



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática



TFG del Grado en Ingeniería Informática

**Dieta por Dientes
Documentación Técnica**



Presentado por Ismael Tobar García
en Universidad de Burgos — 12 de diciembre de 2016
Tutor: D. Álvar Arnaiz González y Dr. José
Francisco Diez Pastor

Índice general

| | |
|--|------------|
| Índice general | I |
| Índice de figuras | III |
| Apéndice A Manuales | 1 |
| A.1. Introducción | 1 |
| A.2. Planificación temporal | 1 |
| A.3. Estudio de viabilidad | 13 |
| Apéndice B Especificación de Requisitos | 15 |
| B.1. Introducción | 15 |
| B.2. Objetivos generales | 15 |
| B.3. Catalogo de requisitos | 16 |
| B.4. Especificación de requisitos | 17 |
| Apéndice C Especificación de diseño | 19 |
| C.1. Introducción | 19 |
| C.2. Diseño de datos | 19 |
| C.3. Diseño procedural | 24 |
| C.4. Diseño arquitectónico | 25 |
| Apéndice D Documentación técnica de programación | 29 |
| D.1. Introducción | 29 |
| D.2. Estructura de directorios | 29 |
| D.3. Manual del programador | 31 |
| D.4. Compilación, instalación y ejecución del proyecto | 33 |
| D.5. Pruebas del sistema | 33 |
| Apéndice E Documentación de usuario | 35 |
| E.1. Introducción | 35 |

ÍNDICE GENERAL

II

| | |
|---|-----------|
| E.2. Requisitos de usuarios | 35 |
| E.3. Instalación | 35 |
| E.4. Manual del usuario | 35 |
| Apéndice F Procesado automático de la imagen | 36 |
| F.1. introducción | 36 |
| F.2. Por procesado de imagen | 36 |
| Bibliografía | 45 |

Índice de figuras

| | |
|---|----|
| A.1. Burndown del sprint 1 | 3 |
| A.2. Burndown del sprint 2 | 5 |
| A.3. Burndown del sprint 3 | 6 |
| A.4. Burndown del sprint 4 | 7 |
| A.5. Burndown del sprint 5 | 8 |
| A.6. Burndown del sprint 6 | 9 |
| A.7. Burndown del sprint 7 | 11 |
| A.8. Burndown del sprint 8 | 12 |
| A.9. Burndown del sprint 9 | 14 |
| C.1. Diagrama de clases inicial | 20 |
| C.2. Diseño de la interfaz de usuario | 21 |
| C.3. Diseño de la interfaz de usuario | 22 |
| C.4. Barra de herramientas de la aplicación | 22 |
| C.5. Barra de herramientas de la imagen | 23 |
| C.6. FigureCanvas de la imagen | 23 |
| C.7. Pestañas | 24 |
| C.8. Diagrama de paquetes inicial | 27 |
| F.1. Resumen visual filtros usados. | 43 |
| F.2. Pasos del procesado. | 44 |

Apéndice A

Manuales

A.1. Introducción

Para la planificación temporal, hemos usado la metodología Scrum [3], que consiste en ir haciendo iteraciones o sprints que van dando valor añadido al proyecto, siempre tenemos versiones, de forma incremental nuestro proyecto va teniendo mas valor.

Desde GitHub seguimos estos pasos cada semana:

- Creamos una Milestone con duración de una semana.
- Creamos la issue o tarea correspondiente a la reunión semanal de una hora.
- Vamos creando issues correspondientes a las demás tareas aquellas que son parecidas se incluyen en la misma tarea que se subdivide en puntos.
- Respecto a la gestión temporal se realiza desde ZenHub, que es una herramienta incluida en el navegador e integrada en GitHub, las tareas se pasan de nuevas a abiertas
- A medida que vamos finalizando las tareas las vamos incluyendo en cerradas, para poder ver el gráfico o burdownchar que nos indica el progreso perfecto frente al progreso real.

A.2. Planificación temporal

Sprint 0 (9/9/2016 - 16/9/2016)

Se ha hablado del problema a resolver.

Se va a hacer un mini prototipo para evaluar las herramientas, librerías y algoritmos necesarios. Posteriormente a la reunión con el cliente (Rebeca) se decidirá el lenguaje y librerías a utilizar.

En esta primera iteración se ha hablado de evaluar las distintas herramientas de gestión, documentación y de programación y las tareas son:

- Probar L^AT_EX
- Probar gestores de tarea:
 - Trello
 - Zenhub
 - Version One
- Gestores de versiones
 - GitHub
 - Bitbucket
- Examinar el problema, evaluar el notebook y las posibilidades de las librerías
- Echar un vistazo al artículo

Cumplido:

Esta semana como aun no sabía muy bien cómo usar el repositorio la gráfica no nos dice nada, porque tuvimos que cambiar el uso de los milestones.

El milestone inicial, ya que no era posible con GitHub usarlo como deseábamos, hasta la semana 1 no pondré burndown porque no refleja nada del trabajo hecho.

Todos los puntos han sido realizados y destacar, la implementación para el algoritmo ha sido amena, y ha funcionado aunque aun tiene cosas que otras semanas mejoraremos.

Sprint 1 (16/9/2016 - 22/9/2016)

En esta semana vamos a tener tareas de interfaz gráfica , de documentación y de codificación y las tareas son:

- Mejora de la detección de las líneas que quedas solapadas.
- Analizar herramientas de interfaces gráficas y comparativa.

- PyQt4.
- WxWidget.
- Prototipado inicial de la herramienta y documentar el prototipo.

Cumplido:

Esta semana hemos hecho algunos de los puntos mas relevantes del proyecto ya que la interfaz ha sido realizada correctamente con uso de layouts para facilitar el re escalado de las pantallas sin que se oculten o descoloquen botones.

Hemos ampliado el rango de frameworks de interfaces con Tkinter y WxPython porque al investigar vimos que también eran muy relevantes en este campo.

En cuanto a la mejora de la detección de líneas también mejoramos el algoritmo ajustando los parámetros y cambiando algunas propiedades.

También al aveces fallar y como aun no sabemos si es posible dejar pasar el fallo hemos implementado por recomendación de los tutores un modo manual para encontrar las líneas que no encontraba el algoritmo, a su vez también valdrá para pintar una imagen vacía manualmente.

Gráfico del sprint 1 [A.1](#).



Figura A.1: Burndown del sprint 1

Sprint 2 (22/9/2016 - 30/9/2016)

En la semana dos vamos a abarcar puntos de la interfaz y puntos de la documentación del proyecto y las tareas son:

- Documentación.
 - Aspectos relevantes.
 - Técnicas y herramientas.
 - Planificación temporal.
- Listas en la interfaz gráfica (Usar tablas para mostrar las rectas que hemos añadido manualmente).
- Cargar imágenes con el file chooser.

Cumplido:

Hemos cumplido los objetivos aunque mas adelante y después de una revisión seguramente tengamos que modificar algunas cosas, añadir mas documentación ya que es la primera semana de documentación del proyecto.

Respecto al punto de la tabla donde aparezcan las listas de líneas que vamos añadiendo queda preguntar, si vendría bien añadir tres botones mas al modo manual.

Cosa que en la reunión con el Cliente (Rebeca) voy a exponer y posteriormente si parece bien desarrollar.

En cuanto al punto de cargar las imágenes con un *file chooser* de paso, como me sobró algo de tiempo añadí una pantalla de inicio con un mensaje, así la primera vez que abramos la herramienta no se muestren tablas vacías ni una imagen predefinida.

Opte por hacer usar una pagina de inicio a modo de fachada y cuando se cargue la imagen ya iniciar todas las funcionalidades de la aplicación. Gráfico del sprint 2 [A.2](#).

Sprint 3 (30/9/2016 - 7/10/2016)

En la semana tres vamos a abarcar puntos de la interfaz GUI , generar el informe, y pasar actualizar el código de PyQt4 a PyQt5 y las tareas son:

- Acabar la GUI.
 - Mostrar todas las líneas en la tabla.
 - Poder borrar la línea seleccionada.
- Informe.
 - Calcular estadísticas.
 - Mirar documentación Python.
- Pasar código de PyQt4 a PyQt5.



Figura A.2: Burndown del sprint 2

Cumplido:

Hemos cumplido los objetivos de esta semana y hemos generado funcionalidad a la tabla para agregar las líneas, también añadido que se ilumine la línea seleccionada dentro de la imagen en color amarillo. Hemos añadido funcionalidad para borrar la línea que tenemos seleccionada y también para poder limpiar la tabla completa.

Respecto a la tarea del informe, hemos calculado los estadísticos de todas las líneas, también su clasificación y escritura dentro de un fichero CSV, también la generación de una tabla L^AT_EX que se actualiza a cada ejecución con los datos que han salido para poder pegarla fácilmente a un informe.

Como nos dimos cuenta que la versión de PyQt se había actualizado de la cuatro a la cinco pues hemos pasado el código a la nueva versión y no ha sido muy difícil.

Gráfico del sprint 3 [A.3](#).

Sprint 4 (7/10/2016 - 14/10/2016)

En esta semana vamos a abarcar el diseño software de la aplicación así como sacarlo de los NoteBooks y pasarlo a un IDE en condiciones con su subdivisión en clases y paquetes y las tareas son:

- Diseño de la aplicación.
 - Diagrama de clases.
 - Diagrama de paquetes.



Figura A.3: Burndown del sprint 3

- Implementación del código.
- Corrección de las memorias.
- Herramienta SonarQube.
 - Ejecutar la aplicación.
 - Corregir las horas de débito.

Cumplido:

Hemos cumplido los objetivos del sprint. En paralelo hemos implementado el diseño y el código a medida que teníamos una parte del diseño, de forma incremental, por lo que no se queda del todo reflejado cuando cerramos las tareas.

Hemos elegido Eclipse como IDE junto con su plugin PyDev para Python.

La división en clases hemos conseguido tener una primera estimación de como estaba la aplicación. Respecto a la herramienta SonarQube hemos corregido todos los errores y defectos que salían, a mencionar que no había código repetido ni errores graves.

También detectamos un Bug y fue corregido.

Gráfico del sprint 4 [A.4](#).



Figura A.4: Burndown del sprint 4

Sprint 5 (13/10/2016 - 22/10/2016)

Esta semana vamos a continuar con el diseño de la aplicación aplicar los patrones correspondientes y paquetes de las clases.

- Aplicar patrones de diseño
 - Fachada.
 - Mediador.
 - Comando.
- Documentar sobre los patrones usados.
- XML mirar documentación sobre ello.

Cumplido:

Esta semana, hemos cumplido con los objetivos, hemos documentado los patrones y decidido que con el mediador ya nos servía y el fachada como entrada a la aplicación.

El patrón comando en Python no le hemos visto mucho sentido ya que en Python el connect con la función se hace en una sola linea por lo que no hace falta utilizar un patrón comando.

Respecto al XML hemos mirado documentación y lo hemos implementado que guarde los nombres de todos los archivos que generan un proyecto.

Gráfico del sprint 5 [A.5](#).



Figura A.5: Burndown del sprint 5

Sprint 6 (22/10/2016 - 28/10/2016)

Esta semana vamos a intentar ir terminando las tareas para poder la semana que viene tener un prototipo entregable de la aplicación.

- Completar menús de la GUI.
 - Guardar.
 - Cargar.
 - Acerca de.
 - Ayuda.
- Completar documentación de los fuentes.
 - Paquete código.
 - Paquete gui.
- Pruebas unitarias.
 - Test paquete código.
 - Test paquete gui.

Cumplido:

Esta semana, hemos cumplido con los objetivos, aunque en el gráfico no queda reflejado por el ataque del día 21 de octubre de 2016 [5] no pude cerrar las tareas del anterior sprint ni crear las de este. Respecto a los menús de la gui, hemos añadido y renombrado los campos.

Las funcionalidades añadidas son: Poder guardar los cambios, podamos abrir y modificar un proyecto, el acerca de y el botón sobre el que va a colgar la ayuda. También hemos documentado los métodos de las clases del código fuente Python para poder generar la documentación automática con PyDoc.

También hemos desarrollado las pruebas unitarias de la aplicación de todas aquellas funciones que se podía comprobar.

Gráfico del sprint 6 **A.6.**



Figura A.6: Burndown del sprint 6

Sprint 7 (28/10/2016 - 04/11/2016)

Esta semana vamos a añadir funcionalidades para que el usuario tenga mas comodidad al usar la aplicación, Evitando que pueda cometer errores o falsos positivos.

- Detectar y calcular la referencia.
 - Detectar la referencia.
 - Calcular cuanto es la referencia.
- Detectar la región pintable.
- Revisar toda la aplicación.
 - Revisar el código con SonarQube.
 - Ajustar los cálculos en algunos puntos.
 - Añadir botón de cerrar.

- Corregir las memorias.
 - Memorias.
 - Anexos.

Cumplido:

Esta semana, hemos cumplido todos los objetivos que nos habíamos propuesto, también hemos documentado bastante, no solo corregir lo que teníamos planteado.

El gráfico de esta semana se ha quedado muy desplazado hacia la derecha, porque hemos echo unas pruebas en ZenHub y al sacar varias tareas que ya estaban en cerradas, se ha despintado de la gráfica, pero todas ellas fueron terminadas en tiempo.

Las funcionalidades añadidas son:

- Detectar la región pintable: Era uno de los puntos que nos faltaban, para poder conseguir que solo se pudieran pintar, a mano, las lineas dentro del cuadrado pintado por el usuario, en las imágenes. Porque solo esta permitido para este estudio las lineas que se encuentran dentro del recuadro en la imagen.
- Detectar y calcular la referencia: En las imágenes de microscopio, se les hace una marca con los aumentos con los que se han echo las imágenes, la referencia, que marca la escala y otros datos importantes. Nuestro trabajo ha sido detectarlo y calcular cuantos píxeles son.
- Hemos añadido el botón de cerrar la aplicación que era un punto en el que aun no habíamos pensado y es necesario.
- La mejora de la calidad del software pasando la herramienta SonarQube para corregir las posibles mejoras que nos proponga dicha herramienta.

Gráfico del sprint 7 [A.9](#).

Sprint 8 (04/11/2016 - 11/11/2016)

Esta semana, va a consistir en investigar sobre el modo automático, dar valor al código en cuanto a calidad aplicando algún concepto de patrones de diseño, haciendo una fachada para la entrada salida.

- Diseño:
 - Desacoplar del mediador lo que debe ir en la fachada.

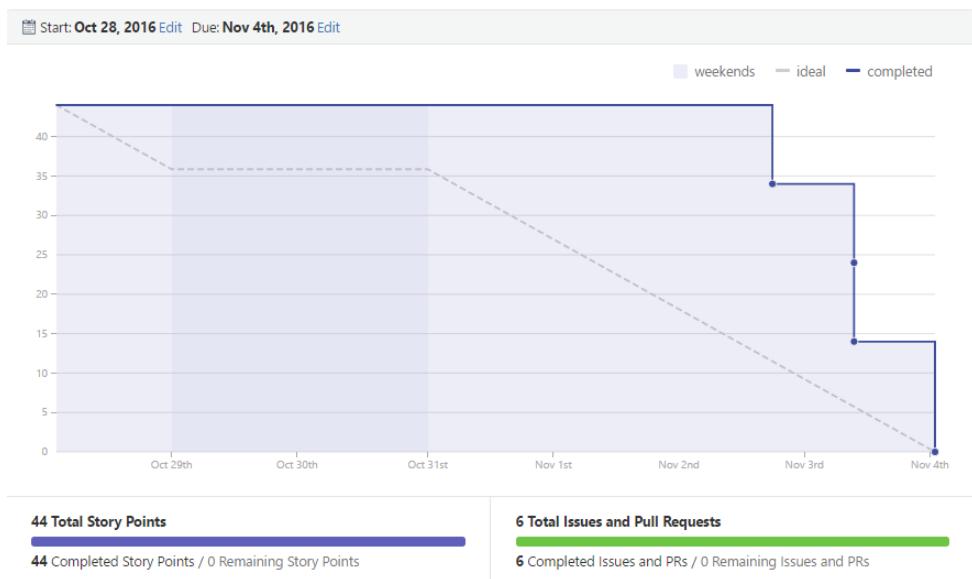


Figura A.7: Burndown del sprint 7

- Comprobar que los mediadores solo interfieren con aspectos de la GUI.
- Añadir Configuraciones al calculo.
 - Añadir las repeticiones.
 - Añadir la longitud mínima para filtrar rectas.
- Investigación detección de bordes por procesado
 - Vamos a usar todos los algoritmos y métodos disponibles en las librerías.
 - Programar otros conocidos pero que no existan en la librería.
 - Buscar mucha mas información sobre los metodos conocidos para detección de bordes.
- Investigación detección de bordes por Deep learning.
 - StrudturedForest.
 - Retina-Unet.

Cumplido:

En esta semana, hemos cumplido los objetivos, aparte ademas de buscar información sobre los métodos hemos creado un notebook con las pruebas de las ejecuciones de todos ellos.

A parte en la semana pasada implementamos la configuración, pero estaba en el código, no podía el usuario cambiarlo en función de sus necesidades. Por lo que hemos extraído esa parte a la interfaz gráfica, así puede modificar los valores de configuración, guardarse en el XML, para cuando se cargue el proyecto, también se carguen esos valores.

Gráfico del sprint 8 **A.8.**



Figura A.8: Burndown del sprint 8

Sprint 9 (04/11/2016 - 11/11/2016)

Esta semana, vamos a centrarnos en ejecutar y probar todos los métodos estudiados en la semana anterior, algunas modificaciones para facilitar al usuario, su simplicidad y mejorar, los posibles escenarios de ejecución.

- Extracción de características.
 - Aplicar Hough con valores ajustados.
 - Extraer y procesar los segmentos.
- Investigar y probar DeepLearning para Python 2.7.
 - Probar en Python 2.7 ya que en 3.5 no funcionan los algoritmos.
- Ejecutable sin dependencias.
 - Crear un .exe pero no ha sido posible ya que no hay versiones actuales de esas herramientas para Python 3.5

- Crear una alternativa.
- Documentar.
 - Documentar Memorias.
 - Documentar Anexos.
- Independencia del color.
 - Modificar la distancia al color para que funcione con todos los colores.

Cumplido:

Esta semana hemos cumplido con los objetivos, pero no todo ha sido posible su implementación o realización. Los algoritmos de DeepLearning no hemos sido capaces de ejecutarlos ya que están bastante desactualizados y las dependencias no están demasiado claras por lo que su ejecución no ha sido posible.

Todos los demás puntos del sprint, han sido realizados con éxito. La parte de la independencia del color, nos ha permitido que ahora nuestra aplicación, funcione para todas las líneas indiferentemente, del color en el que estén pintadas, el usuario elige que color tienen y las busca.

Respecto a la extracción de características, del modo automático por procesado, hemos conseguido que sea decente partiendo de que las líneas no son fácilmente detectables y las que están más marcadas no son las que nos interesan.

Para el ejecutable sin dependencias las opciones de crear el .exe no han sido posibles dada la desactualización de las herramientas destinadas a estos fines. Como alternativa hemos creado una distribución de Miniconda específica para nuestro propósito que solo contendrá las librerías necesarias para ello.

Gráfico del sprint 9 ??.

Sprint 10 (18/11/2016 - 25/11/2016)

A.3. Estudio de viabilidad

Viabilidad económica

Viabilidad legal

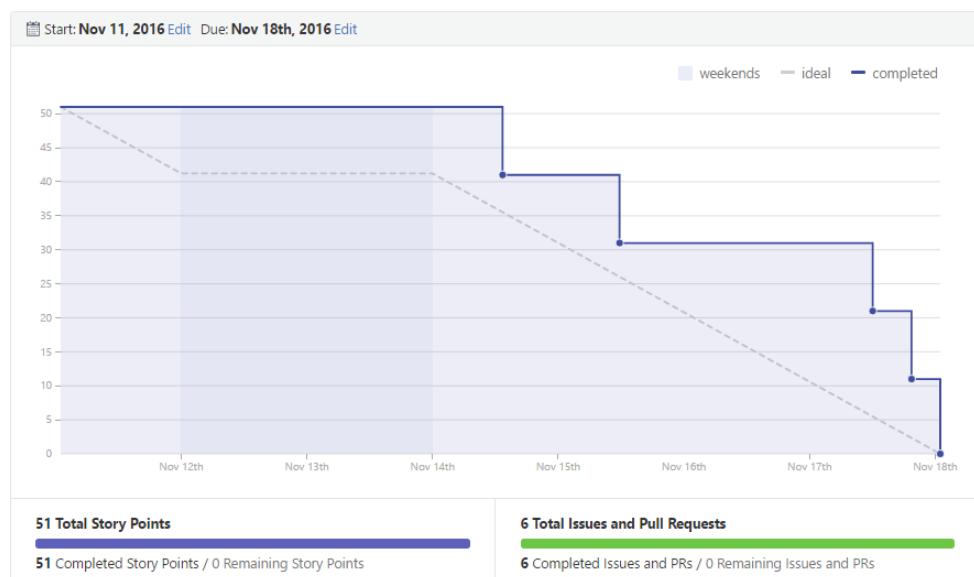


Figura A.9: Burndown del sprint 9

Apéndice B

Especificación de Requisitos

B.1. Introducción

Este anexo contendrá los objetivos de la aplicación y los requisitos que va a tener esta. Consistirán en breves descripciones de todos los objetivos principales, como se subdividen en objetivos mas cortos para conseguir realizarlos y las funcionalidades implementadas.

Requisitos: Contendrá el que consiste el proyecto y las definiciones de los requisitos y objetivos.

B.2. Objetivos generales

Los objetivos que buscamos con este proyecto son:

- Construcción del prototipo de procesado de las imágenes: Con este prototipo la funcionalidad buscada era, hacer las pruebas sobre el código, que mas adelante introduciríamos en la aplicación real, pero así podríamos ajustar los parámetros para que funcionase bien. También de esta forma estaríamos seguros que el proyecto es realizable.
- Construcción de la aplicación: Una vez que tengamos el prototipo funcionando, tendríamos que construir una aplicación para ejecutar de una forma mas profesional que simplemente desde un notebook, esto es algo muy avanzado para un usuario sin experiencia, así facilitar la ejecución sin necesidad de depender del notebook.
- Construcción del prototipo para el modo automático: Como hemos realizado el trabajo en tiempo, hemos optado por acoplar un modo de detección de bordes automatizado, para ver si eramos capaces de detectar

los segmentos que detectaría un humano. El problema de este modo es que algunos segmentos no son casi perceptibles por un humano por lo tanto los que estén mas marcados son los que podremos detectar.

- Añadir el modo automático a la aplicación: Como hemos echo anteriormente con el modo de detección de las lineas pintadas. Esta vez también lo acoplaremos a nuestra aplicación, facilitando la ejecución y aplicación de este modo sobre imágenes que no estuvieran pintadas.

B.3. Catalogo de requisitos

El proyecto consiste en:

- Construcción del prototipo de procesado de las imágenes.
 - Lectura de imagen.
 - Binarización para detectar las pintadas.
 - Procesado de la imagen binaria.
 - Extracción de características.
 - Procesado de las características.
- Construcción de la aplicación.
 - Lectura de imágenes y cargar en el panel.
 - Construcción del panel de pestañas
 - Construcción del modo de automático para lineas pintadas.
 - Construcción del modo manual para corregir los errores o editar las lineas que hemos pintado.
 - Construcción de la pestaña para el modo completamente automático.
- Construcción del prototipo para el modo automático.
 - Lectura de la imagen.
 - Equalizacion de la imagen para distribuir el histograma.
 - Binarización para extraer bordes.
 - Procesado de la imagen binaria para limpiar de ruido.
 - Calculo de las características de la imagen.
 - Procesado de características.

B.4. Especificación de requisitos

- RF1: Construcción del prototipo de procesado de las imágenes pintadas en un notebook interactivo de jupyter.
 - RF1.1: Lectura de imagen que el usuario indique.
 - RF1.2: Binarización para obtener la mascara de los segmentos.
 - RF1.3: Procesado de la imagen binaria para obtener la mascara con las lineas a detectar de grosor mínimo un pixel.
 - RF1.4: Extracción de características de la imagen para obtener los segmentos en forma de puntos de segmentos .
 - RF1.5 Procesado de los segmentos para obtener las estrías reales juntando los segmentos.
- RF2: Construcción de la aplicación de forma gráfica.
 - RF2.1: Construcción de los menús donde mostrar las opciones disponibles en la aplicación.
 - RF2.2: Construcción del panel de pestañas donde estarán las opciones para los diferentes modos de la aplicación y sus funcionalidades
 - RF2.3: Construcción del panel donde se mostrara la imagen junto con una barra de opciones con información de la imagen y botones de acciones sobre la imagen.
 - RF2.4: Permitir al usuario la lectura de la imagen y cargar en el panel.
- RF3: Construcción del modo para calcular las estrías pintadas en las imágenes.
 - RF3.1: Permitir al usuario seleccionar el color de las estrías pintadas que queremos detectar.
 - RF3.2: Permitir al usuario seleccionar la orientación de la imagen,la longitud mínima que debe ignorar y el numero de repeticiones que quiere para calcular los segmentos.
 - RF3.3: El usuario debe poder pedir a la aplicación que use el algoritmo implementado para la detección de las estrías pintadas.
 - RF3.4: La aplicación debe mostrar los segmentos que ha calculado con el algoritmo en el panel donde se muestra la imagen.
- RF4: Construcción del modo manual para corregir los errores o editar las lineas que hemos pintado.

- RF4.1: Permitir que el usuario pueda pintar líneas sobre la imagen, pero solo sobre el cuadrado, para añadir nuevos segmentos.
 - RF4.2: Permitir que el usuario pueda añadir las líneas que ha calculado el algoritmo de detección de estrías pintadas.
 - RF4.3: Permitir añadir, borrar, y limpiar la tabla de los segmentos obtenidos que se corresponden con las estrías.
 - RF4.4: Pintar, permitir mover y fijar el cuadrado donde queremos pintar las estrías manualmente.
 - RF4.5: Permitir al usuario Guardar los segmentos y obtener las estadísticas y mediciones de todos los segmentos que ha calculado el algoritmo y se muestran en la tabla.
- RF5: Construcción del notebook interactivo de jupyter, procesado de imágenes modo automático para detección automática de estrías, para imágenes sin pintar.
- RF5.1: Construcción de funcionalidad para la lectura de la imagen que el usuario indique. RF5.2: Construir funcionalidad para ecualizar la imagen para distribuir el histograma.
 - RF5.3: Construcción de un proceso de binarización para extraer bordes encontrados en la imagen.
 - RF5.4: Construcción de un procesado de la imagen binaria para limpiar de ruido.
 - RF5.5: Construcción de un proceso para calcular las estrías que aparecen en la imagen.
 - RF5.6: Construcción de un proceso para extraer las estrías a las que pertenecen cada conjunto de segmentos.
- RF6: Construcción del modo completamente automático para detectar las estrías en imágenes sin pintar.
- RF6.1: Permitir al usuario seleccionar la orientación de la imagen y la longitud mínima que debe ignorar para calcular los segmentos.
 - RF6.2: Permitir al usuario mandar calcular al algoritmo las estrías que contiene la imagen.
 - RF6.3: La aplicación debe mostrar los segmentos que ha calculado con el algoritmo en el panel donde se muestra la imagen.
 - RF6.4: Pintar, permitir mover y fijar el cuadrado donde queremos quedarnos con las estrías.

Apéndice C

Especificación de diseño

C.1. Introducción

En este apartado vamos a explicar que hemos usado para la resolución del problema en cuanto a herramientas de diseño.

Para la elaboración de los diagramas de paquetes y de clases hemos usado UML como metodología y el programa Astah para implementar los diagramas de clases.

Primero, hemos planeado un prototipo de diseño de cada una de las clases, que a medida que después las implementábamos las hemos ido completando, para que se correspondieran en lo que finalmente quedaba.

C.2. Diseño de datos

Introducción

En este punto vamos a hacer el diseño inicial de las clases de la aplicación y del diseño de paquetes en el que estarán contenidas.

Clases

En esta parte vamos a hacer una breve descripción del contenido de cada clase y su función.

- VentanaInicio: Esta clase simplemente hará de fachada entre la ejecución y el acceso a la aplicación real cuando hayamos elegido una imagen que procesar
- Window: Esta clase contendrá la ventana principal, menús, el panel de pestañas y la imagen sobre la que dibujar.

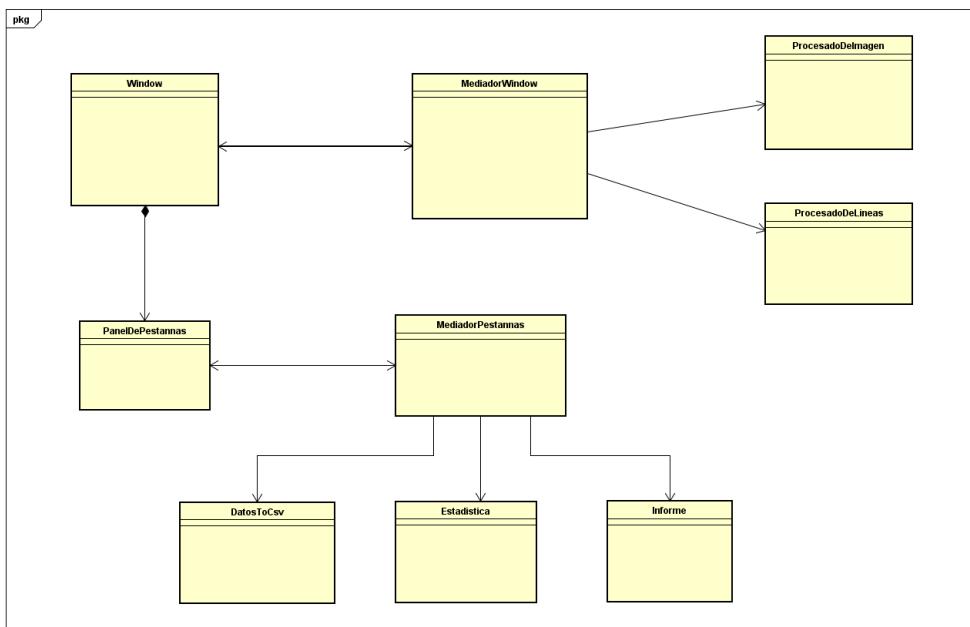


Figura C.1: Diagrama de clases inicial

- **PanelDePestanas**: Esta clase contendrá la botonería y la interacción con la imagen a evaluar.
- **MediadorPestanas**: Esta clase va a ser un mediador para las pestañas para deslocalizar código y complejidad.
- **MediadorVentana**: En esta clase va ser un mediador entre la ventana principal y sus métodos para así reducir complejidad en la clase de ventana.
- **ProcesadoDeImagen**: Esta clase contendrá el tratamiento que tiene que llevar la imagen para la extracción de características a evaluar.
- **ProcesadoDeLineas**: Esta clase contendrá el procesado de las líneas obtenidas por el procesado de la imagen y el resultado final que obtendremos.
- **Informe**: Esta clase contendrá la generación del informe (Tabla) de estadísticas en formato L^AT_EX para poder exportar fácilmente a un documento PDF.
- **DatosToCsv**: Esta clase contendrá el paso de los datos que contiene la tabla a un documento en formato csv del que desde Excel podemos extraer fácilmente los datos.
- **Estadistica**: Esta clase contendrá el cálculo de los datos estadísticos y la clasificación de las líneas según su orientación.

Diseño de la Interfaz

Primera versión

Para el desarrollo de la interfaz gráfica pensé en una planificación espacial acorde con los elementos que esta contendría. Un layout que seria muy adecuado podría ser un border layout pero como no existe en PyQt4 lo he tenido que simular gracias a apilar layouts de otros tipos. Consiguiendo tener una botonería arriba del todo y dos columnas debajo claramente diferenciadas en la que en una estuviera la imagen que vamos a mostrar y en la otra las funcionalidades, como se ilustra en la imagen C.2.

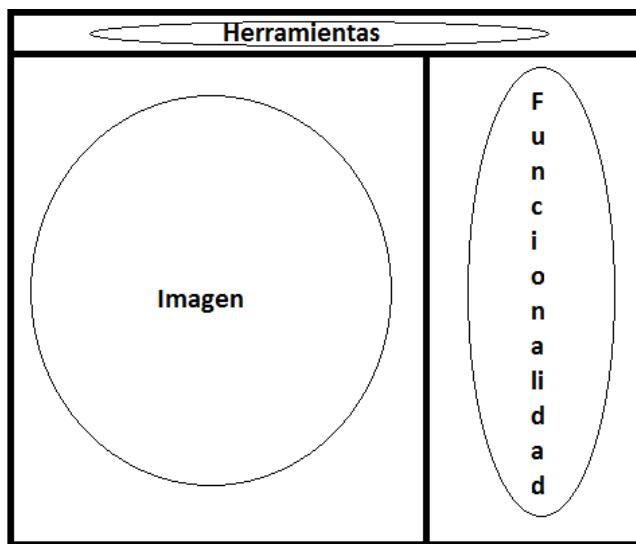


Figura C.2: Diseño de la interfaz de usuario

Versión a evaluar

En otro Sprint del proyecto lo que he usado han sido pestañas para así tener en la zona de funcionalidades los modos de trabajo de la aplicación claramente separados C.3.

- Detección líneas rojas.
- Corrección y pintar líneas.
- Automático.

Barra de herramientas

Es una sección de la interfaz que nos va permitir de momento la carga de imágenes para su procesado C.4.

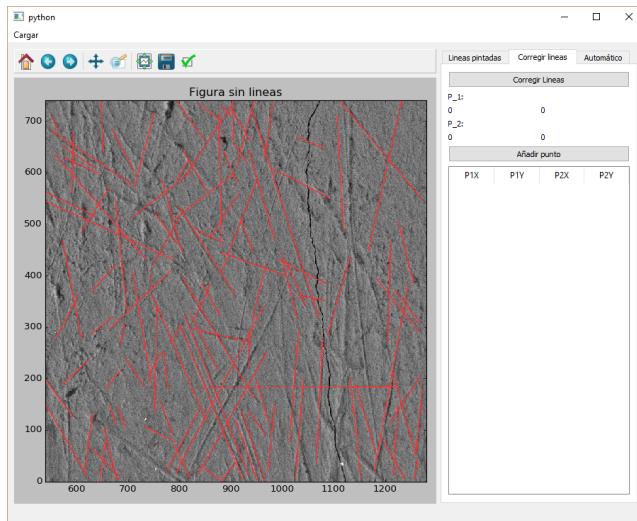


Figura C.3: Diseño de la interfaz de usuario

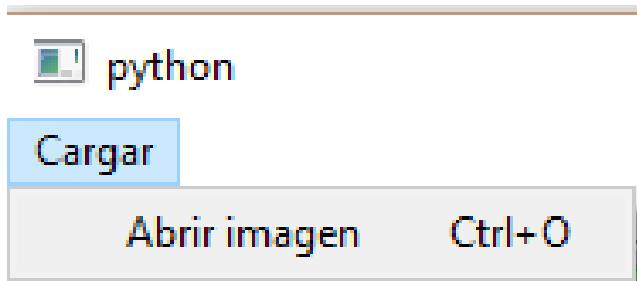


Figura C.4: Barra de herramientas de la aplicación

Barra de herramientas de la imagen

Es una sección de la interfaz que nos permitirá manipular la imagen para realizar estas acciones [C.5](#).

- Volver al principio.
- Retroceder un paso.
- Avanzar un paso.
- Desplazar la imagen.
- Aumentar la región que seleccionemos.
- Configurar la región, bordes, etc.
- Guardar la imagen con su escala.

- configurar el tamaño y numeración de los ejes.
- Coordenadas actuales del ratón.
- Niveles de color de los canales R G B del espacio RGB.



Figura C.5: Barra de herramientas de la imagen

FigureCanvas de la imagen

Es la región donde se muestra la imagen que va a ser ligeramente mas grande que la parte de las pestañas C.6.

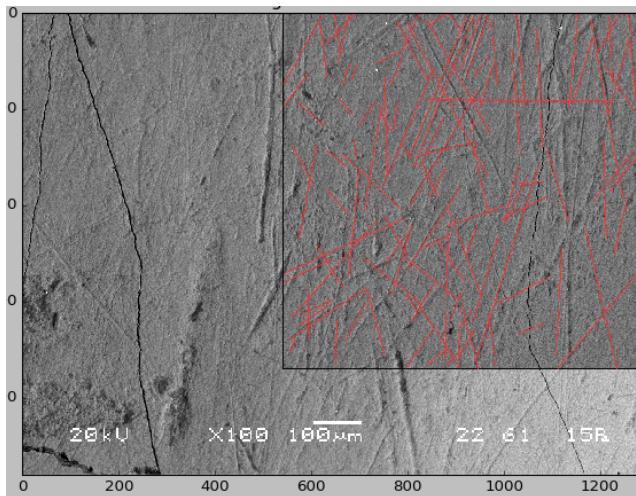


Figura C.6: FigureCanvas de la imagen

Pestañas

Sección de la imagen donde estarán implementadas las funcionalidades para visualizar las líneas que son detectadas por el algoritmo en la imagen C.7.

- El modo primero es la detección de los surcos en rojo en los dientes
- El modo segundo es la corrección/detección manual de las líneas por una persona y quedando reflejadas en la imagen.
- El modo tercero es la ultima parte del proyecto que consistirá en hacer todo el proceso anterior de forma automática.

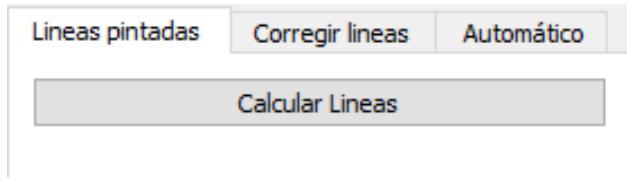


Figura C.7: Pestañas

C.3. Diseño procedimental

En este apartado se van a explicar los pasos que siguen los distintos modos de la aplicación.

- Modo semiautomático: La finalidad de este modo de trabajo en nuestra aplicación, servirá para detectar las estrías que previamente se han pintado en alguna imagen. Para extraer las características de estas.
 - Comienza cuando el usuario cargue la imagen en la aplicación.
 - Posteriormente a la carga, el usuario deberá elegir los parámetros que puede modificar, para la ejecución del algoritmo.
 - Una vez seleccionados los parámetros, podremos continuar mandando ejecutar a la aplicación el algoritmo, para la detección de las estrías pintadas.
 - Este nos va a devolver un conjunto de segmentos, que se corresponderán con las estrías que contiene la imagen pintadas, las añadirá a la tabla donde podremos editarlas o añadir algunas que podría no haber detectado dicho algoritmo.
 - por ultimo podremos guardar las estadísticas, las imágenes original, pintada, longitudes de los segmentos, ángulos, orientaciones, la tabla rellena en formato L^AT_EX para incluir en futuros informes y el fichero de configuración del proyecto.
- Modo automático: La finalidad de este modo de trabajo de la aplicación, consiste en detectar las estrías de dieta de una imagen en blanco desde el cero. Para extraer características de dichas estrías.
 - Primero el usuario deberá cargar una imagen, sin que tenga las estrías pintadas, en la aplicación.
 - posteriormente, deberá elegir los parámetros disponibles para el modo automático, este consistirá en seleccionar la orientación y la longitud mínima que ignora el algoritmo.
 - Una vez que hayan sido seleccionados se habilitara la función de detección de las estrías. Al ejecutar este algoritmo nos mostrara

todas las estrías que detecto en la imagen y nos permitirá mover la región de interés que queremos analizar.

- Al mover la región de interés que nos conviene analizar y fijarlo desechará las estrías que no estén contenidas y también los trozos de las detectadas que se salgan fuera de la región.
- Por ultimo, el paso anterior nos mostrara en la tabla de edición de los segmentos las estrías que ha detectado, pudiendo modificar o eliminar aquellas que nosotros no consideremos útiles. También nos permitirá guardar el proyecto tal y como estaba para obtener las estadísticas y demás datos relevantes de las estrías detectadas.
- Modo manual: La finalidad de este modo no es otra mas que poder incluir el factor humano en la aplicación y corregir los posibles errores o modificar ciertas estrías que nuestro algoritmo ha detectado.
 - Despues de cargar cualquier imagen en la aplicación podremos acceder a las funciones de este modo.
 - Para pintar desde cero una imagen, sin que tenga estrías, podemos hacer clic en el la pestaña de corregir lineas y mover la región de iteres por la imagen en la zona que queremos pintar.
 - Dentro de la pestaña, anteriormente mencionada, en el botón con el mismo nombre al hacer clic, la primera vez fijaremos el cuadrado y procederemos a seleccionar el origen y el final de la estría que queremos detectar, Para añadirla simplemente tendremos que clicar sobre el botón añadir punto y esta quedara incluida en la tabla. Para pintar nuevas estrías en una imagen que ya contenga estrías pintadas procederemos de forma similar, pero no podremos mover la región de interés.
 - Una vez que tengamos alguna estría pintada en nuestra imagen podremos proceder a guardar las estadísticas y todos los datos relevantes del proyecto, de forma similar a los otros modos.

C.4. Diseño arquitectónico

En esta sección del anexo de diseño, vamos a explicar y mostrar de forma general como esta diseñado el proyecto, en cuanto a la distribución de paquetes y los patrones de diseño que hemos aplicado.

Diseño de paquetes

En este punto vamos a hacer una breve descripción de cada paquete y de su contenido dentro de la aplicación.

- Gui: Este paquete contendrá todos los elementos gráficos de la aplicación y todos lo que se corresponde con interacción con el usuario.
 - Mediadores: Este paquete va a contener los mediadores de la interfaz para deslocalizar código y complejidad en esta.
- Código: Este paquete contendrá todas las clases de cálculo que necesitará la aplicación:
 - Estadísticas: Este paquete contendrá las clases del cálculo de estadísticas.
 - Informes: Este paquete contendrá la generación del informe LATEX
 - Procesado: Este paquete se va a encargar tanto del procesado de la imagen como del procesado de los segmentos extraídos hasta conseguir los segmentos finales.

Patrones de diseño

En este apartado vamos a usar de los patrones de diseño que hemos utilizado en nuestra aplicación.

El patrón más importante y con el que vamos a empezar es el medidor que consigue hacer que la interfaz esté limpia simplemente con las declaraciones de las variables que más adelante va a utilizar. La base de los patrones de diseño su libro más significativo es [1].

- Mediador [13]: Se utiliza para encapsular las interacciones entre los objetos de la interfaz por lo que se clasifica como patrón de comportamiento. Como dato curioso, es de los pocos patrones que permiten la bidireccionalidad de comunicación entre las clases, por eso es el patrón idóneo para la interacción de las clases de la interfaz gráfica. Aplicando este patrón reducimos la dependencia del código por lo que de esta forma las interfaces no ven lo que tienen por debajo y reduce la complejidad.

Obtenemos:

- Conseguir desacoplamiento.
 - Simplificar y aclarar la comunicación.
 - Centralizar el control.
 - Interfaz simple para un sistema complejo.
- Comando [11]: Se utiliza para encapsular el contenido de una función, así ni el emisor de la operación ni el receptor saber que hay por debajo, pero si poder acceder a las funciones que hay.

Obtenemos:

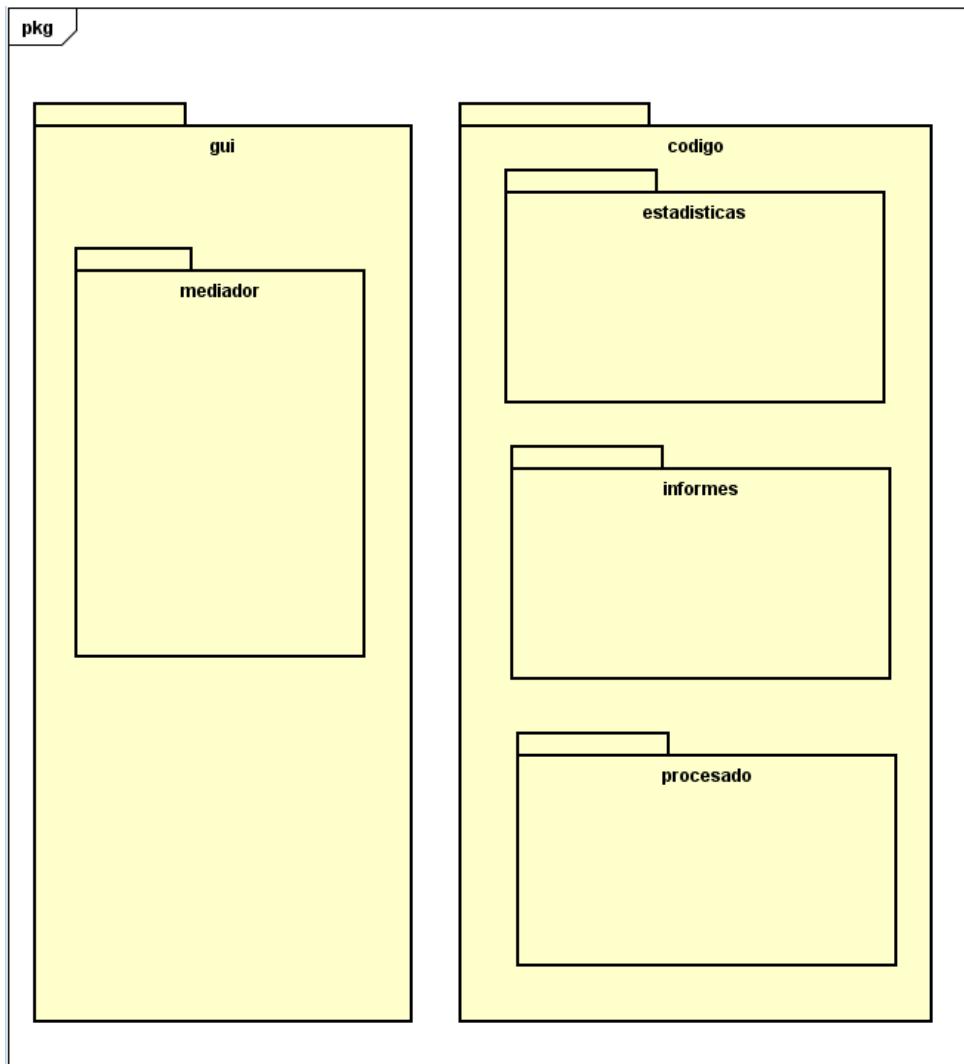


Figura C.8: Diagrama de paquetes inicial

- Permitir la parametrización.
- Visualizar que ordenes se ejecutan en cada paso.
- Construir operaciones complicadas a partir de otras más sencillas.
- Permitir saber que habría que hacer para deshacer una operación.
- Fachada [12]: Es un patrón de diseño estructural, sirve para reducir la complejidad con la división en clases más pequeñas y reduciendo sus dependencias.

Obtenemos:

- Separar en niveles la aplicación.
- Reducir su complejidad.
- Desacoplar el código.
- Interfaz simple para un sistema complejo.
- Potable y reutilizable.

Apéndice D

Documentación técnica de programación

D.1. Introducción

Esta información esta dirigida, a otros desarrolladores, que puedan leer esta documentación o continúen el proyecto, por lo que vamos a describir mas detalladamente el funcionamiento del proyecto y los aspectos que podrían mejorarse o modificarse en el futuro.

D.2. Estructura de directorios

Ejecutables:

- EjecutarGui: Este fichero se corresponde con el ejecutable para lanzar la aplicación en modo GUI.
- EjecutarTest: Este fichero se corresponde con el ejecutable para lanzar los testes de la aplicación en modo consola.
- Documentar: Este fichero se corresponde con el ejecutable para documentar la aplicación de forma recursiva y crear los htmls correspondientes.

Documentación:

En esta carpeta estará contenida la documentación en formato html.

- pydoc: Esta carpeta se corresponderá con la documentación de pydoc

Src:

Esta carpeta contendrá todos los ficheros de código fuente del proyecto, el OCR, y los tesis.

Proyecto:

Esta carpeta contendrá todos los ficheros de código fuente del proyecto, todos los módulos o paquetes. **Código**

- Estadísticas.
 - Estadística.
- Informes:
 - ConfiguracionToXML.
 - DatosToCsv.
 - Informe.
 - InGuardarDatos
- Procesado:
 - ProcesadoAutomatico.
 - ProcesadoDeImagen
 - ProcesadoDeLineas
- Diccionario:
 - Diccionario
 - DiccionarioING
- Gui:
 - Fachadas:
 - FachadaBotonesAndLayaout.
 - FachadaEntradaSalida.
 - Mediadores:
 - MediadorPestannas.
 - MediadorVentana.
 - PanelDePestannas
 - PintarRectangulo
 - VentanaInicio
 - VisorHtml
 - Window

Tesseract:

En esta carpeta contendremos el ejecutable del OCR junto con sus ficheros de configuración para poderlo ejecutar.

Test:

En esta carpeta contendremos los teses del código para comprobar que funcionan correctamente.

Código:

- Estadísticas:
 - TestEstadistica
- Informes:
 - TestConfiguracionToXML
 - TestDatosToCsv
 - TestInforme
- Procesado:
 - TestProcesadoDeImagen
 - TestProcesadoDeLineas

D.3. Manual del programador

En esta sección vamos a describir como se han programado las partes mas importantes que son las que mas nos interesan.

Procesado

Para programar el procesado hemos encadenado sucesivos pasos, que dividiremos en tres secciones.

- ProcesadoDeImagen: Este se corresponde con el procesado de la imagen para la extracción de las características.
 - 1 Hemos calculado de la imagen la distancia a un color: Se corresponde con el que estén pintadas los segmentos, lo selecciona el usuario. Evitamos cualquier color que hiciese que falle el algoritmo, filtrando la selección por el canal del modelo de color HSV, que no indica la saturación. En nuestra imagen lo único que tiene alta saturación alta son los segmentos pintados ya que la imagen esta en grises.

2 Binarizamos la imagen con la distancia al color para obtener la mascara con los objetos que queremos detectar y reducimos el grosor de estos.

3 Sobre la mascara reducida calculamos los segmentos por la transformada de Hough.

- ProcesadoDeLineas: Este módulo contiene el procesado de las lineas, detectadas en el ultimo paso del punto anterior.

1 Primero de todo, lo que tenemos no son segmentos completos, sino sub-segmentos que forman los segmentos reales. Tenemos que unirlos aquellos que sean muy similares, caracterizados por distancia y ángulo.

2 Para realizar la union lo primero que haremos sera ir añadiendo, un camino entre los dos segmentos, a un grafo, de aquellos que cumplan lo anterior.

3 Obtendremos las uno componentes del grafo, que se corresponden, con los clusters, que son los segmentos buscados.

- ProcesadoAutomatico:Este apartado aunque lleva proceso similares a los anteriores no puede ir junto ya que necesita otros pasos intermedios.

1 Usaremos, el detector de bordes que hemos implementado, primero ecualizaremos la imagen para distribuir su histograma, segundo calculamos los autovectores de la matriz Hessiana y nos quedamos con los autovectores largos, tercero binarizamos la imagen y procesamos dicha imagen. Queda reflejado en anexo F

GUI

- Imagen y líneas: Para pintar las imágenes, hemos usado el backend de matplotlib, que nos muestra la imagen sobre unos ejes de coordenadas, que nos indican e informan sobre las coordenadas de cada pixel, gracias a esto y a los métodos de conexión sobre los eventos, hemos podido simplificar al máximo la obtención de los puntos, la forma de pintar los segmentos detectados y la interacción de usuario.
- OCR: Para leer la referencia de la imagen hemos usado un OCR conocido que se llama Tesseract a este le pasamos el recorte de la imagen que contiene la referencia y nos devuelve el numero que contiene.

D.4. Compilación, instalación y ejecución del proyecto

Compilación:

En Python no hace falta compilar el proyecto ya que es un lenguaje interpretado. necesitaremos únicamente tener Python instalado, atraves de Miniconda que es una distribución Python.

Instalación:

Para poder ejecutar deberemos instalar Miniconda en C.Users.TuUsuario, Es el directorio predefinido por Miniconda, Una vez que lo tengamos instalado podremos proceder a hacer doble clic sobre el ejecutable EjecutarGui.bat que instalara las dependencias a las librerías que necesitamos para su ejecución.

Ejecución del proyecto:

Para la primera ejecución si no tenemos las librerías instaladas al hacer doble clic sobre EjecucionGui.bat, tardara un rato en descargarlas y crear el entorno virtual de Miniconda con únicamente las librerías que se usan, pero las siguientes veces únicamente ejecutara la aplicación.

Si la descarga de las librerías se interrumpe por una caída de la red o algo nos dará un error. Pero si volvemos a ejecutar se iniciara donde se paro la descarga, es decir no perdemos lo que ya haya conseguido descargar.

Conclusiones:

Hemos seguido otras formas de conseguir lo que queremos, Python es un lenguaje interpretado y necesitamos tener Python 3.5.

Para facilitar la instalación de Python hemos optado por usar Miniconda, es una distribución que facilita dicha instalación pero sin ninguna librería instalada, por lo que es mas ligero, 50 Megas, frente a Anaconda, cerca de 500 Megas. Quedando a nuestra disposición indicar que librerías instalar.

En la primera ejecución del código nos va a descargar e instalar todas las librerías que necesitamos.

D.5. Pruebas del sistema

En esta sección vamos a informar como ejecutar los tesis que están ubicados en la carpeta Test dentro de la carpeta src del proyecto.

Test Python:

Dichos teses están escritos en Python usando la librería unittest.

Hay muchas comprobaciones sobre las funciones de calculo y aquellas que escriben y leen ficheros, pero para la parte de la interfaz gráfica no hay echas pruebas ya que no se puede controlar del todo esta parte, pero ha sido probada por mi y todos los aspectos añadidos y no produce fallos conocidos.

Para ejecutarlos basta con ejecutarlos desde Eclipse, el main de los teses mainTest.py o también si preferimos podemos ejecutarlos desde la terminal con el mismo fichero. Otra opción para ejecutar los teses es dar doble clic sobre el ejecutable EjecutarTest que esta en la carpeta de los ejecutables.

Aparte de los datos proporcionados en la ejecución hemos puesto una salida informativa para cada test y las partes que comprueba.

Apéndice E

Documentación de usuario

- E.1. Introducción**
- E.2. Requisitos de usuarios**
- E.3. Instalación**
- E.4. Manual del usuario**

Apéndice F

Procesado automático de la imagen

F.1. introducción

En este apartado vamos a explicar lo que hemos investigado sobre la detección de bordes para el modo automático.

Esto lo vamos afrontar desde dos puntos de vista, el primero sera por el procesado de imágenes, esto consistirá en binarizar la imagen usando un kernel conocido y observar si los resultados nos muestran algún resultado factible. Como segunda opción vamos a intentar ejecutar algoritmos de Deep Learning para ver si mejora el resultado usando estos algoritmos.

Para entender este anexo y este procesado, por lo menos es necesario haberse leído los conceptos teóricos y herramientas mencionadas en la memoria ya que no deberíamos duplicar información en varios sitios.

F.2. Por procesado de imagen

Esta sección consistirá, partiendo de una imagen extraer las características, en este caso sera la detección de los bordes contenidos, se corresponderán con las estrías que queremos detectar. Lo vamos a detectar a partir de los métodos explicados a continuación.

Todas las imágenes van a hacer un proceso de convolución, con el Kernel [8] específico en cada caso.

Una convolución es una operación matemática, que no es la multiplicación de matrices tradicional, es el proceso de agregar cada porción de la imagen a sus vecinos locales ponderado por el Kernel.

Cuadro F.1: Kernel Laplace

| | | |
|---|----|---|
| 0 | 1 | 0 |
| 1 | -4 | 1 |
| 0 | 1 | 0 |

Cuadro F.2: Kernel Prewitt

| | | |
|----|----|----|
| 1 | 1 | 1 |
| 0 | 0 | 0 |
| -1 | -1 | -1 |

Este proceso no solo vale para detectar bordes, tiene estas otras aplicaciones.

- Detección de bordes: Detectar los bordes a partir del histograma.
- Normalización: Eliminar algo de ruido o contraste sobre las imágenes para distribuir mas los valores sobre esta.
- Desenfoques y difuminado: Desenfocar o difuminar la imagen a partir de hacer la operación de convolución con una mascara diseñada para ello.
- Eliminar ruido: Sirve para moderar los valores de la imagen y hacer que estén mas centrados en un rango y así eliminar el ruido aleatorio.

Laplace:

Esta función busca bordes usando el operador de Laplace [10]. En nuestro caso de tamaño 3 ya que se puede especificar el tamaño.

Como podemos observar en el Kernel de Laplace F.1.

Observaciones: Como podemos observar en la imagen F.1a, no es un método factible para nuestro propósito, porque no detecta los bordes simplemente deja algunas partes difuminadas, pero nada sobre lo que podamos trabajar para obtener la imagen binarizada. Por lo que no vamos a continuar usándolo.

Prewitt:

Encuentra los bordes usando la transformada de Prewitt [14].

Como podemos observar en el Kernel de Prewitt F.2.

Observaciones: Como podemos ver las grietas, de la imagen F.1b, las detecta bien pero las estrías de dieta son siluetas muy tenues y contiene mucho ruido.

Cuadro F.3: Kernel HScharr

| | | |
|----|-----|----|
| 3 | 10 | 3 |
| 0 | 0 | 0 |
| -3 | -10 | -3 |

Cuadro F.4: Kernel HSobel

| | | |
|----|----|----|
| 1 | 2 | 1 |
| 0 | 0 | 0 |
| -1 | -2 | -1 |

Scharr:

Con este método encontramos los bordes usando la transformada de Scharr [2].

Como podemos observar en el Kernel de HScharr F.3. todo ello partido de 16 y para las verticales es la matriz transpuesta.

Observaciones: Como podemos ver en la imagen F.1c, las estrías de dieta son muy tenues pero ligeramente mejores que en la anterior tampoco demasiado pero ligeramente, el método es algo más rápido también pero no obtenemos algo tangible.

Sobel:

Este método busca bordes usando la transformada de Sobel [16].

Como podemos observar en el Kernel de HSobel F.4.

El kernel de Sobel, partido de 4 para los bordes horizontales. Para los bordes verticales, es El kernel de Sobel, partido de 4 y transpuesto.

Observaciones: Con este método tal y como podemos observar en la imagen F.1d, crea más ruido que los anteriores pero también podemos observar la silueta de las estrías de dieta.

Roberts:

Esta función encuentra los bordes usando la operación cruzada de Robert [15].

Como podemos observar en el Kernel de Roberts F.5.

Observaciones: Como podemos observar en la imagen F.1e. Este método produce mucho ruido y las estrías son líneas demasiado tenues.

Cuadro F.5: Kernel Roberts

| | |
|----|---|
| 0 | 1 |
| -1 | 0 |

Cuadro F.6: Kernel g1

| | | |
|----|----|----|
| 5 | 5 | 5 |
| -3 | 0 | -3 |
| -3 | -3 | -3 |

Cuadro F.7: Kernel g2

| | | |
|----|----|----|
| 5 | 5 | -3 |
| 5 | 0 | -3 |
| -3 | -3 | -3 |

Kirsch:

Esta función encuentra los bordes usando el kernel de Kirsch [9]. para cada dirección. No estaba implementado en Skimage por lo que he implementado el método.

Como podemos observar en el Kernel de kirsch g1 F.6, para los bordes horizontales.

Como podemos observar en el Kernel de kirsch g2 F.7, para los bordes verticales.

Como podemos observar en el Kernel de kirsch g3 F.8, para los bordes diagonal ascendente.

Como podemos observar en el Kernel de kirsch g4 F.9, para diagonal descendente.

Observaciones: Como podemos observar en la imagen F.1i. Este método produce mucho ruido y las estrías son líneas demasiado tenues. No obstante de los métodos hasta ahora usados es en el que mas aprecian.

Cuadro F.8: Kernel g3

| | | |
|---|----|----|
| 5 | -3 | -3 |
| 5 | 0 | -3 |
| 5 | -3 | -3 |

Cuadro F.9: Kernel g4

| | | |
|----|----|----|
| -3 | -3 | -3 |
| 5 | 0 | -3 |
| 5 | 5 | -3 |

Autovectores matriz Hessiana:

Este método consiste en obtener la matriz hessiana [7] y después sus autovectores de la matriz Hessiana y nos devuelve dos matrices la matriz i1 es la matriz con el autovector más largo y la i2 es la matriz con autovector más corto.

Observaciones:

Como podemos observar en la imagen F.1f en escala de grises del autovector de los valores más largos las siluetas de las estrías de dieta son las que más se remarcán sobre un tenue fondo gris pero pudiendo ser observadas por lo que este podría ser un punto de partida.

Canny:**Eliminando ruido:**

Primero ,obtener los bordes, llamar a una función que elimina el ruido y después al detector de bordes Canny [4], para obtener los bordes. Para los parámetros usados en la función de Canny utilizaremos:

- Sigma: Un valor intermedio de 1.4.
- Umbral mínimo: Un valor muy bajo.
- Umbral Máximo: Un valor bajo pero mayor que el umbral mínimo.

Gracias a estos parámetros usados obtenemos una imagen resaltando algunos bordes pero no todo los que queremos ya que no están demasiado marcados.

Observaciones: Desde esta imagen F.1g con bastante ruido ya podemos observar que las más marcadas son detectadas pero no se consiguen diferenciar demasiado bien, pero en comparación con los demás métodos tiene de las mejores salidas aun no siendo buena de ir en esta línea tendíramos que usar esto.

Modificando parámetros Canny:

La segunda opción ha sido usar un detector de bordes Canny solamente modificando sus parámetros pero en esta opción los valores de los umbrales deben ir sin normalizar entre 0 y 1 sino entre 0 y 255.

Observaciones: Esta imagen [F.1h](#) detecta menos ruido que con la otra tentativa pero sigue siendo deficiente en cuanto a las líneas ya que aparte de detectar pocas detecta las que realmente no son estrías de dieta. Pero de ir en alguna línea seria por este camino.

Gabor:

Gabor [6] es un filtro linear con un kernel gausiano que es modulado por un onda sinusoidal plana. Principalmente se usa en visión artificial de clasificación y detección de bordes. Obtenemos un par de imágenes.

Observaciones: Como podemos observar en la imagen usando Gabor con filtro imaginario [F.1j](#). Como podemos observar en la imagen usando Gabor con filtro real [F.1k](#). Este método lo eh probado porque en la obtención de las líneas de sangre en los ojos es lo que se usa para ello pero al no ser líneas continuas y no seguir un patrón no he conseguido buenos resultados. En el filtro real no es tan malo pero en el filtro imaginario es ruido puro.

Comparativa Filtros

Como podemos ver en la figura [F.1](#) hemos usado casi todos los filtros conocidos para hacer esta comparativa, a simple vista ninguno de los métodos mencionados o probados no parece que funcione, ya que el problema no es fácil. Manualmente no observamos algunas de las estrías que los expertos marcan por lo tanto, detectar algunas sera algo efectivo.

Pero dentro de la comparativa la que mejor detecta algunos de los bordes parece el método de la Matriz Hessiana con autovectores largos. aunque en la imagen se presenta poca diferencia entre fondo y las estrías por lo que nos centraremos en ese procesado haber si conseguimos diferenciarlo y extraer las características.

Procesado:

Partiendo del análisis anterior vamos a juntar los mejores resultados y añadir pasos adicionales para obtener la mascara binaria que mas podamos ajustar a nuestras necesidades.

Pasos del algoritmo [F.2](#).

- Ecualizar la imagen original [F.2a](#), consistirá en extender el histograma de la imagen original para que no este centrado en un rango pequeño como podemos observar en su histograma [F.2b](#), pasado este paso obtendremos la imagen ecualizada [F.2c](#). Así conseguimos que su histograma [F.2d](#) este mas repartido por toda la gama de grises y no centrado en un pequeño rango.

- Autovectores de la Hessiana **F.2e**: Para detectar los bordes utilizaremos el método antes mencionado de la matriz Hessiana del cual elegiremos solo los largos, ya que los autovectores pueden ser los cortos o los largos.
- Sustraemos el fondo a la imagen **F.2f**: Como la imagen no tiene de nuevo demasiada diferencia entre el fondo y los bordes, sustraeremos el fondo, para quedarnos únicamente con las estrías detectadas, al tener ruido aparte de los elementos buscados, tendremos que eliminar dicho ruido mas adelante.
- Binarizamos la imagen **F.2g**: Como la imagen después de sustraer el fondo no es binaria, cualidad necesaria para ser una mascara, hay que aplicar un threshold o umbral para pasar a blanco o negro, binarizar, es decir los pixeles que no pasen el umbral serán negros y los que lo pasen formaran parte de los elementos que queremos detectar y seran blancos, entonces la imagen quedara binarizada.
- Eliminamos el ruido **F.2h**: Este paso anterior, en estas condiciones de trabajo genera mucho ruido, por lo que tendremos que intentar reducirlo y para ello eliminamos los trozos que son pequeños, ya que el ruido parece ser aleatorio de fragmentos muy pequeños, esto elimina la mayoría de los puntos de ruido.
- Erosionar con operador diamante **F.2i**: Para suavizar la imagen y evitar que queden líneas finas a modo de antenas, erosionamos la imagen con un operador estructurante en forma de diamante, así conseguimos que pase por la mayoría de las figuras.
- Esqueletizar la imagen **F.2j**: Una vez tengamos la imagen sin tanto ruido, reducimos el grosor de las líneas para que la función de Hough funcione, así nos detecta las líneas que corresponden a esta mascara binaria.
- Eliminar mas ruido **F.2k**: Este paso anterior vuelve a generar ruido porque algunos trozos se dividen en pequeños segmentos y con la segunda reducción de ruido conseguimos hacerlos desaparecer y quedarnos con las líneas grandes.
- Detectar los segmentos **F.2l**: Una vez que tenemos todos los segmentos detectados mediante Hough, que se corresponden con las líneas que hay en la imagen, tenemos que unir los que sean contiguos y muy cercanos, para formar segmentos mas grandes.
- Unir segmentos cercanos **F.2m**: Para este paso vamos a usar lo mismo que hemos utilizado dentro de la parte, detección de las líneas pintadas. La imágenes tienen mucho ruido y aunque hemos conseguido procesar y reducirlo, aun no vamos a detectar un alto por ciento de las estrías.

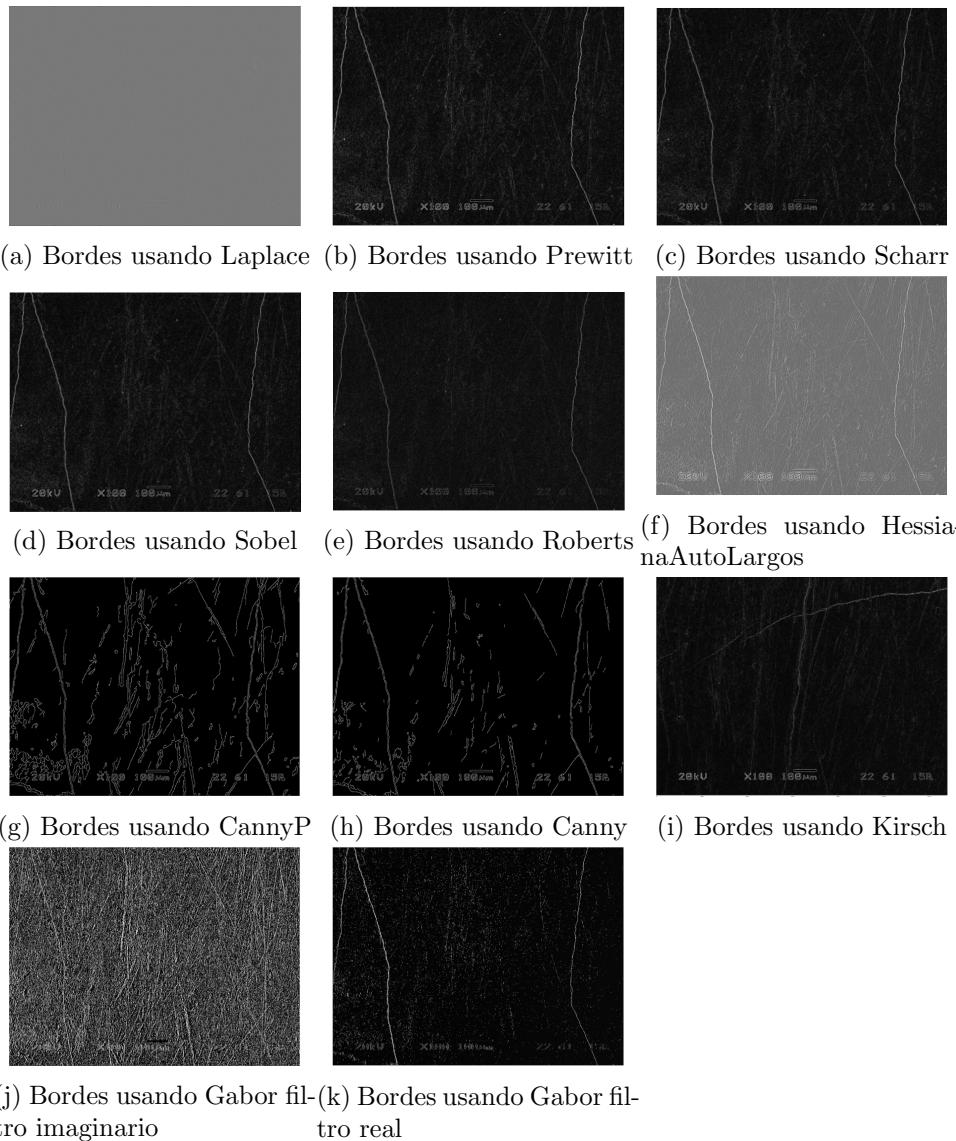


Figura F.1: Resumen visual filtros usados.

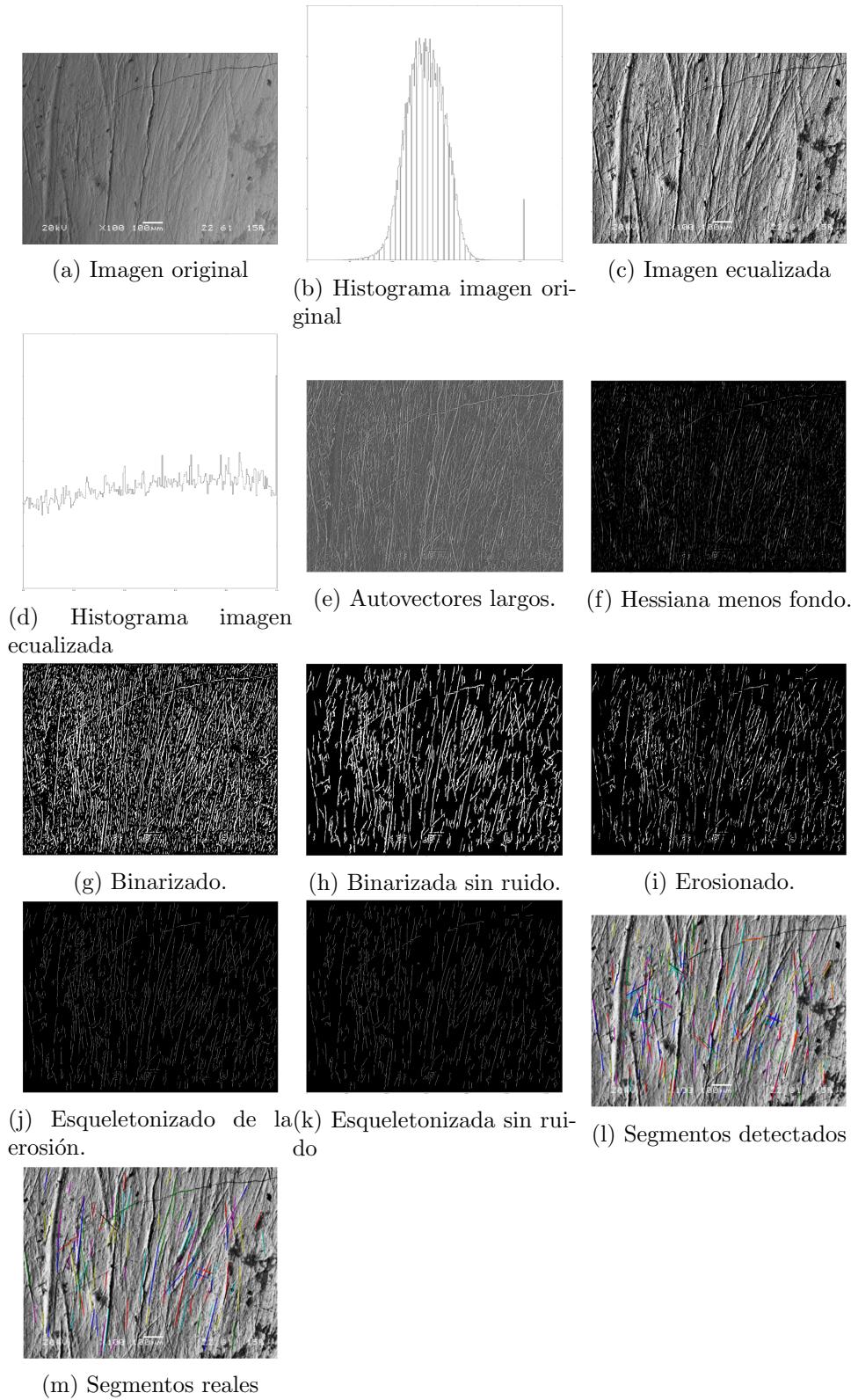


Figura F.2: Pasos del procesado.

Bibliografía

- [1] John Hunt. *Gang of Four Design Patterns*, pages 135–136. Springer International Publishing, Cham, 2013.
- [2] Johncostela. Scharr, 2016. [Kernel de Scharr].
- [3] Claudia Ruata Juan Palacio. *Scrum Manager Gestión de proyectos*. Safe Creative, Enero-2011.
- [4] Wikipedia. Canny, 2016. [Metodo de canny].
- [5] Wikipedia. Ciber ataque dyn, 2016. [Ciber ataque Dyn].
- [6] Wikipedia. Gabor, 2016. [Kernel de Gabor].
- [7] Wikipedia. Hessiana, 2016. [Matriz Hessiana].
- [8] Wikipedia. Kernels, 2016. [Tipos de Kernels].
- [9] Wikipedia. Kirsch, 2016. [Kernel de Kirsch].
- [10] Wikipedia. Laplace, 2016. [Kernel de Laplace].
- [11] Wikipedia. Patron de diseño comando, 2016. [Patron de diseño Comando].
- [12] Wikipedia. Patron de diseño fachada, 2016. [Patron de diseño Comando].
- [13] Wikipedia. Patron de diseño mediador, 2016. [Patron de diseño Mediador].
- [14] Wikipedia. Prewitt, 2016. [Kernel de Prewitt].
- [15] Wikipedia. Roberts, 2016. [Kernel de Roberts].
- [16] Wikipedia. Sobel, 2016. [Kernel de Sobel].