UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

Detección de gasas quirúrgicas en imágenes laparoscópicas empleando la técnica LBP

Autor:

Muñoz García, Álvaro

Tutor:

De la Fuente López, Eusebio
Departamento de Ingeniería de
Sistemas y Automática

Valladolid, Mayo de 2018.

Resumen

Los sistemas automáticos permiten mejorar las capacidades de los cirujanos, especialmente en operaciones de cirugía mínimamente invasiva, las cuales se llevan a cabo empleando endoscopios que restringen en gran medida el campo de visión del cirujano.

El olvido de material quirúrgico dentro del cuerpo del paciente es una situación poco frecuente pero que tiene un fuerte impacto en su salud. En el presente Trabajo Fin de Grado se propone un sistema de detección de gasas utilizando visión artificial. Para ello, se analizarán las imágenes laparoscópicas capturadas por una cámara endoscópica.

Las imágenes han sido analizadas utilizando el operador *Local Binary Pattern* (LBP) y la varianza. El operador LBP es un descriptor de textura muy potente con un coste computacional muy bajo, lo que ha permitido que la aplicación sea apta para su uso en tiempo real sin utilizar hardware especializado.

Summary

Automatic systems enable surgeons to improve their abilities, especially in minimally invasive surgery. This kind of surgery is performed using endoscopes, which widely restrict the surgeon's field of vision.

Retained surgical items inside the patient's body it is an uncommon event, but it has a strong impact on their health. The aim of this dissertation is to develop a gauze detection system using computer vision. For that, laparoscopic images captured by an endoscopic camera will be analysed.

Those images have been analysed using the Local Binary Pattern (LBP) operator, along with the variance. The LBP operator is a theoretically simple yet very powerful method of analysing textures, with very low computational complexity. Due to this, the system is real-time capable without using any specialized hardware.

Palabras clave

Gossypiboma; Cirugía mínimamente invasiva; Análisis de textura en imágenes; OpenCV; Patrones Binarios Locales.

Keywords

Gossypiboma; Minimally invasive surgery; Image texture analysis; OpenCV; Local Binary Patterns.

Agradecimientos

A mis padres, mi hermana y mi tutor Eusebio, por su ayuda y apoyo indondicional.

Índice general

Índice general VII

Índice de figuras XI

Índice de tablas XV

1. Introducción y objetivos 1

1.1. Cirugía mínimamente invasiva . . . 1

1.1.1. Introducción . . . 1

1.1.2. Clasificación . . . 2

1.1.3. Ventajas e inconvenientes . . . 3

1.1.4. Situación actual . . . 3

1.2. Cirugía robótica . . . 4

1.3. Material quirúrgico retenido . . . 7

1.4. Objetivos . . . 10

2. Imagen digital y visión artificial 11

2.1. Imagen digital . . . 11

2.1.1. Representación . . . 11

2.1.2. Tipos de imágenes digitales . . . 13

2.1.2.1. Imágenes ráster . . . 13

2.1.2.2. Imágenes vectoriales . . . 13

2.1.3. Otras características . . . 14

2.1.3.1. Resolución . . . 14

2.1.3.2. Profundidad de color . . . 14

2.2. Visión artificial . . . 15

2.2.1. OpenCV . . . 16

2.2.2. Aplicaciones de la visión artificial . . . 16

2.3. Conclusiones . . . 17

3. Análisis de texturas 19

3.1. Texturas y tipos de descriptores . . . 19

3.1.1. Tipos de textura . . . 19

3.1.2. Características de los descriptores de textura 21
3.1.3. Tipos de descriptores de textura 22
3.2. Local Binary Pattern (LBP) 23
3.2.1. Operador LBP básico 23
3.2.2. Operador LBP genérico 25
3.2.3. Invarianza rotacional y patrones uniformes 26
3.2.4. Textura: contraste y patrones 35
3.2.5. LBP multi-resolución 35
3.2.6. LBP de colores opuestos 37
3.2.7. Operador mLBP 38
3.2.8. Aplicaciones del operador LBP 38

4. Desarrollo del sistema de detección de gasas quirúrgicas 41
4.1. Introducción 41
4.1.1. Variantes del sistema de detección de gasas quirúrgicas . 43
4.1.2. Jerarquía de archivos 44
4.2. Generador de *look-up tables* 45
4.3. Generador de patrón 47
4.3.1. "LBP" y "LBP y varianza por separado" 48
4.3.2. "LBP y varianza combinado, método 1" 56
4.3.3. "LBP y varianza combinado, método 2" 58
4.4. LBP. Programa principal 61
4.4.1. Variante "LBP" 61
4.4.2. Variante "LBP y varianza por separado" 67
4.4.3. Variante "LBP y varianza combinado, método 1" 68
4.4.4. Variante "LBP y varianza combinado, método 2" 69

5. Resultados experimentales 73
5.1. Método de análisis 74
5.2. Influencia del operador 76
5.3. Influencia del umbral 77
5.4. Influencia del tamaño de tesela 79
5.5. Influencia del vecindario 81
5.6. Análisis de las variantes 2, 3 y 4 84
5.6.1. Variante 2. "LBP y varianza por separado" 84
5.6.2. Variante 3. "LBP y varianza combinado, método 1" . . . 89
5.6.3. Variante 4. "LBP y varianza combinado, método 2" . . . 90
5.7. Comparación de resultados 92
5.8. Análisis de tiempos 95
5.9. Conclusiones 96

6. Conclusiones y líneas futuras **97**
6.1. Conclusiones 97
6.2. Líneas futuras 98

Bibliografía **101**

A. Código fuente: LBP **103**
A.1. *main.cpp* ("LBP") 103
A.2. *main.cpp* ("LBP y varianza por separado") 106
A.3. *main.cpp* ("LBP y varianza combinado, método 1") 109
A.4. *main.cpp* ("LBP y varianza combinado, método 2") 112
A.5. *compara.h* y *compara.cpp* 115
A.6. *visualiza.h* y *visualiza.cpp* 120

B. Código fuente: generador de patrón **123**
B.1. *main.cpp* ("LBP" y "LBP y varianza por separado") 123
B.2. *main.cpp* ("LBP y varianza combinado, método 1") 129
B.3. *main.cpp* ("LBP y varianza combinado, método 2") 134
B.4. *datosPatron.xml* ("LBP" y "LBP y varianza por separado") . . . 137
B.5. *datosPatron.xml* ("LBP y varianza combinado, método 1") . . . 138

C. Código fuente: archivos comunes **139**
C.1. *estadist.h* y *estadist.cpp* 139
C.2. *fichero.h* y *fichero.cpp* 145
C.3. *histo.h* y *histo.cpp* 150
C.4. *lbp.h* y *lbp.cpp* 154

D. Código fuente: generador de LUTs **159**
D.1. *main.cpp* 159
D.2. Ejemplo LUT: *ULBP_LUT_8.dat* 162

Índice de figuras

1.1. Comparativa de la colecistectomía mediante cirugía convencional y cirugía mínimamente invasiva. 2
1.2. Puma 560. Primer robot utilizado en cirugía robótica. 4
1.3. Robot AESOP. 5
1.4. Sistema ZEUS 6
1.5. Sistema quirúrgico da Vinci 7

2.1. Dos métodos para obtener imágenes a color 12
2.2. Canales de una imagen a color 12
2.3. Valores de cada uno de los canales de una imagen a color 13
2.4. Diferencias entre una imagen vectorial y ráster 14
2.5. Logo representado con diferentes resoluciones 14
2.6. Imagen representada con diferentes profundidades de color . . . 15
2.7. Modelo CAD de una dentadura 16

3.1. Comparación macrotextura y microtextura 20
3.2. Ejemplos de texturas con diferente granularidad 20
3.3. Comparación de la apariencia de la gasa y su granularidad . . . 21
3.4. Cálculo del valor LBP en su versión original 23
3.5. Ejemplo de imagen LBP 24
3.6. Histograma de la imagen LBP 25
3.7. Ejemplo de diferentes vecindarios utilizando el operador $LBP_{P,R}$. Los valores (P, R) son, de izquierda a derecha, $(8, 1)$, $(16, 2)$ y $(8, 2)$ 26
3.8. Efecto de una imagen rotada en los puntos de un vecindario . . 27
3.9. Comparación de histogramas de dos imágenes LBP rotadas . . . 28
3.10. Los 58 diferentes patrones uniformes en un vecindario (8,R) . . 30
3.11. Diferentes primitivas de textura detectadas por el operador LBP 31
3.12. Comparación de histogramas de dos imágenes $LBP_{8,1}^{riu2}$ rotadas . 33
3.13. Histograma de la imagen ULBP antes y después de equiespaciar sus niveles de gris con Matlab 34

3.14. Tres vecindarios $LBP_{4,R}$ y una combinación imposible de códigos LBP 36
3.15. Ejemplo OCLBP tomando como píxel central el del canal rojo . 37

4.1. Simplificación de los pasos dados por el sistema de detección de gasas quirúrgicas 42
4.2. Ejemplo ejecución del programa básico 42
4.3. Jerarquía de archivos 44
4.4. Representación del vector que contiene la LUT para $P = 8$ . . . 45
4.5. Ejecución del programa generador de LUTs 46
4.6. Gasas patrón sin sangre 49
4.7. Gasas patrón con sangre 50
4.8. Imágenes patrón antes y después de aplicar un filtro gaussiano . 50
4.9. Histograma de la imagen de varianzas del patrón "patron1.png" 51
4.10. Imagen mLBP y ULBP de la primera imagen patrón para un vecindario $P = 8/R = 1$ 52
4.11. Histograma de la imagen ULBP del primer patrón 53
4.12. Suma de los histogramas la varianza y ULBP de todas las imágenes patrón 54
4.13. Histograma cuantificado de la varianza de las imágenes patrón . 54
4.14. Imágenes de la varianza y la varianza cuantificada del primer patrón 57
4.15. Proceso de combinación de la imagen ULBP e imagen varianza . 59
4.16. Suma de histogramas de las imágenes ULBP y varianza combinadas 60
4.17. Histograma patrón utilizando "LBP y varianza combinado" método 2 60
4.18. Conjunto de imágenes laparoscópicas de prueba (misma numeración que los nombres de los archivos originales) 62
4.19. Imagen laparoscópica de ejemplo en color y blanco y negro . . . 63
4.20. Imagen mLBP y ULBP correspondiente a la imagen laparoscópica 4 64
4.21. Imagen ULBP dividida en las 88 teselas 65
4.22. Histograma de dos teselas de la imagen ULBP y patrón ULBP . 66
4.23. Imagen resultado de la imágen laparoscópica 4 67
4.24. Imagen resultado utilizando ULBP y varianza por separado . . . 69
4.25. Imagen resultado utilizando la variante "LBP y varianza combinado, método 1" 70
4.26. Imagen resultado utilizando la variante "LBP y varianza combinado, método 2" 71

5.1. Grupos de imágenes laparoscópicas 74
5.2. Conjunto de muestras 74
5.3. Representación gráfica de la precisión y la sensibilidad 75
5.4. Comparaciones de operadores sobre imagen laparoscópica nº 25 78
5.5. Evolución de la sensibilidad y la precisión utilizando diferentes umbrales 79
5.6. Influencia del tamaño de tesela en la imagen laparoscópica nº4 80
5.7. Estadísticas utilizando diferentes vecindarios 82
5.8. Comportamiento de los parámetros P y R por separado 83
5.9. Influencia del umbral de la varianza 85
5.10. Resultados utilizando los dos criterios de aceptación de gasas . . 86
5.11. Resultados para diferentes combinaciones de umbrales ULBP y VAR 87
5.12. Resultados para un grupo reducido de umbrales 88
5.13. Ejemplo de ejecución de la segunda variante del programa utilizando los umbrales 8 | 160 88
5.14. Resultados para diferentes umbrales utilizando el primer método de la varianza combinada con ULBP 89
5.15. Ejemplo de ejecución de la tercera variante del programa utilizando el umbral 50 90
5.16. Resultados para diferentes umbrales utilizando el segundo método de la varianza combinada con ULBP 91
5.17. Ejemplo de ejecución de la cuarta variante del programa utilizando el umbral 6 91
5.18. Comparación de resultados utilizando las diferentes variantes . . 93

6.1. Ejemplo de un falso positivo aislado 99

Índice de tablas

1.1. Estrategias para prevenir material quirúrgico retenido 9

4.1. Valores del histograma normalizado ULBP y de la varianza . . . 55
4.2. Rango de las diferentes imágenes patrón 58

5.1. Resultados obtenidos para la comparación de operadores 77
5.2. Puntos significativos de la gráfica de umbrales 78
5.3. Umbrales utilizados para los diferentes vecindarios 82
5.4. Puntos significativos de la gráfica de umbrales 85
5.5. Resultados de tiempos de ejecución 95

Capítulo 1

Introducción y objetivos

El presente Trabajo Fin de Grado se ha realizado en el Departamento de Ingeniería de Sistemas y Automática de la Escuela de Ingenierías Industriales de la Universidad de Valladolid.

Se pretende desarrollar un sistema de visión artificial que identifique las gasas en tiempo real durante cirugías laparoscópicas, con el fin de evitar el olvido de este material quirúrgico dentro del paciente. A continuación, se realiza una breve introducción de diferentes campos relacionados con el tema de este TFG: cirugía mínimamente invasiva, cirugía robótica y material quirúrgico retenido.

1.1. Cirugía mínimamente invasiva

1.1.1. Introducción

La cirugía mínimamente invasiva (CMI) se puede definir como el conjunto de técnicas de diagnóstico y terapéuticas que, por visión directa, endoscopia[1] u otras técnicas de imagen, utiliza vías naturales o incisiones mínimas para introducir herramientas y actuar en diferentes partes del cuerpo humano.

Se trata de una técnica relativamente reciente que comenzó su expansión a partir de los años 80. Se suele considerar la colecistectomía laparoscópica (extirpación de la vesícula biliar) como el evento que define el crecimiento de las cirugías mínimamente invasivas. Este procedimiento es realizado por primera vez en 1985 en Alemania. (Figura 1.1).

A partir de entonces, su desarrollo ha sido muy rápido y no tiene precedentes en la historia de la cirugía. Por ejemplo, en 1993, la colecistectomía laparoscópica alcanzó en Estados Unidos un 67 % frente a los procedimientos de cirugía abierta.

[1]Endoscopia: técnica diagnóstica que permite explorar cavidades dentro del cuerpo mediante una sonda flexible que tiene una pequeña cámara y una luz en su extremo. Esta sonda se denomina endoscopio.

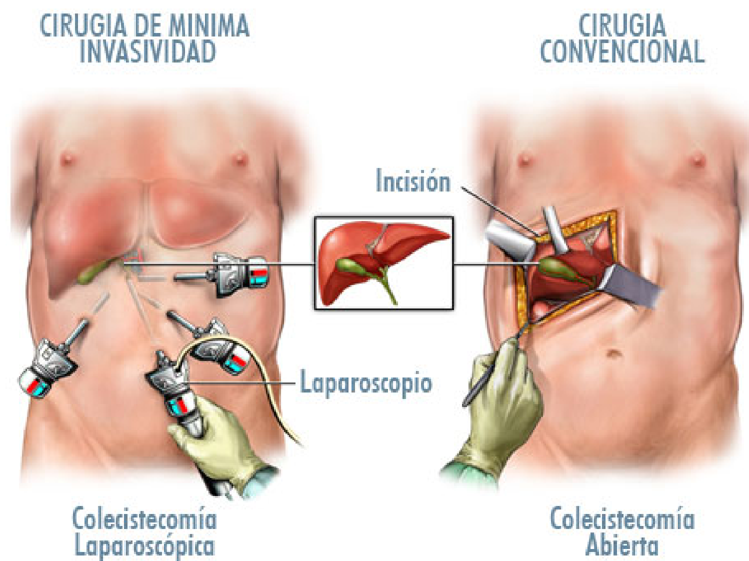


Figura 1.1: *Comparativa de la colecistectomía mediante cirugía convencional y cirugía mínimamente invasiva.*

Pese a la rápida expansión de este procedimiento en particular, la adopción de estas técnicas en otro tipo de operaciones ha sido mucho más lenta. El principal motivo es la dificultad en el aprendizaje de estos procedimientos por parte del colectivo médico.

Aunque el área donde mayor desarrollo se ha alcanzado es el aparato digestivo, la CMI es un concepto global que abarca casi todas las disciplinas médicas (cirugía torácica, pediátrica, ginecología, urología, traumatología, plástica, ortopédica, cardiaca y vascular, neurocirugía y otorrinolaringología) y técnicas (endoscopia, cirugía laparoscópica y percutánea).

1.1.2. Clasificación

Podemos clasificar las CMI según el espacio anatómico donde es aplicada:

- Cirugía endocavitaria: toracoscopia (cavidad torácica), laparoscopia (cavidad abdominal) y artroscopia (articulaciones).

- Cirugía endoluminar: ORL (otorrinolaringología), respiratoria, digestiva, urológica, ginecológica, angioscopia, vascular y pelvioscopia.

- Otros accesos: axilar, mediastino, retroperitoneo, preperitoneo y perivascular.

1.1.3. Ventajas e inconvenientes

Ventajas

La CMI presenta una serie de ventajas que pueden aplicarse a cualquier procedimiento. Estas ventajas se resumen en los siguientes puntos:

- Reducción de la respuesta inflamatoria asociada a la cirugía y mejora en la respuesta inmunológica.
- Disminución del dolor postoperatorio, debido a la ausencia de incisiones quirúrgicas importantes y a la reducción del trauma en los tejidos sanos.
- Posibilidad de evitar anestesia general.
- Menores complicaciones en la herida quirúrgica. Esto es debido a que las heridas tienen diámetros menores, y, por lo tanto, cicatrizan de forma más rápida. Además, raramente presentan complicaciones importantes.
- Mejora del factor estético, debido al reducido tamaño de las incisiones.
- Disminución del postoperatorio y la estancia en el hospital, lo que permite reducir la morbilidad postoperatoria y una inserción laboral más rápida.

Inconvenientes

- Dificultad en la percepción espacial: debido a que las operaciones son controladas a través de monitores, se pierde la visión binocular que nos dota de visión tridimensional. Como consecuencia, se pierde la sensación de profundidad.
- Imposibilidad de palpación y sensación: dado que la sensación y el tacto de la cirugía convencional se pierde, es necesario aprender a palpar con los instrumentos.
- Dificultad o imposibilidad de controlar un sangrado importante durante la intervención.
- Proceso de suturación más lento y complicado.

1.1.4. Situación actual

Actualmente, la CMI se encuentra en fase de evolución y está remplazando lenta y progresivamente a la cirugía convencional, reduciendo la morbilidad de los procedimientos y con un porcentaje de éxitos terapéuticos cada vez mayor.

Muchas de las técnicas de CMI son previamente exploradas en centros académicos y poco a poco se van extendiendo a los hospitales.

Esta implantación generalizada se está viendo acelerada por la difusión en medios de comunicación y el enorme desarrollo tecnológico emprendido por las empresas.

1.2. Cirugía robótica

Aunque el sistema de seguimiento de gasas quirúrgicas que se presenta puede ser empleado directamente por el cirujano, el objetivo es su integración futura en un sistema robotizado.

El primer robot utilizado en cirugía robótica fue el Puma 560 (Figura 1.2), aunque no se trata de un robot especializado en cirugía, sino un brazo robótico industrial de ámbito general. Se utilizó para realizar biopsias neuroquirúrgicas con mayor precisión. En 1988 se utilizó el mismo sistema para llevar a cabo una prostatectomía transuretral.

Figura 1.2: *Puma 560. Primer robot utilizado en cirugía robótica.*

Más adelante, una empresa llamada Integrated Surgical Supplies Ltd. diseñó dos robots destinados únicamente al uso médico:

- PROBOT: fue concebido para la prostatectomía transuretral.
- ROBODOC: sistema robótico encargado del vaciado del fémur con mayor precisión en operaciones de sustitución de cadera. Se trata del primer robot aprobado por la FDA[2].

[2]FDA (Food and Drug Administration): agencia del gobierno de Estados Unidos responsable de la regulación de alimentos, medicamentos, cosméticos, aparatos médicos, productos biológicos y derivados sanguíneos, entre otros.

A continuación, resumimos los sistemas robóticos más importantes diseñados específicamente para cirugía robótica.

AESOP

El sistema endoscópico automático para posicionamiento óptico (AESOP, *Automated Endoscopic System for Optical Positioning*) se trata del primer robot para intervenciones quirúrgicas abdominales aprobado por la FDA. Fue diseñado por Computer Motion en California, Estados Unidos y aprobado en 1994 (Figura 1.3).

Se trata de un brazo robótico que sujeta una cámara laparoscópica y puede ser controlado por voz. Las últimas generaciones han añadido 7 rangos de movimiento, que simulan la mano humana.

Figura 1.3: *Robot AESOP.*

ZEUS

El sistema ZEUS fue diseñado por Computer Motion (al igual que AESOP) en 1998 (Figura 1.4).

Este sistema introdujo el concepto de telerrobótica o telepresencia en la cirugía robótica. Consta de una consola de control para el cirujano con un sistema de vídeo tridimensional y una mesa operatoria con tres brazos robóticos

con cuatro rangos de movimiento. Los brazos derecho e izquierdo simulan los brazos del cirujano, mientras que el tercer brazo es un endoscopio AESOP.

El principal inconveniente del sistema ZEUS es el gran tamaño de los brazos robóticos, lo que limita el espacio en las salas quirúrgicas y causa colisiones entre los trócares[3]. Además, para poder tener visión tridimensional es necesario utilizar una gafas especiales.

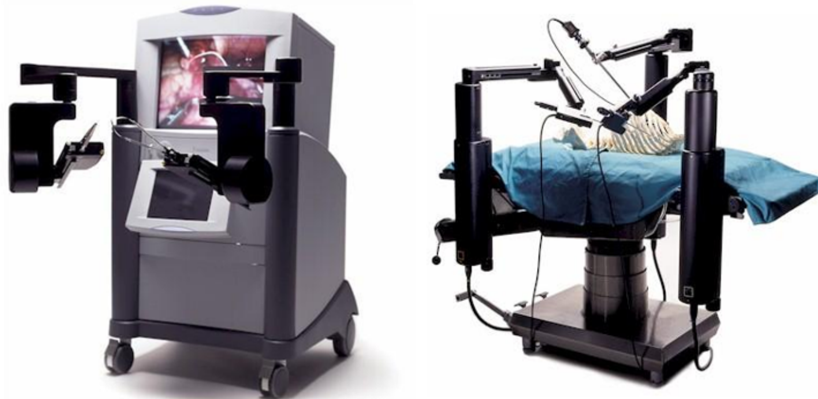


Figura 1.4: *Sistema ZEUS*

Sistema quirúrgico da Vinci

El sistema quirúrgico da Vinci es el sistema de cirugía robótica más completo desarrollado hasta la fecha (Figura 1.5).

Consta de tres componentes:

- El carro de visualización: aloja un equipo de iluminación dual y cámaras dobles.
- La consola del cirujano: consta de dos mandos que controlan los brazos robóticos con 7 rangos de movimiento, un ordenador y un sistema de imágenes 3D.
- El carro móvil: sostiene los tres brazos para instrumentos y el brazo de la cámara.

Un sensor de infrarrojos detecta cuándo el cirujano introduce la cabeza en la consola, activando los dos mandos y los brazos robóticos.

[3]En su forma más simple, un trocar es un instrumento en forma de lápiz con una punta afilada en un extremo triangular. Son utilizados típicamente dentro de un tubo hueco, conocido como cánula, para crear una abertura en el cuerpo y proporcionando acceso a este durante la cirugía. Los trócares robóticos son tubos metálicos sencillos que sirven como punto de entrada para los instrumentos robóticos.

Figura 1.5: *Sistema quirúrgico da Vinci*

Aun así, el sistema quirúrgico da Vinci presenta los siguientes inconvenientes:

- Su gran tamaño limita el espacio en el quirófano.
- Requiere un gran número de conexiones delicadas que se encuentran dentro de la sala de operaciones y que pueden causar accidentes o sufrir daños.
- En algunas operaciones, como la resección del intestino, se necesita acceder a uno o más cuadrantes abdominales, lo que obliga al montaje y desmontaje de los brazos robóticos. Esto implica un aumento en el tiempo de operación, y por lo tanto, el tiempo que tiene que permanecer el paciente con anestesia.

1.3. Material quirúrgico retenido

Gossypiboma, textiloma o gasoma son términos que se refieren a material quirúrgico (como esponjas, gasas, agujas o instrumentos), que ha sido olvidado dentro del cuerpo del paciente durante una operación. Esta situación tiene un fuerte impacto en la salud del paciente, los responsables de la operación y el proveedor de servicios médicos.

En estudios realizados en los años 80, el número de incidencias por material quirúrgico retenido (MQR) variaba entre 1 de cada 1000 y 1 de cada 1500 en operaciones abdominales. En cambio, estudios más modernos muestran que

esta probabilidad ha disminuido entre 1 de cada 5500 y 1 de cada 18760 casos de MQR en operaciones abdominales.

Este amplio rango en la probabilidad se debe a diferentes factores:

- La naturaleza retrospectiva de los estudios.
- La reticencia por parte de los hospitales y clínicas a exponer estos errores públicamente.
- El descubrimiento de MQR tras años de la operación, ya que el paciente puede permanecer asintomático.
- Requisitos de confidencialidad.

Los síntomas son variados y dependen del lugar y del tipo de reacción provocada. A continuación, se muestran diferentes cuadros clínicos:

- Asintomático: detección accidental.
- Sintomático
  - Detección temprana: dolores sin explicación, septicemia[4] generalizada, formación de forúnculos, etc.
  - Detección tardía: los huesos son incapaces de repararse, fístulas internas, etc.

Los objetos retenidos pueden causar múltiples problemas, y su tratamiento requiere una cirugía para su eliminación, incluso cuando el paciente es asintomático. El tiempo que transcurre entre la primera operación, el diagnóstico y la eliminación del MQR, es crítico. La mortalidad es mucho menor cuando el MQR se elimina inmediatamente después de la operación.

Existen diferentes métodos que ayudan a disminuir el MQR. La técnica más simple y utilizada es contar el material que entra y el que sale, de forma que, si hay una discordancia entre ambos números, significa que hay material quirúrgico en el paciente.

Aun así, este método es propenso a fallos. Varios estudios proponen el desarrollo de un protocolo que debe aplicarse en el preoperatorio y múltiples veces durante la operación. Si hay alguna discrepancia en la cuenta de material quirúrgico, se deberá buscar el objeto en el cuerpo de paciente, y en caso de no encontrarlo, se deberá realizar un escáner apropiado. La principal desventaja de estos protocolos es su laboriosidad, ocupando hasta el 14 % del tiempo de operación.

[4]Septicemia: presencia de bacterias en la sangre. Se trata de una infección grave y potencialmente mortal que empeora de forma muy rápida.

Existen otras técnicas para evitar el Gossypiboma. Una de ellas consiste en añadir marcadores radiopacos (no permiten el paso de rayos X) a las esponjas o gasas, de forma que serán visibles en una radiografía convencional. No obstante, la realización de radiografías aumenta el riesgo quirúrgico, dado que aumenta el tiempo en el quirófano bajo los efectos de la anestesia. Además, se somete al paciente a radiación.

Todas las esponjas quirúrgicas comercializadas fabricadas en Estados Unidos contienen estos marcadores radiopacos. Estos marcadores pueden estar contenidos entre el tejido o unidos a la esponja a través de un hilo.

Otra técnica consiste en la utilización de etiquetas por radiofrecuencia (RFID, Radio Frequency Identification). La utilización de estas etiquetas en las esponjas quirúrgicas permite su detección utilizando una antena especializada, evitando tener que realizar una radiografía al paciente.

En la tabla 1.1 se resumen las ventajas e inconvenientes de diferentes estrategias para prevenir el MQR.

Estrategia preventiva	Ventajas	Desventajas
Contar material	• Procedimiento estándar	• Alta ocupación de tiempo • Propenso a errores
Radiografía intraoperatoria	• Fácil de utilizar • Daño clínico insignificante	• Bajo rendimiento • Baja calidad de imagen
Radiografía de alta resolución en el postoperatorio	• Daño clínico insignificante	• Alto coste • Alto porcentaje de falsos negativos (10-25%) • Exposición a radiación innecesaria
Esponjas etiquetadas con un chip de identificación por RF	• Alta precisión en la detección	• Eficacia no contrastada • Falta de prueba de control aleatorizada
Esponjas etiquetadas con un código de barras	• Tecnología en uso en medicina • Mejora la detección de errores al contar • Implementada en una única institución, aunque con buenos resultados	• Aumenta el tiempo necesario para contar • Curva de aprendizaje para adaptarse a nuevas tecnologías • Relación coste/beneficio aún por determinar

Tabla 1.1: *Estrategias para prevenir material quirúrgico retenido*

1.4. Objetivos

El objetivo principal de este Trabajo Fin de Grado es desarrollar un software que permita la identificación de gasas quirúrgicas en cirugías laparoscópicas utilizando visión artificial, con el fin de evitar situaciones de material quirúrgico retenido. Para ello, se realizará un análisis de textura de las imágenes utilizando el operador *Local Binary Pattern* (LBP).

Las ventajas de utilizar visión artificial como estrategia de prevención de MQR son la nula ocupación de tiempo durante la operación y el hecho de no necesitar material especial (gasas con RFID o marcadores radiopacos), ya que únicamente son necesarias las imágenes capturadas por el endoscopio.

La principal dificultad que presenta la detección de gasas utilizando visión artificial es el cambio de apariencia que sufren estas a lo largo de la operación (diferentes orientaciones y presencia o no de fluidos).

El lenguaje de programación utilizado será C++, en combinación con la librería de procesamiento de imágenes OpenCV. La programación se realizará en un entorno Linux, concretamente utilizando la distribución Ubuntu.

Además, se han planteado los siguientes objetivos intermedios:

- Realización de un estudio de las diferentes técnicas de visión artificial de análisis de texturas.
- Comprobar la efectividad de las diferentes variantes del operador LBP.
- Permitir la detección de gasas en diferentes condiciones de sangrado y orientación.
- Añadir el contraste como parámetro para el análisis de las imágenes, a través de la varianza.
- Optimización del código para permitir su uso en tiempo real.

Capítulo 2

Imagen digital y visión artificial

En este capítulo se hace una breve introducción de los fundamentos a partir de la cuales se pude desarrollar este programa: la imagen digital y la visión artificial. También se hablará de OpenCV, una librería que facilita el uso de la visión artificial en diferentes lenguajes de programación.

2.1. Imagen digital

2.1.1. Representación

Una imagen digital es una representación bidimensional de una imagen a partir de una matriz numérica.

Cada posición en esa matriz se denomina píxel, que es la unidad homogénea mínima en una imagen. El valor que tiene cada píxel en una imagen no es más que la intensidad lumínica en ese punto cuantificada a un rango de valores.

Un sensor fotográfico se encarga de convertir la intensidad luminosa en impulsos eléctricos, y posteriormente es digitalizada. El rango de valores que suele utilizarse en la digitalización es de 0 a 255, asignando el valor 0 al negro y el valor 255 al blanco.

De esta forma obtenemos una imagen con 256 tonos de grises.

Para dotar a una fotografía con color, los píxeles del sensor deben ser sensibles a diferentes colores. Hay dos maneras de conseguir esto:

- Utilizar 3 sensores y descomponer la imagen en los 3 colores primarios (rojo, verde y azul), de forma que cada uno de los colores vaya dirigido a un sensor. De esta forma obtenemos 3 imágenes.

- Utilizar un único sensor con filtros ópticos integrados en cada uno de los píxeles. Se utilizaran filtros de los 3 colores primarios, y será necesario interpolar para obtener las 3 imágenes completas de cada uno de los colores.

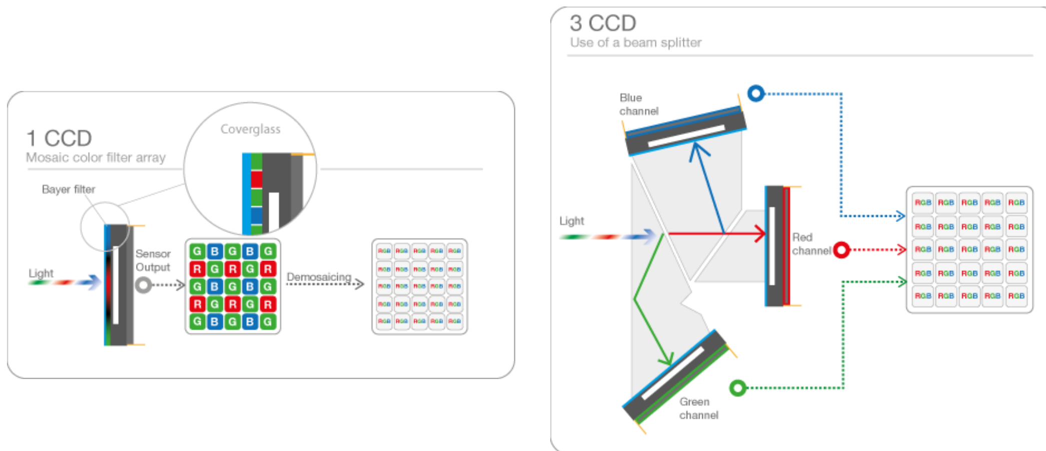


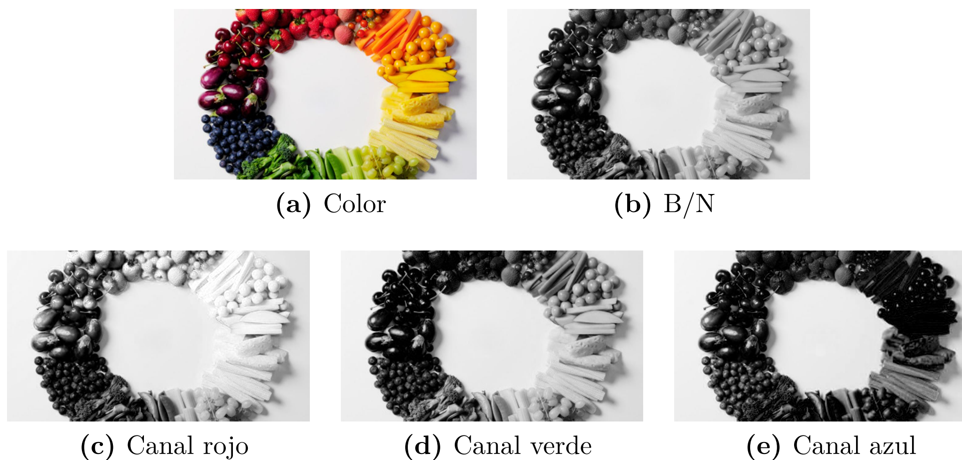
Figura 2.1: *Dos métodos para obtener imágenes a color*

La imagen 2.1 ilustra el funcionamiento de ambas técnicas.

Si queremos obtener una imagen en blanco y negro a partir de los 3 canales a color, se calcula de la siguiente manera.

$$IMG_{B/N} = 0{,}2989 \cdot IMG_{rojo} + 0{,}5870 \cdot IMG_{verde} + 0{,}1140_{azul} \qquad (2.1)$$

En la figura 2.2 se muestra una imagen a color, en blanco y negro y sus tres componentes fundamentales.



(a) Color **(b)** B/N

(c) Canal rojo **(d)** Canal verde **(e)** Canal azul

Figura 2.2: *Canales de una imagen a color*

Se puede observar que en la zona de las berenjenas, el canal rojo aparece más claro, después el azul y por último el verde. En la figura 2.3 mostramos esa región de la imagen, junto con el nivel de rojo, verde y azul de cada píxel.

Figura 2.3: *Valores de cada uno de los canales de una imagen a color*

2.1.2. Tipos de imágenes digitales

Existen dos grandes tipos de imágenes digitales: imágenes ráster (o mapas de bits) e imágenes vectoriales.

2.1.2.1. Imágenes ráster

Las imágenes ráster (o mapas de bits) son aquellas imágenes cuya matriz numérica es estática y cada posición es conocida como píxel.

Una imagen ráster se define a partir de las dimensiones horizontales y verticales de su matriz numérica, expresadas en términos de píxeles. Se trata del tipo de imagen más utilizando en general, y concretamente en el ámbito de la visión artificial.

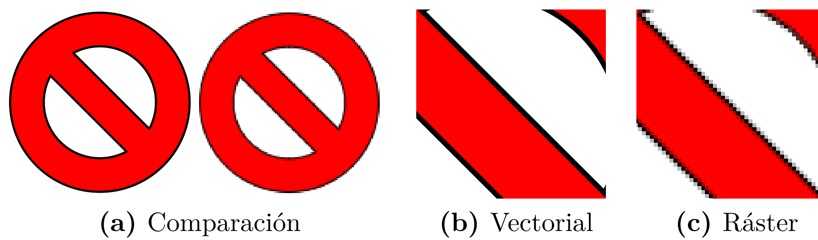
En la figura 2.3 se puede observar los diferentes píxeles por los que está compuesta una imagen ráster.

2.1.2.2. Imágenes vectoriales

Las imágenes vectoriales tienen una matriz numérica dinámica. Todos los elementos de estas imágenes se representan en términos de primitivas matemáticas. Por ejemplo, una línea suele representarse por un punto inicial, uno final y su anchura. Las figuras curvas pueden representarse o bien en forma de pequeñas líneas rectas unidas o segmentos de curvas polinómicas.

Es ampliamente utilizado por artistas y diseñadores a la hora de desarrollar el gráfico, y posteriormente son "rasterizadas" para su distribución.

En la figura 2.4 se realiza una comparación entre una imagen ráster y una imagen vectorial.



(a) Comparación **(b)** Vectorial **(c)** Ráster

Figura 2.4: *Diferencias entre una imagen vectorial y ráster*

Podemos comprobar cómo al acercar la imagen vectorial no se pierde calidad, mientras que con la imagen ráster se pueden diferenciar los píxeles.

2.1.3. Otras características

2.1.3.1. Resolución

La resolución de un sensor es el número de puntos con el que se muestrea la imagen continua que proyecta la óptica sobre el sensor.

De forma análoga, la resolución de una imagen digital es el número de píxeles en su componente tanto horizontal y vertical.

En la figura 2.5 se muestra un logo con diferentes resoluciones. Por norma general, cuanto mayor es la resolución, mayor es el grado de detalle.

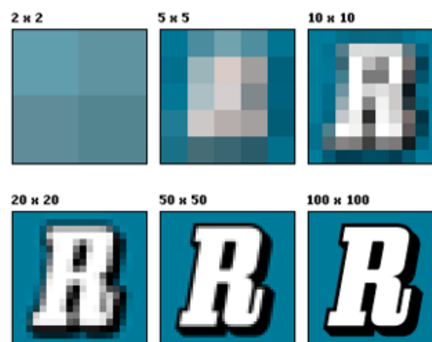


Figura 2.5: *Logo representado con diferentes resoluciones*

2.1.3.2. Profundidad de color

La profundidad de color determina el número de colores que es capaz de representar un determinado formato de imagen.

La profundidad de color más habitual es de 24 bits, 8 bits por cada uno de los 3 canales a color.

Esto hace un total de $2^{24} = 16\,777\,216$ colores representables.

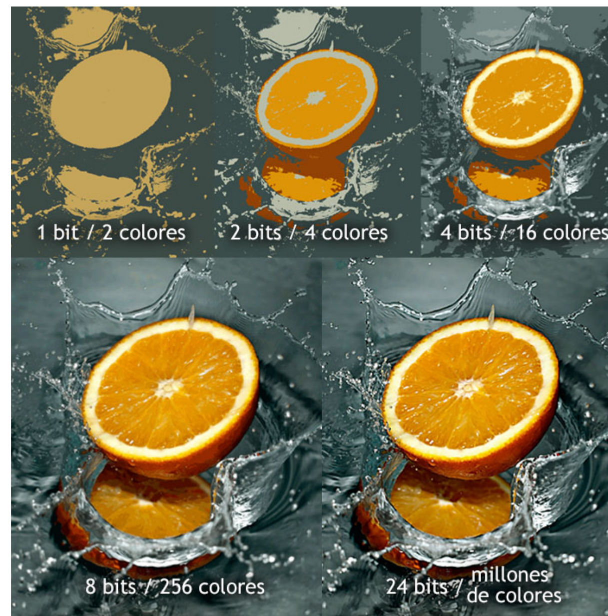


Figura 2.6: *Imagen representada con diferentes profundidades de color*

En la figura 2.6 se muestra la misma imagen representada con diferentes profundidades de color.

2.2. Visión artificial

Desde una perspectiva general, la visión artificial por computador es la capacidad de la máquina para ver el mundo que le rodea, más precisamente, para deducir la estructura y las propiedades del mundo tridimensional a partir de una o más imágenes bidimensionales.

La escena tridimensional es vista por una o más cámaras para producir imágenes monocromáticas o en color. Las imágenes adquiridas pueden ser segmentadas para obtener de ellas características de interés tales como bordes o regiones. Posteriormente, de las características se obtienen propiedades subyacentes mediante el correspondiente proceso de descripción. Tras lo cual se consigue la estructura de la escena tridimensional requerida por la aplicación de interés.

Un sistema de visión artificial está compuesto, generalmente, por 5 elementos principales: la cámara, la óptica, la iluminación, el software y el sistema de procesamiento. Una elección óptima de cámaras, ópticas e iluminación, disminuye en gran medida la complejidad del software, además de mejorar la robustez.

Existen gran cantidad de librerías y plataformas software de tratamiento de imágenes. Muchas de ellas son propietarias, aunque la alternativa más popular

de código abierto se llama OpenCV.

2.2.1. OpenCV

OpenCV (*Open Source Computer Vision Library*) es una librería de visión artificial liberada bajo una licencia BSD, que indica que es gratis para uso tanto académico como comercial. Tiene interfaces para C++, Python y Java, y soporta Windows, Linux, Mac OS, iOS y Android.

Ha sido diseñada pensando en la eficiencia computacional y enfocada en aplicaciones tiempo real. Está escrita en C/C++, pudiendo hacer uso del procesamiento multinúcleo [14].

Se trata de la librería utilizada para el desarrollo del programa de detección de gasas.

2.2.2. Aplicaciones de la visión artificial

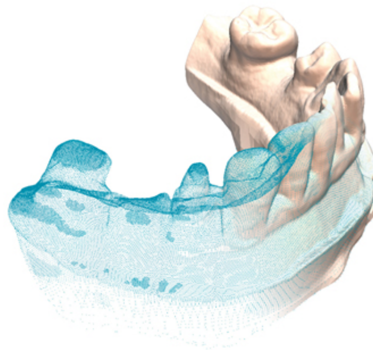


Figura 2.7: *Modelo CAD de una dentadura*

La visión artificial tiene un campo de aplicación muy amplio, como por ejemplo, inspecciones automáticas de calidad, navegación (vehículos autónomos), reconocimiento facial, reconocimiento de caracteres, aplicaciones militares, etc.

Dado el contexto del presente TFG, nos centraremos en las aplicaciones médicas.

En general, los tipos de imágenes con los que se trabaja son imágenes microscópicas, imágenes de rayos X, imágenes de angiografía[1], imágenes ultrasónicas e imágenes de tomografía.

El análisis de estas imágenes permiten la detección de tumores o arteriosclerosis. También se pueden llevar a cabo mediciones de órganos, el flujo de sangre, etc.

[1]Una angiografía es una radiografía de los vasos sanguíneos de una zona determinada del organismo.

Este área de aplicación también contribuye a la investigación médica, proporcionando nueva información, por ejemplo, acerca de la estructura del cerebro o la calidad de los tratamientos médicos.

Otra aplicación de la visión artificial en el área médica es la mejora de las imágenes que son interpretadas por los seres humanos, como las imágenes ultrasónicas o rayos X, reduciendo el ruido en estas.

En la figura 2.7 se muestra un modelo CAD de una dentadura a partir de un molde dental, utilizando dos cámaras y un ordenador comercial.

También existen herramientas que permiten mover el cursor de un ordenador a partir del movimiento ocular utilizando visión artificial, ofreciendo grandes beneficios a personas con capacidades reducidas.

2.3. Conclusiones

En este capítulo se ha presentado muy brevemente la estructura de una imagen digital y algunos de sus conceptos básicos, como la resolución y la profundidad del color.

La visión artificial es una tecnología que se extiende a todos los campos y está en continuo crecimiento. En el presente TFG nos hemos limitado a mencionar algunas aplicaciones en el ámbito médico.

Existen diferentes librerías y programas para la visión artificial, aunque se ha optado por utilizar OpenCV porque es la libraría más empleada y además su uso es gratuito.

Capítulo 3

Análisis de texturas

3.1. Texturas y tipos de descriptores

Aunque la textura es un campo de investigación importante en el ámbito de la visión artificial, no existe una descripción formal de esta.

El principal motivo por el que la definición de textura no es universal, es que las características a menudo son opuestas en diferentes tipos de imagen: regularidad frente a aleatoriedad, uniformidad frente a heterogeneidad, etc.

A lo largo de los años, diferentes investigadores han tratado de definir la textura desde el punto de vista de su naturaleza:

- «Región organizada que puede ser descompuesta en primitivas teniendo unas distribuciones espaciales concretas». Esta definición es conocida como la aproximación estructural, que se basa en la idea de que cada textura está compuesta por una serie de elementos básicos (puntos, aristas, etc.), similar a la que utiliza el ser humano.

- «Una región de la imagen bidimensional, aleatoria y posiblemente periódica». Define la textura desde un punto de vista estocástico.

- «La distribución espacial de color o intensidad en una imagen o en un región de la misma». Definición orientada al análisis de imágenes.

3.1.1. Tipos de textura

En la actualidad, las texturas se pueden clasificar en diferentes categorías, dependiendo del problema que se desee resolver:

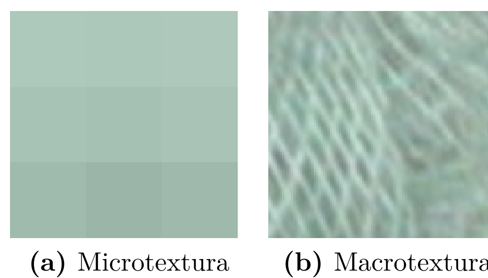
- Microtexturas o macrotexturas.

- Irregulares o regulares.

- Repetitivas o no repetitivas.

- Direccionales o no direccionales.
- Granulada y de baja complejidad, frente a no granulada y de alta complejidad.

Atendiendo a esta clasificación, la textura de las gasas se puede agrupar en las siguientes categorías: por una parte, el operador LBP (se explica a continuación en la sección 3.2) analiza la textura desde un punto de vista de microtextura, mientras que a la hora de determinar la presencia de una gasa, se analizará la macrotextura. La figura 3.1 muestra la zona de acción del operador LBP, frente a la zona de decisión.

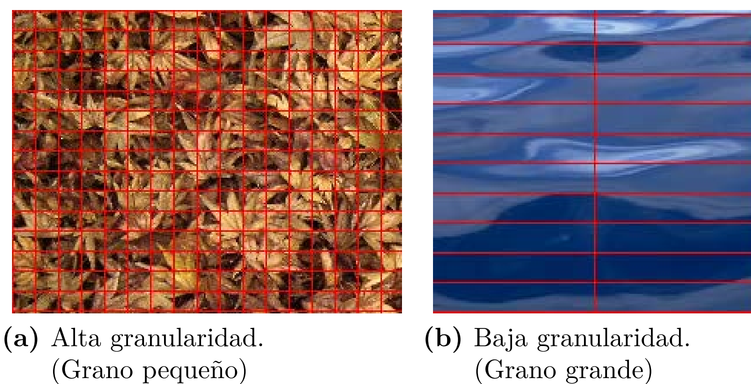


(a) Microtextura **(b)** Macrotextura

Figura 3.1: *Comparación macrotextura y microtextura*

Se trata de una textura irregular, ya que según los pliegues de esta y la presencia o no de sangre, hace que su apariencia cambie significativamente. A pesar de esta irregularidad, se puede considerar una textura repetitiva, ya que para una misma apariencia el patrón se repite a lo largo de toda la gasa.

Respecto a la direccionalidad, la gasa tiene una textura no direccional, ya que su apariencia es independiente de la orientación.



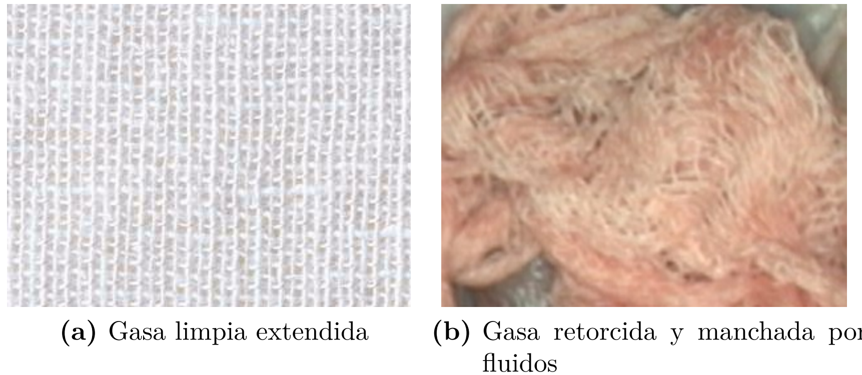
(a) Alta granularidad. (Grano pequeño) **(b)** Baja granularidad. (Grano grande)

Figura 3.2: *Ejemplos de texturas con diferente granularidad*

El grano de una textura es el tamaño de su primitiva de textura. Cuanto mayor sea el tamaño de la primitiva de textura, menor será la granularidad.

Análogamente, cuanto menor sea el tamaño de la primitiva de textura, mayor será la granularidad. En la figura 3.2 mostramos un ejemplo de una textura de cada tipo.

Por lo tanto, cuando la gasa está desplegada y no está manchada por fluidos, se trata de una textura granulada de baja complejidad. En cambio, cuando está retorcida y empapada por fluidos, el grano es mayor y se convierte en una textura no granulada de alta complejidad (Figura 3.3).



(a) Gasa limpia extendida **(b)** Gasa retorcida y manchada por fluidos

Figura 3.3: *Comparación de la apariencia de la gasa y su granularidad*

3.1.2. Características de los descriptores de textura

Debido a la alta complejidad y variabilidad de las texturas, es muy difícil crear un descriptor que se adapte a todas ellas y sea capaz de identificarlas a la perfección. Sin embargo, un buen descriptor de textura debe cumplir los siguiente requisitos:

1. Discriminante, diferenciando entre múltiples tipos de texturas.
2. Robusto frente a variaciones en el escalado y la posición.
3. Robusto frente a variaciones en la iluminación.
4. Robusto frente a la falta de uniformidad espacial.
5. Debe funcionar para imágenes de pequeño tamaño.
6. Eficiente, teniendo baja complejidad computacional.

Sin embargo, es muy difícil encontrar técnicas de descripción de texturas que cumplan con todas estas características. Es por ello que en la actualidad hay gran actividad investigadora en este campo, y cada año aparecen nuevas técnicas que intentan satisfacer los requisitos anteriores.

3.1.3. Tipos de descriptores de textura

Los descriptores de textura pueden clasificarse en diferentes tipos en función de las propiedades de la imagen que se tienen en cuenta a la hora de obtener sus características:

- Estructurales: representan la textura por medio de una jerarquía y primitivas bien definidas. Para ello, primero hay que especificar las primitivas y luego las reglas de posicionamiento. Estos métodos tratan de expresar de manera rigurosa la estructura de la región, por lo que funcionan mejor en texturas regulares y repetitivas, mientras que se comportan peor cuando se tratan de texturas naturales debido a la alta variabilidad que presentan.

- Estadísticos: estas técnicas no pretenden describir la estructura jerárquica de la textura, sino que la representan mediante propiedades basadas en la distribución y la relación entre los niveles de gris de la imagen. Los métodos basados en estadísticas de segundo orden han demostrado obtener buenos resultados, siendo el más conocido la matriz de co-ocurrencia.

- Basados en modelos: estos descriptores utilizan fractales[1] y modelos aleatorios para describir la estructura de la imagen, ajustando los parámetros de dichos modelos hasta encontrar los que mejor representan a la textura. La estimación de los parámetros óptimos conlleva una complejidad computacional elevada, por lo que no son apropiados en sistemas donde el tiempo es un factor clave.

- Basados en transformaciones: representan la imagen en un espacio cuyo sistema de coordenadas tiene una interpretación que está relacionada con las características propias de la textura como la frecuencia o el tiempo. Técnicas como la trasformada de Fourier, Gabor o Wavelet han sido frecuentemente utilizadas, obteniendo esta última resultados muy interesantes, sobre todo para segmentación[2] basada en textura.

[1] Un fractal es un objeto geométrico cuya estructura básica, fragmentada o aparentemente irregular, se repite a diferentes escalas.

[2] La segmentación, en el campo de la visión artificial, es el proceso de dividir una imagen digital en varias partes u objetos. El objetivo de la segmentación es simplificar y/o cambiar la representación de una imagen en otra más significativa y más fácil de analizar.

3.2. Local Binary Pattern (LBP)

El operador *Local Binary Pattern* (LBP) es un método potente y teóricamente muy simple para el análisis texturas. Además, las recientes modificaciones del operador original lo convierten en una herramienta que arroja resultados excelentes en términos de eficiencia computacional y precisión.

El resultado tras aplicar el operador, es una imagen de etiquetas enteras que describe la apariencia de una imagen a pequeña escala. Estas etiquetas, y más en concreto, sus estadísticas (habitualmente en forma de histograma) son utilizadas para el análisis de la imagen. Es un método que concilia el enfoque estructural con el estadístico.

Las versiones más utilizadas del operador están diseñadas para trabajar con imágenes monocromas, aunque también se ha extendido su uso en imágenes a color (con múltiples canales).

3.2.1. Operador LBP básico

El operador LBP básico se basa en el supuesto de que las texturas tienen dos características complementarias, un patrón y su fuerza. En su presentación, el LBP fue propuesto como una versión a dos niveles de la unidad de textura para describir patrones locales.

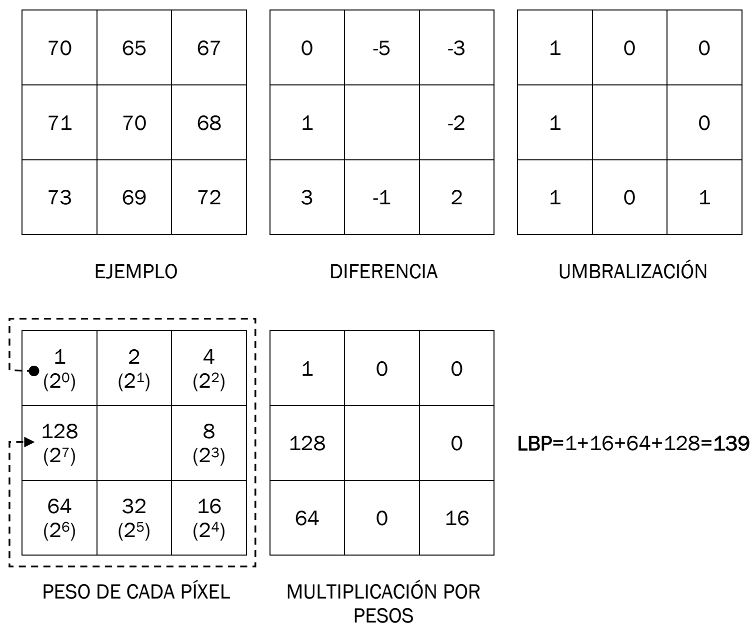


Figura 3.4: *Cálculo del valor LBP en su versión original*

La forma de proceder de este operador es etiquetar los píxeles de una imagen, comparando el nivel de gris de cada píxel con sus 8 vecinos. Para ello, se resta el nivel de gris de cada vecino del nivel de gris del píxel central.

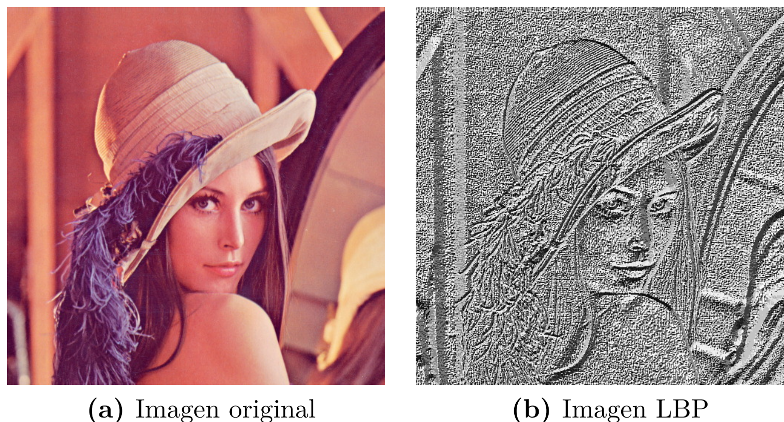
Si el resultado es mayor o igual que 0, se pone el píxel vecino a 1, y en caso contrario, a 0. A este proceso le denominamos umbralización. Estos valores forman un número binario que, convertido a decimal, es el valor LBP. No existe ningún convenio a la hora de elegir el bit de mayor y menor peso, y resulta indiferente siempre y cuando el orden sea consistente. En el ejemplo de la figura 3.4 hemos tomado la esquina superior izquierda como el bit menos significativo, y hemos seguido un sentido horario para completar el número binario.

Se recorrerá toda la imagen, y se aplicará el operador a cada uno de los bloques de 3x3 píxeles, dando lugar a una imagen que tiene 2 columnas y 2 filas menos que la imagen original. Esto es debido a que no se puede calcular el código LBP de los bordes de la imagen, ya que un píxel necesita estar rodeado por sus 8 vecinos.

Como hay 8 vecinos, se pueden obtener un total de $2^8 = 256$ posibles etiquetas.

Dada la asignación de pesos de la figura 3.4, el cálculo del valor decimal a partir de los valores umbralizados es la siguiente: $Valor\,LBP = 10001011 = 1 \cdot 2^0 + 0 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 + 0 \cdot 2^5 + 1 \cdot 2^6 + 1 \cdot 2^7 = 139$.

A continuación, aplicamos el operador LBP a una imagen (Figura 3.5).



(a) Imagen original **(b)** Imagen LBP

Figura 3.5: *Ejemplo de imagen LBP*

Observamos cómo en la imagen LBP se resaltan las diferentes texturas, como la del sombrero o la del pelo. En cambio, las superficies más homogéneas aparecen como un ruido uniforme.

También se adjunta el histograma correspondiente a la imagen LBP, que puede ser utilizado como un descriptor de textura (Figura 3.6).

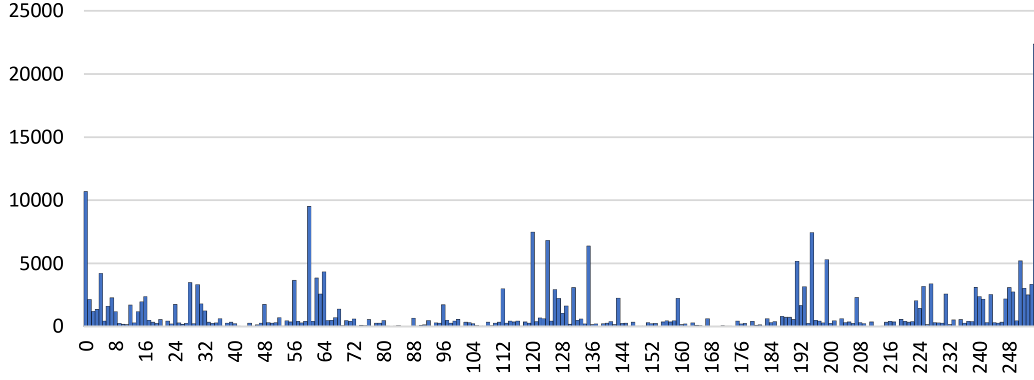


Figura 3.6: *Histograma de la imagen LBP*

3.2.2. Operador LBP genérico

Unos años después de la publicación original, se presentó un operador LBP más genérico. Mientras que el LBP básico únicamente contemplaba 8 vecinos en un bloque de 3x3 píxeles, esta nueva formulación eliminaba esos límites, pudiendo elegir entre bloques de cualquier tamaño y con cualquier número de vecinos.

Consideremos una imagen monocroma $I(x, y)$, siendo g_c el nivel de gris de un píxel arbitrario (x, y), por ejemplo: $g_c = I(x, y)$.

Además, denotamos como g_p al nivel de gris de un vecino de un vecindario circular con P vecinos y radio R, alrededor del punto (x, y):

$$g_p = I(x_p, y_p),\ p = 0, ..., P-1 \tag{3.1}$$

$$x_p = x + R \cdot cos\left(\frac{2\pi p}{P}\right) \tag{3.2}$$

$$y_p = y - R \cdot sen\left(\frac{2\pi p}{P}\right) \tag{3.3}$$

Definimos, además, la función umbral $s(\lambda)$:

$$s(\lambda) = \begin{cases} 1, & \lambda \geq 0 \\ 0, & \lambda < 0 \end{cases} \tag{3.4}$$

Con estas consideraciones, definimos el operador LBP genérico como:

$$LBP_{P,R}(x_c, y_c) = \sum_{p=0}^{P-1} s(g_p - g_c)\, 2^p \tag{3.5}$$

En la práctica, la ecuación 3.5 representa un número binario de P-bits, dando lugar a 2^P códigos LBP diferentes.

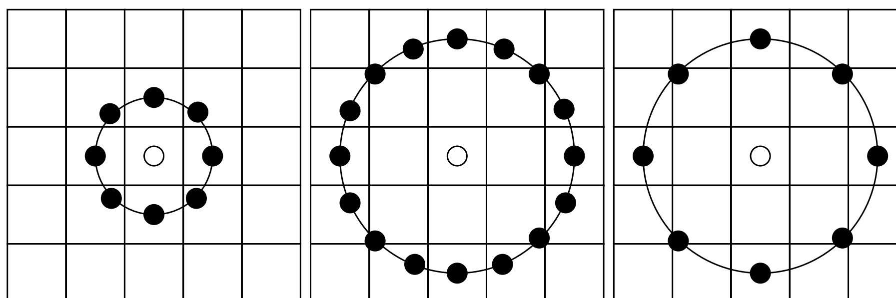


Figura 3.7: *Ejemplo de diferentes vecindarios utilizando el operador* $LBP_{P,R}$*. Los valores* (P, R) *son, de izquierda a derecha,* $(8, 1)$*,* $(16, 2)$ *y* $(8, 2)$

En la figura 3.7 se muestran 3 posibles vecindarios. Dado que se trata de un patrón circular, puede que las coordenadas de un vecino P, (x_p, y_p), no coincidan con el centro de un píxel. En tal caso, se realizará una interpolación bilineal.

La interpolación bilineal es una extensión de la interpolación lineal para interpolar funciones de dos variables. Para ello, se realiza una interpolación lineal en una dirección y después en la otra. Aunque cada paso es lineal, la interpolación en su conjunto no es lineal, sino cuadrática.

3.2.3. Invarianza rotacional y patrones uniformes

Invarianza rotacional

Dado que un código LBP se obtiene mediante un muestreo circular de los píxeles vecinos con respecto a un píxel central, hacer que un código LBP sea invariante respecto a la rotación de la imagen es una tarea relativamente simple.

Aun así, cabe señalar que la «invarianza rotacional» no tiene en cuenta las posibles diferencias en las texturas debido a la posición relativa del objeto respecto a la luz.

Cuando una imagen está rotada, los niveles de gris g_p de los vecinos de un vecindario circular se mueven alrededor del perímetro de un círculo centrado en el píxel central, cuyo nivel de gris es g_c (Figura 3.8).

Como consecuencia, los valores LBP serán diferentes y también lo serán sus histogramas. En la figura 3.9 se muestra un patrón en su forma original y rotado. Se ha calculado el histograma de su imagen LBP y se puede ver que son diferentes. Si se realizara una comparación de histogramas, el resultado no determinaría que los patrones son iguales, aunque realmente si lo son.

Suponiendo que tomamos como origen el vecino situado en la dirección positiva del eje x, y se toma una dirección de giro antihoraria, una imagen

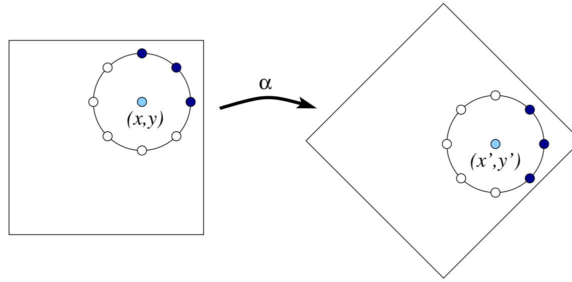


Figura 3.8: *Efecto de una imagen rotada en los puntos de un vecindario*

rotada resulta en un valor $LBP_{P,R}$ diferente. Para eliminar este efecto, cada código LBP deber ser rotado de vuelta a una posición de referencia. Esta transformación puede definirse de la siguiente manera:

$$LBP_{P,R}^{ri} = min\left\{ROR\left(LBP_{P,R}, i\right) \mid i = 0, 1, \ldots, P - 1\right\} \tag{3.6}$$

El superíndice ri significa *rotation invariant* (invarianza rotacional). La función $ROR(x, i)$ desplaza el código binario x i-veces a la izquierda.

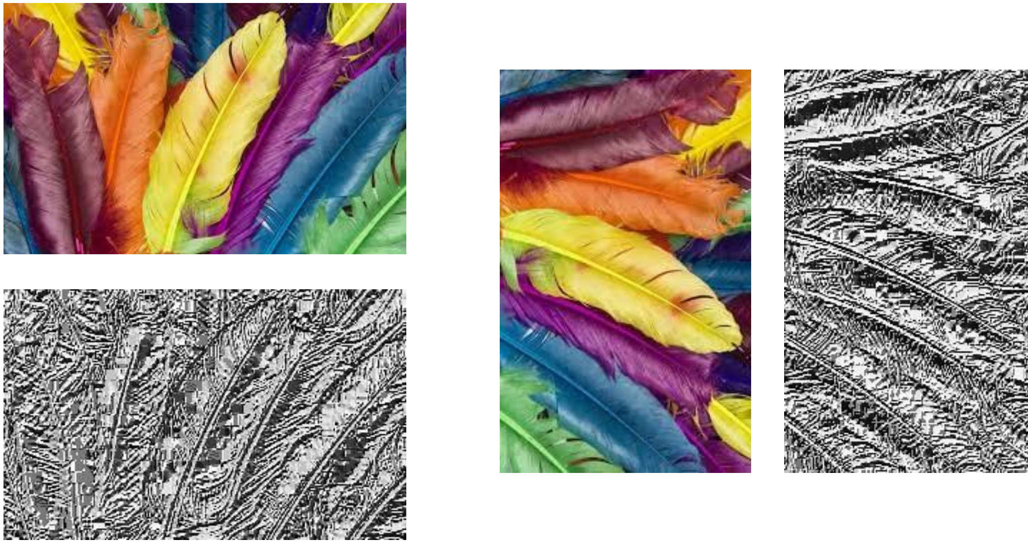
En resumen, el código de la invarianza rotacional se obtiene desplazando los bits hacia la izquierda hasta que se obtiene el valor mínimo. El operador LBPROT es equivalente al $LBP_{8,1}^{ri}$. En total, hay 36 códigos diferentes utilizando una vecindad $P = 8$.

En la práctica, como el vecindario es circular y se trabaja con 8 vecinos, con cada desplazamiento de bit estaríamos rotando la imagen en saltos de $\frac{360^{o}}{8} = 45^{o}$. Esto hace que el operador $LBP_{8,R}^{ri}$ sea poco efectivo, ya que los saltos son demasiado grandes. Además, la frecuencia con la que aparecen los 36 posibles patrones varía bastante.

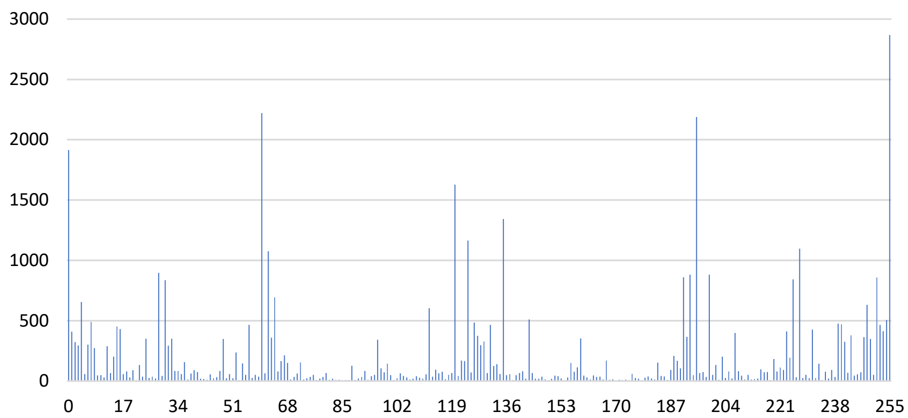
Patrones uniformes

Otra extensión del operador original utiliza los llamados *patrones uniformes*. Para ello, se utiliza una medida de la uniformidad del patrón: U. Esta medida U se define como el número de transiciones de bits de 0 a 1 o viceversa del código LBP, cuando el patrón de bits es considerado circular. Un código LBP es considerado uniforme cuando su medida de la uniformidad es como máximo 2. La definición formal de la uniformidad de un vecindario G es la siguiente:

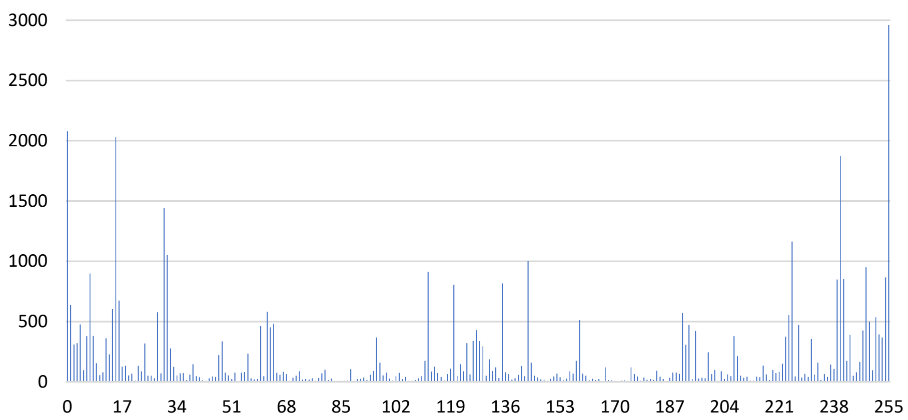
$$U\left(G_P\right) = \mid s\left(g_{P-1} - g_c\right) - s\left(g_0 - g_c\right) \mid + \sum_{p=1}^{P-1} \mid s\left(g_p - g_c\right) - s\left(g_{p-1} - g_c\right) \mid \tag{3.7}$$



(a) Imágenes originales y LBP



(b) Histograma LBP horizontal



(c) Histograma LBP vertical

Figura 3.9: *Comparación de histogramas de dos imágenes LBP rotadas*

Para un número binario de P-bits, el valor U se puede calcular eficientemente de la siguiente forma:

$$U(x) = \sum_{p=0}^{P-1} F(x \; xor \; ROR(x,1), p) \tag{3.8}$$

Siendo x un número binario. La función $F(x,i)$ extrae el i-ésimo bit del número binario x:

$$F(x,i) = ROR(x,i) \; and \; 1 \tag{3.9}$$

Si recuperamos el ejemplo de la figura 3.4, el código LBP asociado a ese vecindario era $1_1 000_2 1_3 0_4 11$, tiene 4 transiciones, por lo tanto, no se trata de un patrón uniforme.

Ejemplos de patrones uniformes podrían ser el 00000000, con 0 transiciones, o el 00110000, con 2 transiciones.

A cada patrón uniforme se le asocia una etiqueta diferente, mientras que a todos los patrones no uniformes se le asocia la misma etiqueta. Por ende, el número total de etiquetas posibles que se pueden obtener utilizando patrones uniformes de P bits es $P(P-1)+2$.

Por ejemplo, una vecindad $P = 8$, que es un número de vecinos muy habitual, utilizando patrones uniformes obtenemos $8(8-1)+2 = 58$ posibles etiquetas. En la figura 3.10 se muestran las 58 posibilidades.

Si, en cambio, utilizamos 16 vecinos, el número de etiquetas aumenta hasta 242.

Existen 2 razones para omitir los patrones que no son uniformes:

- La mayoría de los patrones en imágenes naturales son uniformes. En experimentos realizados, se ha obtenido que el 90 % de los patrones utilizados eran uniformes utilizando una vecindad (8,1), y alrededor del 70 % para una vecindad (16,2). En otros experimentos de reconocimiento facial, el 90.6 % de los patrones de vecindad (8,1) y el 85.2 % de los patrones de vecindad (8,2) eran uniformes.

- La utilización de patrones uniformes en vez de todos los posibles patrones, ha arrojado mejores resultados en muchas aplicaciones. Los estudios indican que los patrones uniformes son más estables, es decir, menos sensibles al ruido. Además, el hecho de considerar únicamente patrones uniformes, reduce significativamente el número de etiquetas posibles, haciendo que una identificación fiable de texturas requieran menos muestras.

En el apartado 3.1 se hablaba de los grandes enfoques para el análisis de texturas. El operador LBP, utilizando patrones uniformes, se puede entender

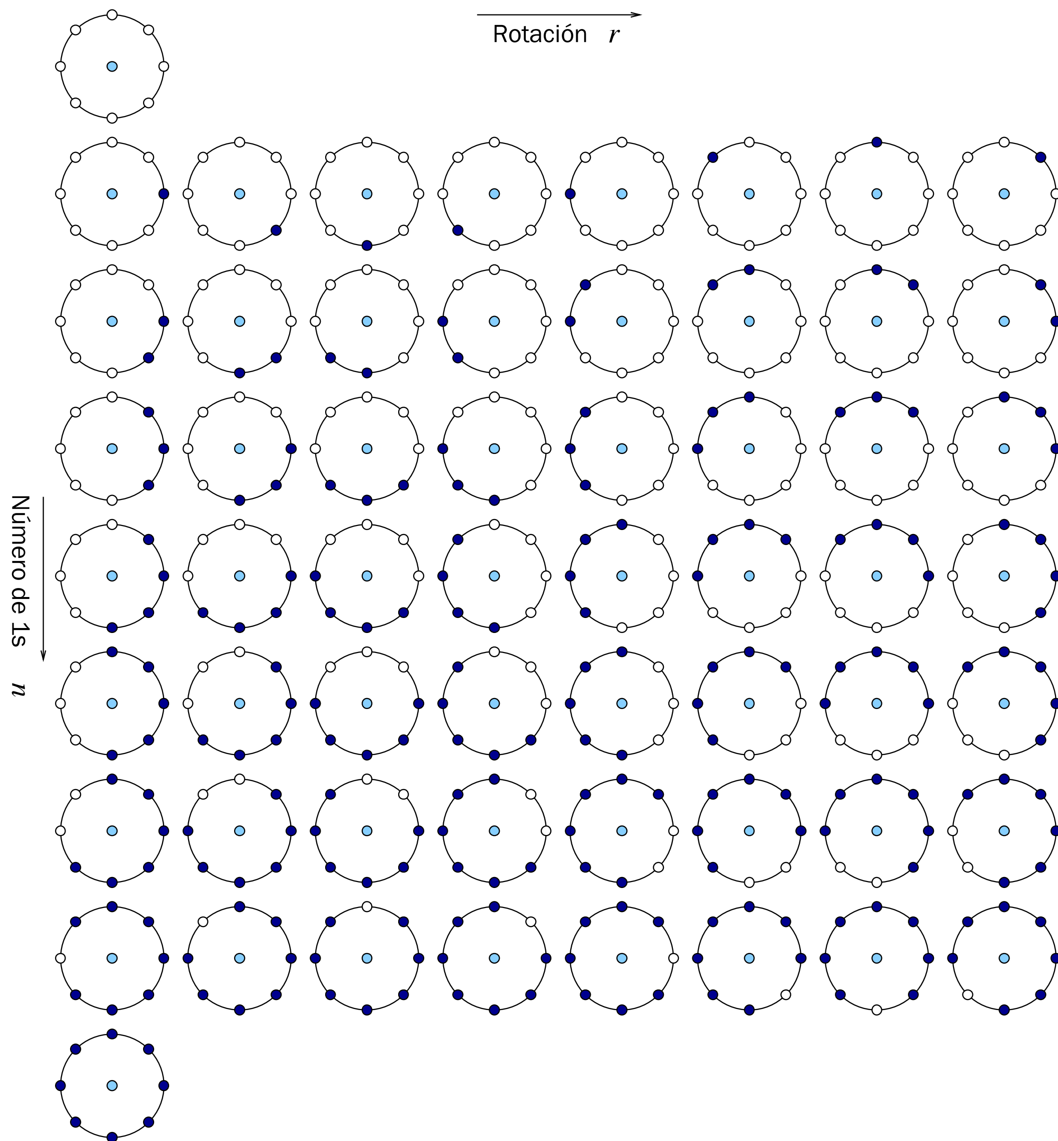


Figura 3.10: *Los 58 diferentes patrones uniformes en un vecindario (8,R)*

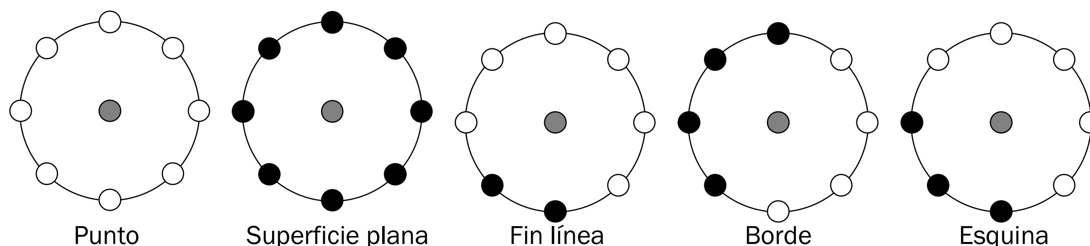


Figura 3.11: *Diferentes primitivas de textura detectadas por el operador LBP*

como un enfoque unificado entre el enfoque estadístico y estructural. Cada píxel está etiquetado con el código de la textura primitiva que mejor encaja con el vecindario, por lo tanto, cada código LBP puede entenderse como un *micro-texton*[3].

Se han detectados diferentes primitivas como puntos, superficies planas, finales de línea, bordes, esquinas, etc. En la figura 3.11 se muestran algunos ejemplos de primitivas utilizando el operador LBP con vecindad 8. Los puntos blancos representan ceros y los puntos negros unos.

La combinación de ambos enfoques surge del hecho de que las distribuciones de *micro-textons* pueden entenderse como reglas estadísticas de colocación. Por lo tanto, las distribuciones LBP tienen las dos propiedades de un análisis estructural: primitivas de textura y reglas de colocación. Por otra parte, esta distribución no es más que una estadística de las no linealidades de la imagen, convirtiendo el método en uno estadístico.

Por estas razones, las distribuciones LBP puede utilizarse para el reconocimiento de texturas de forma exitosa, mientras que los métodos estructurales y estadísticos han sido utilizados normalmente por separado.

Invarianza rotacional uniforme

La combinación de la invarianza rotacional y los patrones uniformes da lugar a la aparición de una nueva extensión del operador LBP, denominado $LBP_{P,R}^{riu2}$. El superíndice $riu2$ significa *rotation invariant uniform* (invarianza rotacional uniforme) y el 2 indica que el umbral para la medida de la uniformidad es 2. El número total de patrones posibles utilizando este operador es $P+2$, ($P+1$ patrones uniformes y 1 patrón no uniforme).

El valor $LBP_{P,R}^{riu2}$ se calcula contando el número de unos en el patrón binario. El resto de patrones no uniformes se agrupan y se etiquetan como "no uniformes".

[3]La palabra inglesa *texton* se refiere a micro estructuras fundamentales en las imágenes naturales, que se pueden considerar los «átomos» de la percepción visual humana.

$$LBP_{P,R}^{riu2} = \begin{cases} \sum_{p=0}^{P-1} s\left(g_p - g_c\right), & U\left(G_P\right) \leq 2 \\ P+1, & otro\ caso \end{cases} \tag{3.10}$$

En la práctica, el operador $LBP_{P,R}^{riu2}$ se implementa a través de una *look-up table*[4] (LUT) que convierte los códigos LBP "normales" en sus correspondientes LBP^{riu2}.

Una manera sencilla de solucionar los saltos de 45º del operador $LBP_{8,R}$ es incrementar el número de vecinos P. Aun así, este número no se puede aumentar arbitrariamente. El número de códigos LBP que hay que mapear a sus correspondientes invariantes rotacionales es 2^P. Esto implica que las *look-up table* aumentarán su tamaño de forma exponencial cada vez que aumente el número de vecinos.

Sería posible realizar un mapeo *on-line*, es decir, en tiempo de ejecución, sin utilizar una *look-up table*, pero a cambio de coste computacional.

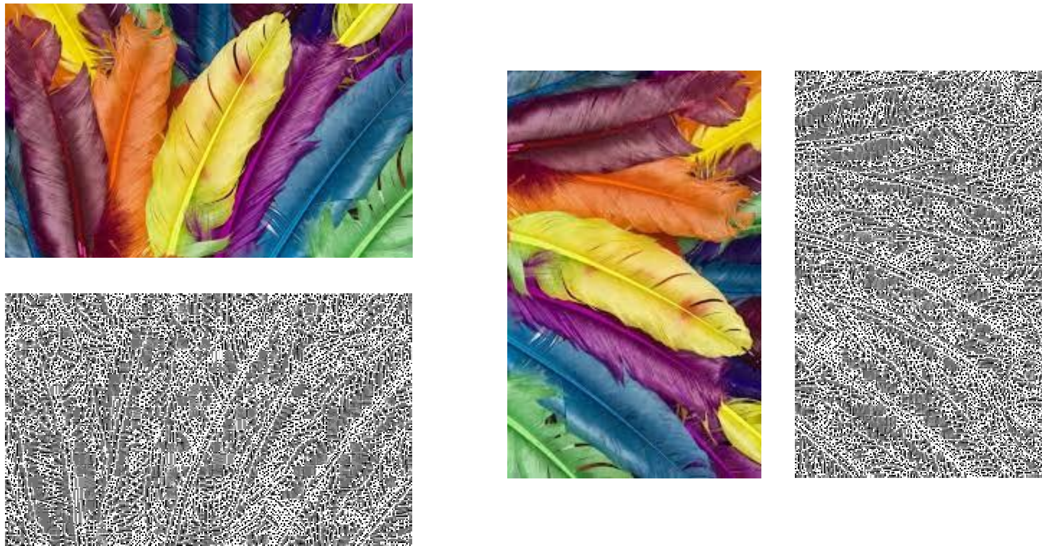
Otra pega de aumentar el valor P es que, por ejemplo, en un vecindario de radio 1, si el número de vecinos es mayor de 8 estaríamos muestreando información redundante.

Recuperando el ejemplo de la figura 3.9, en la figura 3.12 aplicaremos el operador $LBP_{8,1}^{riu2}$ a la imagen, y compararemos sus histogramas para determinar si realmente es efectivo. Dado que utilizamos patrones uniformes, el número de niveles de gris posibles varían entre 0 y 9. Los valores del 0 al 8 respresentan los 9 posibles patrones uniformes, mientras que el valor 9 agrupa todos los patrones no uniformes.

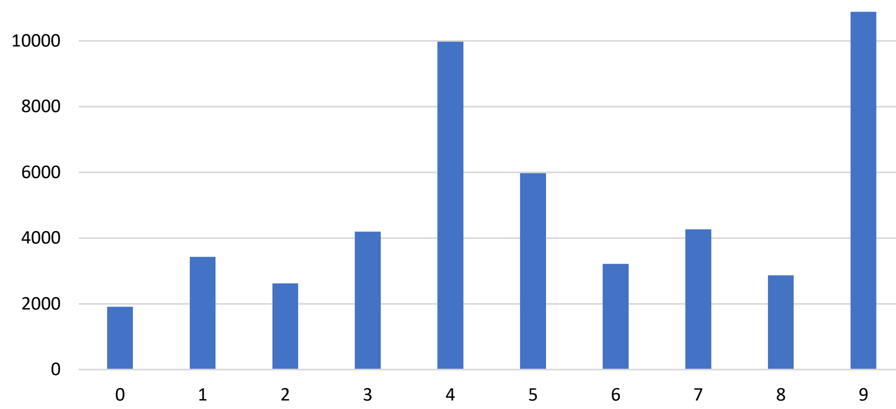
Analizando los resultados, observamos que aunque los histogramas no son exactamente iguales, por ejemplo, hay menos píxeles con valor 0 en la imagen normal que en la imagen rotada, en general la apariencia es mucho más similar que en el ejemplo anterior.

Como la imagen ULBP es muy oscura, se ha utilizado el programa Matlab para equiespaciar esos 10 niveles de grises en la escala 0-255 (Figura 3.13). Cabe mencionar que sólo se ha realizado para mejorar su visualización, ya que el histograma ha sido generado a partir de la imagen ULBP original.

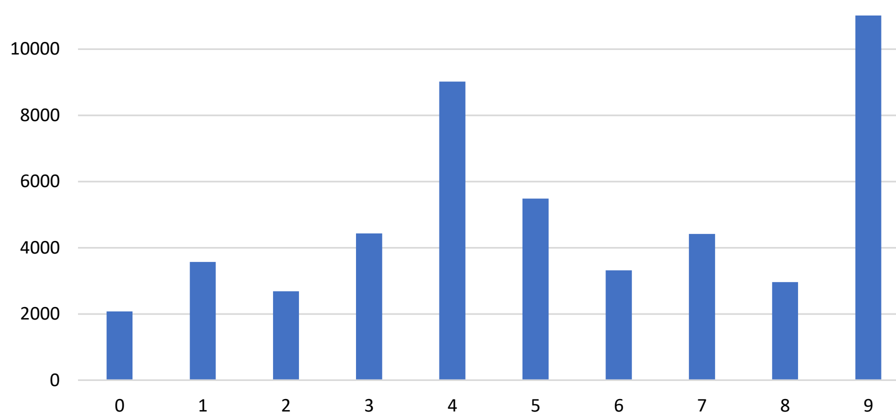
[4]Una *look-up table* es, en informática, una estructura de datos, normalmente un vector, que se usa para sustituir una rutina de computación con una simple indexación de los vectores. Resultan muy útiles a la hora de ahorrar tiempo de procesamiento.



(a) Imágenes originales y ULBP

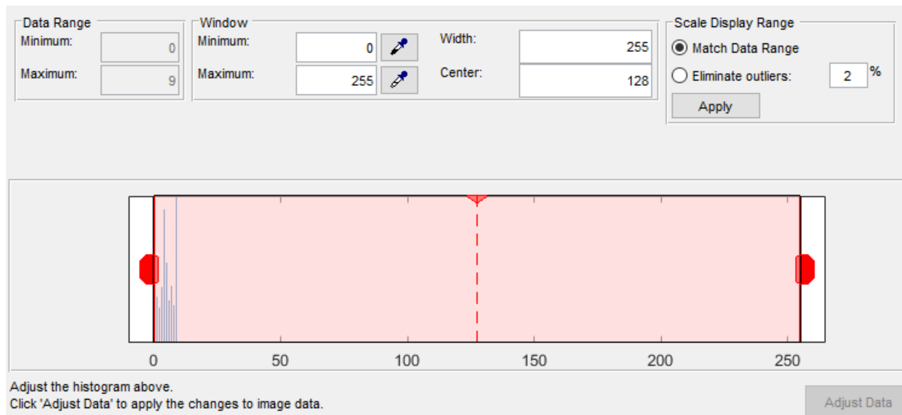


(b) Histograma ULBP horizontal

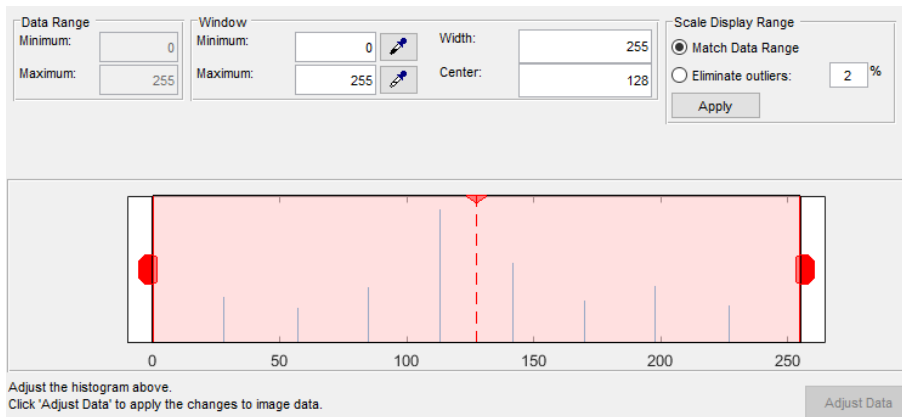


(c) Histograma ULBP vertical

Figura 3.12: *Comparación de histogramas de dos imágenes* $LBP_{8,1}^{riu2}$ *rotadas*



(a) Antes



(b) Después

Figura 3.13: *Histograma de la imagen ULBP antes y después de equiespaciar sus niveles de gris con Matlab*

3.2.4. Textura: contraste y patrones

El contraste es una propiedad que suele ser muy importante en un sistema de visión artificial, pero el operador LBP por sí mismo no es capaz de tener en cuenta la magnitud de las diferencias entre los niveles de grises. En muchas aplicaciones, especialmente en inspección industrial, la iluminación puede controlarse de forma precisa. En tal situación, un operador que no tiene en cuenta el contraste desaprovecha mucha información útil.

La textura puede entenderse como un fenómeno bidimensional caracterizado por dos propiedades ortogonales: patrón (estructura espacial) y contraste (la fuerza de esos patrones). La información del patrón es independiente del nivel de gris, mientras que el contraste no. Por otro lado, el contraste no se ve afectado por la rotación, mientras que los patrones sí. Estas dos medidas se complementan la una con la otra de una forma muy útil.

El contraste rotacionalmente invariante puede medirse en un vecindario circular simétrico a través de la varianza:

$$VAR_{P,R} = \frac{1}{P}\sum_{p=0}^{P-1}(g_p - \mu)^2 \tag{3.11}$$

Siendo μ la media calculada de la siguiente manera:

$$\mu = \frac{1}{P}\sum_{p=0}^{P-1} g_p \tag{3.12}$$

El operador $VAR_{P,R}$ es, por definición, invariante respecto a desplazamientos en la escala de grises. Dado que el contraste se mide de forma local, esta medida puede ignorar diferencias de iluminación dentro de la misma imagen, siempre que las diferencias absolutas entre niveles de grises no sean muy grandes.

La unión del operador LBP y el operador VAR da lugar a un operador rotacionalmente invariante en términos de detección texturas y la fuerza de estas. Se denota como $LBP^{riu2}_{P_1,R_1}/VAR_{P_2,R_2}$. Normalmente, los valores P y R se toman tal que $P_1 = P_2$ y $R_1 = R_2$.

3.2.5. LBP multi-resolución

La principal limitación del operador LBP es el pequeño área sobre el que trabaja. La información obtenida de un vecindario de tamaño 3x3 no puede reflejar las características dominantes a gran escala. Sin embargo, los códigos LBP adyacentes no son completamente independientes el uno del otro.

En la figura 3.14 se muestran 3 vecindarios adyacentes de 4 vecinos y radio arbitrario. Asumiendo que el primer bit (teniendo en cuenta que el bit inicial

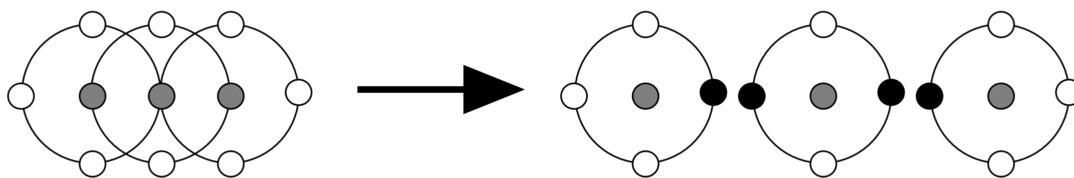


Figura 3.14: *Tres vecindarios $LBP_{4,R}$ y una combinación imposible de códigos LBP*

se encuentra en el eje positivo del eje x y se sigue un orden antihorario) del código de la izquierda es 0, el tercer bit del código de la derecha debe ser 1. Igualmente, el primer bit del código del centro y el tercer bit del código a la derecha, deben ser o diferentes o ambos iguales a 1.

La mitad derecha de la figura 3.14 muestra una combinación imposible de códigos LBP. Esto significa que cada código LBP limita los posibles códigos de los vecindarios adyacentes, haciendo que el área efectiva de un vecindario sea mayor a 3x3 píxeles.

Aun así, el operador no es demasiado robusto frente a cambios de textura locales, causados, por ejemplo, por cambiar el punto de vista o la direccionalidad de la iluminación. Es necesario, por lo tanto, un operador con un área de acción mayor.

Una de las estrategias para conseguir este objetivo se denomina "combinación de operadores".

Combinación de operadores

Una manera simple de aumentar el área de acción es combinar la información obtenida utilizando N operadores LBP con diferentes valores P y R. De esta manera, cada píxel de una imagen tendría N códigos LBP asociados. Se obtendrá información más precisa si se analiza la información combinada y no la de cada operador por separado.

Sin embargo, esta información combinada puede estar muy dispersa en una imagen de tamaño razonable.

Por ejemplo, la unión de las distribuciones de los operadores $LBP_{8,1}$, $LBP^{u2}_{16,3}$ y $LBP^{u2}_{24,5}$ contendría $256 \cdot 242 \cdot 554 = 34\,321\,408$ cubetas[5]. Por lo tanto, sólo las distribuciones marginales de los diferentes operadores serán consideradas, aunque la independencia estadística de los diferentes operadores para un mismo píxel no se puede garantizar.

La suma de las diferencias entre el patrón y una muestra se puede calcular como la suma de las diferencias entre las distribuciones marginales:

[5]Se denomina cubeta, o *bin* en inglés, a cada una de las divisiones del eje horizontal de un histograma.

$$L_N = -\sum_{n=1}^{N} L(P^n, M^n) \quad (3.13)$$

Siendo P^n la distribución marginal del patrón y M^n la de la muestra, extraída por el operador n-ésimo. También se puede utilizar la distancia chi-cuadrado para calcular la diferencia entre ambas distribuciones.

En la mayoría de las aplicaciones, esta manera de construir un operador LBP multi-escala arroja unos resultados muy buenos, aunque a costa de complejidad computacional.

3.2.6. LBP de colores opuestos

El operador LBP de colores opuestos (*Opponent Color Local Binary Pattern*, OCLBP) fue desarrollado como un operador conjunto color-textura para comparar características de las texturas tanto a color como en blanco y negro.

El uso del término "colores opuestos" es una convención adoptada por sus desarrolladores: todos los pares de canales de color se llaman "colores opuestos". Es decir, los colores opuestos son colores que se perciben como pares opuestos por los humanos, como por ejemplo el rojo y el verde, o el amarillo y el azul.

En LBP de colores opuestos, el operador LBP es aplicado a cada canal por separado. Además, también se aplica entre pares de canales diferentes, utilizando el píxel central de un canal y los vecinos del otro. En la figura 3.15 se muestran tres situaciones de códigos LBP en los cuales el píxel central corresponde al canal rojo.

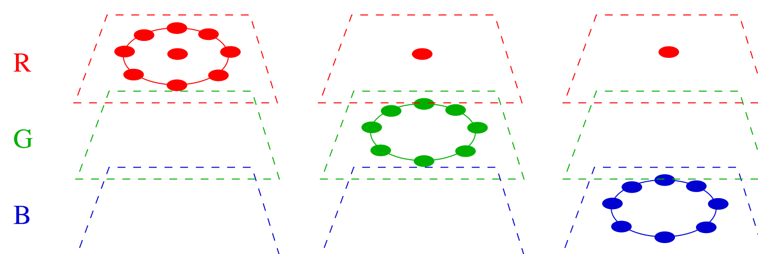


Figura 3.15: *Ejemplo OCLBP tomando como píxel central el del canal rojo*

En total, se obtienen 3 histogramas LBP de cada canal y 6 combinando diferentes canales. A continuación, los histogramas son concatenados en una misma distribución. Dado que los pares opuestos, como rojo-verde y verde-rojo, contienen información redundante, realizar una única comparación será suficiente para el análisis. Por lo tanto, reducimos en 3 unidades el número de histogramas a generar. El descriptor de textura es, por lo tanto, 6 veces más grande que la versión en blanco y negro, lo que lo hace poco apropiado para algunas aplicaciones.

Aunque el operador OCLBP funciona bien en comparación con otras medidas de la textura a color, no es siempre la mejor opción juntar color y textura. Se ha demostrado, utilizando un gran número de patrones naturales, que la unión entre descriptores de textura a color y las características combinadas de colores y textura funcionan peor que utilizando únicamente texturas a color o en blanco y negro.

La utilidad obtenida utilizando un operador combinado como el UCLBP es mínima.

3.2.7. Operador mLBP

En la ecuación 3.5 se definía el operador LBP genérico. Existe una variante que en vez de utilizar el píxel central para la comparación, utiliza el nivel de gris medio del vecindario. Esto aporta robustez al operador, ya que si hay ruido en la imagen, un valor erróneo en un píxel del vecindario desembocaría en un código LBP que no describiría correctamente la textura. En cambio, utilizando la media, se amortigua la influencia del ruido.

Definimos la función media m de la siguiente manera:

$$m = \frac{1}{P} \sum_{p=0}^{P-1} g_p \tag{3.14}$$

El operador $mLBP$ se puede expresar como:

$$mLBP_{P,R}\left(x_c, y_c\right) = \sum_{p=0}^{P-1} s\left(g_p - m\right) 2^p \tag{3.15}$$

Siendo s la misma función umbral que en la ecuación 3.4.

Al igual que utilizando el operador LBP genérico, obtenemos 256 códigos mLBP diferentes y se pueden convertir a patrones uniformes.

El operador mLBP también se puede implementar considerando el píxel central como un vecino más. De esta forma se compararían los P vecinos más el píxel central con el valor de la media del vecindario. El problema de esta última variante es que el número de códigos mLBP aumentaría hasta 512, incrementando el coste computacional. Es por ello que a la hora de implementar el operador mLBP en el programa se utilizará el descrito en la ecuación 3.15.

3.2.8. Aplicaciones del operador LBP

Dado que el operador LBP es muy simple, resulta adecuado para aplicaciones en las cuales es necesaria la extracción rápida de características. A continuación se muestran diferentes ejemplos de aplicación.

Inspección visual industrial

Una de las primeras aplicaciones industriales del operador LBP fue la inspección de superficies metálicas. Más adelante, se introdujo un sistema por el cual un subconjunto de códigos LBP se utilizaban para caracterizar defectos en válvulas de motores de automóviles.

Otra aplicación industrial utilizaba el operador para el análisis de detergentes.

Una de las aplicaciones más comúnmente reportadas es la inspección de madera, utilizando los canales a color. Por último, la inspección de papel es una aplicación en la que el operador LBP funciona satisfactoriamente.

Recuperación de imágenes

Algunos investigadores han utilizado los LBP como parte de sus sistemas de recuperación de imágenes. Se presentó un sistema llamado *Imagespace*, que utilizaba características de texturas basadas en distribuciones como modelos de textura. Entre esas características estaban los LBP y los *Trigrams*, que son una variante del operador LBP utilizando los bordes de las imágenes.

También se ha aplicado el operador LBP para la recuperación de imágenes comprimidas JPEG.

Análisis de escenas

La interpretación de escenas naturales ha sido uno de los objetivos principales de la visión artificial desde sus primeros días. Sin embargo, este objetivo resulta muy complejo y únicamente se han obtenido buenos resultados en muy pocas aplicaciones.

Detección y reconocimiento facial

Para la detección de rostros se pueden utilizar diferentes estrategias. Por ejemplo, se puede centrar en detectar los ojos de las personas para estimar la posición de la cara, teniendo en cuenta que el ojo es una región de la imagen con una alta variabilidad con respecto a las zonas adyacentes del mismo.

El operador LBP también se puede utilizar para reconocimiento facial. Igualmente se pueden utilizar diferentes técnicas, aunque se ha demostrado que dividir el rostro en subregiones para posteriormente describirlas usando LBP incrementaba notablemente la precisión de los sistemas de clasificación frente a la descripción del rostro por completo.

Capítulo 4

Desarrollo del sistema de detección de gasas quirúrgicas

4.1. Introducción

El sistema de detección de gasas quirúrgicas ha sido desarrollado en C++, utilizando la librería de procesamiento de imágenes OpenCV. El IDE[1] utilizado ha sido Code::Blocks, corriendo sobre la distribución de Linux, Ubuntu.

En el capítulo introductorio se fijaron los objetivos de este TFG. A continuación, se habló de la imagen digital y de la visión artificial, que es la base a partir de la cual se puede desarrollar este sistema. Por último, en el capítulo anterior, se abarcaba el tema del análisis de texturas, y más en concreto, el del operador LBP. Una vez creado este contexto, se procede a la explicación de los programas desarrollados y sus diferentes variantes, dejando para el siguiente capítulo el análisis de resultados.

En la figura 4.1 se muestra, a grandes rasgos, los pasos que se siguen para determinar si en una imagen de entrada hay gasas o no.

Cada uno de estos pasos será explicado detalladamente en la sección 4.4.

A modo de ejemplo, en la figura 4.2 se muestra la ejecución del programa, mostrando los pasos de la figura 4.1 de forma gráfica.

Al igual que en la subsección 3.2.3, dónde se hablaba de invarianza rotacional y patrones uniformes, la imagen ULBP ha sido modificada para una mejor visualización. En la última imagen, el número que aparece en cada tesela[2] corresponde al valor de la comparación de histogramas.

[1]IDE: *Integrated Development Enviorment*, en español, entorno de desarrollo integrado. Es una aplicación informática que proporciona servicios integrales para facilitar el desarrollo de software a la hora de programar.

[2]Llamamos tesela a cada uno de los fragmentos en los que dividimos la imagen para su análisis. En el código fuente se utiliza la palabra inglesa *tile*.

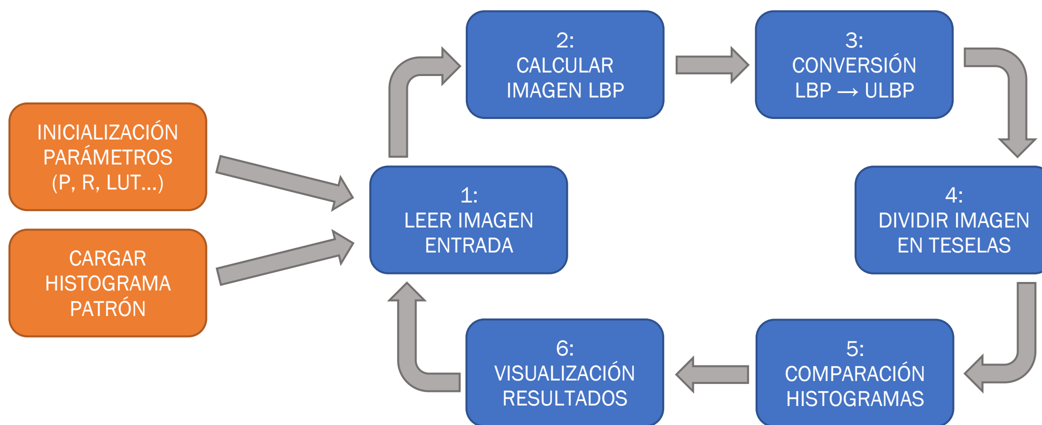
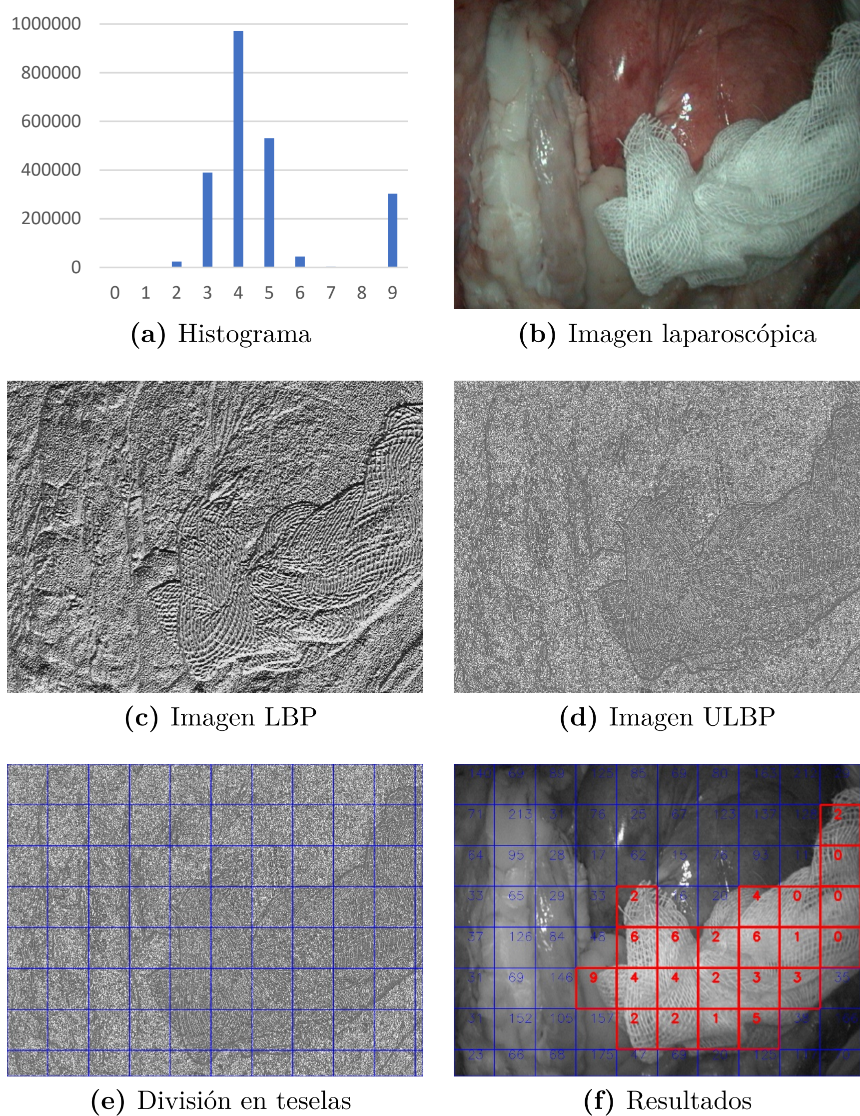


Figura 4.1: *Simplificación de los pasos dados por el sistema de detección de gasas quirúrgicas*



(a) Histograma **(b)** Imagen laparoscópica

(c) Imagen LBP **(d)** Imagen ULBP

(e) División en teselas **(f)** Resultados

Figura 4.2: *Ejemplo ejecución del programa básico*

4.1.1. Variantes del sistema de detección de gasas quirúrgicas

Aunque el principio de funcionamiento mostrado en la figura 4.1 se mantiene en todas las versiones, hay pasos que cambian y otros que se añaden. Cabe mencionar que en cualquier versión se pueden elegir hasta 32 vecinos y un radio arbitrario, siempre que el vecindario no sea mayor que el tamaño de la imagen.

Se han desarrollado 4 versiones diferentes:

1. Utilizando únicamente el operador LBP[3] (o mLBP).

2. Utilizando LBP (o mLBP) y la varianza por separado.

3. Utilizando LBP (o mLBP) y la varianza de forma conjunta (método 1).

4. Utilizando LBP (o mLBP) y la varianza de forma conjunta (método 2).

La principal razón por la que se han desarrollado tres versiones con la varianza es la siguiente: en la variante 2, se dispone de dos histogramas y se obtienen dos valores de comparación. Por lo tanto, hay dos criterios de aceptación posibles: *and* (ambos operadores tienen que coincidir en la decisión) u *or* (solo es necesario que uno de ellos considere como gasa un tesela para ser aceptada como tal). Para no tener que lidiar con dos valores de comparación diferentes, la variante 3 y 4 generan un histograma que contiene información tanto de la varianza como LBP utilizando diferentes métodos, y se obtendrá un único valor de comparación.

El programa principal viene acompañado por dos programas secundarios. El primero es el "generador de patrón", encargado de crear el histograma LBP y de la varianza, que posteriormente se utilizará para las comparaciones. Este es creado a partir de todas las imágenes contenidas en una carpeta llamada "*patternImg*". Comparte gran parte del código con el programa principal, por lo tanto, habrá diferencias entre las 4 variantes. Únicamente es necesario ejecutarlo la primera vez y cuando cambien los parámetros P y/o R, ya que el patrón cambiará. Deberá ser ejecutado antes que el programa principal, ya que si no, no tendrá ningún patrón con el que realizar las comparaciones.

El segundo es el "generador de *look-up tables*". Genera un archivo que posteriormente leerá y cargará tanto el programa principal como el "generador de patrón". La terminología del fichero será *"ULBP_LUT_P.dat"*, siendo P el número de vecinos. La LUT deberá haber sido generada antes de ejecutar cualquiera de los dos programas. Sólo es necesaria una sola ejecución por cada valor P.

[3] A partir de ahora se hablará de operador LBP de forma genérica, aunque su implementación utiliza patrones uniformes. Su notación completa sería $LBP_{P,R}^{riu2}$ y $mLBP_{P,R}^{riu2}$.

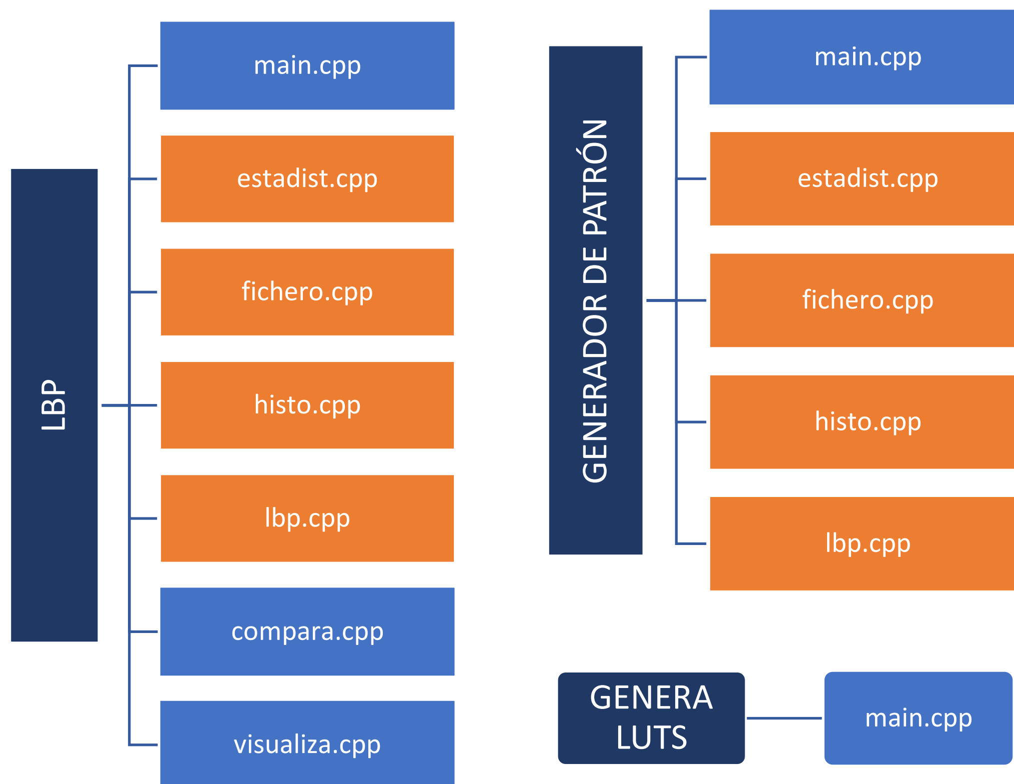


Figura 4.3: *Jerarquía de archivos*

4.1.2. Jerarquía de archivos

Para facilitar la programación y la comprensión del código, cada uno de los programas ha sido dividido en ficheros. En la figura 4.3 se muestra de forma gráfica los archivos de los que están compuestos los programas.

Cada uno de los ficheros *main.cpp* son únicos para cada programa y son los encargados de llamar a todas la funciones correspondientes. Los ficheros que están en un cuadro naranja (*estadist.cpp*, *fichero.cpp*, *histo.cpp* y *lbp.cpp*) son compartidos por los programas LBP y el generador de patrón. El hecho de que sean archivos compartidos permite, además de evitar código duplicado, garantizar que el patrón es generado siguiendo exactamente los mismos pasos que se siguen al analizar las imágenes laparoscópicas en el programa principal. Por último, el programa LBP depende de dos archivos más, *compara.cpp* y *visualiza.cpp*.

Todos los ficheros que no son *main.cpp* vienen acompañados por una cabecera ".h" con el mismo nombre que el archivo ".cpp".

A continuación se describen brevemente las funciones que tienen asociadas cada uno de los archivos:

- *estadist.cpp*: agrupa todas las funciones relacionadas con el cálculo de la varianza y de la media de las imágenes.

- *fichero.cpp*: encargado de la lectura de archivos externos, como los datos patrón, las LUT o las imágenes laparoscópicas de entrada.
- *histo.cpp*: permite calcular el histograma de una imagen y también dibujarlo.
- *lbp.cpp*: contiene las funciones que calculan imágenes LBP y mLBP para un radio y número de vecinos arbitrario, además de realizar la conversión en patrones uniformes (ULBP).
- *compara.cpp*: su función es comparar los histogramas de cada tesela con el histograma patrón.
- *visualiza.cpp*: encargado de mostrar los resultados de la comparación sobre la imagen original.

4.2. Generador de ***look-up tables***

Este programa es el encargado de generar las *look-up tables* que convertirán los códigos LBP en ULBP. Antes de comenzar con la explicación del programa, se va a explicar su utilidad y cómo se utilizan.

Un patrón uniforme es aquel cuyo código LBP binario tiene un máximo de 2 transiciones de bits de 1 a 0 o viceversa. Si tomamos como ejemplo un vecindario $P = 8$, los siguientes patrones darían lugar al mismo patrón uniforme:

100000000, 010000000, 00100000, 00010000, 00001000, 00000100, 00000010 y 00000001, asignándole la etiqueta "1" (la etiqueta "0" corresponde al patrón 00000000). Los número binarios de estos códigos LBP son, respectivamente: 128, 64, 32, 16, 8, 4, 2 y 1.

En la figura 4.4 se muestra el vector que contiene la LUT. La primera fila representa el contenido del vector, y la segunda el índice asociado a esa posición. Para convertir un código LBP en ULBP, será tan simple como acceder a la posición del vector correspondiente al valor decimal del código LBP. De esta forma, nos evitamos tener que determinar si un patrón es uniforme o no, y qué etiqueta le corresponde cada vez que se calcula una imagen ULBP, sino que únicamente se determina una vez al generar la LUT.

0	1	1	...	1	...	1	...	1	...	1	...	1	...	1	...
0	1	2		4		8		16		32		64		128	

Figura 4.4: *Representación del vector que contiene la LUT para* $P = 8$

Una vez comprendido el principio de funcionamiento, se procede a explicar como son generadas en el programa. El código fuente se puede encontrar en el anexo D.

La macrodefinición *NUM_BITS* permite elegir el número de vecinos. El tamaño de la LUT será 2^P, siendo P el número de vecinos (*NUM_BITS*). A continuación, inicializamos la LUT a cero.

La forma de proceder es la siguiente: a partir del patrón compuesto únicamente por unos, desplazamos los bits a la derecha *NUM_BITS* veces, y almacenamos los valores para cada desplazamiento. De esta forma obtenemos patrones con 2 transiciones como máximo.

Para $P = 8$ la secuencia sería la siguiente: $11111111 \rightarrow 01111111 \rightarrow 00111111 \rightarrow \ldots \rightarrow 00000001$. Con cada uno de estos números, desplazamos los bits de forma cíclica hacia la derecha, de forma que obtenemos todas las combinaciones.

En la figura 4.5 se muestra una ejemplo de ejecución hasta este punto. La "N" representa el número inicial a partir del cual se realizan los 8 desplazamientos cíclicos, y la "d" el número de desplazamientos cíclicos. Por ejemplo, para $N = 127$, el primer desplazamiento a la derecha sería: $01111111 \rightarrow 10111111 = 191$.

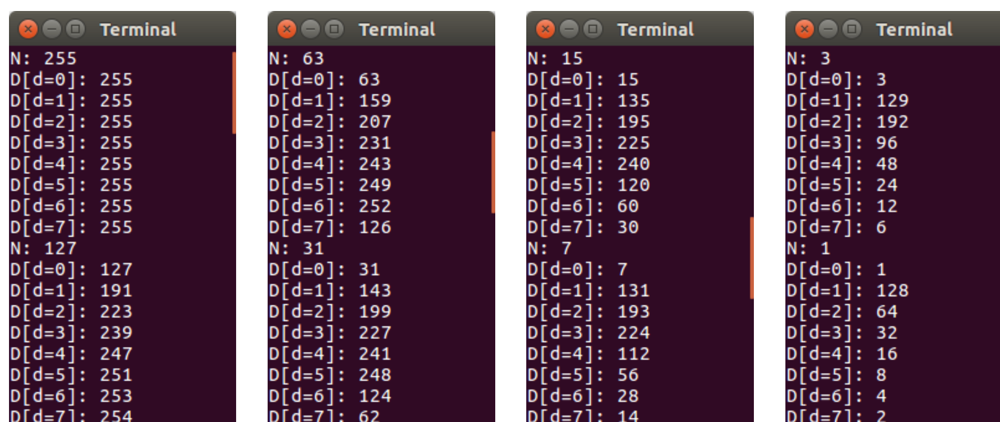


Figura 4.5: *Ejecución del programa generador de LUTs*

A continuación, guardamos en cada uno de los índices de la LUT el valor mínimo de su grupo. Por ejemplo, para $N = 127$, se almacenará en los índices $127, 191, 223, \ldots, 254$ el valor 127, ya que es el mínimo de todos.

Seguidamente, actualizamos estas etiquetas temporales por las definitivas, comenzando por el número 2 (ya que la etiqueta temporal "1", es igual que la definitiva ULBP). Finalmente, recorremos el vector en busca de ceros, que corresponden a los patrones no uniformes (esta es la razón por la cual se inicializó el vector a cero) y se cambian por el siguiente código ULBP no utilizado, que en el caso de $P = 8$, es 9.

Por último, exportamos la LUT a un archivo ".dat" para que posteriormente lo puedan utilizar el resto de programas. Los nombres de estos archivos serán "*ULBP_LUT_P.dat*", siendo P el número de vecinos/bits.

En la sección D.2 del anexo se adjunta la LUT para 8 vecinos.

4.3. Generador de patrón

El programa "generador de patrón" es el encargado de generar el histograma patrón y demás información adicional a partir de un conjunto de imágenes suministradas.

Como se explicó en la subsección 4.1.1, hay 4 variantes del programa principal que afectan también a la generación de patrones. Aun así, como la versión "LBP" y "LBP con varianza por separado" únicamente difieren en el hecho de que el primero no necesita los cálculos de varianza, utilizarán el mismo patrón y la versión "LBP" omitirá la información que no necesita. En cambio, en la dos versiones de "LBP y varianza combinado", los cambios son mayores y sí es necesario cambiar más partes del código.

El código fuente de las 3 versiones se encuentra disponible en los apartados B.1, B.2 y B.3 del anexo. Los archivos comunes están en el anexo C.

Antes de comenzar la función *main()*[4], hay dos macrodefiniciones que cambiarán los parámetros del histograma de la varianza (excepto en "LBP y varianza combinado, método 2") :

- *NUM_BINS_VAR*: es el número de cubetas que tendrá el histograma cuantificado de la varianza. El valor por defecto es 16 y debe ser potencia de 2.

- *VAR_MAX*: el número de cubetas del histograma de varianzas no cuantificado. El valor por defecto es 600.

El programa comienza estableciendo los parámetros del vecindario, P y R, y también el factor de escala. Si se descomenta la macrodefinición *PARAM_ MANUALES*, el programa pedirá de forma interactiva los parámetros P y R, si no, los valores por defecto son $P = 8$, $R = 1$ y factor de escala 1.

A continuación, se carga la LUT para la conversión LBP a ULBP. Para ello se llama a la función *leeFicheroLUT()* (*fichero.cpp*), y se le pasa el número de vecinos P. Previamente ha debido de generarse la LUT para el número de vecinos seleccionado, ya que, si no, dará un mensaje de error y se cerrará el programa.

El siguiente paso es leer la primera imagen patrón. El programá buscará en la carpeta "*../patternImg*" el archivo "*patron1.png*", ya que al menos debe haber un archivo. Más adelante continuará buscando imágenes patrón con el formato

[4]La función *main*, en español, función principal, es una función que siempre debe estar presente en el lenguaje de programación C/C++. Es la primera función que se ejecuta de un programa.

"*patronX.png*" hasta que no existan más. Si no existe el archivo "*patron1.png*", el programá se cerrará sacando por pantalla un mensaje de error.

Las imágenes patrón que se utilizarán se pueden dividir en 2 grupos: gasas sin sangre y gasas con sangre. En la figura 4.6 se muestran las gasas sin sangre y en la figura 4.7 las gasas con sangre.

A partir de ahora, la ejecución del programa es diferente según la variante que se esté utilizando.

4.3.1. "LBP" y "LBP y varianza por separado"

Si el factor de escala elegido por el usuario ha sido menor que 1, se reducirá la imagen. Antes de ello, se aplica un filtro *gaussiano*. El tamaño del operador es de 3x3 y un valor *sigma* de 1 tanto en la dirección x como y. Esto ayuda a eliminar ruido de la imagen. En la figura 4.8 se muestra la imagen original y después dos imágenes sobre las que se ha aplicado el filtro *gaussiano* utilizando diferentes operadores.

Como se puede observar, la diferencia es muy sutil y prácticamente inapreciable, pero precisamente ese es el objetivo, no modificar demasiado la imagen original a la vez que nos beneficiamos de sus efectos. Para ilustrar el funcionamiento del filtro, la imagen de la derecha muestra el resultado tras someter a la imagen al mismo filtro pero con un tamaño de operador mayor (9x9 y valor *sigma* en ambas direcciones de 3).

En la sección 3.2.5 se hablaba de LBP multi-resolución, cuyo objetivo era combinar la información de múltiples vecindarios para que la zona de acción fuera mayor. Reduciendo el tamaño de la imagen, conseguimos aumentar el área de acción manteniendo el mismo tamaño de vecindario, y por lo tanto, sin incrementar el coste computacional.

A continuación, calculamos la varianza de una imagen y su histograma. Si el vecindario es $P = 8/R = 1$, se llamará a la función *varianza()*, en cambio, para un vecindario arbitrario, la función a la que se llamará será *VARLBP()*. Ambas funciones están en el archivo *estadist.cpp*.

Aún así, los pasos a dar son equivalentes. La fórmula utilizada para calcular la varianza es la siguiente:

$$Var\left[X\right] = E\left[X^2\right] - \left(E\left[X\right]\right)^2 \tag{4.1}$$

$E\left[X\right]$ representa la media de la imagen para un vecindario dado.

Utilizado la función *varianza()*, esta llama a las funciones auxiliares *media()*, que calcula la media de un vecindario arbitrario, y *alCuadrado()*, que eleva cada píxel de una imagen dada al cuadrado. Ambas funciones se encuentran en el mismo archivo que *varianza()*. Si el vecindario no es el estándar, la

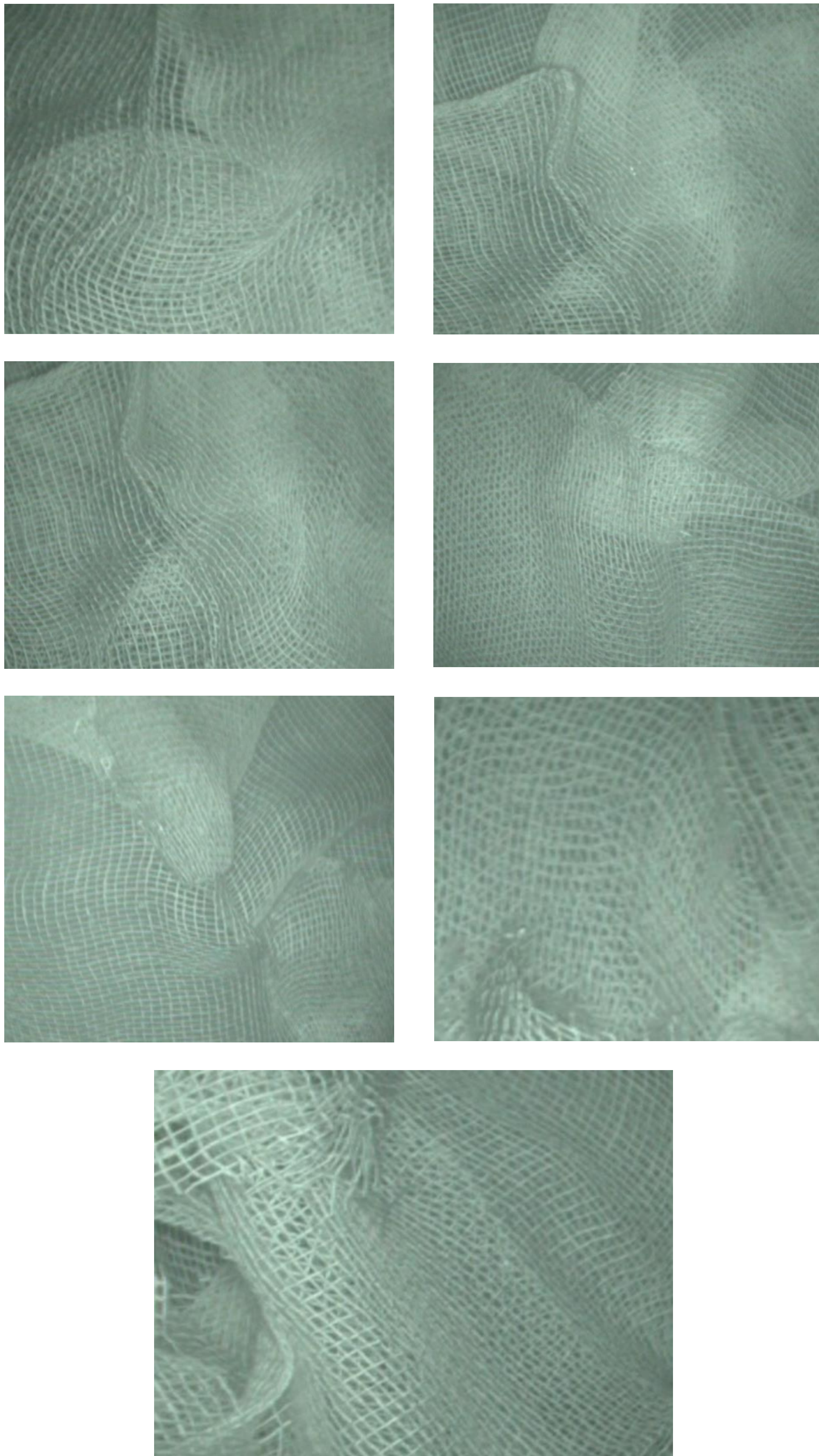


Figura 4.6: *Gasas patrón sin sangre*

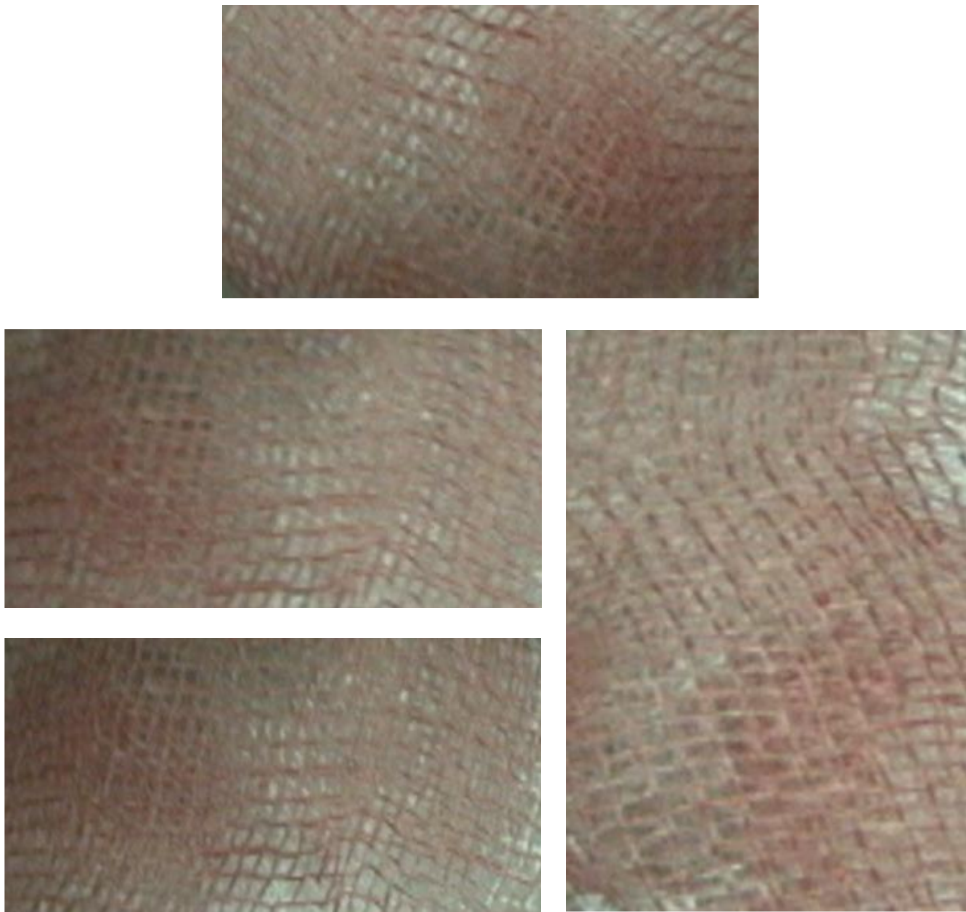
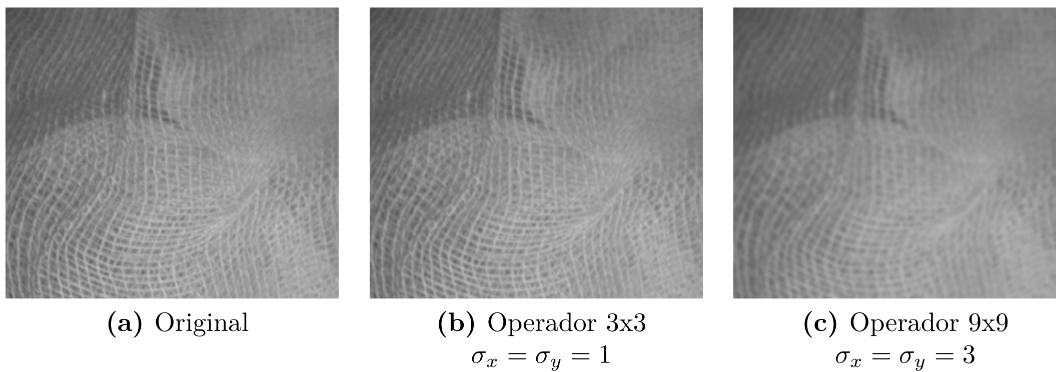


Figura 4.7: *Gasas patrón con sangre*



(a) Original

(b) Operador 3x3 $\sigma_x = \sigma_y = 1$

(c) Operador 9x9 $\sigma_x = \sigma_y = 3$

Figura 4.8: *Imágenes patrón antes y después de aplicar un filtro gaussiano*

función *VARLBP()* realiza todos los cálculos necesarios, incluidas interpolaciones y la imagen de medias, dentro de la misma función.

En ambos casos, la imagen resultado tiene un $2 \cdot R$ filas y columnas menos, al igual que si se utilizara un operador LBP.

A continuación, se calcula el histograma de la imagen de varianzas. Para ello se llama a la función *calcHisto()* (*histo.cpp*). Esta función recibe la imagen de entrada, la imagen en la que se quiere almacenar el histograma, el número de cubetas, el valor mínimo y el valor máximo. Es necesario establecer un nivel máximo, ya que los valores de varianza no están acotados como lo puede estar un código LBP.

Para el histograma de la varianza, el número de cubetas y el valor máximo será *VAR_MAX* (600 por defecto) y el valor mínimo 0.

La figura 4.9 muestra el histograma de la varianza de la primera imagen patrón.

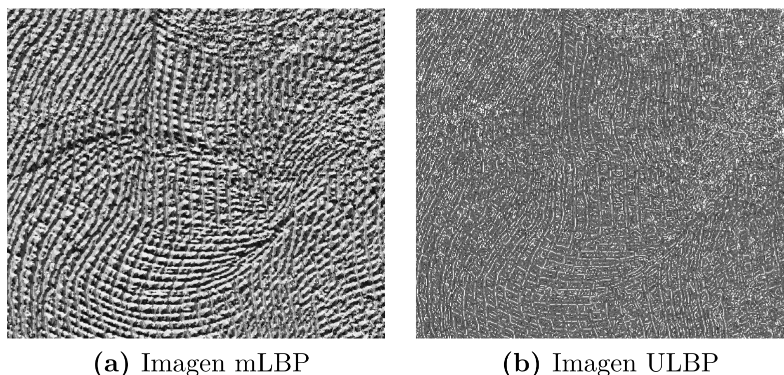
Figura 4.9: *Histograma de la imagen de varianzas del patrón "patron1.png"*

Se puede observar cómo la parte izquierda del histograma está mucho más poblada que la derecha. Para un número total de 93 802 píxeles, 92 450 de ellos se encuentran en el rango $[0, 150]$, lo que supone el 98.6 % de los píxeles totales. El resultado es un histograma muy poco equilibrado, y por ello, más adelante se realizará una cuantificación que permita equilibrar más los grupos.

El siguiente paso es calcular la imagen LBP del patrón. Aquí puede elegirse usar la versión LBP o mLBP, comentando la línea de código no deseada. Tanto la función *LBP()* como *mLBP()* están presentes en el archivo *lbp.cpp*. Si se utiliza la versión mLBP, se deberá pasar la imagen de medias a la función.

En ambos casos, devuelve una imagen con $2 \cdot R$ filas y columnas menos. A continuación, llamamos a la función *LBP2ULBP()* para realizar la conversión en LBP uniforme. Esta función también se encuentra en el archivo *lbp.cpp*. La

figura 4.10 muestra la imagen mLBP y ULBP de la primera imagen patrón, utilizando un vecindario $P = 8/R = 1$.



(a) Imagen mLBP **(b)** Imagen ULBP

Figura 4.10: *Imagen mLBP y ULBP de la primera imagen patrón para un vecindario $P = 8/R = 1$*

Esta función utiliza plantillas para poder utilizar diferentes formatos de imagen. Por ejemplo, al calcular la imagen LBP con un vecindario $P = 8/R = 1$, el tipo de imagen es *CV_8UC1*. El 8 indica el número de bits utilizado para almacenar cada píxel, la *U* se refiere a *unsigned*, es decir, sin signo, lo que significa que utiliza una variable sin signo. Por último, *C1* indica que se utiliza un sólo canal.

Utilizando una variable de 8 bits sin signo (*unsigned char*), el rango de valores que puede tomar es de 0 a 255, por lo que el tamaño máximo de vecindario es $P = 8$.

En cambio, para vecindarios mayores, se utiliza el tipo de imagen *CV_32SC1* (32 bits, con signo y un sólo canal), en la que se puede trabajar hasta vecindarios de tamaño $P = 32$.

El último paso es el cálculo del histograma de la imagen ULBP. Los parámetros del histograma serán:

- Número de cubetas: $P + 2$
- Nivel de gris mínimo: 0
- Nivel de gris máximo: $P + 1$

El valor $P + 2$ es el número de patrones uniformes más la etiqueta que engloba los patrones no uniformes. La figura 4.11 muestra el histograma de la imagen ULBP del primer patrón.

Este proceso se realiza hasta encontrar la última imagen patrón en la carpeta "*patternImg*".

Ahora hay que combinar la información de todos los patrones. Esto es una tarea bastante sencilla, ya que únicamente tenemos que sumar todos los

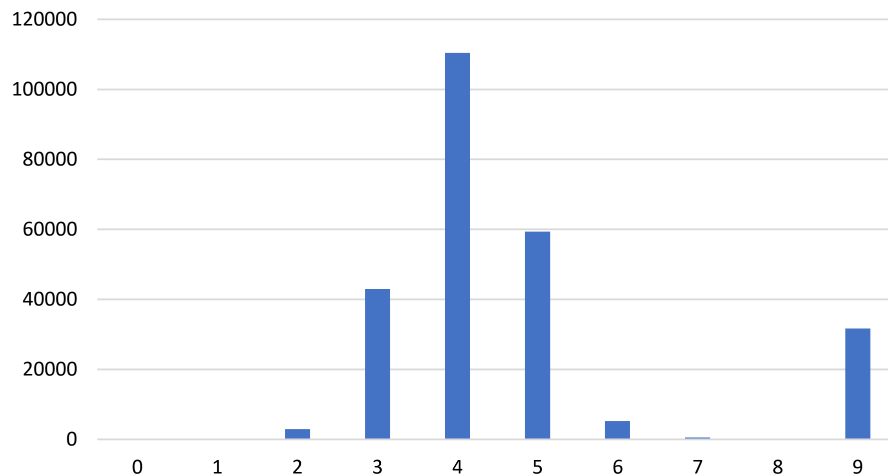


Figura 4.11: *Histograma de la imagen ULBP del primer patrón*

histogramas de cada tipo. Obtendremos, por lo tanto, dos histogramas. El de la varianza y el de las imágenes ULBP, que son una representación de la unión de los diferentes patrones.

En la figura 4.12 se muestra la suma de ambos histogramas.

Una vez tenemos los dos histogramas finales, hay que llevar a cabo las últimas operaciones sobre los histogramas. La primera de ellas es cuantificar la varianza. Como se ha dicho anteriormente, el histograma de la varianza está muy poco equilibrado y la población está concentrada en la parte izquierda, mientras que la parte central y derecha está muy despoblada.

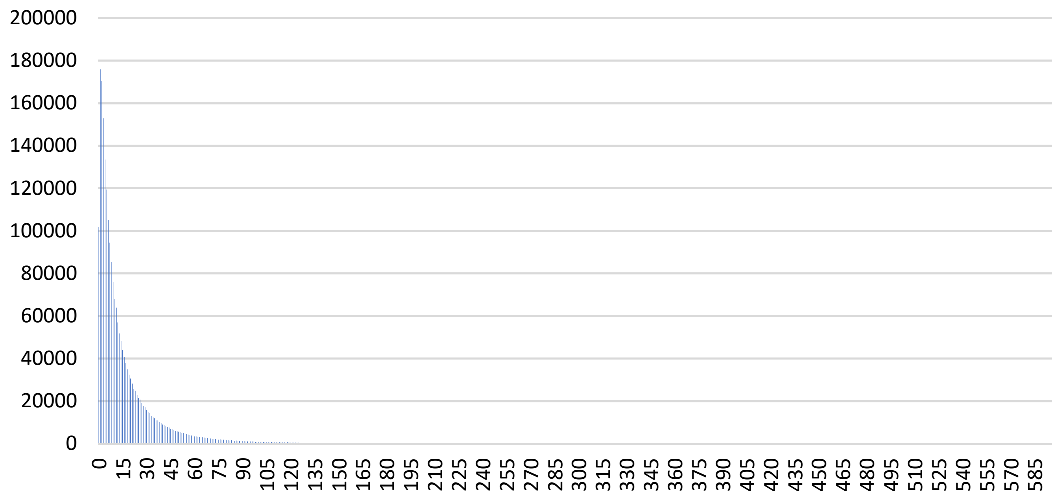
La cuantificación consiste en generar un nuevo histograma a partir del original cuyas cubetas estén lo más equilibradas posible. También se reduce de forma significativa el número de estas, pasando de 600 (*VAR_MAX*) a 16 (*NUM_BINS_VAR*). El hecho de reducir el número de cubetas nos permite que el coste computacional de comparar los histogramas sea mucho menor. Para ello, llamamos a la función *calculaCuantificacionVar()* que devuelve un vector con los puntos de corte del histograma original. Pasamos este vector a la función *cuantifHistoVar()* y obtenemos el nuevo histograma a 16 niveles cuantificado. Ambas funciones están definidos en el archivo principal *main.cpp*.

En este ejemplo, el número total de píxeles es 2 269 806, que tenemos que reorganizar en 16 cubetas, es decir, 141 862 píxeles por cubeta. En cambio, en la primera cubeta del histograma original tenemos 101 850 píxeles, y en la segunda 175 828, lo que sumados hace un total de 277 678 píxeles, doblando prácticamente el número total que debería haber en cada cubeta.

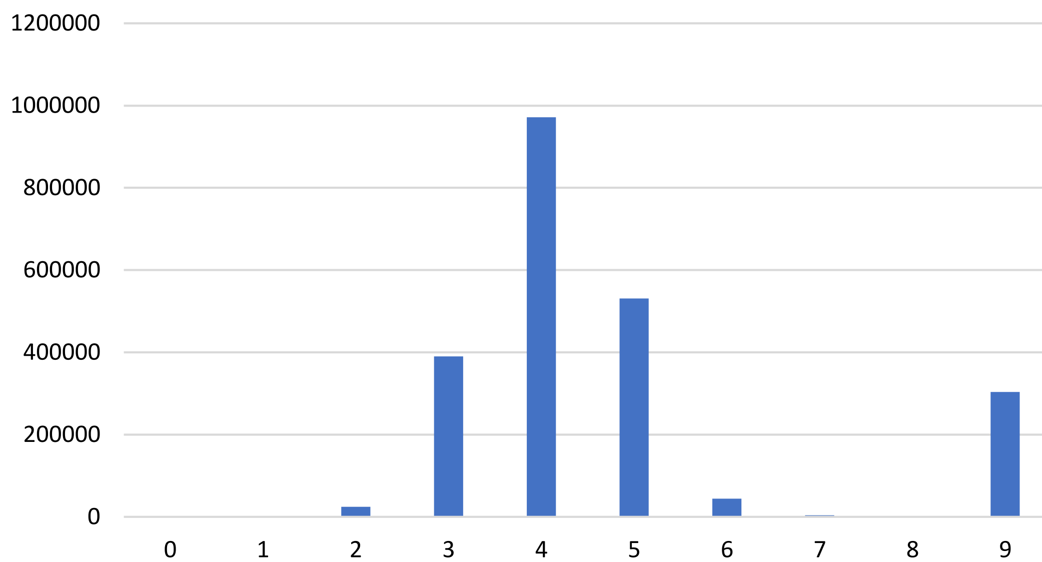
Esto es un problema que no tiene una fácil solución, dada la gran acumulación de píxeles con valores de varianza baja.

El histograma cuantificado se muestra en la figura 4.13.

Como la concentración de puntos en la zona derecha del histograma es tan



(a) Histograma varianza



(b) Histograma ULBP

Figura 4.12: *Suma de los histogramas la varianza y ULBP de todas las imágenes patrón*

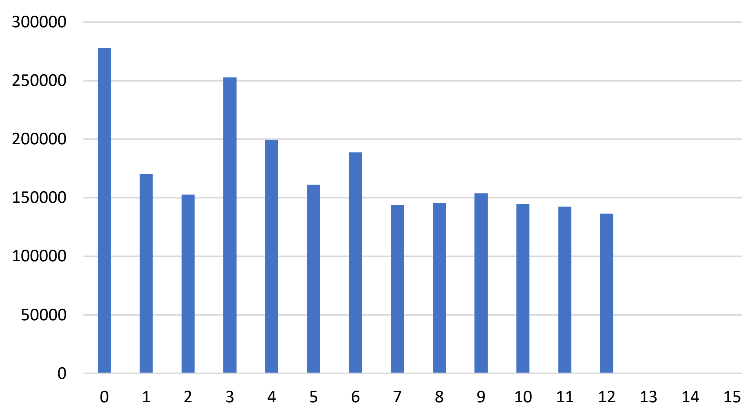


Figura 4.13: *Histograma cuantificado de la varianza de las imágenes patrón*

pequeña, las 3 últimas cubetas están vacías. Todos los puntos dispersos de esos últimos valores del histograma están en la cubeta 12.

La última operación es la normalización de histogramas. La normalización permite eliminar la dependencia de un histograma con el tamaño de la imagen. Un histograma no es más que una contabilización del número de píxeles que hay para un rango de niveles de gris dado, por lo tanto, cuanto mayor sea la imagen, mayor será este número.

La información que nos interesa de los histogramas son sus valores relativos. OpenCV ofrece una función llamada *normalize()* que permite normalizar vectores (histogramas) con diferentes normas: normal L1, norma L2, normal L-inf y norma MINMAX. Simplemente hay que elegir una norma, fijar el techo inferior y superior, y pasar la imagen origen y destino. En el desarrollo de este programa se ha utilizado la norma MINMAX, que hace que el mínimo valor del histograma coincida con el techo inferior, y el valor máximo con el techo superior, y el resto de valores son calculados de forma proporcional. El techo inferior elegido ha sido 0, y el techo superior, 1.

No se muestran los histogramas porque sus perfiles son prácticamente idénticos a los mostrados en la figura 4.12 y 4.13, como cabría esperar. Aún así, en la tabla 4.1 se muestran los valores de los histogramas normalizados.

Cubeta	Valor
0	0
1	0,0010684256
2	0,025428118
3	0,40113825
4	1
5	0,54690319
6	0,045895983
7	0,0035737704
8	0,00015954333
9	0,31217483

(a) Histograma ULBP

Cubeta	Valor
0	1
1	0,61358482
2	0,5503245
3	0,9106195
4	0,71859491
5	0,58023685
6	0,67964697
7	0,51840621
8	0,52481651
9	0,55334955
10	0,52081549
11	0,51273781
12	0,49110481
13	0
14	0
15	0

(b) Histograma varianza

Tabla 4.1: *Valores del histograma normalizado ULBP y de la varianza*

El penúltimo paso antes de finalizar el programa es exportar los datos a un archivo para que posteriormente pueda ser utilizado por el programa

principal. Este archivo se llamará "datosPatron.xml" y estará ubicado en la carpeta "patternData".

Se guarda la siguiente información en el archivo:

- Fecha: contiene la fecha y la hora a la cual se ha generado el archivo.
- Factor de escala.
- Radio.
- Número de vecinos.
- Tamaño del histograma de la varianza.
- Vector de cuantificación: indica los puntos de corte del histograma de tamaño 600 para convertirlo en el de tamaño 16 cuantificado, de forma que se hagan las mismas particiones tanto en el histograma patrón cómo en las imágenes laparoscópicas a analizar.
- Histograma de la varianza cuantificado y normalizado.
- Histograma ULBP normalizado.

En el anexo B.4 se muestra un ejemplo de archivo "datosPatron.xml".

Finalmente, sacamos por pantalla los histogramas y los almacenamos en el disco de forma gráfica. La función que genera una imagen a partir del histograma es *dibujaHistograma()*, contenida en el archivo *histo.cpp*, que se encarga de unir los puntos máximos de las cubetas de un histograma.

4.3.2. "LBP y varianza combinado, método 1"

El primer paso es calcular el vector de cuantificación de la varianza. Para ello, recorremos todas las imágenes patrón y se calculan la varianza y los histogramas (sólo de la varianza) utilizando las mismas funciones que en la otra variante. Una vez sumados todos los histogramas, llevamos a cabo una cuantificación de la varianza utilizando la función *calculaCuantificacionVar()*. Obtenemos el mismo vector que se calculó en la variante anterior.

Únicamente necesitamos el vector de cuantificación, por lo que no generaremos el histograma cuantificado.

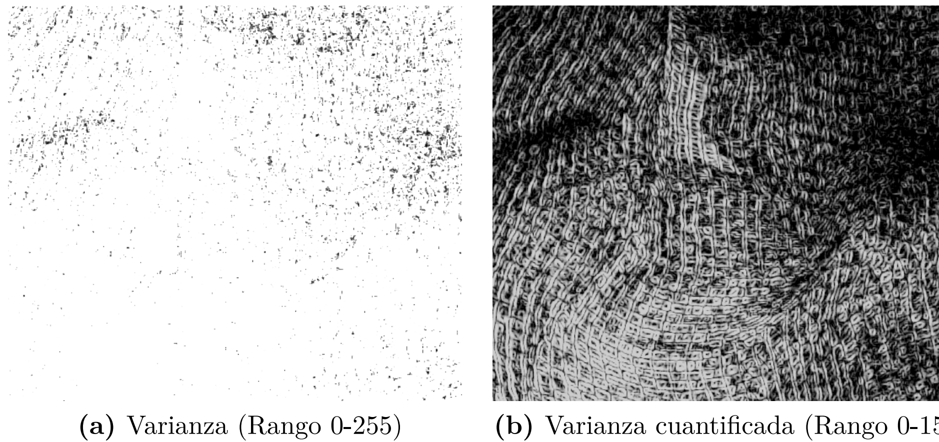
A continuación, volvemos a leer todos los archivos patrón para generar el histograma que utilizaremos para la comparación.

El primer paso es calcular la imagen de varianzas otra vez. Podríamos haber almacenado y reutilizado las imágenes calculadas anteriormente, pero dado que este programa sólo se ejecuta una vez, y el tiempo necesario no es muy elevado, se ha decidido repetir el cálculo.

Después, se calcula la imagen mLBP o LBP, según elija el usuario. Seguidamente se calcula la imagen ULBP, al igual que en la variante anterior.

A continuación, utilizamos el vector de cuantificación calculado anteriormente para cuantificar la imagen de la varianza. Para ello, se utiliza la función *cuantImgVar()* que recibe la imagen de la varianza original y el vector de cuantificación, y devuelve la imagen cuantificada (no el histograma).

En la figura 4.14 se muestran ambas imágenes. Cabe mencionar, que la varianza sin cuantificar es una imagen del tipo *CV_32FC1*, lo que significa que utiliza 32 bits para la representación de los píxeles, y el formato es coma flotante, por lo que los números serán decimales. Los formatos convencionales de imagen no soportan valores de píxeles decimales, por lo tanto la imagen que se muestra tiene sus valores redondeados.



(a) Varianza (Rango 0-255) **(b)** Varianza cuantificada (Rango 0-15)

Figura 4.14: *Imágenes de la varianza y la varianza cuantificada del primer patrón*

Una vez calculadas las imágenes ULBP y la de la varianza cuantificada, se procede a crear una imagen que es la combinación de ambas. Esta se obtiene de la siguiente manera:

$$IMG_{ULBP/Var} = 16 \cdot IMG_{ULBP} + IMG_{VAR} \quad (4.2)$$

La razón por la que se multiplica por 16 los valores de la imagen ULBP es que el rango de valores de la imagen de la varianza es de 0 a 15, y de la imagen ULBP de 0 a $P + 1$. De esta forma, la parte izquierda del histograma corresponderá a la varianza, mientras que la parte derecha corresponde a valores ULBP. Podría pensarse que los valores 0 colisionan, pero como se veía en la tabla 4.1a, nunca aparece el patrón 0. Y aunque apareciera, no tendría por qué tener un impacto negativo, ya que la forma de proceder es consistente.

En la tabla 4.2 se ilustra cuáles son los rangos de las diferentes imágenes utilizadas en función de P, y en el caso particular en el que $P = 8$.

Imagen	Rango (P)	Rango ($P = 8$)
ULBP	$0 \rightarrow P + 1$	$0 \rightarrow 9$
$16 \cdot$ ULBP	$0 \rightarrow 16\,(P + 1)$	$0 \rightarrow 144$
Varianza	$0 \rightarrow 15$	$0 \rightarrow 15$
$16 \cdot$ ULBP + Varianza	$0 \rightarrow 16\,(P + 1) + 15$	$0 \rightarrow 159$

Tabla 4.2: *Rango de las diferentes imágenes patrón*

En la figura 4.15 mostramos la imagen ULBP, la imagen ULBP multiplicada por 16, la imagen de la varianza y por último la suma $16 \cdot IMG_{ULBP} + IMG_{VAR}$. También se adjunta su histograma.

Esta vez no se ha modificado la escala de los niveles de gris de las imágenes, de forma que se puedan apreciar mejor las diferencias entre estas.

Una vez calculada esta imagen combinada, se genera su histograma haciendo uso de la función *calcHisto()*. Los parámetros del histograma son: $(P + 1) \cdot 16 + 16$ cubetas, valor mínimo 0 y valor máximo $(P + 1) \cdot 16 + 15$. Al igual que en la anterior variante, recorremos todas las imágenes y se suman todos los histogramas.

En la figura 4.16 se muestra el histograma resultante de la suma de todas las imágenes.

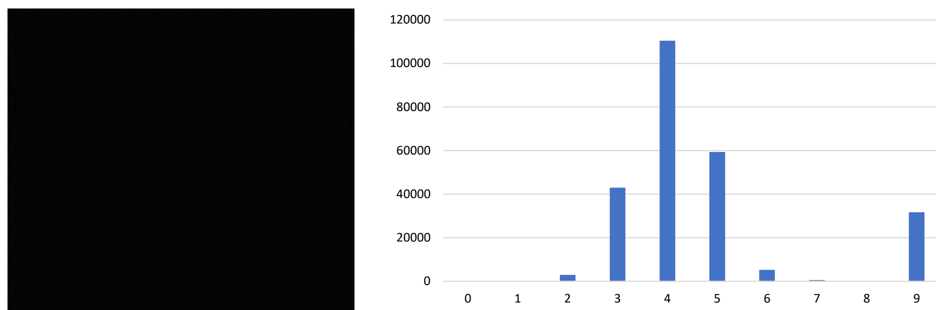
A continuación, los pasos realizados son idénticos. Se normaliza el histograma patrón y guardamos en un fichero todos los datos necesarios, salvo que únicamente es necesario almacenar un histograma. Los campos del fichero son los siguientes: fecha, factor de escala, radio, número de vecinos, tamaño del histograma de la varianza (con el que se ha calculado el vector de cuantificación), vector de cuantificación e histograma ULBP + VAR.

En el anexo B.5 se muestra un ejemplo de archivo de salida.

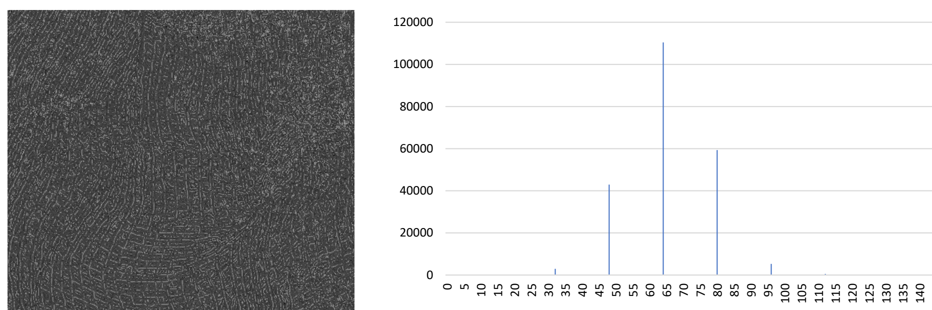
4.3.3. "LBP y varianza combinado, método 2"

En este caso, no se calcula el vector de cuantificación de la varianza, ya que por la forma de calcular el histograma, no es necesario.

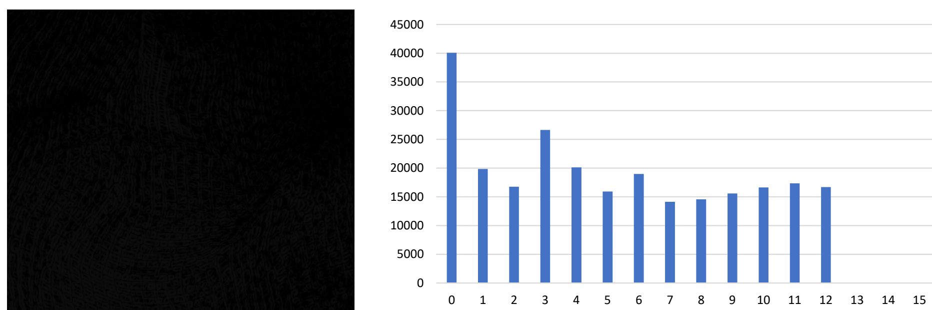
Por lo tanto, se recorren todas las imágenes patrón y se llevan a cabo los siguientes pasos: primer calculamos la imagen de varianzas y mLBP (o LBP) para el vecindario dado y hacemos la conversión LBP-ULBP. A continuación, calculamos el histograma de la imagen ULBP de una forma peculiar. La forma convencional de calcular un histograma es recorrer toda la imagen, y para cada píxel sumar una unidad a la cubeta correspondiente a su nivel de gris. En esta variante, en vez de sumar una unidad, se sumará el valor de la varianza. De esta forma, integramos la varianza dentro del histograma ULBP, y evitamos tener que realizar la tediosa cuantificación de la varianza. Como excepción, cuando



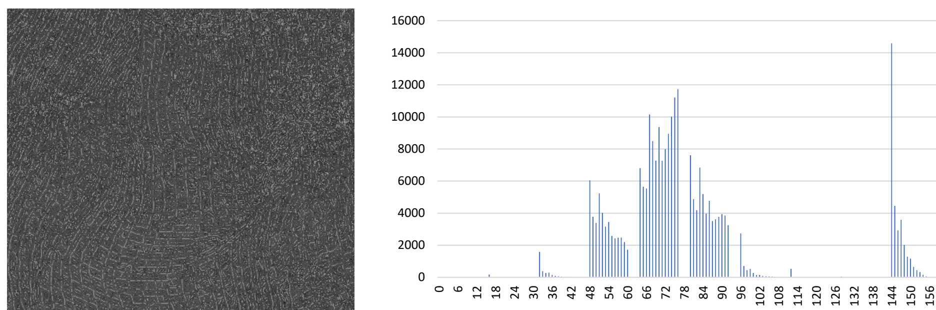
(a) ULBP



(b) 16 · ULBP



(c) Varianza



(d) 16 · ULBP + Varianza

Figura 4.15: *Proceso de combinación de la imagen ULBP e imagen varianza*

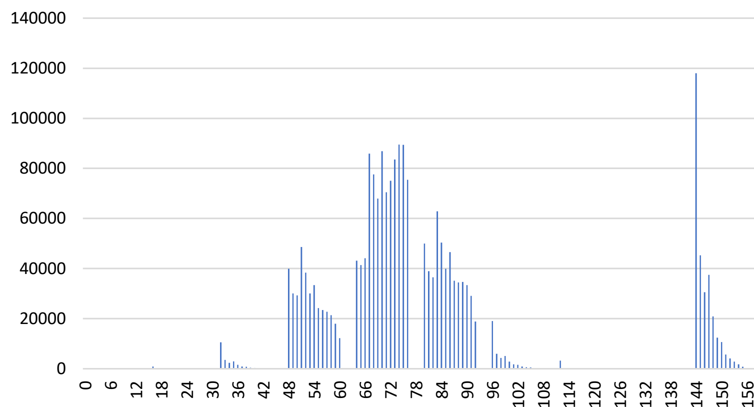


Figura 4.16: *Suma de histogramas de las imágenes ULBP y varianza combinadas*

el valor de la varianza es 0, se sumará 1 a la cubeta, ya que se ha demostrado experimentalmente que funciona mejor. La función *calcularHistoVar()* es la encargada de llevar a cabo esta tarea.

Una vez recorridas todas las imágenes y habiendo sumado sus histogramas, el histograma obtenido se muestra en la figura 4.17.

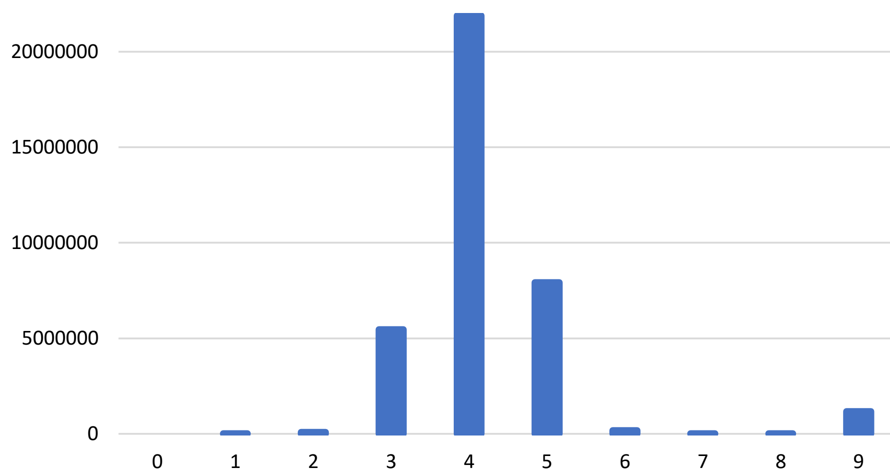


Figura 4.17: *Histograma patrón utilizando "LBP y varianza combinado" método 2*

Si se compara con el histograma patrón ULBP de la figura 4.12b, se puede ver que tienen una forma similar.

A continuación, normalizamos el histograma utilizando la norma *MINMAX* y guardamos los datos del patrón en un archivo. En esta variante únicamente es necesario almacenar la fecha, el factor de escala, el radio, el número de vecinos y el histograma normalizado.

4.4. LBP. Programa principal

El programa "LBP" es el programa principal y su objetivo es determinar si en una imagen de entrada hay alguna gasa quirúrgica o no, y localizarla espacialmente.

Previamente, se han debido de ejecutar los programas "generador de LUTs", para poder realizar la conversión LBP-ULBP, y "generador de patrón", para tener la información del patrón de referencia de las gasas.

Las imágenes con las que se van a ensayar se muestran en la figura 4.18. Siguen la misma numeración que las imágenes presentes en la carpeta "images".

Estas imágenes de prueba permiten comprobar el funcionamiento del programa en diferentes escenarios. Hay imágenes en las que hay gasas con y sin sangre, y con elementos de fondo con mayor y menor homogeneidad.

Este programa sí que tiene 4 versiones diferentes, que corresponden con cada una de las variantes. Se explicará de forma extensa la variante principal "LBP", y a continuación las otras tres variantes en las que habrá pequeñas modificaciones y adiciones.

4.4.1. Variante "LBP"

El código fuente de la variante "LBP" se encuentra en el anexo A.1.

Antes de ejecutarse la función *main()*, encontramos 3 macrodefiniciones que modifican diferentes parámetros del programa:

- *TAM_TILE*: indica el lado del cuadrado, en píxeles, en el que dividimos la imagen para la comparación de histogramas. El valor predeterminado es 100.

- *GRADO_SUPERP_TILE*: establece el tanto por uno de superposición que tienen las diferentes teselas. El valor por defecto es 0.

- *UMBRAL_HISTO_ULBP*: fija cual es el umbral a partir del cual un valor de comparación de histograma es considerado gasa o no. El valor por defecto es 10.

El programa comienza pidiendo al usuario que introduzca un número de imagen. Se puede elegir entre las 26 imágenes de la figura 4.18 y otras 6 que no forman parte del estudio, numeradas del 1 al 32 (las 6 imágenes restantes se pueden analizar y están presentes en la carpeta "images", pero dado que son imágenes de gasas con fondo blanco, no se tendrán en cuenta).

Una vez elegida la imagen, el programa lee el archivo *datosPatron.xml,* generado anteriormente, a través de una llamada a la función *leeFicheroDa-*

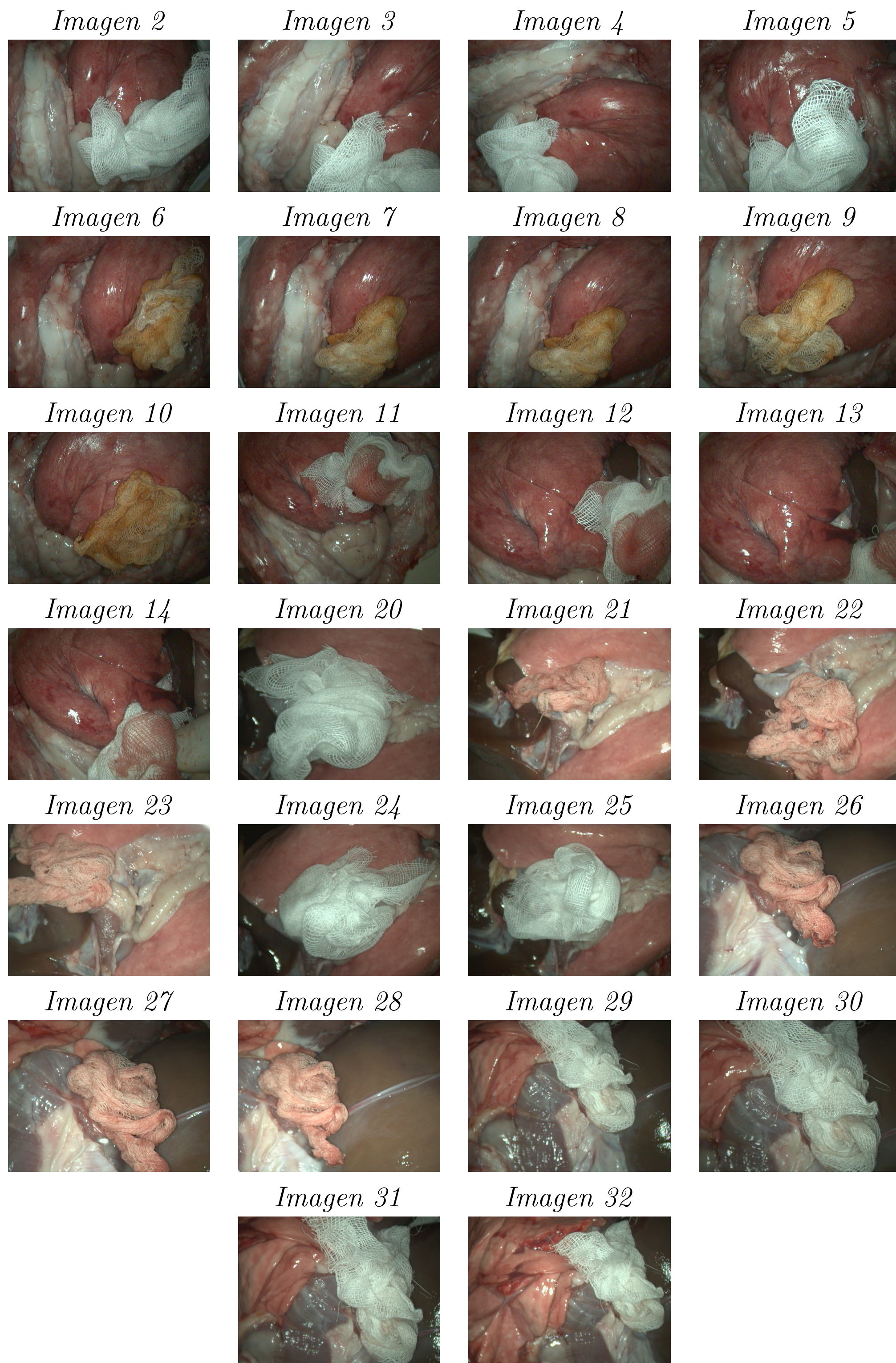


Figura 4.18: *Conjunto de imágenes laparoscópicas de prueba (misma numeración que los nombres de los archivos originales)*

tosPatron(). Pasamos como argumentos las variables *factorEscala*, *R*, *P* y *histoPatronULBP* y las inicializa con los datos presentes en el archivo.

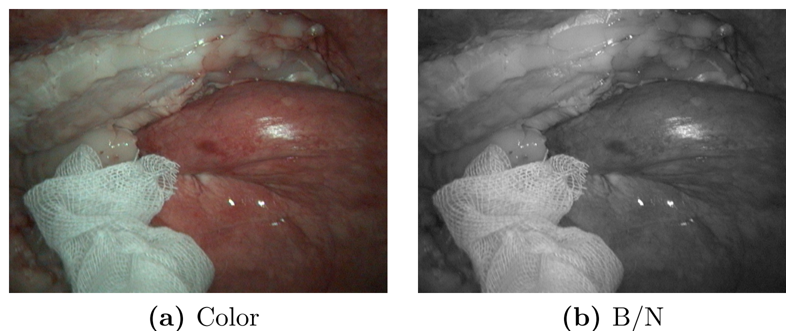
Después se llama a la función *leeFicheroLUT()* que inicializa el vector que contiene la *look-up table*. Esta función recibe el número de vecinos *P*, y debe haberse generado la LUT para ese número de vecinos anteriormente.

Por último, la función *leeFicheroImagen()* almacena en la variable *imgColor* la imagen laparoscópica elegida por el usuario.

Estas tres funciones están presentes en el archivo *fichero.cpp*. Si no se encuentra alguno de los archivos a leer, se lanzará un mensaje de error y se finalizará la ejecución del programa.

A continuación, convertimos la imagen de entrada (*imgColor*) en blanco y negro (*img*). Para ello, utilizamos la función nativa de OpenCV *cvtColor()*, y le pasamos el argumento *COLOR_BGR2GRAY*, para indicar que la conversión es de color a blanco y negro.

En la figura 4.19 se muestra la imagen de muestra que se utilizará como ejemplo (número 4) y la conversión a blanco y negro.



(a) Color **(b)** B/N

Figura 4.19: *Imagen laparoscópica de ejemplo en color y blanco y negro*

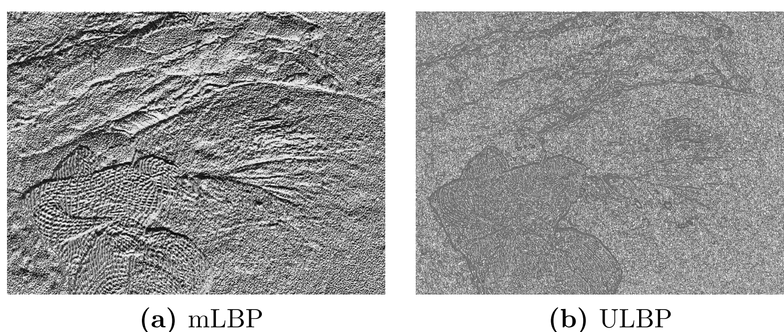
El siguiente paso es sólo necesario si utilizamos el operador mLBP en vez de LBP. Haciendo uso de la función *media()*, calculamos la imagen de medias para una vecindad dada.

Después, se llama a la función *mLBP()* o *LBP()*, en función de la decisión del usuario, y almacenamos en la variable *imgLBP* la imagen resultante. En este ejemplo se ha utilizado una vecindad $P = 8/R = 1$ y el operador mLBP.

Utilizando cualquiera de los dos operadores, el siguiente paso es convertir la imagen LBP en ULBP. Una vez más, se utilizará la función *LBP2ULBP()*, como ya se explicó en el programa "generador de patrón".

En la figura 4.20 mostramos las imágenes mLBP y ULBP (escala 0-9 trasladada a 0-255) correspondientes a la imagen de muestra 4.

Una vez obtenida la imagen ULBP, llamamos a la función *comparaHistosTiles()*. Esta función recibe la imagen ULBP (*imgULBP*), el histograma ULBP



(a) mLBP **(b)** ULBP

Figura 4.20: *Imagen mLBP y ULBP correspondiente a la imagen laparoscópica 4*

patrón normalizado (*histoPatronULBP*), el tamaño de la tesela (*TAM_TILE * factorEscala*) y el grado de superposición de las teselas (*GRADO_SUPERP_TILE*). Devuelve un vector con el valor de la comparación de histogramas de cada tesela con el histograma patrón.

La razón por la que es necesario dividir la imagen en teselas, es porque en gran parte de las situaciones, la gasa no ocupará toda la escena, sino sólo una porción de ella. Si se aplicara la comparación de histogramas a toda la imagen, rara vez determinaría la presencia de gasas, ya que gran parte de la información de la imagen correspondería al fondo y no a las gasas.

La función *comparaHistosTiles()* realiza los siguientes pasos. Primero, extrae de la imagen original una tesela y es almacenada en una variable auxiliar. Calcula el histograma de la tesela utilizando los mismos parámetros que en el histograma patrón. A continuación, normaliza el histograma utilizado la norma *MINMAX*.

Ya estamos en condiciones de realizar una comparación de histogramas. Esta está implementada de forma nativa en OpenCV (*compareHist)* y ofrece 4 métodos de comparación: correlación, chi-cuadrado, intersección y distancia Bhattacharyya. Al igual que en la normalización, las diferencias no son muy significativas y se ha elegido el método chi-cuadrado, siendo el más empleado. La distancia chi-cuadrado entre dos histogramas viene dada por la siguiente expresión:

$$d_{chi-cuadrado}\left(H_1, H_2\right) = \sum_I \frac{\left(H_1\left(I\right) - H_2\left(I\right)\right)^2}{H_1\left(I\right)} \tag{4.3}$$

Cuanto menor es el valor de la distancia chi-cuadrado, más parecidos son los histogramas comparados.

Por último, almacenamos en un vector el valor de esta comparación multiplicada por 100 y convirtiendo el tipo de dato a entero. Esto permite una visualización más cómoda de los resultados, al no tener que usar decimales.

Este procedimiento se realiza para cada una de las teselas en las que está dividida la imagen. Algunas de estas pueden ser de un tamaño menor, ya que es posible que la imagen no pueda ser dividida exactamente en teselas de tamaño *TAM_TILE*. El orden que tiene el valor de comparación de cada tesela en el vector es de izquierda a derecha y de arriba a abajo.

En la figura 4.21 se muestra cada una de las teselas en las que se ha dividido la imagen ULBP.

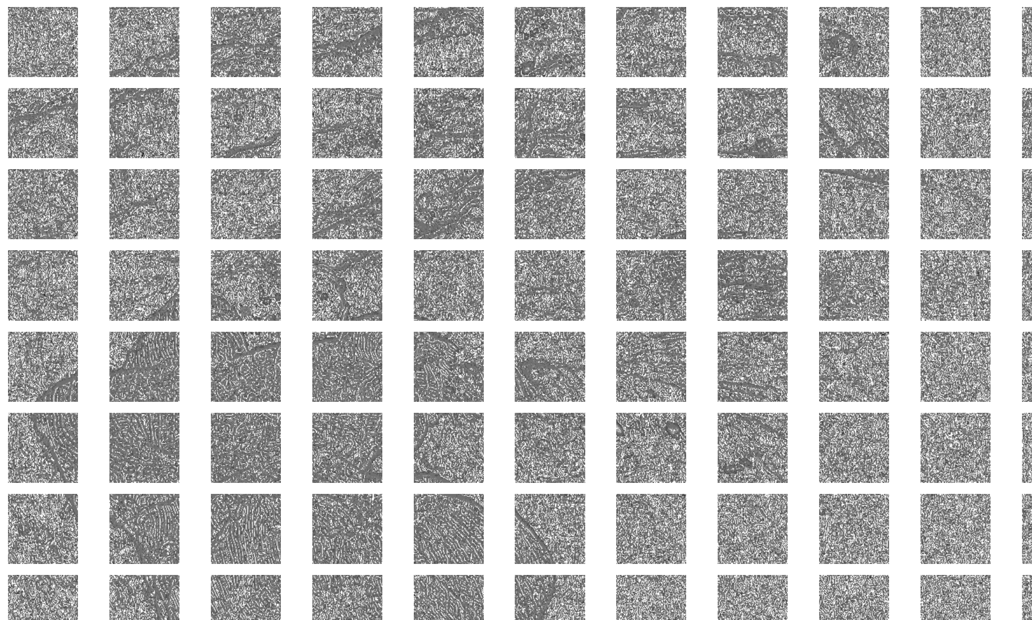


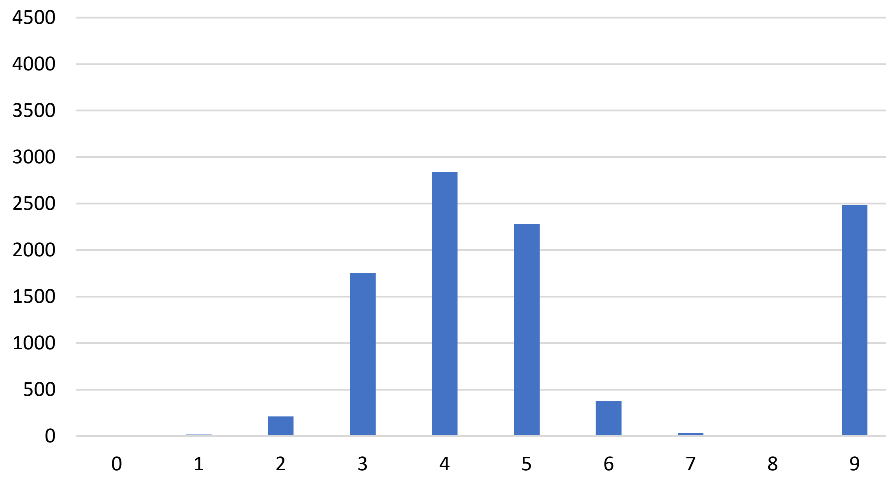
Figura 4.21: *Imagen ULBP dividida en las 88 teselas*

En la figura 4.22 mostramos el histograma de la primera tesela (primera fila, primera columna) y de la tesela número 57 (sexta fila, segunda columna), junto con el valor de la comparación de histogramas utilizando la distancia chi-cuadrado y el histograma ULBP patrón.

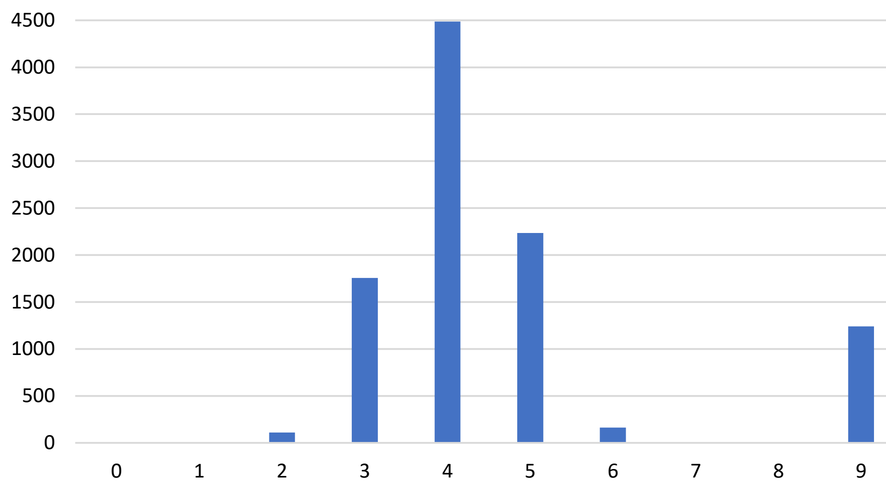
Ambas teselas tienen un tamaño de 100x100 píxeles, por lo tanto la suma total de píxeles es de 10 000. El histograma patrón se ha modificado proporcionalmente, de forma que la suma total de píxeles sea también 10 000 y que la comparación sea justa. Además, se ha utilizado la misma escala en el eje vertical, haciendo más sencilla la comparación de histogramas de forma visual.

A simple vista, vemos como el histograma de la tesela 57 se parece mucho más al histograma patrón que el de la tesela 1. Esta diferencia se ve reflejada en los valores de la comparación, que en la tesela 1 es significativamente mayor que en la tesela 57, indicando que los histogramas no son muy semejantes y por lo tanto, la tesela 1 no se considera gasa. En cambio, el valor de la comparación para la tesela 57 es suficientemente pequeño como para considerar que es gasa.

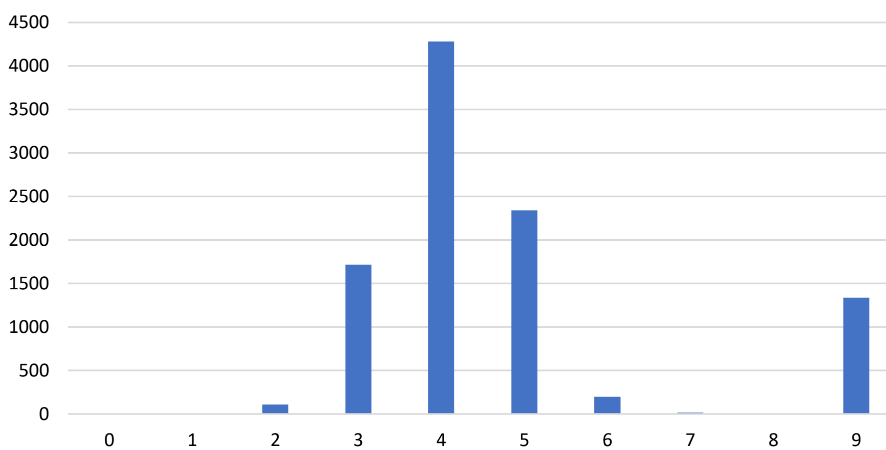
Una vez obtenido el vector de comparaciones, dibujamos sobre la imagen



(a) Histograma tesela 1. Valor de la comparación: 1,5640



(b) Histograma tesela 57. Valor de la comparación: 0,0134



(c) Histograma patrón ULBP (reajustado)

Figura 4.22: *Histograma de dos teselas de la imagen ULBP y patrón ULBP*

de entrada los valores de comparación de cada tesela, y se resaltan en rojo las teselas cuyo valor de comparación sea estrictamente menor que el valor umbral fijado (*UMBRAL_HISTO_ULBP*). La función que realiza esta tarea se llama *visualizaValoresEnTiles()* y se encuentra en el archivo *visualiza.cpp*.

En la figura 4.23 se muestra la imagen que se obtiene al ejecutar esta función.

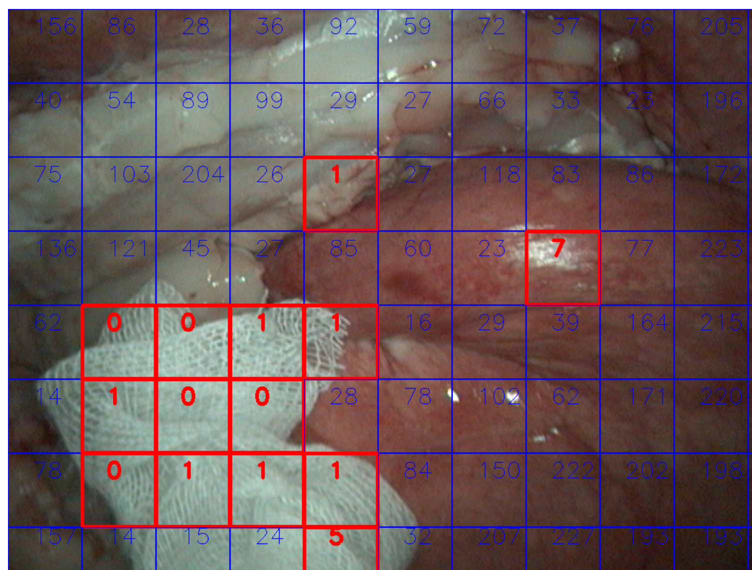


Figura 4.23: *Imagen resultado de la imágen laparoscópica 4*

En la imagen resultado podemos ver como los valores de comparación en las teselas que no son gasas son bastante más altos que en las que corresponden a gasa, excepto dos de ellas. Aún así, la delimitación de la venda es bastante precisa.

4.4.2. Variante "LBP y varianza por separado"

El código fuente de la variante "LBP y varianza por separado" se encuentra en el anexo A.2.

Esta variante, además de las tres macrodefiniciones de la variante "LBP", (cambiando *UMBRAL_HISTO_ULBP* por 8, en vez de 10) define *UMBRAL_HISTO_VAR*, que fija el umbral a partir del cual se considera venda en la imagen de la varianza. El valor predeterminado es 160.

Tras pedir el número de la imagen al usuario, ejecuta la función *leeFicheroPatronVar()* en vez de *leeFicheroPatron()*, que además de inicializar las variables del factor de escala, el radio, el número de vecinos y el histograma patrón ULBP, inicializa el vector de cuantización y el histograma de la varianza.

A continuación, se lleva a cabo el mismo procedimiento, aunque antes de realizar la comparación de teselas, se tienen que realizar dos pasos adicionales.

El primero es calcular la imagen de la varianza. Para ello, si el vecindario es $P = 8/R = 1$, llamaremos a la función *varianza()*, mientras que para un vecindario de tamaño arbitrario, se debe llamar a la función *VARLBP()*, pasando como argumento el número de vecinos y el radio del vecindario.

En cualquier caso, obtenemos la imagen de varianzas para el vecindario establecido. A continuación, cuantificamos la imagen de la varianza. La cuantificación se realiza llamando a la función *cuantImgVar()*, y le tenemos que pasar el vector de cuantificación cargado anteriormente.

Estas tres últimas funciones se encuentran en el fichero *estadist.cpp*.

El siguiente paso es llevar a cabo la comparación de histogramas. Al igual que con la imagen ULBP, la imagen de la varianza cuantificada se divide en las mismas teselas y se compara con el histograma patrón. La función *comparaHistosVar()* (*compara.cpp*) lleva a cabo esta tarea, y al igual que *comparaHistosTiles()*, devuelve un vector con el valor de la comparación de cada tesela con el patrón.

Por último, llamamos a la función *visualizaValoresEnTiles()*, que se encuentra en el fichero *visualiza.cpp*. Es la misma función que en la variante "LBP", pero anteriormente se le pasaba el valor -1 en el campo del umbral de la varianza. Eso indicaba que no se utilizaba la varianza para la comparación, ahora en cambio se le pasa la macrodefinición *UMBRAL_HISTO_VAR* y sí que la tiene en cuenta en la comparación.

El operador utilizado para combinar la información ULBP y de la varianza ha sido el *or*.

En la figura 4.24 mostramos la imagen de salida. En cada tesela aparecen 2 números. El superior indica el valor de comparación ULBP y el inferior el de la varianza.

Cuando el número superior aparece en rojo, significa que según el operador ULBP se considera gasa, mientras que si aparece en azul, ha superado el umbral. De igual manera, si el número inferior es rojo, significa que la varianza considera que se trata de una gasa, y en azul si no es así.

4.4.3. Variante "LBP y varianza combinado, método 1"

El código fuente de la variante "LBP y varianza combinado, método 1" se encuentra en el anexo A.3.

Esta variante tiene un único umbral para la comparación de histogramas. La macrodefinición se llama *UMBRAL_HISTO_ULBP_VAR*, y su valor predeterminado es 50.

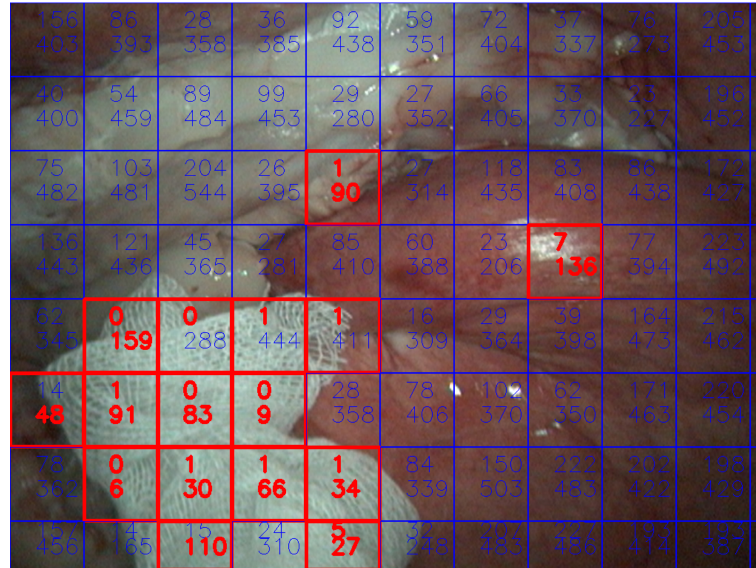


Figura 4.24: *Imagen resultado utilizando ULBP y varianza por separado*

La función encargada de inicializar las variables se llama *leeFicheroDatosPatronULBPVar()*. Devuelve el factor de escala, el radio del vecindario, número de vecinos, vector de cuantificación e histograma patrón, que combina la información ULBP y la de la varianza.

El resto de pasos son idénticos que con la variante "LBP y varianza por separado", hasta que se llega a la comparación de teselas. Antes, hay que generar la imagen combinación de la siguiente manera:

$$IMG_{ULBP/Var} = 16 \cdot IMG_{ULBP} + IMG_{VAR} \tag{4.4}$$

A continuación, llamamos a la función *comparaHistoTilesULBPVar()*, que es similar a la de las otras variantes. La única diferencia es que al guardar el valor de comparación en el vector, en vez de multiplicar por 100, lo hace por 10.

Por último, la función *visualizaValoresEnTiles()* genera la imagen resultado con los valores de comparación de cada una de las teselas.

En la figura 4.25 se muestra un ejemplo de ejecución con la imagen laparoscópica 4.

4.4.4. Variante "LBP y varianza combinado, método 2"

El código fuente de la variante "LBP y varianza combinado, método 2" se encuentra en el anexo A.4. El valor umbral de la comparación de histogramas se encuentra en la macrodefinición *UMBRAL_HISTO_ULBP_VAR_2* y tiene un valor predeterminado de 6.

Para cargar los datos del patrón utilizaremos la misma función que en

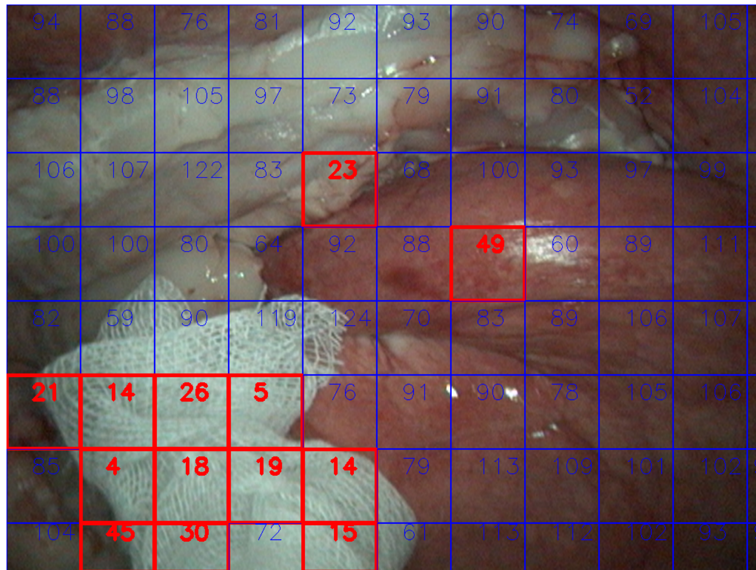


Figura 4.25: *Imagen resultado utilizando la variante "LBP y varianza combinado, método 1"*

la variante "LBP", *leeFicheroDatosPatron()*. A continuación, convertimos la imagen a color, calculamos la imagen de medias, la imagen LBP (o mLBP), la convertimos a ULBP y calculamos la imagen de varianza, utilizando las mismas funciones vistas ahora ahora.

La comparación de teselas es la que varía respecto a otras variantes, y la función encargada se llama *comparaHistosTilesULBPVar2()*. A esta función hay que pasarle la imagen ULBP, la imagen de varianzas, el histograma patrón, el tamaño de las teselas y el grado de superposición de teselas. Devuelve un vector con el valor de comparación de cada una de las teselas.

Internamente, el histograma de cada tesela lo calcula utilizando la función *calcularHistoVar()*, que genera un histograma combinado de los valores ULBP y de la varianza, como se ha explicado en la subsección 4.3.3. El valor de comparación se multiplica por 150 para que los números sean más cómodos de visualizar.

Por último, visualizamos los resultados de la comparación utilizando la función *visualizaValoresEnTiles()*.

La figura 4.26 muestra la imagen de resultados utilizando la imagen laparoscópica 4.

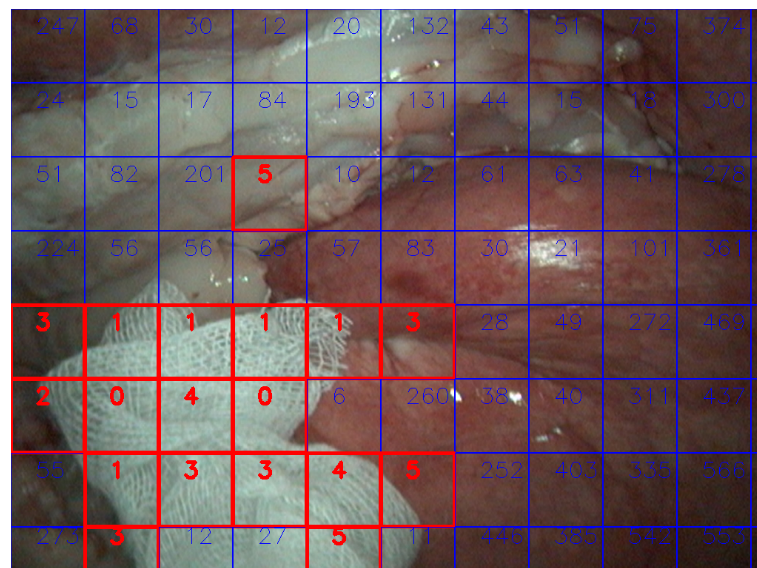


Figura 4.26: *Imagen resultado utilizando la variante "LBP y varianza combinado, método 2"*

Capítulo 5

Resultados experimentales

Antes de comenzar con el análisis de resultados, vamos a resumir los diferentes parámetros que se pueden cambiar a la hora de ejecutar el programa.

- En primer lugar, podemos elegir entre las 4 variantes. La primera que únicamente utiliza el operador LBP. La segunda incorpora la varianza utilizando el operador $VAR_{P,R}$ (ecuación 3.11). Se generan dos histogramas diferentes, uno del operador LBP y otro de la varianza. La tercera y cuarta variante generan un histograma único, integrando en este los valores obtenidos por ambos operadores de forma diferente.

- Respecto al operador LBP puede utilizarse una de las siguiente variantes: $LBP_{P,R}^{riu2}$ (ecuación 3.10) o $mLBP_{P,R}^{riu2}$ (ecuación 3.15).

- También puede cambiar el vecindario, modificando el número de vecinos P y radio R.

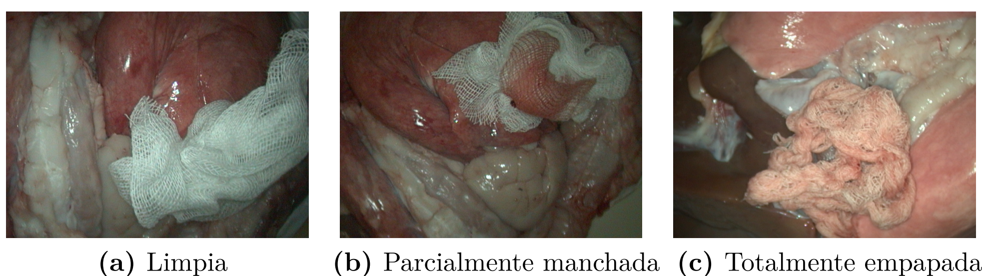
- El tamaño de las teselas pueden cambiar.

- El umbral a partir del cual se considera gasa.

Debido al elevado número de parámetros, no es posible realizar un análisis pormenorizado de cada una de las combinaciones. Por lo tanto, se presentará un análisis generalizado del comportamiento que tiene la modificación de ciertos parámetros, de forma que se este comportamiento se pueda extrapolar a diferentes combinaciones. Por ejemplo, la modificación del tamaño de las teselas va a tener el mismo impacto en cualquiera de las 4 variantes, y únicamente será necesario explicar el impacto que tiene de forma general.

En algunas comparaciones se dividirán los resultados en tres grupos: gasas limpias, gasas parcialmente manchadas de sangre y gasas totalmente empapadas en sangre u otros fluidos. En la figura 5.1 se muestra una imagen laparoscópica de cada uno de los grupos.

El número de imágenes disponibles de cada grupo son las siguientes:



(a) Limpia (b) Parcialmente manchada (c) Totalmente empapada

Figura 5.1: *Grupos de imágenes laparoscópicas*

- Gasas limpias: 11
- Gasas parcialmente manchadas de sangre: 4
- Gasas totalmente empapadas en sangre u otros fluidos: 11

Por último, también se analizará el tiempo de ejecución para determinar si es apto o no para una aplicación en tiempo real.

5.1. Método de análisis

A continuación se explicará cuál será el método a seguir para identificar un buen o mal comportamiento del programa, y cuales son los objetivos que se buscan.

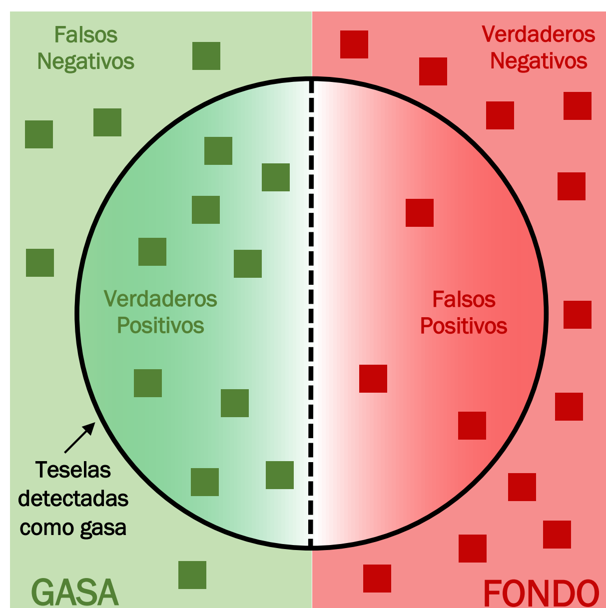


Figura 5.2: *Conjunto de muestras*

En la figura 5.2 se representan visualmente los cuatro grupos en los que se puede clasificar una tesela. Cada cuadrado corresponde a una tesela, y será

clasificada dentro de estos cuatro grupos según su posición. La mitad izquierda corresponde a la gasa en su totalidad (zona verde) y la mitad derecha a todo aquello que no es gasa, el fondo (zona roja). El círculo negro representa la elección del programa, por lo tanto, todas las teselas contenidas en el círculo han sido consideradas como gasa. Los 4 grupos en los que se puede clasificar una tesela son los siguientes:

- Falsos negativos (*fn*): teselas no consideradas como gasas pero sí que son gasa.
- Verdaderos positivos (*vp*): teselas que son gasa y han sido correctamente identificadas.
- Falsos positivos (*fp*): teselas consideradas como gasa pero que no forman parte de esta.
- Verdaderos negativos (*vn*): teselas consideradas correctamente como fondo.

Una vez contabilizadas las teselas que corresponden a cada uno de los grupos, calculamos las siguientes variables:

$$Precisión = \frac{vp}{vp + fp} \tag{5.1}$$

$$Sensibilidad = \frac{vp}{vp + fn} \tag{5.2}$$

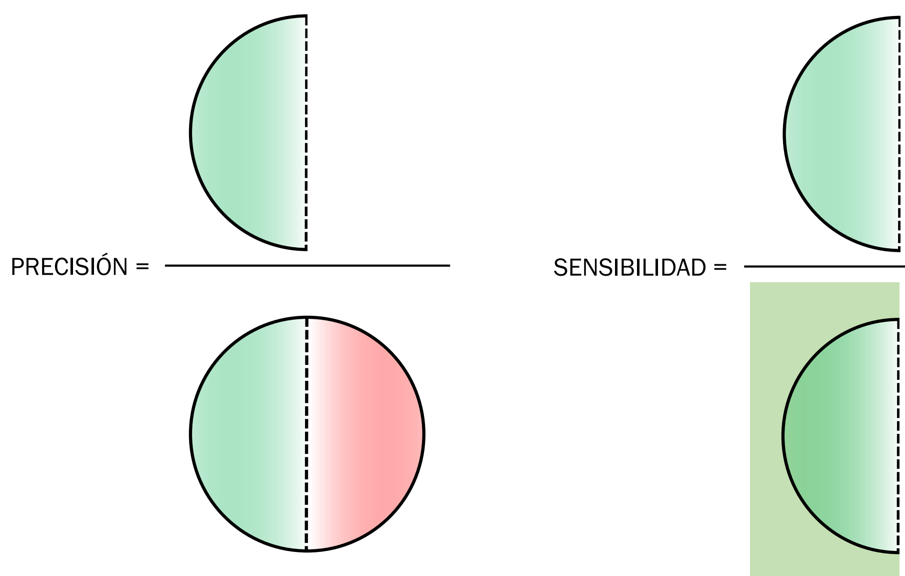


Figura 5.3: *Representación gráfica de la precisión y la sensibilidad*

En la figura 5.3 representamos gráficamente los conceptos de precisión y sensibilidad.

La precisión mide la *calidad* de la detección, es decir, el porcentaje de teselas correctamente identificadas frente al total de teselas consideradas como gasa.

En cambio, la sensibilidad es una medida de la *cantidad* de la detección, es decir, el porcentaje de teselas correctamente identificadas frente al número total de teselas que son gasa.

Un buena precisión indica que cada vez que detectamos una gasa, tenemos garantías de que la gasa realmente está presente en la imagen. Por otra parte, la delimitación de la gasa será más completa cuanto mayor sea la sensibilidad, aunque un porcentaje muy alto puede traer consigo un aumento en el número de falsos positivos, derivando en una disminución de la precisión.

El criterio que se adoptará será priorizar la precisión sobre la sensibilidad, ya que es más importante la fiabilidad a la hora de identificar una gasa y no tanto el grado de completitud de su delimitación.

Una medida que será útil para la comparación de resultados es el valor-F, que combina la información de la precisión y la sensibilidad, realizando la media armónica.

$$F = 2 \cdot \frac{precisión \cdot sensibilidad}{precisión + sensibilidad} \tag{5.3}$$

5.2. Influencia del operador

A continuación, compararemos los resultados obtenidos utilizando el operador $LBP_{P,R}^{riu2}$ y $mLBP_{P,R}^{riu2}$. Para ello, utilizaremos la primera variante (es decir, sin utilizar la varianza), un tamaño de tesela de 100x100 píxeles y un vecindario del tipo $P = 8/R = 1$. El valor umbral utilizado ha sido 10 para el operador mLBP y 15 para el operador LBP. Tras analizar todas las imágenes laparoscópicas, se han obtenido los resultados de la tabla 5.1.

Los resultados de la última columna corresponden a la media de las 26 imágenes laparoscópicas, o lo que es lo mismo, la media ponderada de los 3 grupos.

Fijándonos en los valores medios, vemos como la precisión y la sensibilidad para el operador mLBP es superior a la del operador LBP, y como consecuencia, su valor-F. La diferencia es especialmente notable en la sensibilidad, ya que la precisión se mantiene en unos valores parecidos, por lo tanto, podríamos decir que el operador mLBP nos ofrece una mayor sensibilidad (mejor delimitación de la gasa) para un mismo valor de precisión.

Si nos fijamos en los 3 grupos de imágenes laparoscópicas, ambos operadores tiene una tendencia muy parecida: los valores de precisión se mantienen

	Limpia	Manchada	Empapada	Media
Precisión	88 %	86 %	78 %	**84 %**
Sensibilidad	65 %	31 %	68 %	**61 %**
Valor-F	75 %	46 %	73 %	**71 %**

(a) Operador LBP

	Limpia	Manchada	Empapada	Media
Precisión	92 %	81 %	85 %	**87 %**
Sensibilidad	82 %	48 %	78 %	**75 %**
Valor-F	87 %	60 %	81 %	**81 %**

(b) Operador mLBP

Tabla 5.1: *Resultados obtenidos para la comparación de operadores*

cercanos para cualquier tipo de imagen, mientras que la sensibilidad es menor para las gasas parcialmente manchadas de sangre, y se mantiene con valores parecidos para los otros dos grupos. Por ello, no podemos determinar que un operador se comporta mejor en un escenario u otro, ya que los resultados son equivalentes.

En la figura 5.4 comparamos el resultado de aplicar ambos operadores sobre la imagen laparoscópica nº 25.

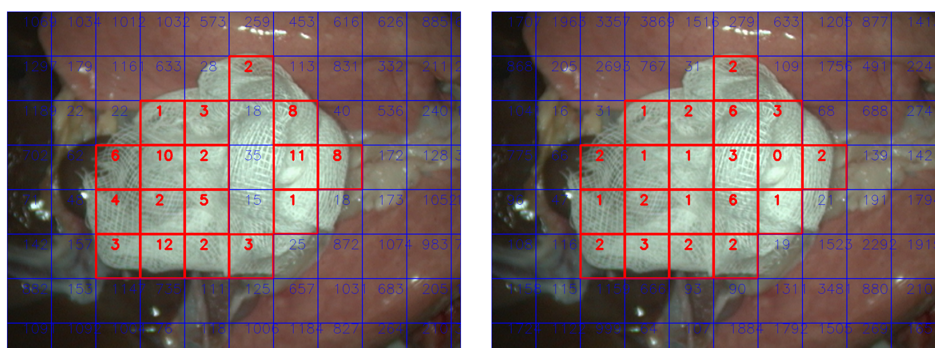
A la vista de los resultados, podemos corroborar que tenemos una precisión muy parecida (no encontramos falsos negativos) y la sensibilidad (completitud en la delimitación de la gasa) es mejor para el operador mLBP. En la subfigura 5.4c se muestra la delimitación esperada de la gasa. Vemos como en las teselas 17, 29 y 61 es difícil determinar si se considera gasa o no, por lo tanto, a todos los resultados y estadísticas hay que considerar un pequeño margen de error para compensar por las teselas límite que tienen una clasificación complicada. El criterio que se ha seguido ha sido considerar como gasas aquellas teselas en las que la gasa ocupe aproximadamente la mitad, o más, de la tesela.

A partir de ahora se utilizará el operador mLBP, dado que ofrece mejores resultados sin aumentar significativamente el coste computacional.

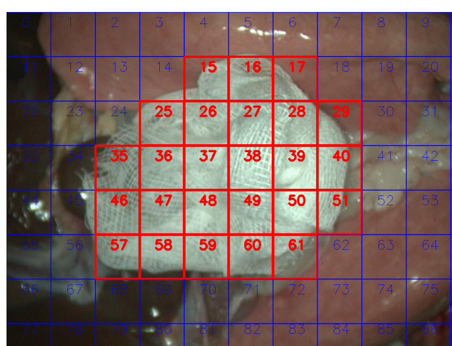
5.3. Influencia del umbral

En esta sección analizaremos la influencia del umbral en los resultados. Es uno de los parámetros más importantes, ya que condicionará la relación entre sensibilidad y precisión. Utilizaremos la primera variante del programa y el operador mLBP con un vecindario $P = 8/R = 1$. El tamaño de tesela será de 100x100 píxeles.

En la figura 5.5 se muestra una gráfica que ilustra la evolución de la sensibilidad, la precisión y el valor-F utilizando diferentes umbrales. Los porcentajes



(a) Operador LBP **(b)** Operador mLBP



(c) Delimitación esperada (Los números identifican las teselas y no hacen referencia a un valor de comparación)

Figura 5.4: *Comparaciones de operadores sobre imagen laparoscópica nº 25*

utilizados corresponden a la media de todas las imágenes.

Para valores umbrales bajos, obtenemos los mejores valores de precisión pero con muy poca sensibilidad, lo que significa que habrá un número muy bajo de falsos positivos, pero la calidad con la que delimitamos la gasa será muy baja. En cambio, para valores umbrales altos, la sensibilidad es mayor que la precisión, por lo que el número de falsos positivos habrá aumentado y la delimitación de la gasa será cada vez más completa.

Como adelantamos en la sección 5.1 (Método de análisis), priorizaremos la precisión frente a la sensibilidad.

En la tabla 5.2 se muestran 3 puntos significativos de la gráfica.

Umbral	Sensibilidad	Precisión
7	64 %	94 %
10	**75 %**	**87 %**
14	84 %	84 %

Tabla 5.2: *Puntos significativos de la gráfica de umbrales*

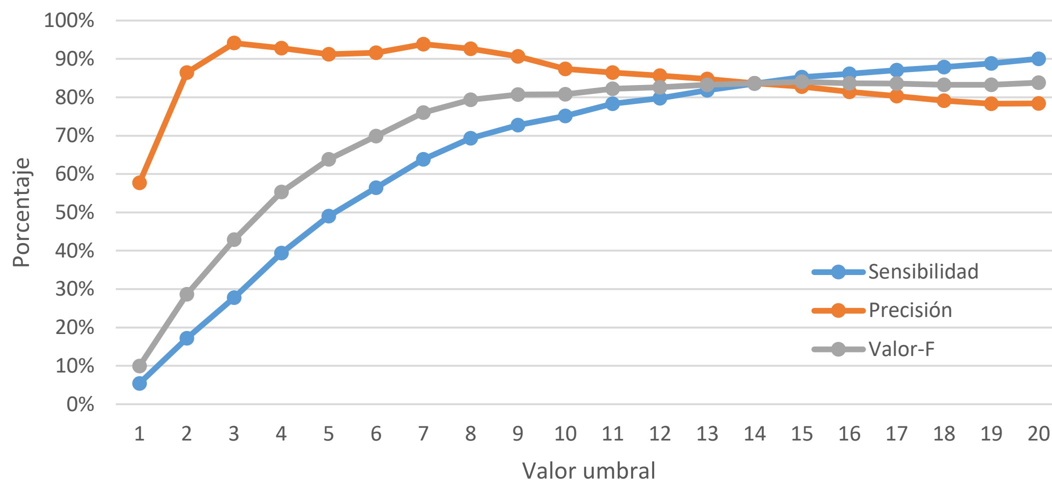


Figura 5.5: *Evolución de la sensibilidad y la precisión utilizando diferentes umbrales*

Para el umbral 7 obtenemos un máximo en la precisión con la mayor sensibilidad posible (ya que para el umbral 3 obtenemos la misma precisión pero con menor sensibilidad). En el umbral 14 se cortan las líneas de sensibilidad y precisión, y a partir de ese punto la sensibilidad será mayor que la precisión. Es por ello, que el umbral elegido se encuentra entre estos dos valores.

Aunque con el umbral 7 tendríamos una precisión muy buena, se sacrificaría la sensibilidad. Para el umbral 14, sería el último punto que podría cumplir la condición de primar la precisión sobre la sensibilidad. Es por ello, que un punto medio entre ambos valores de precisión será el óptimo para nuestra aplicación, manteniendo la precisión en unos valores altos pero con una buena sensibilidad.

Aun así, la elección del umbral dependerá en gran medida de la situación. En determinadas circunstancias, puede ser necesario elevar la precisión al máximo, para tener la máxima garantía de que cuando se detecte una gasa sea un verdadero positivo, aunque sea a costa de una mala delimitación.

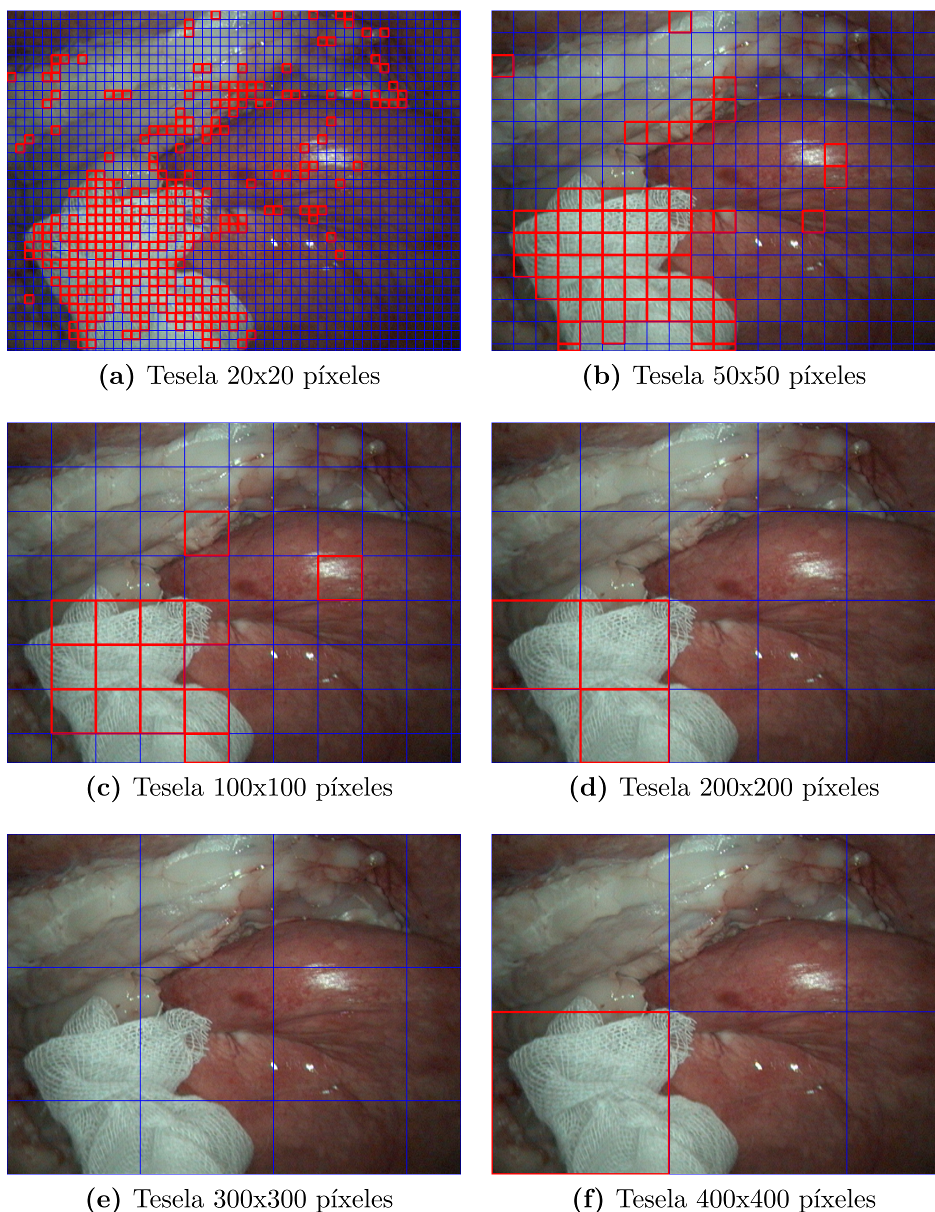
5.4. Influencia del tamaño de tesela

En un principio, el tamaño de la tesela sólo repercutirá en una mejor o peor delimitación de la gasa, aunque hay que tener en cuenta las consecuencias de variar su tamaño.

Por una parte, cuanto mayor sea el número de teselas, mayor será el coste computacional, ya que aumenta el número de comparaciones, pero mejor será la delimitación de la gasa. En cambio, si utilizamos un tamaño de tesela grande, el número de comparaciones necesarias disminuirá pero la delimitación de la gasa no será tan precisa. Aún así, la precisión a la hora de establecer los límites

de la gasa no es un requisito primordial, pues el sistema está encaminado a hacer un seguimiento-detección de las gasas que pueden aparecer en la imagen.

El la figura 5.6 se muestran los resultados obtenidos tras aplicar sobre la imagen laparoscópica nº4 la primera variante del programa, con un operador mLBP de vecindario $P = 8/R = 1$ y con un valor umbral de 10.



(a) Tesela 20x20 píxeles **(b)** Tesela 50x50 píxeles

(c) Tesela 100x100 píxeles **(d)** Tesela 200x200 píxeles

(e) Tesela 300x300 píxeles **(f)** Tesela 400x400 píxeles

Figura 5.6: *Influencia del tamaño de tesela en la imagen laparoscópica nº4*

No obstante, los resultados ponen de manifiesto que el tamaño de la tesela no sólo afecta a la precisión de la delimitación, sino también a la correcta detección de la gasa. Observamos que para tamaños de teselas pequeños (20x20 y 50x50) aumenta considerablemente el número de errores, y especialmente para un tamaño de tesela de 20x20 el número de aciertos se ve reducido. Esto

se debe a que el área que sea analiza no es suficientemente significativa como para abarcar la textura de una gasa, además, es mucho más sensible al ruido. Es decir, las teselas pequeñas son estadísticamente menos estables.

Para tamaños de tesela grandes (200x200, 300x300 y 400x400) las áreas de análisis son suficientemente grandes y significativas, pero surgen otros problemas. Por ejemplo, para un tamaño de 200x200 vemos como la tesela de la tercera columna y última fila no es detectada, mientras que utilizando tamaños de tesela más pequeños, esa zona sí ha sido correctamente identificada. Para un tamaño de 300x300 nos encontramos con el mayor problema de trabajar con teselas tan grandes, que es posible no detectar la gasa. Si da la casualidad de que todas las teselas contengan tanto fondo como gasa, es posible que la contribución de la gasa al histograma no sea suficiente como para poder considerarse como gasa. Por último, para un tamaño de 400x400, aunque detecta correctamente la gasa, estamos ante el mismo problema que con el tamaño de tesela anterior, sólo que la posición de la gasa ha sido beneficiosa en este caso. Podemos concluir que con tamaños demasiado grandes, se pasa a depender del azar para una correcta identificación de la gasa, lo cual no es aceptable.

Es por ello que el tamaño de tesela elegido ha sido de 100x100. Tiene un tamaño suficientemente grande como para ser significativo, pero suficientemente pequeño como para no depender de que la totalidad de la región correspondiente a la gasa se encuentre en la frontera de varias teselas sin ocupar ninguna de ellas de manera significativa (como ocurría para un tamaño de 300x300 píxeles). Como consecuencia, la delimitación es razonablemente precisa y el coste computacional no es muy elevado.

Cabe mencionar que las imágenes laparoscópicas con las que se están trabajando tienen una resolución de 1024x768, por lo tanto, si cambiara la resolución, habría que cambiar el tamaño de las teselas de forma proporcional.

5.5. Influencia del vecindario

Los últimos parámetros que se pueden modificar son el número de vecinos P y el radio del vecindario R. Para verificar su influencia en el desempeño del programa, se utilizarán vecindarios de radio 1, 2 y 3, con 8, 12 y 16 vecinos, lo que hace un total de 9 vecindarios diferentes.

Se ha utilizado la primera variante del programa, el operador mLBP y un tamaño de tesela de 100x100 píxeles.

Los umbrales utilizados para los diferentes vecindarios se muestran en la tabla 5.3. Para su cálculo, se ha seguido el mismo método que en la sección 5.3 (Influencia del umbral), de forma que la comparación de resultados será justa.

En la figura 5.7 se muestra un gráfico de barras en el que las barras azules

P/R	1	2	3
8	*10*	*40*	*20*
12	*60*	*70*	*30*
16	*90*	*60*	*20*

Tabla 5.3: *Umbrales utilizados para los diferentes vecindarios*

corresponden a la sensibilidad, las naranjas a la precisión y las grises al valor-F. Se han ordenado según su valor-F, de mayor a menor. Como los umbrales elegidos ya han priorizado la precisión sobre la sensibilidad siguiendo el mismo procedimiento, el valor-F es un buen indicador de un mejor o peor desempeño, siendo mejor cuanto más alto sea este valor.

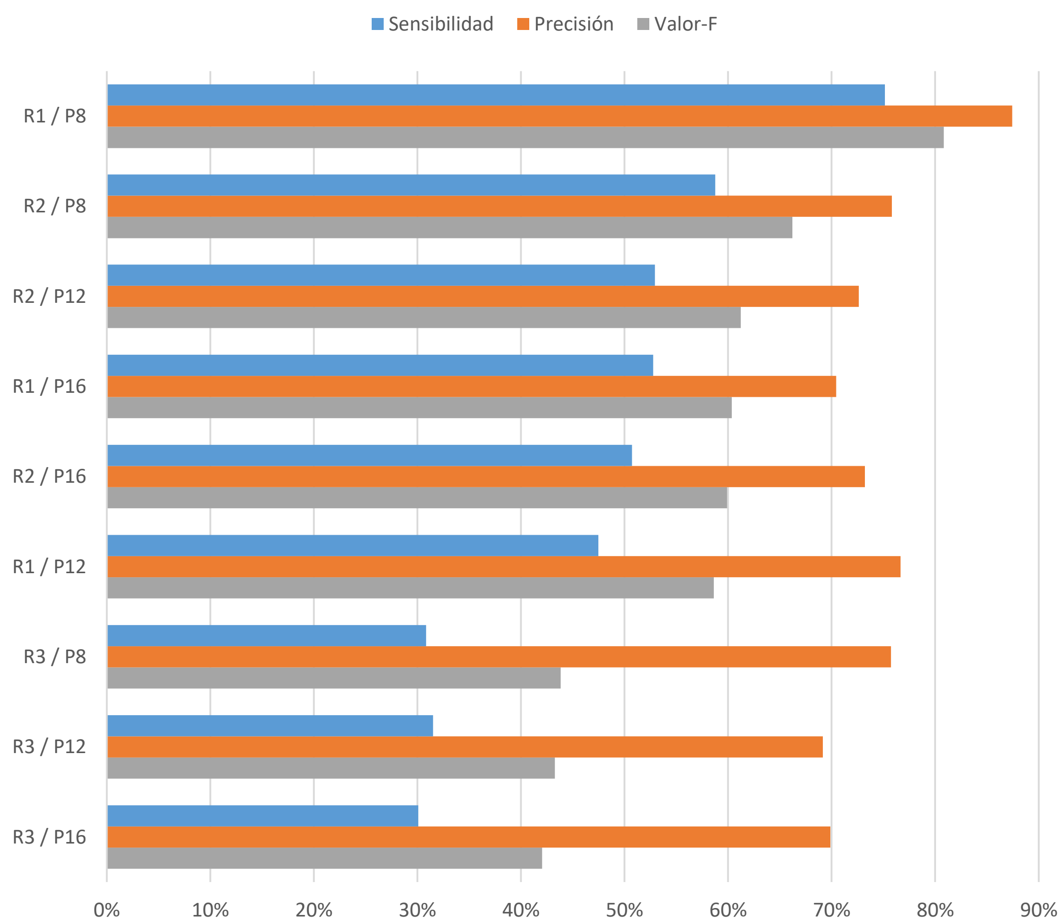


Figura 5.7: *Estadísticas utilizando diferentes vecindarios*

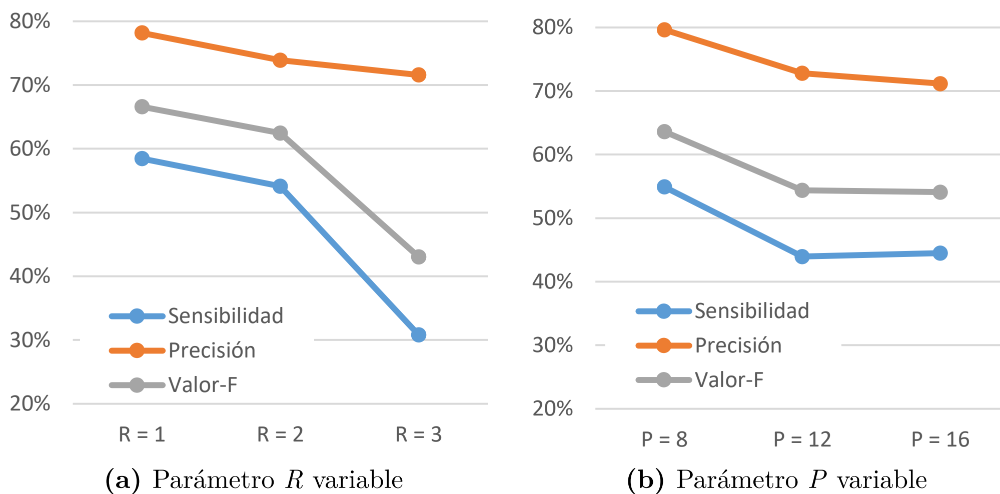
La gráfica muestra que los mejores resultados se obtienen con un vecindario de radio 1 y 8 vecinos, con un valor-F del 81 %, una precisión del 87 % y un sensibilidad del 75 %. Un vistazo general al gráfico muestra que el resto de vecindarios tienen un valor-F significativamente menor, partiendo desde el 66 % hasta el 42 % en el peor caso.

El siguiente mejor vecindario es el $P = 8/R = 2$, con un valor-F del 66 %. Tanto la precisión como la sensibilidad disminuyen, aunque esta última lo hace en mayor medida.

A continuación, los siguientes 4 vecindarios tienen unos valores-F muy similares, sobre el 60 %. En este grupo encontramos los vecindarios de 12 y 16 vecinos, tanto para radio 1 como 2. No hay grandes variaciones en su precisión (máxima diferencia del 7 %, con un promedio del 73 %) ni en su sensibilidad (máxima diferencia del 6 %, con un promedio del 51 %).

Por último, encontramos los tres vecindarios de radio 3, con unos valores-F muy similares en torno al 43 %. La sensibilidad se mantiene prácticamente constante para los tres vecindarios, sobre el 31 %. En cambio la precisión es ligeramente mejor utilizando 8 vecinos (76 %), frente a 12 y 16 vecinos (69 % y 70 %, respectivamente).

Una vez analizados qué vecindarios ofrecen los mejores resultados, se va a analizar la influencia del radio y del número de vecinos por separado (Figura 5.8). Para el análisis del radio, se ha calculado la media de los tres vecindarios con los distintos valores P (8, 12 y 16) para cada uno de los 3 radios. Se ha procedido de forma análoga para el análisis del número de vecinos.



(a) Parámetro R variable **(b)** Parámetro P variable

Figura 5.8: *Comportamiento de los parámetros* P *y* R *por separado*

En la subfigura 5.8a analizamos la influencia del radio, y vemos que según este aumenta, disminuye el valor-F, por lo que las prestaciones se ven deterioradas. La tendencia de la precisión es decreciente con una pendiente pequeña y constante, pero la sensibilidad (y como consecuencia el valor-F) comienza con una pendiente descendente similar a la de la precisión, pero en el cambio de radio 2 a radio 3, la pendiente aumenta drásticamente, bajando de una sensibilidad del 54 % al 31 %. Por lo tanto, la sensibilidad se ve especialmente afectada por el aumento del radio.

En la subfigura 5.8b se analiza cómo afecta el número de vecinos a la detección de gasas. El valor-F disminuye a la vez que el número de vecinos aumenta, tal y como pasaba con el radio (aunque el valor-F para 12 y 16 vecinos es el mismo, podemos considerar que realmente se trata de una tendencia descendente). La precisión disminuye con el aumento de vecinos, de forma más marcada en el salto de 8 a 12. La sensibilidad disminuye en un 11 % cuando pasamos de 8 a 12, mientras que de 12 a 16 vecinos, aumenta un 1 %. Al igual que con el valor-F, podemos considerar igualmente una tendencia descendente, ya que el aumento en sensibilidad es despreciable frente al descenso.

Después de analizar los datos de las 3 gráficas, concluimos que resulta conveniente mantener tanto el radio como el número de vecinos en un valor bajo, concretamente utilizado un vecindario $P = 8/R = 1$, que arroja los mejores resultados con una amplia diferencia. El objetivo que motivó el aumento del radio y del número de vecinos fue ampliar la zona de acción del operador, aunque se ha demostrado que, contra-intuitivamente, el operador funciona mejor en un entorno de microtextura y no de macrotextura.

5.6. Análisis de las variantes 2, 3 y 4

Una vez analizados los diferentes parámetros, y habiendo encontrado sus valores óptimos, procedemos a determinar los umbrales óptimos para el resto de variantes. En todas ellas, el vecindario será $P = 8/R = 1$, el operador utilizado será mLBP y el tamaño de tesela será de 100x100 píxeles. Dado que la variante 1 ya ha sido analizada en las secciones anteriores, comenzaremos por la variante 2. Por último, en la sección 5.7 compararemos los resultados de las 4 variantes.

5.6.1. Variante 2. "LBP y varianza por separado"

En esta variante será necesario establecer dos umbrales, el de comparación de histogramas ULBP e histogramas de varianza. El umbral ULBP óptimo es el mismo que en la variante 1, 10.

En la figura 5.9 graficamos los resultados obtenidos utilizando únicamente la varianza, sin ninguna contribución por parte de los códigos ULBP. Al igual que en la sección 5.1 (Método de análisis), buscaremos un umbral cuya precisión esté aproximadamente entre el valor máximo y el valor de corte. En la tabla 5.4 mostramos los porcentajes de los 3 puntos significativos.

Por lo tanto, el umbral óptimo utilizando únicamente la varianza se encuentra en un valor de 200, con una sensibilidad del 67 % y una precisión del 88 %.

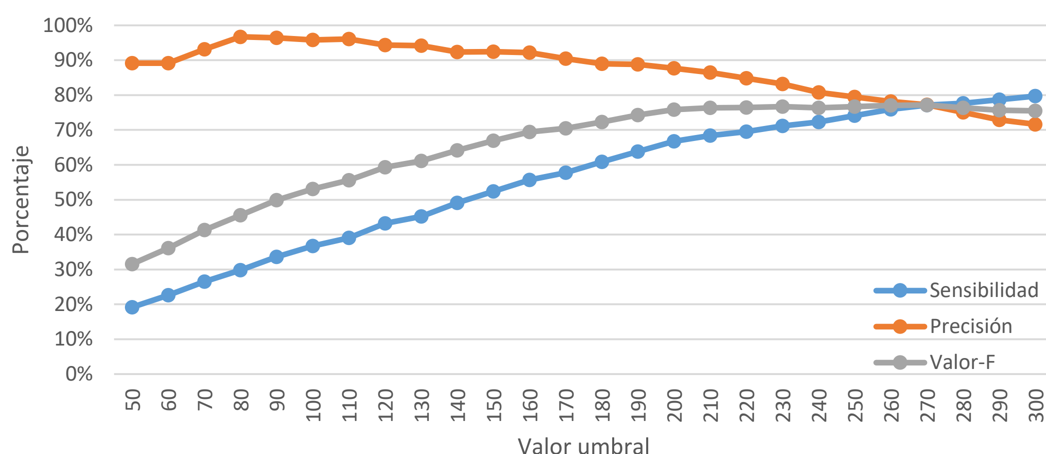


Figura 5.9: *Influencia del umbral de la varianza*

Umbral	Sensibilidad	Precisión
80	30 %	97 %
200	**67 %**	**88 %**
270	77 %	77 %

Tabla 5.4: *Puntos significativos de la gráfica de umbrales*

Una vez calculados ambos umbrales por separado, hay que comprobar cómo se comportan de forma conjunta. Para ello tenemos dos criterios de aceptación de gasas:

- ULBP *and* VAR: es necesario que ambos operadores coincidan en la elección para considerar como gasa a una tesela.

- ULBP *or* VAR: una tesela será considerada gasa si uno de los dos operadores (o ambos) lo consideran.

Utilizando los umbrales óptimos, 10 para ULBP y 200 para la varianza, obtenemos los siguientes resultados (Figura 5.10).

En la gráfica vemos que utilizando el operador *and* (&) obtenemos una sensibilidad bastante baja y una precisión muy alta. En cambio, con el operador *or* (|) conseguimos una sensibilidad muy alta pero una precisión menor. El valor-F es significativamente mayor utilizando el operador *or* respecto al *and*.

A la vista de los resultados, la utilización del operador *and* hace que la precisión sea muy grande, ya que ambos operadores tienen que coincidir en la decisión para poder considerar una tesela como gasa. Este aumento en precisión es obtenido a costa de una drástica disminución en la sensibilidad, haciendo que su valor-F, disminuya. El operador *and* resultaría adecuado en el supuesto de que priorizáramos la precisión sin importar la sensibilidad, lo cual no es el caso.

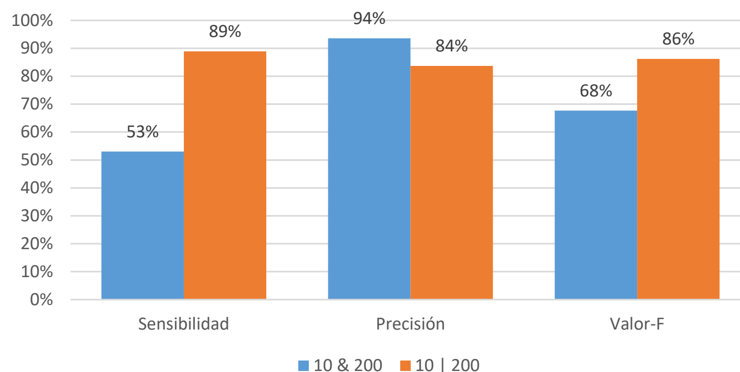


Figura 5.10: *Resultados utilizando los dos criterios de aceptación de gasas*

Por otro lado, el operador *or* mantiene unos valores muy altos tanto de precisión como de sensibilidad, dando lugar a un valor-F alto. La disminución en la precisión nos indica que la proporción entre falsos positivos y los verdaderos positivos ha aumentado, lo cual es razonable dado que la política de aceptación de la gasa es menos restrictiva. Esta política menos restrictiva es también la responsable de aumentar la sensibilidad, ya que teselas no identificadas por un operador ahora pueden ser identificadas por el otro. Es por ello que en nuestra aplicación utilizaremos el operador *or*.

Aunque los umbrales elegidos son óptimos para un uso aislado del operador ULBP o el operador VAR, no resultan los más adecuados para cumplir las prioridades que se han impuesto. De hecho, utilizando el operador *or* con esos valores umbrales, se está priorizando la sensibilidad sobre la precisión, y buscamos justamente lo contrario. Por ello, se realizarán pruebas para todas las combinaciones de umbrales posibles, utilizando umbrales ULBP del 5 al 10, y umbrales VAR del 150 al 200, lo que hacen un total de 36 umbrales diferentes (Figura 5.11).

Dado que resulta bastante complicado determinar el mejor umbral visualizando la gráfica de la figura 5.11, en la figura 5.12 se han descartado aquellos umbrales cuyo porcentaje de sensibilidad sea mayor al de precisión, y sólo se han mantenido aquellos con valor-F máximo ($86 \pm 0{,}5\,\%$). Con estas dos condiciones, hemos reducido el número de umbrales de 36 a 13.

Podemos observar que nos estamos moviendo en unos rangos de porcentaje muy altos, entre el 82 % y el 90 %, es por ello que el umbral elegido ha sido el 8 | 160 (umbral ULBP | umbral VAR), en el que hay un máximo de precisión (90 %), y aunque sea prácticamente un mínimo en la sensibilidad, tiene un valor del 83 %, lo cual es bastante alto.

A continuación mostramos 4 imágenes en las que observamos cómo se complementan estos operadores (Figura 5.13). El código de colores es el siguiente:

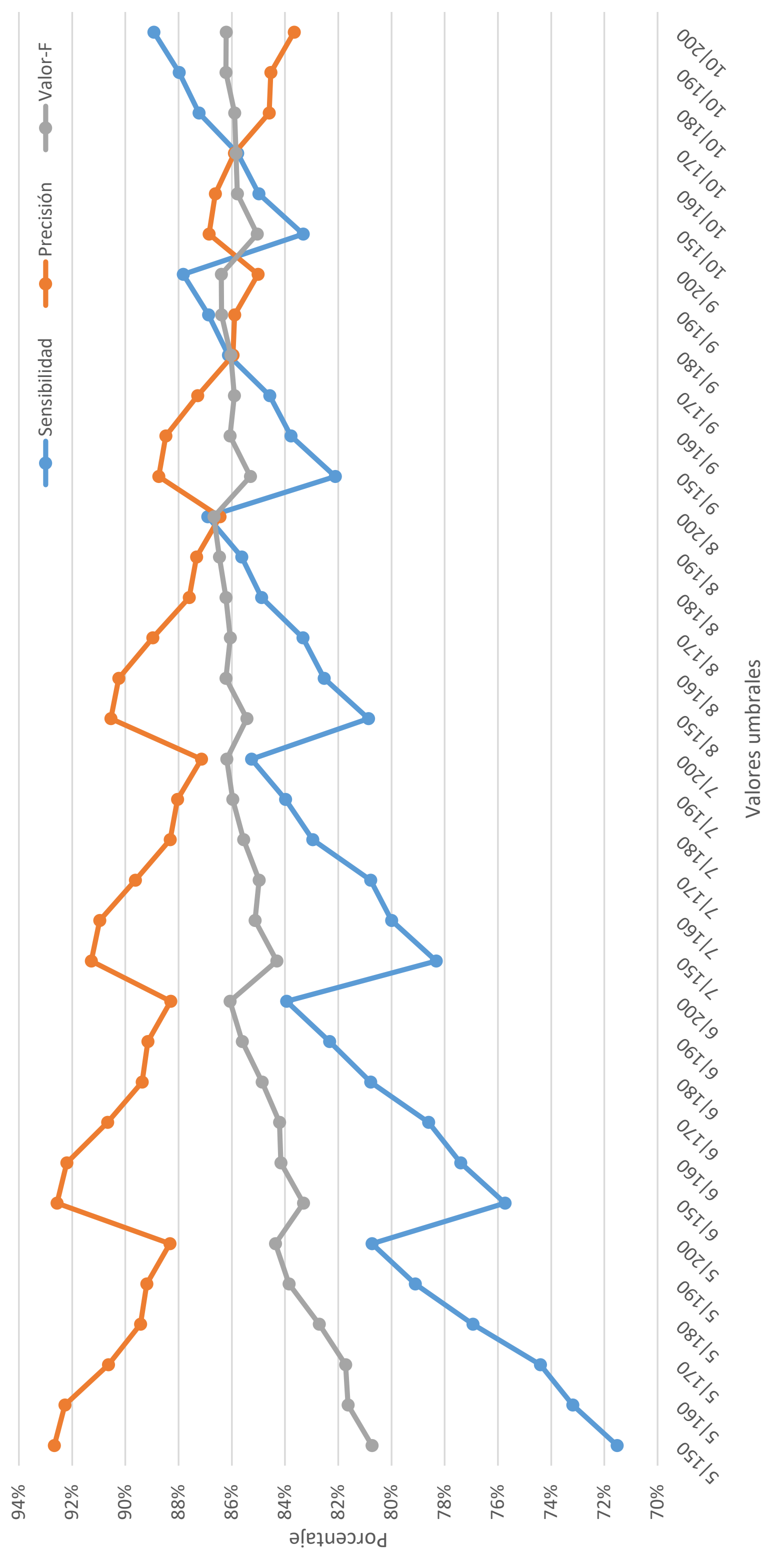


Figura 5.11: *Resultados para diferentes combinaciones de umbrales ULBP y VAR*

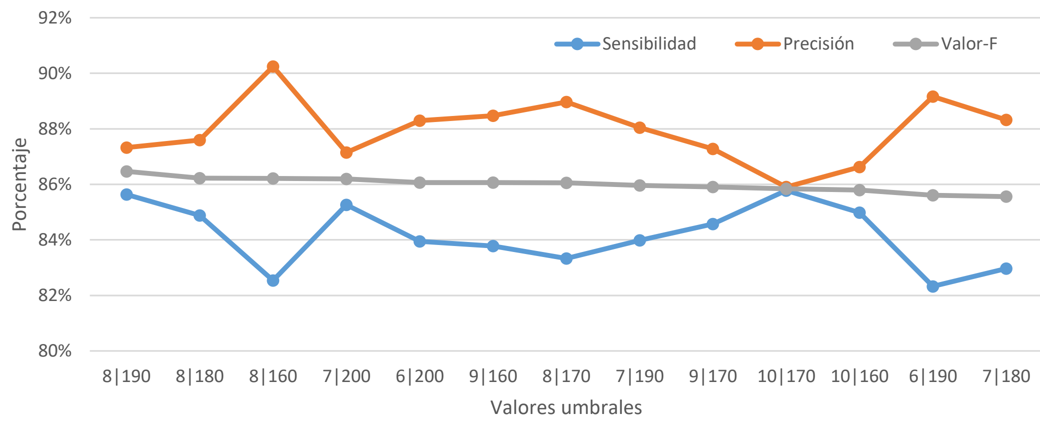
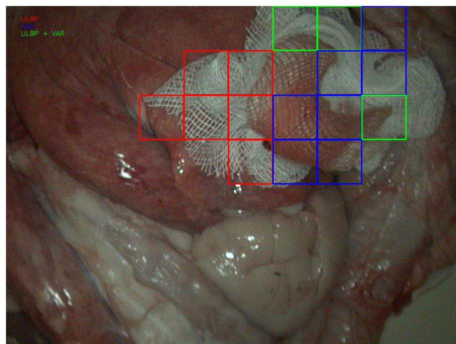
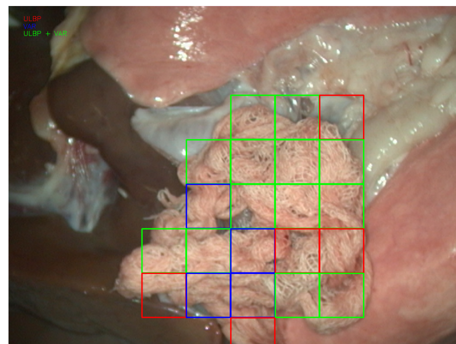


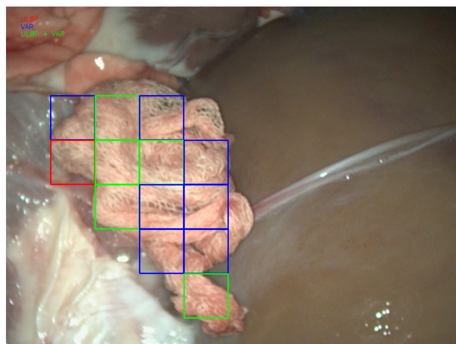
Figura 5.12: *Resultados para un grupo reducido de umbrales*



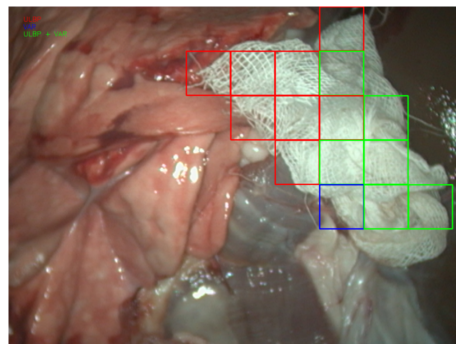
(a) Imagen nº 11



(b) Imagen nº 22



(c) Imagen nº 28



(d) Imagen nº 32

Figura 5.13: *Ejemplo de ejecución de la segunda variante del programa utilizando los umbrales 8 / 160*

- Rojo: gasa detectada únicamente por el operador ULBP.
- Azul: gasa detectada únicamente por el operador VAR.
- Verde: gasa detectada por ambos operadores (corresponden a las gasas obtenidas si el criterio de aceptación hubiera sido *and* en vez de *or*).

Observamos que la delimitación de la gasa es muy buena (alta sensibilidad), y aunque hay teselas no detectadas, gran parte de ellas sí que son identificadas correctamente. Además, el número de falsos negativos es muy bajo (alta precisión). El código de colores nos permite comprobar visualmente cómo se complementan ambos operadores, ya que aunque hay zonas en las que ambos operadores detectan la gasa (verde) hay otras en las que sólo uno de ellos lo hace (azul y rojo).

5.6.2. Variante 3. "LBP y varianza combinado, método 1"

En esta variante será necesario fijar un único umbral, ya que combina la información ULBP y de la varianza en un mismo histograma. En la figura 5.14 se muestra la gráfica con los resultados para diferentes umbrales.

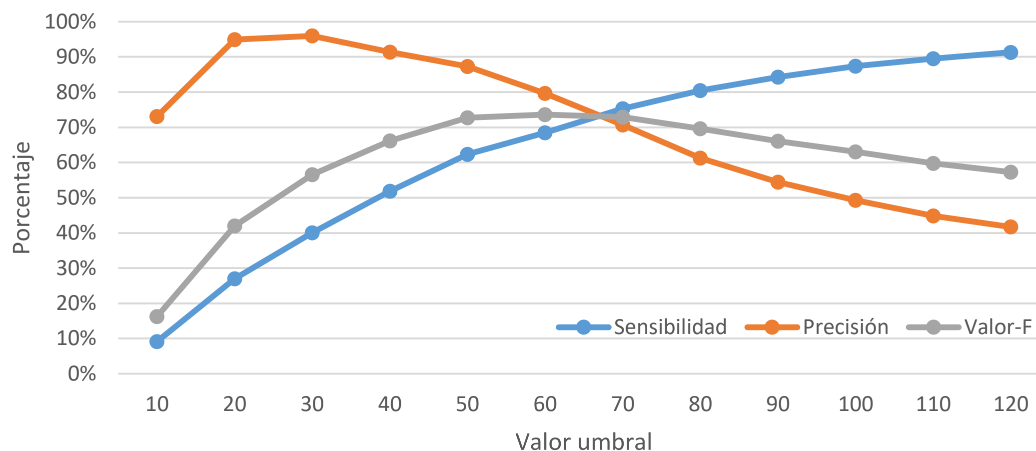
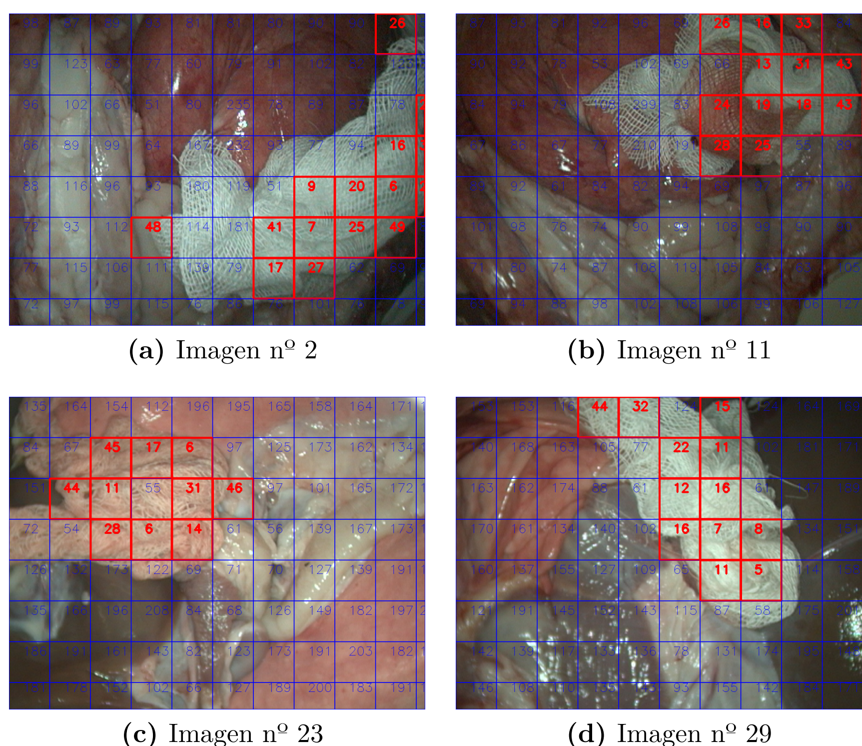


Figura 5.14: *Resultados para diferentes umbrales utilizando el primer método de la varianza combinada con ULBP*

Al igual que hemos hecho hasta ahora, tomamos el punto intermedio entre el umbral de máxima precisión y el umbral en el que el valor de la sensibilidad y la precisión es igual. El umbral que se utilizará será el 50, con una sensibilidad del 62 %, una precisión del 87 % y un valor-F del 73 %.

A continuación mostramos 4 imágenes laparoscópicas que ilustran el comportamiento de esta variante (Figura 5.15).



(a) Imagen nº 2 **(b)** Imagen nº 11

(c) Imagen nº 23 **(d)** Imagen nº 29

Figura 5.15: *Ejemplo de ejecución de la tercera variante del programa utilizando el umbral 50*

Las imágenes muestran que aunque gran parte de la gasa es identificada correctamente, hay regiones significativamente grandes en las que no es capaz de detectarla correctamente. Aún así, por ejemplo, en la subfigura 5.15c, la tesela en el centro de la gasa tiene un valor de comparación de 55, por lo tanto no se aleja demasiado del umbral elegido, mientras que zonas del fondo tienen valores bastante superiores. Por otra parte, el número de falsos negativos es bajo.

Este comportamiento efectivamente corresponde a una baja sensibilidad y una buena precisión

5.6.3. Variante 4. "LBP y varianza combinado, método 2"

Por último, esta cuarta variante integra los valores ULBP y de la varianza en un mismo histograma siguiendo otro método. Los resultados utilizando diferentes umbrales se muestran a continuación (Figura 5.16).

La precisión máxima la obtenemos para un valor umbral de 4, y el punto de corte para un umbral de 9. El porcentaje medio se da aproximadamente para un valor umbral 6, que será el seleccionado, con una sensibilidad del 54 %, una precisión del 74 % y un valor-F del 62 %.

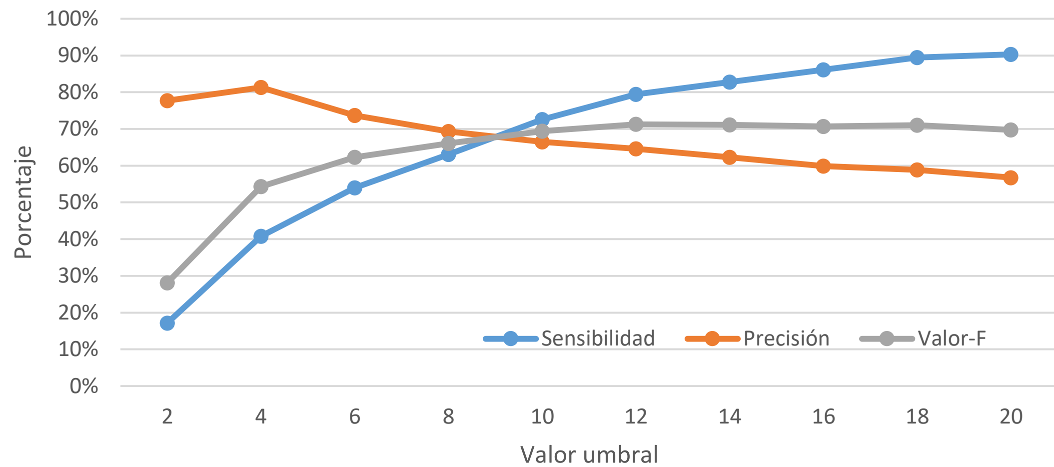
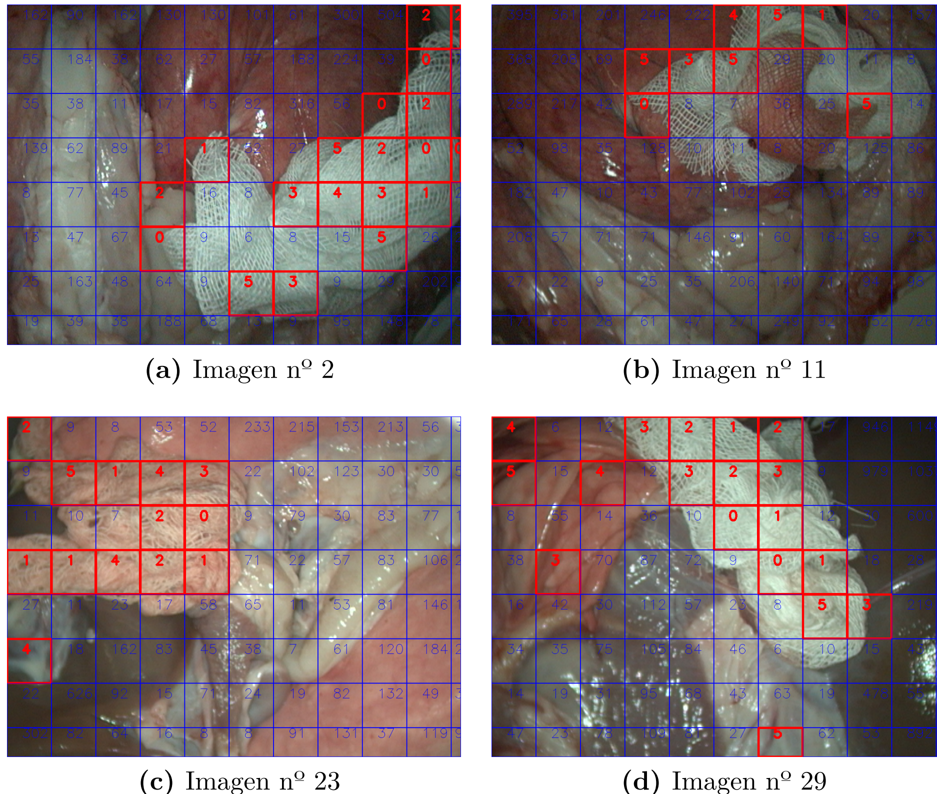


Figura 5.16: *Resultados para diferentes umbrales utilizando el segundo método de la varianza combinada con ULBP*

Dada la similitud entre las variantes 3 y 4, utilizaremos las mismas imágenes laparoscópicas a modo de comparación (Figura 5.17).

Observamos que aparecen un número elevado de falsos negativos (baja precisión) y una delimitación poco completa de la gasa (baja sensibilidad).



(a) Imagen nº 2 **(b)** Imagen nº 11

(c) Imagen nº 23 **(d)** Imagen nº 29

Figura 5.17: *Ejemplo de ejecución de la cuarta variante del programa utilizando el umbral 6*

5.7. Comparación de resultados

Una vez analizadas todas las variantes y todos los parámetros, y habiendo encontrado sus valores óptimos, podemos proceder a comprar los resultados de las 4 variantes.

En los 3 diagramas de barras de la figura 5.18 se muestran los porcentajes medios de sensibilidad, precisión y valor-F (subfigura 5.18a), a continuación desglosamos los porcentajes de precisión en imágenes laparoscópicas limpias, parcialmente manchadas con sangre y totalmente empapadas en sangre u otros fluidos (subfigura 5.18b) y por último desglosamos de igual manera los porcentajes de sensibilidad (subfigura 5.18c).

Porcentajes medios

Comenzaremos la comparación por los porcentajes medios. La medida que se ha utilizado a lo largo de este capítulo para realizar comparaciones es el valor-F, una vez fijado un umbral que priorice la precisión frente a la sensibilidad. Atendiendo a este valor, encontramos que la variante que mejor funciona es "LBP y varianza por separado" (variante 2), después "LBP" (variante 1), a continuación "LBP y varianza combinado, método 1" (variante 3) y por último "LBP y varianza combinado, método 2" (variante 4).

Se puede deducir que la varianza correctamente tratada por separado aporta mucha información y arroja unos resultados excelentes junto al operador LBP, pero en cambio, cuando se combina la información de la varianza y del operador LBP para un análisis conjunto, perdemos esa sinergia y obtenemos unos resultados mucho peores.

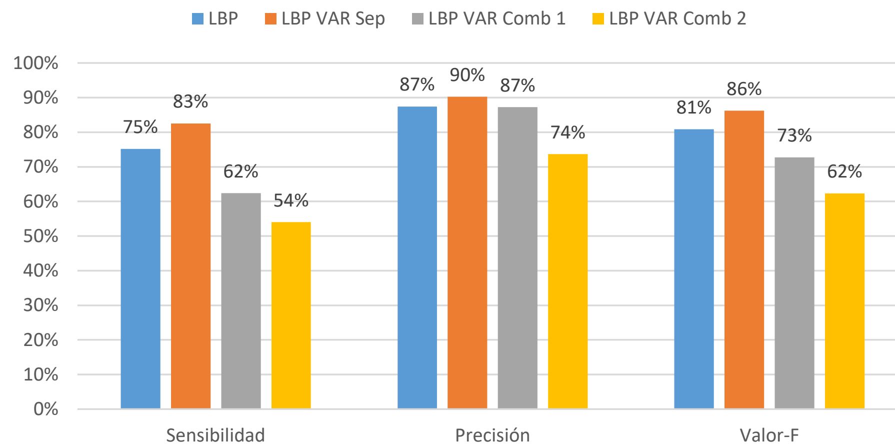
También es remarcable el excelente comportamiento del operador LBP por separado, sin aportaciones de la varianza, obteniendo unos porcentajes de precisión y sensibilidad muy buenos con un coste computacional muy bajo.

Si nos fijamos en la precisión y la sensibilidad de las diferentes variantes, observamos que los valores de precisión para las tres primeras variantes son prácticamente iguales, aunque las mayores diferencias se encuentran en la sensibilidad, que disminuye en el orden mencionado anteriormente

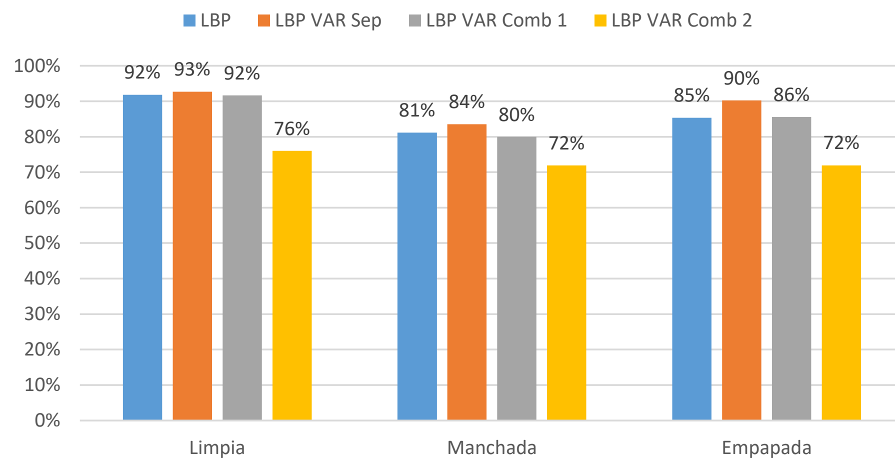
La cuarta variante es con diferencia la que peor resultados arroja tanto en términos de precisión como de sensibilidad.

Porcentajes de precisión desglosados

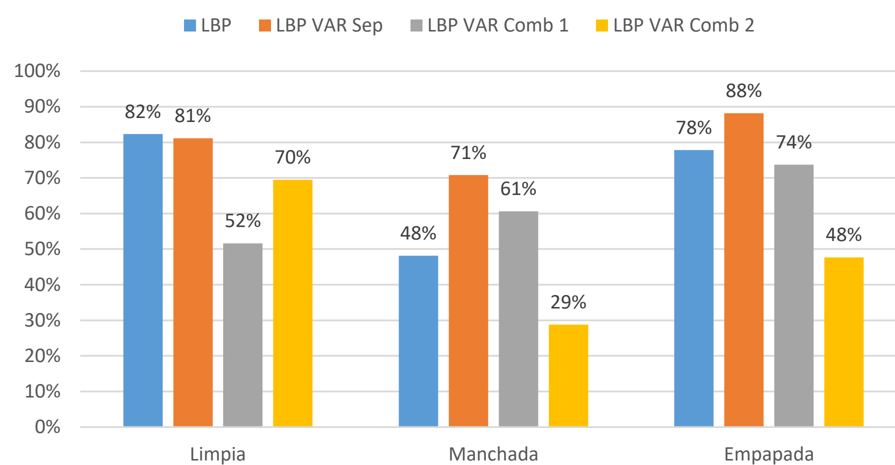
Analizando los porcentajes de precisión para diferentes escenarios, vemos como siguen el mismo patrón que con los valores medios. Las tres primeras variantes arrojan valores muy parecidos (aunque siempre se mantiene superior la segunda variante), mientras que la cuarta claramente se comporta peor.



(a) Porcentajes medios



(b) Porcentajes de precisión desglosados



(c) Porcentajes de sensibilidad desglosados

Figura 5.18: *Comparación de resultados utilizando las diferentes variantes*

Descartando la cuarta variante, y hablando en términos de las otras tres, observamos que la mejor precisión se da cuando la gasa está limpia (fenómeno que es razonable, ya que gran parte de la imágenes patrón está integrado por gasas limpias), a continuación se encuentran las gasas completamente empapadas en sangre y otros fluidos y por último las gasas parcialmente manchadas de sangre.

Aunque observemos esta disminución de precisión en algunos escenarios, podemos considerar que la precisión se mantiene razonablemente estable, ya que las diferencias en los porcentajes no son muy grandes.

Porcentajes de sensibilidad desglosados

Por último, procedemos a analizar la sensibilidad según el tipo de imagen laparoscópica.

Comenzando por la primera variante, es la que obtiene la mejor sensibilidad al analizar gasas limpias (aunque solo un 1 % por encima de la segunda variante), y un buen comportamiento para gasas empapadas, en cambio, la sensibilidad cae drásticamente para gasas parcialmente manchadas de sangre.

La segunda variante tiene mayor sensibilidad para gasas empapadas, después gasas limpias y por último gasas manchadas, obteniendo los mayores porcentajes excepto para gasas limpias.

La tercera y cuarta variante arrojan unos resultados muy interesantes si se analizan conjuntamente. La sensibilidad de la variante 3 es mayor para gasas manchadas y empapadas, mientras que es menor para gasas limpias, en comparación con la variante 4. Lo interesante de esta comparación es que ambas variantes combinan la misma información de la varianza y del operador LBP, aunque siguiendo diferentes métodos, y en cambio, los resultados son totalmente opuestos ante la presencia de fluidos.

Globlamente observamos cómo los porcentajes de sensibilidad no se mantienen tan estables como la precisión a la hora de analizar diferentes tipos de imágenes laparoscópicas. Dentro de esta falta de estabilidad, la variante que mantiene sus valores más constantes (71 %, 81 % y 88 %), es la segunda variante, "LBP y varianza por separado".

Si nos fijamos únicamente en la variante 1 y 2, cuya única diferencia es que la segunda variante cuenta con la información adicional de la varianza, podemos concluir que la sensibilidad al analizar imágenes laparoscópicas con gasas manchadas o empapadas se ve significativamente mejorada al utilizar la varianza conjuntamente con el operador LBP.

5.8. Análisis de tiempos

Uno de los requisitos del programa era un coste computacional bajo para poder ser utilizado en tiempo real. A continuación mostraremos los tiempos de ejecución para la primera y segunda variante, que son las que mejor funcionan y por lo tanto las candidatas a ser implementadas en un sistema real.

Para realizar las pruebas, el programa ha analizado las 26 imágenes laparoscópicas. El tiempo se ha medido desde su lectura hasta que devuelve los valores de comparación. La ejecución se ha repetido 5 veces y se ha calculado su media.

Las pruebas se han realizado en un ordenador portátil con un procesador Intel Core i7-3630QM (4 núcleos a 2.4GHz con *Turbo Boost* hasta 3.4GHz, 8 hilos) y 8GB de RAM DDR3, corriendo Ubuntu 16.04 en una máquina virtual que permite utilizar el 100 % de la capacidad del procesador y tiene asignada 4GB de RAM.

En la tabla 5.5 mostramos los resultados de ambos programas.

Ejecución	Media (ms)	Imágenes por segundo
1	31.4	31.9
2	29.3	34.1
3	31.3	31.9
4	31.5	32.8
5	30.4	32.9
	30.6	**32.7**

(a) "LBP"

Ejecución	Media (ms)	Imágenes por segundo
1	64.1	15.6
2	64.6	15.5
3	64.5	15.5
4	62.9	15.9
5	63.0	15.9
	63.8	**15.7**

(b) "LBP y varianza por separado"

Tabla 5.5: *Resultados de tiempos de ejecución*

En la segunda columna (media) mostramos el tiempo medio en milisegundos que tarda en analizar una imagen laparoscópica y en la tercera columna, el número de imágenes que sería capaz de analizar en un segundo a ese ritmo. En la última fila, mostramos en negrita la media después de realizar 5 ejecuciones del programa.

Vemos que la variante "LBP" es capaz de analizar 32.7 imágenes por segundo, frente a las 15.7 imágenes por segundo de la variante "LBP y varianza

por separado". Una cámara convencional graba a una tasa de 30 fotogramas por segundo, por lo tanto, utilizando la primera variante seríamos capaces de analizar todos los fotogramas, mientras que utilizando la segunda variante sólo seríamos capaces de analizar la mitad de estos.

Aún así, ambas variantes serían válidas para una aplicación en tiempo real, ya que incluso analizar la presencia de gasas menos veces de 15 veces por segundo permitiría mantener un control de estas.

5.9. Conclusiones

Una vez finalizado el análisis de resultados, podemos concluir que las mejores variantes que funcionan son la "LBP" y "LBP y varianza por separado". Las otras dos variantes fueron desarrolladas buscando una comparación de histogramas más sencilla y una reducción del coste computacional. El primer método nos permitía analizar un único histograma aunque teníamos que seguir calculando el vector de cuantificación de la varianza, lo cual añade tiempo de ejecución. En el segundo método, se eliminó la necesidad de calcular esa cuantificación, reduciendo el tiempo necesario para analizar cada tesela.

En cambio, se ha demostrado que la varianza y la información ULBP debe analizarse por separado para no perder información de ninguna de las partes.

Respecto al tamaño del vecindario, en primera instancia parece que es más sencillo encontrar la gasa si su área de acción es más grande, ya que la textura de la gasa estará mejor definida. En cambio, los resultados muestran que es mejor trabajar a un nivel de microtextura y no de macrotextura.

Finalmente, se han conseguido unos porcentajes de precisión y sensibilidad muy elevados, especialmente para la segunda variante. Además, dejando atrás los porcentajes, en ninguna imagen laparoscópica se ha dejado de encontrar la gasa (independientemente de la mejor o peor delimitación), que es básicamente el objetivo del programa, determinar la presencia o no de gasas.

Capítulo 6

Conclusiones y líneas futuras

6.1. Conclusiones

El objetivo principal del presente TFG era desarrollar un programa que fuera capaz de analizar imágenes laparoscópicas y encontrar en estas las gasas presentes. También se marcaron unos objetivos intermedios, que a continuación recapitularemos para verificar su cumplimiento.

El primero de ellos fue realizar un estudio de las diferentes técnicas para el análisis de texturas. Hay diferentes enfoques y diferentes técnicas, pero el operador elegido para el análisis de texturas ha sido el *Local Binary Pattern*, ya que ofrece muy buenos resultados manteniendo un coste computacional muy bajo.

El siguiente objetivo era comprobar la efectividad de las variantes del operador LBP. Para ello, se realizaron pruebas con diferentes variantes y vecindarios: operador LBP, operador mLBP, utilización o no de patrones uniformes, diferente radio y diferente número de vecinos.

Los resultados mostraron que el operador que mejor funcionaba era el mLBP utilizando patrones uniformes, y con un vecindario de radio 1 y 8 vecinos.

A continuación, nos planteábamos detectar las gasas en diferentes condiciones de sangrado y orientación. La utilización de patrones uniformes permite identificar las gasas en diferentes orientaciones. Por otro lado, la detección de gasas en diferentes condiciones de sangrado se ha conseguido gracias a un buen análisis de texturas, que consigue detectar la gasa aunque su apariencia sufra cambios. Además, gracias a la inclusión de la varianza, conseguimos que la sensibilidad mejore notablemente en gasas manchadas y empapadas.

El penúltimo objetivo era utilizar la varianza conjuntamente con el operador LBP, para obtener más información acerca de la textura. Se desarrollaron 3 variantes que integraban la información de la varianza junto con la del ope-

rador LBP, siguiendo diferentes métodos. Tras el análisis de estas 3 variantes, se comprobó que la mejor forma de analizar la información del operador LBP y la varianza es por separado, ya que cuanto se intentan combinar en un mismo histograma, los resultados son significativamente peores. La varianza ha resultado especialmente útil en imágenes parcialmente manchadas de sangre y totalmente empapadas en sangre y otros fluidos, mejorando la sensibilidad. También mejora la precisión en cualquiera de los 3 estados de la gasa, aunque no tan significativamente.

Por último, el programa debía de permitir su uso en tiempo real. Gracias a la eficiencia computacional del operador LBP y a la optimización del código del programa, obtenemos un tiempo medio de análisis de la imagen de 15.7ms para la variante que utilizado sólo el operador LBP, y 32.7ms para la variante que utiliza el operador LBP y la varianza por separado, siendo tiempos perfectamente aptos para un uso en tiempo real.

Podemos concluir que todos los objetivos secundarios se han conseguido, y como consecuencia, también se ha conseguido el objetivo principal de este TFG, la identificación de gasas quirúrgicas en imágenes laparoscópicas.

6.2. Líneas futuras

Una vez finalizado el Trabajo Fin de Grado, y habiendo cumplido los objetivos propuestos en el primer capítulo, se pasa a enumerar diferentes vías que se pueden seguir para continuar el desarrollo del programa de detección de gasas.

- Implementación tiempo real: tal y como esta programado el sistema de detección de gasas, únicamente es capaz de analizar una imagen de entrada seleccionada por el usuario. El siguiente paso en su desarrollo es tomar como origen de datos una cámara conectada al ordenador y que analice los fotogramas en tiempo real. Según la variante elegida y la tasa de fotogramas a la que grabe la cámara, se podrán analizar todos los fotogramas o únicamente una parte de ellos.

- Creación de un mapa de gasas: una vez implementado el análisis de imágenes en tiempo real, el siguiente paso es crear un mapa que combine las imágenes laparoscópicas obtenidas por la cámara. Este mapa deberá ampliarse a medida que el cirujano acceda a nuevas zonas, y actualizarse cuando esté en zonas ya presentes en el mapa. De esta forma, se puede hacer un seguimiento del número y localización de las gasas que hay dentro de un paciente en un determinado instante.

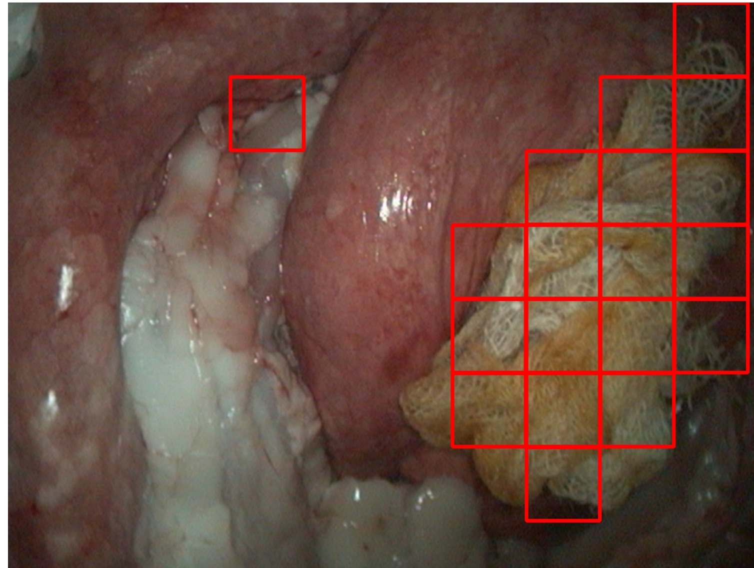


Figura 6.1: *Ejemplo de un falso positivo aislado*

- Morfología[1] para eliminar falsos positivos aislados (ejemplo en la figura 6.1): para ello, hay que crear una imagen binaria que contenga tantos píxeles como teselas tiene la imagen original. De estos píxeles solo estarían activos aquellos que se han clasificado como gasa. Una operación de apertura morfológica eliminaría los píxeles aislados o débilmente conectados sin afectar a las regiones significativas. Una forma de hacer esta operación mas robusta sería esperar a tener 2 fotogramas consecutivos, después se generarían las 2 imágenes binarias y se aplicaría una apertura con un elemento estructural tridimensional de 3x3x2 píxeles. De esta forma, cualquier región detectada como gasa tendría que tener una extensión mínima 3x3 y además haber aparecido en el fotograma anterior para mantenerse como tal. Descartaríamos, por lo tanto, el falso positivo únicamente cuando este presente en las dos imágenes. Esto induciría un retardo de un fotograma a los resultados, lo cual, dado el bajo tiempo de computación, no sería muy significativo. Aún así, un problema de este enfoque es que al haber movimiento en la cámara endoscópica, es probable que hubiera que alinear ambas imágenes para poder comparar las teselas activas, aumentando el tiempo de ejecución.

- Paralelización del programa: una manera de mejorar el tiempo de ejecución es hacer uso del procesamiento multinúcleo, de forma que puedan

[1] La morfología matemática es una herramienta muy utilizada en el procesamiento de imágenes. Las operaciones morfológicas pueden simplificar los datos de una imagen, preservar las características esenciales y eliminar aspectos irrelevantes. Las operaciones fundamentales del procesamiento morfológico son la dilatación y la erosión.

realizarse diferentes operaciones al mismo tiempo. Por ejemplo, una vez cargada la imagen laparoscópica, se puede dividir la ejecución en cálculo y comparación del histograma LBP y en el cálculo y comparación del histograma de la varianza, ya que no dependen el uno del otro. Una de las librerías más interesantes que existen para la programación de modelos multinúcleo es *Intel Threading Building Blocks* (TBB). TBB es una librería para el lenguaje de programación C++ con un alto nivel de abstracción, ofreciendo un rendimiento muy parecido en comparación con otras técnicas de programación de bajo nivel. Esta librería no define el paralelismo en términos de hilos[2], sino en tareas de grano fino[3]. Las tareas permiten al programador abstraerse del manejo y control del paralelismo para concentrase únicamente en el paralelismo lógico. Las tareas especificadas por el usuario se ejecutarán en diferentes hilos, según la disponibilidad de estos. El procesamiento multinúcleo no se ha implementando en este TFG, pero se ha demostrado su eficacia [4].

[2]En sistemas operativos, un hilo de ejecución es la unidad de procesamiento más pequeña que puede ser planificada. Normalmente, por cada núcleo del procesador hay dos hilos (empleando la técnica *multi-threaded*), aunque también puede darse el caso de tener el mismo número de hilos que de núcleos.

[3]En computación paralela, una aplicación muestra un paralelismo de grano fino si sus subtareas deben comunicarse muchas veces por segundo, paralelismo de grano grueso si no se comunican muchas veces por segundo y paralelismo muy grosero si nunca o casi nunca se tienen que comunicar.

Bibliografía

[1] Alegre, E., Pajares, M., & Escalera, A. (2016). *Conceptos y métodos en visión por computador*(pp. 115-117). España: CEA.

[2] *Aplicaciones / Visión artificial*. (2018). Infaimon.com. Recuperado en Abril 2018, a partir de `https://www.infaimon.com/es/menu/aplicaciones`

[3] Cáceres, J. (2006). La visión artificial y las operaciones morfológicas en imágenes binarias. *Campus Multidisciplinar En Percepción E Inteligencia, 2*, 5.

[4] De la Fuente, E., Trespaderne, F., Santos, L., Fraile, J., & Turiel, J. (2017). Parallel computing for real time gauze detection in laparoscopy images. *2017 2nd International Conference On Bio-Engineering For Smart Technologies (Biosmart)*. doi:10.1109/biosmart.2017.8095328

[5] *Digital images and image formats*. (2007). *Uio.no*. Recuperado en Abril 2018, a partir de `http://www.uio.no/studier/emner/matnat/math/MAT-INF1100/h07/undervisningsmateriale/kap8.pdf`

[6] *Digital Imaging Basics*. (2005). *Ischool.utexas.edu*. Recuperado en Abril 2018, a partir de `https://www.ischool.utexas.edu/technology/tutorials/graphics/digital/`

[7] *Endoscopia*. (2018). *Medlineplus.gov*. Recuperado en Abril 2018, a partir de `https://medlineplus.gov/spanish/ency/article/003338.htm`

[8] García, M., & Toribio, C. (2004). *El Futuro de la Cirugía Mínimamente Invasiva. Panelfenin.es*. Recuperado en Abril 2018, a partir de `http://panelfenin.es/uploads/fenin/documento_estudios/pdf_documento_18.pdf`

[9] Hariharan, D., & Lobo, D. (2013). Retained surgical sponges, needles and instruments. *The Annals Of The Royal College Of Surgeons Of England, 95*(2), 87-92. doi:10.1308/003588413x13511609957218

[10] Kim, H., Chung, T., Suh, S., & Kim, S. (2018). MR Imaging Findings of Paravertebral Gossypiboma. *American Journal Of Neuroradiology, 28*(4), 709-713.

[11] Lanfranco, A., Castellanos, A., Desai, J., & Meyers, W. (2004). Robotic Surgery. *Annals Of Surgery, 239*(1), 14-21. doi:10.1097/01.sla.0000103020.19595.7d

[12] Liang, H., & Weller, D. (2016). Edge-based texture granularity detection. *2016 IEEE International Conference On Image Processing (ICIP)*. doi:10.1109/icip.2016.7533023

[13] Mäenpää, T., & Pietikäinen, M. (2005). Texture analysis with local binary patterns. *Handbook Of Pattern Recognition And Computer Vision*, 197-216. doi:10.1142/9789812775320_0011

[14] *OpenCV library*. (2018). *Opencv.org*. Recuperado en Abril 2018, a partir de `https://opencv.org/`

[15] Pietikäinen, M., Hadid, A., Zhao, G., & Ahonen, T. (2011). *Computer vision using local binary patterns*(pp. 13-47). London: Springer.

[16] *Septicemia*. (2016). *Medlineplus.gov*. Recuperado en Abril 2018, a partir de `https://medlineplus.gov/spanish/ency/article/001355.htm`

[17] *Trócares laparoscópicos*. (2018). *Laparoscopica.es*. Recuperado en Abril 2018, a partir de `http://www.laparoscopica.es/instrumentos/trocar`

[18] Valero, R., Ko, Y., Chauhan, S., Schatloff, O., Sivaraman, A., & Coelho, R. et al. (2011). Cirugía robótica: Historia e impacto en la enseñanza. *Actas Urológicas Españolas, 35*(9), 540-545. doi:10.1016/j.acuro.2011.04.005

[19] *What does FDA regulate?*. (2018). *Fda.gov*. Recuperado en Abril 2018, a partir de `https://www.fda.gov/AboutFDA/Transparency/Basics/ucm194879.htm`

[20] Zhu, S., Guo, C., Wang, Y., & Xu, Z. (2005). What are Textons?. *International Journal Of Computer Vision, 62*(1-2), 121-143. doi:10.1007/s11263-005-4638-1

Anexo A

Código fuente: LBP

A.1. *main.cpp* ("LBP")

```
1  /***************************************************************************/
2  /**                         Trabajo Fin de Grado                          **/
3  /**                                                                       **/
4  /**               Detección de Gasas Quirúrgicas en imágenes              **/
5  /**                 laparoscópicas empleando la técnica LBP               **/
6  /**                                                                       **/
7  /**         Grado en Ingeníeria Electrónica Industrial y Automática       **/
8  /**                         Universidad de Valladolid                     **/
9  /**                                                                       **/
10 /** Autor: Álvaro Muñoz García                                            **/
11 /** Tutor: Eusebio de la Fuente López                                     **/
12 /***************************************************************************/
13 /** Proyecto: LBP                                                         **/
14 /** Archivo: main.cpp                                                     **/
15 /** Variante: LBP                                                         **/
16 /**                                                                       **/
17 /** Descripción: se trata del programa principal. Se manda elegir al      **/
18 /**     usuario una imagen laparoscópica (1-32) que se analizará. Para    **/
19 /**     ello, primero se carga los datos del patrón y la LUT. A           **/
20 /**     continuación, convertimos la imagen a B/N, calculamos la imagen   **/
21 /**     de medias, la imagen LBP (o mLBP) y por último la convertimos a   **/
22 /**     ULBP. Después, dividimos la imagen en tiles y comparamos el       **/
23 /**     histograma de cada tile con el histograma patrón. Por último,     **/
24 /**     sacamos por pantalla los resultados de la comparación de cada tile **/
25 /**                                                                       **/
26 /**     Si el cuadrado de un tile es rojo significa que se considera gasa, **/
27 /**     si no, será de color azul.                                        **/
28 /**                                                                       **/
29 /** Nota: debe ejecutarse previamente el programa "generador LUTs" y el   **/
30 /**     "generador de patrón".                                            **/
31 /***************************************************************************/
32
33 #include "opencv2/opencv.hpp"
34 #include "../../include/fichero.h"
35 #include "../../include/lbp.h"
36 #include "../../include/estadist.h"
37 #include "../../include/histo.h"
38 #include "../../include/compara.h"
39 #include "../../include/visualiza.h"
40
```

```
41 #define TAM_TILE 100          // Lado del cuadrado en el que troceamos la
       imagen
42 #define GRADO_SUPERP_TILE 0.0 // 0.0 no hay superposicion en los bloques
43 #define UMBRAL_HISTO_ULBP  10 // Valor a partir del cual se considera gasa
44
45 using namespace std;
46 using namespace cv;
47
48 int main()
49 {
50     double factorEscala=1.0; // Factor de reduccion de la imagen
51     int R=0, P=0;            // R: radio del vecindario, P: numero de vecinos
52     Mat histoPatronULBP;     // Mat::zeros(P+2, 1, CV_32F);
53     Mat imgColor;
54     vector<int> LUT;         // LUT para la transformacion lbp -> ulbp
55
56     cout << "Introducir NUMERO imagen (IMG00NUM.jpg): ";
57     string fileNumber;
58     cin >> fileNumber;
59
60     try
61     {
62         /// 1. Inicializacion de variables de acuerdo al patron almacenado en
               datosPatron.xlm
63         leeFicheroDatosPatron(factorEscala, R, P, histoPatronULBP);
64
65         /// 2. Inicializamos la variable LUT que contiene la look-up table que
                convierte valores LBP en ULBP.
66         leeFicheroLUT(LUT, P);
67
68         /// 3. Leemos la imagen elegida por el usuario
69         leeFicheroImagen(imgColor, fileNumber);
70     }
71
72     catch(int excepcion)
73     {
74         if(excepcion == ERROR_NO_FILE_PATRON)
75             cerr << "ERROR " << excepcion << " leyendo fichero de datos." <<
                   endl;
76         else if(excepcion == ERROR_NO_FILE_LUT)
77             cerr << "ERROR " << excepcion << " leyendo fichero LUT." << endl;
78         else if(excepcion == ERROR_NO_FILE_IMG)
79             cerr << "ERROR " << excepcion << " leyendo fichero Imagen." <<
                   endl;
80         exit(EXIT_FAILURE);
81     }
82
83     /// 4. Convertimos la imagen a color en B/N
84     Mat img;
85     cvtColor(imgColor, img, COLOR_BGR2GRAY);
86
87     /// 5. Calculamos la media de cada vecindario y lo almacenamos en la
           variable EX
88     Mat EX;
89     media<uchar>(img, EX, R, P);
90
91     /// 6. Calculamos la imagen mLBP (utiliza media en vez de pixel central)
92     Mat imgLBP;
93     mLBP(img, EX, imgLBP, R, P);
94     //LBP(img, imgLBP, R, P);
95
```

```
96      /// 7. Convertimos la imagen LBP "normal" en uniforme (ULBP) utilizando la
             LUT
97      Mat imgULBP;
98      if(imgLBP.type() == CV_8UC1)
99          LBP2ULBP<uchar>(imgLBP, LUT, imgULBP);
100     else
101         LBP2ULBP<int>(imgLBP, LUT, imgULBP);
102
103     /// 8. Realizamos la comparación de cada mosaico (tile) con el histograma
            patrón.
104     ///    Se almacenan los resultados en el vector tilesULBP (tilesVar no
            utilizado).
105     vector<int> tilesULBP, tilesVar;
106     tilesULBP = comparaHistosTiles(imgULBP,  histoPatronULBP, TAM_TILE*
            factorEscala, GRADO_SUPERP_TILE);
107
108     /// 9. Dibuja sobre la imagen de entrada los valores de la comparacion de
            histogramas
109     visualizaValoresEnTiles(imgColor, tilesULBP, UMBRAL_HISTO_ULBP, tilesVar,
            -1, TAM_TILE*factorEscala, GRADO_SUPERP_TILE);
110
111     return 0;
112 }
```

A.2. *main.cpp* ("LBP y varianza por separado")

```
/*************************************************************************/
/**                         Trabajo Fin de Grado                        **/
/**                                                                     **/
/**               Detección de Gasas Quirúrgicas en imágenes            **/
/**                laparoscópicas empleando la técnica LBP              **/
/**                                                                     **/
/**        Grado en Ingeníeria Electrónica Industrial y Automática      **/
/**                         Universidad de Valladolid                   **/
/**                                                                     **/
/** Autor: Álvaro Muñoz García                                          **/
/** Tutor: Eusebio de la Fuente López                                   **/
/*************************************************************************/
/** Proyecto: LBP                                                       **/
/** Archivo: main.cpp                                                   **/
/** Variante: LBP y varianza por separado                               **/
/**                                                                     **/
/** Descripción: se trata del programa principal. Se manda elegir al    **/
/**     usuario una imagen laparoscópica (1-32) que se analizará. Para  **/
/**     ello, primero se carga los datos del patrón y la LUT. A         **/
/**     continuación, convertimos la imagen a B/N, calculamos la imagen **/
/**     de medias, la imagen LBP (o mLBP) y por último la convertimos a **/
/**     ULBP. Después, calculamos la varianza de la imagen laparoscópica **/
/**     y la cuantificamos utilizando el vector obtenido en los datos del **/
/**     patrón.                                                         **/
/**     A continuación, dividimos la imagen en tiles y comparamos el    **/
/**     histograma de cada tile con el histograma patrón (tanto ULBP como **/
/**     varianza). Por último, sacamos por pantalla los resultados de la **/
/**     comparación de cada tile.                                       **/
/**                                                                     **/
/**     Si el cuadrado de un tile es rojo significa que se considera gasa, **/
/**     si no, será de color azul. El valor superior corresponde a la   **/
/**     comparación ULBP y el inferior a la varianza.                   **/
/**                                                                     **/
/** Nota: debe ejecutarse previamente el programa "generador LUTs" y el **/
/**     "generador de patrón".                                          **/
/*************************************************************************/

#include "opencv2/opencv.hpp"
#include "../../include/fichero.h"
#include "../../include/lbp.h"
#include "../../include/estadist.h"
#include "../../include/histo.h"
#include "../../include/compara.h"
#include "../../include/visualiza.h"

#define TAM_TILE 100          // Lado del cuadrado en el que troceamos la
    imagen
#define GRADO_SUPERP_TILE 0.0 // 0.0 no hay superposicion en los bloques
#define UMBRAL_HISTO_ULBP   8  // Valores a partir del cual se considera gasa
#define UMBRAL_HISTO_VAR   160

using namespace std;
using namespace cv;

int main()
{
```

```
56      double factorEscala=1.0; // Factor de reduccion de la imagen
57      int R=0, P=0;            // R: radio del vecindario, P: numero de vecinos
58      Mat histoPatronVar;      // Mat::zeros(P+2, 1, CV_32F);
59      Mat histoPatronULBP;     // Mat::zeros(P+2, 1, CV_32F);
60      Mat imgColor;
61      vector<int> LUT;         // LUT para la transformacion lbp -> ulbp
62      vector<int> vectQ;       // Vector para la cuantificacion de la varianza
63
64      cout << "Introducir NUMERO imagen (IMG00NUM.jpg): ";
65      string fileNumber;
66      cin >> fileNumber;
67
68      try
69      {
70          /// 1. Inicializacion de variables de acuerdo al patron almacenado en
                datosPatron.xlm
71          leeFicheroDatosPatronVar(factorEscala, R, P, vectQ, histoPatronULBP,
                histoPatronVar);
72
73          /// 2. Inicializamos la variable LUT que contiene la look-up table que
                 convierte valores LBP en ULBP.
74          leeFicheroLUT(LUT, P);
75
76          /// 3. Leemos la imagen elegida por el usuario
77          leeFicheroImagen(imgColor, fileNumber);
78      }
79
80      catch(int excepcion)
81      {
82          if(excepcion == ERROR_NO_FILE_PATRON)
83              cerr << "ERROR " << excepcion << " leyendo fichero de datos." <<
                    endl;
84          else if(excepcion == ERROR_NO_FILE_LUT)
85              cerr << "ERROR " << excepcion << " leyendo fichero LUT." << endl;
86          else if(excepcion == ERROR_NO_FILE_IMG)
87              cerr << "ERROR " << excepcion << " leyendo fichero Imagen." <<
                    endl;
88          exit(EXIT_FAILURE);
89      }
90
91      /// 4. Convertimos la imagen a color en B/N
92      Mat img;
93      cvtColor(imgColor, img, COLOR_BGR2GRAY);
94
95      /// 5. Calculamos la media de cada vecindario y lo almacenamos en la
            variable EX
96      Mat EX;
97      media<uchar>(img, EX, R, P);
98
99      /// 6. Calculamos la imagen mLBP (utiliza media en vez de pixel central)
100     Mat imgLBP;
101     mLBP(img, EX, imgLBP, R, P);
102     //LBP(img,imgLBP,R,P);
103
104     /// 7. Convertimos la imagen LBP "normal" en uniforme (ULBP) utilizando la
             LUT
105     Mat imgULBP;
106     if(imgLBP.type() == CV_8UC1)
107         LBP2ULBP<uchar>(imgLBP, LUT, imgULBP);
108     else
109         LBP2ULBP<int>(imgLBP, LUT, imgULBP);
```

```
110
111     /// 8. Calculamos la varianza de la imagen de entrada
112     Mat imgVar;
113     if(R == 1 && P == 8)
114         varianza(img, imgVar);
115     else
116         VARLBP(img, imgVar, R, P);
117
118     /// 9. Cuantificamos la imagen de la varianza
119     Mat imgVarQ;
120     cuantImgVar(imgVar, imgVarQ, vectQ);
121
122     /// 10. Realizamos la comparación de cada mosaico (tile) con el histograma
             patrón.
123     ///     Se almacenan los resultados en el vector tilesULBP y tilesVar.
124     vector<int> tilesULBP, tilesVar;
125     tilesULBP = comparaHistosTiles(imgULBP, histoPatronULBP, TAM_TILE*
            factorEscala, GRADO_SUPERP_TILE);
126     tilesVar = comparaHistosVar(imgVarQ, histoPatronVar, TAM_TILE*factorEscala
            , GRADO_SUPERP_TILE);
127
128     /// 11. Dibuja sobre la imagen de entrada los valores de la comparacion de
             histogramas
129     visualizaValoresEnTiles(imgColor, tilesULBP, UMBRAL_HISTO_ULBP, tilesVar,
            UMBRAL_HISTO_VAR, TAM_TILE*factorEscala, GRADO_SUPERP_TILE);
130
131     return 0;
132 }
```

A.3. *main.cpp* ("LBP y varianza combinado, método 1")

```
/***************************************************************************/
/**                          Trabajo Fin de Grado                         **/
/**                                                                       **/
/**                 Detección de Gasas Quirúrgicas en imágenes            **/
/**                  laparoscópicas empleando la técnica LBP              **/
/**                                                                       **/
/**          Grado en Ingeníeria Electrónica Industrial y Automática      **/
/**                          Universidad de Valladolid                    **/
/**                                                                       **/
/** Autor: Álvaro Muñoz García                                            **/
/** Tutor: Eusebio de la Fuente López                                     **/
/***************************************************************************/
/** Proyecto: LBP                                                         **/
/** Archivo: main.cpp                                                     **/
/** Variante: LBP y varianza combinado - Método 1                         **/
/**                                                                       **/
/** Descripción: se trata del programa principal. Se manda elegir al      **/
/**     usuario una imagen laparoscópica (1-32) que se analizará. Para    **/
/**     ello, primero se carga los datos del patrón y la LUT. A           **/
/**     continuación, convertimos la imagen a B/N, calculamos la imagen   **/
/**     de medias, la imagen LBP (o mLBP) y por último la convertimos a   **/
/**     ULBP. Después, calculamos la varianza de la imagen laparoscópica  **/
/**     y la cuantificamos utilizando el vector obtenido en los datos del **/
/**     patrón.                                                           **/
/**     El siguiente paso es generar la imagen combinación de ambas,      **/
/**     multiplicando por 16 la imagen ULBP y sumándola a la imagen de la **/
/**     varianza cuantificada.                                            **/
/**     A continuación, dividimos la imagen en tiles y comparamos el      **/
/**     histograma de cada tile con el histograma patrón. Por último,     **/
/**     sacamos por pantalla los resultados de la comparación de cada tile **/
/**                                                                       **/
/**     Si el cuadrado de un tile es rojo significa que se considera gasa, **/
/**     si no, será de color azul.                                        **/
/**                                                                       **/
/** Nota: debe ejecutarse previamente el programa "generador LUTs" y el   **/
/**     "generador de patrón".                                            **/
/***************************************************************************/

#include "opencv2/opencv.hpp"
#include "../../include/fichero.h"
#include "../../include/lbp.h"
#include "../../include/estadist.h"
#include "../../include/histo.h"
#include "../../include/compara.h"
#include "../../include/visualiza.h"

#define TAM_TILE 100             // Lado del cuadrado en el que troceamos la
    imagen
#define GRADO_SUPERP_TILE 0.0    // 0.0 no hay superposicion en los bloques
#define UMBRAL_HISTO_ULBP_VAR 50 // Valores a partir del cual se considera
    gasa

using namespace std;
using namespace cv;

int main()
```

```
55  {
56      double factorEscala=1.0; // Factor de reduccion de la imagen
57      int R=0, P=0;            // R: radio del vecindario, P: numero de vecinos
58      Mat histoPatronULBPVar;  // Mat::zeros(P+2, 1, CV_32F);
59      Mat imgColor;
60      vector<int> LUT;         // LUT para la transformacion lbp -> ulbp
61      vector<int> vectQ;       // Vector para la cuantificacion de la varianza
62
63      cout << "Introducir NUMERO imagen (IMG00NUM.jpg): ";
64      string fileNumber;
65      cin >> fileNumber;
66
67      try
68      {
69          /// 1. Inicializacion de variables de acuerdo al patron almacenado en
                datosPatron.xlm
70          leeFicheroDatosPatronULBPVar(factorEscala, R, P, vectQ,
                histoPatronULBPVar);
71
72          /// 2. Inicializamos la variable LUT que contiene la look-up table que
                 convierte valores LBP en ULBP.
73          leeFicheroLUT(LUT, P);
74
75          /// 3. Leemos la imagen elegida por el usuario
76          leeFicheroImagen(imgColor, fileNumber);
77      }
78
79      catch(int excepcion)
80      {
81          if(excepcion == ERROR_NO_FILE_PATRON)
82              cerr << "ERROR " << excepcion << " leyendo fichero de datos." <<
                    endl;
83          else if(excepcion == ERROR_NO_FILE_LUT)
84              cerr << "ERROR " << excepcion << " leyendo fichero LUT." << endl;
85          else if(excepcion == ERROR_NO_FILE_IMG)
86              cerr << "ERROR " << excepcion << " leyendo fichero Imagen." <<
                    endl;
87          exit(EXIT_FAILURE);
88      }
89
90      /// 4. Convertimos la imagen a color en B/N
91      Mat img;
92      cvtColor(imgColor, img, COLOR_BGR2GRAY);
93
94      /// 5. Calculamos la media de cada vecindario y lo almacenamos en la
            variable EX
95      Mat EX;
96      media<uchar>(img, EX, R, P);
97
98      /// 6. Calculamos la imagen mLBP (utiliza media en vez de pixel central)
99      Mat imgLBP;
100     mLBP(img, EX, imgLBP, R, P);
101     //LBP(img,imgLBP,R,P);
102
103     /// 7. Convertimos la imagen LBP "normal" en uniforme (ULBP) utilizando la
             LUT
104     Mat imgULBP;
105     if(imgLBP.type() == CV_8UC1)
106         LBP2ULBP<uchar>(imgLBP, LUT, imgULBP);
107     else
108         LBP2ULBP<int>(imgLBP, LUT, imgULBP);
```

```
109
110     /// 8. Calculamos la varianza de la imagen de entrada
111     Mat imgVar;
112     if(R == 1 && P == 8)
113         varianza(img, imgVar);
114     else
115         VARLBP(img, imgVar, R, P);
116
117     /// 9. Cuantificamos la imagen de la varianza
118     Mat imgVarQ;
119     cuantImgVar(imgVar, imgVarQ, vectQ);
120
121     /// 10. Calculamos imagen combinación de ambas
122     Mat imgULBPVar = 16*imgULBP + imgVarQ; // Multiplico por 16 porque la
            varianza está cuantificada a 16 niveles
123
124     /// 11. Realizamos la comparación de cada mosaico (tile) con el histograma
             patrón.
125     ///     Se almacenan los resultados en el vector tiles.
126     vector<int> tiles;
127     tiles = comparaHistosTilesULBPVar(imgULBPVar, histoPatronULBPVar, TAM_TILE
            *factorEscala, GRADO_SUPERP_TILE);
128
129     /// 12. Dibuja sobre la imagen de entrada los valores de la comparacion de
             histogramas
130     visualizaValoresEnTiles(imgColor, tiles, UMBRAL_HISTO_ULBP_VAR, tiles, -1,
             TAM_TILE*factorEscala, GRADO_SUPERP_TILE);
131
132     return 0;
133 }
```

A.4. *main.cpp* ("LBP y varianza combinado, método 2")

```
1  /**************************************************************************/
2  /**                        Trabajo Fin de Grado                          **/
3  /**                                                                      **/
4  /**               Detección de Gasas Quirúrgicas en imágenes             **/
5  /**                laparoscópicas empleando la técnica LBP               **/
6  /**                                                                      **/
7  /**         Grado en Ingeniería Electrónica Industrial y Automática      **/
8  /**                       Universidad de Valladolid                      **/
9  /**                                                                      **/
10 /** Autor: Álvaro Muñoz García                                           **/
11 /** Tutor: Eusebio de la Fuente López                                    **/
12 /**************************************************************************/
13 /** Proyecto: LBP                                                        **/
14 /** Archivo: main.cpp                                                    **/
15 /** Variante: LBP y varianza combinado -  Método 2                       **/
16 /**                                                                      **/
17 /** Descripción: se trata del programa principal. Se manda elegir al     **/
18 /**     usuario una imagen laparoscópica (1-32) que se analizará. Para   **/
19 /**     ello, primero se carga los datos del patrón y la LUT. A          **/
20 /**     continuación, convertimos la imagen a B/N, calculamos la imagen  **/
21 /**     de medias, la imagen LBP (o mLBP) y por último la convertimos a  **/
22 /**     ULBP. Después, calculamos la varianza de la imagen laparoscópica.**/
23 /**     A continuación, dividimos la imagen en tiles y comparamos el     **/
24 /**     histograma de cada tile con el histograma patrón. La forma de    **/
25 /**     calcular el histograma se describe en la cabecera de la función  **/
26 /**     calcularHistoVar() en el archivo ../common/histo.cpp             **/
27 /**     Por último, sacamos por pantalla los resultado de la comparación **/
28 /**     de cada tile.                                                    **/
29 /**                                                                      **/
30 /**     Si el cuadrado de un tile es rojo significa que se considera gasa,**/
31 /**     si no, será de color azul.                                       **/
32 /**                                                                      **/
33 /** Nota: debe ejecutarse previamente el programa "generador LUTs" y el  **/
34 /**     "generador de patrón".                                           **/
35 /**************************************************************************/
36 
37 #include "opencv2/opencv.hpp"
38 #include "../../include/fichero.h"
39 #include "../../include/lbp.h"
40 #include "../../include/estadist.h"
41 #include "../../include/histo.h"
42 #include "../../include/compara.h"
43 #include "../../include/visualiza.h"
44 
45 #define TAM_TILE 100 // Lado del cuadrado en el que troceamos la imagen
46 #define GRADO_SUPERP_TILE 0.0 // 0.0 no hay superposicion en los bloques
47 #define UMBRAL_HISTO_ULBP_VAR_2 6 // Valores a partir del cual se considera
       gasa
48 
49 using namespace std;
50 using namespace cv;
51 
52 int main()
53 {
54     double factorEscala=1.0; // Factor de reduccion de la imagen
55     int R=0, P=0;            // R: radio del vecindario, P: numero de vecinos
```

```
56      Mat histoPatronULBP;      // Mat::zeros(P+2, 1, CV_32F);
57      Mat imgColor;
58      vector<int> LUT;          // LUT para la transformacion lbp -> ulbp
59      vector<int> vectQ;        // Vector para la cuantificacion de la varianza
60
61      cout << "Introducir NUMERO imagen (IMG00NUM.jpg): ";
62      string fileNumber;
63      cin >> fileNumber;
64
65      try
66      {
67          /// 1. Inicializacion de variables de acuerdo al patron almacenado en
                datosPatron.xlm
68          leeFicheroDatosPatron(factorEscala, R, P, histoPatronULBP);
69
70          /// 2. Inicializamos la variable LUT que contiene la look-up table que
                 convierte valores LBP en ULBP.
71          leeFicheroLUT(LUT, P);
72
73          /// 3. Leemos la imagen elegida por el usuario
74          leeFicheroImagen(imgColor, fileNumber);
75      }
76
77      catch(int excepcion)
78      {
79          if(excepcion == ERROR_NO_FILE_PATRON)
80              cerr << "ERROR " << excepcion << " leyendo fichero de datos." <<
                    endl;
81          else if(excepcion == ERROR_NO_FILE_LUT)
82              cerr << "ERROR " << excepcion << " leyendo fichero LUT." << endl;
83          else if(excepcion == ERROR_NO_FILE_IMG)
84              cerr << "ERROR " << excepcion << " leyendo fichero Imagen." <<
                    endl;
85          exit(EXIT_FAILURE);
86      }
87
88      /// 4. Convertimos la imagen a color en B/N
89      Mat img;
90      cvtColor(imgColor, img, COLOR_BGR2GRAY);
91
92      /// 5. Calculamos la media de cada vecindario y lo almacenamos en la
            variable EX
93      Mat EX;
94      media<uchar>(img, EX, R, P);
95
96      /// 6. Calculamos la imagen LBP/mLBP
97      Mat imgLBP;
98      mLBP(img, EX, imgLBP, R, P);
99      //LBP(img,imgLBP,R,P);
100
101     /// 7. Convertimos la imagen LBP "normal" en uniforme (ULBP) utilizando la
             LUT
102     Mat imgULBP;
103     if(imgLBP.type() == CV_8UC1)
104         LBP2ULBP<uchar>(imgLBP, LUT, imgULBP);
105     else
106         LBP2ULBP<int>(imgLBP, LUT, imgULBP);
107
108     /// 8. Calculamos la varianza de la imagen de entrada
109     Mat imgVar;
110     if(R == 1 && P == 8)
```

```
111         varianza(img, imgVar);
112     else
113         VARLBP(img, imgVar, R, P);
114
115     /// 9. Realizamos la comparación de cada mosaico (tile) con el histograma
        patrón.
116     ///    Se almacenan los resultados en el vector tilesULBP.
117     vector<int> tilesULBP, tilesVar;
118     tilesULBP = comparaHistosTilesULBPVar2(imgULBP, imgVar, histoPatronULBP,
        TAM_TILE*factorEscala, GRADO_SUPERP_TILE, P);
119
120     /// 10. Dibuja sobre la imagen de entrada los valores de la comparacion de
         histogramas
121     visualizaValoresEnTiles(imgColor, tilesULBP, UMBRAL_HISTO_ULBP_VAR_2,
        tilesVar, -1, TAM_TILE*factorEscala, GRADO_SUPERP_TILE);
122
123     return 0;
124 }
```

A.5. *compara.h* y *compara.cpp*

compara.h

```
1  /****************************************************************************/
2  /**                          Trabajo Fin de Grado                          **/
3  /**                                                                        **/
4  /**              Detección de Gasas Quirúrgicas en imágenes                **/
5  /**               laparoscópicas empleando la técnica LBP                  **/
6  /**                                                                        **/
7  /**       Grado en Ingeníeria Electrónica Industrial y Automática          **/
8  /**                         Universidad de Valladolid                      **/
9  /**                                                                        **/
10 /** Autor: Álvaro Muñoz García                                             **/
11 /** Tutor: Eusebio de la Fuente López                                      **/
12 /****************************************************************************/
13 /** Proyecto: Común                                                        **/
14 /** Archivo: compara.h                                                     **/
15 /****************************************************************************/
16
17 #ifndef COMPARA_H_INCLUDED
18 #define COMPARA_H_INCLUDED
19
20 #include "opencv2/opencv.hpp"
21
22 std::vector<int> comparaHistosTiles(const cv::Mat &img, const cv::Mat &histo,
       const int N, const double gradoSuperp);
23 std::vector<int> comparaHistosVar(const cv::Mat &imgVarQ, const cv::Mat &
       histoRef, const int N, const double gradoSuperp);
24 std::vector<int> comparaHistosTilesULBPVar(const cv::Mat &img, const cv::Mat &
       histo, const int N, const double gradoSuperp);
25 std::vector<int> comparaHistosTilesULBPVar2(const cv::Mat &imgULBP, const cv::
       Mat &imgVar, const cv::Mat &histo, const int N, const double gradoSuperp,
       int P);
26
27 #endif // COMPARA_H_INCLUDED
```

compara.cpp

```
/*************************************************************************/
/**                        Trabajo Fin de Grado                         **/
/**                                                                     **/
/**               Detección de Gasas Quirúrgicas en imágenes            **/
/**                laparoscópicas empleando la técnica LBP              **/
/**                                                                     **/
/**        Grado en Ingeniería Electrónica Industrial y Automática      **/
/**                        Universidad de Valladolid                    **/
/**                                                                     **/
/** Autor: Álvaro Muñoz García                                          **/
/** Tutor: Eusebio de la Fuente López                                   **/
/*************************************************************************/
/** Proyecto: Común                                                     **/
/** Archivo: compara.cpp                                                **/
/*************************************************************************/

#include "../include/compara.h"
#include "../include/histo.h"
#include "../include/estadist.h"
#include <string>

using namespace std;
using namespace cv;

/*************************************************************************/
/** Función: comparaHistosTiles                                         **/
/**                                                                     **/
/** Descripción: divide la imagen en tiles, crea sus histogramas, los   **/
/**     normaliza y los compara con el histograma de patrón. Para la    **/
/**     comparación se utiliza la función nativa de OpenCV cv::compareHist **/
/**     utilizando la opción chi-cuadrado.                              **/
/**     Los valores de la comparación se guardan en un vector multiplicado **/
/**     por 100.                                                        **/
/**     Cuando el grado de superposición es 0.5, se mete hasta la mitad de **/
/**     tiles adyacentes.                                               **/
/**                                                                     **/
/** Tipo de imagen: ULBP                                                **/
/**                                                                     **/
/** Nota: el orden de las comparaciones en el vector es de izquierda a  **/
/**     derecha y de arriba a abajo.                                    **/
/*************************************************************************/
vector<int> comparaHistosTiles(const Mat &img, const Mat &histo, const int N,
    const double gradoSuperp)
{
    vector<int> comparacionTiles;
    int marco = gradoSuperp*N; // Número de pixeles que superpone sobre el
        cuadrado base NxN

    for (int r = marco; r < img.rows-marco; r += N)
    {
        for (int c = marco; c < img.cols-marco; c += N)
        {
            cv::Mat tile = img( Range(r - marco, min(r + N + marco, img.rows -
                marco)  ),
                                Range(c - marco, min(c + N + marco, img.cols -
                                    marco)  )   );

            Mat histoTileULBP;
            calcHisto(tile, histoTileULBP, histo.rows, 0, histo.rows);
```

```
56              normalize(histoTileULBP, histoTileULBP, 0, 1, NORM_MINMAX, -1, Mat
                    ());
57              histoTileULBP = comprobarHistograma(histoTileULBP);
58
59              auto comp = compareHist( histo, histoTileULBP, HISTCMP_CHISQR);
60              comparacionTiles.push_back(static_cast<int>(comp*100));
61          }
62      }
63
64      return comparacionTiles;
65  }
66
67  /***************************************************************************/
68  /** Función: comparaHistosVar                                            **/
69  /**                                                                      **/
70  /** Descripción: divide la imagen en tiles, crea sus histogramas, los    **/
71  /**     normaliza y los compara con el histograma de patrón. Para la     **/
72  /**     comparación se utiliza la función nativa de OpenCV cv::compareHist **/
73  /**     utilizando la opción chi-cuadrado.                               **/
74  /**     Los valores de la comparación se guardan en un vector multiplicado **/
75  /**     por 100.                                                         **/
76  /**     Cuando el grado de superposición es 0.5, se mete hasta la mitad de **/
77  /**     tiles adyacentes.                                                **/
78  /**                                                                      **/
79  /** Tipo de imagen: VAR                                                  **/
80  /**                                                                      **/
81  /** Nota: el orden de las comparaciones en el vector es de izquierda a   **/
82  /**     derecha y de arriba a abajo.                                     **/
83  /***************************************************************************/
84  vector<int> comparaHistosVar(const Mat &imgVarQ, const Mat &histoRef, const
        int N, const double gradoSuperp)
85  {
86      vector<int> comparacionTiles;
87      int marco = gradoSuperp * N; // Numero de pixeles que superpone sobre el
            cuadrado base NxN
88
89      int numBins = histoRef.rows;
90
91      for (int r = marco; r < imgVarQ.rows-marco; r += N)
92      {
93          for (int c = marco; c < imgVarQ.cols-marco; c += N)
94          {
95              cv::Mat tile = imgVarQ( Range(r - marco, min(r + N + marco,
                    imgVarQ.rows - marco)),
96                                      Range(c - marco, min(c + N + marco,
                                            imgVarQ.cols - marco)));
97
98              Mat histoTileVarQ;
99              calcHisto(tile, histoTileVarQ ,numBins,0, numBins);
100             normalize(histoTileVarQ, histoTileVarQ, 0, 1, NORM_MINMAX, -1, Mat
                    ());
101             histoTileVarQ = comprobarHistograma(histoTileVarQ);
102
103             auto comp = compareHist(histoRef, histoTileVarQ, HISTCMP_CHISQR);
104             comparacionTiles.push_back(static_cast<int>(comp*100));
105         }
106     }
107     return comparacionTiles;
108 }
109
110 /***************************************************************************/
```

```
111 /** Función: comparaHistosTilesULBPVar                                   **/
112 /**                                                                       **/
113 /** Descripción: divide la imagen en tiles, crea sus histogramas, los     **/
114 /**     normaliza y los compara con el histograma de patrón. Para la      **/
115 /**     comparación se utiliza la función nativa de OpenCV cv::compareHist **/
116 /**     utilizando la opción chi-cuadrado.                                **/
117 /**     Los valores de la comparación se guardan en un vector multiplicado **/
118 /**     por 10.                                                           **/
119 /**     Cuando el grado de superposición es 0.5, se mete hasta la mitad de **/
120 /**     tiles adyacentes.                                                 **/
121 /**                                                                       **/
122 /** Tipo de imagen: ULBP + VAR (Método 1)                                 **/
123 /**                                                                       **/
124 /** Nota: el orden de las comparaciones en el vector es de izquierda a    **/
125 /**     derecha y de arriba a abajo.                                      **/
126 /*************************************************************************/
127 vector<int> comparaHistosTilesULBPVar(const Mat &img, const Mat &histo, const
        int N, const double gradoSuperp)
128 {
129     vector<int> comparacionTiles;
130     int marco = gradoSuperp*N; // Numero de pixeles que superpone sobre el
            cuadrado base NxN
131 
132     for (int r = marco; r < img.rows-marco; r += N)
133     {
134         for (int c = marco; c < img.cols-marco; c += N)
135         {
136             cv::Mat tile = img( Range(r - marco, min(r + N + marco, img.rows -
                     marco) ),
137                                 Range(c - marco, min(c + N + marco, img.cols -
                                     marco) ) );
138 
139             Mat histoTileULBPVar;
140             calcHisto(tile, histoTileULBPVar, histo.rows, 0, histo.rows);
141             normalize(histoTileULBPVar, histoTileULBPVar, 0, 1, NORM_MINMAX,
                    -1, Mat());
142             histoTileULBPVar = comprobarHistograma(histoTileULBPVar);
143 
144 
145             auto comp = compareHist( histo, histoTileULBPVar, HISTCMP_CHISQR);
146             comparacionTiles.push_back(static_cast<int>(comp*10));
147         }
148     }
149 
150     return comparacionTiles;
151 }
152 
153 /*************************************************************************/
154 /** Función: comparaHistosTilesULBPVar2                                  **/
155 /**                                                                       **/
156 /** Descripción: divide la imagen en tiles, crea sus histogramas con la   **/
157 /**     función calcularHistoVar, los normaliza y los compara con el      **/
158 /**     histograma de patrón. Para la comparación se utiliza la función   **/
159 /**     nativa de OpenCV cv::compareHist utilizando la opción chi-cuadrado **/
160 /**     Los valores de la comparación se guardan en un vector multiplicado **/
161 /**     por 150.                                                          **/
162 /**     Cuando el grado de superposición es 0.5, se mete hasta la mitad de **/
163 /**     tiles adyacentes.                                                 **/
164 /**                                                                       **/
165 /** Tipo de imagen: ULBP + VAR (Método 2)                                 **/
166 /**                                                                       **/
```

```
167 /** Nota: el orden de las comparaciones en el vector es de izquierda a     **/
168 /**     derecha y de arriba a abajo.                                         **/
169 /***************************************************************************/
170 vector<int> comparaHistosTilesULBPVar2(const Mat &imgULBP, const Mat &imgVar,
        const Mat &histo, const int N, const double gradoSuperp, int P)
171 {
172     vector<int> comparacionTiles;
173     int marco = gradoSuperp*N; // Numero de pixeles que superpone sobre el
            cuadrado base NxN
174
175     for (int r = marco; r < imgULBP.rows-marco; r += N)
176     {
177         for (int c = marco; c < imgULBP.cols-marco; c += N)
178         {
179             cv::Mat tileULBP = imgULBP( Range(r - marco, min(r + N + marco,
                    imgULBP.rows - marco) ),
180                                         Range(c - marco, min(c + N + marco, imgULBP.
                                                cols - marco) )  );
181
182             cv::Mat tileVar = imgVar( Range(r - marco, min(r + N + marco,
                    imgVar.rows - marco) ),
183                                         Range(c - marco, min(c + N + marco, imgVar.
                                                cols - marco) )  );
184
185             Mat histoTileULBPVar;
186             calcularHistoVar(tileULBP, tileVar, histoTileULBPVar, P+2);
187             normalize(histoTileULBPVar, histoTileULBPVar, 0, 1, NORM_MINMAX,
                    -1, Mat());
188             histoTileULBPVar = comprobarHistograma(histoTileULBPVar);
189
190             auto comp = compareHist(histo, histoTileULBPVar, HISTCMP_CHISQR);
191             comparacionTiles.push_back(static_cast<int>(comp*150));
192         }
193     }
194
195     return comparacionTiles;
196 }
```

A.6. *visualiza.h* y *visualiza.cpp*

visualiza.h

```
/*************************************************************************/
/**                          Trabajo Fin de Grado                       **/
/**                                                                     **/
/**             Detección de Gasas Quirúrgicas en imágenes              **/
/**              laparoscópicas empleando la técnica LBP                **/
/**                                                                     **/
/**      Grado en Ingeníeria Electrónica Industrial y Automática        **/
/**                       Universidad de Valladolid                     **/
/**                                                                     **/
/** Autor: Álvaro Muñoz García                                          **/
/** Tutor: Eusebio de la Fuente López                                   **/
/*************************************************************************/
/** Proyecto: Común                                                     **/
/** Archivo: visualiza.h                                                **/
/*************************************************************************/

#ifndef VISUALIZA_H_INCLUDED
#define VISUALIZA_H_INCLUDED

#include "opencv2/opencv.hpp"

void visualizaValoresEnTiles(const cv::Mat &img, const std::vector<int> &
    tilesHisto, const int umbralHistoULBP, const std::vector<int> & tilesVar,
    const int umbralHistoVar, int N, double gradoSuperp);

#endif // VISUALIZA_H_INCLUDED
```

visualiza.cpp

```
/*****************************************************************************/
/**                          Trabajo Fin de Grado                            **/
/**                                                                          **/
/**              Detección de Gasas Quirúrgicas en imágenes                  **/
/**               laparoscópicas empleando la técnica LBP                    **/
/**                                                                          **/
/**        Grado en Ingeníeria Electrónica Industrial y Automática           **/
/**                          Universidad de Valladolid                       **/
/**                                                                          **/
/** Autor: Álvaro Muñoz García                                               **/
/** Tutor: Eusebio de la Fuente López                                        **/
/*****************************************************************************/
/** Proyecto: Común                                                          **/
/** Archivo: visualiza.cpp                                                   **/
/*****************************************************************************/

#include "opencv2/opencv.hpp"
#include "../include/visualiza.h"

#define COLOR_MARCA_VENDA Scalar(0,0,255)
#define ANCHO_RETICULA 10

using namespace std;
using namespace cv;

/*****************************************************************************/
/** Función: visualizaValoresEnTiles                                         **/
/**                                                                          **/
/** Descripción: se muestran los resultados de la comparación sobre la       **/
/**     imagen original. Para ello se dibuja una retícula, de forma que se  **/
/**     pueden ver los tiles en los cuales se ha dividido la imagen para    **/
/**     su comparacion. Si el color del recuadro de un tile es rojo, se     **/
/**     considera que es una venda, y sino sera azul. También se añade el   **/
/**     valor de la comparación de histogramas LBP y varianzas.             **/
/**     Si el umbral tanto de histogramas LBP como varianzas, es negativo,  **/
/**     no se dibujará nada.                                                **/
/*****************************************************************************/
void visualizaValoresEnTiles(const Mat &img, const vector<int> &tilesHisto,
    const int umbralHistoULBP, const vector<int> & tilesVar, const int
    umbralHistoVar, int N, double gradoSuperp)
{
    int marco = gradoSuperp*N;
    Mat imgDrawing = img(Range(1,img.rows-1),Range(1,img.cols-1));
    //cvtColor(imgDrawing, imgDrawing, COLOR_GRAY2BGR); // Convertiamos imagen
         de B/N a color
    int numTile = 0;

    for (int r = marco; r < img.rows-marco; r += N)
    {
        for (int c = marco; c < img.cols-marco; c += N)
        {
            int width=min(N,imgDrawing.cols-c);
            int height=min(N,imgDrawing.rows-r);
            Rect miTile=Rect( Point(c,r), Size(width,height));

            /// Valores comparación LBP. Rojo: venda, Azul: no
            Scalar colorTexto;
            int thickness;
            if(umbralHistoULBP >= 0) // Solo umbrales positivos
```

```
57                  {
58                      if(tilesHisto[numTile] < umbralHistoULBP) // Es venda
59                      {
60                          colorTexto = Scalar(0,0,255);
61                          thickness = 3;
62                      }
63                      else // No es venda
64                      {
65                          colorTexto = Scalar(255,0,0);
66                          thickness = 1;
67                      }
68
69                      string text = to_string (tilesHisto[numTile]);
70                      Point org = Point(c,r) + Point(width/3,height/3);
71                      putText(imgDrawing, text, org, FONT_HERSHEY_SIMPLEX, 1,
                            colorTexto, thickness);
72                      rectangle(imgDrawing,miTile, colorTexto, thickness);
73                  }
74
75                  /// Valores comparación varianza
76                  if(umbralHistoVar >= 0)
77                  {
78                      if(tilesVar[numTile] < umbralHistoVar) // Es venda
79                      {
80                          colorTexto = Scalar(0,0,255);
81                          thickness = 3;
82                      }
83                      else // No es venda
84                      {
85                          colorTexto = Scalar(255,0,0);
86                          thickness = 1;
87                      }
88
89                      string text = to_string (tilesVar[numTile]);
90                      Point org = Point(c,r) + Point(width/3,2*height/3);
91                      putText(imgDrawing, text, org, FONT_HERSHEY_SIMPLEX, 1,
                            colorTexto, thickness);
92                      if(thickness == 3)
93                          rectangle(imgDrawing, miTile, colorTexto, thickness);
94                  }
95
96                  ++numTile;
97              }
98          }
99
100         namedWindow("Resultados");
101         moveWindow("Resultados",0,0);
102         imshow("Resultados", imgDrawing);
103         waitKey(0);
104     }
```

Anexo B

Código fuente: generador de patrón

B.1. *main.cpp* ("LBP" y "LBP y varianza por separado")

```
/*************************************************************************/
/**                         Trabajo Fin de Grado                        **/
/**                                                                     **/
/**             Detección de Gasas Quirúrgicas en imágenes              **/
/**              laparoscópicas empleando la técnica LBP                **/
/**                                                                     **/
/**       Grado en Ingeníeria Electrónica Industrial y Automática       **/
/**                         Universidad de Valladolid                   **/
/**                                                                     **/
/** Autor: Álvaro Muñoz García                                          **/
/** Tutor: Eusebio de la Fuente López                                   **/
/*************************************************************************/
/** Proyecto: Generador de patrón                                       **/
/** Archivo: main.cpp                                                   **/
/** Variante: LBP / LBP y varianza por separado                         **/
/**                                                                     **/
/** Descripción: se encarga de generar el archivo "datosPatron.xml" en la **/
/**     carpeta "patternData" que contiene la información de todas las  **/
/**     imágenes patrón. Para ello, crea el histograma ULBP y de varianzas **/
/**     de la combinación de todas las imágenes patrón. Despues cuantifica **/
/**     el histograma de varianzas y por último normaliza ambos         **/
/**     histogramas, con los que genera el archivo de salida.           **/
/*************************************************************************/

#include "opencv2/opencv.hpp"
#include "../../include/histo.h"
#include "../../include/fichero.h"
#include "../../include/lbp.h"
#include "../../include/estadist.h"

/// Parámetros del histograma de varianza
#define NUM_BINS_VAR 16  // Numero de cubetas del histograma cuantificado.
    Debe ser potencia de 2.
#define VAR_MAX 600      // Numero de cubetas del histograma inicial
```

```
34 //#define PARAM_MANUALES // Descomentar para elegir P y R.
35
36 using namespace std;
37 using namespace cv;
38
39 vector<int> calculaCuantificacionVar(const Mat &histoVarSuma, int numBins);
40 void cuantifHistoVar(const Mat &histoVarSuma, vector<int>vectQuant, Mat &
       histoVarQ);
41
42 int main()
43 {
44     /// 1. Parámetros del codigo LBP y declaración de variables
45     int R, P;
46     double factorEscala = 1.0;
47
48 #ifdef PARAM_MANUALES
49     cout << "Intro R(radio) y P(num.vecinos)" << endl;
50     cout << "R: ";
51     cin >> R;
52     cout << "P: ";
53     cin >> P;
54 #else
55     R = 1;
56     P = 8;
57 #endif // PARAM_MANUALES
58
59     bool hayQueReducirTamImg = false;
60     if((1.0 - factorEscala) > 0.1)
61         hayQueReducirTamImg = true;
62
63     Mat imgPatron; // Contiene la imagen patron
64     Mat histoVar;  // Histograma de varianzas
65     Mat histoVarSuma = Mat::zeros(VAR_MAX, 1, CV_32F); // 1 columna, VAR_MAX
           filas
66     Mat imgVar;    // Varianza de cada vecindario
67     Mat imgLBP, imgULBP;
68     Mat histoULBP;
69     Mat histoULBPsuma = Mat::zeros(P+2, 1, CV_32F);    // 1 columna, P+2 filas
70
71     string fileName = "../patternImg/patron"; // Nombre base
72     bool hayMasFicheros = true; // Itera hasta que sea false
73     string strNum;              // Numero del fichero
74
75     /// 2. Inicialización look-up table para transformación LBP -> ULBP
76     vector<int> LUT;
77     leeFicheroLUT(LUT, P);
78
79     /// 3. Leemos primera imagen patrón
80     int numFich = 1;
81     strNum = to_string(numFich);
82     imgPatron = imread("../patternImg/patron1.png", IMREAD_GRAYSCALE);
83
84     if(!imgPatron.data)
85     {
86         cout << "No se encuentra fichero \"patron1.png\"" << endl;
87         cout << "Debe haber al menos un fichero en el directorio ../patternImg
               : " << "el fichero patron1.png" << endl;
88         return -1;
89     }
90
91     else
```

```
92          cout << "Leyendo primer fichero: \"patron1.png\"" << endl;
93
94      /// 4. Bucle que lee todos los patrones
95      while(hayMasFicheros)
96      {
97          /// 5. Reducimos patrón si es necesario
98          if(hayQueReducirTamImg)
99          {
100             cout << "Se va a reducir por " << factorEscala << endl;
101             GaussianBlur(imgPatron,imgPatron,Size(3,3),1,1);
102             Size dsize = Size(imgPatron.cols*factorEscala, imgPatron.rows*
                    factorEscala);
103             resize(imgPatron, imgPatron, dsize, INTER_AREA);
104         }
105
106         /// 6. Calculamos la varianza y su histograma
107         if(R == 1 && P == 8)
108             varianza(imgPatron, imgVar);
109         else
110             VARLBP(imgPatron, imgVar, R, P);
111
112         calcHisto(imgVar, histoVar, VAR_MAX, 0, VAR_MAX); // Histograma de
                VAR_MAX niveles
113         histoVarSuma += histoVar; // Suma histogramas de varianzas
114
115         /// 7. Calculamos imagen de medias de cada vecindario
116         Mat EX;
117         media<uchar>(imgPatron, EX, R, P);
118
119         /// 8. Calculamos imagen LBP y generamos su histograma
120         //LBP(imgPatron, imgLBP, R, P);
121         mLBP(imgPatron, EX, imgLBP, R, P);
122
123         if(imgLBP.type() == CV_8UC1)
124             LBP2ULBP<uchar>(imgLBP, LUT, imgULBP);
125         else
126             LBP2ULBP<int>(imgLBP, LUT, imgULBP);
127
128         calcHisto(imgULBP, histoULBP, P+2, 0, P+2);
129         histoULBPsuma += histoULBP; // Suma histogramas ULBP
130
131         // Carga fichero siguiente si existe, para seguir el mismo proceso
132         // durante la siguiente iteración.
133         numFich++;
134         strNum = to_string(numFich);
135         imgPatron = imread(fileName + strNum + ".png", IMREAD_GRAYSCALE);
136
137         if(!imgPatron.data)
138         {
139             cout << "No encuentro fichero "<< fileName+strNum+".png" << endl;
140             cout << "Termina lectura de ficheros. "  << endl;
141             hayMasFicheros=false;
142         }
143
144         else
145             cout << "Leyendo " << fileName + strNum + ".png" << endl;
146     }
147     cout << endl;
148
149     /// 9. Cuantificación de la varianza
150     Mat imgVarQ;
```

```
151     vector<int> vectQuant;
152     Mat histoVarQ(NUM_BINS_VAR, 1, CV_32FC1);
153
154     vectQuant = calculaCuantificacionVar(histoVarSuma, NUM_BINS_VAR);
155     cuantifHistoVar(histoVarSuma, vectQuant, histoVarQ);
156
157     cout << "suma patron: " << histoULBPsuma << endl;
158
159     /// 10. Normalización de los histogramas de varianza y ULBP
160     normalize(histoVarQ,  histoVarQ, 0, 1, NORM_MINMAX, -1, Mat());
161     normalize(histoULBPsuma, histoULBPsuma, 0, 1, NORM_MINMAX, -1, Mat());
162
163     histoVarQ = comprobarHistograma(histoVarQ);
164     histoULBPsuma = comprobarHistograma(histoULBPsuma);
165
166     /// 11. Guardar resultados en fichero datosPatron.xml
167     string fichPatron = DATA_FOLDER + string("datosPatron.xml");
168     cout << endl << "Grabando datos en el fichero " << fichPatron << endl;
169     FileStorage fp(fichPatron,FileStorage::WRITE);
170     time_t rawtime;
171     time(&rawtime);
172
173     fp << "Date" << asctime(localtime(&rawtime));
174     fp << "factorEscala" << factorEscala;
175     fp << "radio" << R;
176     fp << "numVecinos" << P;
177     fp << "tamHistoVar" << NUM_BINS_VAR;
178     fp << "vectorQuant" << vectQuant;
179     fp << "histoVar"  << histoVarQ;
180     fp << "histoULBP" << histoULBPsuma;
181
182     fp.release();
183
184     /// 12. Dibujar histogramas y guardarlos en el disco
185     Mat imgDrawing=Mat::zeros(ALTO_IMG_HISTO, ANCHO_IMG_HISTO, CV_8UC1);
186     dibujaHistograma(imgDrawing, histoULBPsuma, P+2);
187     imwrite(string(DATA_FOLDER)+string("histoULBP.png"), imgDrawing);
188
189     namedWindow("Histo ULBP");
190     moveWindow("Histo ULBP", ANCHO_IMG_HISTO, 0);
191     imshow("Histo ULBP", imgDrawing);
192     waitKey(0);
193
194     imgDrawing=Mat::zeros(ALTO_IMG_HISTO, ANCHO_IMG_HISTO, CV_8UC1);
195     dibujaHistograma(imgDrawing, histoVarQ, NUM_BINS_VAR);
196     imwrite(string(DATA_FOLDER)+string("histoVar.png"), imgDrawing);
197
198     namedWindow("Histo Img");
199     moveWindow("Histo Img",0,0);
200     imshow("Histo Img", imgDrawing);
201     waitKey(0);
202 }
203
204 /*****************************************************************************/
205 /** Función: calculaCuantificacionVar                                      **/
206 /**                                                                       **/
207 /** Descripción: se pertende crear un histograma que en vez de tener       **/
208 /**     VAR_MAX cubetas (600), tenga NUM_BINS_VAR (16), y que estas         **/
209 /**     cubetas estén lo más equilibradas posibles. Devuelve un vector      **/
210 /**     indicando las "filas de corte" del histograma original.            **/
211 /*****************************************************************************/
```

```
212 vector<int> calculaCuantificacionVar(const Mat &histoVarSuma, int numBins)
213 {
214     vector<int> vectQuant;
215     int numTotalPixeles=0;
216     int pixPerBin;
217
218     // Sumamos el numero de pixeles del histograma
219     for(int i=0; i<histoVarSuma.rows; i++)
220         numTotalPixeles += histoVarSuma.at<float>(i);
221
222     // Calcula el numero de pixeles por cada bin
223     pixPerBin = static_cast<float>(numTotalPixeles)/numBins;
224     cout << endl << "Numero total de pixeles:" << numTotalPixeles << endl;
225     cout << "Numero de pixeles por bin: " << pixPerBin << endl;
226
227     // Se crea el nuevo histograma, intentando que cada cubeta idx contenga
228     // "pixPerBin" pixeles.
229     int numPixAcc=0; // N de pixeles real de cada cubeta
230     int k=0;         // Fila del histograma original
231
232     for(int idx=0; idx<numBins; idx++)
233     {
234         int umbralNivel= numPixAcc + pixPerBin;
235         cout << "idx:" << idx << " umbral Nivel:" << umbralNivel << endl;
236         numPixAcc += histoVarSuma.at<float>(k++);
237
238         while(numPixAcc<umbralNivel && k<histoVarSuma.rows)
239             numPixAcc += histoVarSuma.at<float>(k++);
240
241         if (k>=VAR_MAX)
242             break;
243
244         vectQuant.push_back(k);
245         cout << " k:" << k << "  numPixAcc" << numPixAcc << endl;
246     }
247
248     return vectQuant;
249 }
250
251 /*************************************************************************/
252 /** Función: cuantifHistoVar                                           **/
253 /**                                                                     **/
254 /** Descripción: generamos el histograma cuantificado a partir de los   **/
255 /**     datos del vector de cuantificación calculado en la función      **/
256 /**     calculaCuantificacionVar.                                       **/
257 /*************************************************************************/
258 void cuantifHistoVar(const Mat &histoVarSuma, vector<int>vectQuant,  Mat &
        histoVarQ)
259 {
260     int numPixAcc=0;
261     int k=0;
262
263     for(size_t idx=0; idx<vectQuant.size(); idx++)
264     {
265         numPixAcc=0;
266         while( k<vectQuant[idx])
267             numPixAcc +=histoVarSuma.at<float>(k++);
268
269         histoVarQ.at<float>(idx)=numPixAcc;
270     }
271
```

```
272     int numPixRestantes=0;
273     for(int i=vectQuant[vectQuant.size()-1]; i<histoVarSuma.rows; i++)
274         numPixRestantes += histoVarSuma.at<float>(i);
275
276     histoVarQ.at<float>(vectQuant.size())=numPixRestantes;
277 }
```

B.2. *main.cpp* ("LBP y varianza combinado, método 1")

```
1  /*****************************************************************************/
2  /**                        Trabajo Fin de Grado                            **/
3  /**                                                                        **/
4  /**                Detección de Gasas Quirúrgicas en imágenes              **/
5  /**                 laparoscópicas empleando la técnica LBP                **/
6  /**                                                                        **/
7  /**         Grado en Ingeníeria Electrónica Industrial y Automática        **/
8  /**                        Universidad de Valladolid                       **/
9  /**                                                                        **/
10 /** Autor: Álvaro Muñoz García                                             **/
11 /** Tutor: Eusebio de la Fuente López                                      **/
12 /*****************************************************************************/
13 /** Proyecto: Generador de patrón                                          **/
14 /** Archivo: main.cpp                                                      **/
15 /** Variante: LBP y varianza combinado - Método 1                          **/
16 /**                                                                        **/
17 /** Descripción: se encarga de generar el archivo "datosPatron.xml" en la  **/
18 /**     carpeta "patternData" que contiene la información de todas las     **/
19 /**     imágenes patrón. Para ello, crea el histograma ULBP y de varianzas **/
20 /**     de la combinación de todas las imágenes patrón. Despues cuantifica **/
21 /**     el histograma de la varianza y ULBP combinado y por último         **/
22 /**     normaliza el histograma, con el que genera el archivo de salida.   **/
23 /*****************************************************************************/
24
25 #include "opencv2/opencv.hpp"
26 #include "../../include/histo.h"
27 #include "../../include/fichero.h"
28 #include "../../include/lbp.h"
29 #include "../../include/estadist.h"
30
31 /// Parámetros del histograma de varianza
32 #define NUM_BINS_VAR 16  // Numero de cubetas del histograma cuantificado.
       Debe ser potencia de 2.
33 #define VAR_MAX 600      // Numero de cubetas del histograma inicial
34 //#define PARAM_MANUALES // Descomentar para elegir P y R
35
36 using namespace std;
37 using namespace cv;
38
39 vector<int> calculaCuantificacionVar(const Mat &histoVarSuma, int numBins);
40 void cuantifHistoVar(const Mat &histoVarSuma, vector<int>vectQuant,  Mat &
       histoVarQ);
41
42 int main()
43 {
44     /// 1. Parámetros del codigo LBP y declaración de variables
45     int R, P;
46     double factorEscala = 1.0;
47
48 #ifdef PARAM_MANUALES
49     cout << "Intro R(radio) y P(num.vecinos)" << endl;
50     cout << "R: ";
51     cin >> R;
52     cout << "P: ";
53     cin >> P;
54 #else
```

```
55      R = 1;
56      P = 8;
57  #endif // PARAM_MANUALES
58
59      Mat imgPatron; // Contine imagen del patron
60      Mat histoVar;  // Histograma de varianzas
61      Mat histoVarSuma = Mat::zeros(VAR_MAX, 1, CV_32F);
62      Mat imgVar;    // Varianza de cada vecindario
63
64      string fileName = "../patternImg/patron"; // Nombre base
65      bool hayMasFicheros = true; // Itera hasta que sea false
66      string strNum;              // Numero del fichero
67
68      /// 2. Inicialización look-up table para transformación LBP -> ULBP
69      vector<int> LUT;
70      leeFicheroLUT(LUT, P);
71
72      /// 3. Leemos primera imagen patrón
73      int numFich = 1;
74      strNum = to_string(numFich);
75      imgPatron = imread("../patternImg/patron1.png", IMREAD_GRAYSCALE);
76
77      if(!imgPatron.data)
78      {
79          cout << "No se encuentra fichero \"patron1.png\"" << endl;
80          cout << "Debe haber al menos un fichero en el directorio ../patternImg
            : " << "el fichero patron1.png" << endl;
81          return -1;
82      }
83
84      else
85          cout << "Leyendo primer fichero: \"patron1.png\"" << endl;
86
87      /// 4. Bucle que lee todos los patrones
88      while(hayMasFicheros)
89      {
90          /// 5. Calculamos la varianza y su histograma
91          if(R == 1 && P == 8)
92              varianza(imgPatron, imgVar);
93          else
94              VARLBP(imgPatron, imgVar, R, P);
95
96          calcHisto(imgVar, histoVar, VAR_MAX, 0, VAR_MAX); // Histograma de
            VAR_MAX niveles
97          histoVarSuma += histoVar; // Suma histogramas de varianzas
98
99          // Carga fichero siguiente si existe, para seguir el mismo proceso
100         // durante la siguiente iteración.
101         numFich++;
102         strNum = to_string(numFich);
103         imgPatron = imread(fileName + strNum + ".png", IMREAD_GRAYSCALE);
104
105         if(!imgPatron.data)
106         {
107             cout << "No encuentro fichero "<< fileName+strNum+".png" << endl;
108             cout << "Termina lectura de ficheros. "  << endl;
109             hayMasFicheros=false;
110         }
111
112         else
113             cout << "Leyendo " << fileName + strNum + ".png" << endl;
```

```
114     }
115     cout << endl;
116
117     /// 6. Cuantificación de la varianza
118     vector<int> vectQuant;
119     vectQuant = calculaCuantificacionVar(histoVarSuma, NUM_BINS_VAR);
120
121     /// 7. Volvemos a leer los archivos patrón para generar el histograma
122     ///    de ULBP y varianzas
123     hayMasFicheros = true;
124     numFich = 1;
125     strNum = to_string(numFich);
126     imgPatron = imread("../patternImg/patron1.png", IMREAD_GRAYSCALE);
127
128     Mat imgLBP, imgULBP; // Imagenes LBP y ULBP
129     Mat histoULBPVar = Mat::zeros((P+2)*16+16, 1, CV_32F); // Histograma
            conjunto de la varianza y el código ULBP
130     Mat histoULBPVarSuma = Mat::zeros((P+2)*16+16, 1, CV_32F);
131
132     while(hayMasFicheros)
133     {
134         /// 8. Calculamos la imagen de varianzas
135         if(R == 1 && P == 8)
136             varianza(imgPatron, imgVar);
137         else
138             VARLBP(imgPatron, imgVar, R, P);
139
140         /// 9. Calculamos imagen LBP y ULBP
141         Mat EX;
142         media<uchar>(imgPatron, EX, R, P);
143
144         mLBP(imgPatron, EX, imgLBP, R, P); // mLBP solo para P=1 y R=8
145         //LBP(imgPatron, imgLBP, R, P); // LBP para tamaño arbitrario
146
147         if(imgLBP.type() == CV_8UC1)
148             LBP2ULBP<uchar>(imgLBP, LUT, imgULBP);
149         else
150             LBP2ULBP<int>(imgLBP, LUT, imgULBP);
151
152         /// 10. Cuantificación de la varianza utilizando "vectQuant"
153         Mat imgVarQ;
154         cuantImgVar(imgVar, imgVarQ, vectQuant);
155
156         /// 11. Calculo de la imagen e histograma de ULBP + Var
157         Mat imgULBPVar(imgVar.rows, imgVar.cols, CV_8UC1);
158
159         imgULBPVar = 16*imgULBP + imgVarQ; // Multiplico por 16 porque la
                varianza está cuantificada a 16 niveles
160         calcHisto(imgULBPVar, histoULBPVar, (P+2)*16+16, 0, (P+2)*16+16);
161         histoULBPVarSuma += histoULBPVar;
162
163         // Carga el fichero siguiente si existe, para seguir el mismo
164         // proceso durante la siguiente iteración.
165         numFich++;
166         strNum = to_string(numFich);
167         imgPatron = imread(fileName + strNum + ".png", IMREAD_GRAYSCALE);
168
169         if(!imgPatron.data)
170         {
171             cout << "No encuentro fichero "<< fileName+strNum+".png" << endl;
172             cout << "Termina lectura de ficheros. "  << endl;
```

```
173                 hayMasFicheros=false;
174             }
175 
176             else
177                 cout << "Leyendo " << fileName + strNum + ".png" << endl;
178         }
179 
180         /// 9. Normalización del histograma ULBP + Var
181         normalize(histoULBPVarSuma, histoULBPVarSuma, 0, 1, NORM_MINMAX, -1, Mat()
                );
182         histoULBPVarSuma = comprobarHistograma(histoULBPVarSuma);
183 
184         /// 10. Guardar resultados en fichero datosPatron.xml
185         string fichPatron = DATA_FOLDER + string("datosPatron.xml");
186         cout << "\nGrabando datos en el fichero " << fichPatron << endl;
187         FileStorage fp(fichPatron,FileStorage::WRITE);
188         time_t rawtime;
189         time(&rawtime);
190 
191         fp << "Date" << asctime(localtime(&rawtime));
192         fp << "factorEscala" << factorEscala;
193         fp << "radio" << R;
194         fp << "numVecinos" << P;
195         fp << "tamHistoVar" << NUM_BINS_VAR;
196         fp << "vectorQuant" << vectQuant;
197         fp << "histoULBPVar" << histoULBPVarSuma;
198 
199         fp.release();
200 
201         /// 11. Dibujar histograma y guardarlos en disco
202         Mat imgDrawing=Mat::zeros(ALTO_IMG_HISTO, ANCHO_IMG_HISTO, CV_8UC1);
203         dibujaHistograma(imgDrawing, histoULBPVarSuma, (P+2)*16+16);
204         imwrite(string(DATA_FOLDER)+string("histoULBPVar.png"), imgDrawing);
205 
206         namedWindow("Histo ULBPVar");
207         moveWindow("Histo ULBPVar",ANCHO_IMG_HISTO, 0);
208         imshow("Histo ULBPVar", imgDrawing);
209         waitKey(0);
210 
211         return 0;
212     }
213 
214     /***************************************************************************/
215     /** Función: calculaCuantificacionVar                                   **/
216     /**                                                                     **/
217     /** Descripción: se pertende crear un histograma que en vez de tener     **/
218     /**     VAR_MAX cubetas (600), tenga NUM_BINS_VAR (16), y que estas       **/
219     /**     cubetas estén lo más equilibradas posibles. Devuelve un vector    **/
220     /**     indicando las "filas de corte" del histograma original.          **/
221     /***************************************************************************/
222     vector<int> calculaCuantificacionVar(const Mat &histoVarSuma, int numBins)
223     {
224         vector<int> vectQuant;
225         int numTotalPixeles=0;
226         int pixPerBin;
227 
228         // Sumamos el numero de pixeles del histograma
229         for(int i=0; i<histoVarSuma.rows; i++)
230             numTotalPixeles += histoVarSuma.at<float>(i);
231 
232         // Calcula el numero de pixeles por cada bin
```

```
233     pixPerBin = static_cast<float>(numTotalPixeles)/numBins;
234     cout << endl << "Numero total de pixeles:" << numTotalPixeles << endl;
235     cout << "Numero de pixeles por bin: " << pixPerBin << endl;
236 
237     // Se crea el nuevo histograma, intentando que cada cubeta idx contenga
238     // "pixPerBin" pixeles.
239     int numPixAcc=0; // N de pixeles real de cada cubeta
240     int k=0;         // Fila del histograma original
241 
242     for(int idx=0; idx<numBins; idx++)
243     {
244         int umbralNivel= numPixAcc + pixPerBin;
245         cout << "idx:" << idx << " umbral Nivel:" << umbralNivel << endl;
246         numPixAcc += histoVarSuma.at<float>(k++);
247 
248         while(numPixAcc<umbralNivel && k<histoVarSuma.rows)
249             numPixAcc += histoVarSuma.at<float>(k++);
250 
251         if (k>=VAR_MAX)
252             break;
253 
254         vectQuant.push_back(k);
255         cout << " k:" << k << "  numPixAcc" << numPixAcc << endl;
256     }
257 
258     return vectQuant;
259 }
260 
261 /***************************************************************************/
262 /** Función: cuantifHistoVar                                              **/
263 /**                                                                       **/
264 /** Descripción: generamos el histograma cuantificado a partir de los     **/
265 /**     datos del vector de cuantificación calculado en la función        **/
266 /**     calculaCuantificacionVar.                                         **/
267 /***************************************************************************/
268 void cuantifHistoVar(const Mat &histoVarSuma, vector<int>vectQuant,  Mat &
        histoVarQ)
269 {
270     int numPixAcc=0;
271     int k=0;
272 
273     for(size_t idx=0; idx<vectQuant.size(); idx++)
274     {
275         numPixAcc=0;
276         while( k<vectQuant[idx])
277             numPixAcc +=histoVarSuma.at<float>(k++);
278 
279         histoVarQ.at<float>(idx)=numPixAcc;
280     }
281 
282     int numPixRestantes=0;
283     for(int i=vectQuant[vectQuant.size()-1]; i<histoVarSuma.rows; i++)
284         numPixRestantes += histoVarSuma.at<float>(i);
285 
286     histoVarQ.at<float>(vectQuant.size())=numPixRestantes;
287 }
```

B.3. *main.cpp* ("LBP y varianza combinado, método 2")

```
1   /******************************************************************************/
2   /**                         Trabajo Fin de Grado                             **/
3   /**                                                                          **/
4   /**                 Detección de Gasas Quirúrgicas en imágenes               **/
5   /**                  laparoscópicas empleando la técnica LBP                 **/
6   /**                                                                          **/
7   /**         Grado en Ingeníeria Electrónica Industrial y Automática          **/
8   /**                         Universidad de Valladolid                        **/
9   /**                                                                          **/
10  /** Autor: Álvaro Muñoz García                                               **/
11  /** Tutor: Eusebio de la Fuente López                                        **/
12  /******************************************************************************/
13  /** Proyecto: Generador de patrón                                            **/
14  /** Archivo: main.cpp                                                        **/
15  /** Variante: LBP y varianza combinado - Método 2                            **/
16  /**                                                                          **/
17  /** Descripción: se encarga de generar el archivo "datosPatron.xml" en la   **/
18  /**     carpeta "patternData" que contiene la información de todas las       **/
19  /**     imágenes patrón. Para ello, crea un histograma ULBP en el que en     **/
20  /**     vez de sumar una unidad cuando se encuentra un valor de píxel de     **/
21  /**     un determinado valor, suma el valor de la varianaza en esa           **/
22  /**     posición. Por último, normalizamos el histograma y generamos un      **/
23  /**     archivo con esta información.                                        **/
24  /******************************************************************************/
25
26  #include "opencv2/opencv.hpp"
27  #include "../../include/histo.h"
28  #include "../../include/fichero.h"
29  #include "../../include/lbp.h"
30  #include "../../include/estadist.h"
31
32  /// Parámetros del histograma de varianza
33  #define NUM_BINS_VAR 16  // Numero de cubetas del histograma cuantificado.
        Debe ser potencia de 2.
34  #define VAR_MAX 600      // Numero de cubetas del histograma inicial
35  //#define PARAM_MANUALES // Descomentar para elegir P, R y factor de escala.
36
37  using namespace std;
38  using namespace cv;
39
40  int main()
41  {
42      /// 1. Parámetros del codigo LBP y declaración de variables
43      int R, P;
44      double factorEscala = 1.0;
45
46  #ifdef PARAM_MANUALES
47      cout << "Intro R(radio) y P(num.vecinos)" << endl;
48      cout << "R: ";
49      cin >> R;
50      cout << "P: ";
51      cin >> P;
52  #else
53      R = 1;
54      P = 8;
55  #endif // PARAM_MANUALES
```

```
56
57      Mat imgPatron; // Contiene la imagen patron
58      Mat histoVar;  // Histograma de varianzas
59      Mat histoVarSuma = Mat::zeros(VAR_MAX, 1, CV_32F); // 1 columna, VAR_MAX
            filas
60      Mat imgVar;    // Varianza de cada vecindario
61      Mat imgLBP, imgULBP;
62      Mat histoULBP;
63      Mat histoULBPsuma = Mat::zeros(P+2, 1, CV_32F);    // 1 columna, P+2 filas
64
65      string fileName = "../patternImg/patron"; // Nombre base
66      bool hayMasFicheros = true; // Itera hasta que sea false
67      string strNum;              // Numero del fichero
68
69      /// 2. Inicialización look-up table para transformación LBP -> ULBP
70      vector<int> LUT;
71      leeFicheroLUT(LUT, P);
72
73      /// 3. Leemos primera imagen patrón
74      int numFich = 1;
75      strNum = to_string(numFich);
76      imgPatron = imread("../patternImg/patron1.png", IMREAD_GRAYSCALE);
77
78      if(!imgPatron.data)
79      {
80          cout << "No se encuentra fichero \"patron1.png\"" << endl;
81          cout << "Debe haber al menos un fichero en el directorio ../patternImg
                : " << "el fichero patron1.png" << endl;
82          return -1;
83      }
84
85      else
86          cout << "Leyendo primer fichero: \"patron1.png\"" << endl;
87
88      /// 4. Bucle que lee todos los patrones
89      while(hayMasFicheros)
90      {
91          /// 5. Calculamos la varianza
92          if(R == 1 && P == 8)
93              varianza(imgPatron, imgVar);
94          else
95              VARLBP(imgPatron, imgVar, R, P);
96
97          /// 6. Calculamos imagen LBP
98          Mat EX;
99          media<uchar>(imgPatron, EX, R, P);
100         mLBP(imgPatron, EX, imgLBP, R, P); // mLBP solo para P=1 y R=8
101         //LBP(imgPatron, imgLBP, R, P); // LBP para tamaño arbitrario
102
103         if(imgLBP.type() == CV_8UC1)
104             LBP2ULBP<uchar>(imgLBP, LUT, imgULBP);
105         else
106             LBP2ULBP<int>(imgLBP, LUT, imgULBP);
107
108         /// 7. Calculamos histograma ULBP + Var
109         calcularHistoVar(imgULBP, imgVar, histoULBP, P+2);
110         histoULBPsuma += histoULBP; // Suma histogramas ULBP
111
112         // Carga fichero siguiente si existe, para seguir el mismo proceso
113         // durante la siguiente iteración.
114         numFich++;
```

```
115         strNum = to_string(numFich);
116         imgPatron = imread(fileName + strNum + ".png", IMREAD_GRAYSCALE);
117
118         if(!imgPatron.data)
119         {
120             cout << "No encuentro fichero "<< fileName+strNum+".png" << endl;
121             cout << "Termina lectura de ficheros. "  << endl;
122             hayMasFicheros=false;
123         }
124
125         else
126             cout << "Leyendo " << fileName + strNum + ".png" << endl;
127     }
128
129     /// 8. Normalización del histograma
130     normalize(histoULBPsuma, histoULBPsuma, 0, 1, NORM_MINMAX, -1, Mat());
131     histoULBPsuma = comprobarHistograma(histoULBPsuma);
132
133     /// 9. Guardar resultados en fichero datosPatron.xml
134     string fichPatron = DATA_FOLDER + string("datosPatron.xml");
135     cout << endl << "Grabando datos en el fichero " << fichPatron << endl;
136     FileStorage fp(fichPatron,FileStorage::WRITE);
137     time_t rawtime;
138     time(&rawtime);
139
140     fp << "Date" << asctime(localtime(&rawtime));
141     fp << "factorEscala" << factorEscala;
142     fp << "radio" << R;
143     fp << "numVecinos" << P;
144     fp << "histoULBP" << histoULBPsuma;
145
146     fp.release();
147
148     /// 10. Dibujar histogramas y guardarlos en disco
149     Mat imgDrawing=Mat::zeros(ALTO_IMG_HISTO, ANCHO_IMG_HISTO, CV_8UC1);
150     dibujaHistograma(imgDrawing, histoULBPsuma, P+2);
151     imwrite(string(DATA_FOLDER)+string("histoULBP.png"), imgDrawing);
152
153     namedWindow("Histo ULBP");
154     moveWindow("Histo ULBP", ANCHO_IMG_HISTO, 0);
155     imshow("Histo ULBP", imgDrawing);
156     waitKey(0);
157 }
```

B.4. *datosPatron.xml* ("LBP" y "LBP y varianza por separado")


```
1  <?xml version="1.0"?>
2  <opencv_storage>
3  <Date>"Wed May 16 09:30:01 2018&#x0a;"</Date>
4  <factorEscala>1.</factorEscala>
5  <radio>1</radio>
6  <numVecinos>8</numVecinos>
7  <tamHistoVar>16</tamHistoVar>
8  <vectorQuant>
9    2 3 4 6 8 10 13 16 20 26 35 53</vectorQuant>
10 <histoVar type_id="opencv-matrix">
11   <rows>16</rows>
12   <cols>1</cols>
13   <dt>f</dt>
14   <data>
15     1. 6.13584816e-01 5.50324500e-01 9.10619497e-01 7.18594909e-01
16     5.80236852e-01 6.79646969e-01 5.18406212e-01 5.24816513e-01
17     5.53349555e-01 5.20815492e-01 5.12737811e-01 4.91104811e-01 0. 0. 0.</data
         ></histoVar>
18 <histoULBP type_id="opencv-matrix">
19   <rows>10</rows>
20   <cols>1</cols>
21   <dt>f</dt>
22   <data>
23     0. 1.06842560e-03 2.54281182e-02 4.01138246e-01 1. 5.46903193e-01
24     4.58959825e-02 3.57377040e-03 1.59543328e-04 3.12174827e-01</data></
         histoULBP>
25 </opencv_storage>
```

B.5. *datosPatron.xml* ("LBP y varianza combinado, método 1")


```xml
<?xml version="1.0"?>
<opencv_storage>
<Date>"Wed May 16 10:28:52 2018&#x0a;"</Date>
<factorEscala>1.</factorEscala>
<radio>1</radio>
<numVecinos>8</numVecinos>
<tamHistoVar>16</tamHistoVar>
<vectorQuant>
  2 3 4 6 8 10 13 16 20 26 35 53</vectorQuant>
<histoULBPVar type_id="opencv-matrix">
  <rows>176</rows>
  <cols>1</cols>
  <dt>f</dt>
  <data>
    0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 7.86917936e-03
    4.81447671e-04 1.24512328e-04 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
    8.84203538e-02 3.03644054e-02 2.07437538e-02 2.56827418e-02
    1.38623724e-02 8.33402481e-03 7.17190979e-03 3.88478464e-03
    2.91358843e-03 1.85108325e-03 1.01270026e-03 6.05959969e-04
    2.15821361e-04 0. 0. 0. 3.35004568e-01 2.54328877e-01 2.49091059e-01
    4.15555745e-01 3.30347806e-01 2.60106236e-01 2.91060001e-01
    2.12700263e-01 2.07744658e-01 2.03038096e-01 1.78600475e-01
    1.61011040e-01 1.36365905e-01 0. 0. 0. 3.62164855e-01 3.48966539e-01
    3.73063833e-01 7.29011357e-01 6.62032008e-01 5.82667887e-01
    7.47821033e-01 6.08425319e-01 6.51108146e-01 7.29077756e-01
    7.24960566e-01 7.57051528e-01 7.88088322e-01 0. 0. 0. 4.19374108e-01
    3.28704238e-01 3.10376018e-01 5.36515296e-01 4.32331711e-01
    3.44293177e-01 4.03569341e-01 3.05038601e-01 3.00622553e-01
    3.04582059e-01 2.72848010e-01 2.50643313e-01 2.01568857e-01 0. 0. 0.
    1.60438284e-01 5.12575731e-02 3.72872911e-02 4.36706245e-02
    2.55665313e-02 1.53316176e-02 1.44268284e-02 7.98539072e-03
    5.62795717e-03 4.46584215e-03 2.29932764e-03 1.27832650e-03
    4.89748491e-04 0. 0. 0. 2.69859713e-02 1.09570846e-03 3.56935343e-04
    2.90528755e-04 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.28662738e-03 0.
    0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 3.90802681e-01
    2.65510082e-01 3.28297496e-01 1.85232833e-01 1.10824272e-01
    9.56586674e-02 5.11413626e-02 3.76608297e-02 2.67286468e-02
    1.54395280e-02 7.95218721e-03 2.26612436e-03 0. 0. 0. 0. 0. 0. 0. 0.
    0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.</data></histoULBPVar>
</opencv_storage>
```

Anexo C

Código fuente: archivos comunes

C.1. *estadist.h* y *estadist.cpp*

estadist.h

```
/*************************************************************************/
/**                      Trabajo Fin de Grado                           **/
/**                                                                     **/
/**            Detección de Gasas Quirúrgicas en imágenes               **/
/**             laparoscópicas empleando la técnica LBP                 **/
/**                                                                     **/
/**       Grado en Ingeníeria Electrónica Industrial y Automática       **/
/**                      Universidad de Valladolid                      **/
/**                                                                     **/
/** Autor: Álvaro Muñoz García                                          **/
/** Tutor: Eusebio de la Fuente López                                   **/
/*************************************************************************/
/** Proyecto: Común                                                     **/
/** Archivo: estadist.h                                                 **/
/*************************************************************************/

#ifndef VARIANZA_H_INCLUDED
#define VARIANZA_H_INCLUDED

void VARLBP(const cv::Mat& img, cv::Mat& imgVar, int R, int numVec);
void varianza(const cv::Mat& img, cv::Mat& imgVarianza);
template <typename Tipo> void media(const cv::Mat& img, cv::Mat& imgMedia, int
     R, int P);
template <typename Tipo> void alCuadrado(const cv::Mat& img, cv::Mat& imgCuad)
    ;
void cuantImgVar(const cv::Mat& imgVar, cv::Mat& imgCuant, std::vector<int>&
    vectQuant);

#endif // VARIANZA_H_INCLUDED
```

estadist.cpp

```
/****************************************************************************/
/**                         Trabajo Fin de Grado                           **/
/**                                                                        **/
/**                Detección de Gasas Quirúrgicas en imágenes              **/
/**                 laparoscópicas empleando la técnica LBP                **/
/**                                                                        **/
/**          Grado en Ingeniería Electrónica Industrial y Automática       **/
/**                         Universidad de Valladolid                      **/
/**                                                                        **/
/** Autor: Álvaro Muñoz García                                             **/
/** Tutor: Eusebio de la Fuente López                                      **/
/****************************************************************************/
/** Proyecto: Común                                                        **/
/** Archivo: estadist.cpp                                                  **/
/****************************************************************************/

#include "opencv2/opencv.hpp"
#include "../include/estadist.h"

using namespace std;
using namespace cv;

/****************************************************************************/
/** Función: VARLBP                                                        **/
/**                                                                        **/
/** Descripción: calcula la varianza de una imagen con vecindarios de      **/
/**     radio y número de vecinos arbitrario. La imagen resultado es       **/
/**     imgVar.                                                            **/
/****************************************************************************/
void VARLBP(const Mat& img, Mat& imgVar, int R, int numVec)
{
    imgVar = Mat::zeros(img.rows-2*R, img.cols-2*R, CV_32FC1);
    max(min(numVec,31),1); // Fuerza que 0 < P < 32

    Mat EX  = Mat::zeros(img.rows-2*R, img.cols-2*R, CV_32FC1); //E[X]
    Mat E2X = Mat::zeros(img.rows-2*R, img.cols-2*R, CV_32FC1); //(E[X])^2
    Mat EX2 = Mat::zeros(img.rows-2*R, img.cols-2*R, CV_32FC1); //(E[X^2])

    for(int n=0; n<numVec; n++)
    {
        // Puntos de muestreo
        float x = static_cast<float>(R) * cos(2.0*M_PI*n/static_cast<float>(
            numVec));
        float y = static_cast<float>(R) * -sin(2.0*M_PI*n/static_cast<float>(
            numVec));

        // Indices relativos
        int fx = static_cast<int>(floor(x));
        int fy = static_cast<int>(floor(y));
        int cx = static_cast<int>(ceil(x));
        int cy = static_cast<int>(ceil(y));

        // Parte decimal
        float ty = y - fy;
        float tx = x - fx;

        // Pesos de la interpolacion
        float w1 = (1 - tx) * (1 - ty);
        float w2 =      tx  * (1 - ty);
```

```
58          float w3 = (1 - tx) *      ty;
59          float w4 =      tx   *      ty;
60
61          // Iteramos sobre la imagen de entrada
62          for(int i=R; i < img.rows-R; i++)
63          {
64              float *pEX = EX.ptr<float>(i-R);
65              float *pEX2 = EX2.ptr<float>(i-R);
66              for(int j=R; j < img.cols-R; j++)
67              {
68                  float t = w1*img.at<uchar>(i+fy,j+fx) + w2*img.at<uchar>(i+fy,
                        j+cx)
69                        + w3*img.at<uchar>(i+cy,j+fx) + w4*img.at<uchar>(i+
                              cy,j+cx);
70                  pEX[j-R]   += t;
71                  pEX2[j-R]  += t*t;
72              }
73          }
74      }
75
76      // Var[X] = E[X^2] - (E[X])^2
77      EX  =  EX / numVec;
78      EX2 =  EX2 / numVec;
79
80      // Ahora se calcula (E[X])^2, que se ha llamado E2X
81      for(int i=0; i < EX.rows; i++)
82      {
83          const float *pEX = EX.ptr<float>(i);
84          float *pE2X   = E2X.ptr<float>(i);
85
86          for(int j=0; j < EX.cols; j++)
87              pE2X[j] = pEX[j]*pEX[j];
88      }
89
90      // Calculamos el resultado
91      for(int i = 0; i < imgVar.rows; i++)
92      {
93          float *pimgVar = imgVar.ptr<float>(i);
94          const float *pEX2 = EX2.ptr<float>(i);
95          const float *pE2X = E2X.ptr<float>(i);
96
97          for(int j = 0; j < imgVar.cols; j++)
98              pimgVar[j]=pEX2[j]-pE2X[j];
99      }
100 }
101
102 /*****************************************************************************/
103 /** Función: varianza                                                      **/
104 /**                                                                        **/
105 /** Descripción: se calcula la varianza de una imagen para vecindarios     **/
106 /**     P = 8 y R = 1. Para ello se sigue la fórmula:                      **/
107 /**     Var[X] = E[X^2] - (E[X])^2                                         **/
108 /*****************************************************************************/
109 void varianza(const Mat& img, Mat& imgVarianza)
110 {
111     Mat EX(img.rows-2, img.cols-2, CV_32FC1);  // E[X]
112     Mat E2X(img.rows-2, img.cols-2, CV_32FC1); // (E[X])^2
113     Mat X2(img.rows, img.cols, CV_32FC1);      // X^2
114     Mat EX2(img.rows-2, img.cols-2, CV_32FC1); // E[X^2]
115
116     media<uchar>(img, EX, 1, 8);
```

```
117     alCuadrado<float>(EX, E2X);
118     alCuadrado<uchar>(img, X2);
119     media<float>(X2, EX2, 1, 8);
120     imgVarianza = EX2 - E2X; // Var[X] = E[X^2] - (E[X])^2
121 }
122
123 /*****************************************************************************/
124 /** Función: media                                                         **/
125 /**                                                                        **/
126 /** Descripción: calcula la imagen de medias de un vecindario arbitrario.  **/
127 /**     Para ello recibe la imagen img y los parámetros R y P, y devuelve  **/
128 /**     la imagen de medias imgMedia.                                      **/
129 /*****************************************************************************/
130 template <typename Tipo>
131 void media(const Mat& img, Mat& imgMedia, int R, int P)
132 {
133     imgMedia = Mat::zeros(img.rows-2*R, img.cols-2*R, CV_32FC1);
134
135     if(R == 1 && P == 8)
136     {
137         for(int i=1; i < img.rows-1; i++)
138         {
139             const Tipo *p_ant = img.ptr<Tipo>(i-1);
140             const Tipo *p     = img.ptr<Tipo>(i);
141             const Tipo *p_sig = img.ptr<Tipo>(i+1);
142
143             float *pimgMedia = imgMedia.ptr<float>(i-1);
144             for(int j=1; j < img.cols-1; j++)
145             {
146                 float suma;
147                 float media;
148                 suma =  p_ant[j-1] +  p_ant[j] +  p_ant[j+1]  +
149                         p[j-1]                 +  p[j+1]      +
150                         p_sig[j-1] +  p_sig[j] +  p_sig[j+1]  ;
151                 media = suma / 8.0;
152
153                 pimgMedia[j-1]=media;
154             }
155         }
156     }
157
158     else
159     {
160         for(int n = 0; n < P; n++)
161         {
162             // Puntos de muestreo
163             float x = static_cast<float>(R) * cos(2.0*M_PI*n/static_cast<float
                    >(P));
164             float y = static_cast<float>(R) * -sin(2.0*M_PI*n/static_cast<
                    float>(P));
165
166             // Indices relativos
167             int fx = static_cast<int>(floor(x));
168             int fy = static_cast<int>(floor(y));
169             int cx = static_cast<int>(ceil(x));
170             int cy = static_cast<int>(ceil(y));
171
172             // Parte decimal
173             float ty = y - fy;
174             float tx = x - fx;
175
```

```
176             // Pesos de la interpolacion
177             float w1 = (1 - tx) * (1 - ty);
178             float w2 =      tx  * (1 - ty);
179             float w3 = (1 - tx) *      ty;
180             float w4 =      tx  *      ty;
181 
182             // Iteramos sobre la imagen de entrada
183             for(int i=R; i < img.rows-R; i++)
184             {
185                 float *pimgMedia = imgMedia.ptr<float>(i-R);
186 
187                 for(int j=R; j < img.cols-R; j++)
188                 {
189                     float t = w1*img.at<uchar>(i+fy,j+fx) + w2*img.at<uchar>(i
                            +fy,j+cx)
190                                   + w3*img.at<uchar>(i+cy,j+fx) + w4*img.at<uchar
                                        >(i+cy,j+cx);
191                     pimgMedia[j-R] += t;
192                 }
193             }
194         }
195 
196         imgMedia  =  imgMedia / P;
197     }
198 }
199 
200 /*************************************************************************/
201 /** Función: alCuadrado                                               **/
202 /**                                                                   **/
203 /** Descripción: se eleva el valor de cada pixel al cuadrado y lo almacena **/
204 /**     en la imagen imgCuad.                                         **/
205 /*************************************************************************/
206 template <typename Tipo>
207 void alCuadrado(const Mat& img, Mat& imgCuad)
208 {
209     imgCuad = Mat::zeros(img.rows, img.cols, CV_32FC1);
210 
211     for(int i=0; i < img.rows; i++)
212     {
213         const Tipo *p     = img.ptr<Tipo>(i);
214         float *pimgCuad   = imgCuad.ptr<float>(i);
215 
216         for(int j=0; j < img.cols; j++)
217             pimgCuad[j] = p[j]*p[j];
218     }
219 }
220 
221 /*************************************************************************/
222 /** Función: cuantImgVar                                              **/
223 /**                                                                   **/
224 /** Descripción: se cuantifica una imagen varianza de entrada (imgVar)   **/
225 /**     a partir del vector de cuantización (vectQuant) y se guardan en   **/
226 /**     una imagen nueva (imgCuant).                                  **/
227 /*************************************************************************/
228 void cuantImgVar(const Mat& imgVar, Mat& imgCuant, vector<int>& vectQuant)
229 {
230     imgCuant = Mat::zeros(imgVar.rows, imgVar.cols, CV_8UC1);
231 
232     for(int i=0; i < imgVar.rows; i++)
233     {
234         const float *pVar = imgVar.ptr<float>(i);
```

```
235         uchar *pimgCuant  = imgCuant.ptr<uchar>(i);
236
237         for(int j=0; j < imgVar.cols; j++)
238         {
239             size_t k=0;
240             while((vectQuant[k] < pVar[j]) && (k < vectQuant.size()))
241                 k++;
242             pimgCuant[j] = k;
243         }
244     }
245 }
```

C.2. *fichero.h* y *fichero.cpp*

fichero.h

```
/****************************************************************************/
/**                          Trabajo Fin de Grado                          **/
/**                                                                        **/
/**            Detección de Gasas Quirúrgicas en imágenes                  **/
/**             laparoscópicas empleando la técnica LBP                    **/
/**                                                                        **/
/**       Grado en Ingeníeria Electrónica Industrial y Automática          **/
/**                          Universidad de Valladolid                     **/
/**                                                                        **/
/** Autor: Álvaro Muñoz García                                             **/
/** Tutor: Eusebio de la Fuente López                                      **/
/****************************************************************************/
/** Proyecto: Común                                                        **/
/** Archivo: fichero.h                                                     **/
/****************************************************************************/

#ifndef FICHERO_H_INCLUDED
#define FICHERO_H_INCLUDED

#include <fstream>
#include "opencv2/opencv.hpp"

#define DATA_FOLDER "../patternData/"
#define IMG_FOLDER "../images/"
#define LUTS_FOLDER "../LUTS/"

#define ERROR_NO_FILE_PATRON 1
#define ERROR_NO_FILE_IMG 2
#define ERROR_NO_FILE_LUT 3

void leeFicheroDatosPatron(double &factorEscala, int &R, int &P, cv::Mat &
    histoPatronULBP);
void leeFicheroDatosPatronVar(double &factorEscala, int &R, int &P, std::
    vector<int>& vectQ, cv::Mat &histoPatronULBP, cv::Mat &histoPatronVar);
void leeFicheroDatosPatronULBPVar(double &factorEscala, int &R, int &P, std::
    vector<int>& vectQ, cv::Mat &histoPatronULBPVar);
void leeFicheroLUT(std::vector<int>& ulbpLUT, int P);
void leeFicheroImagen(cv::Mat &imgColor, std::string fileNumber);

#endif // FICHERO_H_INCLUDED
```

fichero.cpp

```
/****************************************************************************/
/**                        Trabajo Fin de Grado                            **/
/**                                                                        **/
/**               Detección de Gasas Quirúrgicas en imágenes               **/
/**                laparoscópicas empleando la técnica LBP                 **/
/**                                                                        **/
/**        Grado en Ingeniería Electrónica Industrial y Automática         **/
/**                       Universidad de Valladolid                        **/
/**                                                                        **/
/** Autor: Álvaro Muñoz García                                             **/
/** Tutor: Eusebio de la Fuente López                                      **/
/****************************************************************************/
/** Proyecto: Común                                                        **/
/** Archivo: fichero.cpp                                                   **/
/****************************************************************************/

#include "../include/fichero.h"

using namespace std;
using namespace cv;

/****************************************************************************/
/** Función: leeFicheroDatosPatron                                         **/
/**                                                                        **/
/** Descripción: encargado de inicializar las variables factorEscala, R, P **/
/**     e histoPatronULBP a partir de los datos del patrón.                **/
/****************************************************************************/
void leeFicheroDatosPatron(double &factorEscala, int &R, int &P, Mat &
    histoPatronULBP)
{
    string filePatron = DATA_FOLDER + string("datosPatron.xml");
    cout << "Leyendo datos del fichero "  << filePatron << endl;
    FileStorage fs(filePatron, FileStorage::READ);

    /// Leemos el archivo datosPatron.xml e inicializamos variables
    if(fs.isOpened())
    {
        fs["factorEscala"] >> factorEscala;
        fs["radio"] >> R;
        fs["numVecinos"] >> P;
        fs["histoULBP"] >> histoPatronULBP;
    }

    else
    {
        cerr << "No puedo abrir fichero " << filePatron << endl;
        throw ERROR_NO_FILE_PATRON;
    }

    fs.release();

    /// Sacamos por pantalla los datos leidos
    cout << "radio: " << R << endl;
    cout << "numVecinos: " << P << endl;

    /// Comprobamos que ha cargado bien el histograma
    if(histoPatronULBP.empty() )
    {
        cerr << "ERROR al cargar fichero de datos " << filePatron << endl;
```

```
59            throw ERROR_NO_FILE_PATRON;
60        }
61
62        else
63            cout << "Lectura fichero de datos patron finalizada con exito" << endl
                  << endl;
64    }
65
66    /***************************************************************************/
67    /** Función: leeFicheroDatosPatronVar                                    **/
68    /**                                                                      **/
69    /** Descripción: encargado de inicializar las variables factorEscala, R, P **/
70    /**     vectQ, histoPatronULBP e histoPatronVar a partir de los datos del  **/
71    /**     patrón.                                                          **/
72    /***************************************************************************/
73    void leeFicheroDatosPatronVar(double &factorEscala, int &R, int &P, vector<int
          >& vectQ,  Mat &histoPatronULBP, Mat &histoPatronVar)
74    {
75        string filePatron = DATA_FOLDER + string("datosPatron.xml");
76        cout << "Leyendo datos del fichero "  << filePatron << endl;
77        FileStorage fs(filePatron, FileStorage::READ);
78
79        /// Leemos el archivo datosPatron.xml e inicializamos variables
80        if(fs.isOpened())
81        {
82            fs["factorEscala"] >> factorEscala;
83            fs["radio"] >> R;
84            fs["numVecinos"] >> P;
85            fs["vectorQuant"] >> vectQ;
86            fs["histoVar"] >> histoPatronVar;
87            fs["histoULBP"] >> histoPatronULBP;
88        }
89
90        else
91        {
92            cerr << "No puedo abrir fichero " << filePatron << endl;
93            throw ERROR_NO_FILE_PATRON;
94        }
95
96        fs.release();
97
98        /// Sacamos por pantalla los datos leidos
99        cout << "radio: " << R << endl;
100       cout << "numVecinos: " << P << endl;
101       cout << "vectorQuant:" ;
102       for(size_t i = 0; i < vectQ.size(); i++)
103           cout << vectQ[i] << " , ";
104       cout << endl;
105
106       /// Comprobamos que han cargado bien los histogramas
107       if(histoPatronULBP.empty() && histoPatronVar.empty())
108       {
109           cerr << "ERROR al cargar fichero de datos " << filePatron << endl;
110           throw ERROR_NO_FILE_PATRON;
111       }
112
113       else
114           cout << "Lectura fichero de datos patron finalizada con exito" << endl
                 << endl;
115   }
116
```

```
117 /*************************************************************************/
118 /** Función: leeFicheroDatosPatronULBPVar                               **/
119 /**                                                                     **/
120 /** Descripción: encargado de inicializar las variables factorEscala, R, P **/
121 /**     vectQ e histoPatronULBPVar.                                     **/
122 /*************************************************************************/
123 void leeFicheroDatosPatronULBPVar(double &factorEscala, int &R, int &P, vector
        <int>& vectQ,  Mat &histoPatronULBPVar)
124 {
125     string filePatron = DATA_FOLDER + string("datosPatron.xml");
126     cout << "Leyendo datos del fichero "  << filePatron << endl;
127     FileStorage fs(filePatron, FileStorage::READ);
128 
129     /// Leemos el archivo datosPatron.xml e inicializamos variables
130     if(fs.isOpened())
131     {
132         fs["factorEscala"] >> factorEscala;
133         fs["radio"] >> R;
134         fs["numVecinos"] >> P;
135         fs["vectorQuant"] >> vectQ;
136         fs["histoULBPVar"] >> histoPatronULBPVar;
137     }
138 
139     else
140     {
141         cerr << "No puedo abrir fichero " << filePatron << endl;
142         throw ERROR_NO_FILE_PATRON;
143     }
144 
145     fs.release();
146 
147     /// Sacamos por pantalla los datos leidos
148     cout << "radio: " << R << endl;
149     cout << "numVecinos: " << P << endl;
150     cout << "vectorQuant:" ;
151     for(size_t i = 0; i < vectQ.size(); i++)
152         cout << vectQ[i] << " , ";
153     cout << endl;
154 
155     /// Comprobamos que han cargado bien los histogramas
156     if(histoPatronULBPVar.empty())
157     {
158         cerr << "ERROR al cargar fichero de datos " << filePatron << endl;
159         throw ERROR_NO_FILE_PATRON;
160     }
161 
162     else
163         cout << "Lectura fichero de datos patron finalizada con exito" << endl
              << endl;
164 }
165 
166 /*************************************************************************/
167 /** Función: leeFicheroLUT                                              **/
168 /**                                                                     **/
169 /** Descripción: carga en la variable ulbpLUT la look-up table          **/
170 /**     corresponiente al número de vecinos P. Debe haber sido generada **/
171 /**     previamente.                                                    **/
172 /**     0-8: patrones uniformes, 9: patrones no uniformes.              **/
173 /*************************************************************************/
174 void leeFicheroLUT(vector<int>& ulbpLUT, int P)
175 {
```

```
176     int tamLUT = pow(2,P);
177     string fileName = string(LUTS_FOLDER) + string("ULBP_LUT_") + to_string(P)
            + ".dat";
178     ifstream fileLUT(fileName);
179 
180     if(!fileLUT.good())
181     {
182         cout << "ERROR abriendo fichero "<< fileName<< endl;
183         throw ERROR_NO_FILE_LUT;
184     }
185 
186     else
187         cout << "Leyendo fichero "<< fileName<< endl;
188 
189     /// Cargamos LUT en la variable ulbpLUT
190     for(int i=0; i<tamLUT; i++)
191     {
192         int code;
193         fileLUT >> code;
194         ulbpLUT.push_back(code);
195     }
196     fileLUT.close();
197 }
198 
199 /*************************************************************************/
200 /** Función: leeFicheroImagen                                          **/
201 /**                                                                    **/
202 /** Descripción: carga en la variable imgColor la imagen elegida por el **/
203 /**     usuario. La imagen se encuentra en la ruta ../images/IMG00#.jpg  **/
204 /*************************************************************************/
205 void leeFicheroImagen(Mat &imgColor, string fileNumber)
206 {
207     string zeros;
208     if (stoi(fileNumber)<10)
209         zeros="000";
210     else zeros="00";
211 
212     string fileName = string(IMG_FOLDER) + "IMG" + zeros + fileNumber + ".jpg"
            ;
213 
214     /// Leemos la imagen elegida por el usuario
215     imgColor = imread(fileName, IMREAD_COLOR);
216     if(!imgColor.data)
217     {
218         cerr <<  "No encuentro fichero imagen " << fileName << endl;
219         throw ERROR_NO_FILE_IMG;
220     }
221 }
```

C.3. *histo.h* y *histo.cpp*

histo.h

```
1  /******************************************************************************/
2  /**                          Trabajo Fin de Grado                            **/
3  /**                                                                          **/
4  /**               Detección de Gasas Quirúrgicas en imágenes                 **/
5  /**                laparoscópicas empleando la técnica LBP                   **/
6  /**                                                                          **/
7  /**        Grado en Ingeniería Electrónica Industrial y Automática           **/
8  /**                          Universidad de Valladolid                       **/
9  /**                                                                          **/
10 /** Autor: Álvaro Muñoz García                                               **/
11 /** Tutor: Eusebio de la Fuente López                                        **/
12 /******************************************************************************/
13 /** Proyecto: Común                                                          **/
14 /** Archivo: histo.h                                                         **/
15 /******************************************************************************/
16
17 #ifndef HISTO_INCLUDED
18 #define HISTO_INCLUDED
19
20 #include "opencv2/opencv.hpp"
21
22 #define ANCHO_IMG_HISTO 512 // Ancho de la imagen donde visualizaremos
       histograma
23 #define ALTO_IMG_HISTO 200  // Alto de la imagen donde visualizaremos
       histograma
24
25 void calcHisto(const cv::Mat& imagen, cv::Mat& hist, int numLevels, float
       minLevel, float maxLevel);
26 void dibujaHistograma(cv::Mat& imgGrisHisto, cv::Mat& histo, int numLevels);
27 cv::Mat comprobarHistograma(cv::Mat histograma);
28 void calcularHistoVar(const cv::Mat& imagenULBP, const cv::Mat& imagenVar, cv
       ::Mat& hist, int numLevels);
29
30 #endif // HISTO_INCLUDED
```

histo.cpp

```
/***************************************************************************/
/**                          Trabajo Fin de Grado                        **/
/**                                                                       **/
/**              Detección de Gasas Quirúrgicas en imágenes              **/
/**               laparoscópicas empleando la técnica LBP                **/
/**                                                                       **/
/**       Grado en Ingeníeria Electrónica Industrial y Automática        **/
/**                          Universidad de Valladolid                   **/
/**                                                                       **/
/** Autor: Álvaro Muñoz García                                           **/
/** Tutor: Eusebio de la Fuente López                                    **/
/***************************************************************************/
/** Proyecto: Común                                                      **/
/** Archivo: histo.cpp                                                   **/
/***************************************************************************/

#include "../include/histo.h"

#define NG_MAX 10000 // Nivel de gris máximo para el histograma de la varianza

using namespace std;
using namespace cv;

/***************************************************************************/
/** Función: calcHisto                                                   **/
/**                                                                       **/
/** Descripción: calcula el histograma de una imagen de entrada utilizando **/
/**     la función nativa de OpenCV cv::calcHist.                        **/
/***************************************************************************/
void calcHisto(const Mat& imagen, Mat& hist, int numLevels, float minLevel,
    float maxLevel)
{
    int histSize[1];        // Un canal
    float hranges[2];       // Rango niveles de gris (minLevel,maxLevel)
    const float* ranges[1]; // Un canal
    int channels[]= {0};    // Canal 0

    hranges[0]  = minLevel;  // Establece el valor minimo
    hranges[1]  = maxLevel;  // Establece el valor maximo
    histSize[0]= numLevels; // Establece el tamaño del histograma
    ranges[0]   = hranges;

    /// Calcula el histograma con la función cv::calcHist
    cv::calcHist(&imagen,
                 1,          // Histograma de una imagen solo
                 channels,   // Canal empleado
                 cv::Mat(),  // No se emplea mascara
                 hist,       // Histograma resultante
                 1,          // Es un histograma 1D
                 histSize,   // Numero de bins en vector histo
                 ranges      // Rango de valores de los pixeles
                );
}

/***************************************************************************/
/** Función: dibujaHistograma                                            **/
/**                                                                       **/
/** Descripción: dibuja el histograma contenido en la imagen histo en la  **/
/**     imagen imgGrisHisto. Para ello une los extremos de las barras    **/
```

```
59  /**     del histograma.                                                 **/
60  /*************************************************************************/
61  void dibujaHistograma(Mat& imgGrisHisto, Mat& histo, int numLevels)
62  {
63      Mat histoNorm;
64      auto color = Scalar(255);
65      int ratio = cvRound( (double) ANCHO_IMG_HISTO/(numLevels-1));
66
67      // Normaliza resultado entre 0 y ALTO_IMG_HISTO (altura de imagen
            histograma)
68      normalize(histo, histoNorm, 0, ALTO_IMG_HISTO, NORM_MINMAX);
69
70      //Dibuja lineas uniendo los extremos de las barras del histograma
71      for(int i = 1; i < numLevels; i++)
72      {
73          line( imgGrisHisto,
74                Point( (i-1) * ratio, ALTO_IMG_HISTO - cvRound(histoNorm.at<
                      float>(i-1)) ),
75                Point(  i * ratio,    ALTO_IMG_HISTO - cvRound(histoNorm.at<
                      float>(i)) ),
76                color, 2, 8, 0  );
77      }
78  }
79
80  /*************************************************************************/
81  /** Función: comprobarHistograma                                        **/
82  /**                                                                     **/
83  /** Descripción: comprueba que en el histograma no haya valores muy     **/
84  /**     pequeños o negativos, sustituyéndolos por 0. También cambia por 1  **/
85  /**     aquellos valores que sean muy próximos.                         **/
86  /**     Devuelve un histograma con los cambios que hayan sido necesarios.  **/
87  /*************************************************************************/
88  Mat comprobarHistograma(Mat histograma)
89  {
90      for(int i = 0; i != histograma.rows-1; i++)
91      {
92          float valorPixel = histograma.at<float>(i,0);
93
94          if(valorPixel < 1e-4)
95              histograma.at<float>(i,0) = 0.0;
96
97          if(valorPixel > 0.999)
98              histograma.at<float>(i,0) = 1.0;
99      }
100
101     return histograma;
102 }
103
104 /*************************************************************************/
105 /** Función: calcularHistoVar                                           **/
106 /**                                                                     **/
107 /** Descripción: calcula el histograma de una imagen de entrada. En vez de **/
108 /**     sumar una unidad en la cubeta correspondiente cuando se encuentre  **/
109 /**     un píxel con un valor de gris, se sumará el valor de la varianza   **/
110 /**     para esa misma posición. Como excepción, si el valor de la        **/
111 /**     varianza es 0, se sumara 1.                                     **/
112 /*************************************************************************/
113 void calcularHistoVar(const Mat& imagenULBP, const Mat& imagenVar, Mat& hist,
        int numLevels)
114 {
115     hist = Mat::zeros(numLevels, 1, CV_32F);
```

```
116
117     for(int i = 1; i != imagenULBP.rows-1; i++) // Filas
118     {
119         const uchar *fila_ULBP = imagenULBP.ptr<uchar>(i);
120         const float *fila_Var = imagenVar.ptr<float>(i);
121
122         for(int j = 1; j != imagenULBP.cols-1; j++) // Columnas
123         {
124             uchar valorULBP = fila_ULBP[j];
125             float valorVar = fila_Var[j];
126
127             float *fila_Histo = hist.ptr<float>(valorULBP);
128             fila_Histo[0] += static_cast <int>(valorVar);
129
130             if(valorVar == 0.0)
131                 fila_Histo[0] += 1;
132         }
133     }
134 }
```

C.4. *lbp.h* y *lbp.cpp*

lbp.h

```
1   /***************************************************************************/
2   /**                         Trabajo Fin de Grado                          **/
3   /**                                                                       **/
4   /**               Detección de Gasas Quirúrgicas en imágenes              **/
5   /**                laparoscópicas empleando la técnica LBP                **/
6   /**                                                                       **/
7   /**        Grado en Ingeniería Electrónica Industrial y Automática        **/
8   /**                       Universidad de Valladolid                       **/
9   /**                                                                       **/
10  /** Autor: Álvaro Muñoz García                                            **/
11  /** Tutor: Eusebio de la Fuente López                                     **/
12  /***************************************************************************/
13  /** Proyecto: Común                                                       **/
14  /** Archivo: lbp.h                                                        **/
15  /***************************************************************************/
16
17  #ifndef LBP_H_INCLUDED
18  #define LBP_H_INCLUDED
19
20  void LBP(const cv::Mat& img, cv::Mat& imgLBP, int R, int P);
21  void mLBP(const cv::Mat& img, const cv::Mat& imgMedia, cv::Mat& imgLBP, int R,
         int P);
22
23  /***************************************************************************/
24  /** Función: LBP2ULBP                                                     **/
25  /**                                                                       **/
26  /** Descripción: convierte la imagen LBP (o mLBP) en su equivalente ULBP  **/
27  /**     utilizando la look-up table que se ha cargado anteriormente.      **/
28  /***************************************************************************/
29  template <typename Tipo>
30  void LBP2ULBP(const cv::Mat& imgLBP, const std::vector<int>& LUT, cv::Mat&
        imgULBP)
31  {
32      int nl = imgLBP.rows; // Numero de filas de imgLBP
33      int nc = imgLBP.cols; // Numero de columnas de imgLBP
34      imgULBP = cv::Mat::zeros(nl, nc, CV_8UC1);
35
36      for (int i=0; i<nl; i++)
37      {
38          const Tipo* ptrFila= imgLBP.ptr<Tipo>(i);
39          uchar* ptrUFila= imgULBP.ptr<uchar>(i);
40
41          for (int j=0; j<nc; j++)
42          {
43              Tipo nivelGris = ptrFila[j];
44              ptrUFila[j]=LUT[nivelGris];
45          }
46      }
47  }
48
49  #endif // LBP_H_INCLUDED
```

lbp.cpp

```
/*************************************************************************/
/**                       Trabajo Fin de Grado                        **/
/**                                                                   **/
/**           Detección de Gasas Quirúrgicas en imágenes              **/
/**           laparoscópicas empleando la técnica LBP                 **/
/**                                                                   **/
/**     Grado en Ingeníeria Electrónica Industrial y Automática       **/
/**                    Universidad de Valladolid                      **/
/**                                                                   **/
/** Autor: Álvaro Muñoz García                                        **/
/** Tutor: Eusebio de la Fuente López                                 **/
/*************************************************************************/
/** Proyecto: Común                                                   **/
/** Archivo: lbp.cpp                                                  **/
/*************************************************************************/

#include "opencv2/opencv.hpp"
#include "../include/lbp.h"

#define ERROR_P_TOO_HIGH 32

using namespace std;
using namespace cv;

/*************************************************************************/
/** Función: LBP                                                      **/
/**                                                                   **/
/** Descripción: genera la imagen LBP de una imagen de entrada. Si R = 1 y **/
/**      P = 8, el vecindario será cuadrado y no se realizará interpolación **/
/**      En otro caso, el vecindario es cirular y se lleva a cabo una  **/
/**      interpolación.                                               **/
/*************************************************************************/
void LBP(const Mat& img, Mat& imgLBP, int R, int P)
{
    if((R==1) && (P==8)) // No hay que hacer interpolacion
    {
        // Imagen resultado: 8 bits, unsigned -> uchar
        imgLBP = Mat::zeros(img.rows-2, img.cols-2, CV_8UC1);

        for(int i = 1; i != img.rows-1; i++)
        {
            const uchar *p_ant = img.ptr<uchar>(i-1); // Fila anterior
            const uchar *p     = img.ptr<uchar>(i);   // Fila actual
            const uchar *p_sig = img.ptr<uchar>(i+1); // Fila siguiente
            uchar *pimgLBP = imgLBP.ptr<uchar>(i-1);  // Fila de la imagen LBP

            for(int j = 1; j != img.cols-1; j++)
            {
                uchar centro = p[j];
                uchar code = 0;
                if( p_ant[j-1]   >=  centro ) code=128;
                if( p_ant[j]     >=  centro ) code+=64;
                if( p_ant[j+1]   >=  centro ) code+=32;
                if( p[j+1]       >=  centro ) code+=16;
                if( p_sig[j+1]   >=  centro ) code+=8;
                if( p_sig[j]     >=  centro ) code+=4;
                if( p_sig[j-1]   >=  centro ) code+=2;
                if( p[j-1]       >=  centro ) code+=1;
                pimgLBP[j-1]=code;
```

```
60                  }
61              }
62          }
63
64          else // Hay que interpolar
65          {
66              // Imagen resultado: 32 bits, signed -> int
67              imgLBP = Mat::zeros(img.rows-2*R, img.cols-2*R, CV_32SC1);
68
69              for(int n=0; n<P; n++)
70              {
71                  // Puntos de muestreo
72                  float x = static_cast<float>(R) * cos(2.0*M_PI*n/static_cast<float
                        >(P));
73                  float y = static_cast<float>(R) * -sin(2.0*M_PI*n/static_cast<
                        float>(P));
74
75                  // Indices relativos
76                  int fx = static_cast<int>(floor(x));
77                  int fy = static_cast<int>(floor(y));
78                  int cx = static_cast<int>(ceil(x));
79                  int cy = static_cast<int>(ceil(y));
80
81                  // Parte decimal
82                  float ty = y - fy;
83                  float tx = x - fx;
84
85                  // Pesos de la interpolacion
86                  float w1 = (1 - tx) * (1 - ty);
87                  float w2 =      tx  * (1 - ty);
88                  float w3 = (1 - tx) *      ty;
89                  float w4 =      tx  *      ty;
90
91                  // Iteramos sobre la imagen de entrada
92                  auto potenciaDos = pow(2, n);
93
94                  for(int i=R; i < img.rows-R; i++)
95                  {
96                      int *pimgLBP = imgLBP.ptr<int>(i-R);
97                      const uchar *p  = img.ptr<uchar>(i);
98
99                      for(int j=R; j < img.cols-R; j++)
100                     {
101                         uchar centro = p[j];
102                         float t = w1*img.at<uchar>(i+fy,j+fx) + w2*img.at<uchar>(i
                                +fy,j+cx)
103                                 + w3*img.at<uchar>(i+cy,j+fx) + w4*img.at<uchar
                                    >(i+cy,j+cx);
104
105                         if( t >= centro )
106                             pimgLBP[j-R] += static_cast <int>(potenciaDos);
107                     }
108                 }
109             }
110         }
111     }
112
113     /**********************************************************************/
114     /** Función: mLBP                                                    **/
115     /**                                                                  **/
116     /** Descripción: al igual que la función LBP, calcula la imagen LBP, pero  **/
```

```
117 /**     en vez de utilizar el píxel central, se sustituye su valor por la  **/
118 /**     media de los niveles de gris de todos los vecinos.                 **/
119 /***********************************************************************/
120 void mLBP(const Mat& img, const Mat& imgMedia, Mat& imgLBP, int R, int P)
121 {
122     if((R==1) && (P==8)) // No hay que hacer interpolacion
123     {
124         // Imagen resultado: 8 bits, unsigned -> uchar
125         imgLBP = Mat::zeros(img.rows-2, img.cols-2, CV_8UC1);
126 
127         for(int i = 1; i != img.rows-1; i++)
128         {
129             const float *pMedia= imgMedia.ptr<float>(i-1);
130             const uchar *p_ant = img.ptr<uchar>(i-1);
131             const uchar *p     = img.ptr<uchar>(i);
132             const uchar *p_sig = img.ptr<uchar>(i+1);
133             uchar *pimgLBP = imgLBP.ptr<uchar>(i-1);
134 
135             for(int j = 1; j != img.cols-1; j++)
136             {
137                 float media = pMedia[j-1];
138                 uchar code = 0;
139                 if( p_ant[j-1]   >=  media ) code=128;
140                 if( p_ant[j]     >=  media ) code+=64;
141                 if( p_ant[j+1]   >=  media ) code+=32;
142                 if( p[j+1]       >=  media ) code+=16;
143                 if( p_sig[j+1]   >=  media ) code+=8;
144                 if( p_sig[j]     >=  media ) code+=4;
145                 if( p_sig[j-1]   >=  media ) code+=2;
146                 if( p[j-1]       >=  media ) code+=1;
147                 pimgLBP[j-1]=code;
148             }
149         }
150     }
151 
152     else // Hay que interpolar
153     {
154         // Imagen resultado: 32 bits, signed -> int
155         imgLBP = Mat::zeros(img.rows-2*R, img.cols-2*R, CV_32SC1);
156 
157         for(int n = 0; n < P; n++)
158         {
159             // Puntos de muestreo
160             float x = static_cast<float>(R) * cos(2.0*M_PI*n/static_cast<float
                    >(P));
161             float y = static_cast<float>(R) * -sin(2.0*M_PI*n/static_cast<
                    float>(P));
162 
163             // Indices relativos
164             int fx = static_cast<int>(floor(x));
165             int fy = static_cast<int>(floor(y));
166             int cx = static_cast<int>(ceil(x));
167             int cy = static_cast<int>(ceil(y));
168 
169             // Parte decimal
170             float ty = y - fy;
171             float tx = x - fx;
172 
173             // Pesos interpolacion
174             float w1 = (1 - tx) * (1 - ty);
175             float w2 =      tx  * (1 - ty);
```

```
176                 float w3 = (1 - tx) *      ty;
177                 float w4 =      tx   *      ty;
178 
179                 // Iteramos sobre la imagen de entrada
180                 auto potenciaDos = pow(2, n);
181 
182                 for(int i=R; i < img.rows-R; i++)
183                 {
184                     int *pimgLBP = imgLBP.ptr<int>(i-R);
185                     const float *p  = imgMedia.ptr<float>(i-R);
186 
187                     for(int j=R; j < img.cols-R; j++)
188                     {
189                         float media = p[j];
190                         float t = w1*img.at<uchar>(i+fy,j+fx) + w2*img.at<uchar>(i
                                +fy,j+cx)
191                                   + w3*img.at<uchar>(i+cy,j+fx) + w4*img.at<uchar
                                        >(i+cy,j+cx);
192 
193                         if( t >= media )
194                             pimgLBP[j-R] += static_cast <int>(potenciaDos);
195                     }
196                 }
197             }
198         }
199     }
```

Anexo D

Código fuente: generador de LUTs

D.1. *main.cpp*

```
/**************************************************************************/
/**                        Trabajo Fin de Grado                          **/
/**                                                                      **/
/**               Detección de Gasas Quirúrgicas en imágenes             **/
/**                laparoscópicas empleando la técnica LBP               **/
/**                                                                      **/
/**        Grado en Ingeníeria Electrónica Industrial y Automática       **/
/**                        Universidad de Valladolid                     **/
/**                                                                      **/
/** Autor: Álvaro Muñoz García                                           **/
/** Tutor: Eusebio de la Fuente López                                    **/
/**************************************************************************/
/** Proyecto: Genera LUTs ULBP                                           **/
/** Archivo: main.cpp                                                    **/
/**                                                                      **/
/** Descripción: genera look-up tables que mapean los códigos LBP en ULBP. **/
/**     Graba los resultados en un archivo llamado ULBP_LUT_X.dat.       **/
/**     El número de vecinos puede cambiarse en la macrodefinción NUM_BITS **/
/**************************************************************************/

#include <iostream>
#include <vector>
#include <algorithm>
#include <fstream>
#include "opencv2/opencv.hpp"

#define NUM_BITS 8

using namespace std;
using namespace cv;

int desplazaDcha(int N, int numDesplaz, int numBits);
void actualizaLista(int viejo, int lista[], int nuevo, int numBits);

int main()
{
    int numDesplazam[NUM_BITS];
```

```
38      int tamLUT = pow(2, NUM_BITS);
39      int codigosLUT[tamLUT];
40
41      /// 1. Inicializamos la LUT a 0
42      for(int i = 0; i < tamLUT; ++i)
43          codigosLUT[i] = 0;
44
45      /// 2. Obtenemos todos los patrones uniformes a partir de desplazamientos
46      ///    a la derecha. Almacenmos temporalmente en los índices de los
47      ///    patrones uniformes el menor valor de su grupo.
48      codigosLUT[0] = 0; // Patron todo ceros: etiqueta ULBP 0
49      int Num = tamLUT - 1;
50
51      for(int c = 0; c < NUM_BITS; c++)
52      {
53          int N = Num >> c;
54
55          for(int d = 0; d < NUM_BITS; d++)
56              numDesplazam[d] = desplazaDcha(N,d,NUM_BITS);
57
58          int minimo = *min_element(numDesplazam, numDesplazam+NUM_BITS);
59          for(int i = 0; i < NUM_BITS; i++)
60              codigosLUT[numDesplazam[i]] = minimo;
61      }
62
63      /// 3. Cambiamos la etiqueta temporal (mínimo valor del grupo) por la
64      ///    etiqueta ULBP definitva.
65      int codeULBP = 2; // Las etiquetas 0 y 1 ya son correctas
66
67      for(int i = 2; i < tamLUT; ++i)
68      {
69          int codigoAct = codigosLUT[i];
70          if(codigoAct < codeULBP)
71              continue;
72
73          else
74          {
75              actualizaLista(codigoAct, codigosLUT, codeULBP, NUM_BITS);
76              codeULBP++;
77          }
78      }
79
80      /// 4. Las combinaciones no uniformes las ponemos el siguiente código
81      ///    ULBP sin utilizar, que será siempre P+1.
82      for(int i = 1; i < tamLUT; ++i)
83          if(!codigosLUT[i])
84              codigosLUT[i] = codeULBP;
85
86      /// 5. Escribimos la LUT en un fichero
87      string fileName = "../LUTS/ULBP_LUT_" + to_string(NUM_BITS) + ".dat";
88      ofstream fichSalida(fileName);
89
90      if(!fichSalida.good()) // No se ha podido abrir el fichero
91      {
92          cout << "No puedo grabar LUT!!";
93          exit(-1);
94      }
95
96      else
97          for(int i = 0; i < tamLUT; ++i)
98              fichSalida << codigosLUT[i] << " " ;
```

```
99
100     fichSalida << endl;
101     fichSalida.close();
102
103     cout << "LUT grabada con exito en fichero: " << fileName << endl;
104
105     return 0;
106 }
107
108 /**************************************************************************/
109 /** Función: desplazaDcha                                               **/
110 /**                                                                     **/
111 /** Descripción: se hace un desplazamiento de bits "numDesplaz" veces a la **/
112 /**     izquierda sobre el número "N". El desplazamiento es circular, y   **/
113 /**     los bits que salen por la derecha vuelven a entrar por la         **/
114 /**     izquierda. "numBits" corresponde al número de bits del código LBP, **/
115 /**     es decir, el número de vecinos.                                   **/
116 /**************************************************************************/
117 int desplazaDcha(int N, int numDesplaz, int numBits )
118 {
119     int desplazado = N;
120     int maskIzda = pow(2, numBits-1); // Mascara: 1000...0000
121
122     for(int b = 0; b < numDesplaz; b++)
123     {
124         if(desplazado&1)
125         {
126             desplazado = desplazado >> 1;
127             desplazado = desplazado|maskIzda; // Se mete un 1 por la izda
128         }
129
130         else
131             desplazado = desplazado >> 1;
132     }
133
134     return desplazado;
135 }
136
137 /**************************************************************************/
138 /** Función: actualizaLista                                              **/
139 /**                                                                     **/
140 /** Descripción: cambiamos las etiquetas temporales "viejo" por la nueva  **/
141 /**     etiqueta ULBP definitiva, "nuevo".                                **/
142 /**     Para ello recorre la LUT ("lista[]") en busca del valor viejo para **/
143 /**     cambiarlo por el nuevo.                                           **/
144 /**************************************************************************/
145 void actualizaLista(int viejo, int lista[], int nuevo, int numBits)
146 {
147     int tamLista = pow(2, numBits);
148
149     for(int i = 0; i < tamLista; i++)
150         if(lista[i] == viejo)
151             lista[i] = nuevo;
152 }
```

D.2. Ejemplo LUT: *ULBP_LUT_8.dat*

```
1   0 1 1 2 1 9 2 3 1 9 9 9 2 9 3 4 1 9 9 9 9 9 9 2 9 9 9 3 9 4 5 1 9 9 9 9 9 9
        9 9 9 9 9 9 9 9 9 2 9 9 9 9 9 9 9 3 9 9 9 4 9 5 6 1 9 9 9 9 9 9 9 9 9 9 9
        9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 2 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 3
        9 9 9 9 9 9 9 4 9 9 9 5 9 6 7 1 2 9 3 9 9 9 4 9 9 9 9 9 9 9 5 9 9 9 9 9 9
        9 9 9 9 9 9 9 9 9 6 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
        9 9 9 9 7 2 3 9 4 9 9 9 5 9 9 9 9 9 9 9 6 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 7
        3 4 9 5 9 9 9 6 9 9 9 9 9 9 9 7 4 5 9 6 9 9 9 7 5 6 9 7 6 7 7 8
```