



UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería Informática



TFG del Grado en Ingeniería Informática  
**Análisis paleontológico de piezas dentales 2.0**



Presentado por Andrés Miguel Terán  
en Universidad de Burgos — 3 de julio de 2017  
Tutor: Dr. José Francisco Díez Pastor  
Tutor: Dr. Raúl Marticorena Sánchez





UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería Informática



D. José Francisco Díez Pastor y D. Raúl Marticorena Sánchez, profesores del departamento de Ingeniería Civil, área Lenguajes y Sistemas Informáticos.

Exponen:

Que el alumno D. Andrés Miguel Terán, con DNI 71299912-G, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado “Análisis paleontológico de piezas dentales 2.0”.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección de los que suscriben, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 3 de julio de 2017

Vº. Bº. del Tutor:

D. José Francisco Díez Pastor

Vº. Bº. del Tutor:

D. Raúl Marticorena Sánchez



## **Agradecimientos**

Este proyecto pone punto y final a mi etapa de estudios en la universidad. En primer lugar, quiero agradecer a mis padres, Javier y María Jesús, el apoyo incondicional que me han brindado siempre, que ha sido de vital importancia para completar mi formación académica y mi desarrollo como persona.

De forma especial, quiero dar las gracias a mis hermanas, Paula y Esther, por respetar mis horas de estudio, entenderme y sacarme una sonrisa en todo momento mostrándome su cariño.

No puedo olvidarme de mis abuelos, tíos y primos cuyo apoyo y afecto nunca me han faltado. Gracias a mis amigos de siempre, que han sabido aceptar el tiempo que en ocasiones no he podido dedicarles, comprendiendo el esfuerzo y dedicación que estos estudios conllevan.

Quiero dar reconocimiento a las grandes amistades que he forjado en mi paso por la universidad. Han compartido conmigo alegrías, inquietudes, y un sinfín de momentos inolvidables dentro y fuera de las aulas, y han sido para mí una fuente muy importante de motivación.

Por último, también quiero dar gracias a la Universidad de Burgos y a todos los magníficos profesores con los que cuenta este grado, de los que he podido aprender mucho y me llevo una profesión que desarrollaré con pasión y orgullo durante toda mi vida.

Por todo ello, gracias.



## Resumen

La paleontología, junto con la antropología forense, permiten estudiar las poblaciones de la antigüedad a partir de sus restos fósiles. Uno de los elementos que más información nos dan son los dientes y en ellos las perikymata.

Las perikymata son el resultado observable de la formación incremental del esmalte a lo largo de la corona de un diente; visualmente se asemejan a una serie de líneas.

Tanto el número de perikymata, la distancia entre cada una y la zona de la corona en la que se encuentren son importantes para determinar el tipo de homínido al que pertenecen además de aportar información sobre la edad, la velocidad de crecimiento y posibles enfermedades que padeciese el individuo.

La versión anterior de este proyecto [3] automatizaba parcialmente el trabajo a desarrollar para el estudio de las perikymata realizando una unión de imágenes de los fragmentos de un diente tomadas mediante un microscopio electrónico, aplicando varios filtros a la imagen resultante, automarcando las perikymata y finalmente extrayendo los datos.

Este proyecto busca revisar, mejorar y corregir errores de cada una de las fases mediante modificaciones parciales o totales, siendo la del filtrado la más importante para la posterior detección de líneas, marcado de perikymata y extracción de datos.

## Descriptores

Perikymata, paleontología, unión de imágenes, filtrado de imágenes, detección de líneas.

## Abstract

Paleontology, together with forensic anthropology, allow us the study of ancient populations from their fossil remains. One of the elements that more information gives us are the teeth and within them the perikymata.

Perikymata are the observable result of the incremental enamel formation along the crown of a tooth; they resemble a lines sequence.

Number of perikymata, the distance between each other and the area of the crown in which they are found are important to determine the hominid specie to whom the tooth belongs. Furthermore they provide information about the age, the speed of development and possible diseases the individual may had had.

The previous version of this project [3] partially automated the work for the study of perikymata by stitching electron microscope tooth images, applying several filters to the final image, uto-marking perikymata and finally extracting the information.

This project seeks to review, improve and correct errors of each stage performing partial or total modifications considering the filtering stage the most important one for the subsequent marking of perikymata and data extraction.

## Keywords

Perikymata, paleontology, image stitching, image filtering, lines detection.

---

# Índice general

---

<b>Índice general</b>	<b>III</b>
<b>Índice de figuras</b>	<b>v</b>
<b>Introducción</b>	<b>1</b>
<b>Objetivos del proyecto</b>	<b>3</b>
2.1. Objetivos generales . . . . .	3
2.2. Objetivos específicos . . . . .	4
<b>Conceptos teóricos</b>	<b>7</b>
3.1. Ecualización adaptativa del histograma . . . . .	7
3.2. Reducción de ruido - Algoritmo Chambolle . . . . .	8
3.3. Aplicación de filtros . . . . .	9
3.4. Filtros de detección de bordes . . . . .	10
3.5. Marcado de bordes . . . . .	12
3.6. Detección de líneas . . . . .	14
3.7. Espacios de color . . . . .	15
<b>Técnicas y herramientas</b>	<b>17</b>
4.1. Metodología SCRUM . . . . .	17
4.2. Gestión del proyecto y control de versiones . . . . .	18
4.3. Java y JavaFX . . . . .	20
4.4. Python y Scikit-Image . . . . .	21
4.5. Jupyter Notebooks . . . . .	22
4.6. OpenCV . . . . .	23
4.7. IDEs . . . . .	23
4.8. Otras herramientas . . . . .	24
<b>Aspectos relevantes del desarrollo del proyecto</b>	<b>27</b>

5.1. Requisitos generales . . . . .	27
5.2. Stitching . . . . .	28
5.3. Imágenes de dientes . . . . .	31
5.4. Primeros filtrados . . . . .	33
5.5. Servidor en Python y PyInstaller . . . . .	35
5.6. Procesado de la imagen final . . . . .	36
<b>Trabajos relacionados</b>	<b>41</b>
6.1. Análisis paleontológico de piezas dentales - Sergio Chico Carrancio	41
6.2. Estimación de la dieta por análisis de marcas dentales - Ismael Tobar García . . . . .	41
6.3. Artículo sobre los Kernels de Kirsch . . . . .	42
<b>Conclusiones y Líneas de trabajo futuras</b>	<b>43</b>
7.1. Conclusiones . . . . .	43
7.2. Líneas de trabajo futuras . . . . .	44
<b>Bibliografía</b>	<b>47</b>

---

# Índice de figuras

---

1.1.	Fragmento de un diente . . . . .	1
3.2.	Ecuación adaptativa . . . . .	8
3.3.	Reducción de ruido <i>Chambolle</i> . . . . .	8
3.4.	Kernels Prewitt . . . . .	9
3.5.	Kernels Sobel . . . . .	9
3.6.	Proceso de convolución . . . . .	10
3.7.	Kernels Kirsch: Norte, Noroeste y Oeste . . . . .	11
3.8.	Efectos del filtro Kirsch . . . . .	11
3.9.	Filtro Frangi . . . . .	12
3.10.	Efectos de la binarización . . . . .	13
3.11.	Efectos de la esqueletización . . . . .	13
3.12.	Imagen con líneas detectadas . . . . .	14
3.13.	Color rojo en RGBA . . . . .	15
3.14.	Matices HSV . . . . .	16
4.15.	Logo de GitHub . . . . .	18
4.16.	Tablero ZenHub . . . . .	19
4.17.	Burndown Chart . . . . .	19
4.18.	Logo de Java . . . . .	20
4.19.	Logo de <i>Scikit-Image</i> . . . . .	22
5.20.	Opción de carpeta temporal . . . . .	30
5.21.	Puntos comunes de <i>stitching</i> con Python . . . . .	30
5.22.	Resultado de <i>stitching</i> con Python . . . . .	31
5.23.	Imagen del diente 1 . . . . .	32
5.24.	Imagen del diente 2 . . . . .	32
5.25.	Procesado Frangi con la imagen y la máscara . . . . .	33
5.26.	Procesado Frangi sobre el diente 2 . . . . .	34
5.27.	Detección de líneas sobre la imagen procesada . . . . .	34
5.28.	Pieza dental . . . . .	36
5.29.	Imagen del diente ecualizada y sin ruido . . . . .	37
5.30.	Imagen del diente con un filtro Kirsch Este . . . . .	37

5.31. Imagen del diente binarizada . . . . .	38
5.32. Imagen esqueletonizada con detección de líneas . . . . .	38
5.33. Imagen con perikymata detectadas . . . . .	39
5.34. Imagen con perikymata detectadas en la versión anterior . . . . .	39

---

# Introducción

---

La antropología forense se centra en el estudio de las poblaciones humanas a nivel sociocultural y biológico. Esta ciencia unida a la paleontología nos permiten estudiar la evolución humana a partir de los restos fósiles encontrados, siendo de gran importancia a nivel mundial los encontrados en la sierra de Atapuerca en Burgos a partir a finales del siglo XX [2].

Dentro de estos restos, unos de los que más información aportan son los dientes, ya que nos permiten conocer detalles relativos a la edad de un homínido, a su alimentación, su velocidad de crecimiento y a enfermedades que padeció.

Nuestro caso de estudio serán las perikymata. Son una serie de líneas que aparecen en el esmalte de la corona del diente durante de su formación. Tienen el aspecto de la figura 1.1.

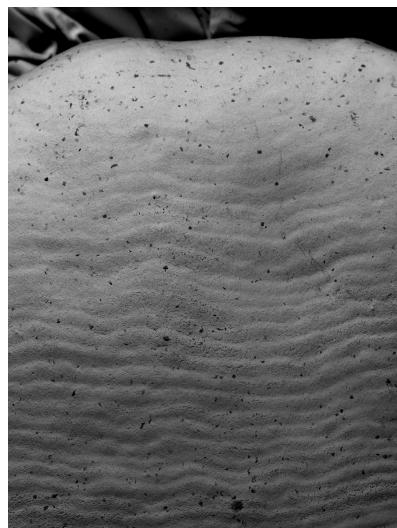


Figura 1.1: Fragmento de un diente

Este proyecto colabora con el Laboratorio de Evolución Humana de la Universidad de Burgos cuyo personal investigador extrae información de las perikymata en un proceso manual: en primer lugar se toman varias imágenes de distintas zonas de un diente con un microscopio electrónico, luego se unen mediante una herramienta software como Gimp [6] y posteriormente se divide la corona del diente en 10 partes (deciles) en los que se marcan las perikymata.

El proyecto anterior [3] automatizaba en parte esa tarea con una aplicación de escritorio que se desarrollaba en las siguientes fases:

- Unión de imágenes (*stitching*). Gracias al zoom de un microscopio electrónico se tomaban imágenes de distintas áreas de la pieza dental y estas se unían para conseguir una imagen completa sobre la que trabajar.
- Filtrado de la imagen completa para resaltar las perikymata. Se aplicaba un filtro de desenfoque gaussiano [23] para eliminar el ruido de la imagen y un filtro Prewit [29] para resaltar las perikymata.
- División de la corona en deciles y automarcado de las perikymata. El usuario indicaba donde empezaba y acababa la corona del diente para dividirla en deciles, además de indicar la escala de tamaño de la pieza y dibujar una línea que atravesase las perikymata. Posteriormente, se detectaban y se permitía al usuario añadirlas o quitarlas donde fuera necesario.
- Extracción de datos a un fichero. Se exportaban a un fichero *csv* el número total de perikymata, sus coordenadas en la imagen, el decil al que pertenecen y la distancia a la perikymata anterior.

En este proyecto se partirán de los conceptos y técnicas desarrolladas en la versión anterior [3] con el fin último de mejorar la detección automática de las perikymata, dando especial importancia al procedimiento y filtros aplicados para resaltarlas. Además, se abordarán las complicaciones y deficiencias de las fases anteriores, así como los errores que los usuarios han reportado.

Se incluirán también las pruebas realizadas, las herramientas utilizadas, la hoja de ruta a seguir, los problemas planteados durante el desarrollo de este proyecto y las soluciones desarrolladas para resolverlos.

---

# **Objetivos del proyecto**

---

## **2.1. Objetivos generales**

De la versión anterior del proyecto [3] heredamos una serie de requisitos generales acerca de la aplicación a desarrollar:

- Se deben poder por unir los fragmentos para conseguir una imagen completa del diente.
- El usuario debe poder aplicar un filtro o filtros a fin de resaltar las perikymata.
- La corona del diente se dividirá en diez partes (deciles) sobre la que el usuario dibujará una línea para poder automarcar las perikymata detectadas.
- Los datos conseguidos sobre el número de perikymata y su localización podrán ser exportados a un fichero.

Teniendo en cuenta esos objetivos, se profundizará en mejorar la detección y automarcado de las perikymata gracias a la aplicación de otros filtros de imagen y procedimientos más adecuados para resaltarlas.

## 2.2. Objetivos específicos

### Unión de imágenes (*stitching*)

Como mencionábamos en la introducción, la primera fase será la unión de imágenes (*stitching* de aquí en adelante), que nos permitirá obtener una imagen completa de la pieza dental a partir de las imágenes de fragmentos tomadas por un microscopio electrónico. Los objetivos en esta fase serán:

- Investigar, valorar y probar la utilización de otro lenguaje de programación para realizar el *stitching*.
- Eliminar la dependencia total de archivos de bibliotecas de enlace dinámicos (*dll*) que en la versión anterior necesitaban estar en la misma ruta que la aplicación Java para funcionar.
- Conseguir que la aplicación de *stitching* sea multiplataforma de modo que no esté restringida a sistemas Windows de 64 bits.
- Solucionar el error a la hora de hacer *stitching* con imágenes en ciertas rutas.

### Filtrado de imagen y detección de perikymata

El filtrado de la imagen completa del diente es la parte más importante del proyecto puesto que un buen filtrado permitirá resaltar mejor las perikymata. Los objetivos de esta fase son:

- Investigar otros filtros de imagen relacionados con la detección de bordes.
- Realizar y documentar pruebas de los diferentes filtros, los distintos métodos empleados y como afectarán a la detección de líneas.
- Elaborar un procedimiento que, aplicado junto con los de nuevos filtros de imagen, permita marcar mejor las perikymata que con el filtro Prewitt [29].
- Emplear una técnica de detección de líneas adecuada en base al proceso de filtrado que se utilice finalmente.

### Aplicación

A nivel de la aplicación marcaremos los siguientes objetivos:

- Adaptar la aplicación Java para seleccionar el subprograma de *stitching* adecuado en base al sistema operativo y la arquitectura en la que nos encontramos.

- Incluir la opción de seleccionar una carpeta para los archivos temporales del *stitching*.
- Integrar adecuadamente en la aplicación el nuevo proceso de filtrado que pueda surgir.
- Incluir un modo sencillo de filtrado por defecto para facilitar al usuario la interacción con la aplicación y también, un modo avanzado con el que el usuario pueda modificar los posibles parámetros del filtrado.

### Corrección de errores

- Revisar elementos de la aplicación como el sistema de log y mejorar la eficacia en la fase de automarcado de las perikymata.
- Localizar y solucionar el error que se produce en la carga de imágenes al reabrir un proyecto.
- Encontrar y solucionar los errores reportados por el cliente.
- Investigar otros fallos y, en caso de no conseguir solucionarlos, documentarlos como líneas de trabajo futuras.



---

# Conceptos teóricos

---

El procesamiento de imágenes toma un papel muy importante en este proyecto. El objetivo es la detección de perikymata y para ello en esta sección se abordarán conceptos teóricos importantes sobre el preprocesado que se ha llevado a cabo y el marcado y detección de bordes.

El preprocesado está formado por la *ecualización adaptativa del histograma*, la *reducción de ruido* y la *aplicación de filtros* de detección de bordes. El marcado de bordes comprende la *binarización* de la imagen y la *esqueletización* y para finalizar la detección de bordes se lleva a cabo mediante la aplicación de la *transformada de Hough* y el uso del espacio de color *HSV*.

## 3.1. Ecualización adaptativa del histograma

La operación de ecualización adaptativa del histograma<sup>1</sup> se realiza sobre una imagen para conseguir un mayor contraste. Una modificación de esta operación es la que lleva a cabo el algoritmo *CLAHE* [21] (*Contrast Limited Adaptive Histogram Equalization*) el cual toma cada píxel de la imagen y su vecindario sobre los que extrae el histograma y realiza la ecualización, de este modo se consigue una ecualización local en vez de una global y con ello un marcado de bordes más adecuado para el tema que nos ocupa como vemos en la figura 3.2.

---

<sup>1</sup>Se conoce también como *AHE* (*Adaptative Histogram Equalization*).



Figura 3.2: Ecualización adaptativa

### 3.2. Reducción de ruido - Algoritmo Chambolle

El ruido de una imagen [27] consiste en variaciones en el color y brillo de los píxeles, de forma agresiva, al tomar imágenes mediante dispositivos de fotografía digital, como en nuestro caso, el microscopio electrónico. Reducir este ruido permite mejorar la visualización general de las imágenes y los objetos que contiene a cambio de disminuir el enfoque en la misma.

El algoritmo *Chambolle* [4] es una técnica que, mediante la minimización del total de variación de una imagen [34], consigue reducir el ruido sin que ello suponga un suavizado excesivo de los bordes en los objetos de la imagen, lo cual es interesante para este proyecto como vemos en la figura 3.3.

Figura 3.3: Reducción de ruido *Chambolle*

Parte de la premisa de que a una imagen se le ha aplicado un filtro y además se le ha añadido ruido gaussiano [23] y trata de resolver un problema de ingeniería inversa para recuperar la imagen original [11].

### 3.3. Aplicación de filtros

Para poder aplicar un filtro a una imagen tendremos en cuenta los siguientes elementos:

#### Kernel

El kernel, también llamado máscara o matriz de convolución, consiste en una matriz cuadrada de dimensión normalmente impar<sup>2</sup> que, mediante la operación de convolución, aplicamos a una imagen para obtener la imagen filtrada. Existen filtros muy variados [18], algunos centrados en desenfocar la imagen, añadirle contraste, eliminar ruido o realzar bordes para su detección como en nuestro caso.

Dependiendo del filtro que se quiera aplicar encontraremos unos kernels u otros como podemos ver en las figuras 3.4 y 3.5. Un conjunto de kernels se suelen agrupar bajo el nombre de operador.

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Figura 3.4: Kernels Prewitt [29]

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Figura 3.5: Kernels Sobel [32]

Sus dimensiones son impares para poder identificar el elemento central de la matriz que se utilizará en la operación de convolución.

#### Convolución

La operación de convolución nos permitirá conseguir una imagen filtrada, para ello necesitaremos un kernel y una imagen, considerándola como una matriz de dos dimensiones.

---

<sup>2</sup>Encontramos excepciones como el operador de Robert con kernels de 2x2 [19].

Citando el proyecto anterior [3], la operación consiste en pasar por todos los puntos de la imagen, a los que se superpone el kernel, de modo que el elemento central coincide con el punto de la imagen en el que nos encontramos. Seguidamente se multiplican los valores del kernel por los valores de los puntos de la imagen sobre los que está superpuestos y se suman todos. El resultado será el nuevo valor que toma el píxel en el que nos encontramos como podemos ver en la figura 3.6.

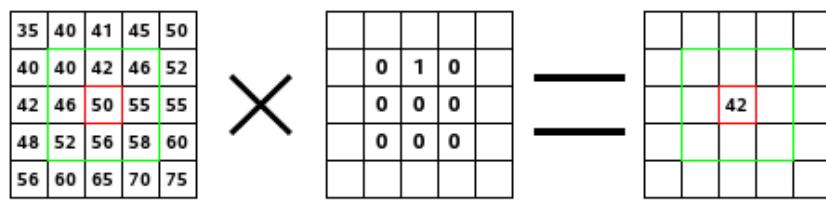


Figura 3.6: Proceso de convolución [3]

### 3.4. Filtros de detección de bordes

Antes de comenzar con la explicación de los filtros, conviene repasar unos conceptos de interés sobre lo que veremos más adelante.

Los filtros de detección de bordes nos permiten conseguir que los valores máximos y mínimos de una imagen queden bien diferenciados. Estos filtros están basados en operadores derivativos ya que, al hacer la derivada a lo largo de la imagen, consiguen que la variación luminosa sea máxima y por tanto se marcan más los bordes [12]. En este proyecto encontraremos los siguientes tipos de filtros:

- Filtros en la primera derivada:

Los operadores que calculan la primera derivada se llaman *gradientes*. Algunos de estos operadores son: Sobel [32], Prewitt [29] y Robert [19]. Generalmente se computan en dos ejes,  $x$  e  $y$  para después sumarse y conseguir el gradiente completo junto con la imagen con bordes resaltados.

- Filtros en la segunda derivada:

En la segunda derivada encontramos filtros como los laplacianos y otros como los hessianos. Los filtros laplacianos calculan la suma de la segunda derivada y dan como resultado un escalar<sup>3</sup>, mientras que los filtros hessianos calculan una matriz con todas las posibles combinaciones de las segundas derivadas, la matriz hessiana [25].

---

<sup>3</sup>Número perteneciente a los reales.

A diferencia de los operadores gradiente estos son todavía más sensibles al ruido de la imagen.

A continuación se mencionan los filtros de detección de bordes más relevantes con los que se ha trabajado y sus características:

### Kirsch

El filtro de Kirsch es un operador de gradiente, por lo que al aplicarlo acentúa los bordes. Según se explica en [17], como característica fundamental tendremos distintos *kernels* en función de la orientación de los bordes que queremos resaltar, algunos como los de la figura 3.7 para marcar bordes horizontales, verticales o ligeramente inclinados.

$$\begin{bmatrix} +5 & +5 & +5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix} \quad \begin{bmatrix} +5 & +5 & -3 \\ +5 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix} \quad \begin{bmatrix} +5 & -3 & -3 \\ +5 & 0 & -3 \\ +5 & -3 & -3 \end{bmatrix}$$

Figura 3.7: Kernels Kirsch: Norte, Noroeste y Oeste [17]

Como se puede observar en la figura 3.8, el efecto conseguido guarda una relación contraria con el nombre del kernel, pues el filtro norte resalta los bordes horizontales y el filtro oeste resalta los verticales.

El nombre de la orientación se refiere a los elementos del kernel que multiplicarán positivamente en la operación de convolución (sección 3.3), en este caso los elementos con valor  $+5$ . Así, el kernel oeste se llama así por encontrar todos los  $+5$  en dirección oeste.

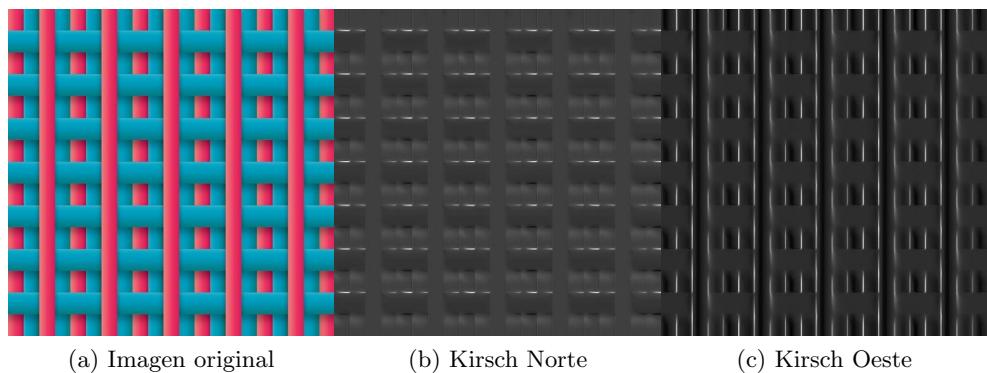


Figura 3.8: Efectos del filtro Kirsch

## Frangi

Frangi [13] es una modificación del filtro hessiano (sección 3.4). Se utiliza en la detección de bordes continuos como por ejemplo ríos, arrugas o venas. Está basado en el cálculo de los máximos eigenvectores<sup>4</sup> sobre la matriz hessiana. Aplicado a la imagen de Lenna [28] quedaría como en la figura 3.9.



Figura 3.9: Filtro Frangi

## 3.5. Marcado de bordes

En esta sección se comentarán los procesos que se llevan a cabo después de la aplicación de los filtros para realzar las perikymata.

### Binarización

El proceso de binarización (figura 3.10) consiste en conseguir que los valores de una imagen tomen solo dos posibles valores, generalmente un *true* o *false* lógicos, lo cual se traduce visualmente en una imagen en blanco y negro.

El procedimiento se basa en pasar por todos los puntos de una imagen, comparando su valor con un cierto valor *umbral* y sustituir el valor del punto por un *true* o *false* (o también 0 o 1) si es mayor o menor que el *umbral*.

---

<sup>4</sup>Valores que no cambian cuando se aplica una modificación [22].

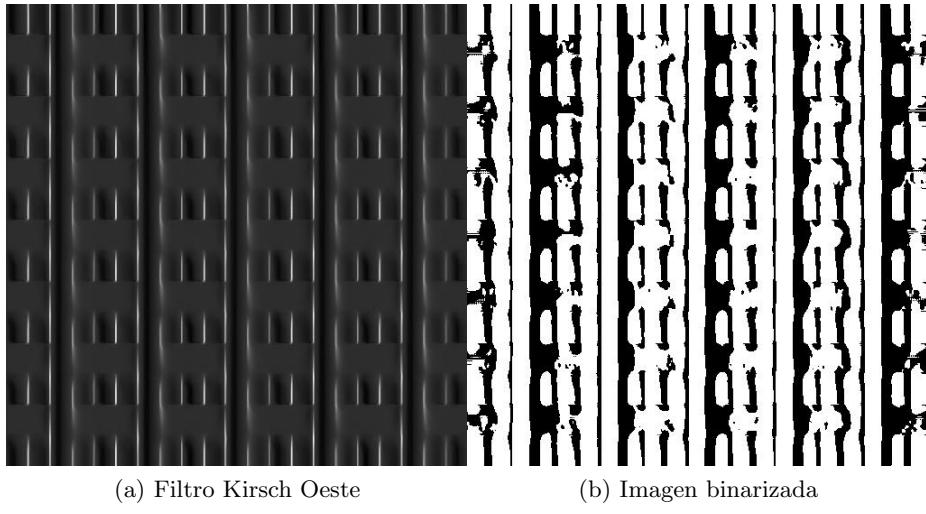


Figura 3.10: Efectos de la binarización

### Esqueletonización

La esqueletonización es el procedimiento que aplicamos a una imagen binarizada para que su valor lógico *true* se reduzca al tamaño de un píxel, normalmente dando lugar a líneas a lo largo de la imagen. Esto es útil, ya que si en la imagen los bordes están realzados, permitirá que queden muy marcados (como en la figura 3.11) y sea fácil su posterior identificación.

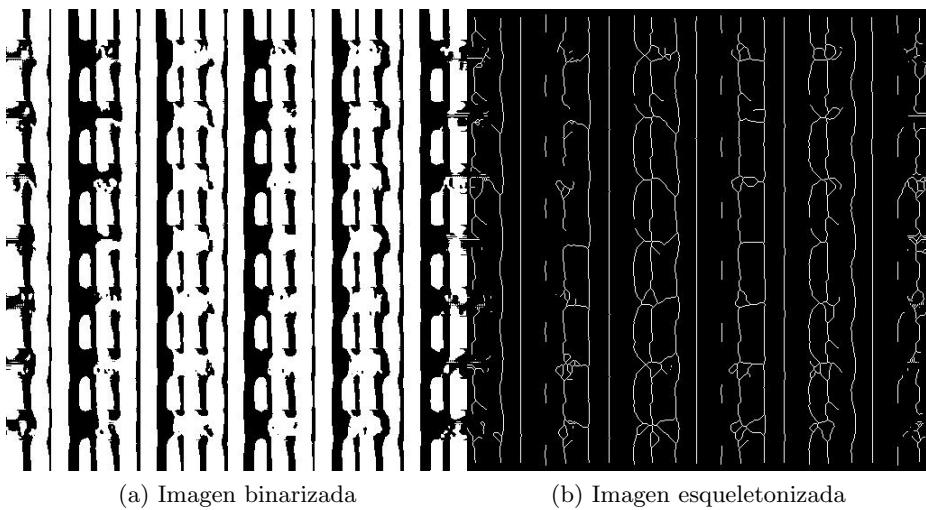


Figura 3.11: Efectos de la esqueletonización

### 3.6. Detección de líneas

La detección de líneas se lleva a cabo mediante la Transformada de Hough.

La *Transformada de Hough* es un mecanismo desarrollado para visión computacional, utilizado normalmente para la extracción de líneas e identificación de círculos y elipses en imágenes. Requiere que las imágenes estén binarizadas y con la menor cantidad de ruido posible.

Cuenta con múltiples variaciones, la función utilizada en el proyecto [14] implementa la *Transformada Probabilística de Hough Progresiva* desarrollada en este artículo [5] y, como característica diferenciadora de la *Transformada Probabilística de Hough* tradicional, reduce el coste computacional y opera sobre los ángulos especificados como vemos en la figura 3.12.

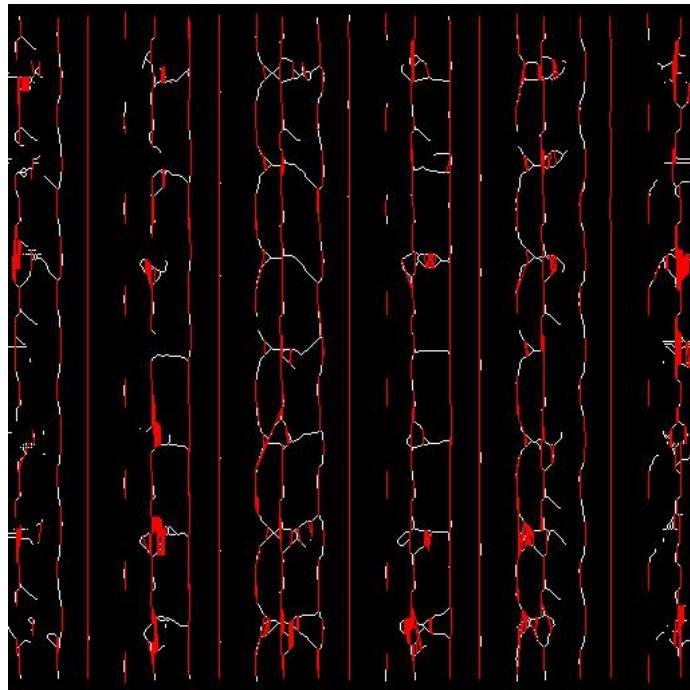


Figura 3.12: Líneas detectadas sobre la figura 3.10b

Esta operación, en vez de desarrollarse sobre toda la imagen, se centra en áreas concretas que tienen mayor probabilidad de contener una líneas.

### 3.7. Espacios de color

Una imagen es considerada una matriz de dos dimensiones, donde cada píxel guarda cierta información que podemos interpretar en distintos modelos de color. En este proyecto se manejan dos modelos de colores.

#### Modelo RGB - RGBA

El modelo *RGB* [31] interpreta el color gracias a la combinación de los tres colores primarios: rojo (R), verde (G) y azul (B).

Los colores primarios toman valores enteros entre 0 y 255, siendo este último el valor de máxima intensidad. Este espacio de color es el utilizado en toma de imágenes digitales.

En determinados tipos de imágenes, como las imágenes *png*, se añade además el canal *alpha* (A), que indica la transparencia del color. Podemos encontrar variaciones en los rangos del canal *alpha* dependiendo de la tecnología usada, por ejemplo, en hojas de estilo CSS el rango del canal es un número decimal entre 0 y 1 y en la lectura de imágenes con Python es un número entero entre 0 y 255.

Al leer los datos de un píxel en Python, siguiendo este modelo, encontraremos un vector como el de la figura 3.13.

$$\begin{bmatrix} 255 & 0 & 0 & 255 \end{bmatrix}$$

Figura 3.13: Color rojo en RGBA

#### Modelo HSV

El modelo *HSV*<sup>5</sup> [26] representa los colores dividiéndolos en:

- *Hue*: consiste en el matiz del color. Se mide en grados sexagesimales.
- *Saturation*: se corresponde con la saturación del color e indica su intensidad. Generalmente se mide con porcentajes.
- *Value*: indica el brillo del color. Al igual que con la saturación se mide en porcentajes.

---

<sup>5</sup>También conocido como *HSB* (*Brightness*), guarda similitud con otros espacios de color como *HSI* o *HSL* aunque no llega a ser el mismo debido a la forma de conversión desde *RGB* [26].

Este modelo es utilizado en selectores de colores debido a la sencillez de compresión para el ser humano. Podemos identificar los colores según el ángulo como en la figura 3.14.

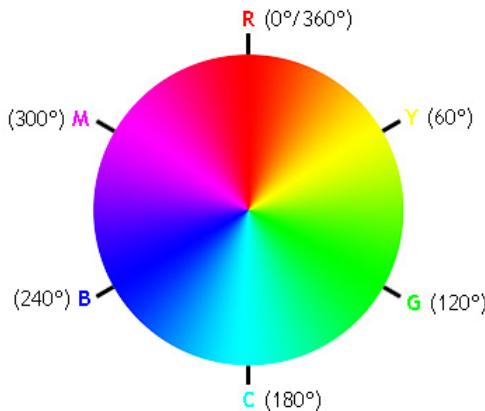


Figura 3.14: Rueda de matices HSV [1]

La utilidad para este proyecto radica en que se definirá un rango dentro del matiz para considerar qué es rojo y qué no y se identificarán como perikymata las líneas con colores en ese rango.

Consideraremos rojo todo color con un matiz entre  $350^\circ$  y  $10^\circ$  y con una saturación y brillo superiores al 80 %.

---

## Técnicas y herramientas

---

### 4.1. Metodología SCRUM

SCRUM ha sido la metodología usada a lo largo del desarrollo del proyecto. Es una metodología ágil que se caracteriza por un desarrollo software en iteraciones (*sprints*).

Al inicio de cada *sprint* se realiza una reunión donde se definen los objetivos y requisitos a cumplir durante la iteración, esto forma lo que llamamos el *sprint backlog*.

La duración de cada *sprint* para este proyecto ha sido, normalmente, de una semana, aunque a veces se ha visto aumentada debido a coincidencias con periodos vacacionales largos o cuando se estimaba más tiempo del habitual para los requisitos del *sprint backlog* en la reuniones.

SCRUM está orientado a la entrega de software funcional de forma rápida, es decir, cada una o pocas iteraciones tendremos una versión del producto, lo que permite obtener una mayor calidad en el software.

Al contrario que las metodologías tradicionales, con SCRUM se reacciona de manera rápida ante los cambios producidos y se reflexiona sobre el rumbo que debe tomar el proyecto durante su desarrollo.

La mayor ventaja que ha aportado a este proyecto es, como mencionábamos, la capacidad de reaccionar antes los cambios, ya que se han necesitado realizar muchas pruebas hasta llegar a la versión final, sobre todo en lo referente a qué filtros aplicar para la detección de perikymata.

## 4.2. Gestión del proyecto y control de versiones

A fin de poder desarrollar el proyecto siguiendo la metodología SCRUM se ha utilizado ZenHub como sistema de gestión de proyectos y *git* como sistema de control de versiones utilizando un repositorio en GitHub.

### Git y GitHub

*Git* [24] es un software de control de versiones de código abierto y gratuito que, entre otras cosas, permite la creación de ramas en las que trabajar sobre distintos aspectos, y posibilita su posterior integración en una única rama o revertir los cambios realizados. Podemos descargar *git* desde: <https://git-scm.com/downloads>.

GitHub (figura 4.15) es una plataforma online basada en *git*, que permite la creación de repositorios públicos<sup>6</sup> y gratuitos, siendo importante de cara al trabajo en equipo y a la realización de mejoras gracias a los *issues* que cualquiera puede sugerir. En el siguiente enlace encontramos información sobre los distintos elementos y operaciones en GitHub: <https://help.github.com/articles/github-glossary/>



Figura 4.15: Logo de GitHub

También es interesante GitHub Desktop, la herramienta *git* proporcionada por GitHub. Dispone de una interfaz sencilla para interactuar con los repositorios en GitHub. Podemos descargarla desde: <https://desktop.github.com/>

### ZenHub

ZenHub es una plataforma de gestión de proyectos que se integra en GitHub, instalándose en el navegador mediante una extensión. Podemos descárgalo desde <https://www.zenhub.com/>.

Nos proporciona un tablero, como el de la figura 4.16, en el que podemos observar, organizar e interaccionar fácilmente con los *issues* que vamos creando, los *milestones* (*sprints*) y los colaboradores del proyecto.

---

<sup>6</sup>La plataforma dispone de planes de pago para obtener repositorios privados.

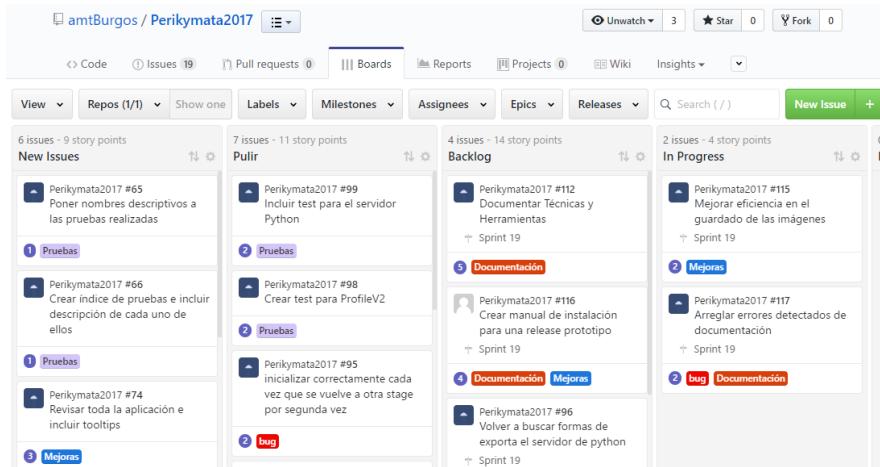


Figura 4.16: Tablero ZenHub

A cada *issue* podemos asignarle una etiqueta que defina su finalidad. Las utilizadas en este proyecto han sido: *bug*, *documentación*, *pruebas*, *mejoras* y *nuevas funcionalidades*.

También permite la visualización de los *burndown charts*, que contienen la progresión en la resolución de los *issues* a lo largo de cada *sprint*, como en la figura 4.17.

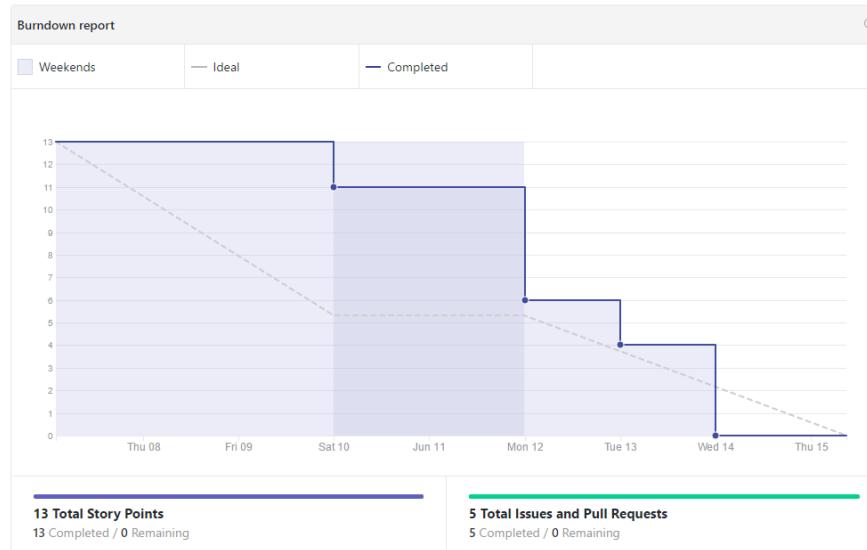


Figura 4.17: Burndown Chart

### 4.3. Java y JavaFX

Al igual que la versión anterior del proyecto [3], se han utilizado Java y JavaFX para el desarrollo de la aplicación con la que interacciona el usuario. Se valoró también la idea de rehacer toda la aplicación en Python, dado la utilización de este para los filtros, pero finalmente se rechazó por la cantidad de tiempo que supondría y teniendo en cuenta que la mejora no sería demasiado notable.

#### Java

Java [16] es un lenguaje orientado a objetos, de alto nivel y estáticamente tipado. El programador se encarga de escribir las clases que son compiladas y ejecutadas en la Máquina Virtual Java (JVM). Estas clases pueden ser transportadas y reutilizadas fácilmente en otros entornos donde tengamos una JVM. En la figura 4.18 podemos ver el logo de Java.

La versión utilizada ha sido Java 8 y podemos conseguirla aquí:  
<http://www.oracle.com/technetwork/java/javase/downloads/>

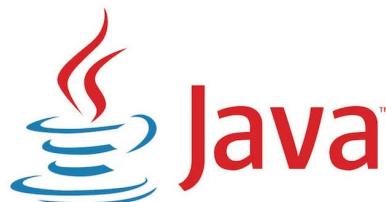


Figura 4.18: Logo de Java

#### JavaFX

JavaFX [10] es una herramienta incluida en las instalaciones de Java, tanto en el kit de desarrollador JDK, como en el entorno de ejecución JRE, y provee un conjunto de paquetes gráficos para desarrollar aplicaciones de cliente enriquecidas, que pueden usarse incluso en páginas web<sup>7</sup>. Desde JavaFX podemos acceder a todas las características y librerías de Java.

La interfaz de usuario puede ser escrita completamente mediante una clase Java, pero JavaFX también permite crear vistas en código FXML<sup>8</sup> que después son cargadas por una clase controlador en la aplicación, y que nos posibilita definir las acciones que se deben llevar a cabo cuando el usuario interactúa con los distintos elementos.

---

<sup>7</sup>Para ello es necesario un plugin. Más información aquí: <http://www.oracle.com/technetwork/java/index-jsp-141438.html>.

<sup>8</sup>Código tipo XML.

Se ha utilizado también JavaFX Scene Builder 2.0, una herramienta que permite la creación de vistas de manera visual e interactiva además de proporcionarnos el cuerpo de la clase controlador a la que esté asociada.

La descarga de Scene Builder podemos encontrarla aquí:

<https://goo.gl/gZMvXe>

Como podemos intuir, en este proyecto se ha continuado utilizando el patrón Modelo-Vista-Controlador (MVC) implementado en la versión anterior [3]. En el modelo encontramos clases Java, en la vista tenemos archivos FXML y los controladores serán clases que accederán a los elementos de las vistas y que desarrollarán la lógica de la aplicación.

También se ha utilizado JavaFX para la lectura de las imágenes en vez de ImageJ [7], que usaba la versión anterior del proyecto, con lo que eliminamos una dependencia innecesaria.

## 4.4. Python y Scikit-Image

### Python

Python [30] es un lenguaje interpretado, multiplataforma y multiparadigma de manera que puede usarse con una orientación a objetos o de forma funcional.

Es un lenguaje dinámicamente tipado, por lo que permite que se asigne una variable con un valor de tipo distinto que con el que se había inicializado.

Algunos de los objetivos más importantes que persigue este lenguaje son la legibilidad y transparencia en la escritura de código.

Otro de los aspectos importantes de Python, es la cantidad de módulos y paquetes de los que disponemos. Encontramos por ejemplo, el caso de Anaconda, que instala Python e incluye una gran variedad de módulos para interactuar con datos científicos.

Encontramos dos versiones del mismo, la versión 2 y la 3. En este proyecto se ha elegido desarrollar con la versión 3 debido a que, en un futuro no muy lejano, será la versión que predominará<sup>9</sup> y también porque actualmente viene incluido en las distribuciones Linux, lo cual evita que el usuario deba realizar más pasos para instalar nuestra aplicación en esos sistemas operativos.

Actualmente 344 de los 360 paquetes más usados en Python son soportados por la versión 3<sup>10</sup> y algunos de ellos como los Jupyter Notebooks ya no

---

<sup>9</sup>En 2020 acabará el mantenimiento oficial: <https://pythonclock.org/>.

<sup>10</sup>Podemos ver el contador de los paquetes más populares disponibles en Python3 aquí: <http://py3readiness.org/>.

sopportan Python2 en sus últimas versiones<sup>11</sup>.

Podemos descargarlo aquí:

<https://www.continuum.io/downloads>.

### Scikit-Image

*Scikit-Image* (figura 4.19) es módulo que contiene algoritmos de procesamiento de imágenes y visión computacional, y complementa a *SciPy* [15], otro módulo utilizado para computación científica en Python.



Figura 4.19: Logo de *Scikit-Image*

Es código abierto y aquí podemos encontrar su repositorio en GitHub:  
<https://github.com/scikit-image/scikit-image>.

Se ha utilizado la última versión estable disponible en el momento del desarrollo de este proyecto, la versión 0.13.0.

Esta ha sido la alternativa a ImageJ [7] para Java utilizada para procesar las imágenes y aplicar filtros. Se ha elegido junto a Python por su sencillez a la hora de desarrollar y aplicar la técnica que nos ha permitido realizar las perikymata.

## 4.5. Jupyter Notebooks

Jupyter Notebooks [9] es una aplicación cliente-servidor, que puede ser ejecutada en el equipo local y sirve para editar *notebooks*: documentos que incluyen código Python ejecutable y también elementos de texto enriquecido.

Ha sido de gran importancia para realizar pruebas con los distintos filtros propuestos pues permite ver los resultados de manera rápida y sencilla.

Esta aplicación viene incluida en la instalación de Anaconda.

---

<sup>11</sup>Desde IPython no sopportan Python2 a partir de la versión 6: <http://ipython.readthedocs.io/>.

## 4.6. OpenCV

OpenCV [8] es una librería utilizada para visión computacional, es multiplataforma<sup>12</sup>, de código abierto y oficialmente da soporte a lenguajes como C++, Java y Python, aunque pueden encontrarse versiones para Ruby e incluso .NET<sup>13</sup>.

En la versión anterior [3] del proyecto, era utilizada para el desarrollo de la aplicación de *stitching* en C++, que se encargaba de unir las imágenes. En este proyecto también se ha probado a utilizarla con Python, aunque el resultado no fue satisfactorio y se desechó su utilización.

La versión de OpenCV utilizada en este proyecto ha sido la 2.4.11 en vez de continuar con la 3.1 por las razones que comentaremos más adelante en la sección 5.2.

## 4.7. IDEs

Un IDE es un entorno de desarrollo que incluye elementos como un editor de texto, un depurador o hasta un compilador e intérprete.

A lo largo del proyecto se han utilizado varios IDEs dependiendo del lenguaje con el que trabajar.

### Eclipse

Eclipse es uno de los entornos de desarrollo para Java más popular, y mediante distintos plugins puede ser utilizado para desarrollar en otros lenguajes. Se distribuye bajo una licencia de código abierto (Eclipse Public License). He elegido Eclipse porque es el IDE con el que he trabajado en Java a lo largo del grado. La versión utilizada ha sido *Eclipse Neon 4.6*.

Podemos encontrar esta y muchas otras versiones en el siguiente enlace:  
<https://www.eclipse.org/downloads/>

### PyCharm

PyCharm es un IDE para desarrollar código Python, creado por la famosa compañía JetBrains<sup>14</sup>, especializada en crear herramientas para desarrolladores.

Se ha utilizado la versión *Community Edition* 2016.3.3, gratuita y de código abierto. Podemos descargarlo desde: <https://www.jetbrains.com/pycharm/download/>

---

<sup>12</sup>Incluye también entornos móviles como Android e iOS.

<sup>13</sup>Más información en <http://www.emgu.com/wiki/index.php/Emgu.CV>.

<sup>14</sup>Más información en su página web <https://www.jetbrains.com/>

## Visual Studio

Visual Studio es un IDE desarrollado por Microsoft. Actualmente se encuentra en la versión de 2017, aunque se ha utilizado la versión gratuita de 2013 (*Visual Studio Express*) para poder compilar la aplicación de stitching para sistemas de 32 y 64 bits junto con OpenCV, como comentaremos en la sección 5.2.

La versión gratuita más actual podemos encontrarla en: <https://www.visualstudio.com/es/downloads/>

## 4.8. Otras herramientas

En este apartado se comentarán otras herramientas que se han utilizado a lo largo del proyecto.

### PyInstaller

Esta herramienta permite crear un ejecutable a partir de código Python, con la intención de simplificar el proceso de instalación. Para ello cuenta con una opción *onefile*, que empaqueta el interprete de Python y los paquetes de los que dependa el código, de modo que solo se necesita el ejecutable resultante para que funcione nuestra aplicación.

Aunque se han realizado pruebas de exportación con PyInstaller de manera satisfactoria, no se ha podido incluir debido a problemas en el ejecutable generado al haber submódulos de Python que no se han podido encontrar, por lo que se decidió utilizar Miniconda<sup>15</sup>.

Más información sobre PyInstaller: <http://www.pyinstaller.org/>

### VirtualBox

VirtualBox es una aplicación open source y multiplataforma que nos permite virtualizar unidades de disco donde poder instalar distintas distribuciones de sistemas operativos.

A cada virtualización se le pueden modificar una gran variedad de características, como por ejemplo, la memoria RAM, el espacio del disco, el número de procesadores reales que se le cede desde el equipo anfitrión y aspectos relativos al audio, pantalla y red.

En este proyecto se ha utilizado para virtualizar cuatro distribuciones de sistemas operativos, dos sistemas operativos Windows 7, uno de 32 bits y otro de 64 y dos sistemas Ubuntu 16.04 LTS, uno de 32 bits y otro de 64.

---

<sup>15</sup>Descarga básica que contiene Python y un gestor de paquetes *conda*.

Podemos descargar VirtualBox desde: <https://www.virtualbox.org/> y Ubuntu desde: <https://www.ubuntu.com/download>

### Latex

L<sup>A</sup>T<sub>E</sub>X [20] es un procesador de texto basado en instrucciones que permite generar documentos de alta calidad.

Suele ser usado en la creación de documentos científicos. Su propósito general es centrarse en el contenido del documento por encima del diseño.

Entre sus puntos fuertes, destacan el poder incluir y personalizar una gran variedad de comandos mediante la importación de paquetes para obtener el documento deseado.

Se ha usado L<sup>A</sup>T<sub>E</sub>X siguiendo la plantilla proporcionada por la universidad y escribiendo en ShareLatex<sup>16</sup>, una plataforma online que permite escribir documentos L<sup>A</sup>T<sub>E</sub>X y visualizar el resultado en formato PDF de forma rápida además de colaborar escribiendo en otros documentos en grupo.

### Google Scholar

Google Scholar (Google Académico) es el buscador de documentos científicos de Google. Entre otras cosas, permite acceder a artículos de revistas científicas, tesis, memorias, informes y extractos de libros técnicos. Permite también, subir documentos propios para que estén accesibles y puedan ser citados por cualquier persona.

Ha sido de gran ayuda para encontrar información sobre temas de detección de bordes y para citar los documentos consultados.

---

<sup>16</sup>Acceso a ShareLatex: <https://es.sharelatex.com/>



---

# **Aspectos relevantes del desarrollo del proyecto**

---

## **5.1. Requisitos generales**

Creo conveniente nombrar los requisitos generales que se establecieron en la versión anterior del proyecto [3] para conocer cual es el objetivo que se persigue. Sobre la aplicación de usuario:

- *Permite la unión de las imágenes automáticamente o semiautomáticamente para obtener una imagen completa de la pieza dental.*
- *Proporciona una interfaz gráfica al usuario, en la cual se pueden aplicar una serie de filtros sobre una imagen de una pieza dental para poder diferenciar las perikymata más fácilmente.*
- *Pide la interacción del usuario para trazar una línea que atraviesa las zonas donde las perikymata estén más marcadas.*
- *Realiza el cálculo y muestra el número total de perikymata detectados y el número de perikymata por decil (división en diez partes de la zona donde aparecen perikymata).*

*(Sergio Chico Carrancio, 2016 [3])*

## 5.2. Stitching

La aplicación de *stitching* se encarga de unir las imágenes de fragmentos del diente que han sido tomadas por un microscopio electrónico, para que el usuario no tenga que recurrir a software de terceros.

En este proyecto se han tenido que solucionar errores relacionados con esta aplicación y además, se han valorado otras formas de desarrollarla con la intención de que mejorase. Para ello debemos hablar de los temas que veremos a continuación.

### Stitching en Windows

En la versión anterior del proyecto [3], se decidió utilizar OpenCV (sección 4.6) para desarrollar, en C++, una aplicación que realizase la operación de *stitching* con las imágenes. Se utilizó la última versión disponible de la librería en ese momento, la versión 3.1. Esta versión incluye sus librerías ya compiladas para Windows con el IDE Visual Studio (sección 4.7) para arquitecturas de 64 bits.

Como Java puede ser ejecutado en sistemas de 32 bits, es una limitación que, por parte de la aplicación de *stitching*, el proyecto quede restringido a sistemas de 64 bits, por lo que se buscó una versión anterior de OpenCV, en concreto la versión 2.4.11, que permite obtener un ejecutable de 32 bits y así abrirse a más sistemas. Por supuesto, se probó que el resultado fuese el mismo que con la otra versión más moderna.

Otra limitación solucionada en cuanto a la aplicación de *stitching* era que, para poder unir las imágenes, el ejecutable debía estar en la misma carpeta que la aplicación Java junto con sus dependencias *dlls*.

La versión 2.4.11 de OpenCV para Windows trae ya compilado su código fuente en forma de librerías dinámicas o estáticas y a la hora de crear una aplicación se utilizan por defecto las dinámicas con la idea de que quien haga uso de la aplicación tenga instalado en su sistema la misma versión de OpenCV.

Eliminar esta dependencia de las *dlls* en la misma carpeta se solucionó configurando Visual Studio para que utilizase las librerías estáticas. Curiosamente las librerías compiladas de la versión 2.4.11 solo dan soporte hasta la versión 2013 de Visual Studio, por lo hubo que utilizar esa versión más antigua del IDE<sup>17</sup>.

---

<sup>17</sup>También podría haberse compilado el código fuente con *CMake*.

## Stitching en Linux

Como este proyecto también buscaba dar soporte a más sistemas, se ha usado OpenCV para crear un ejecutable de *stitching* que pueda ser utilizado en sistemas Linux<sup>18</sup>.

Para ello, se ha tenido que descargar el código fuente de OpenCV y *CMake* para poder compilar la aplicación, de nuevo, configurando un uso de las librerías en modo estático para no generar dependencias.

Se obtuvo un ejecutable de 32 bits y se encontró que no funcionaba en Ubuntu de 64 bits, por lo que se compiló otro para 64.

### Errores encontrados y sus soluciones

La aplicación Java se encarga de ejecutar la aplicación de *stitching* indicándole, en los argumentos, las imágenes a juntar. A veces las imágenes no se podían unir, y se encontró que el error era que las imágenes estaban en rutas con al menos un espacio en blanco y el ejecutable no era capaz de gestionarlo.

Se propuso utilizar las carpetas temporales por defecto de los sistemas Windows y Linux en las que copiar el ejecutable de *stitching* y las imágenes a juntar para solucionar el problema.

En Linux es una buena solución porque la carpeta temporal suele ser */tmp*, que tiene permisos 777, y pueden usar todos los usuarios salvo que el administrador lo modifique.

En Windows se encuentra definida en la variable de entorno *temp*, que suele apuntar a la carpeta *C:/Users/NOMBRE/AppData/Local/Temp/* que es donde recurre Java si no se le indica a la máquina virtual (JVM) ninguna otra carpeta con el comando *java -Djava.io.tmpdir=CARPETA*.

Como vemos, las carpetas temporales pueden llegar a ser muy variables y para solucionarlo, además de poder elegir la carpeta por defecto, se ha añadido una opción para que el usuario seleccione una carpeta temporal válida de su preferencia (figura 5.20).

---

<sup>18</sup>Se ha desarrollado en Ubuntu 16.04 LTS para 32 y 64 bits.

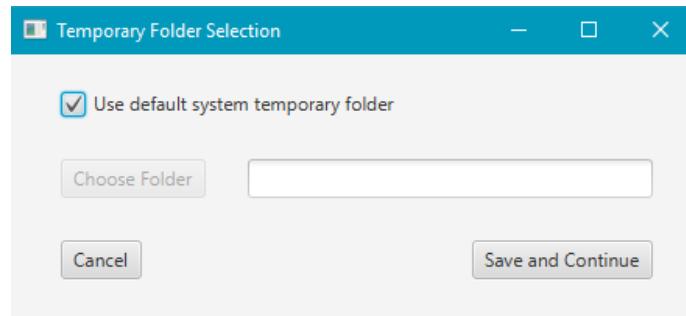


Figura 5.20: Opción de carpeta temporal

Cuando se inicia la aplicación, si ya hay un proyecto creado, se cargan sus imágenes desde la carpeta *Fragments*. Se encontró que no solo se cargaban las imágenes, sino todos los ficheros de la carpeta, y a la hora de unir las imágenes evidentemente, fallaba. La solución fue añadir un filtro para que únicamente se aceptasen ficheros que son imágenes que puede usar la aplicación.

### Stitching con Python

Como previsiblemente el procesado de la imagen del diente para resaltar las perikymata se iba a realizar en Python, se valoró realizar el *stitching* con este lenguaje puesto que otro de los objetivos del proyecto es buscar otras alternativas no probadas en la versión anterior del proyecto [3].

Tras investigar en la web, la recomendaciones sugerían utilizar OpenCV para Python. Durante el proceso, se realizaba un búsqueda de los puntos comunes de las imágenes a unir, como en la figura 5.21.

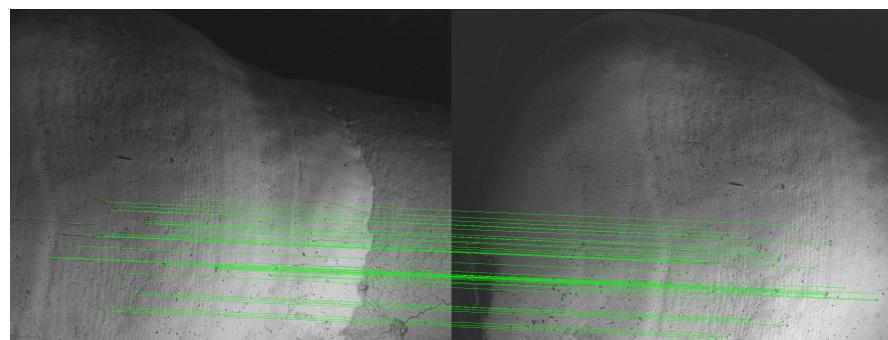


Figura 5.21: Puntos comunes de *stitching* con Python

Después se unían las imágenes teniendo como resultado la imagen de la figura 5.22.

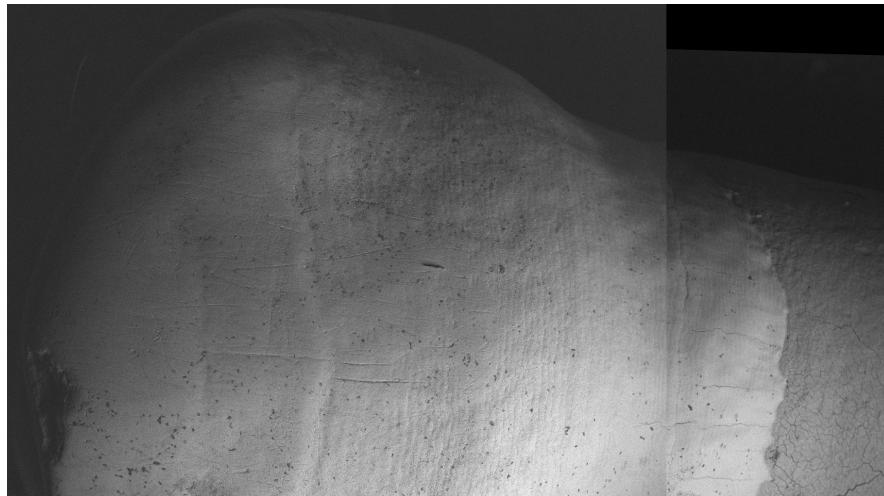


Figura 5.22: Resultado de *stitching* con Python

Este resultado es peor que con la aplicación en C++ (también desarrollada en OpenCV) porque, como se ve, se nota la unión de las imágenes y el resultado no queda armonizado, es decir, no se consigue una homogeneización en el color y la intensidad en la unión de las dos imágenes.

Por todo esto, se decidió no invertir más tiempo en la operación de *stitching* y continuar con los demás aspectos del proyecto.

### 5.3. Imágenes de dientes

Tanto este proyecto como el anterior, han estado basados fundamentalmente en las perikymata de las dos imágenes de dientes que fueron facilitadas desde el Laboratorio de la Evolución Humana. A lo largo de los archivos del proyecto podemos encontrarlas nombradas como *diente 1* y *diente 2*.

Estas dos imágenes han sido las únicas disponibles hasta, prácticamente, el último mes de desarrollo del proyecto.

Realizar pruebas sobre la primera de ellas (figura 5.23) ha sido complicado, porque a simple vista la mayoría de las perikymata ni siquiera se intuían. Conforme avanzaba el proyecto, se desechó la idea de trabajar sobre la imagen del diente 1 porque los filtrados probados no eran lo suficientemente prometedores en ningún caso.

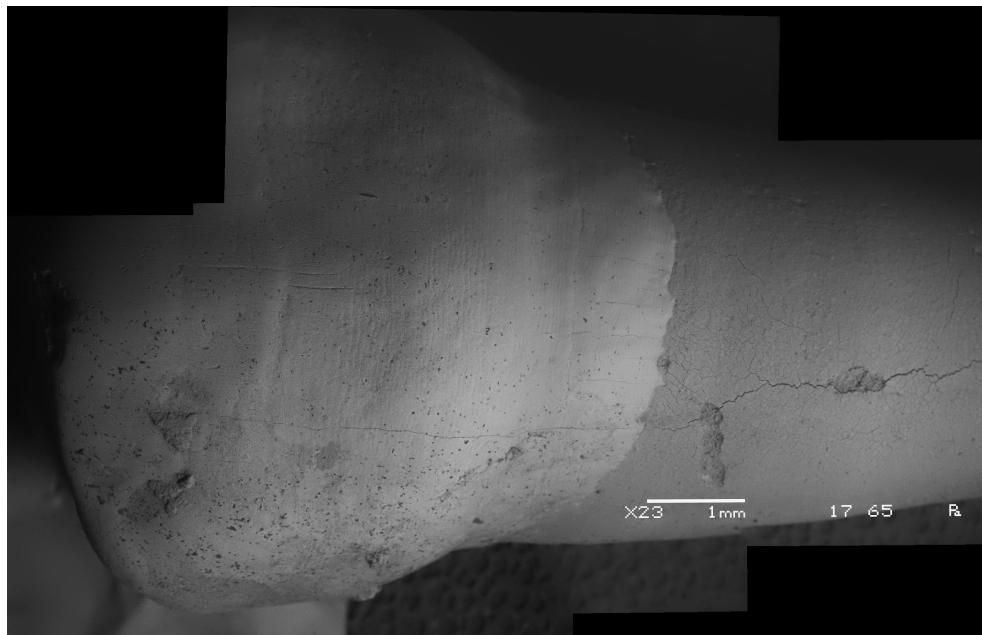


Figura 5.23: Imagen del diente 1

A falta de más imágenes, se continuó trabajando sobre el *diente 2* (figura 5.24), que respondía mejor a los filtrados aplicados.

Estas dos imágenes de los dientes generaban incertidumbre respecto a cual es el caso habitual de las imágenes, es decir, si normalmente las perikymata son distinguibles o no.

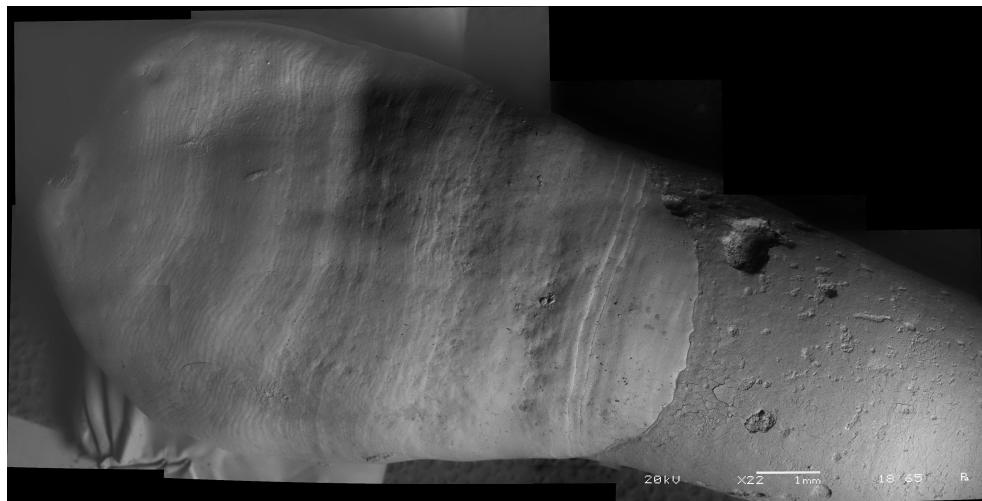


Figura 5.24: Imagen del diente 2

A finales de mayo se consiguieron más imágenes y de hecho, las perikymata se aprecian bastante mejor que en las dos primeras como veremos en el apartado [5.6](#).

Cuando nos facilitaron más, nos comentaron que es complicado obtener imágenes de dientes, porque en muchas ocasiones las pocas muestras que hay están muy deterioradas y necesitan ser cuidadosamente limpiadas antes de tomar las fotografías con el microscopio electrónico.

## 5.4. Primeros filtrados

Al principio, se trabajó elaborando un procesado basado en el filtro Frangi (sección [3.4](#)).

Para poder comprobar como de eficaz era el filtrado se utilizaron máscaras. Una máscara es una imagen elaborada por una persona que sirve para evaluar de manera objetiva métodos de visión artificial.

Se tomaron fragmentos pequeños de la imagen donde aparecían las perikymata a modo de prototipo. Con estos fragmentos, se elaboraron máscaras donde se marcaban las zonas de la imagen que sí eran perikymata, se procesaban con el filtrado y se comparaban con los resultados del procesado con Frangi sobre un fragmento sin alterar (figura [5.25](#)).

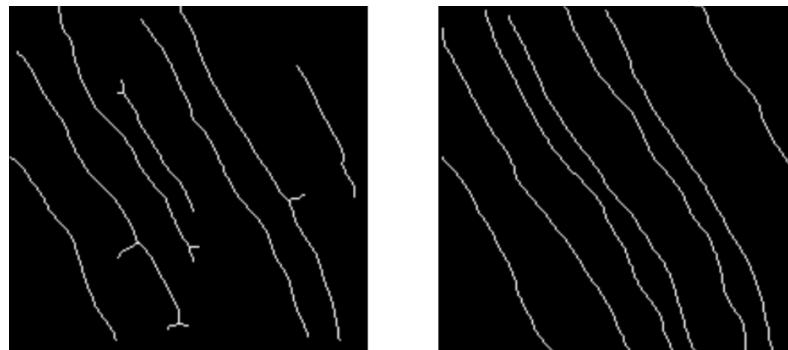


Figura 5.25: Procesado Frangi con la imagen (izda.) y la máscara (dcha.)

El resultado fue muy satisfactorio, pero al aplicarlo a una imagen del tamaño de las que se utilizan en la aplicación, no fue suficientemente bueno, encontrando que las perikymata no eran capaces de distinguirse como vemos en la figura [5.26](#) (las imágenes han sido recortadas para apreciar el resultado).

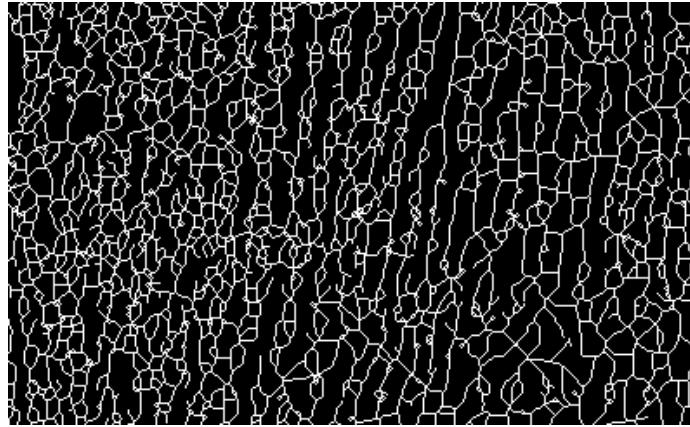


Figura 5.26: Procesado Frangi sobre el diente 2

Con esas imágenes con líneas indistinguibles se trató de reutilizar y modificar algunas funciones del Trabajo de Fin de Grado de Ismael Tobar (sección 6.2), que utilizan teoría de grafos [33], para unir las líneas verticales y formar las perikymata, aunque la mala visibilidad no permitía identificar correctamente a qué perikyma<sup>19</sup> correspondía cada línea (figura 5.27). Este proceso también tenía un coste computacional alto por lo que se desechó finalmente.

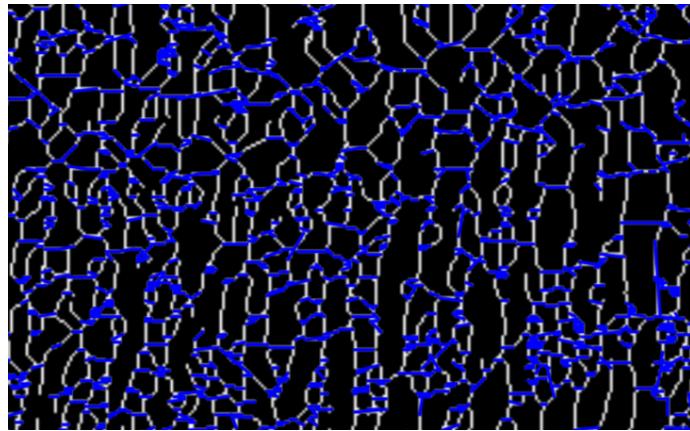


Figura 5.27: Detección de líneas sobre la imagen procesada

Con estos resultados, se dio más importancia a preparar y realzar las perikymata de una manera lo suficientemente buena antes de detectarlas y para ello finalmente se utilizó un filtrado basado en Kirsch (sección 3.4), mediante sus distintas orientaciones [17], siguiendo el proceso que se comentará en el apartado 5.6.

---

<sup>19</sup>Perikyma es el singular de *perikymata*.

## 5.5. Servidor en Python y PyInstaller

Todo el procesado que se mencionará en la siguiente sección es llevado a cabo en un servidor desarrollado en Python que atiende las peticiones de la aplicación Java a través de sockets.

Al principio, se pensó en utilizar la herramienta PyInstaller (sección 4.8) para conseguir un ejecutable sin dependencias y así evitar más instalaciones. Al igual que con el ejecutable de *stitching*, lo arrancaríamos desde Java.

En las pruebas realizadas con PyInstaller, los ejecutables tardaban bastante en arrancar, debido a que tenían contenido el código de todos los módulos usados y de las dependencias de estos, además del intérprete de Python para poder ser ejecutado.

Configurándolo adecuadamente, la exportación era satisfactoria, pero cuando se añadió el módulo de sockets fue imposible realizarla por lo que en la instalación de la aplicación finalmente debería tenerse como requisito tener instalado Python 3. En las últimas distribuciones Linux viene incluido, pero en Windows no, por lo que se pensó utilizar Miniconda3.

Se pensó distribuir el servidor de Python utilizando Anaconda Projects<sup>20</sup>, que si se tiene *conda* instalado, permite descargar el proyecto con un comando.

Finalmente este paso se consideró innecesario porque era más sencillo incluir el servidor con el resto de la aplicación y elaborar una serie de scripts para Linux y Windows que instalaran *scikit-image* (sección 4.4) y sus dependencias.

---

<sup>20</sup>Más información aquí: <http://anaconda-project.readthedocs.io/en/latest/>.

## 5.6. Procesado de la imagen final

Como bien se recoge en la memoria y anexos de la versión anterior del proyecto [3], el filtrado de la imagen consistía en aplicar una reducción de ruido con un filtro gaussiano [23] y después aplicar un filtro Prewit [29] para realzar las perikymata, todo ello con parámetros que podía definir el usuario.

Con la intención de resaltar mejor las perikymata, se ha realizado otro proceso de filtrado que comparte características con otros proyectos de fin de grado como mencionaremos más adelante en la sección 5.6 sobre trabajos relacionados.

A continuación se mostrarán los resultados del proceso de filtrado utilizando los conceptos explicados en la sección de conceptos teóricos.

### Imagen inicial

Después de unir las imágenes mediante el *stitching* encontraremos una imagen completa de una pieza dental como la figura 5.28

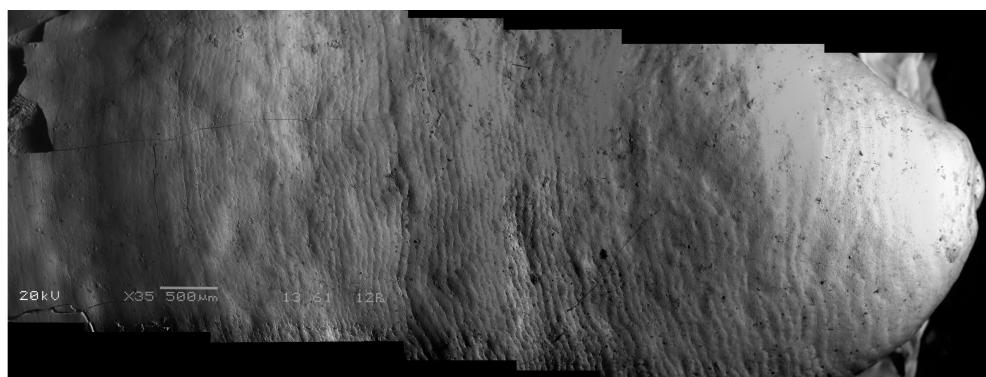


Figura 5.28: Pieza dental

A continuación veremos los cambios que se van produciendo en la imagen según aplicamos los distintos procesos.

### Ecualización adaptativa y reducción de ruido

Con la ecualización adaptativa (sección 3.1) aplicada de manera local mediante el algoritmo *CLAHE* [21] conseguimos una imagen en la que los bordes quedan más contrastados. A ello le reducimos el ruido mediante una función que utiliza el Algoritmo *Chambolle* (sección 3.2) y obtenemos una imagen como en la figura 5.29.

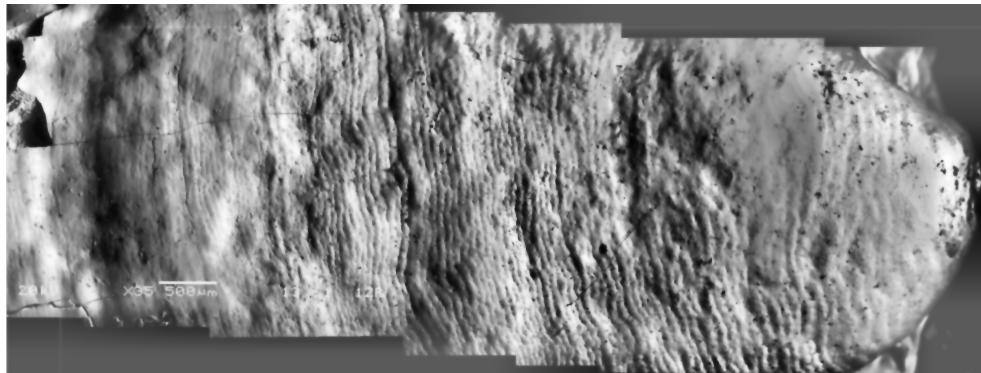


Figura 5.29: Imagen del diente ecualizada y sin ruido

### Filtro Kirsch

Aplicando un filtro Kirsch (sección 3.4) [17] de orientación Este<sup>21</sup> mediante la operación de convolución (sección 3.3), conseguimos marcar más las zonas verticales dando un resultado como en la figura 5.30.

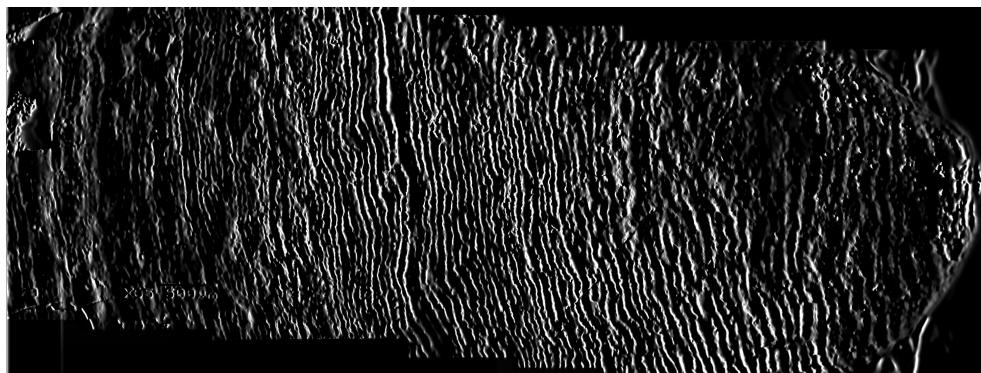


Figura 5.30: Imagen del diente con un filtro Kirsch Este

---

<sup>21</sup>Recordemos que el filtro Este se caracteriza por tener todos los valores +5 del kernel en la zona Este de la matriz.

## Binarización

La binarización (sección 3.5) permitirá tener únicamente dos colores en la imagen, como en la figura 5.31. A nivel de computador, se interpreta como una matriz de dos dimensiones con valores *true* y *false*.

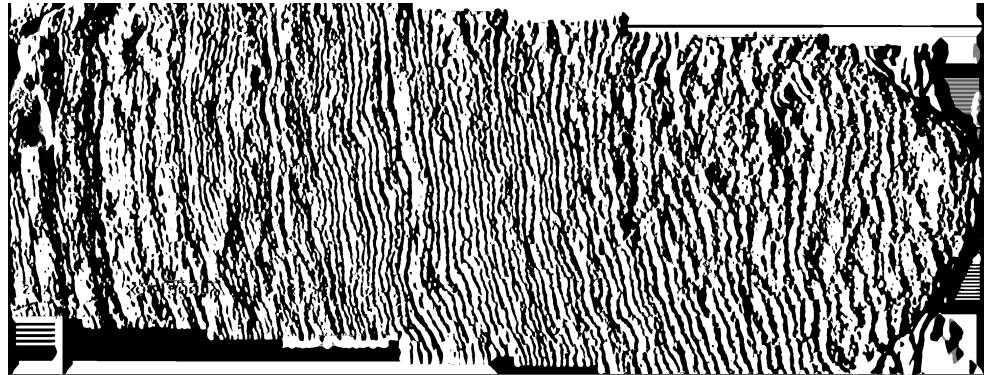


Figura 5.31: Imagen del diente binarizada

Como vemos, esta imagen justifica bien todo el proceso, pues ya obtenemos un resultado donde una persona puede distinguir, de manera aceptable, las perikymata.

## Esqueletonización y detección de líneas

La esqueletonización (sección 3.5) permitirá reducir las líneas gruesas de la imagen binarizada a una anchura de un píxel. Además, se eliminarán objetos pequeños de la imagen que puedan interferir en el proceso. Después, mediante el uso de una función que utiliza la Transformada de Hough (sección 3.6), se detectarán las líneas que tienen la orientación que nos interesa y se marcarán en la imagen, como podemos ver en la figura 3.5 (Se ha recortado la imagen para apreciar la esqueletonización y detección de perikymata).

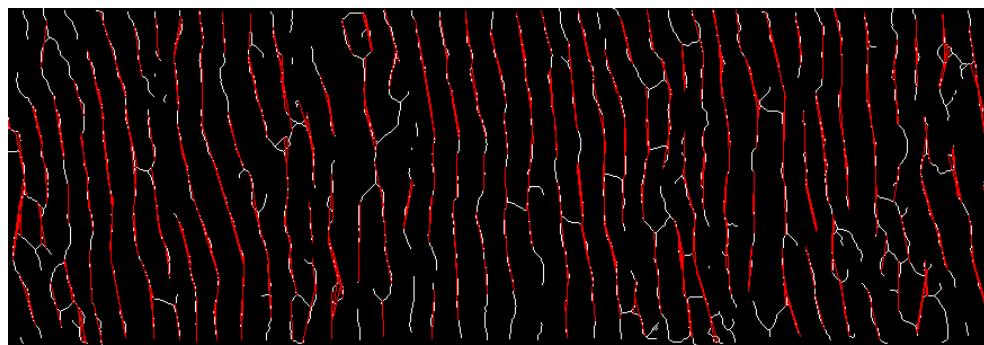


Figura 5.32: Imagen esqueletonizada con detección de líneas

## Resultado y presentación

Una vez realizado todo el proceso, se plasman las líneas detectadas sobre la imagen original y después, es responsabilidad del usuario dibujar una línea que atraviese la zona de la imagen en la que mejor se ven las perikymata.

Seguidamente, el usuario pedirá a la aplicación que marque las perikymata que atraviesan la línea, y para ello se hace uso del Modelo HSV (sección 3.7) para, finalmente, presentar una imagen como la figura 5.33.

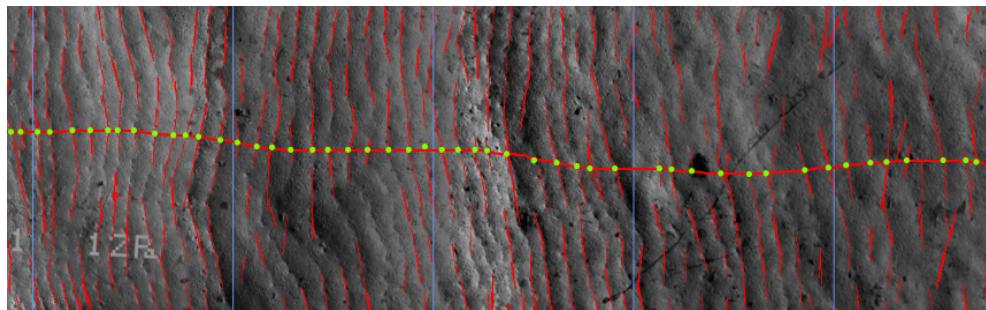


Figura 5.33: Imagen con perikymata detectadas

Este resultado y su presentación, difieren notablemente (gracias al filtrado aquí descrito) de lo que obtendríamos utilizando la versión anterior de la aplicación sobre la misma pieza dental.

Si observamos detenidamente la figura 5.34, sí que se detectaban perikymata, pero había fallos en algunas zonas y en otras se marcaban más de las que deberían. Esto es debido a que se usaba un filtrado básico y una variable *umbral* para la detección. Si el valor de esta variable era alto, no se detectaban perikymata y si era bajo, se marcaban más de las necesarias, por lo que en el punto medio, algunas las marcaba bien, otras mal y otras no las marcaba.

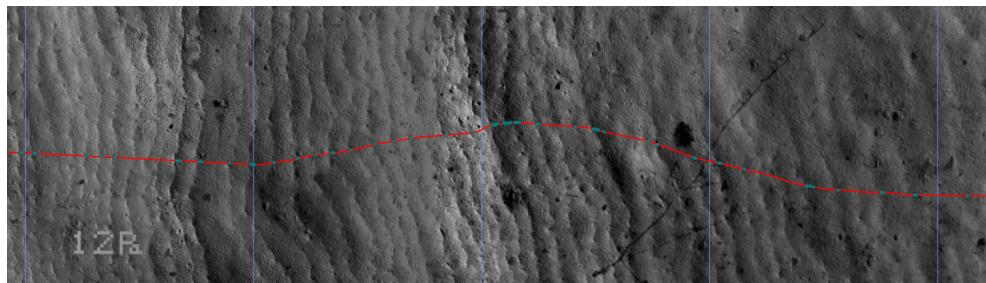


Figura 5.34: Imagen con perikymata detectadas en la versión anterior

A la vista de los resultados, podemos afirmar que se ha mejorado notablemente la detección de la perikymata y por tanto, se ha cumplido el objetivo principal de este proyecto.



---

## **Trabajos relacionados**

---

### **6.1. Análisis paleontológico de piezas dentales - Sergio Chico Carrancio**

El Trabajo de Fin de Grado de Sergio Chico es el precedente del que parte este proyecto. En los documentos que él elaboró, se detallan los primeros procesos desarrollados, las alternativas probadas, los resultados obtenidos y las decisiones que se fueron tomando, para crear una aplicación de escritorio basada en Java que permitiese detectar de forma automática la perikymata.

Ha sido de gran ayuda de cara a no volver a valorar opciones que ya se habían probado. También ha servido para sentar las bases del desarrollo en el que primero hay que procesar la imagen para resaltar las perikymata y posteriormente detectarlas y automarcarlas.

La vistas de la aplicación han sido modificas en esta versión, pero se ha continuado con la estructura general que en ese proyecto se decidió.

El proyecto puede encontrarse en esta dirección:  
<https://github.com/Serux/perikymata>

### **6.2. Estimación de la dieta por análisis de marcas dentales - Ismael Tobar García**

En el Trabajo de Fin de Grado de Ismael Tobar también se utilizaban los restos fósiles de los dientes para extraer información. En concreto, el proyecto trataba de estimar la dieta que consumían los homínidos a partir de las marcas que dejaban los alimentos en sus dientes.

El tipo de dieta se puede estimar en función de parámetros como la longitud y el ángulo de las marcas.

El autor desarrolló un procesado que permitía detectar pequeños segmentos de las marcas, y utilizaba la Teoría de Grafos [33] para unir los que pertenecían a la misma marca.

El trabajo de Ismael Tobar fue un caso de éxito y ha tenido especial importancia en este proyecto porque usaba las funciones de binarización, esqueletización y detección de líneas, entre otras, que desde las primeras reuniones con los tutores se pensó en aplicar a la detección de perikymata.

Otra función de este trabajo, como la de detectar y unir segmentos, se consiguió aplicar a la detección de perikymata sin éxito, debido a que las líneas verticales de la imagen esqueletizada estaban demasiado juntas, como hemos comentado anteriormente en la figura 5.27 de la sección 5.4.

Podemos acceder al proyecto de Ismael Tobar desde aquí:  
[https://github.com/Itg0001/TFG\\_DietaPorDientes/](https://github.com/Itg0001/TFG_DietaPorDientes/)

### **6.3. Artículo sobre los Kernels de Kirsch**

En este artículo [17] se utiliza un algoritmo para realzar conjuntos de microcalcificaciones en mamografías usando los kernels de Kirsch en las ocho direcciones que define cualquier brújula: norte, noreste, este, sureste, etcétera.

De este artículo surge la idea de usar las distintas direcciones de los kernels de Kirsch y aplicarlos al realce de las perikymata, porque en algunas imágenes encontrábamos que las líneas no eran puramente verticales u horizontales, y tener un kernel como el noroeste nos permitía marcar líneas verticales que también tienen zonas inclinadas hacia un lateral.

El artículo puede descargarse desde aquí: <https://goo.gl/cjyucv>

---

# **Conclusiones y Líneas de trabajo futuras**

---

## **7.1. Conclusiones**

Este ha sido un proyecto muy interesante a muchos niveles. Al principio me llevó tiempo situarme, porque se partía de una aplicación ya hecha e implicaba tener que conocer a fondo todos los procesos y desarrollos aplicados, las alternativas valoradas y los resultados obtenidos previamente.

La forma de resaltar las perikymata mezclando filtros de detección de bordes con otros procesos como la esqueletización, requiere invertir mucho tiempo en pruebas porque se pueden combinar de muchas maneras, aunque el resultado general que se ha conseguido en esta versión del proyecto mejora bastante el anterior.

En cuanto a la organización del código se ha tratado de ser continuista al añadir nuevas funcionalidades como el uso de carpetas temporales. También se ha intentado hacer que la aplicación sea algo más sencilla de usar para el usuario y se han añadido *tooltips* a lo largo de la aplicación para explicar cual es el papel de cada elemento en la interfaz.

Respecto a los filtros, ahora con un único botón se aplican por defecto los parámetros que en las pruebas se adaptaban mejor a las distintas imágenes para conseguir resultados aceptables. También se ha añadido una opción avanzada por si el usuario quiere ser más preciso respecto a ciertos parámetros.

El dar soporte multiplataforma ha sido complicado, sobre todo en el tema de las vistas, porque en Linux la interfaz de JavaFX no se comporta igual que en Windows, siendo a veces inestable. Cada vez que se modificaba la interfaz, ha sido necesario cargar el proyecto en el entorno Linux de desarrollo y revisar que todos los componentes funcionaban igual que en Windows.

En cuanto a mi experiencia personal, he podido aprender y trabajar con muchas tecnologías distintas, siendo Python la que más me ha gustado y la que más he agradecido por la facilidad a la hora de escribir código y realizar pruebas, algunas de ellas interactivas mediante *notebooks*.

Este proyecto me ha permitido poner en práctica muchos de los conocimientos que he aprendido a lo largo del grado, de asignaturas como por ejemplo: Gestión de Proyectos, Sistemas Distribuidos, Hardware de Aplicación Específica, Ingeniería del Software y todas las relacionadas con programación. También ha sido gratificante poder trabajar en una aplicación que facilitará el trabajo a otras personas y desde luego, estoy satisfecho con los resultados obtenidos.

## 7.2. Líneas de trabajo futuras

A continuación se expondrán las posibles modificaciones y nuevos enfoques que podrían incluirse en futuras versiones para mejorar el funcionamiento y eficacia general de la aplicación:

### Reconstrucción de la corona

Algunas piezas dentales no se encuentran en perfecto estado y tienen zonas de la corona rotas. Esto se traduce en que los deciles no están bien medidos y el área en el que está cada perikyma puede no ser la que debería. Por eso es importante tratar de recomponer la corona del diente si se encuentra dañada.

Durante este proyecto se ha tenido acceso a un artículo<sup>22</sup> que explica la forma en la que se realiza manualmente esta reconstrucción de la corona.

Desde un punto de vista de programación, creemos que la reconstrucción puede llevarse a cabo utilizando técnicas que utilicen un polinomio interpolador para estimar la curva de la zona de la corona dañada. Algunas de estas técnicas podrían llevarse a cabo en Python con librerías como Scipy<sup>23</sup>.

### Stitching

Actualmente, la aplicación de *stitching* funciona correctamente si el usuario tiene especial cuidado seleccionando los fragmentos del diente y rotando todos para que queden orientados en el mismo sentido. El fallo reside en que a veces, cuando se introducen todos los fragmentos disponibles, obtenemos una imagen en negro.

---

<sup>22</sup>En la fecha de entrega de esta memoria el artículo aún no se ha publicado.

<sup>23</sup>Aquí encontramos más información: <https://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html>.

Tras muchas pruebas, se cree que el error es debido a que la aplicación no es capaz de gestionar la unión de dos fragmentos casi idénticos, y los rechaza realizando mal toda la unión.

Se propone mejorar esta fase creando una vista que junte las imágenes de dos en dos de manera automática, y permita participar al usuario para que indique cuales son las correctas en caso de que falle o de que el resultado no sea el esperado.

### Uso de servicios web

Otra posible mejora sería ofrecer la aplicación como un servicio web. Para ello habría que trasladar toda la aplicación<sup>24</sup> y sub-aplicaciones a un servidor.

El usuario únicamente debería disponer de conexión a Internet para poder realizar su trabajo, subiría las imágenes necesarias al servidor y después tendría que interactuar con la aplicación. Además, el mantenimiento y desarrollo serían más sencillos en cuanto a que no hay que tener cuidado en ajustar la aplicación para Windows y Linux con cada cambio.

También se tendrían ventajas en cuanto al sistemas de log, donde cada vez que se produjera un fallo, se podría enviar un correo al desarrollador indicando los errores.

Otra posible interpretación sería tener la interfaz de usuario desarrollada con tecnología HTML5, CSS y Javascript que hiciera uso del procesado en Python que se ha propuesto en este proyecto.

### Otras mejoras

Otras mejoras que se pueden incorporar son:

- En cada versión de este proyecto se debería invertir un tiempo en la investigación de nuevos filtros de detección de bordes y nuevos procesados, pues esta fase siempre será mejorable aunque ahora se encuentre en un punto más que aceptable.
- Se debería crear un sistema de ayuda en la propia aplicación, para que el usuario no necesite los manuales en caso de duda.
- Se podría hacer un estudio con el personal del Laboratorio de la Evolución Humana, a fin de determinar cual es la mejor forma de diseñar las interfaces con las que interacciona el usuario.
- Si se descarta la opción de usar la aplicación como un servicio web, debería considerarse pasarla a Python íntegramente a fin de unificar el

---

<sup>24</sup>Como veíamos en la sección 4.3, JavaFX puede usarse en un navegador.

lenguaje con el que desarrollarla, ya que actualmente depende de Java con JavaFX, C++ y Python.

- En caso de continuar con la implementación que se ha seguido desde el proyecto anterior, habría que realizar mejoras en la estabilidad de la aplicación en Linux y por supuesto, pensar en nuevas formas de interacción sean intuitivas para el usuario.

---

# Bibliografía

---

- [1] OpenCV Answers. Hsv hue, 2017. (Imagen descargada).
- [2] Fundación Atapuerca. Yacimientos de atapuerca. <http://www.atapuerca.org/apartado/142/los-yacimientos>. (Accedido 10/06/2017).
- [3] Sergio Chico Carrancio. Análisis paleontológico de piezas dentales. Trabajo de fin de grado, Universidad de Burgos, 2016.
- [4] Antonin Chambolle. An algorithm for total variation minimization and applications. *Journal of Mathematical imaging and vision*, 20(1):89–97, 2004.
- [5] C Galamhos, Jose Matas, and Josef Kittler. Progressive probabilistic hough transform for line detection. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 1, pages 554–560. IEEE, 1999.
- [6] GIMP. Gimp descargas, tutoriales y foros. alternativa a photoshop gratis y libre. <http://www.gimp.org.es/>. (Accedido 10/06/2017).
- [7] ImageJ. Imagej. <https://imagej.nih.gov/ij/index.html>, 2017. (Accedido 19/06/2017).
- [8] Itseez. *The OpenCV Reference Manual*, 2.4.9.0 edition, April 2014.
- [9] Jupyter Notebook. What is jupyter notebook?, 2017. (Accedido 19/06/2017).
- [10] Oracle. JavaFx overview. <http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784>, 2017. (Accedido 19/06/2017).

- [11] Stanley Osher, Martin Burger, Donald Goldfarb, Jinjun Xu, and Wotao Yin. An iterative regularization method for total variation-based image restoration. *Multiscale Modeling & Simulation*, 4(2):460–489, 2005.
- [12] César Represa Pérez. Procesamiento digital de imagen y vídeo v2.2. Apuntes Hardware de Aplicación Específica, Universidad de Burgos, 2016. (Secciones 2 y 3).
- [13] Scikit-Image. Frangi filter. <http://scikit-image.org/docs/dev/api/skimage.filters.html#skimage.filters.frangi>, 2017. (Accedido 09/06/2017).
- [14] Scikit-Image. Probabilistic hough line. [http://scikit-image.org/docs/dev/api/skimage.transform.html#skimage.transform.probabilistic\\_hough\\_line](http://scikit-image.org/docs/dev/api/skimage.transform.html#skimage.transform.probabilistic_hough_line), 2017. (Accedido 11/06/2017).
- [15] Scipy. Scipy overview. <https://www.scipy.org/about.html>, 2017. (Accedido 19/06/2017).
- [16] Raúl Marticorena Sánchez, David H. Martín Alonso, José Miguel López Robledo, and Carlos Pardo Aguilar. Introducción al lenguaje java. Apuntes Metodología de la programación, Universidad de Burgos, 2013.
- [17] A Venmathi, E Ganesh, and N Kumaratharan. Kirsch compass kernel edge detection algorithm for micro calcification clusters in mammograms. *Middle-East Journal of Scientific Research*, 24(4):1530–1535, 2016.
- [18] Wikipedia. Procesamiento digital de imágenes — wikipedia, la enciclopedia libre, 2016. (Accedido 11/06/2017).
- [19] Wikipedia. Roberts cross — wikipedia, the free encyclopedia, 2016. (Accedido 09/06/2017).
- [20] Wikipedia, 2017. (Accedido 19/06/2017).
- [21] Wikipedia. Adaptive histogram equalization — wikipedia, the free encyclopedia, 2017. (Accedido 10/06/2017).
- [22] Wikipedia. Eigenvalues and eigenvectors — wikipedia, the free encyclopedia, 2017. (Accedido 08/06/2017).
- [23] Wikipedia. Gaussian noise — wikipedia, the free encyclopedia, 2017. (Accedido 08/06/2017).
- [24] Wikipedia. Git — wikipedia, la enciclopedia libre, 2017. (Accedido 19/06/2017).
- [25] Wikipedia. Hessian matrix — wikipedia, the free encyclopedia, 2017. (Accedido 11/06/2017).

- [26] Wikipedia. Hsl and hsv — wikipedia, the free encyclopedia, 2017. (Accedido 11/06/2017).
- [27] Wikipedia. Image noise — wikipedia, the free encyclopedia, 2017. (Accedido 10/06/2017).
- [28] Wikipedia. Lenna — wikipedia, the free encyclopedia, 2017. (Accedido 12/06/2017).
- [29] Wikipedia. Prewitt operator — wikipedia, the free encyclopedia, 2017. (Accedido 10/06/2017).
- [30] Wikipedia. Python — wikipedia, la enciclopedia libre, 2017. (Accedido 19/06/2017).
- [31] Wikipedia. Rgb color model — wikipedia, the free encyclopedia, 2017. (Accedido 11/06/2017).
- [32] Wikipedia. Sobel operator — wikipedia, the free encyclopedia, 2017. (Accedido 09/06/2017).
- [33] Wikipedia. Teoría de grafos — wikipedia, la enciclopedia libre, 2017. (Accedido 23/06/2017).
- [34] Wikipedia. Total variation denoising — wikipedia, the free encyclopedia, 2017. (Accedido 10/06/2017).