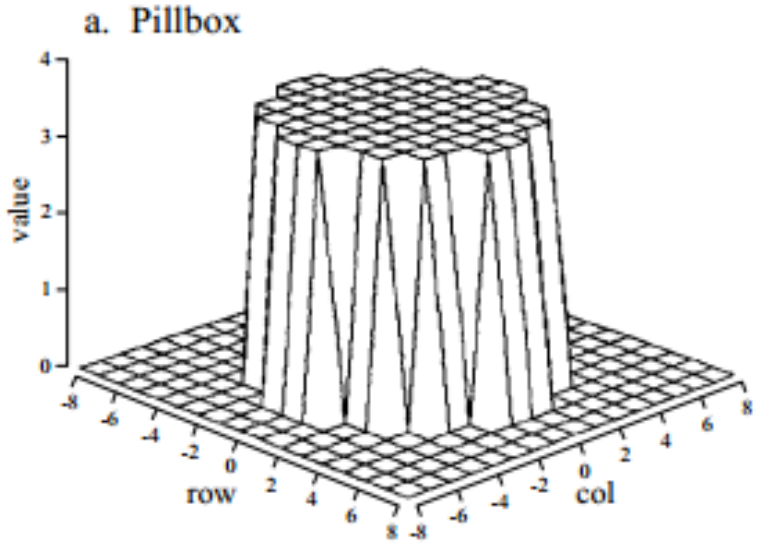

Figura 18. PSF del Filtro de Promedio Circular [17].

0	0	0	0,0012496	0,00496695	0,00625993	0,00496695	0,0012496	0	0	0
0	3,20E-05	0,00615707	0,01239587	0,0127324	0,0127324	0,0127324	0,01239587	0,00615707	3,20E-05	0
0	0,00615707	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,00615707	0
0,0012496	0,01239587	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,01239587	0,0012496
0,00496695	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,00496695
0,00625993	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,00625993
0,00496695	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,00496695
0,0012496	0,01239587	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,01239587	0,0012496
0	0,00615707	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,0127324	0,00615707	0
0	3,20E-05	0,00615707	0,01239587	0,0127324	0,0127324	0,0127324	0,01239587	0,00615707	3,20E-05	0
0	0	0	0,0012496	0,00496695	0,00625993	0,00496695	0,0012496	0	0	0

Tabla 2. Máscara del Filtro de Promedio Circular Predeterminado de MatLab®. Tamaño 11x11.

En la Figura 19 se puede observar la representación gráfica del filtro de promedio circular de la Tabla 2.

Ahora, para evidenciar los efectos de difuminado, se mostrarán en la Figura 21 los efectos del filtro de promedio circular predeterminado de MatLab® sobre la imagen original de la Figura 20.

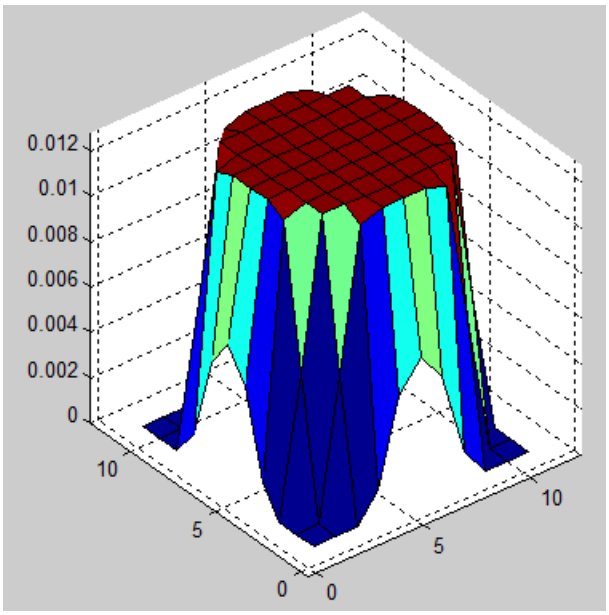

Figura 19. Representación Espacial del Filtro Gaussiano predeterminado de MatLab®.

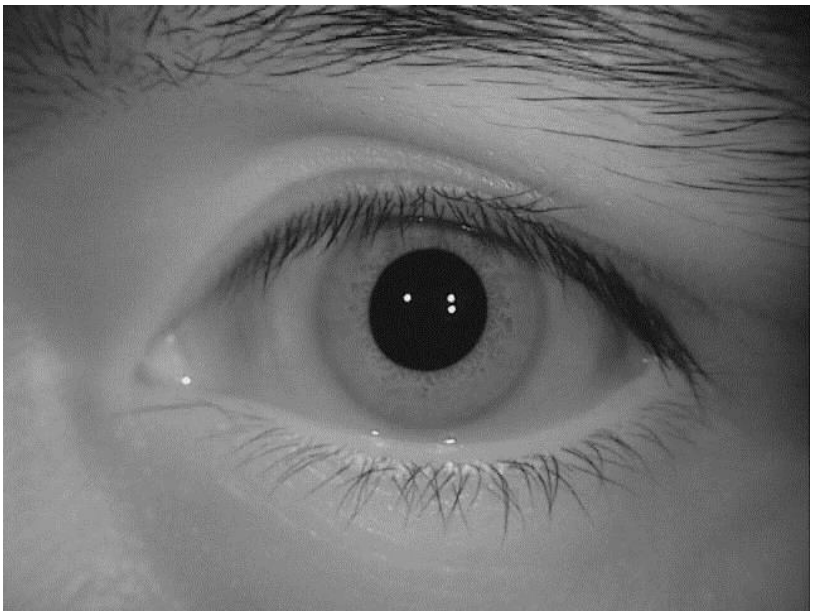

Figura 20. Imagen original antes de ser filtrada.

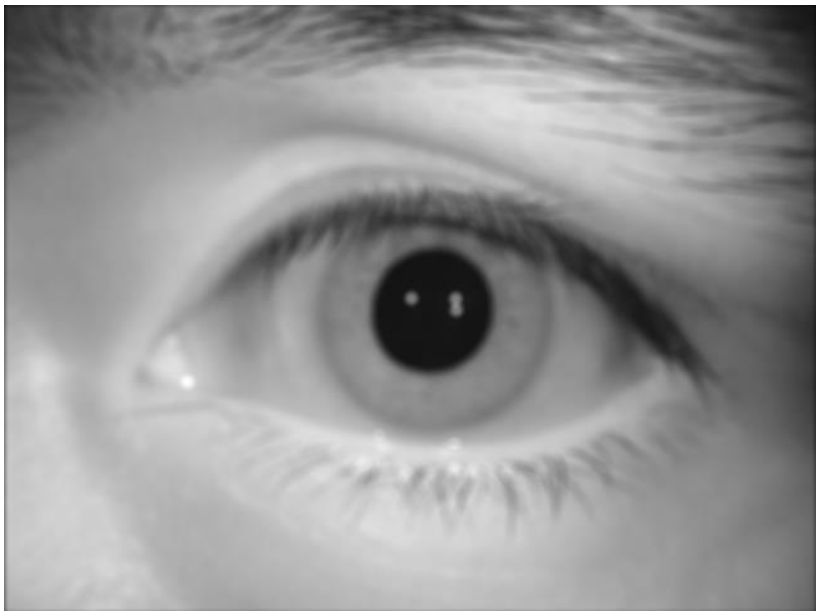

Figura 21. Imagen filtrada por el Filtro de Promedio Circular predeterminado de MatLab®.

2.5 Operador Sobel

Es una de las técnicas de detección de bordes más ampliamente usadas en el procesamiento digital de imágenes. Básicamente el operador Sobel calcula la magnitud y dirección del cambio de intensidad de un pixel con respecto a sus pixeles vecinos; así, estos cambios de intensidad pueden indicar la existencia de bordes en una imagen y la dirección en la que se mueve el borde. Esta técnica ofrece ventajas sobre la gradiente normal, ya que esta realiza el cálculo de la diferencia intensidad entre cada pixel vecino, siendo esto muy fácilmente afectado por el ruido, mientras que el operador Sobel, realiza el cálculo de la diferencia de intensidad de un pixel, promediando la diferencia entre los pixeles vecinos, de este modo, mejorando la inmunidad al ruido.

Como se menciona en [18], y como se indica en [19], la ecuación (8) define matemáticamente el gradiente de una imagen, la cual básicamente consiste en la primera derivada horizontal y vertical de la imagen.

$$\nabla f(x,y) = \begin{bmatrix} G_x & G_y \end{bmatrix} = \begin{bmatrix} \frac{\delta f}{\delta x} & \frac{\delta f}{\delta y} \end{bmatrix} \quad (8)$$

Además, de lo anterior se tiene que, como se menciona en [18], las máscaras de convolución que se aplican a una imagen digital para calcular la gradiente vertical G_y y la gradiente horizontal G_x, son descritas en la ecuación (9).

$$G_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix};\ G_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad (9)$$

De este mismo modo se tiene que la magnitud y dirección de la gradiente de una imagen viene dado por las ecuaciones (10) y (11) respectivamente, como se menciona en [18].

$$|\nabla f| = \sqrt{G_x^2 + G_y^2} \quad (10)$$

$$\angle \nabla f = \arctan\left(\frac{G_y}{G_x}\right) \quad (11)$$

Ahora, en la Figura 22 se tiene la imagen original, a la cual se le calculó la derivada vertical y la derivada horizontal, obteniéndose las Figuras 23 y 24 respectivamente.

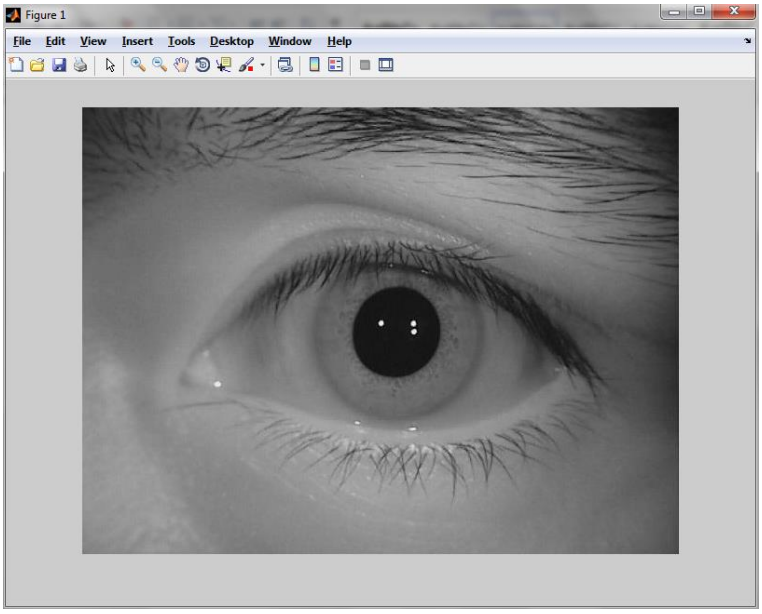

Figura 22. Imagen Original.

Después de obtener la derivada horizontal y vertical de la imagen, se procede a calcular la magnitud del gradiente de la imagen usando la ecuación (10), obteniéndose la Figura 25 y la dirección del gradiente de la imagen usando la ecuación (11), obteniéndose la Figura 26.

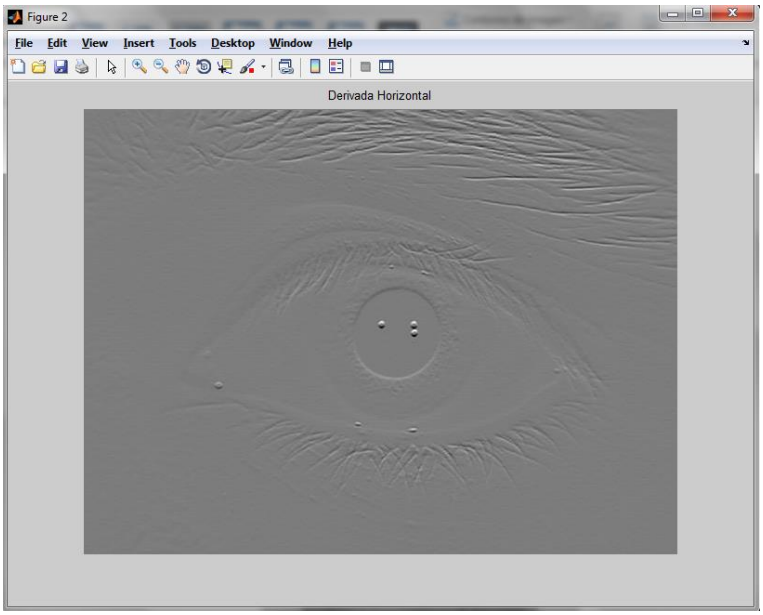

Figura 23. Derivada Horizontal G_x.

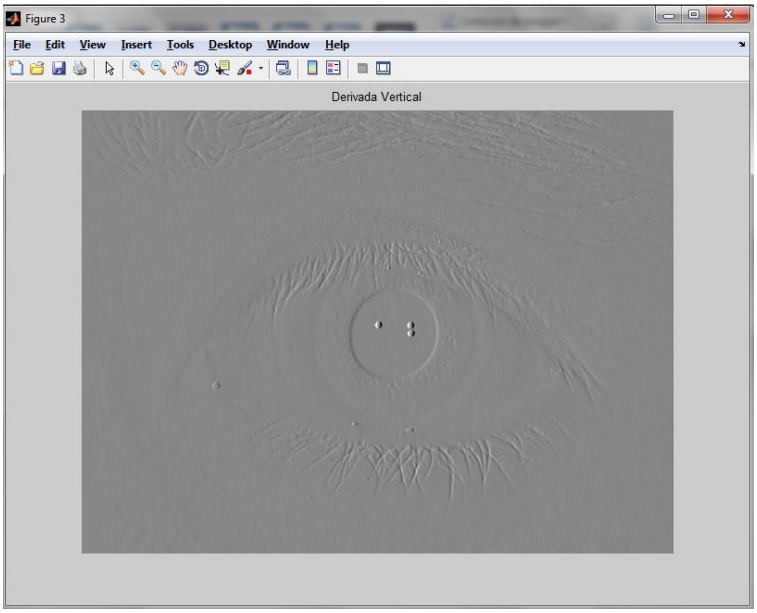

Figura 24. Derivada Vertical G_y.

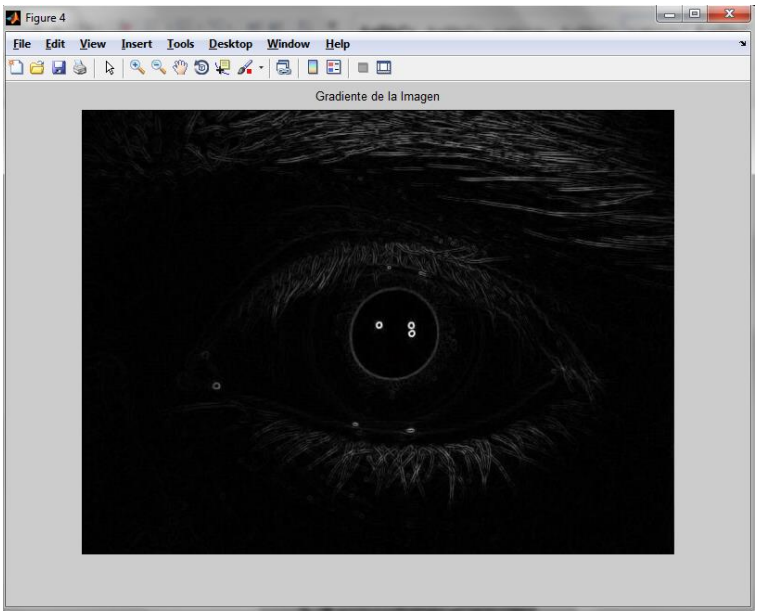

Figura 25. Magnitud del Gradiente.

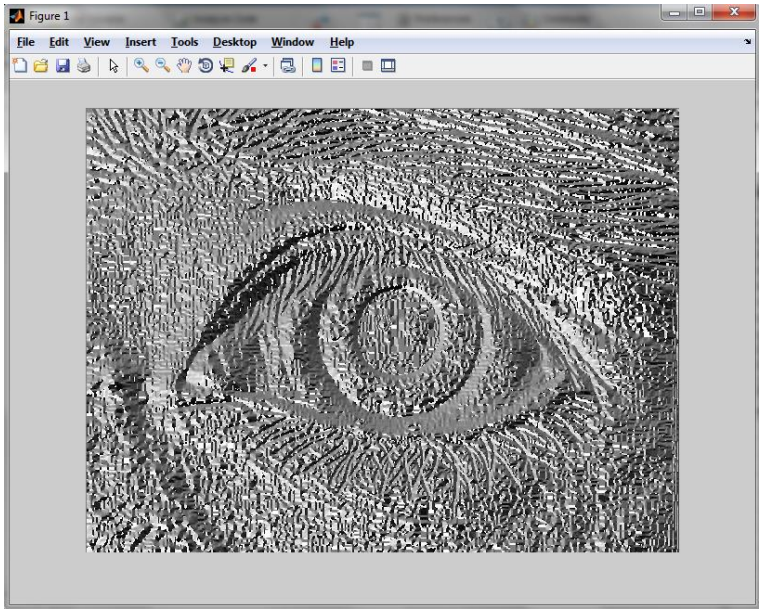

Figura 26. Dirección del Gradiente.

2.6 Envolvente Convexa

Como se menciona en [20] "*La envolvente convexa es un polígono cuyos vértices son elementos de S. Si de forma intuitiva consideramos los puntos de S como clavos sombre un panel de madera, entonces podemos pensar en la frontera de la*

envolvente convexa como la forma que toma una liga elástica que encierra a todos los clavos. ”

De una manera más gráfica y como se muestra en [20], en la Figura 27 se puede observar el concepto de envolvente convexa.

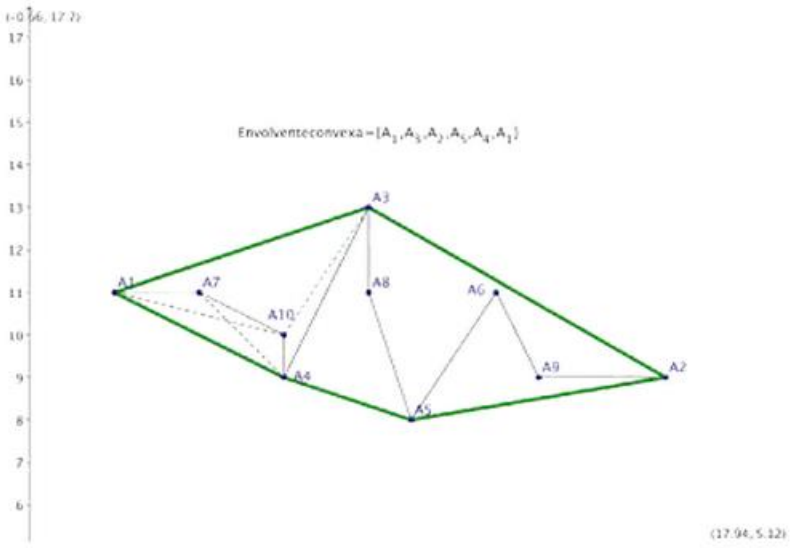

Figura 7.7: *Envolvente convexa de S.*

Figura 27. Envolvente Convexa de un Grupo de Puntos [20]

3
Procesador Digital de Señales ADSP-BF533

3.1 Descripción General [21]

El *ADSP-BF533 EZ-Kit Lite*® de la Figura 28, es un kit de desarrollo ofrecido por *Analog Devices,* el cual está basado en el procesador digital de señales ADSP-BF533, perteneciente a la famosa familia de procesadores Blackfin® de Analog Devices. Este kit ofrece al diseñador un hardware especialmente diseñado para el desarrollo de aplicaciones que requieran el procesamiento digital de audio y video tal como el decodificador de video ADV-7183, el codificador-decodificador de audio AD-1836 y el codificador de video ADV-7171 entre otros. Además este kit incorpora el software *VisualDSP*++, el cual le da la posibilidad al diseñador de trabajar desde un entorno de programación totalmente desarrollado y enfocado en su dispositivo, de este modo mejorando la compatibilidad y el acceso más rápido y sencillo a todas las características de la tarjeta.

Figura 28. Tarjeta de Desarrollo *ADSP-BF533 EZ-Kit Lite*® de Analog Devices [21].

Básicamente la tarjeta cuenta con un rendimiento de hasta 600MHz en velocidad de procesamiento con un reloj base de 27MHz, esto acompañado de un conjunto de memorias, una SDRAM de 64MB y una memoria Flash de 2KB. En la parte de procesamiento de audio, la tarjeta incorpora el codificador-decodificador AD-1836, con velocidad de muestreo de hasta 96Khz, además de contar con 4 conectores de entrada RCA y 6 conectores RCA de salida. Respecto a video, esta tarjeta cuenta

con un canal entrada, el cual tiene un decodificador ADV-7183 y 3 conectores RCA, mientras que el canal de salida, se tiene un codificador ADV-7171 y sus 3 conectores RCA. Para interacción externa con el usuario, la tarjeta posee 10 LED's, 5 botones pulsadores y 4 banderas programables; y finalmente también se cuenta con un transmisor-receptor asíncrono universal (Universal Asychronous Receiver Transmitter, UART por sus siglas en inglés).

De este modo, *ADSP-BF533 EZ-Kit Lite*® brinda la posibilidad de desarrollar e implementar aplicaciones de audio y video de alto rendimiento, desde una misma tarjeta de desarrollo, con todos los periféricos requeridos fácilmente accesibles y configurables; y todo trabajado desde un entorno de programación en C++ amigable con el usuario.

A continuación se describirá un poco más a fondo la arquitectura del procesador ADSP-BF533 y los diferentes periféricos usados en la implementación de este sistema de reconocimiento de iris, sin embargo para conocer más detalles de configuraciones de registros, módulos de comunicación, y más características técnicas de esta tarjeta de desarrollo, se podrá consultar en [21], [22], [23] y [24].

3.2 Arquitectura del *ADSP-BF533 EZ-Kit Lite*® [21]

Este kit de desarrollo posee un hardware especialmente diseñado e implementado en la tarjeta para el procesamiento en tiempo real de aplicaciones que requieran un alto rendimiento en audio y video; de este modo, a continuación se mencionarán las diferentes características de hardware de la tarjeta:

3.2.1 Características Generales

- Procesador Blackfin® ADSP-BF533 de Analog Devices.
  Este procesador ofrece un rendimiento hasta de 600MHz con un oscilador de reloj base de 27 MHz bajo un empaquetamiento mini-BGA de 160 pines.

- Memoria Síncrona Dinámica de Acceso Aleatorio (Synchronous dynamic random access memory, SDRAM por sus siglas en inglés).
  Memoria MT48LC32M16 de 64 MB (32M registros de 16 bits cada uno).

- Memoria Flash.
  Memoria de 2 MB (512K registros de 16 bits cada uno en 2 chips diferentes).

- Interfaz de Audio Análogo.
  Esta interfaz posee un codificador de Audio de 96KHz AD1836, 4 conectores de entrada (2 Canales de Audio Estéreo) y 6 Conectores de salida tipo RCA (3 Canales de Audio Estéreo).

- Interfaz de Video Análogo.
  Esta interfaz cuenta con un decodificador de entrada de video ADV7183 y un codificador de salida de video ADV7171, cada uno con 3 conectores tipo RCA.

- Módulo UART.
  Este módulo de comunicación posee un controlador-Receptor ADM3202 RS-232 y un conector macho DB9.

- LED's.
  Esta tarjeta tiene 10 Leds, los cuales se distribuyen en 1 LED de encendido (verde), 1 LED de reinicio de la tarjeta (rojo), 1 LED de conexión USB (verde), 6 LED's de propósito general (ámbar), y un 1 LED monitor USB (ámbar).

- Botones Pulsadores.
  Se cuenta con 5 botones pulsadores y 4 banderas programables.

- Interfaces de Expansión.
  Interfaz para Periféricos Paralelos (Peripheral Parallel Interface, PPI por sus siglas en inglés), Interfaz para Periféricos Serie (Serial Peripheral Interface, SPI por sus siglas en inglés). La unidad de interfaz de bus externo, (External Bus Interface Unit, EBIU por sus siglas en inglés), 3 Contadores, banderas programables y dos puertos serie síncronos de alta velocidad (SPORT0 Y SPORT1).

- Otras Características.
  Conector de 14 pines JTAG ICE.

En la Figura 29 se puede observar la arquitectura en diagrama de bloques de la tarjeta ADSP-BF533 EZ-Kit Lite®, en la que se puede ver más concretamente el modo en que los dispositivos mencionados anteriormente se interconectan para componer la tarjeta.

3.2.2 Procesador ADSP-BF533. [22]

El procesador ADSP-BF33, como ha mencionado anteriormente, es un procesador de alto rendimiento en el procesamiento de audio y video digital, diseñado por

Analog Devices para ofrecer una excelente eficiencia en este tipo de aplicaciones, todo con un bajo consumo de potencia en comparación con anteriores procesadores de la familia Blackfin®; esto gracias a su manejo de potencia dinámico, el cual permite modificar la frecuencia del reloj y el voltaje de operación del núcleo en tiempo real.

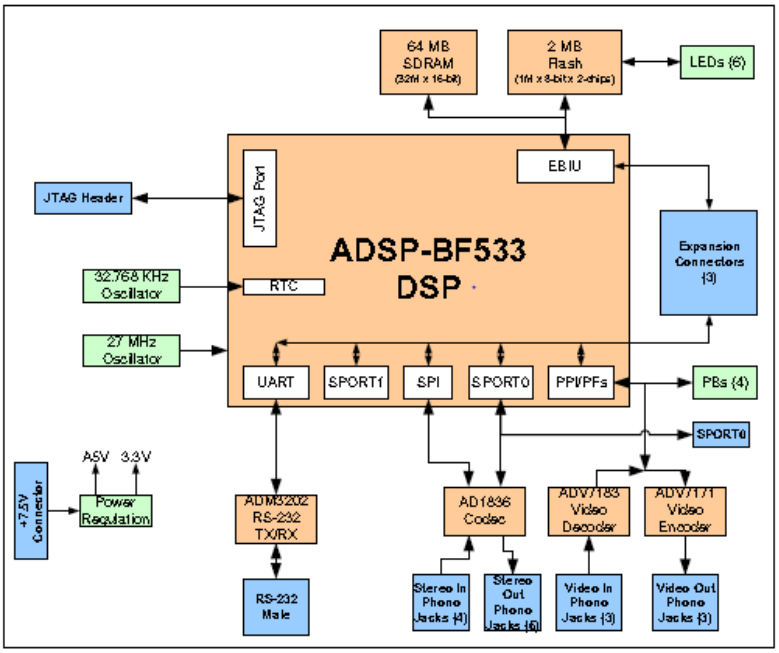

Figura 29. Arquitectura del ADSP-BF533 EZ-Kit Lite®. [21]

Además de disponer de todos los periféricos de la tarjeta mencionados anteriormente, el procesador Blackfin® posee los siguientes módulos:

- Reloj de Tiempo Real (Real Time Clock, RTC por sus siglas en inglés).
- Contadores de Propósito General (Timers).
- Entradas y Salidas de Propósito General con banderas programables.
- Watchdog Timer.

El control de todos los periféricos exceptuando el RTC, los Timers y las entradas y salidas de propósito general; están basados en un diseñado de estructura dinámica de Acceso Directo a Memoria (Direct Memory Access, DMA por sus siglas en inglés); inclusive se tienen reservados dos canales DMA para el manejo de la transferencia de datos entre el procesador y las memorias; un canal para las memorias internas(L1 de Datos y L1 de Instrucciones) y otro para las memorias externas como la SDRAM y la memoria asíncrona; además de esto, todos los periféricos están conectados mediante buses de gran ancho de banda. Así, al

utilizar una configuración DMA y estar conectados por buses de alto rendimiento, se genera un flujo de datos constante y a tiempo hacia el procesador, manteniendo su tiempo de funcionamiento lo más alto posible, mejorando notablemente la eficiencia y velocidad de respuesta de los periféricos. Esta estructura de conexión y manejo de periféricos puede ser observada más fácilmente en el diagrama de bloques de la Figura 30.

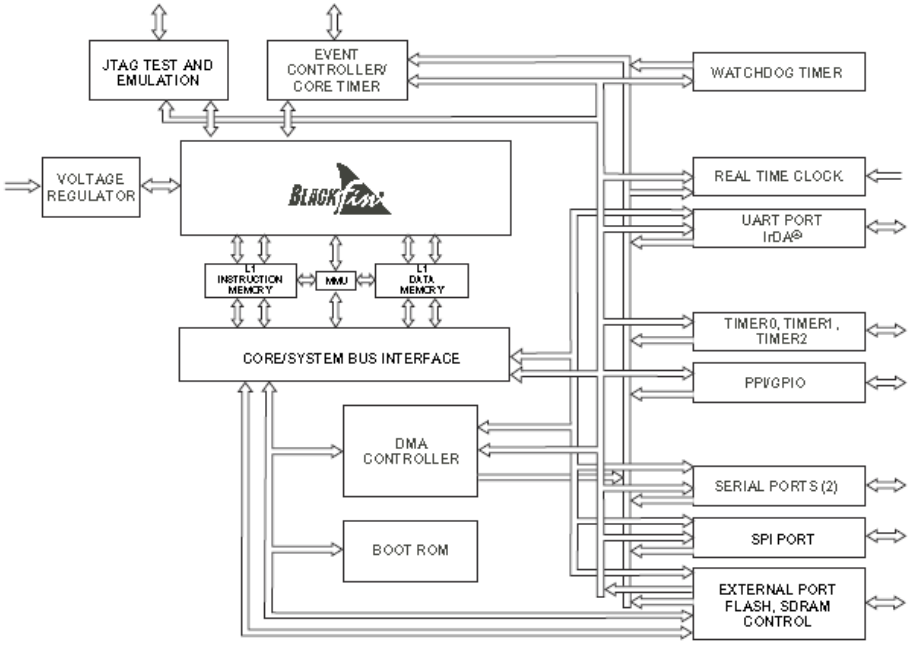

Figura 30. Diagrama de Bloque del ADSP-BF533. [22]

3.2.3 Núcleo de Procesador ADSP-BF533

Este es fundamentalmente el módulo del procesador encargado de realizar las operaciones aritmético-lógicas a bajo nivel, ejecutando las diferentes instrucciones disponibles en su set de instrucciones. Para conocer más acerca de sus componentes, a continuación se enumerarán sus diferentes módulos, los cuales pueden ser vistos la Figura 31, la cual muestra el diagrama de bloques del núcleo del procesador.

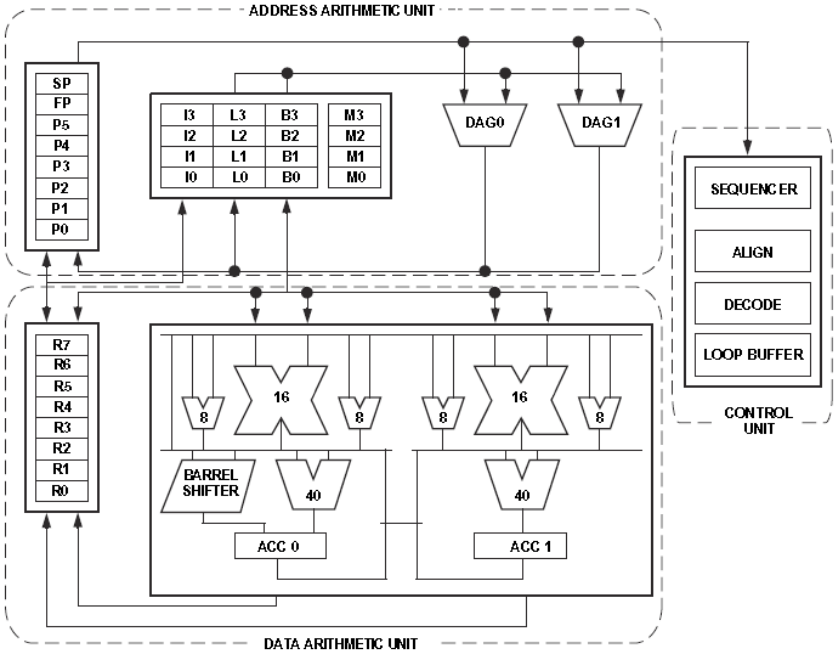

Figura 31. Diagrama de Bloques del Núcleo del ADSP-BF533. [21]

Esencialmente el núcleo de procesador ADSP-BF533 está compuesto por:

- Dos MAC (Multiplier accumulator MAC, por sus siglas en inglés) de 16 bits cada uno.
  Los MAC son multiplicadores de 16 bits x 16 bits con resultados de 32 bits que se pueden llevar a un acumulador de 40 bits para realizar operaciones de suma o resta. Estos MAC permiten realizar operaciones de multiplicaciones y multiplicación-acumulación de punto fijo en un solo ciclo de trabajo, con formatos numérico con o sin singo.

- Dos Unidades Aritmético Lógicas (Arithmetic Logic Unit, ALU por sus siglas en inglés) de 40 bits.
  La ALU ofrece las operaciones aritmético-lógicas comunes tanto en 32 bits, 16 bits o 8 bits; permitiendo operaciones como suma y resta de valores inmediatos o de registros en punto fijo, funciones especiales como redondeo, valor máximo, mínimo y valor absoluto; todo esto con el fin de ofrecer un conjunto de operaciones optimizadas para el procesamiento digital de audio y video.

- Cuatro ALU's de Video de 8 bits cada una.
  Para el procesamiento de video se cuenta con estas 4 ALU's de video de 8 bits cada una, las cueles están especialmente diseñadas para realizar

operaciones con este tipo de datos, las cuales son básicamente suma, resta, promedio, empaquetamiento, extracción, resta absoluta, resta acumulativa, todas de 8 bits cuádruples.

- Una Unidad de Corrimiento (Barrel Shifter) de 40 bits.
  El Barrel Shifter es el encargado de realizar las operaciones de desplazamientos lógicos, aritméticos, rotaciones, extracción y empaquetamiento de paquetes de bits; todo estos posible con entradas de 16 bits, 32 bits o 40 bits.

- Dos Acumuladores de 40 bits cada uno.

El banco de registros de trabajo se compone de 8 registros de 32bits cada uno para realizar las diferentes operaciones de cálculo; además, si es necesario, cada registro se puede dividir en dos registros independientes de 16 bits cada uno.

3.2.4 Arquitectura de Memoria.

La distribución de las memorias en el DSP-BF533 EZ Kit es de ordenamiento jerárquico, donde se busca ofrecerle al usuario la mejor velocidad y rendimiento posible del procesador frente a un costo económico relativamente aceptable, es decir, se distribuyen las memorias dentro del procesador y de la tarjeta dependiendo de sus características técnicas y de su importancia para el núcleo de procesador; todo con la mejor configuración que ofrezca una buena relación costo/rendimiento. De este modo se tiene que se debe ofrecer memorias de alto rendimiento (poca o casi nula latencia) cerca del núcleo del procesador para garantizarle un acceso a datos lo más rápido imposible; sin embargo estas memorias son de poco tamaño, ya que su costo es elevado por sus características de alta velocidad. Así, se puede crear una jerarquía en donde la estructura básica se fundamenta en el hecho de que entre más necesaria sea la memoria para el núcleo, esta va a ser más rápida pero más pequeña y, entre menos importante o necesaria sea, esta memoria puede ser más grande y más lenta.

En el procesador ADSP-BF533 se encuentra una jerarquía como la que se mencionó anteriormente la cual está direccionada por registros de 32 *bytes* y un espacio de direcciones unificado de 4*Gbytes*. Siendo esta una estructura DMA, todas las memorias tanto internas y externas, y los registros de control de los periféricos ocupan un espacio común en esta memoria de direccionamiento.

Como se observa en la Figura 32, el procesador divide sus memorias en dos grupos. Las memorias internas al propio circuito integrado del procesador (On-Chip) y las memorias fueras del circuito integrado del procesador (Off-Chip) que se comunican mediante los diferentes buses de datos o interfaces.

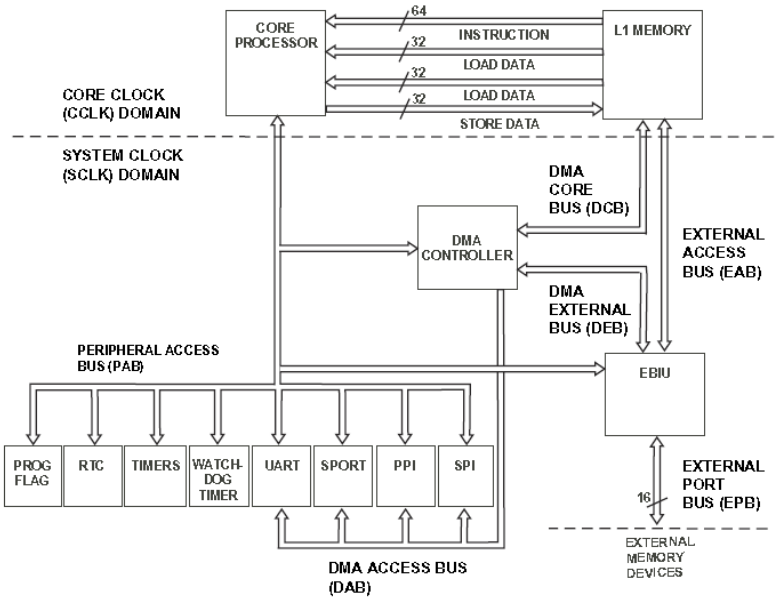

Figura 32. Arquitectura de Memoria del Procesador ADSP-BF533.

Las memorias On-Chip son las más rápidas y de gran ancho de banda pero a la vez son pequeñas en tamaño. Así, en el procesador se tienen tres tipos de memorias. La memoria de instrucciones L1 de tipo SRAM y de 80K bytes y la memoria de datos L1 de tipo SRAM con 2 bancos de 32K bytes cada; ambas de velocidad de acceso igual a la del procesador y configura bles como *cache*; y la memoria *Scratchpad SRAM* de tipo SRAM y de 4K bytes la cual no es configurable como cache. En el caso de las memorias off-chip, se cuenta con diferentes tecnologías de memorias asíncronas como las SRAM, FLASH, EEPROM y ROM, y en el caso de memorias síncronas se tienen memorias como la SDRAM.

Específicamente para este proyecto, se hizo uso de la memoria SDRAM ya que ofrece al proyecto la capacidad y direccionamiento suficiente para el manejo de las imágenes del iris cargadas a la tarjeta; ofreciendo una capacidad de 64MB divididos en 32M registros de 16bits cada uno.

4
Sistema de Reconocimiento de Iris

En este capítulo se realizará las descripción del sistema de reconocimiento de iris implementado en MatLab® y en el ADSP-BF533 EZ-Kit Lite®. Básicamente este sistema hace uso de las características aleatorias y completamente únicas del iris de cada persona para proporcionar una clave única de acceso al sistema; de este modo, el sistema verifica la identidad del usuario mediante el análisis del iris del usuario a ser validado por el sistema contra una base de datos de usuarios autorizados previamente y determina si este está autorizado o no.

4.1 Descripción General

Este sistema permite dos funciones básicas que son el *Registro de Usuario,* la cual admite ingresar un nuevo usuario valido al sistema y generar una plantilla del usuario, la cual es almacenada en la base de datos del sistema; y la *Validación de Usuario,* la cual verifica la identidad del usuario que se está analizando y genera un resultado de si el usuario está o no registrado en la base de datos del sistema.

Otra característica importante de este sistema es que fue desarrollado para admitir un único usuario registrado, es decir, sólo un usuario válido es permitido en el sistema. Esto es debido a que ingresar más usuarios válidos al sistema exigían algoritmos más complejos en el módulo de Análisis de Identidad lo cual se extralimitaría en los objetivos del proyecto. Sin embargo, este punto abre la puerta para que posteriormente se puedan realizar proyectos que implementen mejoras y algoritmos más complejos sobre algunos de los módulos aquí desarrollados.

A continuación se mencionarán las funciones cumplidas por cada bloque del sistema de reconocimiento de iris. En el capítulo 5 serán descritos por aparte y con más detalle los algoritmos usados para las implementaciones tanto en MatLab® como en el ADSP-BF533 EZ-Kit Lite®.

4.2 Diagrama de Bloques General del Sistema de Reconocimiento de Iris

En el siguiente diagrama de bloques de la Figura 33 se mostrará el funcionamiento general del sistema de reconocimiento de iris, en el cual se especifican las dos funciones principales del sistema y sus diferentes bloques funcionales. Se describirán los bloques más sencillos y, los bloques más complejos serán descritos en más profundidad posteriormente.

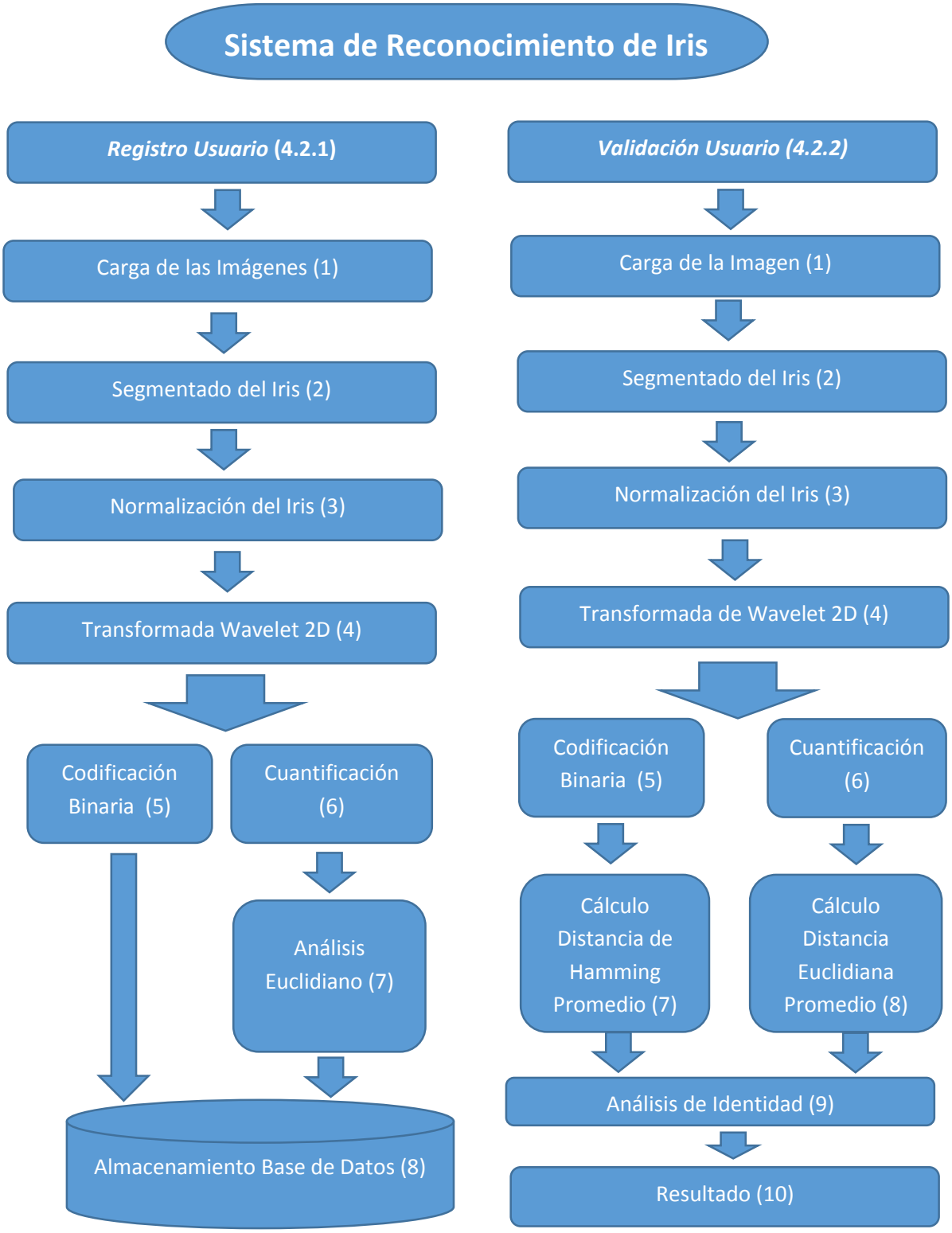

Figura 33. Diagrama de Flujo General del Sistema de Reconocimiento de Iris.

4.2.1 Registro de Usuario

Es una de las dos funciones principales del sistema y es la encargada de agregar un nuevo usuario autorizado por el sistema. Para registrar un usuario nuevo se utilizan 19 de las 20 imágenes del iris disponibles del usuario del ojo a registrar; las cuales serán procesadas para extraer sus características mediante la Transformada Wavelet de Haar 2D, para su posterior análisis por Distancia de Hamming y Distancia Euclidiana, los cuales generan una bases de datos y dos umbrales cada uno; que serán almacenados en la base del sistema para su posterior uso por la función de Validación.

1. **Carga de las Imágenes**
   Como se mencionó anteriormente, se necesitan 19 imágenes diferentes para el registro del nuevo usuario, de este modo, las imágenes utilizadas para las pruebas del sistema fueron tomadas de un usuario de la base de datos CASIA Iris Database 4.0, así, este módulo se encarga de cargar las imágenes desde un directorio específico en MatLab o importándolas desde el PC al DSP.

2. **Segmentado del Iris**
   Este módulo es el encargado de encontrar y entregar las coordenadas y radios de los círculos pertenecientes al iris y la pupila. Será descrito con más detalle en el numeral 4.3.

3. **Normalización del Iris**
   Es el módulo encargado de convertir la región del iris, comprendida entre los círculos concéntricos del iris y la pupila, en una imagen rectangular de tamaño 264x64 pixeles, mediante el uso de cambio de coordenadas e interpolación bilineal. Será descrito con más detalle en el numeral 4.4.

   La normalización del iris fue implementada en MatLab® gracias al desarrollo de la función *"normaliseiris.m" por parte de [27],* la cual sirvió de base fundamental para su uso en el desarrollo del sistema en MatLab® y su posterior modificación, simplificación e implementación *en* el DSP.

4. **Transformada Wavelet de Haar 2D (CoefW)**
   Módulo encargado de aplicar la Transformada Wavelet de Haar 2D a la imagen normalizada del iris obtenida del módulo anterior. Este método de caracterización de imágenes fue elegido sobre la Transformada de Fourier por sus evidentes ventajas ya mencionadas tanto en el capítulo 3 como en [25] y [26].

Este módulo recibe una imagen de tamaño 128x64 pixeles y entrega una matriz 8x16. Se decidió aplicar una transformada de cuarto nivel a la imagen de entrada, ya que como se menciona en [25], una transformada de menor nivel, entrega un vector de coeficientes muy grande, y debido a que se debe implementar en un DSP, el cual tiene recursos de memoria bastante más limitados que un computador personal; de este modo, para una imagen de 128x64 pixeles, la transformada entregaría un vector de características de 128 coeficientes.

En el diagrama de flujo de la Figura 34 se puede observar el proceso seguido para la implementación de la Transformada Wavelet de Haar. Es de anotar que el proceso descrito en la Figura 34 es el mismo tanto para las implementaciones en MatLab® como en el ADSP-BF533 EZ-Kit Lite®, ya que no se usó el toolbox de Transformada Wavelet de MatLab®, sino que se implementó el algoritmo en ambas plataformas, usando las operaciones matemáticas y operados básicos disponibles en C, todo sin el uso funciones especiales.

Extracción de Características

Después de obtener la matriz de 19x128 correspondiente a la Transformada Wavelet de Haar 2D de las 19 imágenes, se procede a extraer las características necesarias para ser usadas posteriormente por el proceso de análisis de identidad del sistema. Se utiliza al tiempo los métodos de comparación entre vectores de características usados en [25] y [26], en los cuales se menciona la Distancia de Hamming y la Distancia Euclidiana para determinar la similitud entre los vectores.

A continuación se describirá cada uno de los métodos implementados paralelamente para la extracción de características y análisis de identidad por estos dos métodos.

5. Codificación Binaria (BaseH)

Como se menciona en [25], se realiza una codificación binaria en donde los coeficientes de la Transformada Wavelet de Haar 2D (CoefW) que son negativos se codifican a 0 y los positivos se codifican a 1. Al final se obtiene una matriz de 19x128 de valores binarios. Esta matriz de 19x128 se almacenará posteriormente como BaseH para uso del sistema.

6. Cuantificación (BaseE)

Este módulo se encarga de realizar una modificación a la matriz de 19x128 de coeficiente de la Transformada Wavelet de Haar 2D (CoefW) con el objetivo de mejorar el reconocimiento de patrones para el módulo de análisis de identidad.

El método de cuantificación usado consiste en realizar un análisis por histograma de cada uno de los 19 vectores de 128 valores provenientes de la Transformada Wavelet de Haar 2D; el cual entrega información de los rangos donde se concentra la información de los coeficientes. Posteriormente se toman los cinco rangos donde más se concentra la información, donde cada uno de los 128 coeficientes es aproximado al rango más cercano, para a continuación realizar una cuantificación.

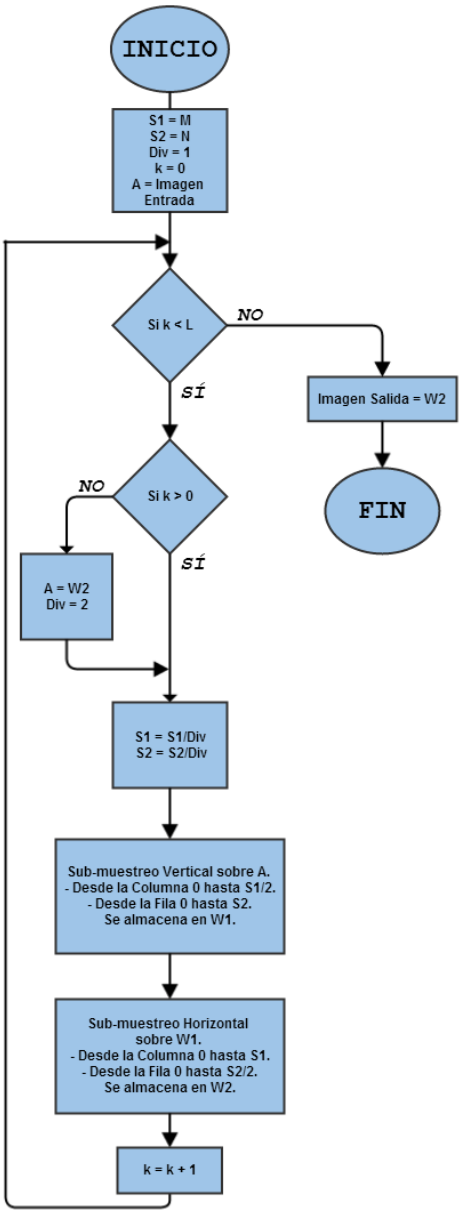

Figura 34. Diagrama de Flujo de la Implementación de la Transformada Wavelet de Haar 2D.

En la Figura 35 se puede observar el histograma de 95 vectores diferentes de 128 valores cada uno, en el cual se aprecia la distribución no uniforme de la información en los vectores, de este modo, el método de cuantificación explicado anteriormente se adapta y elige los rangos de cuantificación donde se encuentre la mayor parte de la información. A la salida de este módulo se obtiene una matriz de 19x128 valores, la cual es llamada BaseE

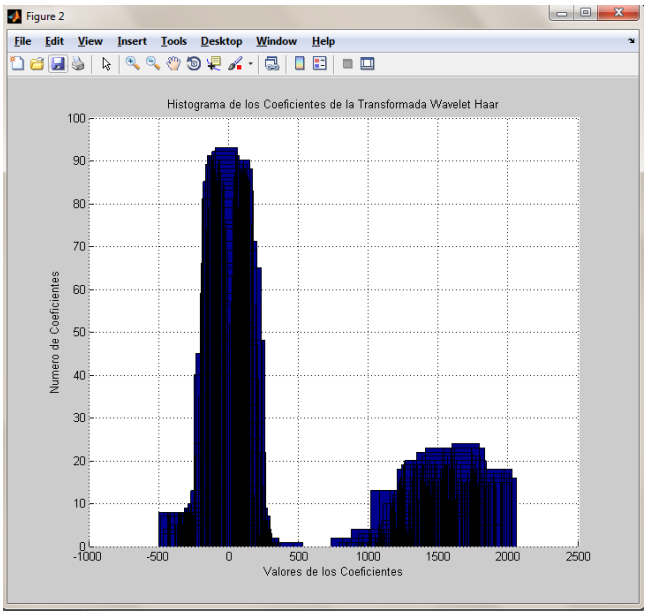

Figura 35. Histograma 95 Vectores Diferentes de Coeficientes de la Transformada Wavelet de Haar 2D.

7. Análisis Euclidiano

Después de realizar la cuantificación mencionada anteriormente, se procede a ejecutar un análisis de la misma base de datos Euclidiana (BaseE) para determinar la Distancia Euclidiana y el promedio de la desviación estándar de estas distancias entre los 19 vectores de la base de datos. A continuación se describirá este módulo más a fondo.

7.1 Cálculo Distancia Euclidiana (DistE) [26]

Posteriormente se procede a calcular la Distancia Euclidiana entre la misma base de datos, es decir, a cada uno de los 19 vectores de 128 datos se le calculará la Distancia Euclidiana con los otros 18 vectores usando la ecuación (12). Esto se realizó básicamente para que posteriormente se puedan generar umbrales automáticos, los cuales generan un valor máximo de la Distancia Euclidiana, así, si un usuario lo supera, será tomado como No Autorizado.

$$D_{(x,y)}^{euclidean} = \sqrt{\sum_{i=0}^{N}(x_i - y_i)^2} \quad (12)$$

A la salida de este módulo se obtiene una matriz de 19x19 (DistE) que contiene los resultados de la Distancia Euclidiana entre los 19 vectores de 128 coeficientes, los cuales se calculan usando la ecuación (13), la cual realiza el cálculo Distancia de la base de datos

$$DistE(j,k) = \sqrt{\sum_{i=0}^{N}\left(BaseE(j,:) - BaseE(k,:)\right)^2} \quad (13)$$

Donde
$BaseE$ es la base de datos de 19 vectores de 128 valores cada uno.
$DistE$ es la matriz de Distancias Euclidianas de la misma base de 19x19.

A continuación se procede a calcular dos umbrales automáticos a partir de la matriz obtenida anteriormente de Distancias Euclidiana, los cuales serán utilizados en la función de validación para determinar la identidad del usuario de prueba.

7.2 Cálculo del Promedio de la Distancia Euclidiana (MeanE)

Este módulo calcula el valor promedio de la matriz DistE; esto con el fin de conocer cuál es la Distancia Euclidiana promedio entre las 19 imágenes de la base de datos.

7.3 Cálculo del Promedio de la Desviación Estándar de la Distancia Euclidiana (StdE)

Este módulo calcula el valor promedio de la desviación estándar de la matriz EE, esto con el fin de agregarle una tolerancia al umbral M2E a la hora de realizar el proceso de análisis de identidad.

7.4 Cálculo Distancia Euclidiana (UmbralE)

Finalmente se procede a realizar el cálculo del umbral para la Distancia Euclidiana de la base de datos. Simplemente es la suma del Promedio de la Distancia Euclidiana y del Promedio de la Desviación Estándar de la Distancia Euclidiana, como se muestra en la ecuación (14).

$$UmbralE = MeanE + StdE \quad (14)$$

8 Almacenamiento Base de Datos

Finalmente se procede a almacenar las variables calculadas anteriormente; las matrices de 19x128 Base de Datos Binarizada BaseH y la Base de Datos Cuantificada BaseE; y la variable UmbralE.

Todas estas variables serán almacenadas en un archivo .m para su posterior uso por la función de Validación Usuario

4.2.2 Validación de Usuario

Esta es la función principal del sistema de reconocimiento de iris. Esta función es la encargada de determinar si un usuario está o no permitido por el sistema, esto, usando la base de datos creada por la función de *Registro Usuario*. Básicamente esta función recibe una imagen del iris del usuario a validad y la base de datos del sistema, entregando un resultado del usuario, ya sea que es permitido o no es permitido por el sistema.

Esta función, al igual que la de *Registro Usuario,* hace uso de los módulos de *Segmentado del Iris, Normalización, Transformada Wavelet de Haar 2D y el bloque de Extracción de Características* la diferencia radica en que esta función solo realiza este proceso *para una sola imagen* (Imagen del usuario a validar),

1. **Carga de la Imagen**
   Este módulo recibe la imagen del usuario de tamaño de 640x480 a ser analizado, cargándola de un directorio específico (MatLab) o a un registro en el DSP.

2. **Segmentado del Iris**
   Después de cargar la imagen en el sistema, se procede a segmentar el iris, es decir buscar las coordenadas (x, y, r) pertenecientes al círculo del iris y de la pupila.

3. **Normalización del Iris**
   Este módulo recibe las coordenadas del iris y la pupila, para convertir la región circular del iris en un rectángulo de tamaño $256x64$.

4. **Transformada Wavelet de Haar 2D (CoefW)**
   Se recibe una imagen de tamaño $256x64$ del iris normalizado y este módulo procede a extraer las características del iris del usuario de prueba al calcular los coeficientes de 4° nivel de la Transformada Wavelet de Haar 2D, entregando un vector de 128 valores.

Extracción de Características
Este módulo se encarga de extraer las características y calcular los valores de Distancia de Hamming y Distancia Euclidiana para el usuario de prueba, para posteriormente ser evaluado contra los umbrales calculados en la base de datos.

5. Codificación Binaria (BaseHP)
Se recibe el vector de coeficientes (CoefW) de la Transformada Wavelet Haar 2D de 128 valores para ser codificados como se mencionó anteriormente, para la final obtener un vector binario de 128 valores (BaseHP).

6. Cálculo Distancia de Hamming Promedio (DistHM)
Aplicando la ecuación (15) se procede a realizar el cálculo de la Distancia de Hamming del vector binarizado (BaseHP) de características del usuario de prueba contra la base de datos del sistema BaseH, de 19x128 valores, dando como resultado 19 distancia de Hamming diferentes. Así se obtiene un vector de 19 valores (DistHP) a la salida de este módulo.

$$DistHP(k) = \frac{1}{N}\sum_{j=1}^{N}\left(BaseHP \text{ xor } \mathrm{BaseH(k)_j}\right) (15)$$

Donde:
$DistHP$ es el vector resultado de 19 valores de Distancia de Hamming entre el vector de prueba y la base de datos.
$BaseHP$ es el vector binarizado del usuario de prueba de 128 valores.
$BaseH_j$ es la matriz de 19 vectores de 128 valores cada uno.

Ahora se procede a calcular el valor promedio de la Distancia de Hamming del vector DistHP, entregando una medida de cuan similar o diferente es el vector de características binario a la base de datos del sistema, siendo este resultado la variable ***DistHM.***

7. Cuantificación (BaseEP)
Del mismo modo que se mencionó anteriormente, se procede a cuantificar el vector de características (CoefW) de la Transformada Wavelet de Haar 2D utilizando el método descrito; así, obteniéndose a la salida un vector de 128 valores enteros entre 1 y 5. (BaseEP)

8. Cálculo Distancia Euclidiana Promedio (DistEM)
Aplicando la ecuación (16) se procede a realizar el cálculo de la Distancia Euclidiana del vector cuantificado de características del usuario de prueba (BaseEP) contra los 19 vectores cuantificados de la base de datos BaseE, dando

como resultado 19 distancias Euclidianas diferentes. Así se obtiene un vector de 19 valores (DistEP) a la salida de este módulo.

$$DistEP(k) = \sqrt{\sum_{i=0}^{N} (BaseE(k) - BaseEP)^2} \quad (16)$$

Ahora se procede a calcular el valor promedio de la Distancia Euclidiana del vector DistEP, el cual entrega una medida de cuan similar o diferente es el vector de características del usuario de prueba BaseEP y la base de datos del sistema BaseE, siendo este resultado la variable ***DistEM.***

9. Análisis de Identidad

Este módulo es el encargado de comparar las Distancias de Hamming y Euclidianas obtenidas del usuario de prueba, contra los valores de los umbrales cargados en la base de datos de usuarios permitidos por el sistema. En el numeral 4.5 se mostrará más específicamente el funcionamiento de este módulo.

10. Resultado

Es la respuesta final del sistema después de haber realizado el análisis y comparación del iris del usuario a validar con la base de datos interna del sistema. Las posibles respuestas son:

Usuario Autorizado: Significa que, tras una comparación, el usuario que se ingresó para validar, pertenece a la base de datos, por lo tanto es permitido por el sistema.

*Usuario no Autorizado: S*ignifica que, tras una comparación, el usuario que se ingresó para validar, no pertenece a la base de datos, por lo tanto no es permitido por el sistema.

4.3 Segmentado del Iris

En la Figura 36, la cual corresponde al diagrama de bloques del módulo de Segmentado de Iris, se explicará más a fondo el funcionamiento de este módulo.

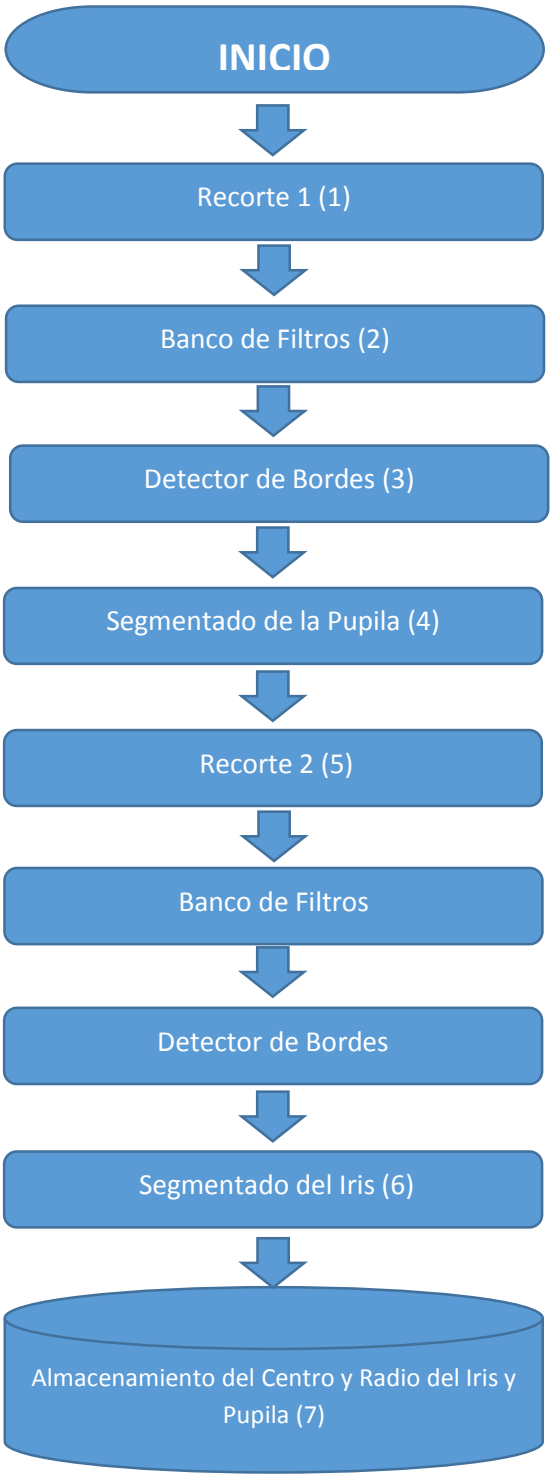

Figura 36. Diagrama de Flujo Módulo “Segmentado del Iris”.

Recorte 1

Este módulo recibe una imagen de tamaño 640x480 y entrega una imagen de tamaño 340x210. El objetivo de este recorte es reducir el tamaño de la imagen a procesar y así mejorar la velocidad de ejecución tanto en MatLab® como en el DSP; además este recorte se realiza eliminando zonas de poco interés para el sistema, enfocando el iris y la pupila del usuario, esto gracias a que la base de datos Casia Iris Database 4.0 ofrece imágenes donde el ojo del usuario se encuentra en el centro de la imagen, haciendo posible, de este modo, reducir el tamaño de la imagen al eliminar los bordes de la imagen. Así, teniendo en cuenta lo anterior, se determinó el tamaño de imagen de salida descrito anteriormete.

1. Banco de Filtros

Este módulo recibe la imagen recortada del módulo anterior de tamaño 340x210 y a su salida entrega otra imagen del mismo tamaño, a la cual se le ha aplicado un filtrado espacial. Este módulo hace uso de dos *Filtros de Promedio Circular* (Pillbox, como se conoce en inglés) en configuración cascada, los cuales tienen el objetivo de resaltar los bordes circulares de la imagen.

En la Tabla 3, se muestran los valores de la máscara de tamaño 5x5 usada por el Filtro de Promedio Circular implementado en el Banco de Filtros, y en la Figura 36, se muestra la forma de la campana de este filtro.

0	0,0170159	0,038115	0,0170159	0
0,0170159	0,0783814	0,0795775	0,0783814	0,0170159
0,038115	0,0795775	0,0795775	0,0795775	0,038115
0,0170159	0,0783814	0,0795775	0,0783814	0,0170159
0	0,0170159	0,038115	0,0170159	0

Tabla 3. Máscara del Filtro de Promedio Circular.

Además de realizar el filtrado espacial de la imagen, se le agregó una funcionalidad más a este módulo y es la de determinar las coordenadas (x, y) del pixel de menor intensidad de la imagen resultante; el cual siempre pertenecerá a la región de la pupila. Esto se realiza con el fin de optimizar tiempo de procesamiento y posteriormente este pixel será el Pixel Semilla para el segmentado de la pupila.

2. Detector de Bordes

Este módulo recibe la imagen filtrada por el módulo anterior de tamaño 340x210 y es el encargado de resaltar los bordes circulares, principalmente, resaltar los bordes del iris y la pupila, para su posterior búsqueda.

El método usado para la detección de bordes hace uso del operador de Sobel, el cual, al ser aplicado a la imagen original, genera una nueva imagen, la cual

corresponde a la gradiente de la intensidad de cada pixel de la imagen, entregando una imagen de tamaño 340x210. En la Figura 37 se puede observar la forma de la campana del filtro de promedio círculo implementado.

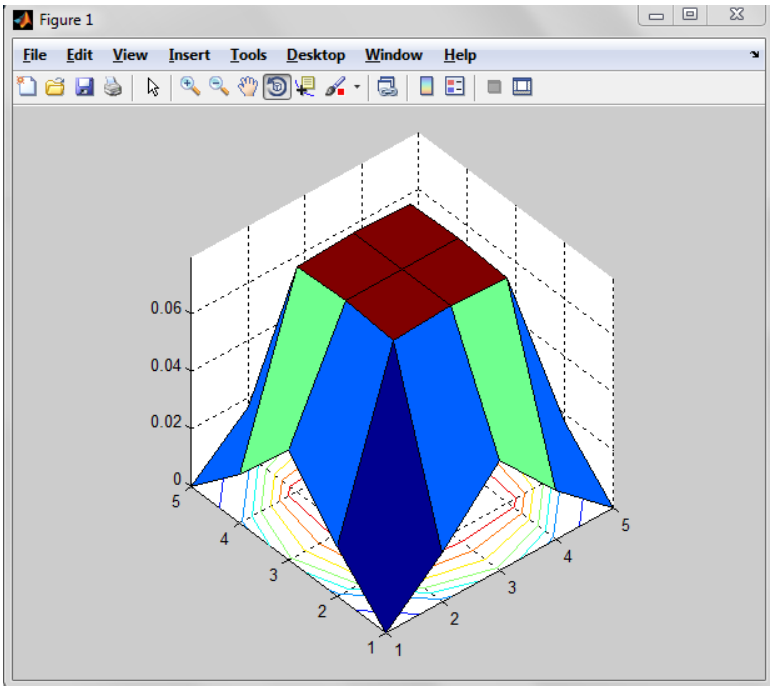

Figura 37. Gráfica de la Campana del Filtro de Promedio Circular Implementado (Pillbox).

3. Segmentado de la Pupila

Este módulo es el encargado de encontrar la posición y radio (x_p, y_p, r_p) pertenecientes al círculo que más se aproxime al contorno de la pupila.

La búsqueda de la región perteneciente a la pupila se inicia con el cálculo de la envolvente convexa, iniciando con el Pixel Semilla y agrupando un conjunto de pixeles que cumplan la condición de que su intensidad esté por debajo de un umbral establecido. Este umbral fue establecido en 0.20 (Valor normalizado de intensidad de 0 a 1) ya que este valor limita el conjunto de pixeles tomados por la envolvente convexa a los de menor intensidad en la imagen.

Como se muestra en la Figura 38, la envolvente convexa no rellena completamente el área total de la pupila, sino que se rellena desde el pixel semilla encontrado hacia abajo, esto debido a que las pestañas que se ubican en la parte superior del ojo, interfieren en la envolvente convexa y genera una zona de la pupila incorrecta.

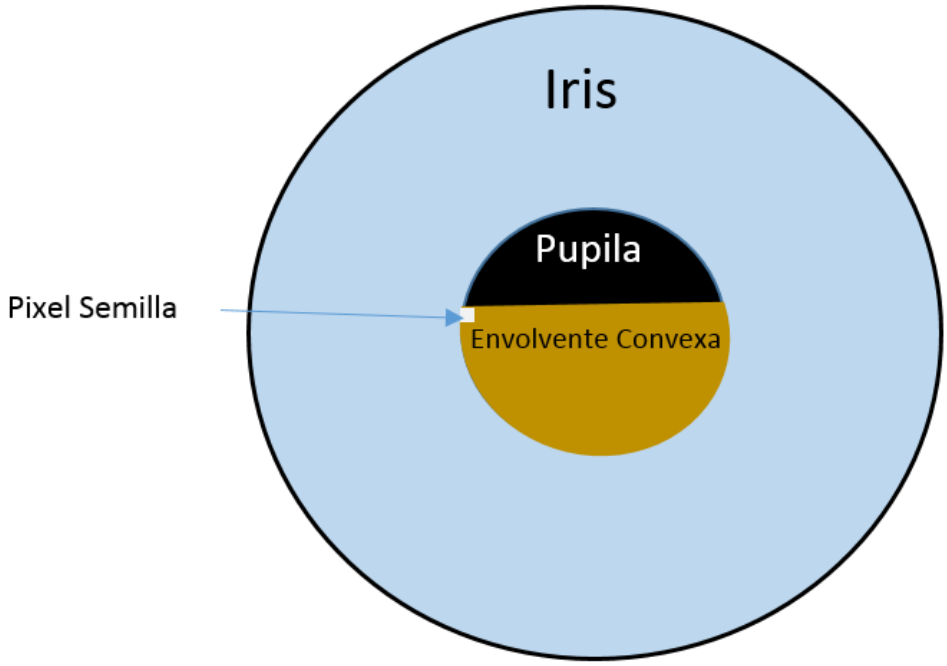

Figura 38. Búsqueda de la Pupila.

Después de obtener la envolvente convexa, se toman tres puntos de la periferia de la región, los cuales son utilizados para calcular las coordenadas del círculo que pasa por estos tres puntos. Con esto se obtiene un círculo cuyas coordenadas y radio se aproximan al contorno de la pupila.

Después de realizar el segmentado de la pupila, se procede a procesar de nuevo la imagen original, realizando el recorte, filtrado, detección de bordes y segmentado del iris. Este procedimiento debe repetirse ya que al realizar el Recorte 1, (recorte estático) en algunos casos algunas regiones del iris quedan por fuera de la imagen, así, la imagen resultante no es la adecuada para extraer el iris. Para solucionar este problema, simplemente se toma la imagen original de nuevo y se le realiza el Recorte 2, pero ya centrando el recorte en las coordenadas de la pupila, de este modo se garantiza que el iris no será afectado por el recorte.

4. Recorte 2

Este módulo se encarga de realizar un recorte sobre la imagen original, recortando de un tamaño inicial de 640x480 a uno de 241x241; esto, teniendo en cuenta el centro de la pupila, esto nuevamente con el objetivo de reducir el tamaño de la imagen y enfocar la búsqueda del iris sobre una imagen más pequeña, lo que reduce el tiempo de ejecución del algoritmo.

5. Segmentado del Iris

Finalmente y utilizando el algoritmo de búsqueda del borde entre el iris y la esclerótica se procede a buscar el iris. Este algoritmo de búsqueda del iris

consiste en trazar una serie de semicírculos, con centro en la pupila (x_p, y_p) y de radio r_i (radio del iris) variable mayor al radio r_p (Radio de la Pupila). Se inicia con un radio r_i superior al radio r_p, ya que como se observa en las imágenes del ojo, el círculo correspondiente al iris posee un radio mayor que el círculo perteneciente a la pupila, de este modo, la búsqueda del círculo perteneciente al iris, se inicia teniendo como base el radio de la pupila, de este modo, limitando el área de búsqueda del algoritmo.

Posteriormente, se procede a calcular el promedio de intensidad de los pixeles pertenecientes a cada semicírculo; y el semicírculo que presente el promedio mayor, será tomado como el perteneciente al iris; esto es, ya que como se observa en la Figura 39, el borde del iris presenta un nivel de intensidad más alto que el interior y el exterior del iris. El algoritmo que calcula los pixeles correspondientes a cada semicírculo de radio r_i de prueba fue desarrollado usando el Algoritmo de Bresenham para el trazado de líneas en un sistema digital.

Es importante resaltar que los semicírculos trazados son semicírculos orientados hacia el párpado inferior, ya que como se mencionó anteriormente, la interferencia del parpado superior y de las pestañas causan errores y mal funcionamiento de este algoritmo. Es posible desarrollar una algoritmo que supere estos inconvenientes, pero su complejidad y costo computacional se incrementarían significativamente. En la Figura 39 se observa cómo se trazan los semicírculos sobre la imagen del iris procesada por el Detector de Bordes.

6. Almacenamiento Resultados

Como resultado final del proceso de segmentado del iris, este módulo entrega a su salida las posiciones (x, y) y los radios r del iris y la pupila para su posterior uso por el módulo de Normalización.

4.4 Normalización del Iris

Este módulo es el encargado de convertir la región de interés del iris, que presenta una forma de toroide, en un rectángulo de tamaño nxm como se muestra en la Figura 39.

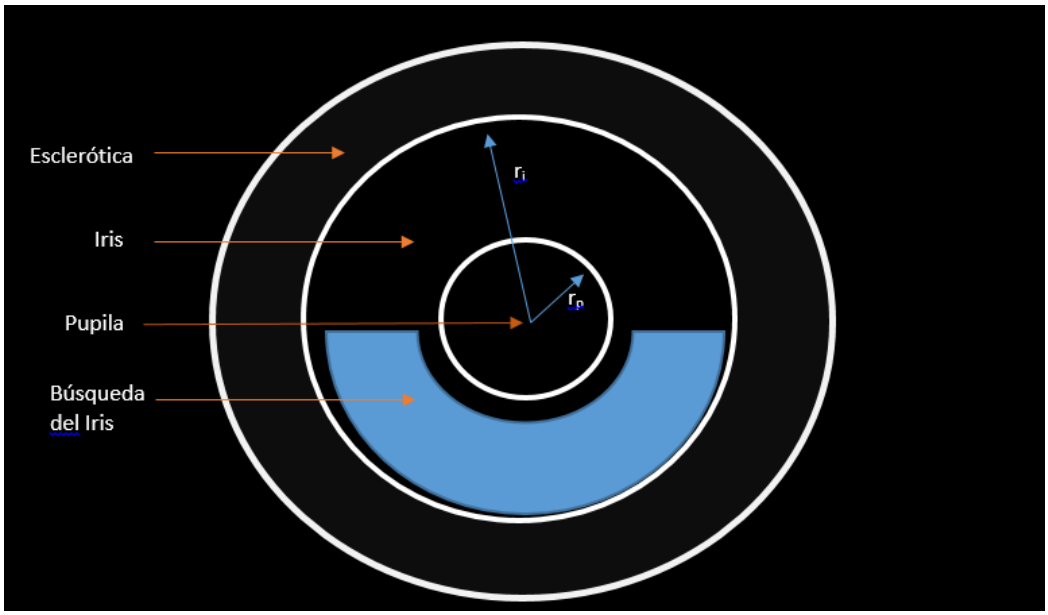

Figura 39. Algoritmo de Búsqueda del Borde Inferior del Iris.

El objetivo de transformar esta región de interés es crear una imagen que represente el iris del usuario de la mejor manera posible; además al realizar el mapeo a coordenadas cartesianas, es mucho más fácil su tratamiento digital, al ser tratada la región como una matriz simple. En la Figura 40 se puede observar mejor el proceso que se realiza con la normalización del iris.

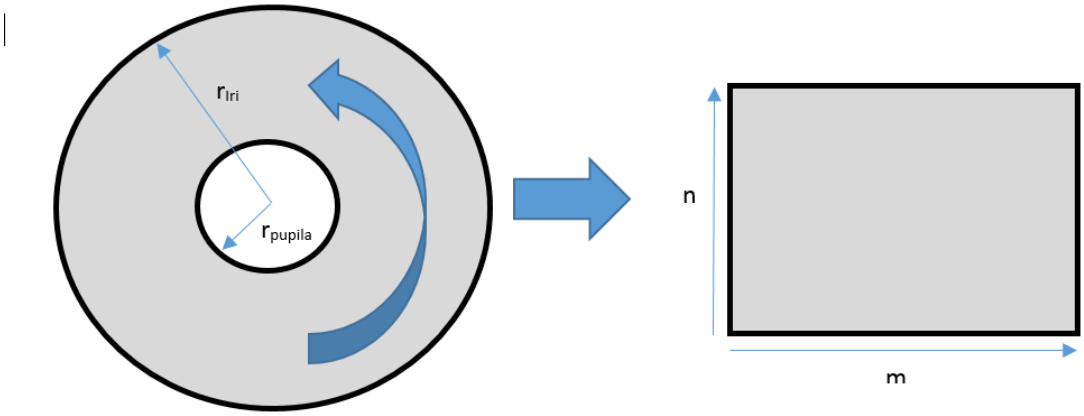

Figura 40. Cambio de Coordenadas Polares a Coordenadas Rectangulares de la Región del Iris. Donde n y m es el tamaño de la imagen de salida.

Para realizar esta normalización el módulo recibe la imagen original del usuario a segmentar, las coordenadas y el radio *(x, y, r)* de la pupila y del iris y el tamaño nxm de la imagen que se desea a la salida.

La región de interés a normalizar está comprendida entre el radio de la pupila y el radio del iris, y como es evidente, esta región tiene una forma de toroide, la cual debe ser mapeada a una cuadricula de tamaño nxm. Teniendo los radios del iris y la pupila, se crean dos matrices de tamaño nxm, de las cuales la primera matriz contiene las coordenadas en x, y la segunda las coordenadas en y de los puntos de interés en la imagen original. (Región comprendidos entre los dos radios). Posteriormente y básicamente usando la ecuación (17), que realiza el cambio de coordenadas polares a cartesianas, se obtienen dos nuevas matrices las cuales contienen cada una las nuevas coordenadas cartesianas de los puntos en *x* e *y.*

$$x = r * \cos(\theta)\,;\; y = r * \sin(\theta) \quad (17)$$

Donde:
(r, θ) son $Coordenadas\ Polares$
(x, y) $son\ Coordenadas\ Cartesianas$

Los puntos (x, y) obtenidos ahora deben corresponder a un valor en escala de grises de la imagen original del iris; sin embargo, estos puntos (x, y) al ser una transformación lineal de otro sistema de coordenadas, poseen valores decimales, y ya que estamos tratando con posiciones en una imagen digital, estos deben ser valores enteros. Debido a lo anterior se hace necesario y para efecto de no introducir ruido o distorsiones a la imagen del iris, se aplica una interpolación bilineal para hallar los valores de intensidad más aproximados correspondientes a cada punto (x, y) entero.

En la Figura 41 correspondiente al diagrama de bloques del módulo de Normalización de Iris, se explica el funcionamiento más a fondo de este módulo.

4.5 Análisis de Identidad

Este módulo es el encargado de evaluar y determinar mediante la comparación de variables si el usuario de prueba está autorizado o no por el sistema. Este módulo recibe las variables DistHM, UmbralH, DistEM y UmbralE, entregando al su salida el resultado de la comparación entre estas variables, como se evidencia el diagrama de bloques de la Figura 42.

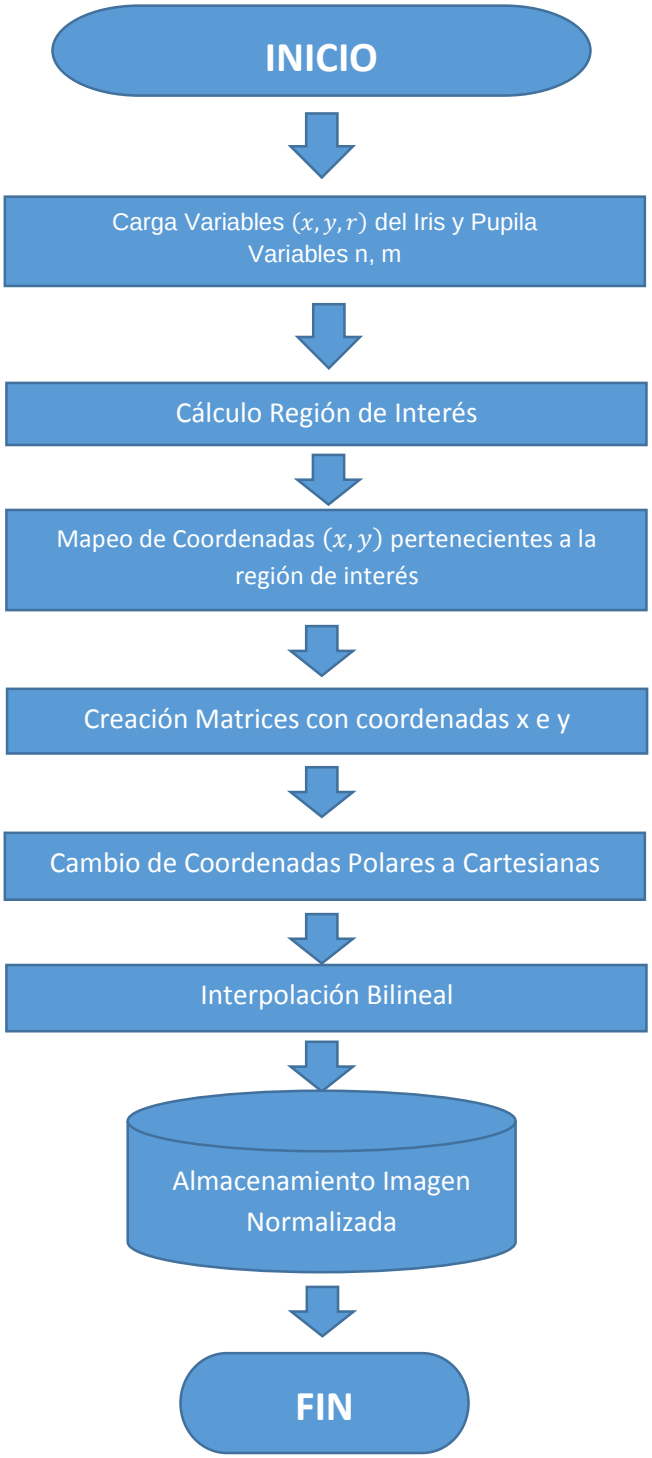

Figura 41. Diagrama de Flujo del Módulo de Normalización

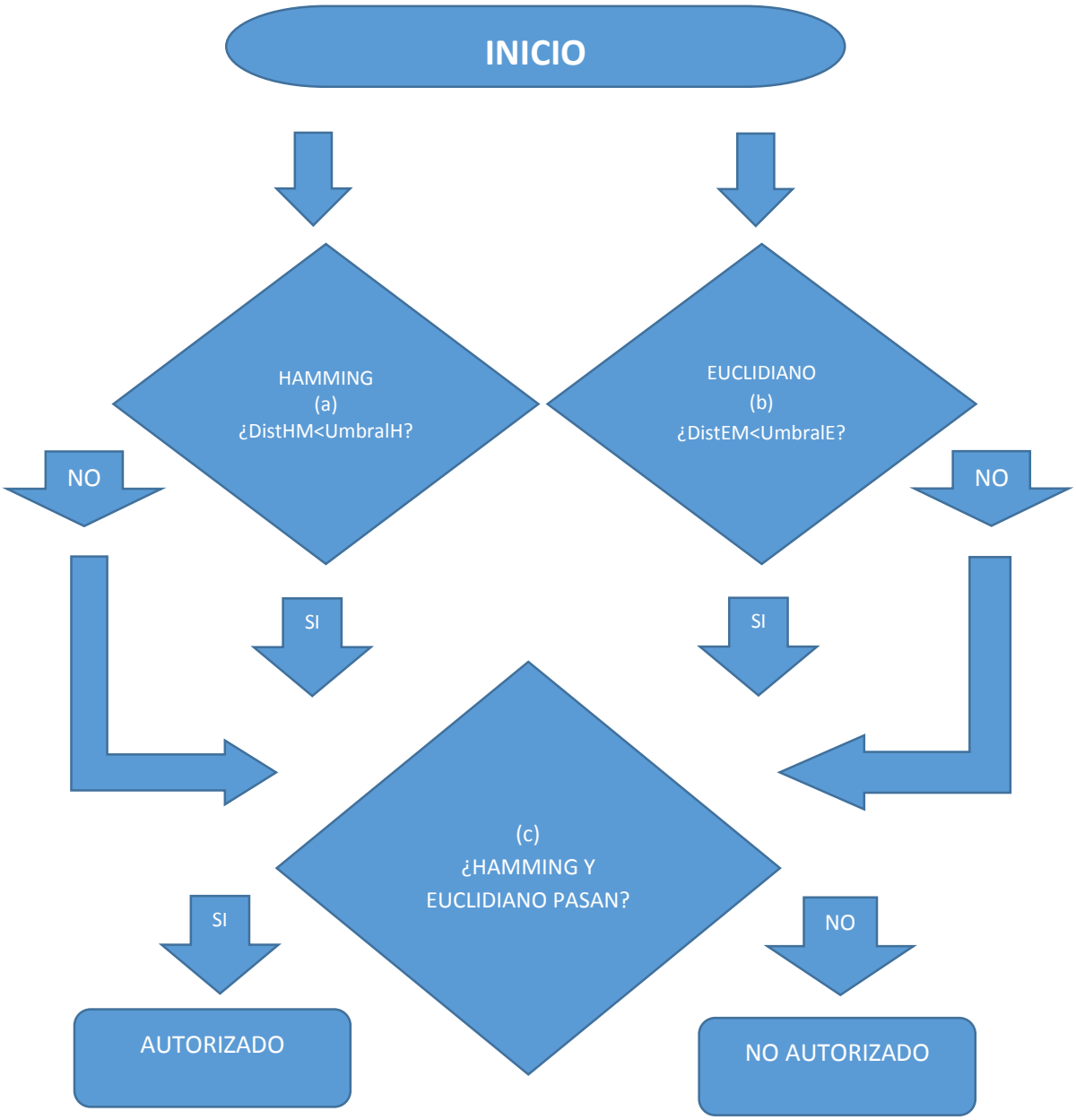

Figura 42. Diagrama de Flujo del Módulo de Análisis de Identidad.

1. Evaluación Umbral Euclidiano

Esta comparación por umbral funciona bajo los mismos principios mencionados anteriormente para la Distancia de Hamming. La Distancia Euclidiana del usuario de prueba (DistEM) debe ser menor a la Distancia Euclidiana del sistema (UmbralE) para ser validado por este módulo. Esto se basa en el principio de que la Distancia Euclidiana es, al igual que la Distancia de Hamming, una medida de cuan similar o diferentes son un conjuntos de

datos; de este modo, el umbral calculado por el sistema es la distancia máxima permitida entre el vector de características del usuario de prueba y la base de datos del sistema. Este módulo emite un resultado binario, siendo 1 Usuario Autorizador por el módulo de Distancia Euclidiana y 0 Usuario No Autorizador por el módulo de Distancia Euclidiana.

2. Evaluación Final

Finalmente, al tener los resultados de los dos módulos descritos anteriormente se procede a tomar la decisión de Autorizar o No Autorizar el usuario de prueba. Básicamente se realiza la operación binaria *AND* entre los resultados de los módulos de Distancia de Hamming y Distancia Euclidiana. De este modo ambos módulos deben autorizar el usuario para que al final, el sistema determine que el usuario de prueba pertenece al usuario registrado en el sistema.

4.6 Cálculo Umbral Distancia de Hamming (UmbralH)

El UmbralH es el umbral máximo permitido para la Distancia de Hamming entre el usuario y la base de datos. Este umbral, a diferencia del UmbralE, el cual es calculado automáticamente usando la base de datos BaseE del usuario registrado; es calculado desarrollando un conjunto de pruebas aleatorias sobre los usuarios utilizados de la base de datos CASIA Iris Database 4.0, de este modo, estas pruebas, entregan un UmbralH el cual es implementado en este Sistema de Reconocimiento de Iris. A continuación se describirán las pruebas realizadas para calcular este umbral.

Para hallar el UmbralH se procedió a realizar un análisis de histograma de las Distancias de Hamming. Para ello se realizaron dos pruebas, obteniéndose 1200 Distancias de Hamming para cada una. La primera prueba (Prueba 1) consiste en comparar la Distancia de Hamming entre la BaseH de un usuario A y el vector binarizado DisHP de la imagen de un usuario de prueba B, obteniéndose el valor de la distancia (DistHM) de un usuario que no hace parte de la BaseH; y la segunda prueba (Prueba 2) consiste en comparar la Distancia de Hamming entre la BaseH de un usuario A y el vector binarizado DisHP de una imagen del mismo usuario A, obteniéndose el valor de la distancia (DistHM) de un usuario que hace parte de la BaseH.

Para la Prueba 1, se tomaron 30 usuarios al azar y se crearon 20 BaseH diferentes, las cuales son comparadas cada una contra dos vectores DisHP de imágenes de usuarios al azar. De este modo se obtienen 1200 valores diferentes DistHM; igualmente para la Prueba 2, se tomaron 30 usuarios al azar y se crean 20 BaseH diferentes, las cuales son comparadas cada una contra dos vectores DisHP de imágenes al azar del mismo usuario. De este modo se obtienen 1200 valores diferentes DistHM.

Teniendo un conjunto de 1200 valores por cada prueba, se procedió a graficar el histograma y curva característica de cada histograma en MatLab® en una misma gráfica, con el objetivo de encontrar el punto de cruce las curvas características de ambos histogramas, obteniéndose la Figura 43.

Teniendo los resultados anteriores, se procedió a realizar un análisis de falso rechazo y falsa aceptación del sistema con 3 puntos alrededor del punto de corte de las curvas características de ambos histogramas, para determinar cual ofrece el mejor rendimiento.

Los puntos escogidos para realizar las pruebas fueron el punto de intercepto de ambas gráficas, el punto mínimo de la gráfica de Usuarios No Registrados (Azul) y el punto medio de ambos puntos. En la Tabla 4 se resume el resultado de las pruebas de falso rechazo y falsa aceptación obtenidas con estos tres umbrales. (Lo conceptos de *Falso Rechazo y Falsa Aceptación* será definidos más adelante en el capítulos de Resultados).

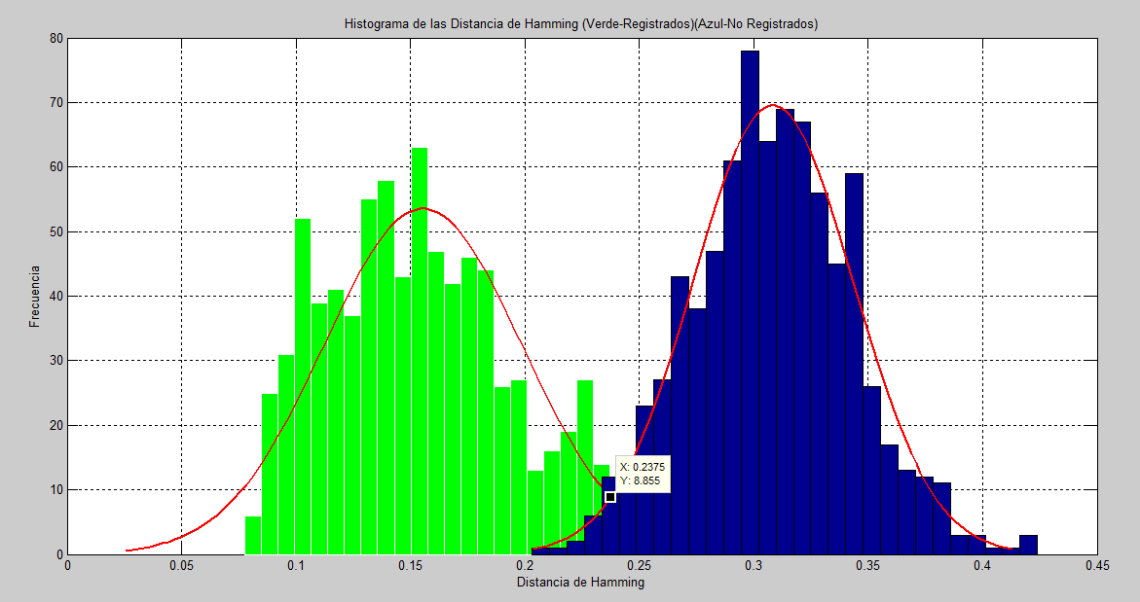

Figura 43. Histograma de las Distancias de Hamming para las Pruebas de Usuarios Registrados en el Sistema (Verde) y de los Usuarios No Registrados en el Sistema (Azul).

	Intercepto	Punto Medio	Mínimo
Autorizados	1113	1087	972
No Autorizados	11	6	0
Umbrales	0,2375	0,2206	0,2036
Falso Rechazo	*7,3%*	*9,4%*	*19,0%*
Falsa Aceptación	*99,1%*	*99,5%*	*100,0%*

Tabla 4. Resultados del Análisis de Confiabilidad para los Umbrales Extraídos del Histograma de Distancias de Hamming.

4.7 Base de Datos de Pruebas CASIA Iris Database V4.0 [28]

Como se ha mencionado, la base de datos utilizada para el desarrollo y pruebas del Sistema de Reconocimiento de Iris es la base de datos CASIA Iris Database 4.0; una base de datos de imágenes de iris de uso académico desarrollada por *Biometrics Ideal Test (BIT)*, que agrupa un conjunto de imágenes de cientos de usuarios, las cuales fueron tomadas desde diferentes distancias, iluminaciones y cámaras; dando como resultado gran variedad de situaciones posibles para las imágenes del iris.

CASIA ofrece seis diferentes sub bases de imágenes del iris, las cuales ofrecen diferentes características en iluminación, resolución de la imagen, número de imágenes, número de usuarios de prueba y distancia a la cámara. A continuación se hará una breve descripción y ejemplo de cada una de las imágenes de las seis diferentes bases de datos disponibles en CASIA Iris Database 4.0

- CASIA Iris Interval

  Imágenes del iris tomadas con una cámara desarrollada por BIT la cual cuenta con iluminación led infrarroja cercana (NIR LED), ofreciendo imágenes de alta resolución del iris. En la Figura 44 se puede observar una imagen del iris de esta base de datos.

- CASIA Iris Lamp

  Imágenes del iris tomadas con una cámara de mano de iluminación variable, la cual permite tomar imágenes desde distintas distancias, condiciones de lumínicas, apertura de pupila del usuario. En la Figura 45 se puede observar una imagen del iris de esta base de datos.

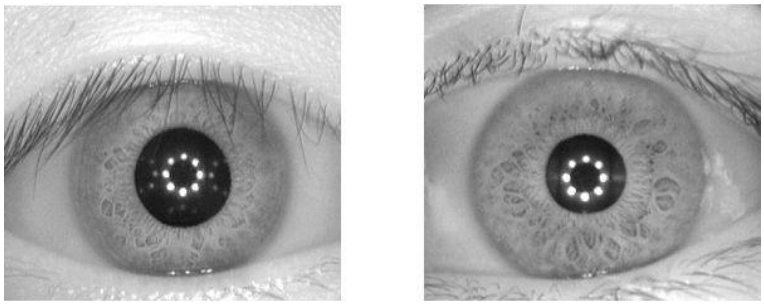

Figura 44. Ejemplo Imagen CASIA Iris Interval. [28].

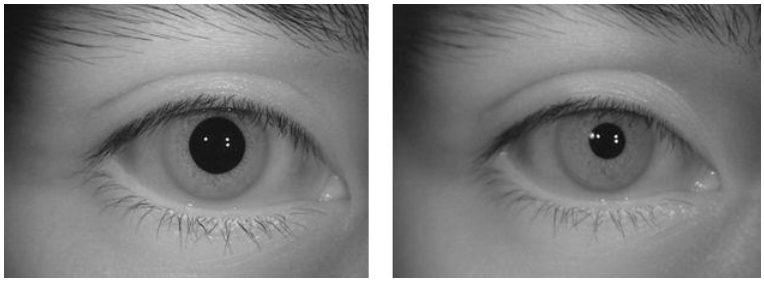

Figura 45. Ejemplo Imagen CASIA Iris Lamp. [28].

- CASIA Iris Twins

Imágenes del iris tomadas a 100 pares de gemelos. En la Figura 46 se puede observar una imagen del iris de esta base de datos.

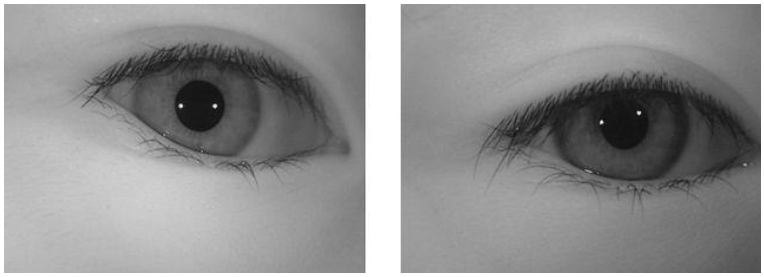

Figura 46. Ejemplo Imagen CASIA Iris Twins. [28].

- CASIA Iris Distance

Imágenes del iris tomadas con una cámara desarrollada por BIT, la cual es capaz de reconocer a una persona a tres metros de distancia, buscar, extraer y tomar una imagen de los dos iris del usuario. En la Figura 47 se puede observar una imagen del iris de esta base de datos.

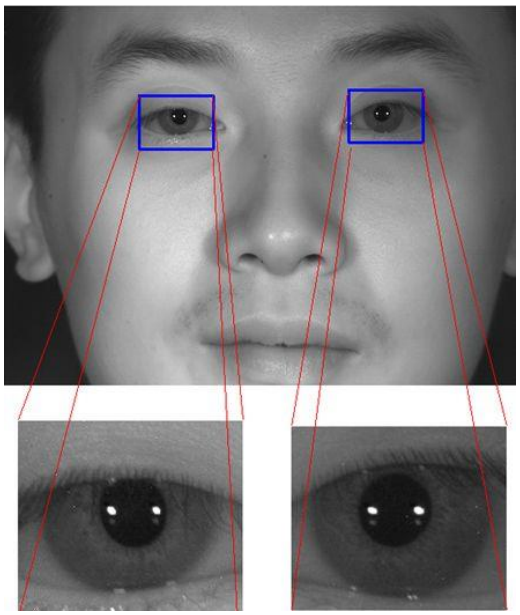

Figura 47. Ejemplo Imagen CASIA Iris Distance. [28].

- CASIA Iris Thounsand.

Imágenes del iris tomadas por la cámara desarrollada por IrisKing. Esta base ofrece 20000 imágenes de 1000 usuarios diferentes. En la Figura 48 se puede observar una imagen del iris de esta base de datos.

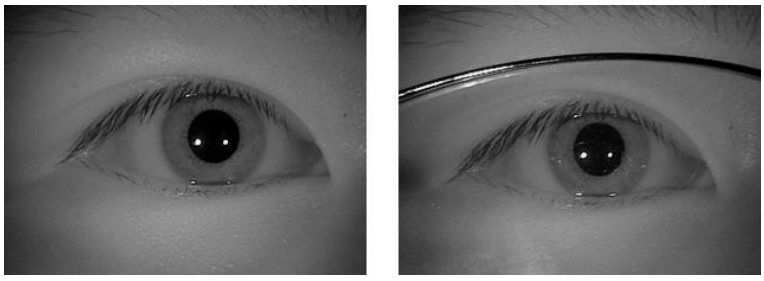

Figura 48. Ejemplo Imagen CASIA Iris Thousand. [28].

- CASIA Iris Syn

Imágenes del iris sintetizadas en las cuales se les agregan diferentes características de textura como ruido y distorsión para simular condiciones reales. En la Figura 49 se puede observar una imagen del iris de esta base de datos.

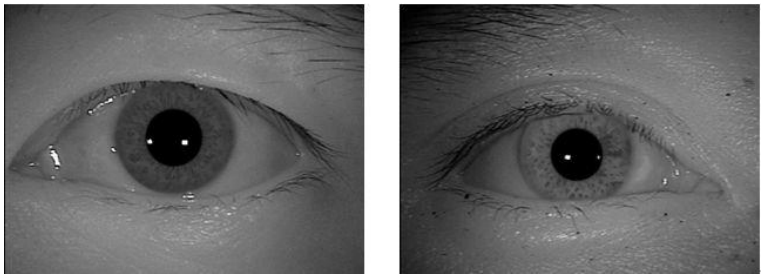

Figura 49. Ejemplo Imagen CASIA Iris Syn. [28].

CASIA Iris Database V4.0 tiene un conjunto de sub bases de datos de diferentes características, las cuales ofrecen un amplio espectro de imágenes del iris, permitiendo al desarrollador de aplicaciones o sistemas embebidos, diseñar y probar el funcionamiento del sistema en diferentes condiciones, tanto lumínicas como de cámaras especializadas. En la Tabla 5, se puede observar más resumida y detalladamente las diferentes características técnicas de las seis bases de datos pertenecientes a CASIA Iris Database V4.0 ofrecida por BIT.

Como se puede observar en la Tabla 5, la base de datos cuenta con una gran cantidad de imágenes disponibles con diferentes características; por lo cual se hizo necesario escoger entre una de las seis sub bases de imágenes disponibles para diseñar y realizar las pruebas de confiabilidad del sistema de reconocimiento de iris. Observando las diferentes sub bases y sus especialidades, se decidió escoger la sub base *CASIA Iris-Lamp,* esto debido a que esta ofrece un conjunto de imágenes donde la iluminación, alineación del ojo a la cámara, y apertura de pupila son variables; de este modo, simulando las posibles imágenes que se obtendrían en una implementación de prototipo de sistema de reconocimiento de iris con captura de imágenes en tiempo real.

Esta base de datos cuenta con un total de 411 usuarios, a los cuales se les tomaron entre 19 y 20 imágenes del iris de cada ojo, dando un total de 16212 imágenes diferentes. Debido a la gran cantidad de imágenes disponibles y el alto tiempo de procesamiento que conllevaría trabajar con todas ellas, se decidió tomar una muestra más pequeña de los usuarios de la base de datos para realizar las pruebas de confiabilidad en MatLab®; de este modo se escogieron 30 usuarios al azar para conformar la base de datos de pruebas del sistema. Además, se decidió solo trabajar con uno de los ojos del usuario, en este caso, se escogió el ojo izquierdo como base de extracción del iris.

Base de Datos	CASIA Iris Interval	CASIA Iris Lamp	CASIA Iris Twins	CASIA Iris Distance	CASIA Iris Thousand	CASIA Iris Syn
Sensor	CASIA close-up iris camera	OKI IRISPASS-H	OKI IRISPASS-H	CASIA long-range iris camera	Irisking IKEMB-100	CASIA iris image synthesis algo
Ambiente	Espacio Cerrado	Espacio Cerrado con Lámpara Encendida/Apagada	Espacio Abierto	Espacio Cerrado	Espacio Cerrado con Lámpara Encendida/Apagada	N/A
Sesión	Dos sesiones para la mayoría de imágenes	Una	Una	Una	Una	Una
Atributos de los Sujetos	La mayoría son estudiantes graduados de CASIA	La mayoría son estudiantes graduados de CASIA	La mayoría son niños participantes del Beijing Twins Festival	La mayoría son estudiantes graduados de CASIA	Estudiantes, trabajadores, agricultores, con gran rango de distribución de edad	Imágenes tomadas de CASIA Iris V1
N° de Sujetos	249	411	200	142	1000	1000
N° de Clases	395	819	400	248	2000	1000
N° de Imágenes	2639	16212	3183	2567	20000	10000
Resolución	320x280	640x480	640x480	2352x1728	640x480	640x480
Características	Alta Definición de los detalles de textura del Iris	Deformación no linear debido a las variaciones de iluminación	La primera publicación de un base de datos de Iris de gemelos	La primera publicación de un base de datos de alta definición de iris y cara de rango amplio	La primera publicación de un base de datos con más de 1000 sujetos	Base de datos de Iris Sintetizados
Total	Un total de 54601 imágenes del iris de más de 1800 sujetos reales y 1000 sujetos artificiales					

Tabla 5. Características Técnicas de la Base de Datos CASIA Iris Database V4.0 de BIT [28].

Así, al final se obtiene una base de datos de 30 usuarios diferentes, los cuales poseen 20 imágenes diferentes del ojo izquierdo, generando un conjunto de 600 imágenes diferentes para el desarrollo de las pruebas de confiabilidad del sistema.

5
Implementación en MatLab®

En este capítulo se mostrará el desarrollo de las implementaciones del sistema de reconocimiento de iris desarrollados en MatLab® y su diagrama de bloques es el mostrado en la Figura 33. A continuación se procederá a mostrar en ambos casos todos los procesos y transformaciones que deben sufrir las imágenes y datos en el paso por los módulos de Registro Usuario y Validación Usuario.

5.1 Descripción General

El sistema de reconocimiento de iris fue implementado en MatLab® desarrollando los algoritmos al más bajo nivel posible, es decir, sin utilizar las funciones disponibles en los toolbox de procesamiento de imágenes y de Transformada de Wavelet; esto con el fin de que los algoritmos y métodos desarrollados e implementados para este sistema sean lo más posiblemente parecidos tanto en MatLab® como en el DSP.

A continuación se mostrará el proceso paso a paso por el cual una imagen es procesada por el Sistema de Reconocimiento de Iris Implementado en MatLab®; el cual consta de dos módulos principales, Registro Usuario y Validación Usuario.

5.2 Registro De Usuario

El módulo de registro usuario recibe un conjunto de 19 imágenes, las cuales son usadas para crear la base de datos del usuario registrado por el sistema. A continuación se mostrará el proceso para una sola imagen, sin embargo, este proceso se repite para las 19 imágenes del usuario a registrar.

1. **Carga de las Imágenes**

Se carga a MatLab® la imagen proveniente de la base de datos CASIA Iris Database 4.0 del usuario que se desea registrar. En la Figura 50 se muestra una de las de las 19 imágenes a procesar.

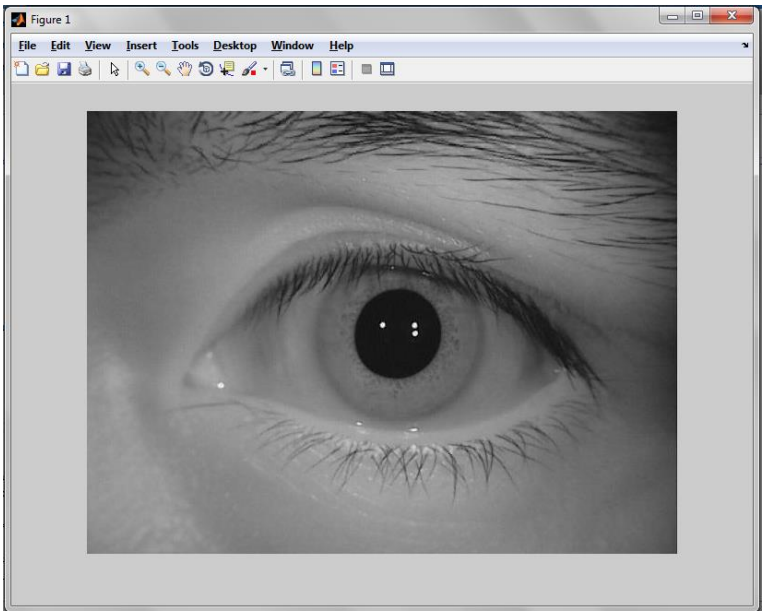

Figura 50. Imagen Original Cargada de la Base de Datos CASIA Iris Database 4.0.

2. Segmentado del Iris

Este módulo recibe la imagen cargada anteriormente y se encarga de encontrar el área de interés, el iris, comprendida entre los círculos concéntricos de la pupila y el iris.

En la implementación en MatLab® se desarrolló la función ***IrisSN.m,*** la cual es la encargada de realizar la segmentación y normalización de las imágenes del iris. La función recibe el número del usuario en la base de datos y el número de la imagen del usuario a cargar; y entrega una imagen del iris normalizada de tamaño $256x64$. A continuación se procederá a describir los diferentes módulos que componen esta función, tanto los de segmentación como los de normalización.

2.1 Recorte 1

Se recorta el área cercana al ojo como se observa en la Figura 51, disminuyendo el tamaño total de la imagen para mejorar el rendimiento de los algoritmos de segmentación del iris y pupila. Para esta tarea se creó la función "*Recortar.m*"

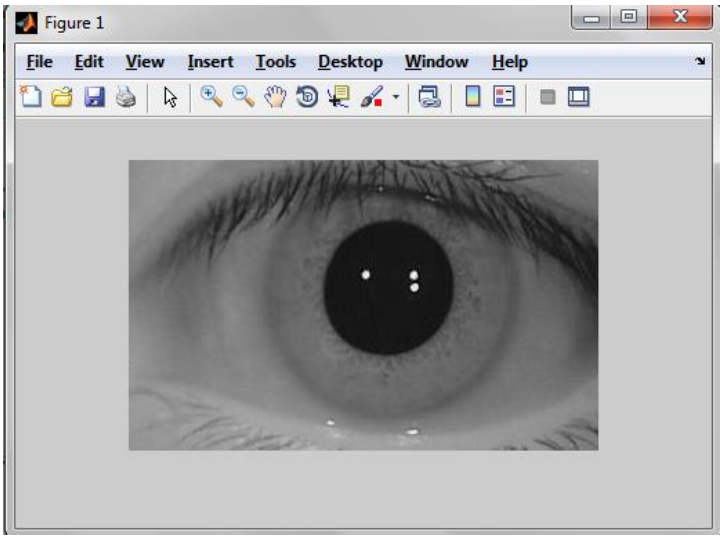

Figura 51. Imagen Después del Recorte 1.

2.2 Banco de Filtros 1

El objetivo de este Banco de Filtros de Promedio Circular (Pillbox en inglés) es resaltar los patrones circulares en la imagen para que el algoritmo Detector de Bordes resalte los bordes circulares de la pupila y disminuir la influencia de los bordes horizontales y verticales presentes en la imagen. Para esta tarea se creó la función "*Pillbox.m*"

2.2.1 Filtro de Promedio Circular 1

En la Figura 52 se observa el efecto del primer filtrado de la imagen.

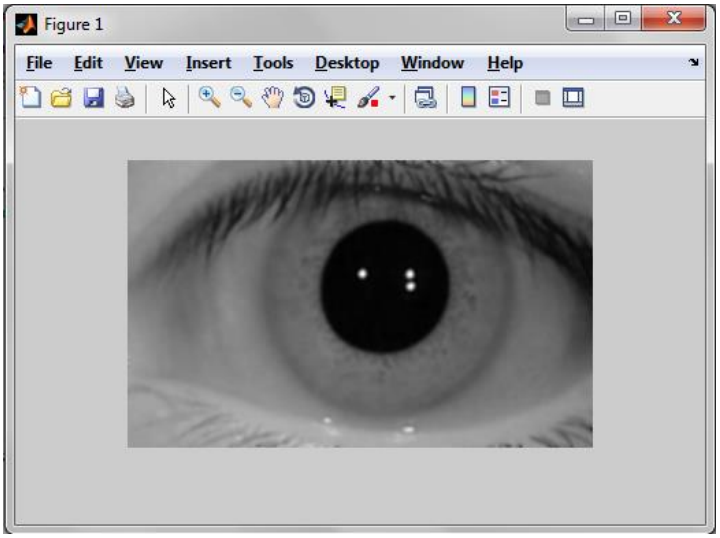

Figura 52. Imagen Filtrada por Filtro de Promedio Circular 1 del Banco de Filtros 1.

2.2.2 Filtro de Promedio Circular 2

En la Figura 53 se observa el efecto del segundo filtrado de la imagen.

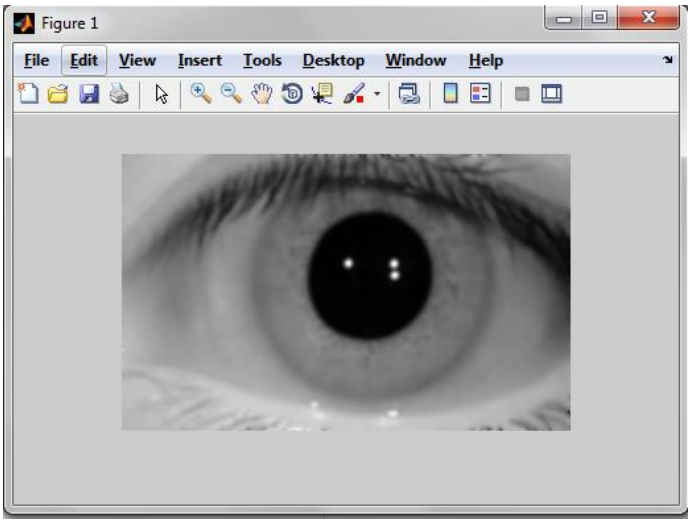

Figura 53. Imagen Filtrada por Filtro de Promedio Circular 2 del Banco de Filtros 1.

2.3 Detector de Bordes

Este módulo hace uso del operador Sobel, el cual filtra la imagen proveniente del banco de filtros y entrega la imagen de la Figura 54, la cual corresponde a la gradiente de los valores de intensidad de los pixeles de la imagen. Esta gradiente resalta los bordes de la imagen, es especial, los bordes circulares, los cuales fueron destacados por el filtrado anterior y son de especial interés para el posterior segmentado de la pupila. Para esta tarea se creó la función “Gradiente.m”

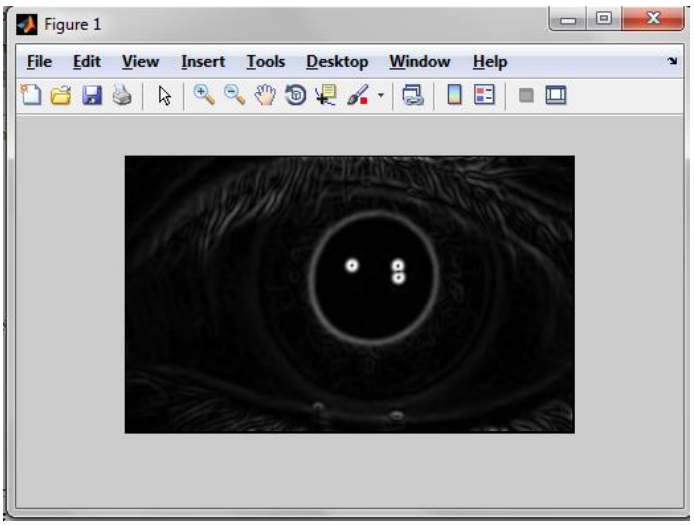

Figura 54. Imagen Después del Detector de Bordes.

2.4 Segmentado de la Pupila

Finalmente y utilizando los algoritmos de crecimiento de región, envolvente convexa y búsqueda del pixel más oscuro en la imagen, se procede a realizar la búsqueda de las coordenadas y radio (x, y, r) del círculo correspondiente a la pupila, dando como resultado la Figura 55. Para esta tarea se creó la función "*EncontrarCentroPupila.m*"

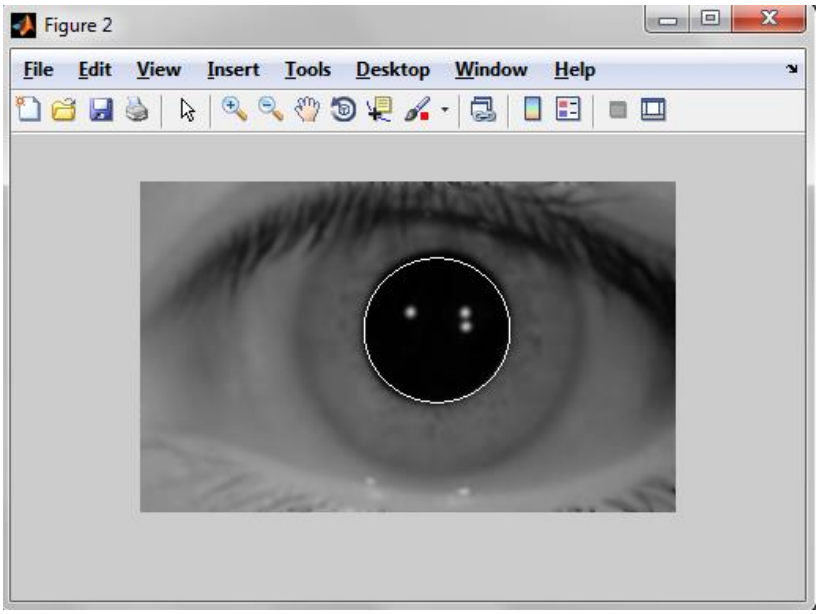

Figura 55. Imagen de la Segmentación de la Pupila.

2.5 Recorte 2

Después de haberse hallado las coordenadas de la pupila, se procede a realizar un nuevo recorte sobre la imagen inicial de la Figura 50. Es de anotar que para mejorar el rendimiento y velocidad de ejecución del algoritmo, se asume que el centro del círculo correspondiente al iris (x_i, y_i), es el mismo hallado anteriormente para la pupila (x_p, y_p). De este modo, este nuevo recorte centra la imagen en las coordenadas del centro de la pupila (x_p, y_p) Para esta tarea se utilizó de nuevo la función "*Recortar.m*", obteniéndose así la Figura 56.

2.6 Banco de Filtros

Nuevamente se procede a filtrar la imagen para resaltar los bordes circulares, esta vez, con el objetivo de resaltar los bordes circulares del iris. El proceso de

filtrado es exactamente el mismo que fue mencionado en el módulo de Banco de Filtros 1. Para esta tarea utilizó de nuevo la función "*Pillbox.m*"

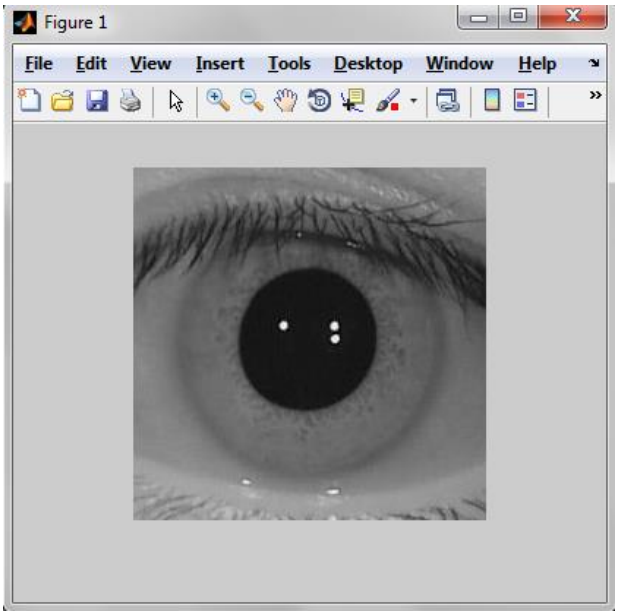

Figura 56. Imagen Original Después del Recorte 2.

2.1.1 Filtro de Promedio Circular 1

En la Figura 57 se observa el efecto del primer filtrado de la imagen.

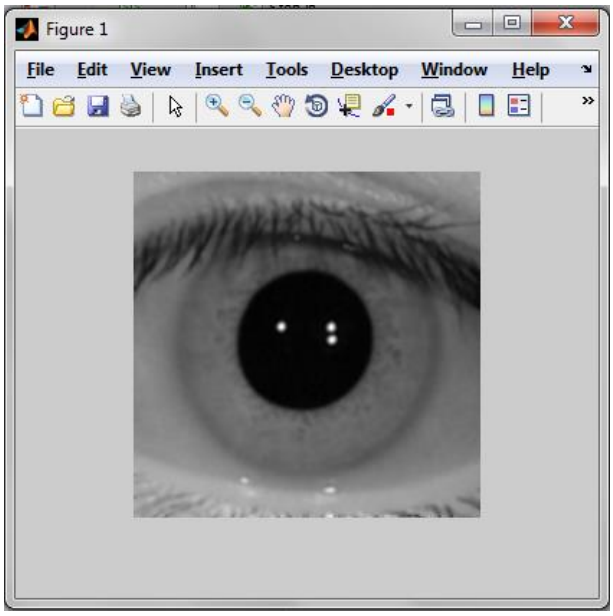

Figura 57. Imagen Filtrada por Filtro de Promedio Circular 1 del Banco de Filtros 2.

2.1.2 Filtro de Promedio Circular 2

En la Figura 58 se observa el efecto del segundo filtrado de la imagen.

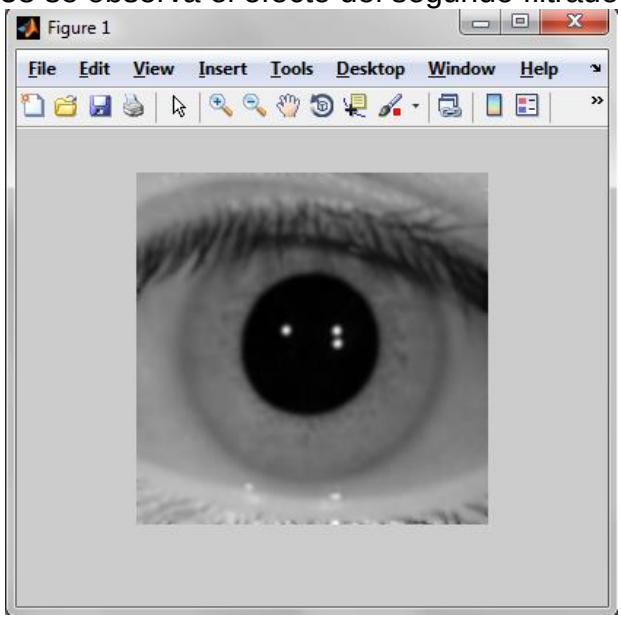

Figura 58. Imagen Filtrada por Filtro de Promedio Circular 2 del Banco de Filtros 2.

2.2 Detector de Bordes 2

De nuevo se aplica el operado Sobel sobre la imagen anteriormente filtrada, para calcular la gradiente de la imagen y resaltar los bordes circulares, esta vez, con el objetivo de resaltar los bordes del iris. Para esta tarea se utilizó de nuevo la función "*Gradiente.m*", obteniéndose la imagen de la Figura 59.

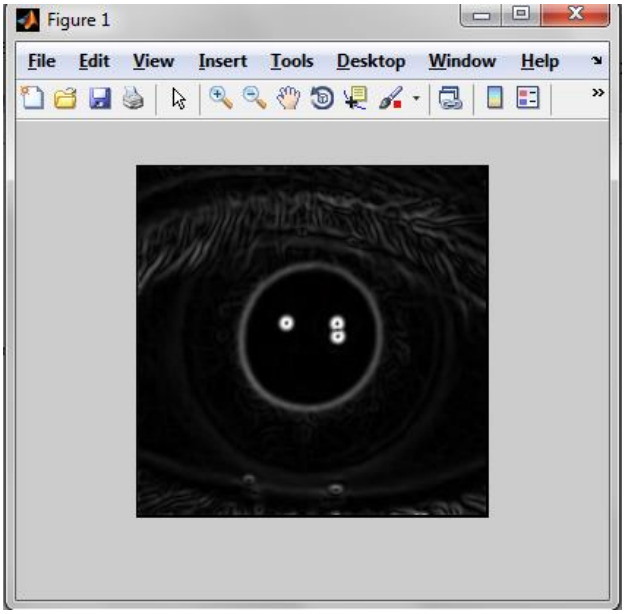

Figura 59. Imagen Después del Detector de Bordes 2.

2.3 Segmentado del Iris

Este módulo es el encargado de encontrar la posición y radio (x_i, y_i, r_i) pertenecientes al círculo que más se aproxime al contorno del iris. Esto lo realiza trazando semicírculos de radio r_{iris} comenzando con el radio menor como el radio de la pupila.

Teniendo en cuenta que después del detector de bordes, los bordes quedan con un valor de intensidad más alto, el algoritmo traza los semicírculos y promedia el valor de intensidad de los pixeles pertenecientes al semicírculo, de este modo, el círculo que posea el promedio más alto, es tomado como el borde del círculo del iris. Para esta tarea se creó la función "*EncontrarIris.m*".

En la Figura 60 se puede observar el resultado del algoritmo anteriormente descrito, encontrando de manera muy acerada para este ejemplo, el círculo perteneciente al iris.

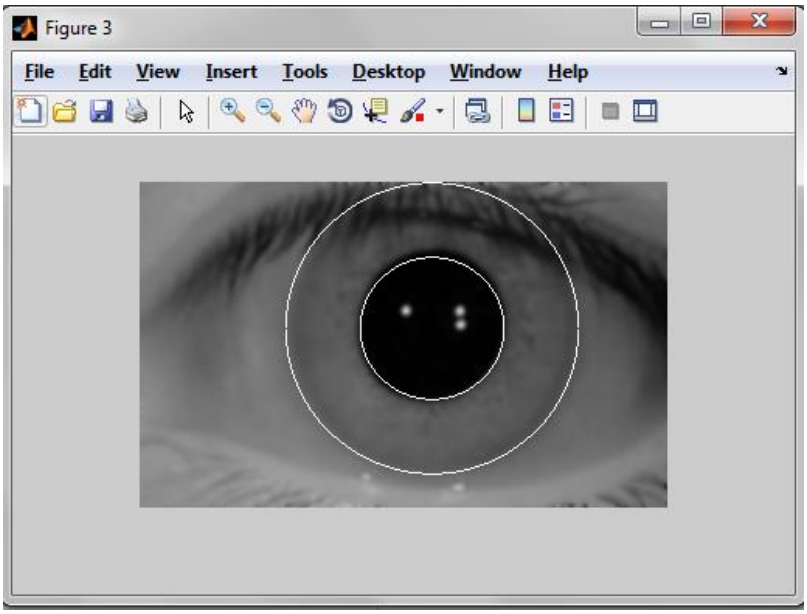

Figura 60. Imagen de la Segmentación del Iris Final.

3 Normalización del Iris

En la Figura 61, se puede observar el resultado de aplicar la normalización del área perteneciente al iris. Esto con el objetivo de extraer la región de interés y convertirla de una región con forma toroidal, a una región rectangular; de este

modo facilitando su procesamiento con posteriores técnicas de caracterización. Para esta tarea se creó la función "*Normalización.m*"

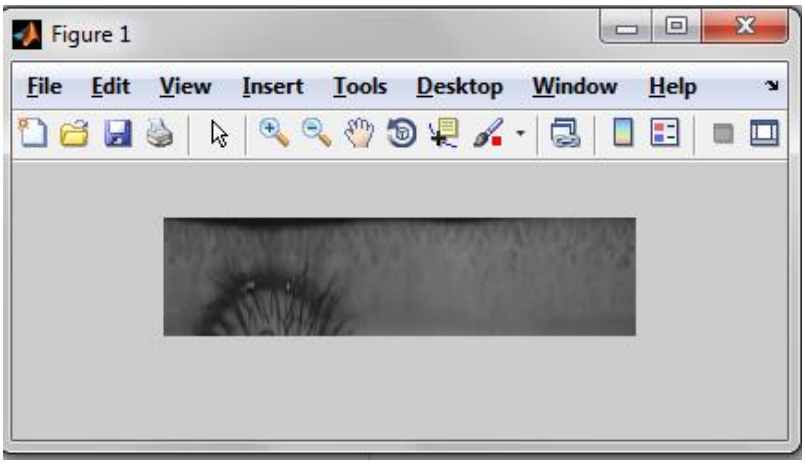

Figura 61. Imagen Normalizada del Iris.

4 Transformada Wavelet de Haar 2D

En este módulo se procede a tomar la imagen normalizada en la Figura 61, la cual pasa por dos procesos principales, el de Recorte y el de la transformada como tal.

4.1 Recorte

En este módulo de recorte, se recibe la imagen de la Figura 61, proveniente de la normalización y con un tamaño de 256x64 y es recortada la parte izquierda de la imagen, dejando como resultado la parte derecha de la imagen, con un tamaño de 128x64. Este recorte es necesario ya que la parte izquierda de la imagen normalizada contiene las pestañas y los párpados del usuario, introduciendo ruido a la caracterización. De este modo, se elimina las posibles interferencias y se garantiza una imagen del iris del usuario lo más limpia posible. En la Figura 62 se puede observar el resultado de este recorte, resultando una imagen sin pestañas ni párpados.

El tamaño final de la imagen de 128x64, fue determinado, ya que la Transformada Wavelet de Haar 2D debe realizarse para matrices con tamaños pertenecientes a potencias de 2.

4.2 Transformada Wavelet de Haar 2D

En la Figura 63, se puede observar el resultado de la Transformada Wavelet de Haar 2D de 4° nivel implementada sobre la imagen recortada del iris normalizada. Esta transformada fue implementada para realizar una caracterización robusta de los patrones del iris. El resultado de esta transformada se puede observar en la Figura 64, la cual es la región de interés

de tamaño 16x8, la cual entrega un vector de características de 128 coeficientes. Para esta tarea se creó la función "*WavelHaar.m*".

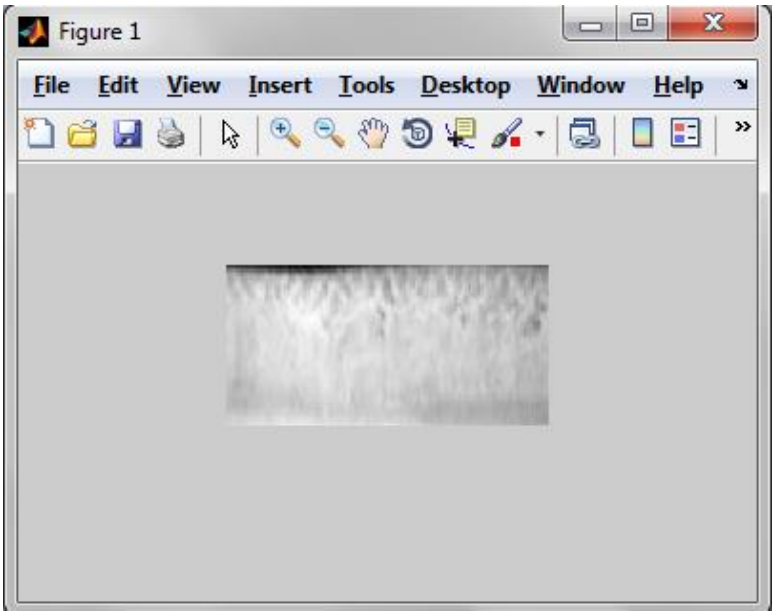

Figura 62. Imagen Normalizada Después del Recorte 3.

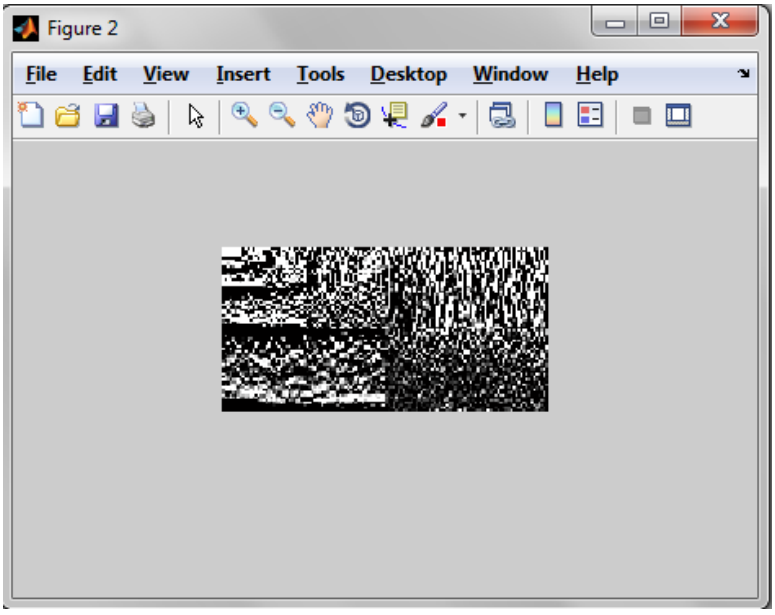

Figura 63. Transformada Wavelet de Haar de 4° Nivel.

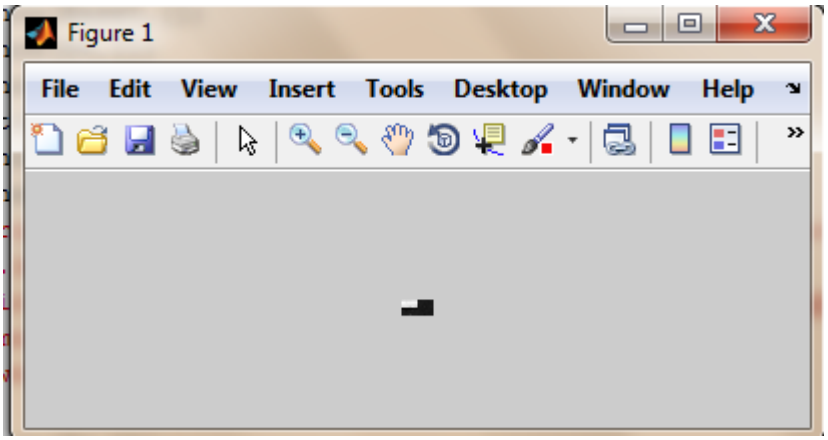

Figura 64. Coeficientes de 4° Nivel la Transformada Wavelet de Haar 2D.

5. Extracción de Características

Posteriormente se obtiene el vector CoefW de 128 valores pertenecientes a los Coeficientes de 4° Nivel de la Transformada Wavelet de Haar 2D de la imagen procesada. Estos coeficientes serán procesados paralelamente por dos métodos distintos, como se mencionó en el capítulo 4, para obtener la base de datos de Distancia de Hamming y Distancia Euclidiana. Para generar dos bases de datos se realizaron los siguientes pasos.

5.1 Codificación Binaria

Se toma el vector de 128 valores de la Transformada Wavelet de Haar 2D y se le aplica la binarización descrita anteriormente, obteniéndose un vector, el cual contiene los 128 valores binarizados. Este proceso se aplica para los 19 vectores obteniéndose una matriz binaria de 19x128 valores, la cual corresponde a la base de datos BaseH, la cual es almacenada por el sistema para ser utilizada en la función *Validación Usuario,* para calcular la Distancia de Hamming con el usuario de prueba.

5.2 Cuantificación

Se toma el vector de 128 valores de la Transformada Wavelet de Haar 2D y se procede a realizar una cuantificación de estos valores ejecutando el proceso descrito en el capítulo anterior, la cual consiste en efectuar un análisis por histograma y determinar los cinco rangos donde se concentra la mayor cantidad de datos, para a continuación cuantificar en cinco valores distintos (1-5) los coeficientes que se encuentren en estos rangos, y los que se encuentren fuera de los rangos, se aproximan al rango más cercano. En la Figura 65 se puede observar el histograma de los coeficientes de la Transformada Wavelet de Haar

2D del usuario a registrar y en la Figura 66 se observa el resultado de la cuantificación de los coeficientes.

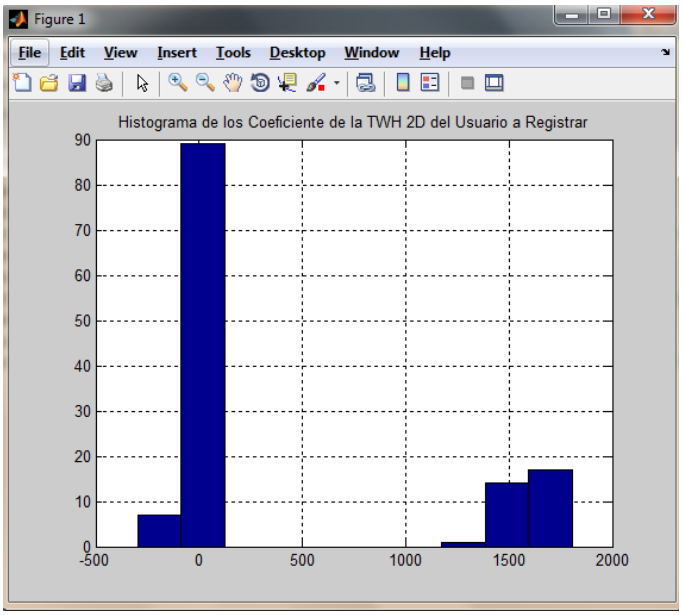

Figura 65. Histograma de los Coeficientes de la Transformada Wavelet de Haar 2D de una de las 19 Imágenes del Usuario a Registrar.

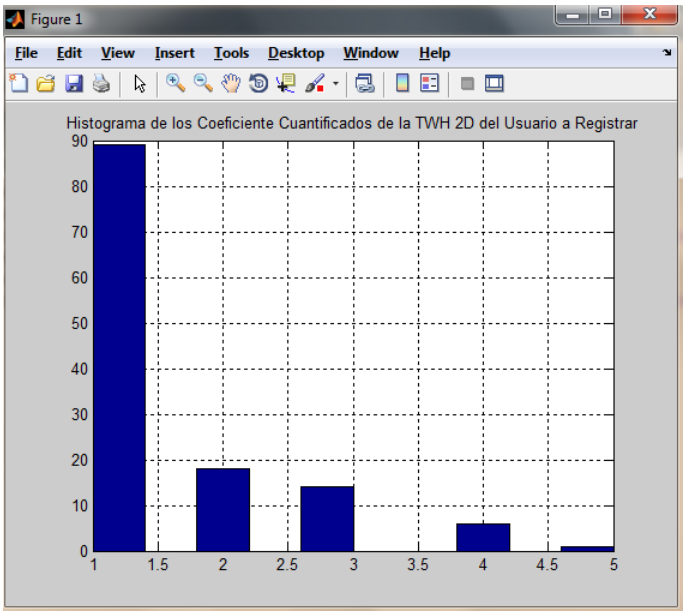

Figura 66. Histograma de los Coeficientes Cuantificados de la Transformada Wavelet de Haar 2D de una de las 19 Imágenes del Usuario a Registrar.

Al final de este proceso se obtiene una matriz de 19x128 valores, llamada BaseE, la cual contiene los 19 vectores cuantificados de los coeficientes de la Transformada Wavelet de Haar 2D de las 19 imágenes del usuario a registrar.

6. Análisis Euclidiano

Ahora se procede a realizar el cálculo automático del umbral de la Distancia Euclidiana a partir de la matriz BaseE calculada anteriormente. Para ello se necesitan hacer los siguientes cálculos.

6.1 Cálculo Distancia Euclidiana (DistE)

Siguiendo el procedimiento del punto 7.1 del numeral 4.2.1 del capítulo 4, se procede a calcular la Distancia Euclidiana entre la misma base de datos, obteniéndose una matriz de 19x19 llamada DistE.

6.2 Cálculo del Promedio de la Distancia Euclidiana (MeanE)

Siguiendo el procedimiento del punto 7.2 del numeral 4.2.1 del capítulo 4, se procede a calcular la el valor promedio de la matriz DistE, el cual determina el valor promedio de las Distancias Euclidianas de la base de datos. Este valor será parte del umbral de Distancia Euclidiana que se usará en la función de Validación Usuario para determinar si un usuario es autorizado o no.

6.3 Cálculo del Promedio de la Desviación Estándar de la Distancia Euclidiana (StdE)

Continuando lo descrito del punto 7.3 del numeral 4.2.1 del capítulo 4, se procede a calcular el valor promedio de la desviación estándar de la matriz DistE, el cual determina la desviación estándar de las Distancias Euclidianas de la base de datos. Este valor será parte del umbral de Distancia Euclidiana que se usará en la función de Validación Usuario para determinar si un usuario es autorizado o no.

6.4 Cálculo UmbralE

Finalmente se calcula el umbral para la Distancia Euclidiana para el usuario que se está registrando a partir de los dos cálculos anteriores. Como se menciona en el punto 7.4 del numeral 4.2.1 del capítulo 4, el UmbralE se calcula al sumar las variables MeanE y StdE.

7. Almacenamiento Base de Datos

La base de datos del sistema para el usuario registrado contiene la matriz binarizada BaseH, la matriz cuantificada BaseE y el umbral de la Distancia Euclidiana UmbralE. Estas tres variables son almacenadas en un archivo .m para su posterior uso por la función Validación Usuario.

5.3 Validación Usuario

La función de Validación Usuario, como se ha mencionado anteriormente, se encarga de determinar si un usuario es autorizado o no por el sistema, haciendo uso de la base de datos creada anteriormente por el sistema. A continuación se procederá a mostrar el funcionamiento de esta función en la implementación desarrollada en MatLab®.

1. Carga de la Imagen

Se carga al sistema la imagen del iris perteneciente al usuario a analizar como se muestra en la Figura 67.

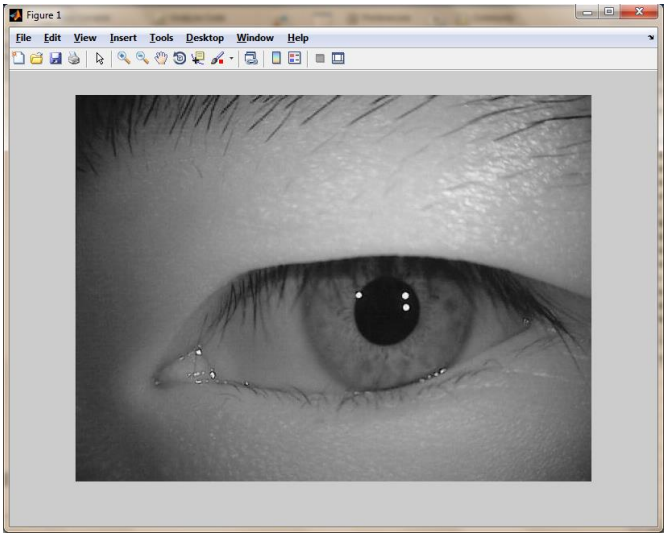

Figura 67. Imagen del Iris del Usuario a Validar.

2. Segmentación y Normalización del Iris.

Posteriormente se utiliza la función *IrisSN.m* para realizar la segmentación y normalización de la imagen del iris a validar, obteniéndose como resultado la Figura 68.

3. Transformada Wavelet de Haar 2D

Con la imagen segmentada y normalizada del iris, se procede a aplicar la Transformada Wavelet de Haar 2D a la imagen del usuario a validar para extraer sus características, de este modo, obteniéndose la Figura 69.

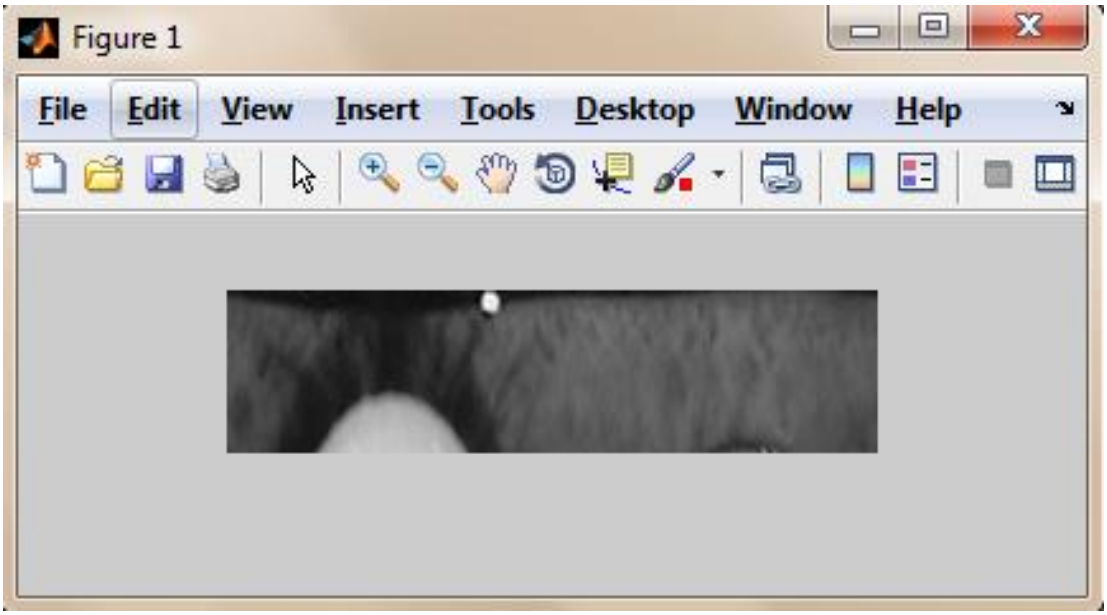

Figura 68. Segmentación y Normalización del Iris del Usuario a Validar.

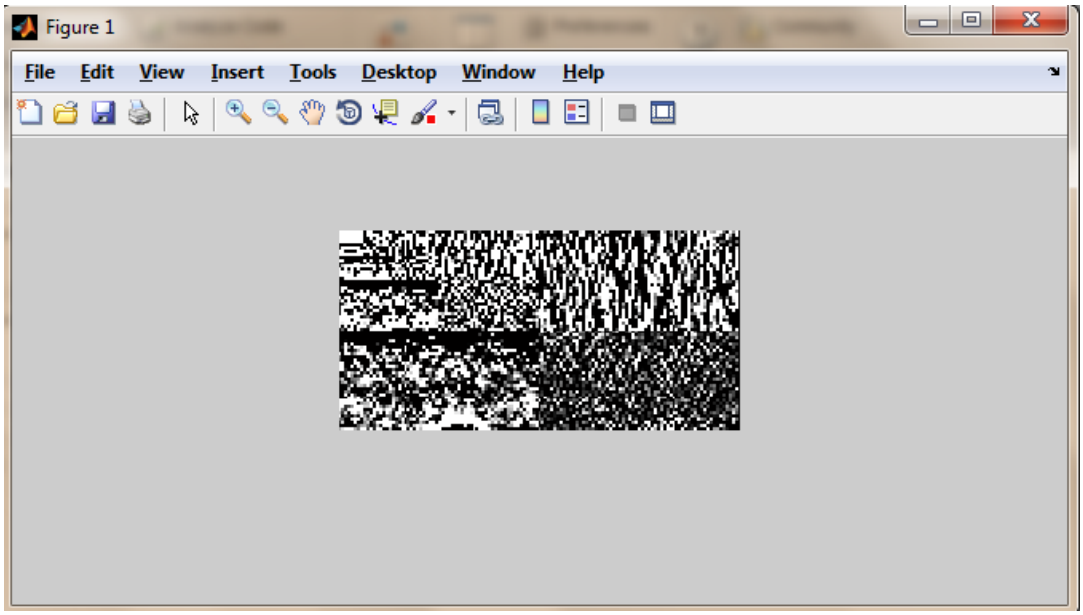

Figura 69. Transformada Wavelet de Haar 2D del Usuario a Validar.

Finalmente, se procede a extraer la región de interés de la Transformada Wavelet de Haar 2D, la cual corresponde a la región comprendida entre los pixeles 1 a 8 en y e 1 a 16 en x, como se observa en la Figura 70.

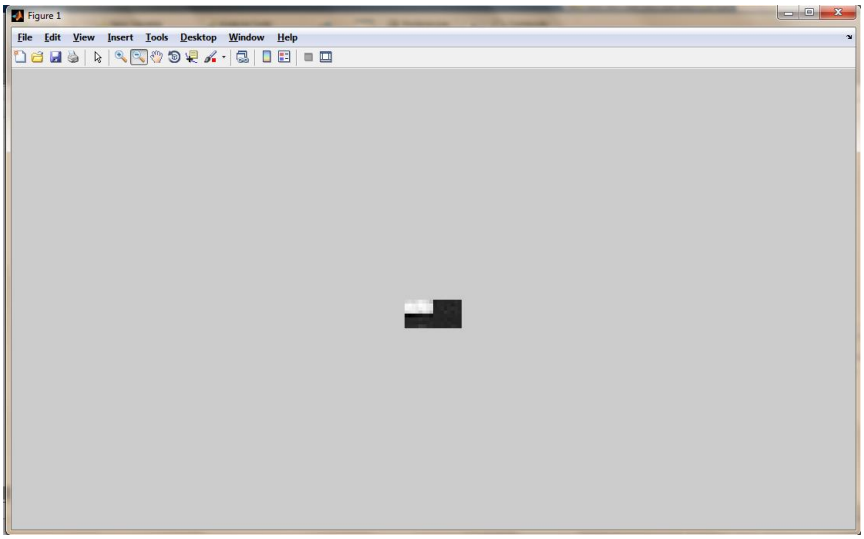

Figura 70. Coeficientes de 4° Nivel de la Transformada Wavelet de Haar 2D del Usuario a Validar.

Al final de este módulo se obtiene el vector CoefW, el cual contiene los 128 coeficientes de la transformada de la imagen del usuario de prueba.

4. Extracción de Características

Ahora, teniendo el vector CoefW, este será procesado por dos métodos distintos paralelamente como se mencionó anteriormente, para obtener la Distancia de Hamming y Distancia Euclidiana entre este vector y las bases de datos BaseH y BaseE respectivamente.

4.1 Codificación Binaria

Se toma el vector CoefW se le aplica la binarización descrita anteriormente, obteniéndose el vector DistHP, el cual contiene los 128 valores binarizados del vector de coeficientes de la Transformada Wavelet de Haar 2D de la imagen del usuario de prueba. Este vector posteriormente será comparado con la matriz BaseH para calcular la Distancia de Hamming entre el vector y la base de datos.

4.2 Cuantificación

Se toma el vector CoefW y se procede a realizar la cuantificación como se ha mencionado anteriormente, En la Figura 71 se puede observar el histograma de los coeficientes de la Transformada Wavelet de Haar 2D del usuario de prueba y en la Figura 72 se observa el resultado de la cuantificación de los coeficientes.

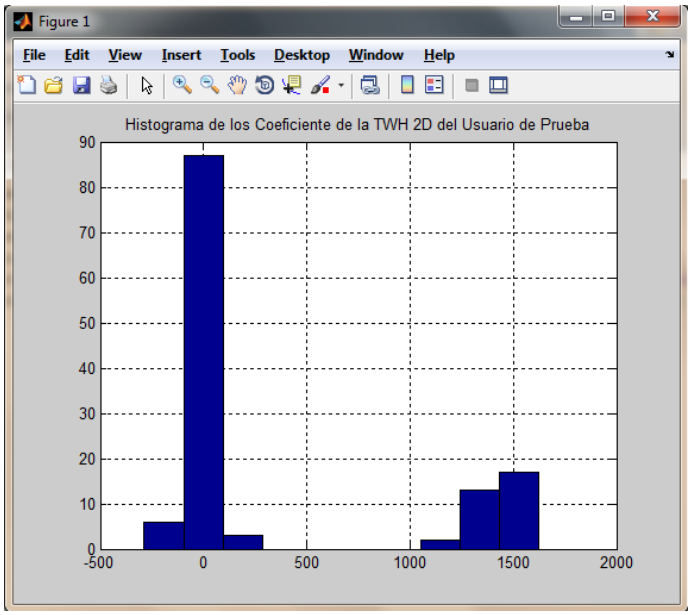

Figura 71. Histograma de los Coeficientes de la Transformada Wavelet de Haar 2D del Usuario de Prueba.

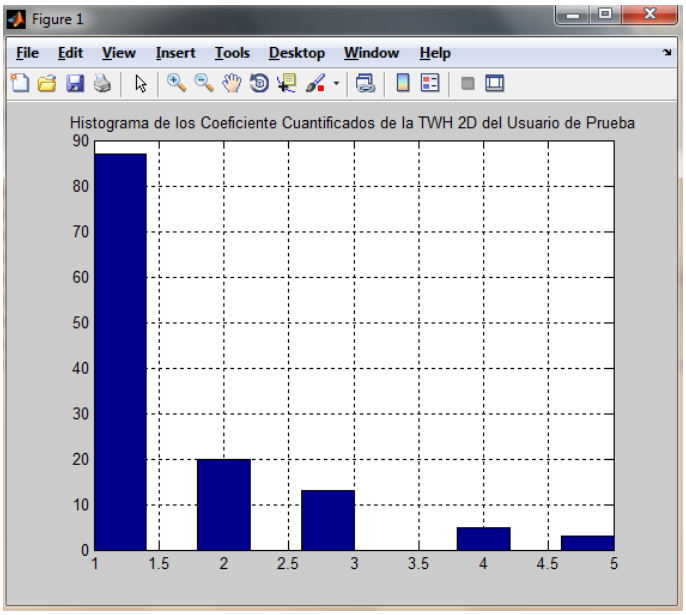

Figura 72.Histograma de los Coeficientes Cuantificados de la Transformada Wavelet de Haar 2D del Usuario de Prueba.

Al final de este proceso se obtiene un vector 128 valores, llamada DistEP, el cual será comparado con la matriz BaseE para calcular la Distancia Euclidiana entre el vector y la base de datos.

5. Cálculo Distancia de Hamming Promedio (DistHM)

Como se menciona en el punto 6 del numeral 4.2.2 del capítulo 4, se procede a calcular las Distancias de Hamming entre el vector DistHP de tamaño 1x128 valores y la base de datos BaseH, la cual contiene 19 vectores de 128 valores cada uno. De este modo, se calculan 19 Distancias de Hamming diferentes, las cuales se promedian para obtener un resultado, el cual indica la Distancia de Hamming Promedio entre el vector de características binarizado del usuario de prueba y la base de datos binarizada del sistema. Este resultado es almacenado en la variable *DistHM* para su comparación posterior en el módulo de Análisis de Identidad.

6. Cálculo Distancia Euclidiana Promedio(DistEM)

Como se menciona en el punto 8 del numeral 4.2.2 del capítulo 4, se procede a calcular las Distancias Euclidianas entre el vector DistEP de tamaño 1x128 valores y la base de datos BaseE, la cual contiene 19 vectores de 128 valores cada uno. De este modo, se calculan 19 Distancias Euclidianas diferentes, las cuales se promedian para obtener un resultado, el cual indica la Distancia Euclidiana Promedio entre el vector de características cuantificado del usuario de prueba y la base de datos cuantificada del sistema. Este resultado es almacenado en la variable *DistEM* para su comparación posterior en el módulo de Análisis de Identidad.

7. Análisis de Identidad

Siguiendo el diagrama de bloques de la Figura 41, este módulo, haciendo uso de las variables anteriormente calculadas DistHM y DistEM, y las variables de la base del sistema UmbralH y UmbralE, determina si el usuario cumple o no la condición

Básicamente, este módulo realiza la comparación de los umbrales del sistema con los resultados de las distancias calculadas para el usuario de prueba y determina si el usuario supera o no esta comparación. En la ecuación (18) se describe que: *"La Distancia de Hamming DistEM del usuario de prueba debe ser menor al Umbral de Hamming UmbralH del sistema y, La Distancia Euclidiana DistEH del usuario de prueba debe ser menor al Umbral Euclidiano UmbralE del sistema".* Así (18) se llama Condición para Usuario Autorizado.

$$(DistHM < UmbralH)\&\&(DistEM < UmbralE) \quad (18)$$

Si la Distancia Euclidiana y la Distancia de Hamming son menores que las determinadas por los umbrales del sistema para cada una, el módulo determina que el usuario de prueba es el usuario registrado en el sistema. Por el contrario, si alguna de las distancias del usuario es mayor a las determinadas por el sistema, el módulo determina que el usuario de prueba no es el registrado en el sistema.

8. Resultado

Finalmente se procede a emitir un resultado de todo el proceso de validación del usuario de prueba. Básicamente el resultado mostrado al usuario es el emitido por el módulo de Análisis de Identidad, ya que es allí donde se determina mediante la ecuación (18) si el usuario de prueba pertenece o no al registrado en la base de datos.

6
Implementación en el ADSP-BF533 EZ-KIT Lite®

En este capítulo se mostrará la implementación del Sistema de Reconocimiento de Iris implementado en el ADSP-BF533 EZ-Kit Lite®, centrándose fundamentalmente en el funcionamiento y procesos llevados a cabo para la ejecución del proyecto en la tarjeta. A continuación se describirá el manejo de memoria en el DSP y las dos funciones principales del sistema de reconocimiento, las cuales son *Registro* y *Validación de Usuario.*

6.1 Administración de Memoria

Para el desarrollo de este proyecto se hizo fundamental un buen manejo y organización de la memoria del DSP, ya que se requirió administrar y gestionar un gran número de registros para el manejo de las variables individuales y las imágenes, de este modo, a continuación se describirá cuál fue el protocolo usado para la administración de la memoria SDRAM; memoria usada para el almacenamiento de los datos en este proyecto.

6.1.1 Administración de Variables

Como se mencionó anteriormente, este proyecto requiere un manejo bastante cuidadoso de las variables, ya que a diferencia de MatLab® en el cual el manejo de la memoria es muy trasparente para el usuario; el DSP exige que el desarrollador determine específicamente el tipo de variable, su posición inicial y su posición final, además se debe tener muy en cuenta la jerarquía de memoria del DSP y el tipo de memoria que se está usando para almacenar la variable. De este modo, para este proyecto se utilizó la memoria SDRAM, ya que esta ofrece la capacidad de almacenamiento necesaria para el manejo de las imágenes procesadas en este proyecto, capacidad que las otras memorias disponibles en el DSP no ofrecen [21].

Para el procesamiento digital de las imágenes se hizo necesario trabajar con dos tipos distintos de variables en el DSP, las de tipo *float* (variable de punto flotante de 32bits, 7 decimales de precisión y representa valores entre $1.18\text{x}10^{-38} \leq x \leq 3.48x10^{38}$), para realizar los cálculos sobre las imágenes, y las variables tipo *unsigned char* (variable entera de 8 bits y representa valores entre $0 \leq x \leq 255$), formato necesario para realizar la visualización de las imágenes en el VisualDSP++.

1. **Acceso a Memoria por Medio de Apuntadores [29]**
   Como se menciona en [29] un apuntador es una herramienta bastante útil y necesaria para el manejo dinámico de los accesos a memoria en un programa. Básicamente un apuntador es una variable que tiene almacenada la posición de

memoria de otra variable, entendiendo que el valor de esta dirección es dinámico así, de este modo se puede acceder de manera dinámica a un conjunto de registros que poseen la información de interés.

En este caso, se utilizaron los apuntadores para almacenar la posición inicial de memoria de una imagen y así, poder recorrer de manera dinámica todos los registros donde se encuentra almacenada la información de los pixeles de la imagen. De este modo, el correcto manejo de estos apuntadores es de vital importancia para poder acceder y guardar información sin que se sobre escriban datos o se presenten consultas a posiciones de memoria incorrectas.

En este proyecto se utilizaron básicamente dos tipo de variables para el manejo de los apuntadores, las variables tipos *float* y las variables tipo *unsigned char*. El apuntador tipo *float* realiza saltos en posiciones de memoria de a 4 bytes, mientras que el apuntador tipo *unsigned char*, realiza saltos en posiciones de memoria de 1 byte. Es de vital importancia tener en cuenta que para los apuntadores tipo *float*, el punto de inicio debe ser el primer byte de cada registro de 32 bits, esto debido a que si el inicio de este apuntador no es el primer byte, el apuntador se dirigirá a un registro incompleto.

6.1.2 Mapa de la Memoria Usada

Debido a la gran cantidad de apuntadores usados en el desarrollo de este proyecto, se hizo necesario crear un mapa de memoria específico para el proyecto, donde se especifica claramente el nombre del apuntador, el formato, la posición inicial y la posición final en la memoria SDRAM de la imagen o vector al cual permite acceder.

En el Anexo 1 se puede encontrar la tabla completa con el mapa de memoria, sin embargo en la Tabla 6, se puede observar una pequeña parte del mapa, en la cual se especifica lo anteriormente mencionado para cada uno de los apuntadores creados.

6.1.3 Configuración del Entorno de Desarrollo VisualDSP++ 5.0

El sistema implementado en el DSP, se desarrolló en la herramienta VisualDSP++, IDE (Entorno de Desarrollo Integrado) de programación muy completo, con el cual no solo se puede realizar el código del algoritmo, sino que también cuenta con herramientas como: *Debug,* para ejecutar segmentos de algoritmos y visualizar en cada instrucción valores en variables y posiciones en la memoria de trabajo, *ImageViewer* para realizar carga de imágenes desde el sistema operativo

directamente a memoria, consola en pantalla para visualizar eventos y estado de compilación, entre otros. [30]

Imagen	Dimensión			Dirección		Nombre Puntero	Descripción
	Numero Bytes	Ancho	Alto	Inicio	Fin		
Imagen Original CASIA (Entrada)	1	640	480	0	0004B000	pImage	GLOBAL
Imagen Original Recortada (Entrada)	4	340	210	0004B004	00090BA4	pImagef	GLOBAL
Imagen Float Final (Salida)	4	340	210	00090BA8	000D6748	pImagefS	GLOBAL
Imagen Char Final (Salida)	1	340	210	000D674C	000E7E34	pImageS	GLOBAL

Tabla 6. Ejemplo del Mapa de Memoria Usada del Sistema de Reconocimiento de Iris implementado en DSP.

La conexión de la tarjeta, se realiza creando una nueva sesión, dando clic en *Session > New Session.* En la ventana de creación de sesión, en el submenú *Select Processor* se elige *ADSP-BF533*, luego en el submenú *Select Conection Type* clic en EZ-KIT Lite. En *Select Plataform* click en *ADSP-BF5xx Single Processor Simulator.* Y finalmente clic en Finish.

Para la creación de un proyecto desarrollado en lenguaje C o C++, partiendo del panel principal del IDE, en la barra de herramientas, se hace clic en *File > New > Project,* luego se visualiza la ventana de creación del proyecto, se selecciona el subgrupo *Select Type* y la opción *Standard Aplication,* luego en el subgrupo *Select Processor* la opción *ADSP-BF533.* En el subgrupo *Aplication Settings,* se elige el tipo de lenguaje en el que se desea desarrollar el proyecto, ya sea C o C++. Por último se hace clic en *Finish.* Después de la creación del proyecto, el entorno es como el mostrado en la Figura 73.

El cuerpo del proyecto cuenta básicamente con 3 carpetas:

Source Files, en la cual se crean todos los script del proyecto, incluyendo la función *main()* que será la ejecutada desde un principio por el DSP. Allí se encuentran los archivos de extensión *.c* y *.cpp*. [2]

Linker Files, que cuenta con archivos tipo *.ldf*, los cuales permiten combinar los objetos creados en el código fuente del proyecto, con ejecutable como *.dxe*, para ser cargados en el emulador del Visual DSP++. Allí se encuentran los archivos de extensión *.ldf* y *.dlb*. [31]

Header Files, en la que se crean los archivos en los cuales se definen los apuntadores y sus respectivas posiciones en memoria. También allí se agregan los archivos fuente de librerías que se deseen añadir al proyecto que no sean nativas del lenguaje. Allí se encuentran los archivos de extensión *.h, .hpp* y *.hxx.* [31]

La creación del proyecto genera un único archivo, main.C. Esta es la función que primero ejecutará el DSP cuando inicie la ejecución del algoritmo. Entonces, es allí donde se inicia a escribir el código del proyecto. Para la generación de un nuevo script, se hace clic en *File* > *New* > *File.* Se crea una hoja en blanco, después de escribir el contenido de este, clic en *File* > *SaveAs* > *File* y se guarda con la extensión deseada. Es de tener en cuenta los tipos de archivos que corresponden a cada carpeta. Después de guardar el archivo en el mismo directorio del proyecto, se añade este archivo a su carpeta correspondiente, dando clic derecho a esta y dando clic en *Add Files To Folder,* se busca el archivo y luego clic en *Aceptar.*

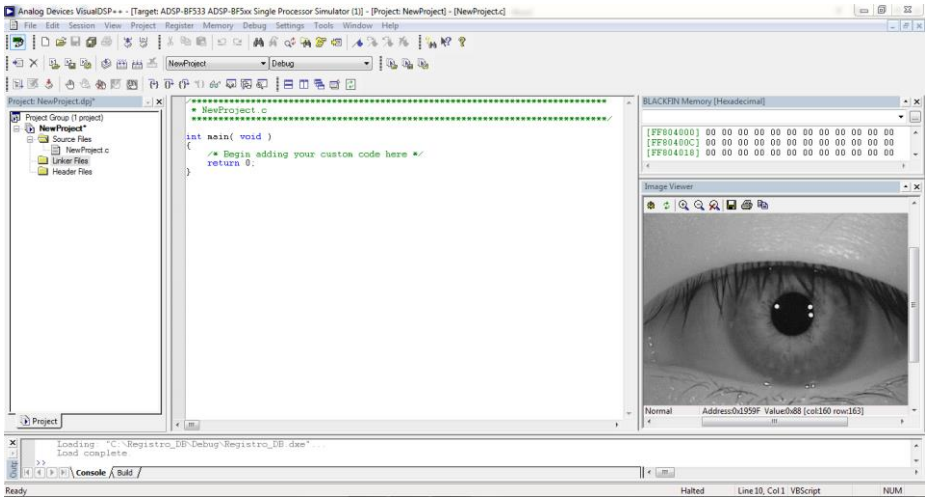

Figura 73. Entorno de Desarrollo VisualDSP++ 5.0.

La barra de herramientas superior, cuenta con funcionalidades como compilación, guardar y abrir proyecto, compilar proyecto (*Build Proyect*), esta hace parte de la barra de herramientas *Proyect.* La barra de herramientas *Debug*, permite ejecutar (*Run*) el proyecto actual en el DSP, ejecutar el código paso a paso o ejecutar este hasta un punto específico (*BreakPoint*). Todas estas pueden ser visualizadas activándolas dando clic en *View* y eligiendo la barra que se desee tener.

6.2 Implementación del Sistema de Reconocimiento de Iris

Después de haber realizado las pruebas de los algoritmos desarrollados para este sistema en MatLab® y comprobar su funcionamiento, se procedió a implementar los mismos algoritmos desarrollados en la tarjeta. A continuación se describirán las funciones principales del sistema, que son *Registro*, la cual se encarga de ingresar un usuario permitido por el sistema y la función *Validación*, la cual se encarga de determinar si un usuario de prueba está o no permitido por el sistema.

6.2.1 Registro Usuario

La función Registro del Sistema de Reconocimiento de Iris fue implementada en el DSP en un proyecto bajo el nombre de *Registro_DB.dpj* en el VisualDSP++® y contiene todos los algoritmos necesarios para registrar a un usuario en la base de datos del sistema. El proceso descrito a continuación será el de una sola imagen a registrar por el usuario, sin embargo este proceso se repite exactamente igual para las otras 18 imágenes del usuario a registrar. A continuación se describirá completamente como fue su implementación en el DSP, centrándose en los procesos desarrollados en la tarjeta y el código completo se encuentra en el Anexo 2.

1. **Inicialización DSP**

   Para la inicialización del DSP se siguió el mismo procedimiento descrito en [23], donde se configura n básicamente el PLL, el reloj del sistema (máxima velocidad disponible) y el acceso a la SDRAM, la cual es configura da para tener el mayor espacio de memoria disponible, ya que como se ha mencionado anteriormente, se hace necesario disponer de gran cantidad de memoria para el manejo de las imágenes. En el Anexo 13 se encuentra el código descrito detalladamente para inicialización del DSP bajo los requerimientos de procesamiento de imágenes implementado en este proyecto, el cual, básicamente es igual al detallado en [23].

2. **Carga de Imágenes**

   Este proceso se realiza mediante el uso de la herramienta *ImageViewer* del VisualDSP++, la cual carga la imagen a una posición de memoria de la SDRAM del DSP. Todas las imágenes se cargan con un tamaño de 640x480 y en formato *unsigned char* (8 bits por pixel y valores de 0 a 255). En la Figura 74 se puede observar la carga de la imagen original del usuario a registrar usando el ImageViewer.

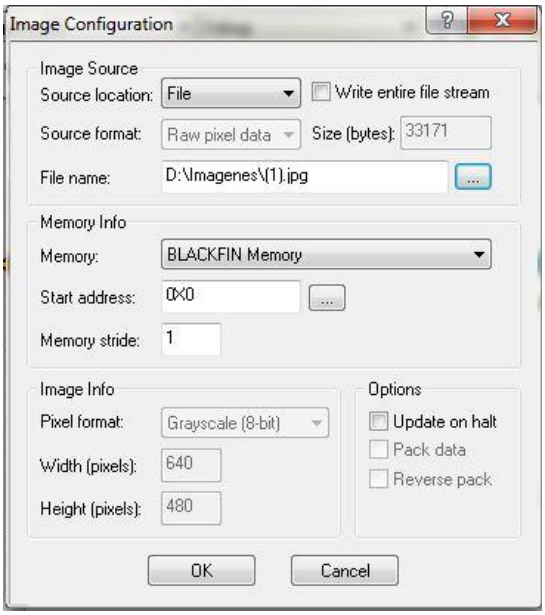

Figura 74. Carga de la Imagen del Usuario a Registrar al DSP usando el ImageViewer del VisualDSP++.

En la Figura 75 se puede observar la imagen del iris del usuario a registrar cargada en el DSP y visualizada en el VisualDSP++®.

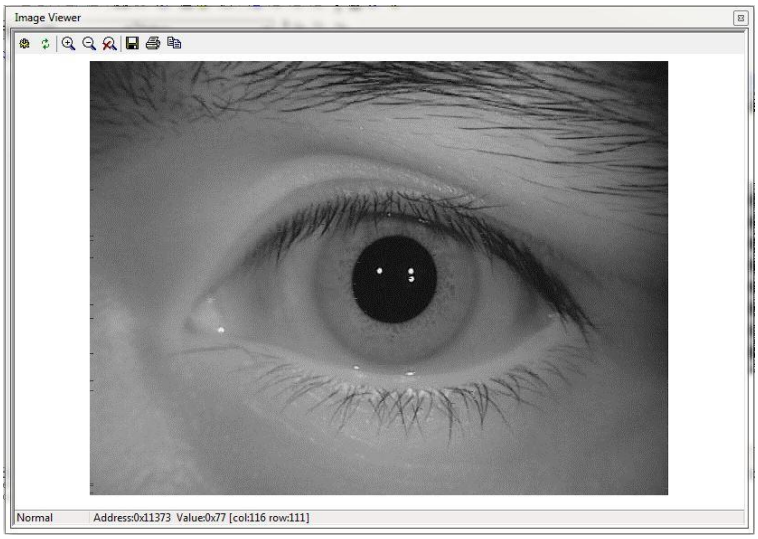

Figura 75. Imagen del Iris Cargada al DSP usando el ImageViewer del VisualDSP++®.

3. Segmentado de Iris

3.1 Recorte 1

Ahora se procede a realizar el primer recorte de la imagen, utilizando la función *Recortar* que se encuentra en el script *Generales.c,* adjunto en el Anexo 10. Esta función recibe dos apuntadores, el primero es el apuntador a la imagen de entrada de tamaño 640x480 en formato *unsigned char*, y el segundo es el apuntador donde se almacenará la imagen de salida de tamaño 340x210 en formato *float*.

Como se ha mencionado anteriormente, este es un recorte estático, donde aprovechando la característica que base de datos CASIA Iris Database 4.0 ofrece imágenes del ojo humano, donde la pupila se encuentra en el centro de la imagen o muy cercano a él; de este modo, se puede eliminar los bordes de la imagen para trabajar sobre una región de interés más pequeña, mejorando el tiempo de procesamiento y el tamaño de la imagen a procesar posteriormente. En la Figura 76 se puede observar el resultado del recorte 1.

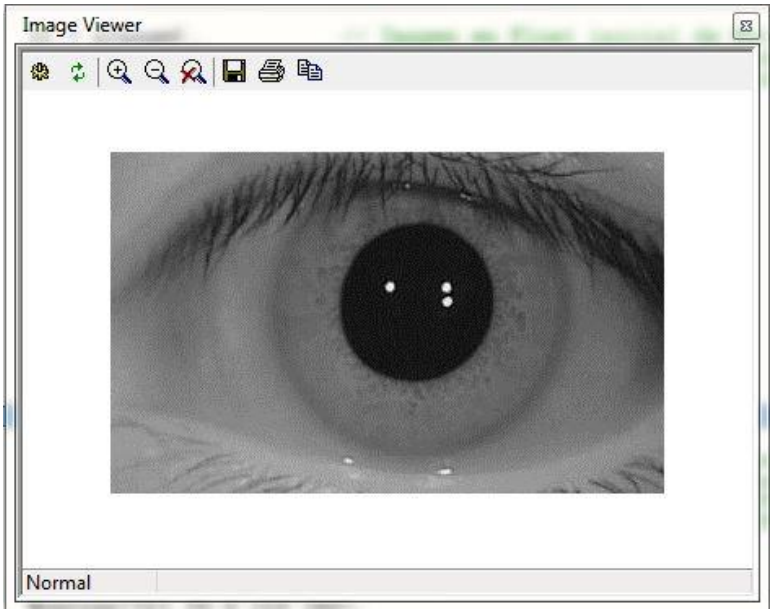

Figura 76. Imagen resultante después del Recorte 1.

3.2 Banco de Filtros

A continuación se procede a realizar un filtrado espacial de la imagen anteriormente recortada, esto utilizando la función *Pillbox* que se encuentra en el script *Pillbox.c,* adjunto en el Anexo 15. Esta función recibe dos apuntadores y dos variables. El primero es el apuntador a la imagen de

entrada de tamaño 340x210 en formato *float*, el segundo es el apuntador donde se almacenará la imagen de salida de tamaño 340x210 en formato *float* y las dos variables nxm de tipo entero que representan el alto y ancho de la imagen de entrada.

Además de realizar el filtrado, se le adicionó a la función *Pillbox* la tarea de ir almacenando las coordenadas (x, y) del pixel con menor intensidad de la imagen; siendo este el pixel semilla para el cálculo posterior de la envolvente convexa en el segmentado de la pupila. Básicamente este banco de filtros es un filtro Pillbox en configuración tipo cascada, la cual permite resaltar los patrones circulares de la imagen del ojo humano.

1. **Filtro 1**

En la Figura 77 se puede observar el resultado del primer filtrado de la imagen del usuario a registrar.

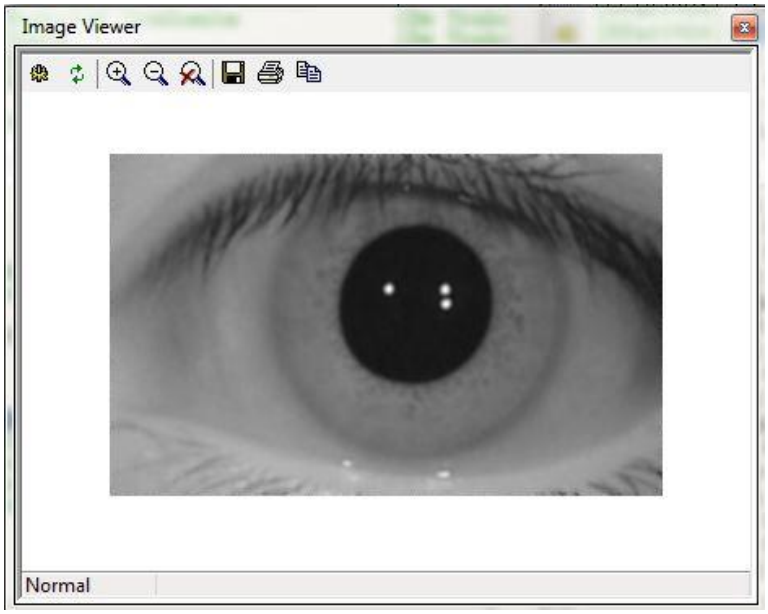

Figura 77. Imagen resultante después del Primer Filtro Pillbox del Primer Banco de Filtros.

2. **Filtro 2**

En la Figura 78 se puede observar el resultado del segundo filtrado de la imagen del usuario a registrar. Además de entregar la imagen final filtrada, este Filtro 2 retorna, como se mencionó anteriormente las coordenadas (x, y) del pixel con la menor intensidad de la imagen.

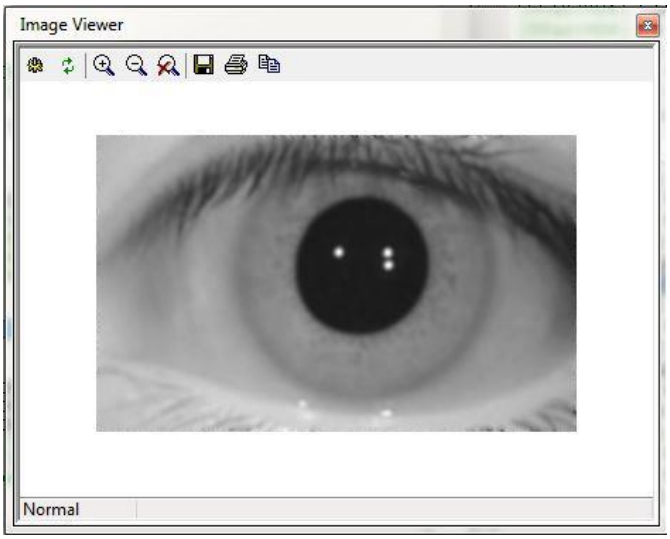

Figura 78. Imagen resultante después del Segundo Filtro Pillbox del Primer Banco de Filtros.

3.3 Detector de Bordes

Después del filtrado se procede a aplicar el detector de bordes de Sobel para resaltar los patrones circulares del iris de la pupila, esto utilizando la función *Gradiente* que se encuentra en el script *Gradiente.c,* adjunto en el Anexo 12. Esta función recibe dos apuntadores y dos variables. El primero es el apuntador a la imagen de entrada de tamaño 340x210 en formato *float*, el segundo es el apuntador donde se almacenará la imagen de salida de tamaño 340x210 en formato *float* y las dos variables nxm de tipo entero que representan el alto y ancho de la imagen de entrada. En la Figura 79 se puede observar la magnitud del gradiente de la imagen.

3.4 Segmentado de Pupila

Ahora se inicia la búsqueda de la pupila para hallar las coordenadas(x_p, y_p, r_p) del círculo de la pupila siguiendo el mismo algoritmo descrito en capítulos anteriores, esto se realiza utilizando la función *EncontrarCentroPupila* que se encuentra en el script *EncontrarCentroPupila.c,* adjunto en el Anexo 6. Esta función recibe dos variables en formato *int*, las cuales corresponden a las coordenadas (x, y) del pixel de menor intensidad calculado anteriormente. En la Figura 80 se puede observar el resultado del segmentado de la pupila.

Figura 79. Imagen resultante del Detector de Bordes para el Segmentado de la Pupila.

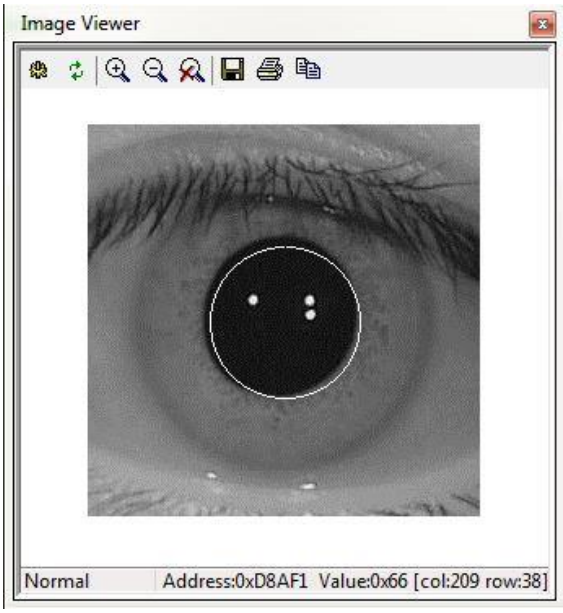

Figura 80. Segmentado de la Pupila en el DSP.

Después de obtener las coordenadas y radio de la pupila (x_p, y_p, r_p), se inicia de nuevo el proceso de recorte, filtrado, detector de bordes pero con el objetivo de segmentar el iris.

3.5 Recorte 2

Ahora se procede a realizar el Recorte 2, el cual es aplicado a la imagen original de la Figura 75, esto utilizando la función *RecortarIris* que se encuentra en el script *Generales.c,* adjunto en el Anexo 10. Esta función recibe dos apuntadores y dos variables, el primero es el apuntador a la imagen de entrada de tamaño 640x480 en formato *unsigned char*, el segundo es el apuntador donde se almacenará la imagen de salida de tamaño 241x241 en formato *float* y las dos variables son las coordenadas de centrado de recorte de la imagen.

A diferencia del Recorte 1, el Recorte 2 es dinámico, esto significa que la imagen es centrada en las coordenadas del círculo de la pupila (x_p, y_p) obtenidas en el paso anterior. En la Figura 81 se puede observar el resultado del Recorte 2 sobre la imagen de la Figura 75.

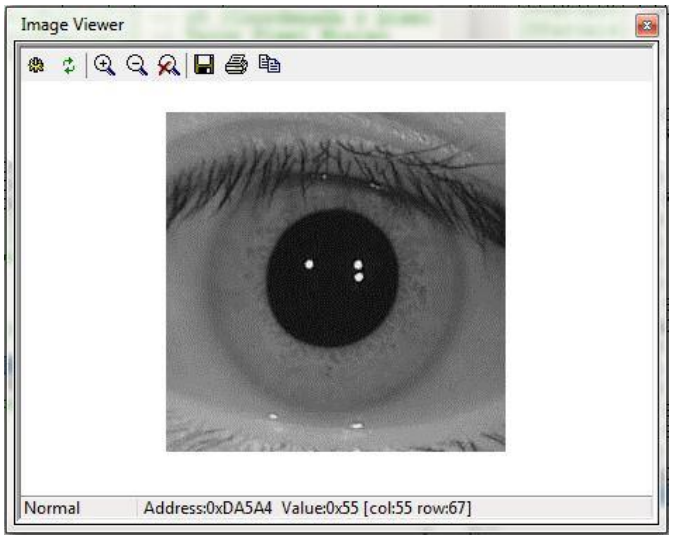

Figura 81. Imagen Resultante del Recorte 2. Recorte Centrado en la Pupila.

3.6 Banco de Filtros

Nuevamente se procede a realizar el filtrado espacial de la imagen para resaltar los patrones circulares, esta vez con el fin de resaltar el iris. Este procedimiento se realiza exactamente igual como se describió anteriormente.

1. **Filtro 1**

   En la Figura 82 se puede observar el resultado del primer filtrado de la imagen del usuario a registrar.

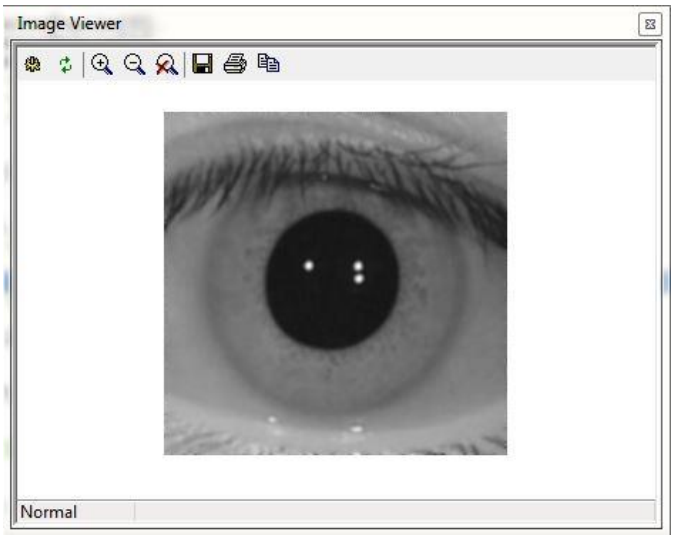

Figura 82. Imagen resultante después del Primer Filtro Pillbox del Segundo Banco de Filtros.

2. Filtro 2

Finalmente, en la Figura 83 se observa el resultado final del segundo banco de filtros, en este caso no se hace uso de la funcionalidad de búsqueda del pixel de menor intensidad, ya que el algoritmo de segmentado del iris es diferente al algoritmo de segmentado de la pupila.

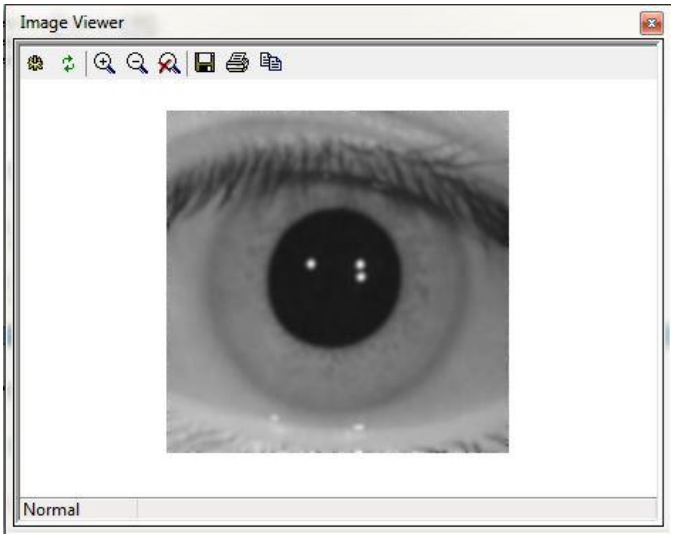

Figura 83. Imagen resultante después del Segundo Filtro Pillbox del Segundo Banco de Filtros.

3.7 Detector de Bordes

Del mismo modo como se describió previamente, este módulo recibe la imagen filtrada y se procede a aplicar el detector de bordes de Sobel, utilizando la misma función *Gradiente*, adjunto en el Anexo 12. En la Figura 84 se observa el resultado del Operador Sobel sobre la imagen recibida del Banco de Filtros 2.

3.8 Segmentado del Iris

Ahora se procede a realizar el segmentado de iris usando la función *EncontrarIris,* la cual se encuentra en el script *EncontrarIris.c,* adjunto en el Anexo 7. Esta función recibe un apuntador y una variable; el apuntador a la imagen de entrada de tamaño 241x241 en formato *float* y la variable que representa el radio de la pupila r_p. La función retorna el radio del iris r_i. En la Figura 85 se puede observar el resultado final del proceso de segmentado del iris implementado en el DSP.

4 Normalización

Después de calcular las coordenadas y radios del iris y la pupila, se procede a realizar el proceso se normalización, el cual consiste en tomar la región del iris de forma de toroide y transformarla en una imagen rectangular. El algoritmo desarrollado en el DSP para realizar esta operación consiste en tomar la región en forma de toroide como un mapa de pixeles en coordenadas polares (r, θ) y realizar su conversión a coordenadas cartesianas, lo cual da como resultado dos matrices que representan las nuevas posiciones (x, y) de los pixeles transformados a coordenadas cartesianas. Sin embargo, esta transformación da como resultado posiciones (x, y) en la imagen con valores decimales, los cuales deben ser procesados para obtener valores enteros, requeridos en un sistema de procesamiento digital de imágenes. Así se tiene que se debe implementar una *Interpolación Bilineal* con el objetivo de obtener el valor de intensidad correcto para cada posición (x, y) de valor entero de la imagen.

En el DSP se creó la función *Normalizar* que se encuentra en el script *Normalizar.c,* adjunto en el Anexo 14; la cual recibe cuatro variables, siendo estas, el ancho y alto de la imagen de salida, y el radio del iris y de la pupila. Esta función no requiere de apuntadores de entrada y salida de imágenes ya que en su código está predeterminada la posición de memoria de donde se toma la imagen de trabajo y donde se guardará la imagen de salida. En el Anexo Mapa de Memoria Usada y los

códigos del DSP se puede observar estas posiciones de memoria. En la Figura 86 se observa el resultado del proceso de normalización del iris.

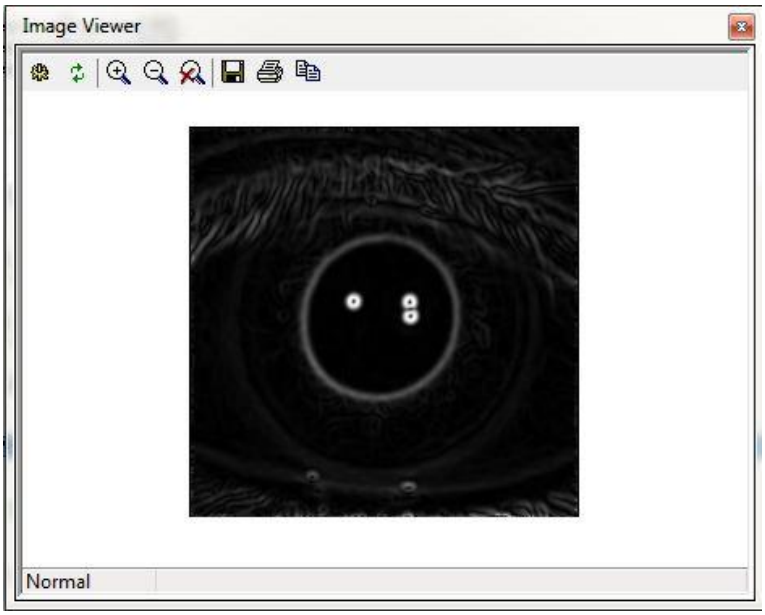

Figura 84. Imagen Resultante del Detector de Bordes para el Segmentado del Iris.

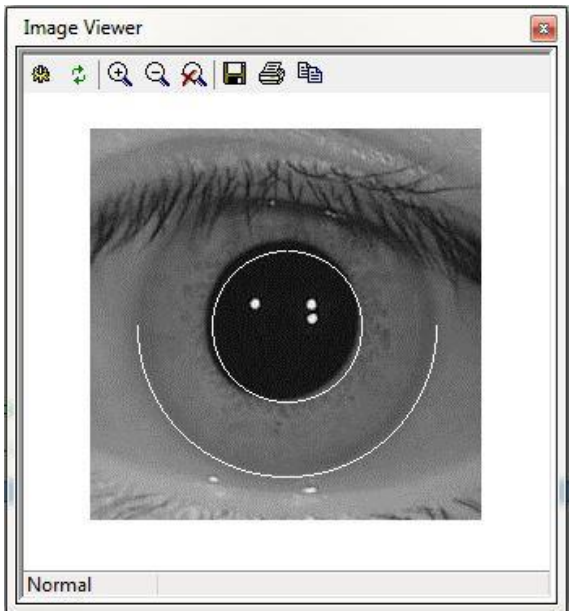

Figura 85. Imagen Resultante del Segmentado del Iris.

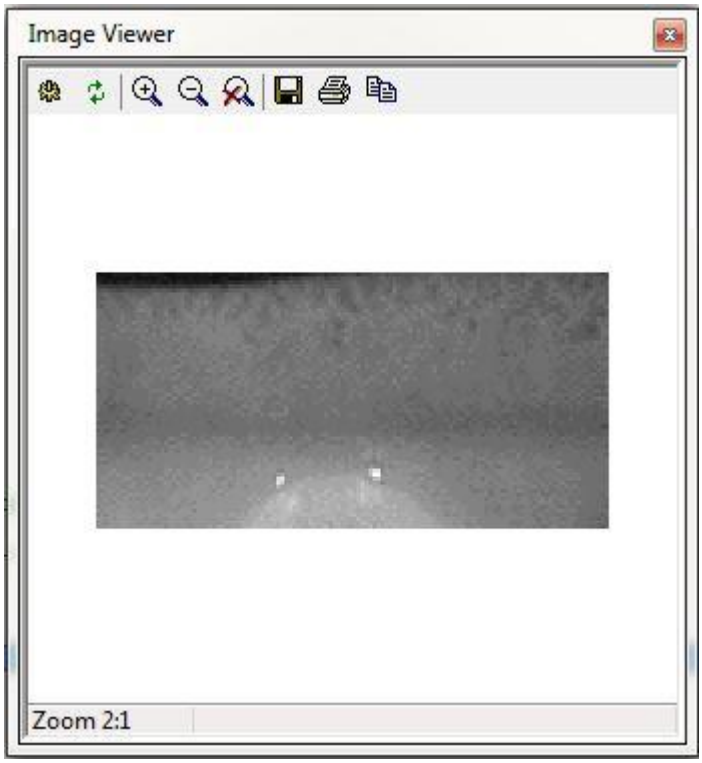

Figura 86. Normalización del Iris obtenido en el DSP.

5 Transformada Wavelet de Haar 2D

Después de obtener una imagen del iris normalizada de un tamaño de $256x64$proveniente del paso anterior, se procede a aplicarle la Transformada Wavelet de Haar 2D para la extracción de los 128 coeficientes de 4° nivel. Esta transformada fue implementada en el DSP en la función *WaveletHaar2D* que se encuentra en el script *WaveletHaar2D.c,* adjunta en el Anexo 17. Esta función recibe tres variables en formato *int*, el nivel de la transformada y el ancho y alto de la imagen que se va a procesar. Esta función retorna un vector de 128 valores en formato *float*, además, esta función no requiere de apuntadores para la imagen de entrada ni para el vector de salida, ya que en su código está predeterminada la posición de memoria de donde se toma la imagen de trabajo y de donde se guardará el vector de salida.

La implementación de esta transformada en el DSP fue implementada siguiendo el diagrama de flujo de la Figura 34, el cual fue diseñado de tipo recursivo, donde para el cálculo de los coeficientes del siguiente nivel, se hace un llamado a sí misma, pero modificando el área de la imagen a procesar. En la Figura 87 se muestra el resultado de la Transformada Wavelet de Haar 2D de 1° nivel.

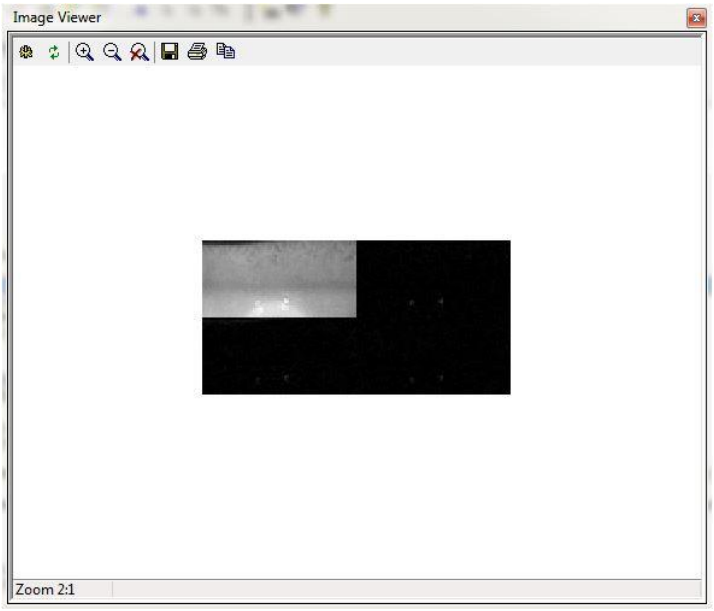

Figura 87. Transformada Wavelet de Haar 2D de Primer Nivel del Usuario a Registrar.

6 Binarización

La binarización de los coeficientes de la Transformada Wavelet de Haar 2D se realiza en la misma función *WaveletHaar2D,* la cual realiza el procedimiento de tomar como 0 los coeficientes negativos y 1 los coeficientes positivos, los cuales son almacenados en un vector.

7 Cuantificación

La cuantificación de los coeficientes de la Transformada Wavelet de Haar 2D se realiza en la misma función *WaveletHaar2D,* la cual realiza el procedimiento de cuantificar en 5 valores distintos los valores de los coeficientes; el resultado de este proceso es almacenado en un vector.

8 Almacenamiento Base de Datos

Para el almacenamiento de la Base de Datos del Usuario registrado en el sistema, se decidió generar un archivo *.txt* para cada una de las dos base de datos; la BaseH en la cual se almacenan los coeficientes binarizados, y la BaseE, donde se almacenan los coeficientes cuantificados.

Como se mencionó anteriormente, en la función *Registro* del Sistema de Reconocimiento de Iris, solo se detalla todo el procesamiento digital en el DSP para una de las 19 imágenes necesarias para registrar un usuario en el sistema; sin embargo, el procedimiento es exactamente el mismo para el resto de las imágenes

a procesar. De este modo, a la hora de almacenar los vectores binarizados y cuantificados en las bases BaseH y BaseE respectivamente; simplemente se concatenan en el archivo txt un vector tras otro, así, al ir procesando cada una de las 19 imágenes, se van creando las base de datos.

6.2.2 Validación

La función *Validación* del Sistema de Reconocimiento de Iris fue implementada en el DSP en un proyecto bajo el nombre de Validacion*.dpj* en el VisualDSP++® y contiene todos los algoritmos necesarios para determinar si un usuario es autorizado o no por el sistema y se encuentra en el Anexo 18.

El procesamiento digital de imágenes para las funciones Validación y Registro de este sistema realizan las mismas funciones que se han venido describiendo a lo largo del proyecto, de este modo, es posible agrupar un conjunto de módulos en una misma función en el DSP y utilizarse tanto para el proceso de Registro de Usuario y Validación de Usuario.

Básicamente los módulos comunes son los de Carga Imagen, Segmentado de Iris, incluyendo todos sus sub módulos Recorte 1, Banco de Filtros 1, Detector de Bordes, Segmentado de Pupila, Recorte 2, Banco de Filtros 2, Detector de Bordes y Segmentado del Iris; también los módulos de Normalización, Transformada Wavelet de Haar 2D, Binarización y Cuantificación.

Para este caso se creó un script que agrupa todo este conjunto de módulos y se llamó *Caracterizacion.c,* adjunta en el Anexo 3. Este script carga una imagen del iris de un usuario mediante el uso de la herramienta ImageViewer del VisualDSP++® y retorna dos vectores de 128 datos cada uno, los cuales corresponden a un vector binarizado y otro cuantificado de los coeficientes de la Transformada Wavelet de Haar 2D de la imagen del iris segmentado.

Así que para la función Validación, las funciones utilizadas son exactamente las mismas que se describieron en la función de Registro, solo que ahora estarán agrupadas en la función *Caracterizacion* del script *Caracterizacion.c.*

1. **Inicialización DSP**

   Se realiza la inicialización del DSP del mismo modo que fue descrito anteriormente, usando el script *Initialization.c* del Anexo 13.

2. Caracterización

Esta función carga mediante el uso del ImageViewer del VisualDSP++® la imagen del iris del usuario que se desea validar y realiza todo el proceso de segmentado del iris, Transformada Wavelet de Haar 2D, cuantificación y binarización de los coeficientes; almacenando en una posición de memoria de la SDRAM prestablecida, estos dos vectores resultantes para su uso posterior.

3. Carga Base de Datos

La función llamada *CargarVectorBaseBasedeDatos* que se encuentra en el script *GestionDeArchivos.c,* adjunta en el Anexo 11; se encarga de cargar a una posición de memoria pre establecida en la SDRAM del DSP los datos de los archivos txt donde está almacenada la base de datos del usuario registrado en el sistema (BaseH y Base E). La función recibe dos variables en formato *int*, siendo la primera variable el número del usuario registrado y la segunda la base de datos a utilizar de ese usuario. (Se tienen múltiples usuarios válidos con múltiples bases de datos, las cuales se usan posteriormente para las pruebas de rendimiento del sistema).

4. Cálculo Distancias de Hamming y Euclidiana

La función *CálculoErrores* que se encuentra en el script *Estadistica.C* se encarga de calcular algunos valores de las distancias euclidianas y la distancia de Hamming. Este módulo en MatLab® se encontraba en la función de Registro Usuario, sin embargo en la implementación en el DSP se decidió realizar esta tarea en la función de Validación Usuario para facilitar el manejo de las bases de datos.

4.1 Análisis Euclidiano (DistE)

Como se describió anteriormente en la implementación del sistema en MatLab®, este algoritmo se encarga de calcular la Distancia Euclidiana de la base de datos BaseE, creando así un vector de 361 valores, llamado DistE, al cual posteriormente se le calculará su promedio (MeanE) y el promedio de su desviación estándar (StdE).

4.2 Distancia Euclidiana(DistEP)

Igualmente se procedió a calcular la Distancia Euclidiana entre el vector de características cuantificado del usuario de prueba y la base de datos cuantificada BaseE, almacenándose en la SDRAM el vector resultante de 19 valores en formato *float* para su posterior uso.

4.3 Distancia de Hamming(DistHP)

También se procedió a calcular la Distancia de Hamming entre el vector de características binarizado del usuario de prueba y la base de datos binarizada BaseH almacenándose en la SDRAM un vector resultante de 19 valores en formato *float* para su posterior uso.

5. Análisis de Identidad y Resultado

Finalmente, el sistema procede a realizar los últimos cálculos y determinar si el usuario de prueba está o no registrado en el sistema. Para ello se procede primero a calcular el umbral automático de Distancia de Euclidiana UmbralE.

5.1 Cálculo del Promedio de la Distancia Euclidiana (MeanE)

Utilizando la función *Promedio* que se encuentra en el script *Estadistica.c*¸ y la cual recibe el puntero DistE, el cual apunta a un registro de 361 valores; esta función procede a calcular el promedio de estos valores, retornando como resultado la variable MeanE. Todas estas operaciones se realizan en formato *float*.

5.2 Cálculo del Promedio de la Desviación Estándar de la Distancia Euclidiana (StdE)

Utilizando la función *Desvest* que se encuentra en el script *Estadistica.c*¸ y la cual recibe el puntero DistE, el cual apunta a un registro de 361 valores; esta función procede a calcular el promedio de la desviación estándar, retornando como resultado la variable StdE. Todas estas operaciones se realizan en formato *float*.

5.3 Cálculo UmbralE

Esta variable es simplemente la suma de MeanE y StdE.

5.4 Análisis de Identidad

Con las variables del sistema UmbralE y UmbralH, y las calculadas para el usuario de prueba DistHP y DistEP, se procede a realizar el análisis de identidad descrito en el Capítulo 4. En el DSP, este análisis fue implementado con dos condicionales *if*, en donde se debe cumplir la ecuación (18) para que un usuario supere el análisis de identidad.

5.5 Resultado

Si el usuario supera el análisis de identidad, la consola del VisualDSP++® mostrará *“Usuario Autorizado”,* de lo contrario, se mostrará en consola el mensaje *“Usuario No Autorizado”*

7
Resultados

En el siguiente capítulo se describirá la metodología usada para el desarrollo de las pruebas de confiabilidad del Sistema de Reconocimiento de Iris; igualmente se mostrarán los resultados de las pruebas de las implementaciones del sistema tanto en MatLab® como en el ADSP-BF533 EZ-Kit Lite®.

Los resultados de las pruebas de confiabilidad y aceptación que se describirán a continuación para ambas plataformas están desarrolladas con el valor umbral de Distancia de Hamming $UmbralH = 0.2375$. Este valor se eligió sobre los otros dos valores expuestos, ya que además de representar el punto de intercepción entre las curvas de los histogramas de las Distancias de Hamming, este valor ofrece una alta tasa de confiabilidad sin sacrificar la aceptación de los usuarios registrados por el sistema.

Las pruebas a continuación descritas además de analizar la confiabilidad del sistema como tal, tienen como objetivo validar los resultados de confiabilidad y aceptación del Sistema de Reconocimiento de Iris mostrados en la Tabla 4 para el UmbralH anteriormente mencionado; esto, ya que los usuarios usados para realizar el análisis de histograma, son diferentes a los usuarios tomados en las siguientes pruebas.

7.1 Metodología de las Pruebas

El desarrollo de las pruebas del Sistema de Reconocimiento de Iris se realizó con el fin de analizar el porcentaje de confiabilidad del sistema. Se analizaron dos características principales de confiabilidad del sistema. La primera es el porcentaje de confiabilidad del sistema frente a intentos de acceso de usuarios no registrados en el sistema, y la segunda característica evaluada es el porcentaje de confiabilidad de intentos de acceso de un usuario registrado por el sistema.

De otro modo y tal como se describe en [24], esto parámetros son llamados “Porcentaje de Falsas Aceptaciones” y “Porcentaje de Falsos Rechazos”, los cuales están definidos por las ecuaciones 21 y 22.

7.1.1 Porcentaje de Falsas Aceptaciones

El Porcentaje de Falsas Aceptaciones se define en (19) como:

$$FAR = \frac{NFA}{NIVA} * 100\% \quad (19)$$

Donde:
FAR es el Porcentaje de Falsas Aceptaciones (False Acceptances Rate, por sus siglas en inglés).
NFA es el Número de Falsas Aceptaciones del Sistema (Number of False Acceptances, por sus siglas en inglés).
NIVA es el Número de Intentos de Usuarios No Válidos (Number of Imposter Verification Attempts, por sus siglas en inglés).

Esta prueba se realiza registrando un usuario en el sistema e intentando acceder con otros usuarios no registrados en el sistema; este procedimiento se realiza un número determinado de veces, para así, al final, determinar el porcentaje Falsas Aceptaciones del Sistema de Reconocimiento de Iris. Esta prueba se realiza implementando una validación cruzada tipo Leave One Out Cross Validation [32] (Validación Cruzada Dejando Uno Afuera) para todos los usuarios de prueba; así, esta validación permite analizar la eficiencia del Sistema de Reconocimiento de Iris para determinar que un usuario no se encuentra registrado en la base de datos del sistema.

7.2.2 Porcentaje de Falsos Rechazos

El Porcentaje de Falsos Rechazos se define en (20) como:

$$FRR = \frac{NFR}{NEVA} * 100\% \ (20)$$

Donde:
FRR es el Porcentaje de Falsos Rechazos (False Rejection Rate, por sus siglas en inglés).
NFR es el Número de Falsos Rechazos del Sistema (Number of False Rejections, por sus siglas en inglés).
NEVA es el Número de Intentos del Usuario Válido (Number of Enrollee Verification Attempts, por sus siglas en inglés).

Siguiendo la misma metodología de las pruebas para Falsa Aceptación y utilizando la validación cruzada tipo Leave One Out, esta prueba pretende determinar qué porcentaje de los intentos de acceso al sistema, este rechaza un usuario que se encuentra registrado en la base de datos del sistema.

7.2 Resultados Pruebas del Sistema de Reconocimiento de Iris Implementado en MatLab®

A continuación se mostrarán los resultados de las pruebas de confiabilidad y aceptación desarrolladas para analizar el Sistema de Reconocimiento de Iris Implementado en MatLab®. Estos resultados determinan que tan seguro y que tan robusto es el sistema rechazando los usuarios inválidos y aceptando al usuario registrado.

7.2.1 Pruebas de Falsas Aceptaciones

Para determinar el porcentaje de falsas aceptaciones del sistema, se tomaron 20 de los 70 usuarios disponibles en la base de datos para realizar las pruebas, las cuales fueron desarrolladas de la siguiente manera:

Se cuenta con 20 usuarios diferentes, los cuales tienen 20 imágenes diferentes del iris; así, se procede a registrar en el sistema un usuario con 19 de las 20 imágenes (se eliminan una de las 20 imágenes siguiendo lo especificado para una validación tipo Leave One Out). Posteriormente, con el usuario registrado en el sistema, se procede a intentar acceder al sistema tomando 1 imagen escogida al azar del resto de la base de datos. Esto se realiza para los 20 usuarios de la base de datos, de este modo se tiene un total de 400 intentos de validación con usuario no válido.

Con los resultados de esta prueba, se completan las variables en la ecuación (19), para así obtener el resultado del Porcentaje de Falsas Aceptaciones del Sistema, y por consiguiente, la confiabilidad del sistema frente a intentos de acceso de usuarios no autorizados.

Desarrollando las pruebas en el archivo *Pruebas_Falsa_Aceptacion* y almacenando el resultado en el archivo *Resultados_Confiabilidad_MatLab.mat* para completar las variables de la ecuación (19), se obtuvo el resultado de las pruebas de falsas aceptaciones del Sistema de Reconocimiento de Iris implementado en MatLab®, esto descrito en (21):

$$NFA = 1$$

$$NIVA = 400$$

$$\boldsymbol{FAR = \frac{NFA}{NIVA} * 100\% = \frac{1}{400} * 100\% = 0.25\%} \ (\mathbf{21})$$

Se realizó un análisis donde se muestran los errores cometidos por los métodos de Distancia de Hamming y Distancia Euclidiana para rechazar a un usuario no registrado. En la Figura 88 se muestra para cada uno de los 20 usuarios analizados, cuál de los dos métodos autorizó el usuario que no debía ser aceptado y la respuesta final de sistema.

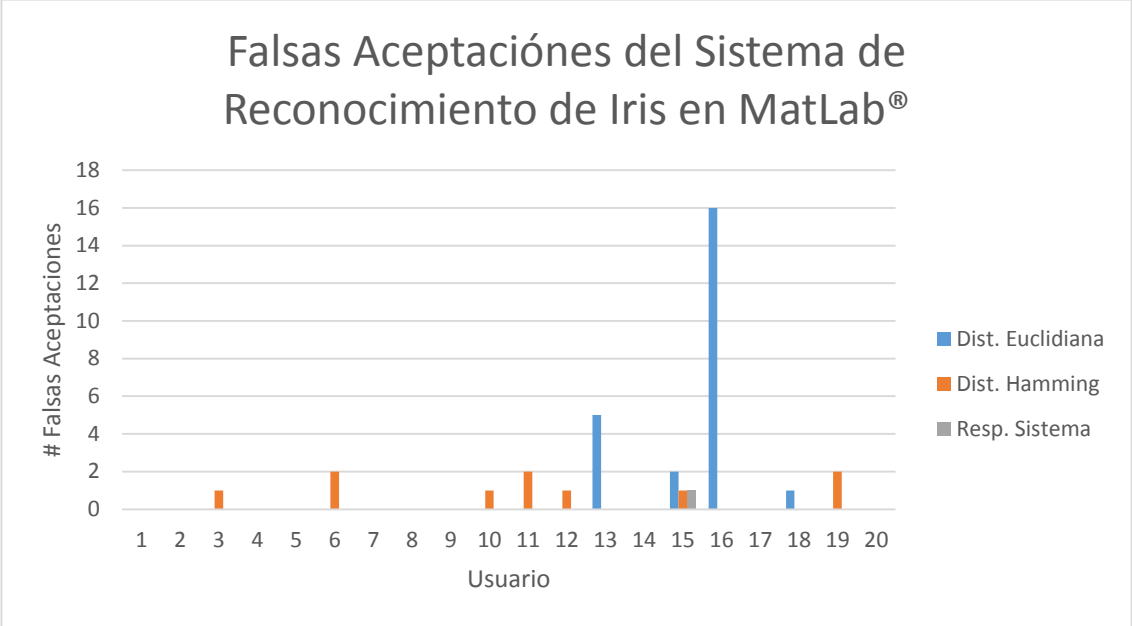

Figura 88. Gráfico de las Falsas Aceptaciones del Sistema Implementado en MatLab® Discriminado por Métodos de Análisis de Identidad Distancia de Hamming (Azul) y Distancia Euclidiana (Naranja) y Resultado Final del Sistema (Gris).

Para cada usuario se realizaron 20 pruebas, que idealmente para cada una de las pruebas, el número de Falsas Aceptaciones por parte del análisis de Distancia Euclidiana y Distancia de Hamming debe ser 0. De este modo y basándose en los resultados anteriormente expuestos, el Sistema de Reconocimiento de Iris implementado en MatLab® obtuvo un rendimiento de confiabilidad del 99.75%, es decir, de cada 400 intentos de acceso de un usuario no autorizado, el sistema implementado en MatLab® autorizará 1.

7.2.2 Pruebas de Falsos Rechazos

Para determinar el porcentaje de falsas rechazos del sistema, se tomaron 20 de los 70 los usuarios disponibles en la base de datos para realizar las pruebas, las cuales fueron desarrolladas de la siguiente manera:

Se cuenta con 20 usuarios diferentes, los cuales tienen 20 imágenes diferentes del iris, así, se procede a realizar una prueba de validación cruzada tipo Leave One Out Cross Validation [31] (Validación Cruzada Dejando Uno Afuera) para todos los usuarios.

Este tipo de validación cruzada toma una sola muestra (imagen) para prueba del sistema (Validación Usuario) y deja el resto de muestras para el entrenamiento del sistema (Registro Usuario). Esta validación se realiza tomando 19 imágenes de un único usuario para entrenar el sistema (Registrarlo) y se deja una imagen del mismo usuario para probar el sistema (Validar); así, este experimento se realiza 20 veces por usuario, para así, rotar las imágenes de prueba.

Al final de los experimentos se tiene un conjunto de 20 experimentos diferentes; cada uno emitiendo 20 respuestas del sistema, entre Usuario Autorizado y Usuario No Autorizado, así, se tiene un total de 400 pruebas distintas. Con los resultados de esta prueba, se completan las variables en la ecuación (20), para así obtener el resultado del Porcentaje de Falsos Rechazos del Sistema, y por consiguiente, la confiabilidad del sistema frente a intentos de acceso de usuarios autorizados.

Desarrollando las pruebas en el archivo *Prueba_Falso_Rechazo.m* y almacenando el resultado en el archivo *Resultados_Aceptacion_MatLab.mat* para completar las variables de la ecuación (20), se obtuvo el resultado de las pruebas de Falsos Rechazos del Sistema de Reconocimiento de Iris Implementado en MatLab®, obteniéndose el siguiente resultado descrito en (22):

$$NFA = 22$$
$$NIVA = 400$$
$$\boldsymbol{FRR = \frac{NFR}{NEVA} * 100\% = \frac{22}{400} * 100\% = 5,5\% \ (22)}$$

Ahora, se expondrá el resultado de un análisis donde se muestran los errores cometidos por los métodos de Distancia de Hamming y Distancia Euclidiana para aceptar a un usuario registrado. En la Figura 89 se muestra para cada uno de los 20 usuarios analizados, cuál de los dos métodos no autorizó el usuario que debía ser aceptado y la respuesta final de sistema.

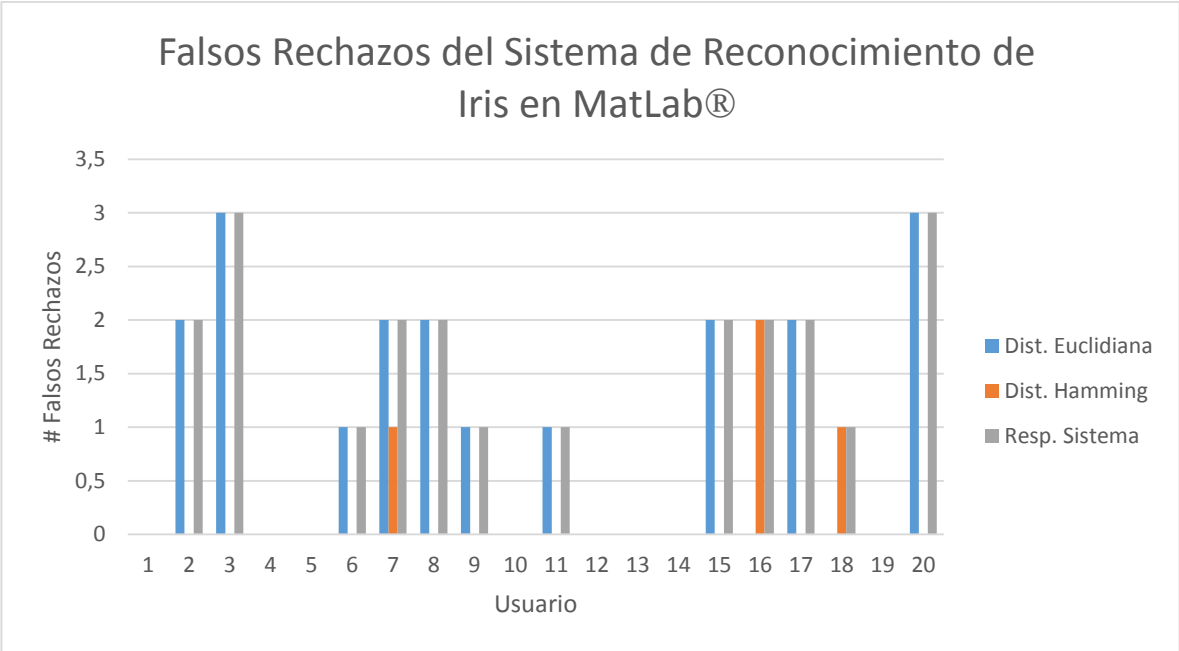

Figura 89. Gráfico de los Falsos Rechazos del Sistema Implementado en MatLab® Discriminado por Métodos de Análisis de Identidad Distancia de Hamming (Azul) y Distancia Euclidiana (Naranja) y Resultado Final del Sistema (Gris).

Para cada usuario se realizaron 20 pruebas, que idealmente para cada una de las pruebas, el número de Falsos Rechazos por parte del análisis de Distancia Euclidiana y Distancia de Hamming debe ser 0. Igualmente teniendo en cuenta los resultados anteriormente expuestos, el Sistema de Reconocimiento de Iris implementado en MatLab® obtuvo un rendimiento de aceptabilidad del 94.50%, es decir, de cada 100 intentos de acceso de un usuario autorizado, el sistema implementado en el ADSP-BF533 EZ-Kit Lite® rechazará 5.

7.3 Resultados Pruebas del Sistema de Reconocimiento de Iris Implementado en el ADSP-BF533 EZ-Kit Lite®.

Ahora se expondrán los resultados de las pruebas de confiabilidad desarrolladas para analizar el Sistema de Reconocimiento de Iris Implementado en el ADSP-BF533 EZ-Kit Lite®. Estos resultados determinan que tan seguro y que tan robusto es el sistema rechazando los usuarios inválidos y aceptando al usuario registrado.

Para registrar un usuario válido en el sistema se utilizó el proyecto *Registro_DB.dpj,* y para realizar la validación de un usuario en el sistema se utilizó el proyecto

Validacion.*dpj.* Ambos proyectos contienen todas las funciones del Sistema de Reconocimiento de Iris.

7.3.1 Pruebas de Falsas Aceptaciones

Siguiendo el mismo procedimiento descrito para las pruebas desarrolladas en MatLab® para el análisis de las Falsas Aceptaciones del Sistema de Reconocimiento de Iris; se procedió en el DSP a realizar 400 pruebas de validación tipo Leave One Out, obteniéndose los resultados de las pruebas de Falsa Aceptación del Sistema de Reconocimiento de Iris Implementado en ADSP-BF533 EZ-Kit Lite®, con los siguientes resultados descritos en (23):

$$NFA = 41$$
$$NIVA = 400$$
$$\boldsymbol{FAR = \frac{NFA}{NIVA} * 100\% = \frac{41}{400} * 100\% = 10.25\% \ (23)}$$

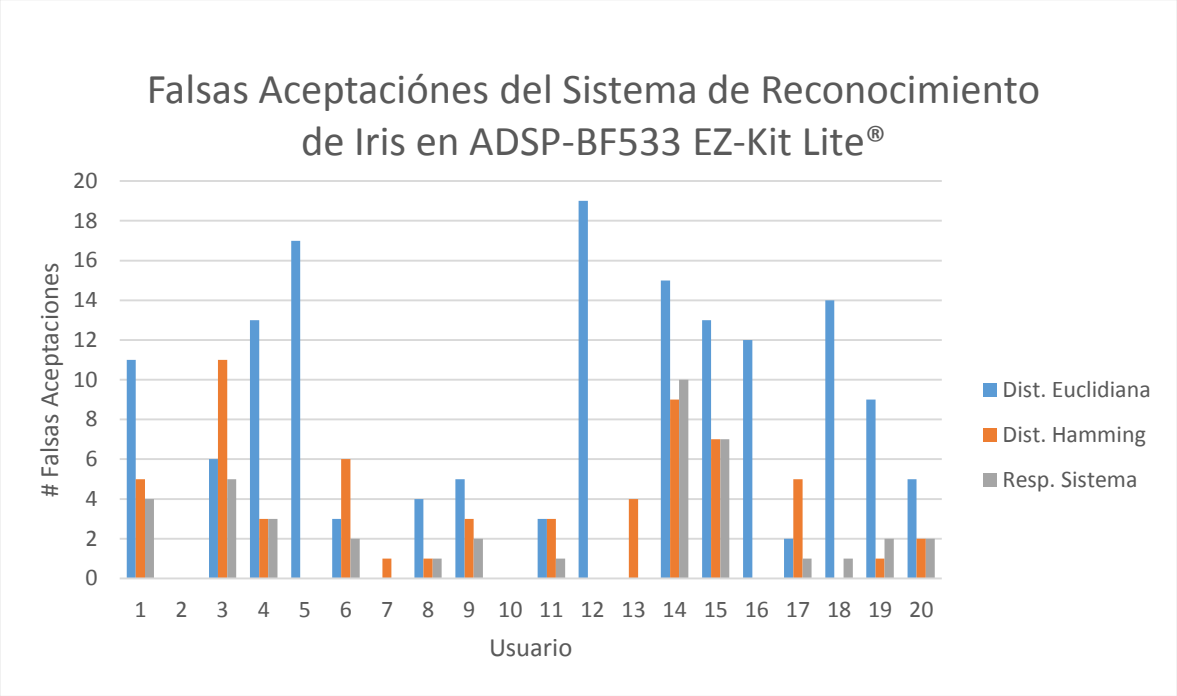

Figura 90. Gráfico de los Falsos Rechazos del Sistema Implementado en el ADSP-BF533 EZ-Kit Lite®, Discriminado por Métodos de Análisis de Identidad de Distancia de Hamming (Azul) y Distancia Euclidiana (Naranja) y Resultado Final del Sistema (Gris).

Igualmente se realizó un análisis donde se muestran los errores cometidos por los métodos de Distancia de Hamming y Distancia Euclidiana para rechazar a un usuario no registrado. En la Figura 90 se muestra para cada uno de los 20 usuarios analizados, cuál de los dos métodos autorizó el usuario que no debía ser aceptado y la respuesta final de sistema.

Para cada usuario se realizaron 20 pruebas, que idealmente para cada una de las pruebas, el número de Falsas Aceptaciones por parte del análisis de Distancia Euclidiana y Distancia de Hamming debe ser 0, de este modo y basándose en los resultados anteriormente expuestos, el Sistema de Reconocimiento de Iris implementado en el ADSP-BF533 EZ-Kit Lite® obtuvo un rendimiento de confiabilidad del 89.75%, es decir, de cada 10 intentos de acceso de un usuario no autorizado, el sistema implementado en el ADSP-BF533 EZ-Kit Lite® autorizará 1.

7.3.2 Desarrollo Pruebas Falsos Rechazos

Realizando el mismo procedimiento descrito para las pruebas desarrolladas en MatLab® para el análisis de las Falsos Rechazos del Sistema de Reconocimiento de Iris; se procedió en el DSP a realizar 400 pruebas de validación tipo Leave One Out, obteniéndose los resultados de las pruebas de Falso Rechazo del Sistema de Reconocimiento de Iris Implementado en ADSP-BF533 EZ-Kit Lite®, con los siguientes resultados descritos en (24):

$$NFA = 36$$
$$NIVA = 400$$
$$\boldsymbol{FRR = \frac{NFR}{NEVA} * 100\% = \frac{36}{400} * 100\% = 9,0\% \ (24)}$$

Igualmente se realizó un análisis donde se muestran los errores cometidos por los métodos de Distancia de Hamming y Distancia Euclidiana para aceptar a un usuario registrado. En la Figura 91 se muestra para cada uno de los 20 usuarios analizados, cuál de los dos métodos no autorizó el usuario que debía ser aceptado y la respuesta final de sistema.

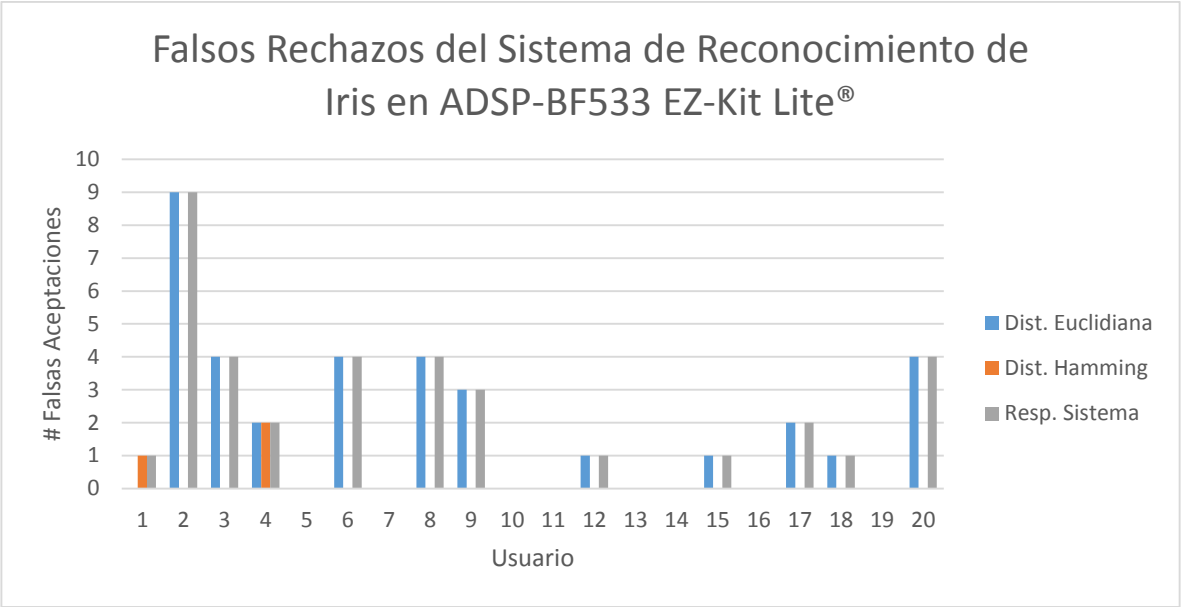

Figura 91. Gráfico de las Falsas Aceptaciones del Sistema Implementado en el ADSP-BF533 EZ-Kit Lite® Discriminado por Métodos de Análisis de Identidad Distancia de Hamming (Azul) y Distancia Euclidiana (Naranja) y Resultado Final del Sistema (Gris).

Para cada usuario se realizaron 20 pruebas, que idealmente para cada una de las pruebas, el número de Falsos Rechazos por parte del análisis de Distancia Euclidiana y Distancia de Hamming debe ser 0, igualmente teniendo en cuenta los resultados anteriormente expuestos, el Sistema de Reconocimiento de Iris implementado en el ADSP-BF533 EZ-Kit Lite® obtuvo un rendimiento de aceptabilidad del 91.00%, es decir, de cada 10 intentos de acceso de un usuario autorizado, el sistema implementado en el ADSP-BF533 EZ-Kit Lite® rechazará 1.

8
Conclusiones

1. Después de exponer los algoritmos desarrollados y los resultados obtenidos de las pruebas realizadas, se puede concluir que se puede implementar un Sistema de Reconocimiento de Iris en la tarjeta de desarrollo ADSP-BF533 EZ-Kit Lite® capaz de ofrecer un rendimiento respecto a confiabilidad y aceptabilidad comparables con los sistemas descritos en la literatura.

2. La implementación de un sistema de control de acceso usando el Sistema de Reconocimiento de Iris desarrollado en este trabajo e implementado en el ADSP-BF533 EZ-Kit Lite® ofrece ventajas significativas de costos y licencias frente a este mismo sistema implementado en MatLab©, ya que la tarjeta de desarrollo cuesta US450 mientras que la implementación en MatLab®, requiere un computador personal de alrededor US500, más la licencia del software para uso comercial que cuestas alrededor de US1000.

3. Observando los resultados obtenidos en confiabilidad y aceptabilidad del Sistema de Reconocimiento de Iris, el algoritmo de comparación de Distancia de Hamming es mejor que el algoritmo de Distancia Euclidiana para discriminar entre un usuario registrado y uno no registrado; de este modo la técnica de emparejamiento de Distancia de Hamming es lo suficientemente robusta como para ser la única técnica para validación de identidad que se implemente en un sistema de reconocimiento de iris. Lo cual aumentaría el rendimiento del sistema (si se usa sólo Hamming).

4. La diferencia entre los porcentajes de falsa aceptación y falso rechazo entre la implementación en MatLab® y el ADSP-BF533 EZ-Kit Lite® se debe a dos elementos básicamente. Primeramente, los valores de intensidad de los pixeles de las imágenes al ser cargadas al sistema, difieren en ambas plataformas, de este modo, siendo estrictos, la imagen procesada no es exactamente la misma en ambos sistema. Finalmente, en MatLab® se manejaron las imágenes en formato *double,* el cual ofrece una precisión de 64 bits, mientras que en DSP, se manejaron las imágenes en formato *float,* el cual tiene una precisión de 32bits. De este modo, MatLab® ofrece mayor presión a la hora de leer la imagen y de realizar los cálculos.

5. Otro factor influyente en la divergencia de los resultados entre la implementación en MatLab® y el ADSP-BF533 EZ-Kit Lite® es la diferencia en los procesadores que realizaron los cálculos. MatLab® fue ejecutado en un procesador Intel® Core i5-3317U con un set de instrucciones de 64bits, mientras que el DSP posee un procesador ADSP-BF533 Blackfin® Processor con un set de instrucciones de 32bits.

6. MatLab es una excelente herramienta de desarrollo para los algoritmos de procesamiento digital de imágenes, ya que ofrece un entorno de desarrollo y bancos de herramientas los suficientemente robustos y amigables con el programador, lo que permite probar y depurar los algoritmos antes de ser implementados directamente en el procesador ADSP-BF533 EZ-Kit Lite®.

7. El Sistema de Reconocimiento de Iris desarrollado en este trabajo puede ser implementado como parte de un sistema de control de acceso basado en el reconocimiento de iris, tanto en su implementación en un computador personal usando los algoritmos desarrollados en MatLab®, como su implementación en el sistema embebido ADSP-BF533 EZ-Kit Lite®.

Bibliografía

[1] Zangh, Kaisheng, y She, Jiao, y Gao, Mingxing, y Ma, Wenbo. 2010. Study on the Embedded Fingerprint Image Recognition System. Shaanxi University of Science and Technology.

[2] Kazuyuki, Miyazawa, y Koichi, Ito, y Takafumi, Aoki, y Koji, Kobayashi, et al. 2008. An Effective Approach for Iris Recognition Using Phase-Based Image Matching. IEEE Transactions on Pattern Analysis and Machine Intelligence, VOL. 30, NO. 10.

[3] Yew fatt, Richard, y Haur, Tay Young, y Ming Mok, Kai. 2009. DSP-Based Implementation and Optimization of an Iris Verification Algorithm using Textural Feature. Sixth International Conference on Fuzzy Systems and Knowledge Discovery.

[4] Jimenez, Judith Liu, y Reillo, Raul Sanchez, y Lindoso Almuneda, y Hurtado, Oscar Miguel. 2007. FPGA Implementation for an Iris Biometric Processor. Microelectronics Group, Dpt. Electronic Technology Universidad Carlos III of Madrid.

[5] Zhengming, Li, y Xiaoqin, Zhu, y Yinghai, Wang. 2007. A Study on Designing Iris Recognition System Based on TMS320DM642. School of Electrical and Information Engineering Jiangsu University.

[6] Shinho Kim, Kyeungido. 2009. Security System and Method by Iris Key System for Door Lock. Patent Application Publication. United States.

[7] Grabowski, Kamil, y Napieralski, Andrzej. 2011. Hardware Architecture Optimized for Iris Recognition. IEEE Transactions on Circuits and Systems for Video Technology, VOL. 21, NO. 9.

[8]Imagen Clásica de Lena, [En Línea]. Disponible en: <http://www.kitware.com/blog/files/128_1232259669.jpg>, [Consulta Octubre 15 de 2013].

[9]Esquema Modelo de Color RGB, [En Línea]. Disponible en: <http://upload.wikimedia.org/wikipedia/commons/thumb/1/11/RGBCube_b.svg/400px-RGBCube_b.svg.png>, [Consulta Octubre 15 de 2013].

[10]Comparación Modelos de Color RGB-CMYK. [En Línea]. Disponible en: <http://upload.wikimedia.org/wikipedia/commons/1/1b/RGB_and_CMYK_comparison.png>, [Consulta Octubre 15 de 2013].

[11] Juuso Olkkonen. Discrete Wavelet Transforms - Theory and Applications.2011. Croacia. Intech.

[12] Martínez Giménez Félix, Peris Manguillot Alfredo, Ródenas Escribá Francisco. Tratamiento de señales digitales mediante wavelets y su uso con Matlab. 2004. Editorial Club Universitario.

[13] A. Jensen, Anders la Cour-Harbo. Ripples in Mathematics: The Discrete Wavelet Transform. 2001. Springer.

[14] Van Fleet Patrick. Discrete Wavelet Transformations: An Elementary Approach with Applications. 2011. John Wiley & Sons.

[15] C. Goswami Jaideva, K. Chan Andrew.Fundamentals of Wavelets: Theory, Algorithms, and Applications. 2011. John Wiley & Sons.

[16]Pertusa Grau, José F.2011. Técnicas de análisis de imagen, (2a ed.): Aplicaciones en Biología. España. Universitat de València.

[17]Steven W. y Smith, Ph.D. The Scientist & Engineer's Guide to Digital Signal Processing. Second Edition. 1999. San Diego, CA, USA. California Technical Publishing.

[18]Esqueda Elizondo José Jaime, Palafox Maestre Luis Enrique. Fundamentos para el procesamiento de imágenes. 2005. Universidad Autónoma de Baja California. UABC.

[19] Alfonso Galipienso Maria Isabel, Cazorla Quevedo Miguel Angel, Colomina Pardo Otto, Escolano Ruiz Francisco, Lozano Ortega Miguel Angel. Inteligencia artificial: modelos, técnicas y áreas de aplicación. .2003. Editorial Paraninfo.

[20]Tortosa Grau Leandro, Vicent Francés José Francisco. Geometría moderna para Ingeniería. 2012. Editorial Club Universitario.

[21]Analog Devices, Inc.ADSP-BF533 EZ-KIT Lite® Evaluation System Manual. Revision 3.2. 2012

[22]Analog Devices, Inc.ADSP-BF533 Blackfin® Processor Hardware Reference. Revisión 3.6. 2013

[23]Zuluaga Arias Carolina, Zuluaga Arias Daniel. Tratamiento de Imágenes Astronómicas en Tiempo Real. 2013. Universidad Tecnológica de Pereira

[24]Rodríguez Játiva Franklin Andrés. Estudio del DSP ADZS-BF561 de Analog Devices, Familia Blackfin y Desarrollo de sus Aplicaciones.2012.Escuela Politécnica del Ejército. Ecuador.

[25] Tze Weng Ng, Thien Lang Tay, Siak Wang Khor. Iris Recognition Using Rapid Haar Wavelet Decomposition.2010.2nd International Conference on Signal Processing Systems (ICSPS). Malaysia

[26] Vithalrao Birgale Lenina, Kokare Manesh. Iris Recognition Using Discrete Wavelet Transform.International Conference on Digital Image Processing. India

[27]Masek Libor, normaliseiris.m, School of Computer Science & Software Engineering, The University of Western Australia.2003.

[28]Base de Datos CASIA Iris DataBase, [En Línea]. Disponible en: <http://biometrics.idealtest.org/dbDetailForUser.do?id=4>, [Consulta Abril 07 de 2014].

[29]A. Gil T. Marcos. Apuntadores en C y C++.2004.Universidad de Carabobo.

[30] Analog Devices, Inc. 2007. Visual DSP++ 5.0 Getting Starter Guide

[31] Analog Devices, Inc. 2011. Visual DSP++ 5.0 Linker And Utilities Manual

[31] Ian H. Witten, Eibe Frank, Mark A. Hall, and Data Mining: Practical Machine Learning Tools and Techniques: Practical Machine Learning Tools and Techniques.Elsevier.2011.

Anexos

ANEXO 1
Tabla de Mapa de Memoria Usada

IMAGEN	DIMENSIÓN			DIRECCIÓN		NOMBRE PUNTERO	DESCRIPCIÓN
	NUMERO BYTES	ANCHO	ALTO	INICIO	FIN		
Imagen Original CASIA (Entrada)	1	640	480	0	0004B000	pImage	Imagen de Entrada.
Imagen Original Recortada (Entrada)	4	340	210	0004B004	000090A4	pImagef	Imagen de Entrada recortada.
Imagen Float Final (Salida)	4	340	210	000090A8	000D6748	pImagefS	Imagen de Trabajo.
Imagen Char Final (Salida)	1	340	210	000D674C	000E7E34	pImageS	Imagen de Trabajo.
Imagen Float 1 (Trabajo)	4	340	210	000E7E38	0012D9D8	pImagef1	Imagen de Trabajo.
Imagen Float 2 (Trabajo)	4	340	210	0012D9DC	0017357C	pImagef2	Imagen de Trabajo.
Imagen Float 3 (Trabajo)	4	340	210	00173580	001B9120	pImagef3	Imagen de Trabajo.
Imagen Float 4 (Trabajo)	4	340	210	001B9124	001FECC4	pImagef4	Imagen de Trabajo.
Imagen Float 5 (Trabajo)	4	340	210	001FECC8	00244868	pImagef5	Imagen de Trabajo.
Imagen Float 6 (Trabajo)	4	241	241	0024486C	0027D3F0	pImagef6	Imagen de Trabajo.
Pixels X Circunferencia	4	596	1	0027D3F4	0027DD44	pPixelsX	Pixels usados en Circulo.C
Pixels Y Circunferencia	4	596	1	0027DD48	0027E698	pPixelsY	Pixels usados en Circulo.C
Pixels X Envolvente Pupila	2	5000	1	0027E69C	00280DAC	pPixEnvolventeX	Pixels usados para Envolvente.
Pixels Y Envolvente Pupila	2	5000	1	00280DB0	002834C0	pPixEnvolventeY	Pixels usados para Envolvente.
Imagen Recortadalris Char Final	1	241	241	002834C4	002917A5	pImageSiris	Imagen de Trabajo.
Imagen NORMALIZACION-Theta	4	1	512	002917A8	00291FA8	pImageNTheta	Imagen de Trabajo Normalización
Imagen NORMALIZACION-Temp	4	1	66	00291FAC	00292084	pImageNTemp	Imagen de Trabajo Normalización
Imagen NORMALIZACION-Rmat	4	64	512	00292088	002B20B8	pImageNRMat	Imagen de Trabajo Normalización
Imagen NORMALIZACION-x0	4	64	512	002B20BC	002D20BC	pImageNx0	Imagen de Trabajo Normalización
Imagen NORMALIZACION-y0	4	64	512	002D20C0	002F20C0	pImageNy0	Imagen de Trabajo Normalización
Imagen NORMALIZACION-Inormalizada	4	64	512	002F20C4	003120C4	pImageNfNorm	Imagen de Trabajo Normalización
Imagen W1 Wavelet	4	64	256	003120C8	003220C8	pImageW1	Imagen de Trabajo Wavelet
Imagen W2 Wavelet	4	64	256	003220CC	003320CC	pImageW2	Imagen de Trabajo Wavelet
VectorResWaveletCuant	4	1	128	003320D0	003322D0	pImageWVector	Imagen de Trabajo Wavelet
VectorBAseDatosCuant	4	19	128	003322D4	003348D4	pVectorDataBase	Vector Base Datos
Error1VectoresValidacion	4	1	19	003348D8	00334924	pVectorError1	Vector Errores Validacion
Error2VectoresValidacion	4	19	19	00334928	00334ECC	pMatrizError2	Matriz Errores Validacion
DesvError2VectoresValidacion	4	1	19	00334ED0	00334F1C	pVectorDesvError2	Vector Desviaciones Estandar F2
VectorResWaveletMuestras	4	1	128	00334F20	00335120	pVectorWMuestras	WAVELETVECTOR Muestras
VectorResWaveletBin	4	1	128	00335124	00335324	pVectorWBinario	WAVELETVECTOR Binario
VectorBAseDatosBinario	4	19	128	00335328	00337928	pVectorDataBaseBinario	Vector Base Datos Hamming
ErrorBin1VectoresValidacion	4	1	19	0033792C	00337978	pVectorErrorBin1	Error Usuario-BaseDatos Hamming
ErrorBin2VectoresValidacion	4	19	19	0033797C	00337F20	pMatrizErrorBin2	Error BaseDatos-BaseDatos Hamming
DesvErrorBin2VectoresValidacion	4	1	19	00337F24	00337F70	pVectorDesvErrorBin2	Desviacion Estandar Hamming
Imagen Iris y Pupila Encontrados	4	241	241	00337F74	00370AF8	pMostrar	Imagen para mostrar iris.

ANEXO 2
Registro_DB.c

```
/*
-----------------------------------------------------------------------

-----------------------------------------------------------------------

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

-----------------------------------------------------------------------

-----------------------------------------------------------------------

Función Main para el registro de un usuario. Las imagenes serán cargadas
mediante la herramienta ImageViewer del VisualDSP++. Este algoritmo
genera los vectores de caracteristicas de cada imagen y las almacena
archivos (.txt) del sistema. Se entiende entonces que estas imagenes y
sus vectores de caracteristicas resultantes representan al usuario que
son aceptados por el sistema de reconocimiento.

La función consta básicamente de dos partes, la primera (función
Registro()) es la extracción de las caracteristicas de la imagen a partir
```

de la transformada Wavelet Haar del iris resultante de la segmentación.

La segunda parte genera los archivos planos de texto con los valores de cuantificación elegidos.

---

*/

```
// Librerias Nativas de C++ y VisualDSP

#include "ccblkfn.h"
#include "sysreg.h"
#include "math.h"
#include "string.h"
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>


#include "Etiquetas.h"                  // Archivo con la definición de los punteros usados.
#include "Slave_global.h"               // Archivo con la definición de todas las funciones
                                        // utilizadas en el proyecto.
```

```
void main(void)
{

    // Con  Figura ción del DSP.
    sysreg_write(reg_SYSCFG, 0x32);
    Init_PLL();                    // Módulo de Con  Figura ción de velocidad del reloj del
                                   // Nucleo y del Sistema
    Init_SDRAM();                  // SDRAM, Módulo de memoria de trabajo.

    volatile clock_t clock_start;  // Variables necesarias para obtener los tiempos
    volatile clock_t clock_stop;   // de ejecución de los segmentos del algoritmo.
    double secs;

    int NumUsuario;
    printf("\n Ingrese El Numero del Usuario a Registrar \n");
    scanf("%d", &NumUsuario);
                                   // Se ingresa un número de usuario para dar nombre al
                                   // archivo de características que se generará.
```

```
int M = 0;
int i = 1;
while(1){                    // Por cada ciclo while, el algoritmo generará un
                             // nuevo vector de caracteristicas.

clock_start = clock();       // Punto de inicio para obtener el tiempo de
                             // ejecución de la función Registro()

Caracterizacion();           // Función Caracterizacion() que se encarga de segmentar,
                             // Normalizar y caracerizar la imagen cargada actualmente
                             // mediante la herramienta ImageViewer.

clock_stop = clock();        // Punto final de ejecución.


secs = ((double) (clock_stop - clock_start))/ CLOCKS_PER_SEC;
                             // El tiempo de ejecución en segundos es igual a la
                             // diferencia de ciclos entre los dos puntos dividido
```

```
                                                    //      la unidad de de
tiempo de la tarjeta (Ciclos/Segundo).

                printf("\nTiempo de ejecución = %f segundos\n",secs);

                GuardarVector(M,NumUsuario);//Función encargada de guardar los
vectores resultantes
                                                    //      en archivos .txt
para su posterior uso en el algoritmo
                                                    //      de verificación.

                printf("\nUsuario Registrado = %d\n",i);
                M = M + 128;
                i++;
        }
}
```

ANEXO 3
Caracterizacion.C

/*

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

Función Caracterizacion() encargada de filtrar, segmentar, normalizar y caracterizar la imagen de entrada. Para este punto, la imagen ha sido cargada en memoria. Al finalizar el proceso esta función genera los vectores de caracteristicas de cada imagen y las almacena los vectores de caracteristicas en en memoria haciendo uso de apuntadores.

Los vectores de caracteristicas resultantes representan al usuario que son aceptados por el sistema de reconocimiento.

---

*/

```
// Librerias Nativas de C++ y VisualDSP
#include "ccblkfn.h"
#include "sysreg.h"
#include "math.h"
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

#include "Etiquetas.h"                    // Archivo con la definición de los punteros usados.
#include "Slave_global.h"                 // Archivo con la definición de todas las funciones
//      utilizadas en el proyecto.

// Se importan las funciones que serán usadas.
extern float Gradiente(float *Pin,float *Pout, int M, int N);
extern void Recortar(unsigned char *Pin,float *Pout);
extern void RecortarIris(unsigned char *Pin,float *Pout,int x0,int y0);
extern float *PillBox(float *Pin,float *Pout,int M,int N);
```

```
extern void Mostrar(float *Pin, unsigned char *Pout,float Pmin, int M, int N);
extern int Circulo(int x0, int y0, int r);
extern int SemiCirculo(int x0, int y0, int r);
extern int EncontrarIris(float *Pin, int r);
extern int *EncontrarCentroPupila(float x0, float y0, float Max);
extern void Normalizar(int Ms,int Ns, int RP, int RI);
extern void WaveletHaar2D(int L,int M,int N);
extern void GuardarVector(int M, int NumUsuario);
extern void CargarVector(void);

void Caracterizacion()
{
// Definición de variables locales.
int i,j;
int n,m,N,k;
n = 640;
m = 480;
N = n*m;
int x,y,r,rP,rI;
int *pX,*pY,*A;
unsigned char *I,*IS,*If6S;
float *If,*IfS,*If1,*If2,*If3,*If4,*If5,*If6;

float Min;
```

```
float *CentroPupila;


I = pImage;                    // Imagen en unsigned char inicial de Entrada
      (Cargada con ImageViewer).

IS = pImageS;                  // Imagen en unsigned char de salida
      (Para Visualización)


If = pImagef;                  // Imagen en Float inicial de Entrada
      (Recortada).

IfS = pImagefS;                // Imagen en Float final de Salida
      (De Trabajo)

If6S = pImageSiris;            // Imagen Iris Recortado de Salida
      (De Trabajo)

If1 = pImagef1;                // Imagen Magnitud Gradiente
            (De Trabajo)

If2 = pImagef2;                // Imagen Theta
                  (De Trabajo)

If3 = pImagef3;                // Imagen EdgeNoise
                  (De Trabajo)

If4 = pImagef4;                // Imagen Edge
                        (De Trabajo)

If5 = pImagef5;                // Imagen Flitada Gaussiano 2
            (De Trabajo)

If6 = pImagef6;                // Imagen Iris Recortado
            (De Trabajo)


Recortar(I,If);                // Se recorta la imagen de entrada a un tamaño de
340x210
```

```
CentroPupila = PillBox(If,IfS,210,340);          //      Se aplican los filtros de promedio circular
CentroPupila = PillBox(IfS,If,210,340);          //      estos devuelven la imagen filtrada y
//      las coordenadas del pixel de menor intensidad
//      para luego ser usado en el algoritmo de
//      segmentación de la pupila.
//
//      CentroPupila[0] -> x0 (Coordenada x pixel de menor intensidad)
//      CentroPupila[1] -> y0 (Coordenada y pixel de menor intensidad)
//      CentroPupila[2] -> Valor Pixel Minimo
//      CentroPupila[1] -> Valor Pixel Maximo

Gradiente(If,If1,210,340);                       //      Filtro Gradiente para el realce de los bodes.

A = EncontrarCentroPupila(CentroPupila[0],       //      Se determina el centro de la pupila a partir de
CentroPupila[1],     //      hallar la envolvente convexa que contiene
CentroPupila[3]);    //      al pixel semilla encontrado en el filtro anterior
//      y trazar el círculo que encierra esta región.

//      A[0] -> x0 (Coordenada x Centro Radio Pupila).
//      A[1] -> y0 (Coordenada y Centro Radio Pupila).
//      A[2] -> Valor Radio de la Pupila.
```

```
rP = A[2];                                          //  Radio de
la Pupila.

RecortarIris(I,If6,A[0],A[1]);                      //  Se recorta de nuevo la
imagen inicial de entrada
//  pero ahora centrada en el centro de la pupila
//  hallado en el paso anterior.

CentroPupila = PillBox(If6,If3,241,241); //  Se aplican de nuevo los filtros de
promedio circular
CentroPupila = PillBox(If3,If4,241,241); //  y realce de bordes para la posterior
segmentación del iris.
Min = Gradiente(If4,If3,241,241);

rI = EncontrarIris( If3 , rP);                      //  Se halla el radio del iris
partiendo de la pupila.

Normalizar(64,256,rP,rI);                           //  Se obtiene la imagen
correspondiente al iris en una nueva
//  imagen de 256x64 pixels.

WaveletHaar2D(4,64,128);                            //  Transformada
de Wavelet Haar a la imagen del Iris, tomando
//  la mitad inferior del iris, esto para excluir la region
```

```
//      superior que normalmente está obtruido por las pestañas
}                                                                    //
        del usuario.
```

ANEXO 4
Circulo.C

```
/*-----------------------------------------------------------------------

-----------------------------------------------------------------------

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

-----------------------------------------------------------------------

-----------------------------------------------------------------------

Script que contiene las funciones para generar círculos o segmentos de
estos, partiendo de las coordenadas x0,y0 y radio (r) del círculo y
generando el conjunto de puntos (x,y) que pertenecen al perímetro del
círculo. Los puntos son generados usando el método o algoritmo de
Bresenham para círculos, el cual, aproxima los puntos de la
circunferencia basado en la ecuacíon de esta ( x^2 + y^2 = r^2 )
tomando para cada punto el de menor error con respecto al real
(ecuación).

-----------------------------------------------------------------------

-----------------------------------------------------------------------*/
```

#include "math.h"

#include "Etiquetas.h"

Circulo.C

```
/*
La función Circulo(int x0, int y0, int r), genera todos los puntos que
hacen parte de la periferia de la circunferencia almacenando estos en
en memoria mediante los punteros PixelX y PixelY, para su posterior uso.*/
int Circulo(int x0, int y0, int r){

   // Definición de Variables y Apuntadores
        int i,j,N,M;
        int x,y;
        int *PixelX,*PixelY;
        float E1,E2;

        PixelX = pPixelsX;
        PixelY = pPixelsY;

        N = 0;                                          //   Valor índice para Punteros de
las coordendas X,Y.

        //  Se genera un primer segmento de circunferencia, perteneciente al primer
```

```
// cuadrante, comprendido entre los puntos en y=r y x=(r*0.707).
        y = r;
        for( i=0;i<ceil(r*0.707);i++ ){
                                        //  Para el valor en x=i, se hallan los errores de los dos
                                        //  posibles valores en y, (y) ó (y-1).
                                        //  Error = x^2 + y^2 - r^2
                E1 = fabs( pow(i,2) + pow(y,2) - pow(r,2) );
                E2 = fabs( pow(i,2) + pow((y-1),2) - pow(r,2) );

                if(E2 <= E1)                    //  Se elige el Error de menor valor.
                        y = y - 1;

                *(PixelY + N) = y0 + y;         //  Y el punto elegido es el almacenado en los vectores.
                *(PixelX + N) = x0 + i;
                N++;
        }

        //  Se genera el siguiente segmento de circunferencia, perteneciente al primer
    //  cuadrante, comprendido entre los puntos en y=0 y x=r.
        x = r;
        for( j=0;j<y;j++ ){
                                        //  Para el valor en y=j, se hallan los errores de los dos
```

```
                                        //  posibles valores en x, (x) ó (x-1).
                                        //  Error = x^2 + y^2 - r^2
            E1 = fabs( pow(x,2) + pow(j,2) - pow(r,2) );
            E2 = fabs( pow((x-1),2) + pow(j,2) - pow(r,2) );

            if(E2 <= E1)                        //  Se elige el Error de menor valor.
                x = x - 1;

            *(PixelY + N) = y0 + j;             //  Y el punto elegido es el almacenado en los vectores.
            *(PixelX + N) = x0 + x;
            N++;
        }

        //  Aquí se tienen los puntos para el primer cuadrante, ahora, para generar
    //  los tres faltantes, se aprovecha la característica simétrica de la
    //  circunferencia para replicar los cuadrantes.

        M = N;
        for(i=1;i<M;i++){

                                        //  Réplica del Cuatro cuadrante.
            *(PixelY + N) = (2*y0) - *(PixelY + i);
            *(PixelX + N) = *(PixelX + i);
```

```
                N++;

                                        //  Réplica del Tercer cuadrante.
                *(PixelY + N) = (2*y0) - *(PixelY + i);
                *(PixelX + N) = (2*x0) - *(PixelX + i);
                N++;

                                        //  Réplica del Segundo cuadrante.
                *(PixelY + N) = *(PixelY + i);
                *(PixelX + N) = (2*x0) - *(PixelX + i);
                N++;

        }
        return (N);
}

/*
La función SemiCirculo(int x0, int y0, int r), genera todos los puntos
que hacen parte de la periferia de el cuarto cuadrante de una
circunferencia, almacenando estos en en memoria mediante los punteros
PixelX y PixelY, para su posterior uso.*/
int SemiCirculo(int x0, int y0, int r){

        // Definición de Variables y Apuntadores
```

```
int i,j,N,M;
int x,y;
int *PixelX,*PixelY;

float E1,E2;

PixelX = pPixelsX;
PixelY = pPixelsY;

N = 0;                                  //  Valor índice para Punteros de las coordendas X,Y.

//  Se genera un primer segmento de circunferencia, perteneciente al primer
//  cuadrante, comprendido entre los puntos en y=r y x=(r*0.707).
y = r;
for( i=0;i<ceil(r*0.707);i++ ){
        E1 = fabs( pow(i,2) + pow(y,2) - pow(r,2) );
        E2 = fabs( pow(i,2) + pow((y-1),2) - pow(r,2) );

        if(E2 <= E1)
                y = y - 1;

        *(PixelY + N) = y0 + y;
        *(PixelX + N) = x0 + i;
```

```
            N++;
        }


        //  Se genera el siguiente segmento de circunferencia, perteneciente al primer
    //  cuadrante, comprendido entre los puntos en y=0 y x=r.x = r;
    x = r;
        for( j=0;j<y;j++ ){
            E1 = fabs( pow(x,2) + pow(j,2) - pow(r,2) );
            E2 = fabs( pow((x-1),2) + pow(j,2) - pow(r,2) );

            if(E2 <= E1)
                x = x - 1;

            *(PixelY + N) = y0 + j;
            *(PixelX + N) = x0 + x;
            N++;
        }

        M = N;
        for(i=1;i<M;i++){

            *(PixelY + N) = *(PixelY + i);
            *(PixelX + N) = (2*x0) - *(PixelX + i);
```

```
                N++;

        }

        return (N);
}
```

ANEXO 5
Circuncentro.c

```
/*----------------------------------------------------------------------
----------------------------------------------------------------------
UNIVERSIDAD TECNOLÓGICA DE PEREIRA
FACULTAD DE INGENIERIAS
INGENIERIA ELECTRONICA
PROYECTO DE GRADO FINAL
SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP
ARLES FELIPE GARCIA MAYA
JUAN CAMILO MORENO RUIZ
2014
----------------------------------------------------------------------
----------------------------------------------------------------------

Script que contiene la función para, a partir de los tres vértices de
un triángulo, encontrar la circunferencia que pasa por estos tres.

El circuncentro de un triángulo, es el punto centro del circulo que pasa
por los tres vértices del triángulo.

----------------------------------------------------------------------
----------------------------------------------------------------------*/
```

```
#include "math.h"
#include "Etiquetas.h"

    /*
    Para hallar este, se encuentran los puntos medios y posteriormente el
    ortocentro del triangulo formado por los tres vértices. Luego a partir
    de las ecuaciones de las rectas que pasan por los puntos medios y
    el ortocentro del triángulo se obtiene un sistema de ecuaciones 2x2 el
    cual por medio del método de sustitucion se hallan los puntos X y Y del
    circuncentro.
    */
int *Circuncentro(float x1, float y1, float x2, float y2, float x3, float y3){

   // Definición de Variables.
        float m1,m2;
        float Xm1,Ym1;
        float Xm2,Ym2;
        float b1,b2;
        float dx,dy;
        int X,Y;
        int r;

        int *C;
```

```
C = (int*)malloc ( 3*sizeof(int) );          // C[0] -> Coordenada x del centro del Circulo.
                                             // C[1] -> Coordenada y del centro del Circulo.
                                             // C[2] -> Radio de la del Circulo.

m1 = - ((x2-x1)/(y2-y1));                    // Pendientes de las rectas que pasan por los puntos medios.
m2 = - ((x3-x1)/(y3-y1));

Xm1 = (x1 + x2)/2;                           // Pendientes de las rectas que pasan por el circuncentro.
Ym1 = (y1 + y2)/2;

Xm2 = (x1 + x3)/2;
Ym2 = (y1 + y3)/2;

b1 = Ym1 - ( m1*Xm1 );                       // Término independiente de la ecuación de la recta.
b2 = Ym2 - ( m2*Xm2 );

X = ceil( (b1 - b2) / (m2 - m1) );           // Se hallan los valores, usando el método de sustitución.
Y = ceil ( b2 + ( m2 * X ) );
```

```
        dx = fabs(x1-X);                    // Diferencias en X y Y, de un vértice al
circuncentro para
        dy = fabs(y1-Y);                    // hallar la distancia entre el circuncentro y el
vértice
                                            // (radio del círculo).


        r = floor(sqrt( pow(dx,2) + pow(dy,2) ) ); // Se halla el radio.
        C[0] = X;
        C[1] = Y;
        C[2] = r;
        return (C);
}
```

ANEXO 6
EncontrarCentroPupila.C

/*----------

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

Script que contiene la función que partiendo de un pixel semilla, dentro de la pupila, halla la envolvente convexa que contiene los pixels que hacen parte de la pupila.

----------*/

```
#include "math.h"
#include "Etiquetas.h"
```

```
//  Función usada para encontrar la circunferencia que más se asemeja a la forma
//  de la pupila.
extern int *Circuncentro(float x1, float y1, float x2, float y2, float x3, float y3);

//  La envolvente se obtiene, analizando la vecindad del pixel semilla y
//  añadiendo al grupo de pixels pertenecientes a la envolvente, aquellos que
//  cumplan un valor de intensidad menor a un umbral establecido.
int *EncontrarCentroPupila(float x0, float y0, float Max){

//  Definición de las variables.
int i,j,x,y,H,L,N;
float Xmin,Xmax,Ymin,Ymax;
float x1,x2,x3;
float y1,y2,y3;
float High;

int *C;
float *ImEnv,*ImGauss;
unsigned short int *PEnvX,*PEnvY;

C = (int*)malloc ( 3*sizeof(int) );          // C[0] -> Coordenada x del centro de la Pupila Encontrada.
// C[1] -> Coordenada y del centro de la Pupila Encontrada.
```

```
// C[2] -> Radio de la Pupila Encontrada.

ImEnv      =      pImagef2;
ImGauss    =      pImagefS;
PEnvX      =      pPixEnvolventeX;        // Vectores que contienen las coordenadas X y Y de los pixels
PEnvY      =      pPixEnvolventeY;        // que se hallan dentro de la envolvente convexa.

// Se genera una imagen para guardar los pixels pertenecientes
// a la envolvente.
for(i=0;i<(210*340);i++)
*( ImEnv + i ) = 0;

N = 340;                    // Ancho de la imagen.
High = 0.23*Max;            // Umbral establecido para hacer parte de la envolvente.

L = 0;                      // Índice del vector de puntos en la envolvente.
H = 1;                      // Índice que indica la posición hasta la cual
// se han analizado los pixels en el vector de puntos
// en la envolvente.

*(PEnvX + L) = x0;          // Se agrega el primer pixel (Semilla)
*(PEnvY + L) = y0;
```

```
Xmin = x0;
Ymin = y0;

Xmax = x0;
Ymax = y0;
x1 = x0;                    // Variables para hallar tres puntos en la periferia
y1 = y0;                    // de la región hallada.
x2 = x0;
y2 = y0;
x3 = x0;
y3 = y0;

// Se analizan los pixels hasta que los pixels analizados
// sea igual a los pixels de la envolvente.
while(L < H){

i = *(PEnvX + L);           // Pixel perteneciente a la envolvente al cual se
j = *(PEnvY + L);           // analizará su vecindad.

*(ImEnv + i + (j*N)) = 255;     // Se cambia el valor en la imagen (Se agrega).

for( x=(i-1);x<=(i+1);x=(x+2) )     // FOR's para recorrer la vecindad.
for( y=j;y<=(j+1);y++ ){
```

```
// Si el valor del pixel es menor al umbral y aún no hace parte de
// la envolvente, se agrega a esta.
if( (*(ImGauss + x + (y*N)) < High) && (*(ImEnv + x + (y*N)) == 0) ){

*(ImEnv + x + (y*N)) = 255;          // Se cambia su valor en la imagen (Se agrega).

*(PEnvX + H) = x;          // Se agrega al vector de pixels dentro de la envolvente.
*(PEnvY + H) = y;
H = H + 1;

if(Xmax < x){              // condicionales para encontrar tres puntos en la periferia
Xmax = x;                  // de la envolvente.
x3 = x;
y3 = y;
}
if(Xmin > x){
Xmin = x;
x1 = x;
y1 = y;
}
if(Ymin < y){
Ymin = y;
x2 = x;
```

```
y2 = y;

}

}

}

L = L + 1;                    // Se aumenta el índice L para analizar el siguiente
pixel.

}

// Luego, a partir de los tres puntos que hacen parte de la

// periferia de la region de la pupila, se halla el círculo

// que pasa por estos tres puntos y así este círculo será

// el que represente la pupila.

C = Circuncentro( x2, y2 , x1, y1, x3, y3 );

return (C);                   // Se retornan los valores del centro y radio de la
circunferencia.


}
```

ANEXO 7
EncontrarIris.C

/*----------

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

Script que contiene la función EncontrarIris(float *Pin, int r), para encontrar la distancia más cercana entre el radio de la pupila y el radio del partiendo del centro de la pupila. Esta función es llamada despues de haberse hallado el centro y radio de la pupila. Valores desde los cuales se parte para hallar la curvatura del iris en la región inferior derecha del ojo.

----------*/

```
#include "math.h"
#include "Etiquetas.h"

//  Función usada para encontrar la semicircunferencia que es usada para hallar
//  la curvatura del Iris.
extern int SemiCirculo(int x0, int y0, int r);

//  El algoritmo parte de un radio inicial basado en el radio de la pupila,
//  luego trazando semicirculos (cuarto cuadrante) cada vez con un radio mayor,
//  se analiza la intensidad de los pixels de la imagen, previamente filtrada
//  con el filtro Gradiente (Realce de Bordes), que coinciden con el
//  semicirculo y se elije el que conincida con el grupo de mayor intensidad.
//  La región de mayor intensidad será aquella donde se encuentra el borde del
//  Iris.
int EncontrarIris(float *Pin, int r){

//  Definición de Variables.
int i,k,N;
int x0,y0,rI;

int *pX,*pY;
float *IGrad;
float Prom,PromM;
```

```
IGrad  = Pin;                          // Imagen de Entrada (Imagen filtrada con
Realce de Bordes)

pX = pPixelsX;

pY = pPixelsY;                         // Vectores con las coordenadas de los pixels
del semicirculo trazado.


x0 = 121;

y0 = 121;                              // El centro de la pupila es estático, gracias a
que previamente

// (en funciones anteriores) se centró la pupila en la posición (121,121).


r = r + 20;                            // Se parte de un valor de radio Inicial.


rI = 119;                              // Radio máximo posible.

PromM = 0;                             // Promedio de los pixels analizados en cada
semicirculo.


for (i=r;i<=115;i++){                  // For para trazar los semicirculos y hacer el
análisis.


N = SemiCirculo(x0,y0,i);              // Se genera el semicirculo. Los puntos
quedan almacenados en los punteros

// pX,pY.


Prom = 0;
```

```
for (k=0;k<N;k++){                    // For para calcular el promedio de todos los
pixels analizados en la región

// del semicirculo actual.

Prom = Prom + *(IGrad + (*(pX + k)) +((*(pY + k))*241) ) ;

}

Prom = Prom / N;                      // Se calcula el valor final.


if( PromM < Prom){                    // Se almacena el caso que tenga el mayor
valor de promedio.

rI = i ;

PromM = Prom;

}

}


return (rI);                          // Se retorna el valor del radio encontrado.


}
```

ANEXO 8
Estadistica.c

/*----------

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

Script que contiene las funciones para calcular el valor promedio y la desviación estandar de un vector o matriz. Estas funciones son usadas en el sistema de validación.

Se hace uso de dos funciones matemáticas de la librería math.h de C.

pow(A,B) -> Esta función devuelve el valor de A^B. A elevado a B.

sqrt(A) -> Esta función devuelve el valor de la raíz cuadrada de A.

También contiene la función CalcularErrores(), la cual calcula las

distancias Euclidiana y Hamming entre el vector de caracteristicas del usuario a validar y los vectores del usuario registrado.

---*/

```
#include "math.h"
#include "Etiquetas.h"

//  Función promedio, la cual recibe el apuntador del vector a calcular el
//  promedio y el tamaño de este.
float Promedio(float *Pin,int N){

   //  Definiciciòn de variables locales.
        int i;
        float Prom;
        float *V;

        V = Pin;

        Prom = 0;                          //  Valor de promedio. Inicia en 0.
        for(i=0;i<N;i++)                   //  Se recorren todos los valores del
              Prom = Prom + *(V + i);      //  vector obteniendo el acumulado total.
```

```
        Prom = Prom/N;                          //  Finalmente el Valor Promedio es igual
                                                //  al Acumulado dividido la cantidad de valores.
        return (Prom);                          //  Se retorna el resultado.
}

/*
  Función Desviación Estandar, la cual recibe el apuntador del vector a calcular,
  y el tamaño de este. La desviacion estandar se calcla siguiendo la siguiente
  ecuación:
          DesviacionEstandar = RaizCuadrada ( Sumatoria( (Xi - X')^2 ) / N )
  Donde:
        X  = Vector de valores.
        Xi = Valor del vector X evaluado en la posición i.
        X' = Valor Promedio de los valores del Vector X.
        N  = Número total de valores en el Vector X.

  Los valores que toma i, son desde 0 hasta N-1.
*/
float Desvest(float *Pin,int N){

    //  Definiciciòn de variables locales.
        int i;
        float Desv,Prom;
        float *V;
```

```
        V = Pin;
        Prom = (V,N);

        Desv = 0;                       //  Valor de promedio. Inicia en 0.
        for(i=0;i<N;i++){               //  For para calcular el valor de la sumatoria
                Desv = Desv + pow(( *(V+i) - Prom ),2);     //  Se acumula el valor de cada término (Xi - X')^2.
        }

        Desv = sqrt(Desv/N);                    //  Finalmente la Desviación Estandar es igual
                                        //  a la raíz cuadrada del Acumulado dividido
                                        //  la cantidad de valores.

        return (Desv);                  //  Se retorna el resultado.
}

//  Función CalcularErrores, HAce uso de los vectores de caracteristicas del
//  usuario ingresado y la Base de Datos del usuario registrado.
void CalcularErrores(){

    // Definición de variables Globales.
        int i,j,k;
```

```
float Desv,Prom;

float *VE,*VEbd,*EE1,*EE2;

float *VH,*VHbd,*EH1,*EH2;

float sum1, sum2;

float sumH1, sumH2;


VE          = pImageWVector;                    // Apuntador
Vector Caracteristicas Euclidiano Usuario a Validar.

VEbd  = pVectorDataBase;                        // Apuntador Vector
Caracteristicas Euclidiano Usuario Base de Datos.

EE1    = pVectorE1;                             // Apuntador Vector Errores
entre vector Usuario a Validar y Base de Datos (Euclidiano).

EE2    = pMatrizError2;                         // Apuntador Vector
Errores entre Vectores Base de Datos (Euclidiano).


VH          = pVectorWBinario;                  // Apuntador
Vector Caracteristicas Hamming Usuario a Validar.

VHbd  = pVectorDataBaseBinario;                 // Apuntador Vector
Caracteristicas Hamming Usuario Base de Datos.

EH1    = pVectorErrorBin1;                      // Apuntador Vector
Errores entre vector Usuario a Validar y Base de Datos (Hamming).

EH2    = pMatrizErrorBin2;                      // Apuntador Vector
Errores Vectores Base de Datos (Hamming).



for(i=0;i<19;i++){
```

```
sum1 = 0;

sumH1 = 0;

                                        //  Cálculo Errores entre vector
Usuario a Validar y Base de Datos.

for(j=0;j<128;j++){

        sum1 = sum1 + pow( ( (*(VE + j)) - (*(VEbd + j + (i*128))) ) , 2 );
// Distancia Euclidiana.


        if( (*(VH + j)) != (*(VHbd + j + (i*128))) ) {                  //
Distancia Hamming (Comparacion XOR).

                sumH1++;

        }

}

*(EE1 + i) = sqrt(sum1);

*(EH1 + i) = sumH1/128;                                 //  Normalización.



for(k=0;k<19;k++){                                      //  Cálculo Errores
entre Vectores Base de Datos.

        sum2 = 0;

        sumH2 = 0;

        for(j=0;j<128;j++){

                sum2 = sum2 + pow( ( (*(VEbd + j + (i*128)) ) - (*(VEbd
+ j + (k*128))) ) , 2 );   // Distancia Euclidiana.
```

```
                                    if( (*(VHbd + j + (i*128)) ) != (*(VHbd + j + (k*128))) ) {
// Distancia Hamming (Comparacion XOR).
                                        sumH2++;
                                    }
                                }
                                *(EE2 + k + (i*19)) = sqrt(sum2);
                                *(EH2 + k + (i*19)) = sumH2/128;          // Normalización.
                            }
                        }
                    }
```

ANEXO 9
Etiquetas.h

```
/*-----------------------------------------------------------------------

-----------------------------------------------------------------------

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

-----------------------------------------------------------------------

-----------------------------------------------------------------------

Script que contiene las direcciones de memoria que son reservadas para

el procesamiento de las imagenes en el DSP y sus respectivas etiquetas.

-----------------------------------------------------------------------

-----------------------------------------------------------------------*/


// Direccion Inicio Imagen Original Cargada (8 Bits)

#define pImage (unsigned char *)0x00000000
```

```
// Direccion Inicio Imagen Original Recortada Float
#define pImagef (float*)0x0004B004

//Direccion Inicio Imagen Salida (8 Bits)
#define pImageS (unsigned char *)0x000D674C

// Direccion Inicio Imagen Salida en Float
#define pImagefS (float*)0x00090BA8

// Direccion Inicio Imagen De Trabajo 1
#define pImagef1 (float*)0x000E7E38

// Direccion Inicio Imagen De Trabajo 1
#define pImagef2 (float*)0x0012D9DC

// Direccion Inicio Imagen De Trabajo 1
#define pImagef3 (float*)0x00173580

// Direccion Inicio Imagen De Trabajo 1
#define pImagef4 (float*)0x001B9124

// Direccion Inicio Imagen De Trabajo 1
#define pImagef5 (float*)0x001FECC8
```

```
// Direccion Inicio Imagen De Trabajo 1
#define pImagef6 (float*)0x0024486C

// Direccion Inicio Vector Pixels X (Uso en funciòn Cìrculo() )
#define pPixelsX (int*)0x0027D3F4

// Direccion Inicio Vector Pixels Y (Uso en funciòn Cìrculo() )
#define pPixelsY (int*)0x0027DD48

// Direccion Inicio Pixels Envolvente en X
#define pPixEnvolventeX (unsigned short int*)0x0027E69C

// Direccion Inicio Pixels Envolvente en Y
#define pPixEnvolventeY (unsigned short int*)0x00280DB0

// Direccion Inicio Imagen Iris Recortado (Para visualización)
#define pImageSiris (unsigned char *)0x002834C4

// Direccion Inicio Vector Theta (Usado en función Normalizar() ).
#define pImageNTheta (float *)0x002917A8

// Direccion Inicio Vector Temp (Usado en función Normalizar().
#define pImageNTemp (float *)0x00291FAC
```

```
// Direccion Inicio Vector Rmat (Usado en función Normalizar().

#define pImageNRMat (float *)0x002920B8


// Direccion Inicio Vector Valores X en coordenadas Cartesianas (Usado en función Normalizar().

#define pImageNx0 (float *)0x002B20BC


// Direccion Inicio Vector Valores Y en coordenadas Cartesianas (Usado en función Normalizar().

#define pImageNy0 (float *)0x002D20C0


// Direccion Inicio Imagen salida Normalizada (Usado en función Normalizar().

#define pImageNfNorm (float *)0x002F20C4


// Direccion Inicio Apuntador imagen parcial transformada vertical (Usado en función WaveletHaar().

#define pImageW1 (float *)0x003120C8


// Direccion Inicio Apuntador imagen parcial transformada horizontal (Usado en función WaveletHaar().

#define pImageW2 (float *)0x003220CC


// Direccion Inicio Vector Apuntador al vector de valores resultado cuantificado (Usado en función WaveletHaar().

#define pImageWVector (float *)0x003320D0
```

```
// Direccion Inicio Vector Base de Datos cargada.

#define pVectorDataBase (float *)0x003322D4


// Direccion Inicio Vector Apuntador al vector de valores resultado (Usado en función WaveletHaar().

#define pVectorWMuestras (float *)0x00334F20


// Direccion Inicio Vector Apuntador al vector de valores resultado binarizado (Usado en función WaveletHaar().

#define pVectorWBinario (float *)0x00335124
```

ANEXO 10
Generales.C

```
/*

---------------------------------------------------------------------

---------------------------------------------------------------------

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

---------------------------------------------------------------------

---------------------------------------------------------------------

Script que contiene las funciones de recorte usadas en el proyecto.
También la función Mostrar() que permite guardar una imagen con el
formato adecuado para ser visualizado por la herramienta ImageViewer.

---------------------------------------------------------------------

---------------------------------------------------------------------

*/

#include "math.h"
```

```
#include "Etiquetas.h"

#include <stdio.h>

#include <stdlib.h>

#include <stdint.h>


//      Esta función recibe el apuntador de la imagen a recortar (*Pin) y el apuntador de la imagen

//      recortada (*Pout). El recorte sustrae la región de interes en la imagen, esto para reducir el

//      tiempo de procesamiento del algoritmo, ya que inicialmente, se desea conocer únicamente la

//      posición del iris en la imagen.

void Recortar(unsigned char *Pin,float *Pout){


        // Definición de variables Globales.

        int i,j;

        int Min, Nin, Mout, Nout;


        unsigned char *In;

        float *Out;


        In = Pin;
                //      Apuntador Imagen de Entrada.

        Out = Pout;
                //      Apuntador Imagen de Salida.
```

```
	Min = 480;
		//	Alto y ancho de la imagen de entrada.
	Nin = 640;


	Mout = 210;
		//	Alto y ancho de la imange de salida.
	Nout = 340;


	for(j=149;j<(Min-120);j++)                                          //
	Se recorren las filas y columnas de la imagen,
		for(i=149;i<(Nin-150);i++)                                  //
	copiando los pixels de la imagen de entrada (In)

			//	a la imagen de salida (Out)
			*(Out + (i-149)+((j-149)*Nout) ) = (float)*(In + i+(j*Nin) );


}


//	Esta función recibe el apuntador de la imagen a recortar (*Pin) y el apuntador de la imagen
//	recortada (*Pout) y las coordenadas del pixel en el cual se centrará la imagen (x0,y0).
void RecortarIris(unsigned char *Pin,float *Pout,int x0,int y0){


	// Definición de variables Globales.
	int i,j;
```

```
int Min, Nin, Mout, Nout,alto,ancho;


unsigned char *In;

float *Out;


In = Pin;
        //      Apuntador Imagen de Entrada.

Out = Pout;
        //      Apuntador Imagen de Salida.


y0 = y0 + 150;

x0 = x0 + 150;
        //      Offset causado por el primer recorte

                //      hecho en la imagen original


Min = 480;
        //      Alto y ancho de la imagen de entrada.

Nin = 640;


Mout = 241;
        //      Alto y ancho de la imange de salida.

Nout = 241;


alto = 0;

for(j=(y0-120);j<=(y0+120);j++){                                //      Se
recorren las filas y columnas de la imagen,
```

```
            ancho = 0;
            for(i=(x0-120);i<=(x0+120);i++){                    //
      copiando los pixels de la imagen de entrada (In)

                  //      a la imagen de salida (Out).
                  *(Out + ancho +(alto*Nout) ) = (float)*(In + i+(j*Nin) );
                  ancho++;
            }
      alto++;
      }

}

//    Esta función recibe el apuntador de la imagen de entrada (*Pin) y el apuntador de la imagen de
//    salida (*Pout), el valor del pixel de menor intensidad (Pmin), y las dimensiones de la imagen (M,N).
//    Esta función ajusta el rango dinánico de la imagen a valores entre 0 y 255 para su correcta visualización.
//    Por esto es que se recibe el valor mínimo de intensidad de la imagen, para tomar este valor como el 0.
void Mostrar(float *Pin, unsigned char *Pout,float Pmin, int M, int N){

      // Definición de variables Globales.
      int i,j;
      unsigned char *Out;
```

```
        float *In,Pmax,Aux;


        Pmax = 0;


        In = Pin;
                //      Apuntador Imagen de Entrada.
        Out = Pout;
                //      Apuntador Imagen de Salida.




        for(i=0;i<(M*N);i++){
        //      Se recorre la imagen para hallar el valor del pixel
                *(In + i ) = Pmin + (*(In + i ));                       //      de
mayor intensidad. Y a su vez se suma el valor mínimo
                if(Pmax < *(In + i ) )
                        Pmax = *(In + i );
        }


        for(i=0;i<(M*N);i++){
        //      Se recorre de nuevo la imagen, ahora ajustando los
                Aux = (255.0f/Pmax) * (*(In + i ));                     //
        valores a un rango de [0-255]
                *(Out + i ) = (unsigned char)Aux ;
        }
}
```

ANEXO 11
GestionDeArchivos.C

```
/*
-----------------------------------------------------------------------

-----------------------------------------------------------------------

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

-----------------------------------------------------------------------

-----------------------------------------------------------------------

Script que contiene la función

GuardarVector(int M, int NumUsuario). Función para generar la base de
datos con los vectores de características correspondientes al usuario
registrado. Esta función se encarga de generar y guardar los archivos
en formato de texto para los vectores de caracteristicas cuantificadas
y binarizadas anteriormente, en una carpeta del sistema operativo. Estas
bases de datos serán usadas porteriormente para las pruebas de
validación del sistema.
```

La función almacena dos bloque de 128 valores (Valores Cuantifiados y Valores Binarios), que equivalen al vector de caracteristicas de una imagen procesada.

Para llevar un orden de las bases de datos generadas, se lleva una nomenclatura para los nombres de los archivos, estas son:

Para Bases de Datos con valores cuantificados:

Nombre -> [NombreUsuario]LCuant.txt

Para Bases de Datos con valores binarios:

Nombre -> [NombreUsuario]LBin.txt

NombreUsuario corresponde al valor string del usuario a registrar, es decir, para el registro del usuario 1, los nombres de los archivos serán:

Bases de Datos con valores cuantificados --> 1LCuant.txt

Para Bases de Datos con valores binarios --> 1LBin.txt

```
---------------------------------------------------------------------

*/

#include "math.h"
#include <stdio.h>
#include "Etiquetas.h"
#include <string.h>

//  Esta función recibe un número de usuario para etiquetar el archivo a generar
//  También recibe un índice M, para conocer la cantidad de datos actuales en
//  el archivo y evitar la sobreescritura de estos.
void GuardarVector(int M, int NumUsuario){

        int i,j,N,O;
        int x,y;
        float *Vcuant,*Vmuest,*Vbin;
        char cCuant[128];
        char cBin[128];

   char path1[100] = "C:/";                    //  Path con el directorio destino, para el archivo características (Valores Cuantificados).
        char path2[100] = "C:/";                    //  Path con el directorio destino, para el archivo características (Valores Binarios).
```

```
FILE *fileDat, *fileCuant, *fileBin;                    // Objetos File, para la generación y guardado del archivo.

Vcuant          = pImageWVector;                        // Puntero de vector con los valores cuantificados.

Vbin    = pVectorWBinario;                              // Puntero de vector con los valores binarios.

char strNumUsuario[2];

snprintf(strNumUsuario, 3, "%d", NumUsuario);           // Cast a string del numbero del usuario.

strcat (path1,strNumUsuario);

strcat (path1,"LCuant.txt");                            // Se concatena al número del usuario el string "LCuant.txt".

strcat (path2,strNumUsuario);

strcat (path2,"LBin.txt");                              // Se concatena al número del usuario el string "LBin.txt".

for (i=0;i<128;i++){                                    // Los datos son almacenados inicialmente en un vector de carácteres.
        j = ( (int)(*(Vcuant+i)) ) + 48;

        cCuant[i] = (char)j;                            // cCuant, vector de valores cuantificados.

        j = ( (int)(*(Vbin+i)) ) + 48;
```

```
            cBin[i] = (char)j;                          //  cBin, vector de valores
binarios.
        }

        if(M == 0){                                     //  Si el índice M es 0, se inícia a
partir de un documento en blanco.

            fileCuant = fopen(path1, "w");              //  Se abre el nuevo
archivo ("w").
            fseek(fileCuant, M, SEEK_SET);
            N = fwrite(cCuant , sizeof(char), 128, fileCuant);     //  Se escriben los
datos, uno tras otro.
            fclose(fileCuant);                          //  Cierre del Archivo.

            fileBin = fopen(path2, "w");                //  Se abre el nuevo archivo
("w").
            fseek(fileBin, M, SEEK_SET);
            O = fwrite(cBin , sizeof(char), 128, fileBin);         //  Se escriben los
datos, uno tras otro.
            fclose(fileBin);                            //  Cierre del Archivo.

        }else{                                          //  Sino se abre el archivo existente y
se concatenan los nuevos 128 datos.

            fileCuant = fopen(path1, "a");              //  Se abre el archivo
existente ("a").
            fseek(fileCuant, M, SEEK_SET);
```

```
            N = fwrite(cCuant , sizeof(char), 128, fileCuant);     //  Se escriben los datos, uno tras otro.

            fclose(fileCuant);                                     //  Cierre del Archivo.


            fileBin = fopen(path2, "a");                           //  Se abre el archivo existente ("a").

            fseek(fileBin, M, SEEK_SET);

            O = fwrite(cBin , sizeof(char), 128, fileBin);         //  Se escriben los datos, uno tras otro.

            fclose(fileBin);                                       //  Cierre del Archivo.

        }

}


void CargarVector(){


        int i,j,N,M;

        int x,y;

        float *V;

        char c[128];


        V = pImageWVector;



        FILE *file = fopen("D:/JuanC/Dropbox/Proyect_Completo/20-10-2013_Gauss+EncPupilaIris(Grad)+Normalización - copia/Vector.txt", "r");

        N = fread(c,sizeof(char),128,file);
```

```
        printf("N = %d", N);
        fclose(file);

        for(i=0;i<128;i++)
                *(V+i) = 0;

        M = 0;
        for (i=0;i<128;i++){
                //*(V+i) = ( (int)(*(V+i)) ) + 48;
                printf("%c \n", c[i]);
        }

}
```

ANEXO 12
Gradiente.c

```
/*
-----------------------------------------------------------------------

-----------------------------------------------------------------------

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

-----------------------------------------------------------------------

-----------------------------------------------------------------------

Script que contiene la función

Gradiente(float *Pin,float *Pout, int M, int N). Función que aplica un
filtro gradiente, es decir, la imagen de salida (Pout), corresponde a la
magnitud del operador Sobel de la imagen de entrada (Pin). Este tipo de
operadores es usado para la detección de bordes.

Para encontrar la magnitud del gradiente para cada punto está definido
como:
```

G = RaizCuadrada (Gx^2 + Gy^2)

Donde, Gx y Gy, representan para cada punto las aproximaciones horizontal y vertical de las derivadas de intensidades, y estos valores son calculados aplicando convolución con la máscara correspondiente a derivada, para la derivada horizontal se tiene una máscara:

Mascara Gx =

-1 0 1

-2 0 2

-1 0 1

Mascara Gy =

-1 -2 -1

0 0 0

1 2 1

---

*/

```
#include "math.h"
#include "Etiquetas.h"
```

```
//  El filtro recorre cada uno de los pixels en la imagen con frontera cerrada,
//  es decir, los pixels correspondientes a las 2 filas y columnas exteriores
//  de la imagen, no serán filtrados.
float Gradiente(float *Pin,float *Pout, int M, int N)
{
   // Definición de variables Globales.
        float *Gmag,*In,*Thetam,*EdgeNoise,*Edge;
        float Min,Max,High,Low;
        float AcumX,AcumY;
        float Mx[9] = {-1,0,1,-2,0,2,-1,0,1};          //  Vector Máscara para
derivada Horizontal (Gx).
        float My[9] = {-1,-2,-1,0,0,0,1,2,1};          //  Vector Máscara para
derivada Vertical   (Gy).
        float Grad,Res;
        float Pi;

        int i,j,k,Ng,Mg,Nn,Mn,Ne,Me;
        int iF,iG,jF,jG;
        int x,y,alto,ancho;
        int Theta;

        Min = 0;
        Max = 0;
```

```
In              = Pin;
Gmag            = Pout;

for(i=0;i<(M*N);i++)                    // Inicialmente, se copia la imagen de entrada a la de salida.
    *(Gmag + i) = 0;                    // Esto para no alterar los valores de los pixels en las 2 filas
                                        //  y columnas de los bordes de la imagen.

 for(j=1;j<M-1;j++){                    // Recorrido de la matriz con frontera cerrada.
     for(i=1;i<N-1;i++){
         AcumX = 0;                     // Valor del Pixel derivada Horizontal.
         AcumY = 0;                     // Valor del Pixel derivada Vertical.
         k = 0;

         ///////      Filtro Gradiente      /////////////

         for(y=(j-1);y < (j + 2);y++)   // Se recorren los 9 pixels de la máscara y para cada uno
             for(x=(i-1);x < (i + 2);x++){
                 AcumX = AcumX + ( *(In + x+(y*N) ) ) * Mx[k];  // se realiza el cálculo y se añade a la sumatoria total.
                 AcumY = AcumY + ( *(In + x+(y*N) ) ) * My[k];
```

```
                    k++;
                }

                ////////////////////////////////////////////////////////

                Grad = sqrt( pow(AcumX,2) + pow(AcumY,2) );          //  Cálculo de la Mágnitud del Gradiente.

                *(Gmag + i+(j*N) ) = Grad;                           //  Se almacena el valor final del pixel en el vector de salida.

                if(Min > Grad)                                       //  Se valida si el nuevo valor es menor al menor actual.
                    Min = Grad;                                      //  Se guarda el valor de este pixel.
                }
            }

            return Min;
}
```

ANEXO 13
Initialization.c

```
/*
-----------------------------------------------------------------------

-----------------------------------------------------------------------

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

-----------------------------------------------------------------------

-----------------------------------------------------------------------

Inicialización de los módulos del DSP. para la inicialización del DSP
se cuenta con básicamente  básicamente dos. Init_PLL(void), la cual
con  Figura  la velocidad de reloj del núcleo y vector de interrupción.

La función Init_SDRAM(void), se encarga de inicializar el módulo SDRAM
en el cual se encuentrna los bancos de memoria externa que se usarán
para almacenar las imagenes y vectores usados. Las posiciones de memoria
reservados para la SDRAM son establecidos y etiquetados en el archivo
Etiquetas.h
```

```
---------------------------------------------------------------------

---------------------------------------------------------------------

                                                                */

#define _USE_LEGACY_CDEF_BEHAVIOUR

#include <cdefBF533.h>
#include <ccblkfn.h>
#include <sysreg.h>
#include "math.h"
#include "Etiquetas.h"
#include "Slave_Global.h"

volatile DMA_Descriptor_Short DMA_1,DMA_2;
int *start_address;

void Init_PLL(void)
{
        int a;

        sysreg_write(reg_SYSCFG, 0x32);                         // Registro
para la con  Figura ción de la inicialización del sistema
```

```
	a = *pIMASK;


	*pIMASK = 0x0;
	*pSIC_IWR = 0x1;


	*pPLL_CTL = 0x2000;
	ssync();
	idle();
	*pIMASK = a;
}



void Init_SDRAM(void)
{
                                        //Registro de Control de frecuencia de Actualización.
	if (*pEBIU_SDSTAT & SDRS)
	{


		*pEBIU_SDRRC = 0x00000817;          //Registro de Control de bancos de Memoria.


		*pEBIU_SDBCTL = 0x00000013;         //Registro de Control de la Memoria Global.
```

```
                *pEBIU_SDGCTL = 0x0091998d;                    //Registro
de Control de la Memoria Global.


                ssync();

        }

}
```

ANEXO 14
Normalizar.c

```
/*
-----------------------------------------------------------------------

-----------------------------------------------------------------------

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

-----------------------------------------------------------------------

-----------------------------------------------------------------------


Normalizar(int Ms,int Ns, int RP, int RI), función encargada de la

normalización del iris, partiendo de la región comprendida entre la

Pupila y el Iris.


Ms es el ancho de la imagen normalizada deseada

Ns es el alto de la imagen normalizada deseada

RP es el radio de la Pupila (Radio Inferior)

RI es el radio del Iris      (Radio Exterior)
```

El algoritmo consta de dos partes. La inicial, es encargada de generar los puntos en coordenadas polares que representan la región del Iris según los valores de alto (Ns) y ancho (Ms) elegidos para la normalización.

Entonces, la componente angular estará relacionada con el número de filas de la imagen normalizada (Ns), entonces, este angulo variará de 0 a 2*pi e irá en pasos de (2*3.14159)/Ns. Estos valores serán almacenados en el apuntador (Theta).

La componente radial estará relacionada con el número de columnas de la imagen normalizada (Ms), entonces, este radio variará de RP a RI e irá en pasos de 1/RI. Estos valores serán almacenados en el apuntador (Theta).

Luego para cada uno de estos se calcula su valor correspondiente en coordenadas cartesianas, realizando una conversión sencilla de coordenadas usando las funciones cos() y sin() de la librería math.h de C. Entocnes, se obtienen dos matrices con los valores X y Y de los puntos finales de la imagen normalizada. Pero estos valores son números decimales, los cuales no pueden representar una matriz.

La segunda parte del algoritmo se encarga entonces de interpolar estos valores a partir de la imagen original, realizando una interpolación

bilineal de la matriz imagen de entrada (puntero I), para los puntos hallados en el paso anterior.

---

*/

```
#include "math.h"
#include "Etiquetas.h"

void Normalizar(int Ms,int Ns, int RP, int RI){

  //  Definición de Variables Locales.
      int i,j,k,x,y;
      int AngDiv, AngularD, RadPix, RadiusP;

      float *I,*INorm;
      float *Theta,*Temp,*Rmat,*x0,*y0;

      float N;
      float X,Y;
      float X1,Y1,X2,Y2,Q11,Q12,Q21,Q22;
      float Pmax,Aux;
```

```
unsigned char *IS;


I              = pImagef6;                  //  Apuntador Imagen de entrada.

INorm          = pImageNfNorm;                  //  Apuntador Imagen de
salida Normalizada.

Theta = pImageNTheta;                       //  Vector Valores de Theta
Coordenadas Polares.

Temp = pImageNTemp;                         //  Vector Valores de Radio
Coordenadas Polares.

Rmat  = pImageNRMat;                       //  Matriz Valores de Radio en
Coordenadas Polares.

x0             = pImageNx0;                 //  Valores X en coordenadas
Cartesianas.

y0             = pImageNy0;                 //  Valores Y en coordenadas
Cartesianas.


IS = pImageS;


x = 120;       y = 120;                    //  Punto central de la Pupila.


AngDiv = Ns;   AngularD  = AngDiv - 1;     //  Variables con el número de
columnas ingresadas

RadPix = Ms;   RadiusP  = RadPix + 2;      //  Variables con el número de
filas ingresadas


*Theta = 0;
```

```
N = (2*3.14159)/AngularD;                    //  Cálculo del tamaño de paso para
Theta.

for(i=1;i<AngDiv;i++)                        //  A partir de este, se construye el
vector Theta.

        *(Theta + i) = *(Theta + (i-1)) + N;



*Temp = 0;

N = 1/( RadiusP - 1 );                       //  Cálculo del tamaño de paso para
Temp, que corresponde a los valores radiales.

for(i=1;i<RadiusP;i++)                       //  A partir de este, se construye el
vector Temp (Radio).

        *(Temp + i) = *(Theta + (i-1)) + N;



for(i=1;i<(RadiusP-1);i++)                   //  Ahora se construye la Matriz de
Valores de Radio en Coordenadas Polares.

        for(k=0;k<AngDiv;k++)

                * (Rmat + k + ((i-1)*Ns) ) = ( (RI - RP) * (*(Temp + i) ) ) + RP;



for(i=0;i<AngDiv;i++)

        for(j=0;j<RadPix;j++){               //  Luego esta matriz es convertida
a coordenadas cartesianas.
```

```
                                        //  Cálculo de las coordenadas en X.
                    *(x0 + i + (j*Ns) ) = ( (* (Rmat + i + (j*Ns) )) * cos(*(Theta + i)) )
+ x;
                                        //  Cálculo de las coordenadas en Y.
                    *(y0 + i + (j*Ns) ) = y - ( (* (Rmat + i + (j*Ns) )) * sin(*(Theta + i))
);

            }

  ///////////////  Algoritmo de Interpolación Bilineal  ////////////////////////////////////////////////////

  //  A partir de la matriz de valores en coordenadas cartesianas, se interpolan
  //  Estos valores a partir de los valores de la imagen de entrada.

      for(i=0;i<RadPix;i++){
            for(j=0;j<AngDiv;j++){

            X = *(x0 + j + (i*Ns) );          //  Valor Coordenada en X imagen
interpolada.
            Y = *(y0 + j + (i*Ns) );          //  Valor Coordenada en Y imagen
interpolada.

            //  Valores los llevados a valores enteros, para ser evaluados en la
      //  imagen de entrada.
            X1 = floor(X);
```

```
                Y1 = floor(Y);


                X2 = X1 + 1;
                Y2 = Y1 + 1;


                //  Luego se extraen los valores de intensidad de la imagen de entrada
                //  evaluados en los puntos anteriores.
                Q11 = *(I + (int)(X1 + (Y1*241)) );
                Q12 = *(I + (int)(X1 + (Y2*241)) );
                Q21 = *(I + (int)(X2 + (Y1*241)) );
                Q22 = *(I + (int)(X2 + (Y2*241)) );


                //  Se calcula el valor que corresponde al punto interpolado entre
        //  los cuatro puntos de la imagen de entrada.
                *(INorm + j + (i*Ns) ) = ((1/((X2-X1)*(Y2-Y1)))*(Q11*(X2-X)*(Y2-Y) +
Q21*(X-X1)*(Y2-Y) + Q12*(X2-X)*(Y-Y1) + Q22*(X-X1)*(Y-Y1)));
                }
        }


    //  Ahora, la imagen de salida es recortada, tomando los cuadrantes 3 y 4
    //  del Iris, esto para eliminar la región superior del Iris, el cual
    //  a menudo es obtruido por factores como las pestañas.
        for(j=0;j<Ms;j++){
                for(i=0;i<Ns/2;i++){
```

```
                    *(I + i + (j*Ns/2)) = *(INorm + (i+(Ns/2)) + (j*Ns));
                }
            }
}
```

ANEXO 15
PillBox.C

/*

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

Script que contiene la función

PillBox(float *Pin,float *Pout,int M,int N). Función que aplica un filtro espacial a la imagen de entrada (Pin) y el resultado es almacenado en el apuntador (Pout). Este es un filtro de promedio circular, el cual realza las Figura s con contornos circulares en la imagen, acentuando las regiones que limitan la Pupila y el Iris, y el Iris con la Esclerótica del ojo.

La matriz máscara usada de dimensiones 5x5, es la siguiente:

0 0.0170159174816308 0.0381149714439322
0.0170159174816308 0

0.0170159174816308 0.0783813541603717 0.0795774715459477
0.0783813541603717 0.0170159174816308

0.0381149714439322 0.0795774715459477 0.0795774715459477
0.0795774715459477 0.0381149714439322

0.0170159174816308 0.0783813541603717 0.0795774715459477
0.0783813541603717 0.0170159174816308

0 0.0170159174816308 0.0381149714439322
0.0170159174816308 0

---*/

```
#include "math.h"
#include "Etiquetas.h"

//  El filtro recorre cada uno de los pixels en la imagen con frontera cerrada,
//  es decir, los pixels correspondientes a las 2 filas y columnas exteriores
//  de la imagen, no serán filtrados.
float *PillBox(float *Pin,float *Pout,int M,int N){

   //  Definición de las Variables.
        int i,j,k,x,y;
        float *In, *Out;
        float *Min;
```

Min = (float*)malloc (4*sizeof(float)); // Min[0] -> Coordenada x de Pixel con Valor Minimo

// Min[1] -> Coordenada y de Pixel con Valor Minimo

// Min[2] -> Valor Pixel Minimo

// Min[1] -> Valor Pixel Maximo

In = Pin;

Out = Pout;

int Ns;

// Definición de la máscara.

float Mk[25] = { 0,0.0170159174816308,0.0381149714439322,0.0170159174816308,0,

0.0170159174816308,0.0783813541603717,0.0795774715459477,0.0783813541603717,0.0170159174816308,

0.0381149714439322,0.0795774715459477,0.0795774715459477,0.0795774715459477,0.0381149714439322,

0.0170159174816308,0.0783813541603717,0.0795774715459477,0.0783813541603717,0.0170159174816308,

0,0.0170159174816308,0.0381149714439322,0.0170159174816308,0

```
        };
    float Acum;


    Ns = N - 4;
    Min[2] = 255;
    Min[3] = 0;


    for(i=0;i<(M*N);i++)                    //  Inicialmente, se copia la imagen de entrada a la de salida.
        *(Out + i) = *(In + i);             //  Esto para no alterar los valores de los pixels en las 2 filas
                                            //  y columnas de los bordes de la imagen.


    for(j=2;j<M-2;j++)
        for(i=2;i<N-2;i++){                 //  Recorrido de la matriz con frontera cerrada.
            k = 0;
            Acum = 0;                       //  Valor del Pixel.
            for(y=(j-2);y < (j + 3);y++)
                for(x=(i-2);x < (i + 3);x++){       //  Se recorren los 25 pixels de la máscara y para cada uno
                    Acum = Acum + ( *(In + x+(y*N) ) ) * Mk[k];   //  se realiza el cálculo y se añade a la sumatoria total.
                    k++;
                }
```

```
                    *(Out + i+(j*N) ) = Acum;                    //  Se almacena el
valor final del pixel en el vector de salida.


                    if(Min[2] > Acum ){                    //  Se valida si el nuevo
valor es menor al menor actual.
                        if( (i>=4) && (i<=(N-5)) ){                //  No se toman en
cuenta los pixels en la frontera
                            if( (j>=4) && (j<=(M-5)) ){                //  (las 2 filas
y columnas exteriores de la imagen).
                                Min[2] = Acum;                //  Se guarda
el valor y las coordenadas de este pixel.
                                Min[0]= i;
                                Min[1]= j;
                            }
                        }
                    }


                    if(Min[3] < Acum )                    //  Se valida si el nuevo
valor es mayor al mayor actual.
                        Min[3] = Acum;                    //  Se guarda el
valor y las coordenadas de este pixel.


                }
        return (Min);                    //  Se retorna el resultado.

}
```

ANEXO 16
Slave_Global.h

```
#define pFlashA_PortA_Dir       ((volatile unsigned short *)0x20270006)
#define pFlashA_PortA_Out       ((volatile unsigned char *)0x20270004)
void Init_SDRAM(void);
void Init_DMA_OUT(void);
void Setup_DMA_IN(void);
void Init_PLL(void);
float Gradiente(float *Pin,float *Pout, int M, int N);
float *PillBox(float *Pin,float *Pout,int M,int N);
int Circulo(int x0, int y0, int r);
int EncontrarIris(float *Pin, int r);
int *EncontrarCentroPupila(float x0, float y0, float Max);
int *Circuncentro(float x1, float y1, float x2, float y2, float x3, float y3);
void Recortar(unsigned char *Pin,float *Pout);
void Mostrar(float *Pin, unsigned char *Pout,float Pmin, int M, int N);
void WaveletHaar2D(int L,int M,int N);
void GuardarVector(int M, int NumUsuario);
void CargarVector(void);
void Caracterizacion(void);
void carga( void );
unsigned int OutputLine(volatile unsigned int *p, unsigned int line);
unsigned char * Insert_Line(volatile unsigned char *p_out , unsigned int line);
```

ANEXO 17

WaveletHaar2D.C

/*

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

WaveletHaar2D(int L,int M,int N), función encargada de realizar a la imagen normalizada del Iris, la transformada de Wavelet de Haar de 4º nivel. Para un tamaño de imagen de entrada de 128x64 pixels, la tranformada de 4º nivel da como resultado una imagen de 16x8 pixels la cual da como resultado un vector de 128 valores, los cuales representan las caracteristicas de la imagen.

Esta función también se encarga de, a partir de este vector de 128 valores, cuantificar estos de dos maneras:

Cuantificación para Análisis Euclidiano :

Esta cuantificación, lleva cada uno de los valores del vector resultado de la transformación a valores de 0, 1, 2, 3, 4, 5:

Si el valor se encuentra entre -500 y 0

Valor = 0

Si el valor se encuentra entre 0 y 500

Valor = 1

Si el valor se encuentra entre 500 y 1000

Valor = 2

Si el valor se encuentra entre 1000 y 1500

Valor = 3

Si el valor se encuentra entre 1500 y 2000

Valor = 4

Sino

Valor = 5

Cuantificación para Análisis Hamming :

Binarizacion de los vectores resultantes, llevando los valores negativos del vector a 0 y los valores iguales y positivos a 1.

---

*/

```
#include "math.h"
#include "Etiquetas.h"

//  La función recibe el número de niveles para la transformada y el ancho
//  y alto de la imagen.
void WaveletHaar2D(int L,int M,int N){

   //  Definición de variables globales
        int i,j,k,Ms,Ns,S1,S2;
        int Div, AngularD, RadPix, RadiusP;

        unsigned char *IS;

        float *W1,*W2,*INorm,*A;
        float *Vcuant,*Vmuest,*Vbin;
        float valorMuestra,valorCuant,valorBin;
```

```
float R;

W1          = pImageW1;                 // Apuntador imagen parcial transformada vertical.
W2          = pImageW2;                 // Apuntador imagen parcial transformada horizontal.
A           = pImagef6;                 // Imagen de entrada.

Vcuant      = pImageWVector;            // Apuntador al vector de valores resultado cuantificado.
Vbin    = pVectorWBinario;              // Apuntador al vector de valores resultado binarizado.

Ms = M;
Ns = N;                                 // Alto y ancho de imagen de entrada.

S1 = M;
S2 = N;                                 // Valores ancho y alto de las regiones a tranformar.

Div = 1;
R = sqrt(2);

for(k=0;k<L;k++){                       // La transformada se realiza L veces. (Niveles de la transformada).
```

```
            if(k>0){                                    // Para la primer iteración no se
cumple la condición.
                    for(i=0;i<(M*N);i++)                // Se actualiza la imagen
de entrada, copiando el resultado de la iteración
                            *(A + i) = *(W2 + i);       // anterior (W2) en
la imagen (A).
                Div=2;                                  // Ahora, se seguirá dividiendo en 2
la región a transformar.
            }

            S1 = S1/Div;
            S2 = S2/Div;                                // Se aplica la división.

            for(i=0;i<((S1/2));i++){                    // Se aplica la transformada
vertical.
                    for(j=0;j<(S2);j++){

                            *(W1 + j + (i*N)) = ( ( *(A + j + ((i*2)*N)) ) + ( *(A + j +
(((i*2)+1)*N)) ) )/R;

                            *(W1 + j + ((i+(S1/2))*N)) = ( ( *(A + j + ((i*2)*N)) ) - ( *(A
+ j + (((i*2)+1)*N)) ) )/R;
                    }
            }

            for(i=0;i<(S1);i++){                        // Se aplica la transformada
horizontal.
```

```
                    for(j=0;j<(S2/2);j++){

                        *(W2 + j + (i*N)) = ( ( *(W1 + (j*2) + (i*N)) ) + ( *(W1 + ((j*2)+1) + (i*N)) ) )/R;

                        *(W2 + (j+(S2/2)) + (i*N)) = ( ( *(W1 + (j*2) + (i*N)) ) - ( *(W1 + ((j*2)+1) + (i*N)) ) )/R;
                    }
                }
            }

            /*
```

Para este punto, la transformada está completa. Los valores de esta, se encuentran en la matriz

comprendida por las primeras 18 filas y 8 columnas de la imagen resultante (W2).

El siguiente segmento del algoritmo se encarga de guardar en dos vectores (Vcuant y Vbin)de

128 cada uno (16x8) los valores equivalentes para; Cuantificación para Análisis Euclidiano

y Cuantificación para Análisis Hamming.

```
            */
            k=0;
            for(j=0;j<8;j++){
                for(i=0;i<16;i++){
```

```
valorCuant =  *(W2 + i + (j*N)) ;                    // Se extrae el valor del pixel.

valorBin =  *(W2 + i + (j*N)) ;


                                                    // Cuantificación para Análsis Euclidiano.

if( (valorCuant >= -500) && (valorCuant < 0) ){
        valorCuant = 0;
}else if( (valorCuant >= 0) && (valorCuant < 500) ){
        valorCuant = 1;
}else if( (valorCuant >= 500) && (valorCuant < 1000) ){
        valorCuant = 2;
}else if( (valorCuant >= 1000) && (valorCuant < 1500) ){
        valorCuant = 3;
}else if( (valorCuant >= 1500) && (valorCuant < 2000) ){
        valorCuant = 4;
}else{
        valorCuant = 5;
}


                                                    // Cuantificación para Análsis Hamming.

if( valorBin < 0 ){
        valorBin = 0;
}else {
        valorBin = 1;
```

```
                    }


                    *(Vcuant + k) = valorCuant;                    // Se almacenan los
valores.
                    *(Vbin + k) = valorBin;
                    k++;
                }
            }
}
```

ANEXO 18
Validacion.c

```
/*
------------------------------------------------------------------------
------------------------------------------------------------------------
UNIVERSIDAD TECNOLÓGICA DE PEREIRA
FACULTAD DE INGENIERIAS
INGENIERIA ELECTRONICA
PROYECTO DE GRADO FINAL
SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP
ARLES FELIPE GARCIA MAYA
JUAN CAMILO MORENO RUIZ
2014
------------------------------------------------------------------------
------------------------------------------------------------------------

Función Main para validación. Las imagenes a validar serán cargadas
mediante la herramienta ImageViewer del VisualDSP++.

Los vectores de caracteristicas del usuario registrado son cargadas
desde archivos de texto plano almacenados en el equipo. Por usuario se
tienen 20 bases de datos, es por esto que el proyecto realiza la
validación para 20 imagenes de entrada (Cargadas con la herramienta
ImageViewer).
```

La función consta básicamente de dos partes, la primera (función Registro()) es la extracción de las caracteristicas de la imagen a partir de la transformada Wavelet Haar del iris resultante de la segmentación. La segunda parte genera los archivos planos de texto con los valores de cuantificación elegidos.

---

*/

```
#include "ccblkfn.h"
#include "sysreg.h"
#include "math.h"
#include "Etiquetas.h"
#include "Slave_global.h"
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

//  Función que realiza la validación de la imagen cargada actualmente y retorna
//  un valor booleano.
extern bool Verificacion(int BDusuario, int NumUsuario);
```

```
void main(void)
{

    // Con  Figura ción del DSP.
    sysreg_write(reg_SYSCFG, 0x32);
    Init_PLL();              // Módulo de Con  Figura ción de velocidad del reloj del
                                                    // Nucleo y del Sistema

    Init_SDRAM();            // SDRAM, Módulo de memoria de trabajo.

    volatile clock_t clock_start; // Variables necesarias para obtener los tiempos
    volatile clock_t clock_stop; // de ejecución de los segmentos del algoritmo.
    double secs;

    //  Definición de variables Globales.
    int M = 1;
    int Val=0;
    int NoVal=0;
    bool Validado;
```

```
//  Se ingresa el número del usuario a validar.
int NumUsuario;
printf("\n Ingrese El Numero del Usuario a Validar \n");
scanf("%d", &NumUsuario);

while(M<=20){
        clock_start = clock();                  //  Punto de inicio para obtener el tiempo de ejecucion.

        Validado = Verificacion(M,NumUsuario);          //  Se llama la función Verificación que retorna el resultado
                                                //  de la validación.

        clock_stop = clock();                   //  Punto final de ejecución.

        if(Validado){                           //  Si es validado o no. Se muestra el resultado en la consola.
                printf("\nAutorizado      -- Muestra %d \n",M);
                Val++;
        }else{
                printf("\nNo Autorizado   -- Muestra %d \n",M);
                NoVal++;
        }
        secs = ((double) (clock_stop - clock_start))/ CLOCKS_PER_SEC;
```

```
                                                    //      El tiempo de ejecución
en segundos es igual a la

//      diferencia de ciclos entre los dos puntos dividido

//      la unidad de de tiempo de la tarjeta (Ciclos/Segundo).
            M++;
        }
        printf("\n\nAutorizados = %d",Val);
        printf("\n\nNo Autorizados = %d",NoVal);                    // Se muestran los
resultados totales.
}
```

ANEXO 19
Verificacion.C

```
/*
------------------------------------------------------------------------

------------------------------------------------------------------------

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

------------------------------------------------------------------------

------------------------------------------------------------------------

Función Verificacion(int BDusuario, int NumUsuario), de tipo Bool que
retorna true o false, dependiendo del resultado de la verificación.
Esta función se encarga de, a partir de la imagen cargada actualmente
(Mendiente la herramienta ImageViewer), caracterizar esta, y realizar
el proceso de validación con la base de datos elegida mediante los
valores de entrada (BDusuario y NumUsuario).

------------------------------------------------------------------------

------------------------------------------------------------------------
```

```
*/

#include "ccblkfn.h"
#include "sysreg.h"
#include "math.h"
#include "Etiquetas.h"
#include "Slave_global.h"
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

// Funciones usadas.
extern float Gradiente(float *Pin,float *Pout, int M, int N);
extern void Recortar(unsigned char *Pin,float *Pout);
extern void RecortarIris(unsigned char *Pin,float *Pout,int x0,int y0);
extern float *Gaussiano(float *Pin,float *Pout,int M,int N);
extern void Mostrar(float *Pin, unsigned char *Pout,float Pmin, int M, int N);
extern int Circulo(int x0, int y0, int r);
extern int SemiCirculo(int x0, int y0, int r);
extern int EncontrarIris(float *Pin, int r);
extern int *EncontrarCentroPupila(float x0, float y0, float Max);
extern void Normalizar(int Ms,int Ns, int RP, int RI);
```

```
extern void WavelHaar(int L,int M,int N);
extern void CargarVectorBaseDatos(int BDusuario, int NumUsuario);
extern void Registro(void);
extern void CalcularErrores(void);
extern float Promedio(float *Pin,int Ini,int N);
extern float Desvest(float *Pin,int N);

//  Esta función recibe un número de usuario al cual se cargará la base de datos.
//  Esta será la base de datos con la cual se verificará la identidad de la
//  imagen ingresada (Cargada por medio de herramienta ImageViewer).
//  También recibe un índice BDusuario, para cargar una determinada base de datos
//  del usuario elegido.
bool Verificacion(int BDusuario, int NumUsuario)
{
        volatile clock_t clock_start;                   //Variables necesarias para obtener los tiempos
        volatile clock_t clock_stop;                    //  de ejecución de los segmentos del algoritmo.
        double secsCar,secsVal;

        //  Definición de variables globales.
        int i,j,M,N,Au,Nau;

        float *VEbd,*VE,*EE1,*EE2,*DesvEE2;
        float ME2,STDE2,UE1,UE2;
```

```
float MH2,STDH2,UH1,UH2;
float *VHbd,*VH,*EH1,*EH2,*DesvEH2;

EE1        =    pVectorE1;
EE2        =    pMatrizError2;
DesvEE2    =    pVectorDesvError2;

EH1        = pVectorErrorBin1;
EH2        = pMatrizErrorBin2;
DesvEH2    = pVectorDesvErrorBin2;

clock_start = clock();        //    Punto de inicio para obtener el tiempo de
                                    //    ejecución de la función Registro()
Caracterizacion();        //    Función Caracterizacion() que se encarga de segmentar,
                                    //    Normalizar y caracerizar la imagen cargada actualmente
                                    //    mediante la herramienta ImageViewer.
clock_stop = clock();        //    Punto final de ejecución.
secsCar = ((double) (clock_stop - clock_start))/ CLOCKS_PER_SEC;
                    //    El tiempo de ejecución en segundos es igual a la
```

```
                                                    //      diferencia      de
ciclos entre los dos puntos dividido
                                                    //      la unidad de de
tiempo de la tarjeta (Ciclos/Segundo).

        clock_start = clock();      //      Punto de inicio para obtener el tiempo de
                                                    //      ejecución      del
segmento Validación.

                                                    //      Se      cargan      los
vectores de Base de Datos del usuario Registrado.
        CargarVectorBaseDatos(BDusuario, NumUsuario);

                        //  Cálculo de los Errores.
        CalcularErrores();
                            //  Cálculo promedios.
        ME2 = Promedio(EE2,0,361);
        MH2 = Promedio(EH2,0,361);

        for(i=0;i<19;i++){          //  Cálculo desviación estandar de los vectores.
                *(DesvEE2 + i) = Desvest(EE2,(i*19));
                *(DesvEH2 + i) = Desvest(EH2,(i*19));
        }

        STDE2 = Promedio(DesvEE2,0,19); //  Desviación estandar promerio.
        STDH2 = Promedio(DesvEH2,0,19);
```

```
        UE1=ME2;
        UH1=MH2;

        UE2=Promedio(EE1,0,19);        //   Promedio de distancias.
        UH2=Promedio(EH1,0,19);

        if ( (UH2 < 0.17) ){          //   Validación Final.
                        //   Se deben cumplir los dos criterios (Euclidian y Hemming)
                        //   Para poder ser aceptado.

                return true;              //   Se retorna el resultado.
        }else{
                return false;             //   Se retorna el resultado.
        }
}
```

Caracterizacion.C

/*

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

Función Caracterizacion() encargada de filtrar, segmentar, normalizar y caracterizar la imagen de entrada. Para este punto, la imagen ha sido cargada en memoria. Al finalizar el proceso esta función genera los vectores de caracteristicas de cada imagen y las almacena los vectores de caracteristicas en en memoria haciendo uso de apuntadores.

Los vectores de caracteristicas resultantes representan al usuario que son aceptados por el sistema de reconocimiento.

```
---------------------------------------------------------------------

---------------------------------------------------------------------

*/

// Librerias Nativas de C++ y VisualDSP

#include "ccblkfn.h"

#include "sysreg.h"

#include "math.h"

#include <time.h>

#include <stdio.h>

#include <stdlib.h>

#include <stdint.h>


#include "Etiquetas.h"                    // Archivo con la
definición de los punteros usados.

#include "Slave_global.h"                 // Archivo con la definición de
todas las funciones

                                          // utilizadas
en el proyecto.


// Se importan las funciones que serán usadas.


extern float Gradiente(float *Pin,float *Pout, int M, int N);

extern void Recortar(unsigned char *Pin,float *Pout);

extern void RecortarIris(unsigned char *Pin,float *Pout,int x0,int y0);
```

```
extern float *PillBox(float *Pin,float *Pout,int M,int N);
extern void Mostrar(float *Pin, unsigned char *Pout,float Pmin, int M, int N);
extern int Circulo(int x0, int y0, int r);
extern int SemiCirculo(int x0, int y0, int r);
extern int EncontrarIris(float *Pin, int r);
extern int *EncontrarCentroPupila(float x0, float y0, float Max);
extern void Normalizar(int Ms,int Ns, int RP, int RI);
extern void WaveletHaar2D(int L,int M,int N);
extern void GuardarVector(int M, int NumUsuario);
extern void CargarVector(void);

void Caracterizacion()
{
	// Definición de variables locales.
	int i,j;
	int n,m,N,k;
	n = 640;
	m = 480;
	N = n*m;
	int x,y,r,rP,rI;
	int *pX,*pY,*A;
	unsigned char *I,*IS,*If6S;
	float *If,*IfS,*If1,*If2,*If3,*If4,*If5,*If6;
```

```
float Min;

float *CentroPupila;


I = pImage;                    // Imagen en unsigned char inicial de
Entrada     (Cargada con ImageViewer).

IS = pImageS;                  // Imagen en unsigned char de salida
      (Para Visualización)


If = pImagef;              // Imagen en Float inicial de Entrada
(Recortada).

IfS = pImagefS;                // Imagen en Float final de Salida
      (De Trabajo)

If6S = pImageSiris;        // Imagen Iris Recortado de Salida
      (De Trabajo)

If1 = pImagef1;                // Imagen Magnitud Gradiente
            (De Trabajo)

If2 = pImagef2;                // Imagen Theta
                  (De Trabajo)

If3 = pImagef3;                // Imagen EdgeNoise
                  (De Trabajo)

If4 = pImagef4;                // Imagen Edge
                        (De Trabajo)

If5 = pImagef5;                // Imagen Flitada Gaussiano 2
            (De Trabajo)

If6 = pImagef6;                // Imagen Iris Recortado
            (De Trabajo)


Recortar(I,If);            // Se recorta la imagen de entrada a un tamaño de
340x210
```

```
CentroPupila = PillBox(If,IfS,210,340);  // Se aplican los filtros de promedio circular
CentroPupila = PillBox(IfS,If,210,340);  // estos devuelven la imagen filtrada y
                                         //
las coordenadas del pixel de menor intensidad
                                         //
para luego ser usado en el algoritmo de
                                         //
segmentación de la pupila.
                                         //
                                         //
CentroPupila[0] -> x0 (Coordenada x pixel de menor intensidad)
                                         //
CentroPupila[1] -> y0 (Coordenada y pixel de menor intensidad)
                                         //
CentroPupila[2] -> Valor Pixel Minimo
                                         //
CentroPupila[1] -> Valor Pixel Maximo


Gradiente(If,If1,210,340);               // Filtro Gradiente para el realce de los bodes.


A = EncontrarCentroPupila(CentroPupila[0],  // Se determina el centro de la pupila a partir de
                          CentroPupila[1],  // hallar la envolvente convexa que contiene
```

CentroPupila[3]); // al pixel semilla encontrado en el filtro anterior

//

y trazar el círculo que encierra esta región.

//

A[0] -> x0 (Coordenada x Centro Radio Pupila).

//

A[1] -> y0 (Coordenada y Centro Radio Pupila).

//

A[2] -> Valor Radio de la Pupila.

rP = A[2]; //
Radio de la Pupila.

RecortarIris(I,If6,A[0],A[1]); // Se recorta de nuevo la imagen inicial de entrada

//

pero ahora centrada en el centro de la pupila

//

hallado en el paso anterior.

CentroPupila = PillBox(If6,If3,241,241); // Se aplican de nuevo los filtros de promedio circular

CentroPupila = PillBox(If3,If4,241,241); // y realce de bordes para la posterior segmentación del iris.

Min = Gradiente(If4,If3,241,241);

```
        rI = EncontrarIris( If3 , rP);                  //  Se halla el radio del iris
partiendo de la pupila.


        Normalizar(64,256,rP,rI);                       //  Se  obtiene  la
imagen correspondiente al iris en una nueva
                                                                                //
        imagen de 256x64 pixels.


        WaveletHaar2D(4,64,128);                        //
        Transformada de Wavelet Haar a la imagen del Iris, tomando
                                                                                //
        la mitad inferior del iris, esto para excluir la region
                                                                                //
        superior que normalmente está obtruido por las pestañas
}                                                                               //
        del usuario.
```

Circulo.C

/*

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

Script que contiene las funciones para generar círculos o segmentos de estos, partiendo de las coordenadas x0,y0 y radio (r) del círculo y generando el conjunto de puntos (x,y) que pertenecen al perímetro del círculo. Los puntos son generados usando el método o algoritmo de Bresenham para círculos, el cual, aproxima los puntos de la circunferencia basado en la ecuacíon de esta (x^2 + y^2 = r^2) tomando para cada punto el de menor error con respecto al real (ecuación).

```
                ------------------------------------------------------------------------
                ------------------------------------------------------------------------*/


#include "math.h"
#include "Etiquetas.h"

/*
La función Circulo(int x0, int y0, int r), genera todos los puntos que
hacen parte de la periferia de la circunferencia almacenando estos en
en memoria mediante los punteros PixelX y PixelY, para su posterior uso.*/
int Circulo(int x0, int y0, int r){

   // Definición de Variables y Apuntadores
        int i,j,N,M;
        int x,y;
        int *PixelX,*PixelY;
        float E1,E2;

        PixelX = pPixelsX;
        PixelY = pPixelsY;

        N = 0;                                          //   Valor índice para Punteros de
las coordendas X,Y.
```

```
        //  Se genera un primer segmento de circunferencia, perteneciente al primer
    //  cuadrante, comprendido entre los puntos en y=r y x=(r*0.707).
        y = r;
        for( i=0;i<ceil(r*0.707);i++ ){
                                                //  Para el valor en x=i, se hallan los
errores de los dos
                                                //  posibles valores en y, (y) ó (y-1).
                                                //  Error = x^2 + y^2 - r^2
                E1 = fabs( pow(i,2) + pow(y,2) - pow(r,2) );
                E2 = fabs( pow(i,2) + pow((y-1),2) - pow(r,2) );

                if(E2 <= E1)                            //  Se elige el Error de menor
valor.
                        y = y - 1;

                *(PixelY + N) = y0 + y;                 //  Y el punto elegido es
el almacenado en los vectores.
                *(PixelX + N) = x0 + i;
                N++;
        }

        //  Se genera el siguiente segmento de circunferencia, perteneciente al primer
    //  cuadrante, comprendido entre los puntos en y=0 y x=r.
        x = r;
```

```
        for( j=0;j<y;j++ ){
                                                //  Para el valor en y=j, se hallan los errores de los dos
                                                //  posibles valores en x, (x) ó (x-1).
                                                //  Error = x^2 + y^2 - r^2
                E1 = fabs( pow(x,2) + pow(j,2) - pow(r,2) );
                E2 = fabs( pow((x-1),2) + pow(j,2) - pow(r,2) );

                if(E2 <= E1)                    //  Se elige el Error de menor valor.
                        x = x - 1;

                *(PixelY + N) = y0 + j;         //  Y el punto elegido es el almacenado en los vectores.
                *(PixelX + N) = x0 + x;
                N++;
        }

        //  Aquí se tienen los puntos para el primer cuadrante, ahora, para generar
    //  los tres faltantes, se aprovecha la característica simétrica de la
    //  circunferencia para replicar los cuadrantes.

        M = N;
        for(i=1;i<M;i++){
```

```
                                                    //  Réplica del Cuatro cuadrante.
                *(PixelY + N) = (2*y0) - *(PixelY + i);
                *(PixelX + N) = *(PixelX + i);
                N++;


                                                    //  Réplica del Tercer cuadrante.
                *(PixelY + N) = (2*y0) - *(PixelY + i);
                *(PixelX + N) = (2*x0) - *(PixelX + i);
                N++;


                                                    //  Réplica del Segundo cuadrante.
                *(PixelY + N) = *(PixelY + i);
                *(PixelX + N) = (2*x0) - *(PixelX + i);
                N++;


        }
        return (N);
}


/*
La función SemiCirculo(int x0, int y0, int r), genera todos los puntos
que hacen parte de la periferia de el cuarto cuadrante de una
circunferencia, almacenando estos en en memoria mediante los punteros
PixelX y PixelY, para su posterior uso.*/
```

```
int SemiCirculo(int x0, int y0, int r){

        // Definición de Variables y Apuntadores
        int i,j,N,M;
        int x,y;
        int *PixelX,*PixelY;

        float E1,E2;

        PixelX = pPixelsX;
        PixelY = pPixelsY;

        N = 0;                                  //  Valor índice para Punteros de las coordendas X,Y.

        //  Se genera un primer segmento de circunferencia, perteneciente al primer
    //  cuadrante, comprendido entre los puntos en y=r y x=(r*0.707).
        y = r;
        for( i=0;i<ceil(r*0.707);i++ ){
                E1 = fabs( pow(i,2) + pow(y,2) - pow(r,2) );
                E2 = fabs( pow(i,2) + pow((y-1),2) - pow(r,2) );

                if(E2 <= E1)
                        y = y - 1;
```

```
            *(PixelY + N) = y0 + y;
            *(PixelX + N) = x0 + i;
            N++;
      }


      //  Se genera el siguiente segmento de circunferencia, perteneciente al primer
//  cuadrante, comprendido entre los puntos en y=0 y x=r.x = r;
x = r;
      for( j=0;j<y;j++ ){
            E1 = fabs( pow(x,2) + pow(j,2) - pow(r,2) );
            E2 = fabs( pow((x-1),2) + pow(j,2) - pow(r,2) );

            if(E2 <= E1)
                  x = x - 1;

            *(PixelY + N) = y0 + j;
            *(PixelX + N) = x0 + x;
            N++;
      }

      M = N;
      for(i=1;i<M;i++){
```

```
            *(PixelY + N) = *(PixelY + i);
            *(PixelX + N) = (2*x0) - *(PixelX + i);
            N++;

        }

        return (N);
}
```

Circuncentro.c

```
/*
-----------------------------------------------------------------------
-----------------------------------------------------------------------
UNIVERSIDAD TECNOLÓGICA DE PEREIRA
FACULTAD DE INGENIERIAS
INGENIERIA ELECTRONICA
PROYECTO DE GRADO FINAL
SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP
ARLES FELIPE GARCIA MAYA
JUAN CAMILO MORENO RUIZ
2014
-----------------------------------------------------------------------
-----------------------------------------------------------------------

Script que contiene la función para, a partir de los tres vértices de
un triángulo, encontrar la circunferencia que pasa por estos tres.

El circuncentro de un triángulo, es el punto centro del circulo que pasa
por los tres vértices del triángulo.

-----------------------------------------------------------------------
-----------------------------------------------------------------------*/
```

```
#include "math.h"
#include "Etiquetas.h"

    /*
    Para hallar este, se encuentran los puntos medios y posteriormente el
    ortocentro del triangulo formado por los tres vértices. Luego a partir
    de las ecuaciones de las rectas que pasan por los puntos medios y
    el ortocentro del triángulo se obtiene un sistema de ecuaciones 2x2 el
    cual por medio del método de sustitucion se hallan los puntos X y Y del
    circuncentro.
    */
int *Circuncentro(float x1, float y1, float x2, float y2, float x3, float y3){

   // Definición de Variables.
        float m1,m2;
        float Xm1,Ym1;
        float Xm2,Ym2;
        float b1,b2;
        float dx,dy;
        int X,Y;
        int r;

        int *C;
```

```
C = (int*)malloc ( 3*sizeof(int) );          // C[0] -> Coordenada x del centro del Circulo.
                                             // C[1] -> Coordenada y del centro del Circulo.
                                             // C[2] -> Radio de la del Circulo.

m1 = - ((x2-x1)/(y2-y1));                    // Pendientes de las rectas que pasan por los puntos medios.
m2 = - ((x3-x1)/(y3-y1));

Xm1 = (x1 + x2)/2;                           // Pendientes de las rectas que pasan por el circuncentro.
Ym1 = (y1 + y2)/2;

Xm2 = (x1 + x3)/2;
Ym2 = (y1 + y3)/2;

b1 = Ym1 - ( m1*Xm1 );                       // Término independiente de la ecuación de la recta.
b2 = Ym2 - ( m2*Xm2 );

X = ceil( (b1 - b2) / (m2 - m1) );           // Se hallan los valores, usando el método de sustitución.
Y = ceil ( b2 + ( m2 * X ) );
```

```
        dx = fabs(x1-X);                    // Diferencias en X y Y, de un vértice al
circuncentro para
        dy = fabs(y1-Y);                    // hallar la distancia entre el circuncentro y el
vértice
                                            // (radio del círculo).


        r = floor(sqrt( pow(dx,2) + pow(dy,2) ) ); // Se halla el radio.
        C[0] = X;
        C[1] = Y;
        C[2] = r;
        return (C);
}
```

EncontrarCentroPupila.C

```
/*-----------------------------------------------------------------------
-----------------------------------------------------------------------
UNIVERSIDAD TECNOLÓGICA DE PEREIRA
FACULTAD DE INGENIERIAS
INGENIERIA ELECTRONICA
PROYECTO DE GRADO FINAL
SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP
ARLES FELIPE GARCIA MAYA
JUAN CAMILO MORENO RUIZ
2014
-----------------------------------------------------------------------
-----------------------------------------------------------------------

Script que contiene la función que partiendo de un pixel semilla, dentro
de la pupila, halla la envolvente convexa que contiene los pixels que
hacen parte de la pupila.

-----------------------------------------------------------------------
-----------------------------------------------------------------------*/

#include "math.h"
#include "Etiquetas.h"
```

```
//  Función usada para encontrar la circunferencia que más se asemeja a la forma
//  de la pupila.
extern int *Circuncentro(float x1, float y1, float x2, float y2, float x3, float y3);

//  La envolvente se obtiene, analizando la vecindad del pixel semilla y
//  añadiendo al grupo de pixels pertenecientes a la envolvente, aquellos que
//  cumplan un valor de intensidad menor a un umbral establecido.
int *EncontrarCentroPupila(float x0, float y0, float Max){

   //  Definición de las variables.
        int i,j,x,y,H,L,N;
        float Xmin,Xmax,Ymin,Ymax;
        float x1,x2,x3;
        float y1,y2,y3;
        float High;

        int *C;
        float *ImEnv,*ImGauss;
        unsigned short int *PEnvX,*PEnvY;

        C = (int*)malloc ( 3*sizeof(int) );          // C[0] -> Coordenada x del centro de
la Pupila Encontrada.
                                                                  //  C[1]  ->
Coordenada y del centro de la Pupila     Encontrada.
```

// C[2] -> Radio de la Pupila Encontrada.

```
ImEnv      =    pImagef2;
ImGauss    =    pImagefS;
PEnvX      =    pPixEnvolventeX;      // Vectores que contienen las coordenadas X y Y de los pixels
PEnvY      =    pPixEnvolventeY;      // que se hallan dentro de la envolvente convexa.

                                      // Se genera una imagen para guardar los pixels pertenecientes
                                      // a la envolvente.
for(i=0;i<(210*340);i++)
        *( ImEnv + i ) = 0;

N = 340;                              // Ancho de la imagen.
High = 0.23*Max;                      // Umbral establecido para hacer parte de la envolvente.

L = 0;                                // Índice del vector de puntos en la envolvente.
H = 1;                                // Índice que indica la posición hasta la cual
                                      // se han analizado los pixels en el vector de puntos
                                      // en la envolvente.

*(PEnvX + L) = x0;                    // Se agrega el primer pixel (Semilla)
```

```
*(PEnvY + L) = y0;

Xmin = x0;
Ymin = y0;

Xmax = x0;
Ymax = y0;
x1 = x0;                    // Variables para hallar tres puntos en la periferia
y1 = y0;                    // de la región hallada.
x2 = x0;
y2 = y0;
x3 = x0;
y3 = y0;

                            // Se analizan los pixels hasta que los pixels analizados
                            // sea igual a los pixels de la envolvente.
while(L < H){

        i = *(PEnvX + L);           // Pixel perteneciente a la envolvente al cual se
        j = *(PEnvY + L);           // analizará su vecindad.

        *(ImEnv + i + (j*N)) = 255;     // Se cambia el valor en la imagen (Se agrega).
```

```
for( x=(i-1);x<=(i+1);x=(x+2) )          // FOR's para recorrer la vecindad.
    for( y=j;y<=(j+1);y++ ){

// Si el valor del pixel es menor al umbral y aún no hace parte de
// la envolvente, se agrega a esta.
        if( (*(ImGauss + x + (y*N)) < High) && (*(ImEnv + x + (y*N)) == 0) ){

            *(ImEnv + x + (y*N)) = 255;          // Se cambia su valor en la imagen (Se agrega).

            *(PEnvX + H) = x;                    // Se agrega al vector de pixels dentro de la envolvente.
            *(PEnvY + H) = y;
            H = H + 1;

        if(Xmax < x){                // condicionales para encontrar tres puntos en la periferia
            Xmax = x;                // de la envolvente.
            x3 = x;
            y3 = y;
        }
        if(Xmin > x){
            Xmin = x;
            x1 = x;
            y1 = y;
```

```
                    }
                    if(Ymin < y){
                        Ymin = y;
                        x2 = x;
                        y2 = y;
                                        }
                                }
                            }
                L = L + 1;                          // Se aumenta el índice L para analizar
el siguiente pixel.
        }
                                        // Luego, a partir de los tres puntos que hacen
parte de la
                                        // periferia de la region de la pupila, se halla el
círculo
                                        // que pasa por estos tres puntos y así este
círculo será
                                        // el que represente la pupila.
        C = Circuncentro( x2, y2 , x1, y1, x3, y3 );
        return (C);                         // Se retornan los valores del centro y radio
de la circunferencia.


}
```

EncontrarIris.C

/*----------

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

Script que contiene la función EncontrarIris(float *Pin, int r), para encontrar la distancia más cercana entre el radio de la pupila y el radio del partiendo del centro de la pupila. Esta función es llamada despues de haberse hallado el centro y radio de la pupila. Valores desde los cuales se parte para hallar la curvatura del iris en la región inferior derecha del ojo.

----------*/

```
#include "math.h"
#include "Etiquetas.h"

//  Función usada para encontrar la semicircunferencia que es usada para hallar
//  la curvatura del Iris.
extern int SemiCirculo(int x0, int y0, int r);

//  El algoritmo parte de un radio inicial basado en el radio de la pupila,
//  luego trazando semicirculos (cuarto cuadrante) cada vez con un radio mayor,
//  se analiza la intensidad de los pixels de la imagen, previamente filtrada
//  con el filtro Gradiente (Realce de Bordes), que coinciden con el
//  semicirculo y se elije el que conincida con el grupo de mayor intensidad.
//  La región de mayor intensidad será aquella donde se encuentra el borde del
//  Iris.
int EncontrarIris(float *Pin, int r){

   //  Definición de Variables.
        int i,k,N;
        int x0,y0,rI;

        int *pX,*pY;
        float *IGrad;
        float Prom,PromM;
```

```
        IGrad  = Pin;                       // Imagen de Entrada (Imagen filtrada
con Realce de Bordes)

        pX = pPixelsX;

        pY = pPixelsY;                      // Vectores con las coordenadas de los
pixels del semicirculo trazado.


        x0 = 121;

        y0 = 121;                           // El centro de la pupila es estático,
gracias a que previamente

                                            // (en funciones anteriores) se centró la pupila en
la posición (121,121).


        r = r + 20;                         // Se parte de un valor de radio Inicial.


        rI = 119;                           // Radio máximo posible.

        PromM = 0;                          // Promedio de los pixels analizados en
cada semicirculo.


        for (i=r;i<=115;i++){               // For para trazar los semicirculos y
hacer el análisis.


                N = SemiCirculo(x0,y0,i);   // Se genera el semicirculo. Los
puntos quedan almacenados en los punteros

                                            // pX,pY.


                Prom = 0;
```

```
            for (k=0;k<N;k++){                  // For para calcular el promedio
de todos los pixels analizados en la región
                                                // del semicirculo actual.
                Prom = Prom + *(IGrad + (*(pX + k)) +((*(pY + k))*241) ) ;
            }
            Prom = Prom / N;                    // Se calcula el valor final.

            if( PromM < Prom){                  // Se almacena el caso que tenga
el mayor valor de promedio.
        rI = i ;
        PromM = Prom;
            }
        }

        return (rI);                    // Se retorna el valor del radio encontrado.

}
```

Estadistica.c

/*----------

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

Script que contiene las funciones para calcular el valor promedio y la desviación estandar de un vector o matriz. Estas funciones son usadas en el sistema de validación.

Se hace uso de dos funciones matemáticas de la librería math.h de C.

pow(A,B) -> Esta función devuelve el valor de A^B. A elevado a B.

sqrt(A) -> Esta función devuelve el valor de la raíz cuadrada de A.

También contiene la función CalcularErrores(), la cual calcula las

distancias Euclidiana y Hamming entre el vector de caracteristicas del usuario a validar y los vectores del usuario registrado.

---*/

```
#include "math.h"
#include "Etiquetas.h"

//  Función promedio, la cual recibe el apuntador del vector a calcular el
//  promedio y el tamaño de este.
float Promedio(float *Pin,int N){

   //  Definiciciòn de variables locales.
        int i;
        float Prom;
        float *V;

        V = Pin;

        Prom = 0;                       //  Valor de promedio. Inicia en 0.
        for(i=0;i<N;i++)                //  Se recorren todos los valores del
              Prom = Prom + *(V + i);   //  vector obteniendo el acumulado total.
```

```
        Prom = Prom/N;                          //  Finalmente el Valor Promedio es igual
                                                //  al Acumulado dividido la cantidad de valores.
        return (Prom);                          //  Se retorna el resultado.
}

/*
  Función Desviación Estandar, la cual recibe el apuntador del vector a calcular,
  y el tamaño de este. La desviacion estandar se calcla siguiendo la siguiente
  ecuación:
          DesviacionEstandar = RaizCuadrada ( Sumatoria( (Xi - X')^2 ) / N )
  Donde:
        X  = Vector de valores.
        Xi = Valor del vector X evaluado en la posición i.
        X' = Valor Promedio de los valores del Vector X.
        N  = Número total de valores en el Vector X.

  Los valores que toma i, son desde 0 hasta N-1.
*/
float Desvest(float *Pin,int N){

    //  Definiciciòn de variables locales.
        int i;
        float Desv,Prom;
        float *V;
```

```
        V = Pin;
        Prom = (V,N);

        Desv = 0;                       //  Valor de promedio. Inicia en 0.
        for(i=0;i<N;i++){               //  For para calcular el valor de la sumatoria
                Desv = Desv + pow(( *(V+i) - Prom ),2);     //  Se acumula el valor de cada término (Xi - X')^2.
        }

        Desv = sqrt(Desv/N);                    //  Finalmente la Desviación Estandar es igual
                                        //  a la raíz cuadrada del Acumulado dividido
                                        //  la cantidad de valores.

        return (Desv);                  //  Se retorna el resultado.
}

//  Función CalcularErrores, HAce uso de los vectores de caracteristicas del
//  usuario ingresado y la Base de Datos del usuario registrado.
void CalcularErrores(){

    // Definición de variables Globales.
        int i,j,k;
```

```
float Desv,Prom;

float *VE,*VEbd,*EE1,*EE2;

float *VH,*VHbd,*EH1,*EH2;

float sum1, sum2;

float sumH1, sumH2;


VE          = pImageWVector;                    // Apuntador
Vector Caracteristicas Euclidiano Usuario a Validar.

VEbd  = pVectorDataBase;                        // Apuntador Vector
Caracteristicas Euclidiano Usuario Base de Datos.

EE1    = pVectorE1;                             // Apuntador Vector Errores
entre vector Usuario a Validar y Base de Datos (Euclidiano).

EE2    = pMatrizError2;                         // Apuntador Vector
Errores entre Vectores Base de Datos (Euclidiano).


VH          = pVectorWBinario;                  // Apuntador
Vector Caracteristicas Hamming Usuario a Validar.

VHbd  = pVectorDataBaseBinario;                 // Apuntador Vector
Caracteristicas Hamming Usuario Base de Datos.

EH1    = pVectorErrorBin1;                      // Apuntador Vector
Errores entre vector Usuario a Validar y Base de Datos (Hamming).

EH2    = pMatrizErrorBin2;                      // Apuntador Vector
Errores Vectores Base de Datos (Hamming).



for(i=0;i<19;i++){
```

```
sum1 = 0;

sumH1 = 0;

                                        //  Cálculo Errores entre vector Usuario a Validar y Base de Datos.

for(j=0;j<128;j++){

        sum1 = sum1 + pow( ( (*(VE + j)) - (*(VEbd + j + (i*128))) ) , 2 ); // Distancia Euclidiana.


        if( (*(VH + j)) != (*(VHbd + j + (i*128))) ) {                  // Distancia Hamming (Comparacion XOR).

                sumH1++;

        }

}

*(EE1 + i) = sqrt(sum1);

*(EH1 + i) = sumH1/128;                                 //  Normalización.



for(k=0;k<19;k++){                                      //  Cálculo Errores entre Vectores Base de Datos.

        sum2 = 0;

        sumH2 = 0;

        for(j=0;j<128;j++){

                sum2 = sum2 + pow( ( (*(VEbd + j + (i*128)) ) - (*(VEbd + j + (k*128))) ) , 2 );   // Distancia Euclidiana.
```

```
                                    if( (*(VHbd + j + (i*128)) ) != (*(VHbd + j + (k*128))) ) {
// Distancia Hamming (Comparacion XOR).
                                        sumH2++;
                                    }
                                }
                                *(EE2 + k + (i*19)) = sqrt(sum2);
                                *(EH2 + k + (i*19)) = sumH2/128;          // Normalización.
                            }
                        }
                    }
```

Etiquetas.h

```
/*-----------------------------------------------------------------------

-----------------------------------------------------------------------

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

-----------------------------------------------------------------------

-----------------------------------------------------------------------

Script que contiene las direcciones de memoria que son reservadas para

el procesamiento de las imagenes en el DSP y sus respectivas etiquetas.

-----------------------------------------------------------------------

-----------------------------------------------------------------------*/


// Direccion Inicio Imagen Original Cargada

#define pImage (unsigned char *)0x00000000


// Direccion Inicio Imagen Original Recortada Float
```

```
#define pImagef (float*)0x0004B004

//Direccion Inicio Imagen Salida 8 Bits
#define pImageS (unsigned char *)0x000D674C

// Direccion Inicio Imagen Salida en Float
#define pImagefS (float*)0x00090BA8

// Direccion Inicio Imagen Salida 2 en Float
#define pImagef1 (float*)0x000E7E38

// Direccion Inicio Imagen Salida 3 en Float
#define pImagef2 (float*)0x0012D9DC

// Direccion Inicio Imagen Salida 3 en Float
#define pImagef3 (float*)0x00173580

// Direccion Inicio Imagen Salida 4 en Float
#define pImagef4 (float*)0x001B9124

// Direccion Inicio Imagen Gaussiano2 en Float
#define pImagef5 (float*)0x001FECC8

// Direccion Inicio Imagen IRIS RECORTADO.
```

```
#define pImagef6 (float*)0x0024486C

// Direccion Inicio Imagen Gaussiano2 en Float
#define pPixelsX (int*)0x0027D3F4

// Direccion Inicio Imagen Gaussiano2 en Float
#define pPixelsY (int*)0x0027DD48

// Direccion Inicio Pixels Envolvente en X
#define pPixEnvolventeX (unsigned short int*)0x0027E69C

// Direccion Inicio Pixels Envolvente en Y
#define pPixEnvolventeY (unsigned short int*)0x00280DB0

// Direccion Inicio Imagen IRIS RECORTADO chaR Salida.
#define pImageSiris (unsigned char *)0x002834C4


// Direccion Inicio Imagen NORMALIZACIÓN-Theta.
#define pImageNTheta (float *)0x002917A8

// Direccion Inicio Imagen NORMALIZACIÓN-Temp.
#define pImageNTemp (float *)0x00291FAC
```

```
// Direccion Inicio Imagen NORMALIZACIÓN-RMat.
#define pImageNRMat (float *)0x002920B8

// Direccion Inicio Imagen NORMALIZACIÓN-x0.
#define pImageNx0 (float *)0x002B20BC

// Direccion Inicio Imagen NORMALIZACIÓN-y0.
#define pImageNy0 (float *)0x002D20C0

// Direccion Inicio Imagen NORMALIZACIÓN-INormalida.
#define pImageNfNorm (float *)0x002F20C4

// Direccion Inicio Imagen WAVELET W1.
#define pImageW1 (float *)0x003120C8

// Direccion Inicio Imagen WAVELET W2.
#define pImageW2                (float *)0x003220CC

// Direccion Inicio Vector WAVELET RESULTADO EUCLIDIANO.
#define pImageWVector           (float *)0x003320D0

// Direccion Inicio Vector WAVELET RESULTADO BINARIO.
#define pVectorWBinario (float *)0x00335124
```

```
// Direccion Inicio Vector BASE DATOS EUCLIDIANO.
#define pVectorDataBase  (float *)0x003322D4

// Direccion Inicio Vector EE1 (Error Vectores Validación Euclidiano).
#define pVectorE1                (float *)0X003348D8

// Direccion Inicio Vector EE2 (Error Vectores Validación Euclidiano).
#define pMatrizError2            (float *)0X00334928

// Direccion Inicio Vector DesvEE2 (Desviaciones Estandar de las Filas de EE2).
#define pVectorDesvError2        (float *)0X00334ED0

// Direccion Inicio Vector BASE DATOS HAMMING.
#define pVectorDataBaseBinario (float *)0x00335328

// Direccion Inicio Vector EH1 (Error Vectores Validación Hamming).
#define pVectorErrorBin1  (float *)0X0033792C

// Direccion Inicio Vector EH2 (Error Vectores Validación Hamming).
#define pMatrizErrorBin2   (float *)0X0033797C

// Direccion Inicio Vector DesvEH2 (Desviaciones Estandar de las Filas de EH2 Hamming).
```

```
#define pVectorDesvErrorBin2 (float *)0X00337F24
```

Generales.C

/*

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

Script que contiene las funciones de recorte usadas en el proyecto.

También la función Mostrar() que permite guardar una imagen con el formato adecuado para ser visualizado por la herramienta ImageViewer.

---

*/

```
#include "math.h"
```

```
#include "Etiquetas.h"

#include <stdio.h>

#include <stdlib.h>

#include <stdint.h>


//      Esta función recibe el apuntador de la imagen a recortar (*Pin) y el apuntador de la imagen

//      recortada (*Pout). El recorte sustrae la región de interes en la imagen, esto para reducir el

//      tiempo de procesamiento del algoritmo, ya que inicialmente, se desea conocer únicamente la

//      posición del iris en la imagen.

void Recortar(unsigned char *Pin,float *Pout){


        // Definición de variables Globales.

        int i,j;

        int Min, Nin, Mout, Nout;


        unsigned char *In;

        float *Out;


        In = Pin;
                //      Apuntador Imagen de Entrada.

        Out = Pout;
                //      Apuntador Imagen de Salida.
```

```
        Min = 480;
            //      Alto y ancho de la imagen de entrada.
        Nin = 640;


        Mout = 210;
            //      Alto y ancho de la imange de salida.
        Nout = 340;


        for(j=149;j<(Min-120);j++)                                          //
        Se recorren las filas y columnas de la imagen,
            for(i=149;i<(Nin-150);i++)                                      //
        copiando los pixels de la imagen de entrada (In)

                //      a la imagen de salida (Out)
                *(Out + (i-149)+((j-149)*Nout) ) = (float)*(In + i+(j*Nin) );


}


//      Esta función recibe el apuntador de la imagen a recortar (*Pin) y el
apuntador de la imagen
//      recortada (*Pout) y las coordenadas del pixel en el cual se centrará la
imagen (x0,y0).
void RecortarIris(unsigned char *Pin,float *Pout,int x0,int y0){


        // Definición de variables Globales.
        int i,j;
```

```
int Min, Nin, Mout, Nout,alto,ancho;


unsigned char *In;

float *Out;


In = Pin;
        //      Apuntador Imagen de Entrada.

Out = Pout;
        //      Apuntador Imagen de Salida.


y0 = y0 + 150;

x0 = x0 + 150;
        //      Offset causado por el primer recorte

                //      hecho en la imagen original


Min = 480;
        //      Alto y ancho de la imagen de entrada.

Nin = 640;


Mout = 241;
        //      Alto y ancho de la imange de salida.

Nout = 241;


alto = 0;

for(j=(y0-120);j<=(y0+120);j++){                                //      Se
recorren las filas y columnas de la imagen,
```

```
            ancho = 0;
            for(i=(x0-120);i<=(x0+120);i++){                                    //
      copiando los pixels de la imagen de entrada (In)

                  //        a la imagen de salida (Out).
                  *(Out + ancho +(alto*Nout) ) = (float)*(In + i+(j*Nin) );
                  ancho++;
            }
      alto++;
      }

}

//    Esta función recibe el apuntador de la imagen de entrada (*Pin) y el
apuntador de la imagen de
//    salida (*Pout), el valor del pixel de menor intensidad (Pmin), y las
dimensiones de la imagen (M,N).
//    Esta función ajusta el rango dinánico de la imagen a valores entre 0 y 255
para su correcta visualización.
//    Por esto es que se recibe el valor mínimo de intensidad de la imagen, para
tomar este valor como el 0.
void Mostrar(float *Pin, unsigned char *Pout,float Pmin, int M, int N){

      // Definición de variables Globales.
      int i,j;
      unsigned char *Out;
```

```
        float *In,Pmax,Aux;


        Pmax = 0;


        In = Pin;
                //      Apuntador Imagen de Entrada.
        Out = Pout;
                //      Apuntador Imagen de Salida.




        for(i=0;i<(M*N);i++){
        //      Se recorre la imagen para hallar el valor del pixel
                *(In + i ) = Pmin + (*(In + i ));                       //      de
mayor intensidad. Y a su vez se suma el valor mínimo
                if(Pmax < *(In + i ) )
                        Pmax = *(In + i );
        }


        for(i=0;i<(M*N);i++){
        //      Se recorre de nuevo la imagen, ahora ajustando los
                Aux = (255.0f/Pmax) * (*(In + i ));                     //
        valores a un rango de [0-255]
                *(Out + i ) = (unsigned char)Aux ;
        }
}
```

GestionDeArchivos.C

/*

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

Script que contiene la función

GuardarVector(int M, int NumUsuario). Función para generar la base de datos con los vectores de características correspondientes al usuario registrado. Esta función se encarga de generar y guardar los archivos en formato de texto para los vectores de caracteristicas cuantificadas y binarizadas anteriormente, en una carpeta del sistema operativo. Estas bases de datos serán usadas porteriormente para las pruebas de validación del sistema.

La función almacena dos bloque de 128 valores (Valores Cuantifiados y Valores Binarios), que equivalen al vector de caracteristicas de una imagen procesada.

Para llevar un orden de las bases de datos generadas, se lleva una nomenclatura para los nombres de los archivos, estas son:

Para Bases de Datos con valores cuantificados:

Nombre -> [NombreUsuario]LCuant.txt

Para Bases de Datos con valores binarios:

Nombre -> [NombreUsuario]LBin.txt

NombreUsuario corresponde al valor string del usuario a registrar, es decir, para el registro del usuario 1, los nombres de los archivos serán:

Bases de Datos con valores cuantificados --> 1LCuant.txt

Para Bases de Datos con valores binarios --> 1LBin.txt

```
-----------------------------------------------------------------------

*/

#include "math.h"
#include <stdio.h>
#include "Etiquetas.h"
#include <string.h>

//  Esta función recibe un número de usuario para etiquetar el archivo a generar
//  También recibe un índice M, para conocer la cantidad de datos actuales en
//  el archivo y evitar la sobreescritura de estos.
void GuardarVector(int M, int NumUsuario){

        int i,j,N,O;
        int x,y;
        float *Vcuant,*Vmuest,*Vbin;
        char cCuant[128];
        char cBin[128];

   char path1[100] = "C:/";                    //  Path con el directorio destino, para el archivo características (Valores Cuantificados).
        char path2[100] = "C:/";                    //  Path con el directorio destino, para el archivo características (Valores Binarios).
```

```
FILE *fileDat, *fileCuant, *fileBin;                    // Objetos File, para la generación y guardado del archivo.

Vcuant          = pImageWVector;                        // Puntero de vector con los valores cuantificados.

Vbin    = pVectorWBinario;                              // Puntero de vector con los valores binarios.

char strNumUsuario[2];

snprintf(strNumUsuario, 3, "%d", NumUsuario);           // Cast a string del numbero del usuario.

strcat (path1,strNumUsuario);

strcat (path1,"LCuant.txt");                            // Se concatena al número del usuario el string "LCuant.txt".

strcat (path2,strNumUsuario);

strcat (path2,"LBin.txt");                              // Se concatena al número del usuario el string "LBin.txt".

for (i=0;i<128;i++){                                    // Los datos son almacenados inicialmente en un vector de carácteres.
        j = ( (int)(*(Vcuant+i)) ) + 48;

        cCuant[i] = (char)j;                            // cCuant, vector de valores cuantificados.

        j = ( (int)(*(Vbin+i)) ) + 48;
```

```
            cBin[i] = (char)j;                          //  cBin, vector de valores
binarios.
        }

        if(M == 0){                                     //  Si el índice M es 0, se inícia a
partir de un documento en blanco.

            fileCuant = fopen(path1, "w");              //  Se abre el nuevo
archivo ("w").
            fseek(fileCuant, M, SEEK_SET);
            N = fwrite(cCuant , sizeof(char), 128, fileCuant);     //  Se escriben los
datos, uno tras otro.
            fclose(fileCuant);                          //  Cierre del Archivo.

            fileBin = fopen(path2, "w");                //  Se abre el nuevo archivo
("w").
            fseek(fileBin, M, SEEK_SET);
            O = fwrite(cBin , sizeof(char), 128, fileBin);          //  Se escriben los
datos, uno tras otro.
            fclose(fileBin);                            //  Cierre del Archivo.

        }else{                                          //  Sino se abre el archivo existente y
se concatenan los nuevos 128 datos.

            fileCuant = fopen(path1, "a");              //  Se abre el archivo
existente ("a").
            fseek(fileCuant, M, SEEK_SET);
```

```
            N = fwrite(cCuant , sizeof(char), 128, fileCuant);     //  Se escriben los datos, uno tras otro.

            fclose(fileCuant);                                     //  Cierre del Archivo.


            fileBin = fopen(path2, "a");                           //  Se abre el archivo existente ("a").

            fseek(fileBin, M, SEEK_SET);

            O = fwrite(cBin , sizeof(char), 128, fileBin);         //  Se escriben los datos, uno tras otro.

            fclose(fileBin);                                       //  Cierre del Archivo.

        }

}


void CargarVector(){


        int i,j,N,M;

        int x,y;

        float *V;

        char c[128];


        V = pImageWVector;



        FILE *file = fopen("D:/JuanC/Dropbox/Proyect_Completo/20-10-2013_Gauss+EncPupilaIris(Grad)+Normalización - copia/Vector.txt", "r");

        N = fread(c,sizeof(char),128,file);
```

```
        printf("N = %d", N);
        fclose(file);

        for(i=0;i<128;i++)
                *(V+i) = 0;

        M = 0;
        for (i=0;i<128;i++){
                //*(V+i) = ( (int)(*(V+i)) ) + 48;
                printf("%c \n", c[i]);
        }

}
```

Gradiente.c

/*

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

Script que contiene la función

Gradiente(float *Pin,float *Pout, int M, int N). Función que aplica un filtro gradiente, es decir, la imagen de salida (Pout), corresponde a la magnitud del operador Sobel de la imagen de entrada (Pin). Este tipo de operadores es usado para la detección de bordes.

Para encontrar la magnitud del gradiente para cada punto está definido como:

G = RaizCuadrada (Gx^2 + Gy^2)

Donde, Gx y Gy, representan para cada punto las aproximaciones horizontal y vertical de las derivadas de intensidades, y estos valores son calculados aplicando convolución con la máscara correspondiente a derivada, para la derivada horizontal se tiene una máscara:

Mascara Gx =

-1 0 1

-2 0 2

-1 0 1

Mascara Gy =

-1 -2 -1

0 0 0

1 2 1

---

*/

```
#include "math.h"
#include "Etiquetas.h"
```

```
//  El filtro recorre cada uno de los pixels en la imagen con frontera cerrada,
//  es decir, los pixels correspondientes a las 2 filas y columnas exteriores
//  de la imagen, no serán filtrados.
float Gradiente(float *Pin,float *Pout, int M, int N)
{
   // Definición de variables Globales.
        float *Gmag,*In,*Thetam,*EdgeNoise,*Edge;
        float Min,Max,High,Low;
        float AcumX,AcumY;
        float Mx[9] = {-1,0,1,-2,0,2,-1,0,1};          //  Vector Máscara
para derivada Horizontal (Gx).
        float My[9] = {-1,-2,-1,0,0,0,1,2,1};          //  Vector Máscara
para derivada Vertical   (Gy).
        float Grad,Res;
        float Pi;

        int i,j,k,Ng,Mg,Nn,Mn,Ne,Me;
        int iF,iG,jF,jG;
        int x,y,alto,ancho;
        int Theta;

        Min = 0;
        Max = 0;
```

```
In              = Pin;
Gmag            = Pout;

for(i=0;i<(M*N);i++)                    //  Inicialmente, se copia la imagen de entrada a la de salida.
    *(Gmag + i) = 0;                    //  Esto para no alterar los valores de los pixels en las 2 filas
                                        //  y columnas de los bordes de la imagen.

 for(j=1;j<M-1;j++){                    //  Recorrido de la matriz con frontera cerrada.
     for(i=1;i<N-1;i++){
         AcumX = 0;                     //  Valor del Pixel derivada Horizontal.
         AcumY = 0;                     //  Valor del Pixel derivada Vertical.
         k = 0;

         ///////      Filtro Gradiente      /////////////

         for(y=(j-1);y < (j + 2);y++)   //  Se recorren los 9 pixels de la máscara y para cada uno
             for(x=(i-1);x < (i + 2);x++){
                 AcumX = AcumX + ( *(In + x+(y*N) ) ) * Mx[k];  //  se realiza el cálculo y se añade a la sumatoria total.
                 AcumY = AcumY + ( *(In + x+(y*N) ) ) * My[k];
```

```
                    k++;
                }

                ////////////////////////////////////////////////////////

                Grad = sqrt( pow(AcumX,2) + pow(AcumY,2) );          //
Cálculo de la Mágnitud del Gradiente.

                *(Gmag + i+(j*N) ) = Grad;                    //  Se almacena el
valor final del pixel en el vector de salida.

                if(Min > Grad)                                //  Se valida si el nuevo
valor es menor al menor actual.
                    Min = Grad;                               //  Se guarda el
valor de este pixel.
                }
            }

            return Min;
}
```

Initialization.c

/*

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

Inicialización de los módulos del DSP. para la inicialización del DSP se cuenta con básicamente básicamente dos. Init_PLL(void), la cual con Figura la velocidad de reloj del núcleo y vector de interrupción.

La función Init_SDRAM(void), se encarga de inicializar el módulo SDRAM en el cual se encuentrna los bancos de memoria externa que se usarán para almacenar las imagenes y vectores usados. Las posiciones de memoria reservados para la SDRAM son establecidos y etiquetados en el archivo Etiquetas.h

```
---------------------------------------------------------------------

---------------------------------------------------------------------

                                                                */

#define _USE_LEGACY_CDEF_BEHAVIOUR

#include <cdefBF533.h>
#include <ccblkfn.h>
#include <sysreg.h>
#include "math.h"
#include "Etiquetas.h"
#include "Slave_Global.h"

volatile DMA_Descriptor_Short DMA_1,DMA_2;
int *start_address;

void Init_PLL(void)
{
        int a;

        sysreg_write(reg_SYSCFG, 0x32);                         // Registro
para la con  Figura ción de la inicialización del sistema
```

```
        a = *pIMASK;


        *pIMASK = 0x0;
        *pSIC_IWR = 0x1;


        *pPLL_CTL = 0x2000;
        ssync();
        idle();
        *pIMASK = a;
}




void Init_SDRAM(void)
{
                                        //Registro de Control de frecuencia de
Actualización.
        if (*pEBIU_SDSTAT & SDRS)
        {


                *pEBIU_SDRRC = 0x00000817;          //Registro de
Control de bancos de Memoria.


                *pEBIU_SDBCTL = 0x00000013;          //Registro de
Control de la Memoria Global.
```

```
                *pEBIU_SDGCTL = 0x0091998d;                    //Registro
de Control de la Memoria Global.


                ssync();

        }

}
```

Normalizar.c

/*

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

Normalizar(int Ms,int Ns, int RP, int RI), función encargada de la normalización del iris, partiendo de la región comprendida entre la Pupila y el Iris.

Ms es el ancho de la imagen normalizada deseada

Ns es el alto de la imagen normalizada deseada

RP es el radio de la Pupila (Radio Inferior)

RI es el radio del Iris (Radio Exterior)

El algoritmo consta de dos partes. La inicial, es encargada de generar los puntos en coordenadas polares que representan la región del Iris según los valores de alto (Ns) y ancho (Ms) elegidos para la normalización.

Entonces, la componente angular estará relacionada con el número de filas de la imagen normalizada (Ns), entonces, este angulo variará de 0 a 2*pi e irá en pasos de (2*3.14159)/Ns. Estos valores serán almacenados en el apuntador (Theta).

La componente radial estará relacionada con el número de columnas de la imagen normalizada (Ms), entonces, este radio variará de RP a RI e irá en pasos de 1/RI. Estos valores serán almacenados en el apuntador (Theta).

Luego para cada uno de estos se calcula su valor correspondiente en coordenadas cartesianas, realizando una conversión sencilla de coordenadas usando las funciones cos() y sin() de la librería math.h de C. Entocnes, se obtienen dos matrices con los valores X y Y de los puntos finales de la imagen normalizada. Pero estos valores son números decimales, los cuales no pueden representar una matriz.

La segunda parte del algoritmo se encarga entonces de interpolar estos valores a partir de la imagen original, realizando una interpolación

bilineal de la matriz imagen de entrada (puntero I), para los puntos hallados en el paso anterior.

---

*/

```
#include "math.h"
#include "Etiquetas.h"

void Normalizar(int Ms,int Ns, int RP, int RI){

  //  Definición de Variables Locales.
      int i,j,k,x,y;
      int AngDiv, AngularD, RadPix, RadiusP;

      float *I,*INorm;
      float *Theta,*Temp,*Rmat,*x0,*y0;

      float N;
      float X,Y;
      float X1,Y1,X2,Y2,Q11,Q12,Q21,Q22;
      float Pmax,Aux;
```

```
        unsigned char *IS;


        I               = pImagef6;                     //  Apuntador Imagen de
entrada.
        INorm           = pImageNfNorm;                 //  Apuntador Imagen de
salida Normalizada.
        Theta = pImageNTheta;                   //  Vector Valores de Theta
Coordenadas Polares.
        Temp = pImageNTemp;                     //  Vector Valores de Radio
Coordenadas Polares.
        Rmat  = pImageNRMat;                    //  Matriz Valores de Radio en
Coordenadas Polares.
        x0              = pImageNx0;                    //  Valores X en coordenadas
Cartesianas.
        y0              = pImageNy0;                    //  Valores Y en coordenadas
Cartesianas.


        IS = pImageS;


        x = 120;        y = 120;                //  Punto central de la Pupila.


        AngDiv = Ns;    AngularD  = AngDiv - 1;         //  Variables con el número de
columnas ingresadas
        RadPix = Ms;    RadiusP  = RadPix + 2;          //  Variables con el número
de filas ingresadas


        *Theta = 0;
```

```
N = (2*3.14159)/AngularD;                    //  Cálculo del tamaño de paso para Theta.

for(i=1;i<AngDiv;i++)                        //  A partir de este, se construye el vector Theta.
        *(Theta + i) = *(Theta + (i-1)) + N;

*Temp = 0;

N = 1/( RadiusP - 1 );                       //  Cálculo del tamaño de paso para Temp, que corresponde a los valores radiales.

for(i=1;i<RadiusP;i++)                       //  A partir de este, se construye el vector Temp (Radio).
        *(Temp + i) = *(Theta + (i-1)) + N;

for(i=1;i<(RadiusP-1);i++)                   //  Ahora se construye la Matriz de Valores de Radio en Coordenadas Polares.
        for(k=0;k<AngDiv;k++)
                * (Rmat + k + ((i-1)*Ns) ) = ( (RI - RP) * (*(Temp + i) ) ) + RP;

for(i=0;i<AngDiv;i++)
        for(j=0;j<RadPix;j++){               //  Luego esta matriz es convertida a coordenadas cartesianas.
```

```
                                        //  Cálculo de las coordenadas en X.
                    *(x0 + i + (j*Ns) ) = ( (* (Rmat + i + (j*Ns) )) * cos(*(Theta + i)) )
+ x;
                                        //  Cálculo de las coordenadas en Y.
                    *(y0 + i + (j*Ns) ) = y - ( (* (Rmat + i + (j*Ns) )) * sin(*(Theta +
i)) );


                }


   ///////////////   Algoritmo de Interpolación Bilineal
//////////////////////////////////////////////////////


   //  A partir de la matriz de valores en coordenadas cartesianas, se interpolan
   //  Estos valores a partir de los valores de la imagen de entrada.

        for(i=0;i<RadPix;i++){
                for(j=0;j<AngDiv;j++){


                X = *(x0 + j + (i*Ns) );          //  Valor Coordenada en X imagen
interpolada.
                Y = *(y0 + j + (i*Ns) );          //  Valor Coordenada en Y imagen
interpolada.


                //  Valores los llevados a valores enteros, para ser evaluados en la
      //  imagen de entrada.
```

```
                X1 = floor(X);
                Y1 = floor(Y);

                X2 = X1 + 1;
                Y2 = Y1 + 1;

                //  Luego se extraen los valores de intensidad de la imagen de
entrada
                //  evaluados en los puntos anteriores.
                Q11 = *(I + (int)(X1 + (Y1*241)) );
                Q12 = *(I + (int)(X1 + (Y2*241)) );
                Q21 = *(I + (int)(X2 + (Y1*241)) );
                Q22 = *(I + (int)(X2 + (Y2*241)) );

                //  Se calcula el valor que corresponde al punto interpolado entre
        //  los cuatro puntos de la imagen de entrada.
                *(INorm + j + (i*Ns) ) = ((1/((X2-X1)*(Y2-Y1)))*(Q11*(X2-X)*(Y2-Y) +
Q21*(X-X1)*(Y2-Y) + Q12*(X2-X)*(Y-Y1) + Q22*(X-X1)*(Y-Y1)));
                }
          }

    //  Ahora, la imagen de salida es recortada, tomando los cuadrantes 3 y 4
    //  del Iris, esto para eliminar la región superior del Iris, el cual
    //  a menudo es obtruido por factores como las pestañas.
          for(j=0;j<Ms;j++){
```

```
                for(i=0;i<Ns/2;i++){
                        *(I + i + (j*Ns/2)) = *(INorm + (i+(Ns/2)) + (j*Ns));
                }
        }
}
```

PillBox.C

/*----------

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

Script que contiene la función

PillBox(float *Pin,float *Pout,int M,int N). Función que aplica un filtro espacial a la imagen de entrada (Pin) y el resultado es almacenado en el apuntador (Pout). Este es un filtro de promedio circular, el cual realza las Figura s con contornos circulares en la imagen, acentuando las regiones que limitan la Pupila y el Iris, y el Iris con la Esclerótica del ojo.

La matriz máscara usada de dimensiones 5x5, es la siguiente:

0 0.0170159174816308 0.0381149714439322
0.0170159174816308 0

0.0170159174816308 0.0783813541603717 0.0795774715459477
0.0783813541603717 0.0170159174816308

0.0381149714439322 0.0795774715459477 0.0795774715459477
0.0795774715459477 0.0381149714439322

0.0170159174816308 0.0783813541603717 0.0795774715459477
0.0783813541603717 0.0170159174816308

0 0.0170159174816308 0.0381149714439322
0.0170159174816308 0

---*/

```
#include "math.h"
#include "Etiquetas.h"

//  El filtro recorre cada uno de los pixels en la imagen con frontera cerrada,
//  es decir, los pixels correspondientes a las 2 filas y columnas exteriores
//  de la imagen, no serán filtrados.
float *PillBox(float *Pin,float *Pout,int M,int N){

   //  Definición de las Variables.
        int i,j,k,x,y;
        float *In, *Out;
        float *Min;
```

```
Min = (float*)malloc ( 4*sizeof(float) );    // Min[0] -> Coordenada x de Pixel con Valor Minimo
                                             // Min[1] -> Coordenada y de Pixel con Valor Minimo
                                             // Min[2] -> Valor Pixel Minimo
                                             // Min[1] -> Valor Pixel Maximo

In = Pin;
Out = Pout;

int Ns;

//  Definición de la máscara.
float Mk[25] = { 0,0.0170159174816308,0.0381149714439322,0.0170159174816308,0,

0.0170159174816308,0.0783813541603717,0.0795774715459477,0.0783813541603717,0.0170159174816308,

0.0381149714439322,0.0795774715459477,0.0795774715459477,0.0795774715459477,0.0381149714439322,

0.0170159174816308,0.0783813541603717,0.0795774715459477,0.0783813541603717,0.0170159174816308,

0,0.0170159174816308,0.0381149714439322,0.0170159174816308,0
```

```
        };
    float Acum;


    Ns = N - 4;
    Min[2] = 255;
    Min[3] = 0;


    for(i=0;i<(M*N);i++)                    //  Inicialmente, se copia la imagen de entrada a la de salida.
        *(Out + i) = *(In + i);             //  Esto para no alterar los valores de los pixels en las 2 filas
                                            //  y columnas de los bordes de la imagen.


    for(j=2;j<M-2;j++)
        for(i=2;i<N-2;i++){                 //  Recorrido de la matriz con frontera cerrada.
            k = 0;
            Acum = 0;                       //  Valor del Pixel.
            for(y=(j-2);y < (j + 3);y++)
                for(x=(i-2);x < (i + 3);x++){       //  Se recorren los 25 pixels de la máscara y para cada uno
                    Acum = Acum + ( *(In + x+(y*N) ) ) * Mk[k];   //  se realiza el cálculo y se añade a la sumatoria total.
                    k++;
                }
```

```
                    *(Out + i+(j*N) ) = Acum;                    //  Se almacena el
valor final del pixel en el vector de salida.


                    if(Min[2] > Acum ){                    //  Se valida si el
nuevo valor es menor al menor actual.

                        if( (i>=4) && (i<=(N-5)) ){                //  No se toman
en cuenta los pixels en la frontera

                            if( (j>=4) && (j<=(M-5)) ){                //  (las 2 filas
y columnas exteriores de la imagen).

                                Min[2] = Acum;                    //  Se
guarda el valor y las coordenadas de este pixel.

                                Min[0]= i;

                                Min[1]= j;

                            }

                        }

                    }


                    if(Min[3] < Acum )                    //  Se valida si el
nuevo valor es mayor al mayor actual.

                        Min[3] = Acum;                    //  Se guarda el
valor y las coordenadas de este pixel.


                }

        return (Min);                    //  Se retorna el resultado.

}
```

Slave_Global.h

```
void Init_SDRAM(void);

void Init_DMA_OUT(void);

void Setup_DMA_IN(void);

void Init_PLL(void);

float Gradiente(float *Pin,float *Pout, int M, int N);

float *PillBox(float *Pin,float *Pout,int M,int N);

int Circulo(int x0, int y0, int r);

int EncontrarIris(float *Pin, int r);

int *EncontrarCentroPupila(float x0, float y0, float Max);

int *Circuncentro(float x1, float y1, float x2, float y2, float x3, float y3);


void Recortar(unsigned char *Pin,float *Pout);

void Mostrar(float *Pin, unsigned char *Pout,float Pmin, int M, int N);

void WaveletHaar2D(int L,int M,int N);

void GuardarVector(int M, int NumUsuario);

void CargarVector(void);

void Caracterizacion(void);

void carga( void );


unsigned int OutputLine(volatile unsigned int *p, unsigned int line);

unsigned char * Insert_Line(volatile unsigned char *p_out , unsigned int line);
```

WaveletHaar2D.C

/*

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERIAS

INGENIERIA ELECTRONICA

PROYECTO DE GRADO FINAL

SISTEMA DE RECONOCIMIENTO DE IRIS IMPLEMENTADO EN UN DSP

ARLES FELIPE GARCIA MAYA

JUAN CAMILO MORENO RUIZ

2014

WaveletHaar2D(int L,int M,int N), función encargada de realizar a la imagen normalizada del Iris, la transformada de Wavelet de Haar de 4º nivel. Para un tamaño de imagen de entrada de 128x64 pixels, la tranformada de 4º nivel da como resultado una imagen de 16x8 pixels la cual da como resultado un vector de 128 valores, los cuales representan las caracteristicas de la imagen.

Esta función también se encarga de, a partir de este vector de 128 valores, cuantificar estos de dos maneras:

Cuantificación para Análisis Euclidiano :

Esta cuantificación, lleva cada uno de los valores del vector resultado de la transformación a valores de 0, 1, 2, 3, 4, 5:

Si el valor se encuentra entre -500 y 0

Valor = 0

Si el valor se encuentra entre 0 y 500

Valor = 1

Si el valor se encuentra entre 500 y 1000

Valor = 2

Si el valor se encuentra entre 1000 y 1500

Valor = 3

Si el valor se encuentra entre 1500 y 2000

Valor = 4

Sino

Valor = 5

Cuantificación para Análisis Hamming :

Binarizacion de los vectores resultantes, llevando los valores negativos del vector a 0 y los valores iguales y positivos a 1.

---

*/

```
#include "math.h"
#include "Etiquetas.h"

//  La función recibe el número de niveles para la transformada y el ancho
//  y alto de la imagen.
void WaveletHaar2D(int L,int M,int N){

   //  Definición de variables globales
        int i,j,k,Ms,Ns,S1,S2;
        int Div, AngularD, RadPix, RadiusP;

        unsigned char *IS;

        float *W1,*W2,*INorm,*A;
        float *Vcuant,*Vmuest,*Vbin;
        float valorMuestra,valorCuant,valorBin;
```

```
float R;

W1          = pImageW1;                 // Apuntador imagen parcial transformada vertical.
W2          = pImageW2;                 // Apuntador imagen parcial transformada horizontal.
A           = pImagef6;                 // Imagen de entrada.

Vcuant      = pImageWVector;            // Apuntador al vector de valores resultado cuantificado.
Vbin    = pVectorWBinario;              // Apuntador al vector de valores resultado binarizado.

Ms = M;
Ns = N;                                 // Alto y ancho de imagen de entrada.

S1 = M;
S2 = N;                                 // Valores ancho y alto de las regiones a tranformar.

Div = 1;
R = sqrt(2);

for(k=0;k<L;k++){                       // La transformada se realiza L veces. (Niveles de la transformada).
```

```
            if(k>0){                                    // Para la primer iteración no
se cumple la condición.
                for(i=0;i<(M*N);i++)                    // Se actualiza la
imagen de entrada, copiando el resultado de la iteración
                    *(A + i) = *(W2 + i);               // anterior (W2) en
la imagen (A).
            Div=2;                                      // Ahora, se seguirá dividiendo en
2 la región a transformar.
            }

            S1 = S1/Div;
            S2 = S2/Div;                                // Se aplica la división.

            for(i=0;i<((S1/2));i++){                    // Se aplica la
transformada vertical.
                for(j=0;j<(S2);j++){

                    *(W1 + j + (i*N)) = ( ( *(A + j + ((i*2)*N)) ) + ( *(A + j +
(((i*2)+1)*N)) ) )/R;

                    *(W1 + j + ((i+(S1/2))*N)) = ( ( *(A + j + ((i*2)*N)) ) - ( *(A
+ j + (((i*2)+1)*N)) ) )/R;
                }
            }

            for(i=0;i<(S1);i++){                        // Se aplica la transformada
horizontal.
```

```
                    for(j=0;j<(S2/2);j++){

                        *(W2 + j + (i*N)) = ( ( *(W1 + (j*2) + (i*N)) ) + ( *(W1 +
((j*2)+1) + (i*N)) ) )/R;

                        *(W2 + (j+(S2/2)) + (i*N)) = ( ( *(W1 + (j*2) + (i*N)) ) - (
*(W1 + ((j*2)+1) + (i*N)) ) )/R;
                    }
                }
            }

            /*
```

Para este punto, la transformada está completa. Los valores de esta, se encuentran en la matriz

comprendida por las primeras 18 filas y 8 columnas de la imagen resultante (W2).

El siguiente segmento del algoritmo se encarga de guardar en dos vectores (Vcuant y Vbin)de

128 cada uno (16x8) los valores equivalentes para; Cuantificación para Análisis Euclidiano

y Cuantificación para Análisis Hamming.

```
            */
                k=0;
                for(j=0;j<8;j++){
                    for(i=0;i<16;i++){
```

```
valorCuant =  *(W2 + i + (j*N)) ;                    // Se extrae el
valor del pixel.

valorBin =  *(W2 + i + (j*N)) ;


                                                    // Cuantificación para Análsis
Euclidiano.

if( (valorCuant >= -500) && (valorCuant < 0) ){
        valorCuant = 0;
}else if( (valorCuant >= 0) && (valorCuant < 500) ){
        valorCuant = 1;
}else if( (valorCuant >= 500) && (valorCuant < 1000) ){
        valorCuant = 2;
}else if( (valorCuant >= 1000) && (valorCuant < 1500) ){
        valorCuant = 3;
}else if( (valorCuant >= 1500) && (valorCuant < 2000) ){
        valorCuant = 4;
}else{
        valorCuant = 5;
}


                                        // Cuantificación para Análsis Hamming.
if( valorBin < 0 ){
        valorBin = 0;
}else {
        valorBin = 1;
```

```
                    }


                    *(Vcuant + k) = valorCuant;                    // Se almacenan
los valores.
                    *(Vbin + k) = valorBin;
                    k++;
                }
            }
}
```