UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Análisis paleontológico de piezas dentales 2.0
Documentación Técnica

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

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

En este apartado se expondrá la metodología y planificación del proyecto adoptada así como las herramientas utilizadas.

La metodología de desarrollo ágil utilizada ha sido *SCRUM*. Cada semana o semana y media se realizaba una reunión con los tutores para definir y organizar las tareas a realizar durante el siguiente *sprint*.

Como sistema de control de versiones se ha usado *git* mediante un repositorio online en GitHub y su aplicación *GitHub Desktop* para Windows y la herramienta de git que ofrece *Eclipse* para entornos Linux.

Se ha utilizado también *ZenHub*, un plugin para navegadores que se integra en GitHub y permite la gestión del proyecto incluyendo además la posibilidad de ver distintos gráficos para comprobar como van los avances. Ha sido muy útil para organizar las tareas y poder seguir la metodología *SCRUM*.

A.2. Planificación temporal

A continuación se muestra el avance a lo largo de las iteraciones que forman el proyecto:

Sprint 1:

27/01/2017 - 03/02/2017 (figura A.1)

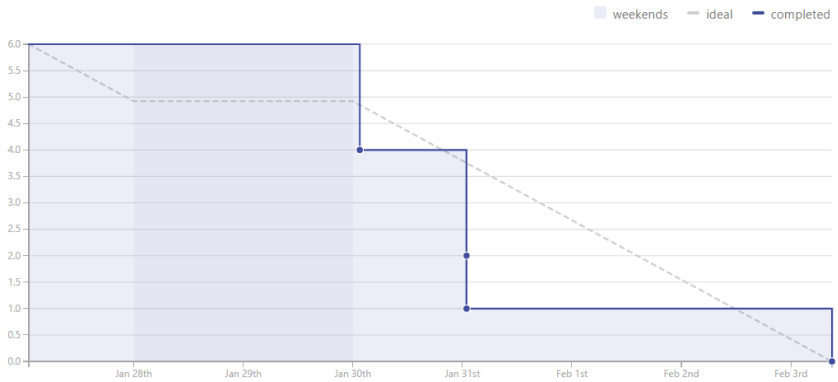


Figura A.1: Burndown Chart Sprint 1

- Instalación y preparación de maquinas virtuales.
- Recompilar y probar la aplicación de *stitching* que hay hecha en un sistema Linux de 64 bits.
- Adaptar la aplicación Java para el *stitching* en Linux de 64 bits.
- Recompilar *Stitching.exe* para sistemas Windows de 32 bits.

Este *sprint* me ha servido para familiarizarme con el proyecto, preparar su entorno y empezar a trabajar con la aplicación de *stitching* a revisar, centrándome en su recompilación para que pueda ser ejecutado en sistemas de distintas arquitecturas y distintos sistemas operativos.

Sprint 2:

04/02/2017 - 15/02/2017 (figura A.2)

Figura A.2: Burndown Chart Sprint 2

- Probar *stitching* Linux en equipos Linux de 32 bits.
- Recompilar *stitching* para Linux 32 bits.
- Adaptar la aplicación para que seleccione un *stitching* u otro dependiendo del sistema operativo y la arquitectura.
- Comenzar la memoria.

En este *sprint* se ha realizado la recompilación del *stitching* para distribuciones Linux de 32 bits. No ha sido posible utilizar esta versión en sistemas Linux de 64 bits, por lo que se ha decidido que el *stitching* para 64 bits forme parte de la aplicación también.

Sprint 3:

17/02/2017 - 01/03/2017 (figura A.3)

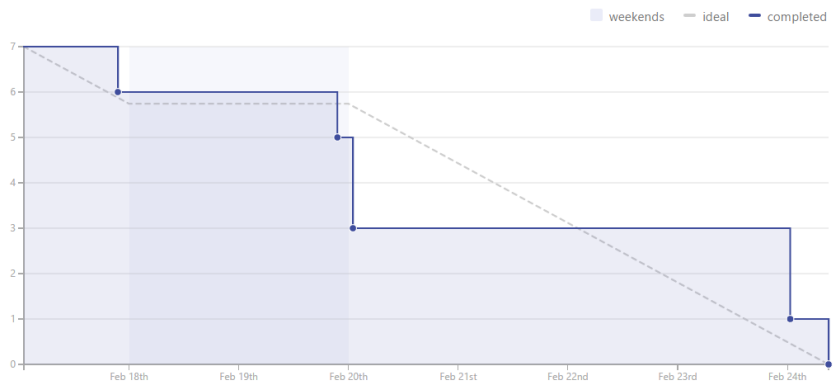


Figura A.3: Burndown Chart Sprint 3

- Comprobar el uso de carpetas temporales y permisos con Java.
- Crear una opción de seleccionar un carpeta para archivos temporales.
- Adaptar el código Java para extraer los fragmentos de las imágenes de rutas sin espacios en blanco.
- Adaptar el código Java para comprobar que los fragmentos no están en una ruta con espacios en blanco.
- Continuar la documentación.

La aplicación de *stitching* no funcionaba en determinadas ocasiones, se ha detectado que es debido a que la ruta de las imágenes contienen espacios en blanco. Para solucionarlo se ha decidido usar una carpeta temporal válida donde poder hacer el *stitching* sin problemas. Esta carpeta es la carpeta temporal del sistema por defecto o una que elija el usuario.

Sprint 4:

26/02/2017 - 02/03/2017 (figura A.4)

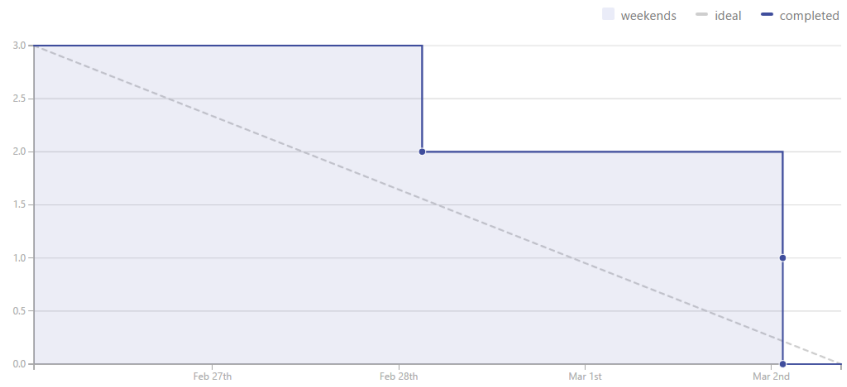


Figura A.4: Burndown Chart Sprint 4

- Solucionar el error ejecución de *stitching* desde el archivo ejecutable *.jar*.
- Realizar un test unitario sobre la clase *StitchingTemporaryUtil.java*.
- Utilizar distintos parámetros mediante *interact* de los *Jupyter notebooks* para comprobar la eficacia en la detección de bordes sobre las imágenes disponibles.

La aplicación Java, cuando se exportaba a un fichero *jar*, no podía hacer uso de la aplicación de *stitching* que tenía comprimida en dentro de la carpeta *rsc*. Se ha adaptado el código para que la aplicación Java pueda descomprimirla y usarla. También se ha comenzado la búsqueda de filtros más adecuados para la detección de perikymata usando los *Jupyter notebooks*.

Al finalizar este sprint se ha sacado una *release* en su versión 1.1 porque el cliente necesita hacer uso de la aplicación ahora que ya tiene solucionado el error del *stitching*.

Sprint 5:

03/03/2017 - 03/03/2017 (figura A.5)

Figura A.5: Burndown Chart Sprint 5

- Estudiar el funcionamiento de la matriz Hessiana y el filtro Frangi para su aplicación a la detección de bordes.
- Probar si directamente *ImageJ* para Java resulta más sencillo de aplicar que *Scikit-Image* para Python.
- Comenzar a evaluar como de buena es una propuesta filtrada.

ImageJ es una librería usada para el procesamiento de imagen que pensábamos emplear, aunque finalmente se va a utilizar *Scikit-Image* para Python porque es más sencillo de usar y aprender. Además, permite ver los resultados de manera rápida a través de los *notebooks*. Posteriormente, todo el código Python habrá que buscar alguna manera de exportarlo a algún ejecutable que pueda ser llamado desde Java.

El filtro de Frangi es con el que he comenzado y una posible forma de evaluar su eficacia es mediante la *distancia de Levenshtein*[1].

[1]Se obtuvo una implementación de aquí: https://es.wikipedia.org/wiki/Distancia_de_Levenshtein.

Sprint 6:

11/03/2017 - 17/03/2017 (figura A.6)

Figura A.6: Burndown Chart Sprint 6

- Comparar máscara y nuevo método con los datos del CSV.
- Realizar evaluación sobre imágenes filtradas (método antiguo Prewitt). Modificar el umbral para realizar las pruebas.
- Obtener recursivamente imágenes para pruebas.
- Crear módulo en Python para la nueva técnica de filtrado.
- Investigar las mejores opciones para exportar la aplicación de Python a ejecutables.
- Crear test con imágenes de distintos tamaños.
- Probar paso a paso ejecución del filtro nuevo con una imagen grande.

En este *sprint* se ha creado una máscara de usuario para comparar con los resultados del filtro de Frangi sobre distintas imágenes.

Los resultados eran prometedores hasta que se han realizado pruebas con una imagen de un diente, en tamaño más grande, y los resultado son notablemente peor que sobre un trozo pequeño de imagen.

Se ha buscado también, crear una función que permita optimizar los parámetros para obtener los mejores resultados sobre la *distancia de Levenshtein*.

Sprint 7:

18/03/2017 - 24/03/2017 (figura A.7)

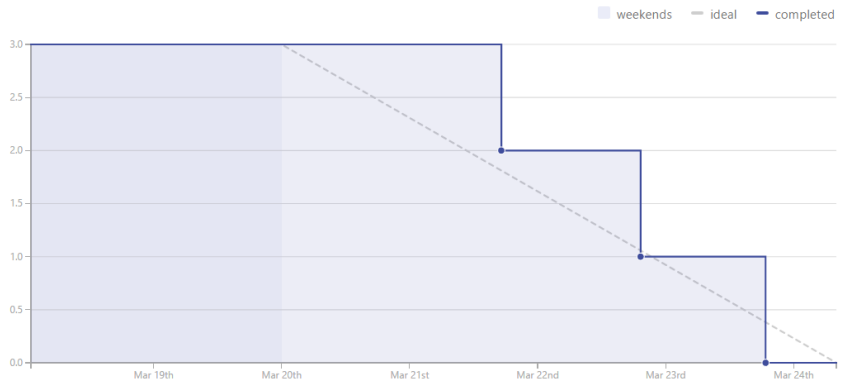


Figura A.7: Burndown Chart Sprint 7

- Incluir y ver que resultados da el filtro Prewitt con el nuevo método.
- Crear un notebook paso a paso para encontrar errores e incluir mejoras.
- Preparar un script para utilizar PyInstaler (instalar todo primero).

En esta iteración se han realizado más pruebas, añadiendo al método creado el filtro Prewitt, para ver como mejora o empeora el resultado sobre una imagen de diente a tamaño completo. También se ha decidido utilizar PyInstaller para exportar código Python a un ejecutable porque es el único de varias alternativas que lo crea con todas las dependencias comprimidas.

Sprint 8:

25/03/2017 - 31/03/2017 (figura A.8)

Figura A.8: Burndown Chart Sprint 8

- Probar si PyInstaller realiza correctamente las importaciones de los módulos en Python.
- Probar la binarización de imagen antes de aplicar Frangi.
- Probar la utilización de umbrales adaptativos para la imagen.
- Incluir en la función a optimizar el tratamiento de excepciones.
- Crear una máscara del diente 2.

En el pequeño script de prueba realizado con PyInstaler no ha habido problemas y funciona correctamente, pero al exportar todo el código Python, la aplicación no consigue funcionar. Queda pendiente para la próxima iteración.

Los umbrales adaptativos han funcionado correctamente y se incluirán en la versión final del procesado de imagen. Para próximos sprints se va a probar con la imagen del diente 2 porque la del diente 1 no es muy apta para los procesados, también se tiene incertidumbre porque solo hay dos imágenes de dientes y se desconoce cual es el caso normal que se encuentran en el Laboratorio de la Evolución Humana, que sea una imagen donde no se aprecian las perikymata o que se vean de forma aceptable.

Sprint 9:

01/04/2017 - 08/04/2017 (figura A.9)

Figura A.9: Burndown Chart Sprint 9

- Probar el *.exe* conseguido con PyInstaller con rutas absolutas.
- Probar el *.exe* conseguido con PyInstaller con imagen hardcodeada.
- Probar el procesado con el diente 2.
- Continuar la documentación.

Se han probado distintas opciones sobre el ejecutable en Windows que crea PyInstaler con el código en Python y se ha descubierto la necesidad de incluir unos archivos llamados *hooks-files* para importar módulos ocultos de Python, por lo que la exportación se realiza satisfactoriamente.

Con la imagen del diente 2 el procesado gana eficacia. Descartamos por tanto, basar el código en conseguir que sea eficaz en la imagen del diente 1 ya que se intuye que el proceso falla más porque en la imagen no se ven las perikymata, que porque el proceso esté mal desarrollado.

El proyecto sobre dietas [2] ya tiene implementado funciones para trabajar con líneas, se probarán en el próximo *sprint*.

Sprint 10:

10/04/2017 - 17/04/2017 (figura A.10)

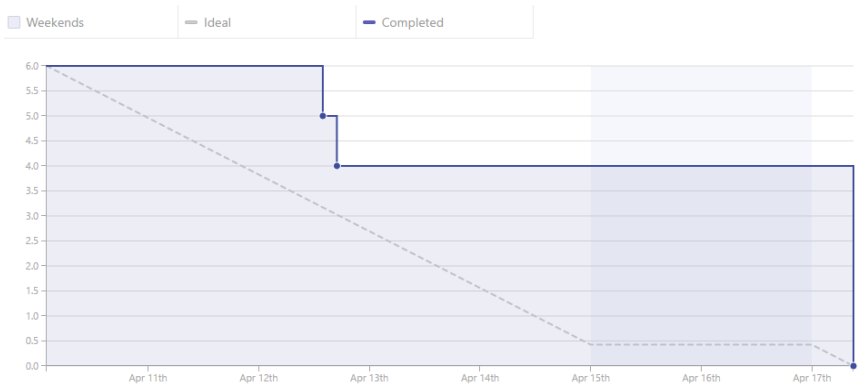


Figura A.10: Burndown Chart Sprint 10

- Estudiar el *notebook* proporcionado sobre extracción de líneas.
- Revisar el algoritmo *k_components* para adaptar o reutilizar.
- Realizar pruebas incluyendo el filtrado por direcciones.
- Implementar una función para cargar recursivamente imágenes de una carpeta.

En este *sprint* se ha revisado la clase de Python *Procesado.py* del trabajo de dietas [2] porque incluye un algoritmo de detección y unión de líneas usando teoría de grafos.

Finalmente no se han utilizado las funciones relacionadas con los grafos, sino que se ha modificado el uso de la función de la *Transformada de Hough* para operar en ciertos ángulos detectando líneas satisfactoriamente.

Investigando se han encontrado otros filtros como Sobel [7] y Kirsch [3], que se probarán más adelante, porque resaltan bordes en direcciones personalizadas.

Sprint 11:

18/04/2017 - 21/04/2017 (figura A.11)

Figura A.11: Burndown Chart Sprint 11

- Probar el filtro Sobel sobre la imagen completa del diente.
- Implementar una función para usar el operador de Kirsch.
- Probar el operador de Kirsch con diferentes filtrados.
- Añadir el filtrado por direcciones.
- Aplicar la operación de convolución manualmente con varios operadores.

Los resultados con el operador de Kirsch, basado en este artículo [3], son los más prometedores porque las perikymata no están siempre en la misma dirección y Kirsch permite elegir la orientación.

Sobel [7] queda descartado por ser demasiado simple (solo opera en los ejes x e y).

Sprint 12:

22/04/2017 - 28/04/2017 (figura A.12)

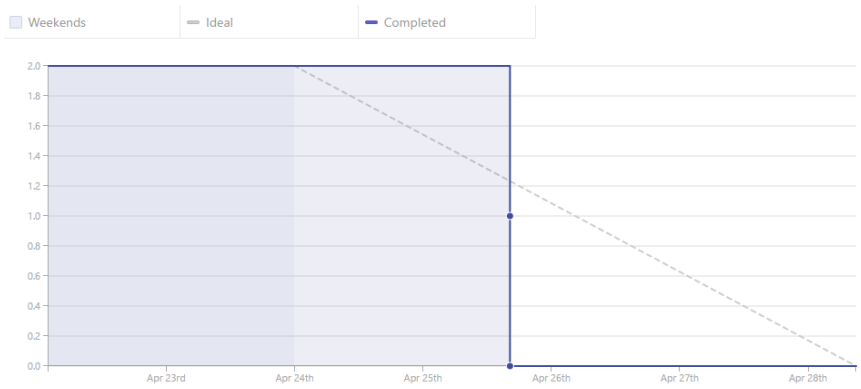


Figura A.12: Burndown Chart Sprint 12

- Probar el operador de Kirsch sobre el diente 1.
- Crear una función en Python que aplique el nuevo procesado de imagen basado en Kirsch.

Al probar Kirsch sobre el diente 1, se detectan mejor algunas perikymata pero queda confirmado que la imagen del diente 1 no es adecuada para procesar, porque no se aprecian bien las perikymata.

En la reunión se ha hablado de simplificar la vista de conteo de perikymata y así ahorrar trabajo al usuario en la aplicación Java.

Sprint 13:

29/04/2017 - 05/05/2017 (figura A.13)

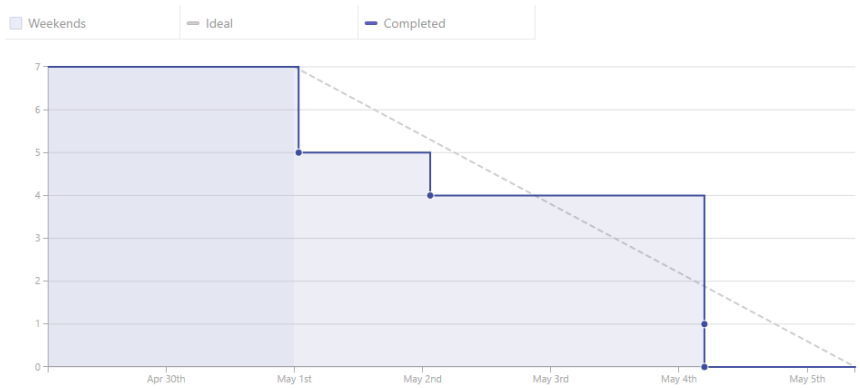


Figura A.13: Burndown Chart Sprint 13

- Crear una función para rotar la imagen los grados que indique el usuario.
- Modificar la aplicación Java para recortar la imagen del diente.
- Ajustar la aplicación Java para incluir un *BorderPane* con las nuevas funcionalidades.
- Ajustar el proceso de rotación para que la imagen se vea rotada.

En este *sprint* se ha creado una nueva vista para rotar la imagen y que el usuario deje las perikymata de manera vertical para poder aplicar el filtro Kirsch y que las detecte correctamente.

Sprint 14:

06/05/2017 - 11/05/2017 (figura A.14)

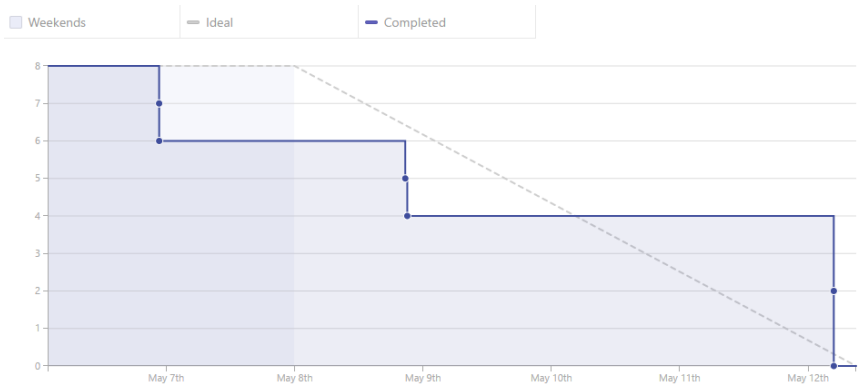


Figura A.14: Burndown Chart Sprint 14

- Estudiar la unión de la aplicación Java con la aplicación Python vía sockets.
- Crear un botón para restablecer la vista de una imagen recortada.
- Pasar la funcionalidad de medida de la escala desde la vista de conteo de perikymata a la vista de rotación de la imagen.
- Poner un fondo negro para el recortado de la imagen.
- Poner una imagen al botón de recortado.
- Limitar la rotación que se puede aplicar.

En esta iteración se ha mejorado la vista y la funcionalidad de rotación y recorte de la imagen.

Sprint 15:

12/05/2017 - 17/05/2017 (figura A.15)

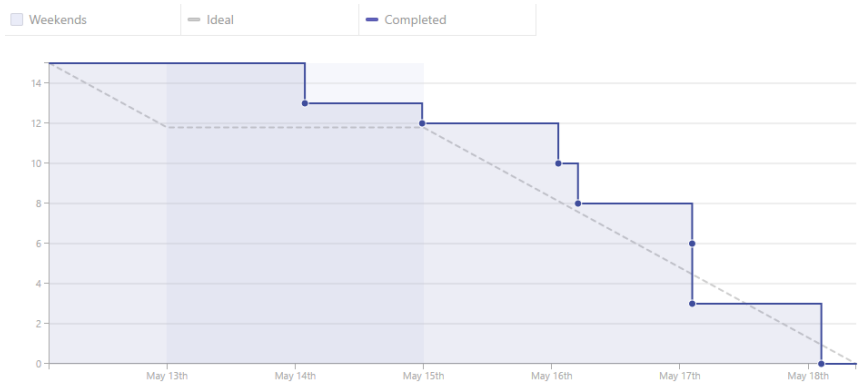


Figura A.15: Burndown Chart Sprint 15

- Incluir la creación de una carpeta *Crop_Image* para guardar la imagen recortada.
- Crear un prototipo de conexión Java-Python a través de sockets.
- Definir un protocolo de comunicación entre Java y Python.
- Implementar una función de *handshake* entre Java y Python.
- Estudiar documento de reconstrucción de la corona del diente.
- Limpiar la vista de conteo de perikymata.
- Incluir migas de pan a lo largo de la aplicación.

Se ha terminado la funcionalidad de rotación y recortado de la imagen y se ha creado un prototipo de conexión por sockets en Python y Java donde Java es el servidor y Python el cliente. Se ha hablado en la que reunión que, para la siguiente iteración, queda pendiente cambiarlo para que el servidor sea Python.

Se han incluido las migas de pan en la aplicación para que el usuario sepa en que etapa está y para que sea más fácil moverse por la aplicación.

La reconstrucción de la corona permitirá calcular mejor las perikymata, se prevé una reunión con el personal del Laboratorio de la Evolución Humana para que nos expliquen el proceso de reconstrucción y dependiendo de su complejidad incluirlo en este proyecto o no.

Sprint 16:

18/05/2017 - 29/05/2017 (figura A.16)

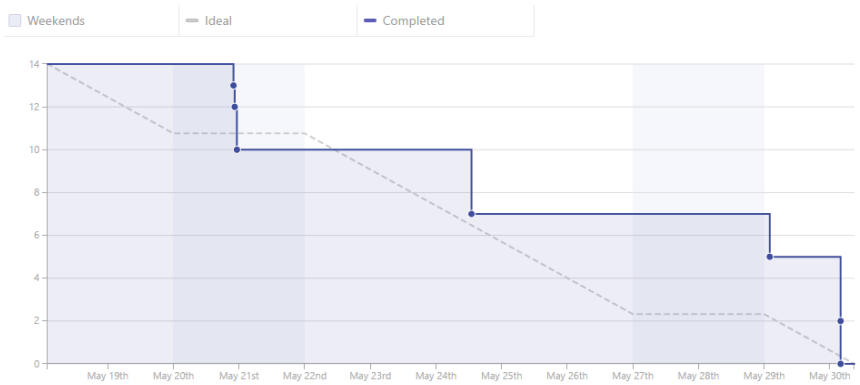


Figura A.16: Burndown Chart Sprint 16

- Implementar una función en Java para detectar píxeles rojos.
- Cambiar el servidor y cliente en Java y Python.
- Preparar el servidor en Python para la conexión con Java.
- Prepara la aplicación Java para la conexión con el servidor de Python.
- Arreglar el funcionamiento de las migas de pan.
- Arreglar el error detectado en la etapa de selección de imágenes.
- Aumentar la fuente y poner en negrita las migas de pan dependiendo donde nos encontremos.

En este sprint se han solucionado errores con las migas de pan y se ha localizado el origen y solucionado el error reportado que sucedía a veces al abrir la aplicación; en la carga inicial de imágenes, carga todos los elementos de la carpeta sean o no imágenes, lo cual da error al hacer el *stitching*.

Se han modificado la aplicaciones de Java y Python y ya se comunican correctamente a través de sockets.

La detección de píxeles rojos se hace usando el modelo RGB [6] y en la reunión se ha hablado de usar el modelo HSI [5] que permite ser más preciso en cuando al rojo.

Sprint 17:

30/05/2017 - 06/06/2017 (figura A.17)

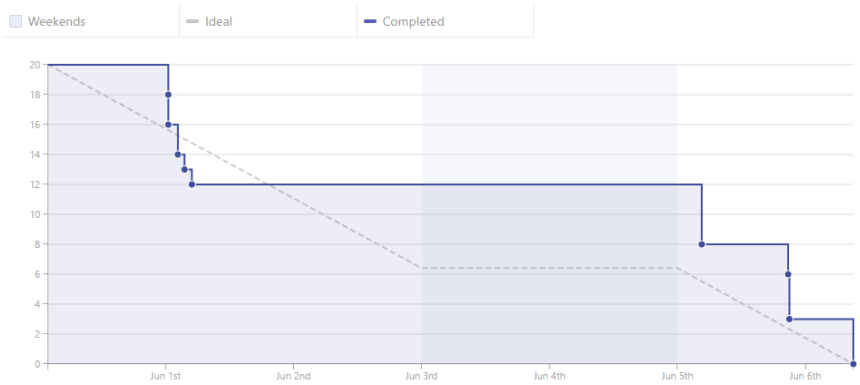


Figura A.17: Burndown Chart Sprint 17

- Incluir opciones avanzadas en la aplicación Java para aplicar filtros.
- Redefinir el protocolo de comunicación en la parte del servidor Python.
- Utilizar el color HSI para extraer el color rojo de los píxeles, en vez de usar el RGB.
- Modificar procesado para guardar la imagen original con líneas detectadas superpuestas.
- Modificar la interfaz Java para alternar la imagen filtrada y la imagen filtrada con líneas superpuestas.
- Eliminar los falsos positivos al detectar perikymata.
- Separarlas coordenadas en la exportación de datos a *csv*.
- Añadir el icono de zoom e icono de dibuja línea en la interfaz de conteo de perikymata.
- Escribir la introducción y objetivos de la memoria.

En este *sprint* se han mejorado aspectos generales de la aplicación y del guardado de las imágenes filtradas. Además, se ha continuado con la memoria.

Finalmente se ha usado el espacio de color HSV[2] que sigue permitiéndonos conocer el matiz del color.

[2]Llamado HSB en Java.

También se han encontrado errores en la interfaz de la aplicación al pasar a pantallas de tamaño 1366x768. Este error es debido a que el desarrollo de la aplicación ha sido en una pantalla de tamaño 1920x1080.

Sprint 18:

07/06/2017 - 15/06/2017

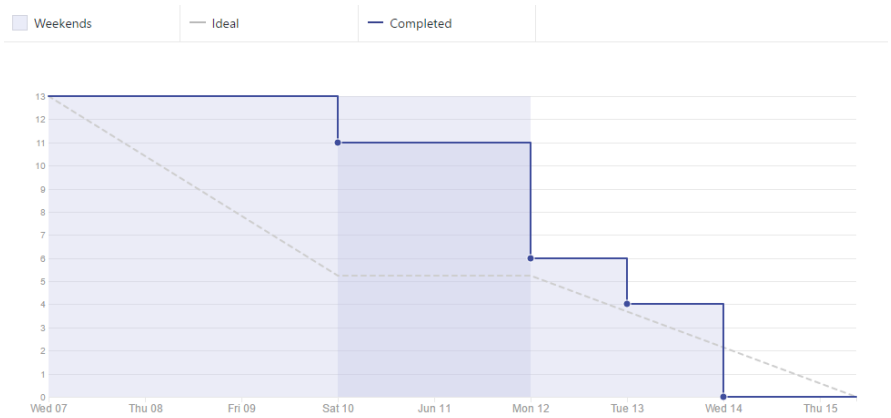


Figura A.18: Burndown Chart Sprint 18

- Revisar la introducción y objetivos.
- Arreglar el redimensionado de la aplicación.
- Arreglar la petición de opciones avanzadas en el servidor Python.
- Documentar los conceptos teóricos.
- Implementar el correcto cerrado del servidor desde Java.

En esta iteración se han solucionado errores con las vistas de la aplicación y con la petición de filtrado avanzado.

Para el próximo *sprint* se adecuará la aplicación para su funcionamiento en entornos Linux y se lanzará una pre-release.

Sprint 19:

16/06/2017 - 22/06/2017 (figura A.19)

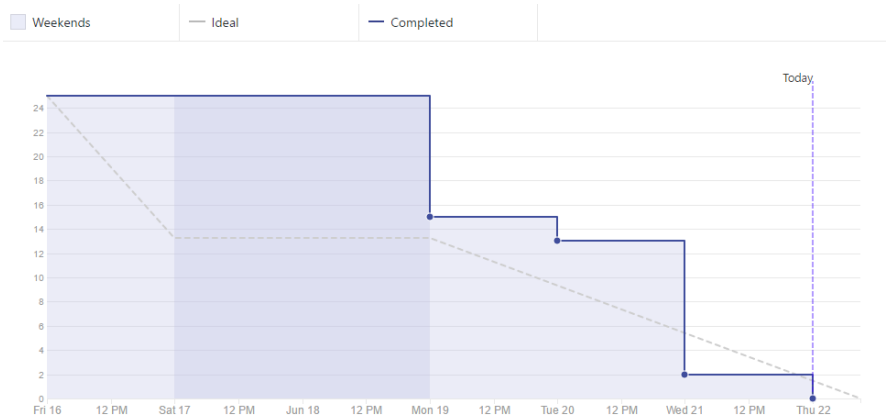


Figura A.19: Burndown Chart Sprint 19

- Volver a buscar formas de exportar el servidor de Python.
- Documentar Técnicas y Herramientas.
- Revisar el *combobox* de selección de kernel en opciones avanzadas de filtrado.
- Mejorar la eficiencia en el guardado de las imágenes filtradas.
- Crear el manual de instalación y lanzar una release prototipo.
- Realizar arreglos en la aplicación para poder instalarla adecuadamente.
- Corregir las vistas para entornos Linux.
- Corregir el cargado de imágenes en Linux.

Este *sprint* se ha caracterizado por preparar la aplicación para su instalación, probarla y adaptarla para entornos Linux.

Se han encontrado problemas a la hora de ver las vistas adecuadamente usando Ubuntu y la aplicación a veces se vuelve inestable, pero es usable.

Para la última iteración se terminará la memoria, se organizarán las pruebas realizadas en los *Jupyter notebooks* y se finalizará el desarrollo de la aplicación.

Sprint 20:

23/06/2017 - 02/07/2017 (figura A.20)

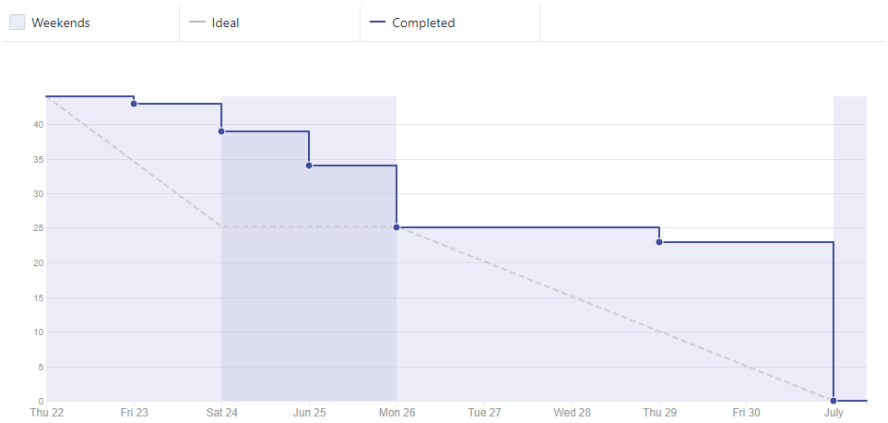


Figura A.20: Burndow Chart Sprint 20

Este último *sprint* pone fin al proyecto, las tareas más relevantes han sido:

- Finalizar toda la documentación del proyecto.
- Dar los últimos retoques a la aplicación.
- Organizar el contenido del proyecto.
- Solucionar los últimos errores encontrados.

A.3. Estudio de viabilidad

En esta sección se hablará del apartado económico que concierne al desarrollo del proyecto. También trataremos el tema de su viabilidad legal.

Viabilidad económica

A continuación iremos nombrando y justificando todos los gastos que hubiera tenido el proyecto si no hubiera formado parte del plan de estudios del grado y se aplicase en un entorno real.

En la descripción del TFG se estima que el trabajo del alumno consiste en 300 horas, pero esto está alejado de la realidad por lo que usamos la herramienta ZenHub y a cada *issue* le añadimos un valor numérico para estimar el tiempo.

Cada unidad de tiempo añadida ha sido tomada con un valor real de 2 horas y tenemos 230 puntos por lo que la estimación de las horas invertidas quedaría en 460 horas.

En el proyecto anterior [1] se estimó la hora de trabajo del programador en 15€ por lo que el coste del proyecto en mano de obra habría sido de:

$$15€/\text{h} \times 460\text{h} = 6900€$$

Con este resultado tenemos que tener en cuenta el coste de la seguridad social que podemos obtener desde su página oficial[3] y que nos dice que han de tenerse en cuenta los siguientes impuestos:

- Contingencias comunes: 23,60 %.
- Contrato duración determinada Tiempo parcial[4]: 6,70 %.
- Fondo de Garantía Salarial: 0,20 %.
- Formación Profesional: 0,60 %.

Esto nos da un total de un 31,1 % en impuestos lo cual hace que el gasto destinado a la seguridad social sea de:

$$6900€ \times 31,1\,\% = 2139€$$

[3] Página de la seguridad Social: https://goo.gl/zRVeq9.
[4] Tomaremos que el proyecto tiene la duración de 5 meses a tiempo parcial.

También tenemos que valorar el coste de las herramientas hardware y software utilizadas en este proyecto de las cuales han tenido costes las siguientes:

- Equipo de desarrollo (ordenador portátil): 730€.
- Periféricos usados (ratón, teclado y pantalla): 250€.
- Licencia Windows para usar en la máquina virtual: 121€[5] .

El equipo de desarrollo y los periféricos hacen un total de 980€ y según la agencia tributaria[6] los equipos para procesos de información tienen una vida útil de 8 años y este proyecto ha tenido una duración de 5 meses, lo cual nos deja unos costes de amortización de:

$$\frac{980€}{12 \text{ Meses} \times 8 \text{ Años}} \times 5 \text{ Meses} = 51,04€$$

La licencia para la máquina virtual se ha usado durante los 5 meses de duración del proyecto pues continuamente se han tenido que probar los cambios realizados. Se estima su vida útil en 6 años por lo que sus costes de amortización serían los siguientes:

$$\frac{121€}{12 \text{ Meses} \times 6 \text{ Años}} \times 5 \text{ Meses} = 8,40€$$

Esto nos deja un coste total en las herramientas utilizadas de 59.44€.

Con todos los costes mencionados, el desglose e importe final podemos observarlo en la tabla A.1

Costes	Importe
Personal	6900€
Seguridad Social	2139€
Herramientas	59,44€
Total	9098,44€

Tabla A.1: Costes del proyecto

[5]Se aplica el descuento a estudiantes. Fuente: https://goo.gl/PwrrcP.
[6]Tablas de amortización 2017: https://goo.gl/UTCRK9.

Viabilidad legal

La versión anterior del proyecto [1] usa una licencia GPL [4] lo cual obliga a que esta también lo tenga y que la aplicación siga siendo software libre. Las nuevas partes añadidas como el servidor en Python y nuevos ficheros se les ha dotado también de una licencia GPL para mantener la coherencia de en toda la aplicación.

En este proyecto se ha prescindido de algunas librerías que usaba la versión anterior pero se han utilizado otras nuevas. También se han seguido usando las mismas imágenes e iconos que en la versión anterior además de añadir otros.

En la tabla A.2 podemos ver las licencias de las herramienta utilizadas.

Herramienta	Licencia
Java JavaFX	Oracle Binary Code License
Python 3	PSF License
Scikit-Image	BSD License
Numpy	BSD License
OpenCV	BSD License
Eclipse	Eclipse Public License
Visual Studio	Microsoft Software License
PyCharm	Apache 2.0 License
PyInstaller	GPL License
VirtualBox	GPL License
White Square icon	Public Domain
DesignContest Icons	CC Attribution 4.0
IconsMind Icons	LinkWare

Tabla A.2: Herramientas utilizadas y sus licencias

Apéndice B

Especificación de Requisitos

B.1. Introducción

En esta sección se hablará de los requisitos y casos de uso que han dado la hoja ruta a seguir para el desarrollo de la aplicación.

B.2. Objetivos generales

Los objetivos generales que persigue la aplicación, nos vienen dados desde el proyecto anterior [1]. Son los siguientes:

- La aplicación debe poder unir las de imágenes de fragmentos del diente para ofrecer al usuario una imagen completa.
- Se debe poder aplicar un filtro o filtros para resaltar las perikymata de la imagen completa de la pieza dental.
- Se debe poder delimitar la corona del diente en la imagen y dividirla en deciles sobre los que se dibujará una línea en la que se marcarán las perikymata detectadas.
- Todos los datos generados podrán ser guardados de forma persistente.
- Se deben corregir errores y mejorar cada una de las etapas mencionadas en la medida de lo posible, a fin de ser más eficaz en la detección de perikymata.

B.3. Catalogo de requisitos

Los requisitos que debe satisfacer la aplicación son:

- Se podrá introducir una imagen completa de una pieza dental. En caso de ser imágenes de fragmentos, se unirán formando la imagen completa.
- Se podrá preparar la imagen completa para su uso a lo largo de la aplicación.
- Se podrá aplicar un filtro a la imagen para resaltar las perikymata.
- Se detectarán y marcarán la perikymata en la imagen filtrada de manera automática, aunque el usuario podrá intervenir para corregir posibles errores.
- La aplicación guardará de manera automática las imágenes generadas y el usuario podrá exportar los datos generados.
- La aplicación deberá proporcionar las funcionalidades básicas de cualquier aplicación.

B.4. Especificación de requisitos

En este apartado se indicarán las especificación de los requisitos a cumplir.

- RF-1: Introducir la imagen o imágenes de la pieza dental que usará la aplicación para conseguir una imagen completa.
  - RF-1.1: Indicar un conjunto de imágenes y unirlas (*stitching*).
    - RF-1.1.1: Indicar un conjunto de imágenes válidas para unir que representan los fragmentos del diente.
    - RF-1.1.2: Unir la imágenes para obtener una imagen dental completa.
  - RF-1.2: Indicar una imagen que represente la pieza dental completa.
- RF-2: Preparar la imagen y datos para las etapas posteriores.
  - RF-2.1: Rotar y recortar la imagen para orientar las perikymata de la forma más vertical posible y delimitar la zona de acción.
  - RF-2.2: Indicar la escala a la que se encuentra la imagen de la pieza dental.

- RF-3: Filtrar la imagen del diente y llevar a cabo la detección de perikymata.
  - RF-3.1: Delimitar, de manera automática, los deciles de la corona dental.
  - RF-3.2: Filtrar la imagen del diente gracias a un modo por defecto.
  - RF-3.3: Filtrar la imagen del diente gracias a un modo avanzado con distintos parámetros de filtrado.
  - RF-3.4: Dibujar una línea y marcar perikymata automáticamente.
    - RF-3.4.1: Dibujar una línea que atraviese las perikymata.
    - RF-3.4.2: Automarcar todas las perikymata que sean posibles.
  - RF-3.5: Añadir o eliminar perikymata sobre la imagen filtrada.
- RF-4: Guardar todos los datos generados[1], incluidas las imágenes que intervienen en el proceso.
  - RF-4.1: Guardar las imágenes que aporte el usuario y las que genere la aplicación en las carpetas del proyecto.
  - RF-4.2: Almacenar, en formato *csv*, los resultados generados sobre la distancia entre cada perikyma[2] y su localización en los deciles.
- RF-5: Proporcionar acceso en todo momento a funcionalidades básicas de la aplicación.
  - RF-5.1: Hacer uso de las *migas de pan*.
    - RF-5.1.1: Ver la etapa actual de la aplicación en la que se encuentre el usuario.
    - RF-5.1.2: Retroceder a etapas anteriores de la aplicación.
  - RF-5.2: Abrir, cerrar y guardar el proyecto actual en cualquier momento.
  - RF-5.3: Indicar la carpeta temporal que se usará en la operación de *stitching*, sino, se usará la carpeta por defecto del sistema.
  - RF-5.4: Ayudar al usuario con explicaciones de cada componente por medio de *tooltips*.
  - RF-5.5: Permitir interactuar al usuario con terceras aplicaciones que utilice la propia aplicación.

[1] Los cálculos realizados no se incluyen.
[2] *Perikyma* es el singular de *perikymata*.

Como requisitos no funcionales encontraremos los siguientes:

- RNF-1: Se corregirán los errores reportados y los nuevos que surjan, en la medida de lo posible.
- RNF-2: Se tratará de hacer más sencilla e intuitiva la aplicación.
- RNF-3: Se investigarán nuevas formas de implementación de cada etapa de la aplicación.

Diagrama de casos de uso

En la figura B.1 podemos ver el diagrama general de casos de uso.

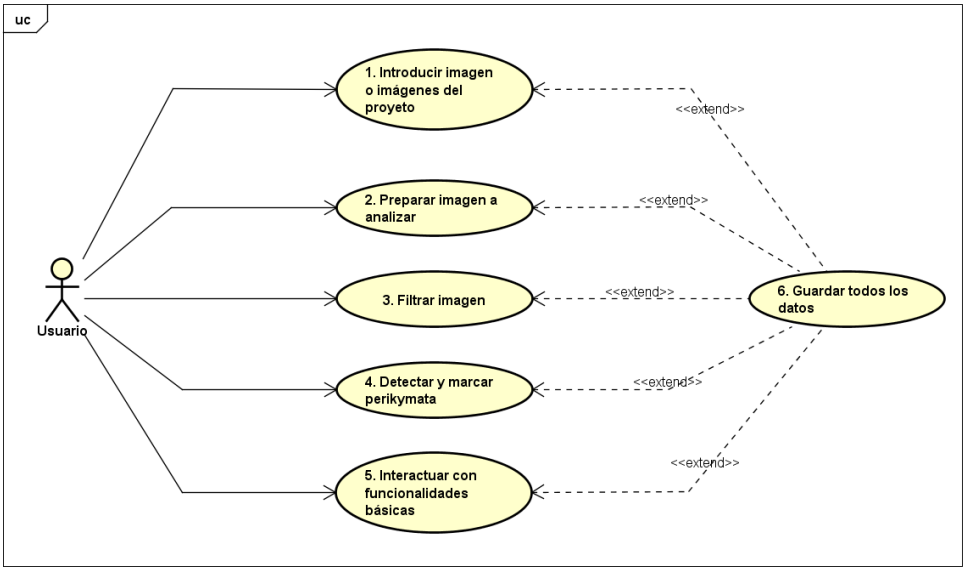


Figura B.1: Diagrama general de casos de uso

En las figuras B.2, B.3, B.4, B.5, B.6 y B.7 podemos observar la especificación de cada caso de uso.

CU1 – Introducir imágenes en la aplicación		
Versión	1.0	
Autor	Sergio Chico Carrancio	
Requisitos	RF-1, RF1.1, RF1.2	
Descripción	Se añade la imagen del proyecto, bien una imagen completa del diente o imágenes de fragmentos que se unen automáticamente	
Precondiciones	Si hay imágenes de fragmentos, deben estar con la misma orientación	
Secuencia normal	Paso	Acción
	A1	Se pulsa un botón para seleccionar imágenes de fragmentos.
	A2	Se seleccionan las imágenes de fragmentos.
	A3	Se pulsa un botón para unir las imágenes.
	B1	Se pulsa un botón para seleccionar una imagen completa de la pieza dental.
	B2	Se selecciona la pieza dental
Postcondiciones	Se carga una imagen completa del diente. Se guardan las imágenes utilizadas.	
Excepciones	Paso	Acción
	A3	No se puede unir la imagen, se informa al usuario en un mensaje.
Rendimiento		
Frecuencia	Baja	
Importancia	Alta	
Urgencia	Alta	
Comentarios	Esta operación es absolutamente necesaria para obtener la imagen del diente con la que trabajar	

Figura B.2: Caso de uso 1

CU2 – Preparar la imagen a analizar		
Versión	2.0	
Autor	Andrés Miguel Terán	
Requisitos	RF-2, RF-2.1, RF-2.2, RF-4, RF-4.1, RF-4.2	
Descripción	La imagen completa es rotada y recortada para delimitar la zona de acción. Además, se debe introducir la escala para la medida de la imagen.	
Precondiciones	Imagen completa de la pieza dental cargada.	
Secuencia normal	Paso	Acción
	1	Se pulsa un botón para seleccionar la zona a recortar.
	2	Se pulsa un botón para recortar la imagen.
	3	Se pulsa un botón dibujar la medida con una línea.
	4	Se pulsa un botón para introducir la medida que se debe tener en cuenta (mm, µm, etc...).
Postcondiciones	Se carga la imagen preparada del diente. Se guarda la imagen preparada.	
Excepciones	Paso	Acción
	4	Si no se introduce la medida, se informa al usuario en un mensaje.
Rendimiento		
Frecuencia	Baja	
Importancia	Media	
Urgencia	Media	
Comentarios	Esta operación es necesaria si la imagen no está ya preparada.	

Figura B.3: Caso de uso 2

CU3 – Filtrar la imagen		
Versión	2.0	
Autor	Andrés Miguel Terán	
Requisitos	RF-3, RF-3.1, RF-3.2, RF-3.3, RF-4, RF-4.1	
Descripción	La imagen preparada se filtra para resaltar las perikymata.	
Precondiciones	Imagen preparada de la pieza dental cargada. La aplicación ha delimitado los deciles automáticamente.	
Secuencia normal	Paso	Acción
	A1	Se pulsa un botón para filtrar la imagen con el modo por defecto.
	B1	Se pulsa un botón y se ajustan parámetros para filtrar en modo avanzado.
Postcondiciones	Se cargan las imágenes filtradas y se guardan.	
Excepciones	Paso	Acción
	A1/B1	El servidor que filtra la imagen no se encuentra disponible, se muestra un mensaje al usuario.
Rendimiento	El tiempo en realizar el filtrado depende de la potencia del equipo donde se ejecute la aplicación.	
Frecuencia	Alta	
Importancia	Alta	
Urgencia	Alta	
Comentarios	Esta es una de las etapas más importantes de la aplicación.	

Figura B.4: Caso de uso 3

CU4 – Detectar y marcar perikymata		
Versión	2.0	
Autor	Andrés Miguel Terán	
Requisitos	RF-3, RF-3.4, RF-3.4.1, RF-3.4.2, RF-3.5	
Descripción	Se dibuja una línea en la imagen filtrada y se marcan las perikymata.	
Precondiciones	Imagen filtrada cargada.	
Secuencia normal	Paso	Acción
	1	Se dibuja una línea que atraviese la zona que mejor se vean las perikymata en la imagen filtrada.
	2	Se pulsa un botón y la aplicación marca las perikymata que detecte.
	3	Se pulsa un botón para añadir o quitar puntos que sí son perikymata.
Postcondiciones	Perikymata detectadas sobre la imagen.	
Excepciones	Paso	Acción
	1	Si la línea dibujada excede los límites de la imagen, se muestra un mensaje al usuario.
Rendimiento	El marcado de las perikymata debe ser rápido.	
Frecuencia	Alta	
Importancia	Alta	
Urgencia	Alta	
Comentarios		

Figura B.5: Caso de uso 4

CU5 – Interactuar con funcionalidades básicas		
Versión	2.0	
Autor	Andrés Miguel Terán	
Requisitos	RF-5, RF-5.1, RF-5.1.1, RF-5.1.2, RF-5.2, RF-5.3, RF-5.4, RF-5.5	
Descripción	La aplicación permite interactuar con funcionalidades básicas de guardado, cerrado y configuración.	
Precondiciones	Tener un proyecto creado y abierto en la aplicación.	
Secuencia normal	Paso	Acción
	1	Se pulsa un botón y se guardan los datos de la aplicación.
	2	Se pulsa un botón y se cierra la aplicación.
	3	Se pulsa un botón y se abre un nuevo proyecto
	4	Se pulsa un botón y se elige la carpeta para los archivos temporales.
	5	Se mantiene el ratón sobre un componente y un *tooltip* explica su función.
Postcondiciones	Se realizan cada una de las acciones descritas.	
Excepciones	Paso	Acción
Rendimiento		
Frecuencia	Baja	
Importancia	Baja	
Urgencia	Baja	
Comentarios		

Figura B.6: Caso de uso 5

CU6 – Guardar todos los datos		
Versión	1.0	
Autor	Sergio Chico Carrancio	
Requisitos	RF-4, RF-4.1, RF-4.2	
Descripción	Se guardan las imágenes generadas y se pueden exportar los datos del marcado de perikymata	
Precondiciones	Tener un proyecto creado, abierto, con imagen filtrada y perikymata marcadas.	
Secuencia normal	Paso	Acción
	1	Se pulsa un botón y se exportan a *csv* los datos obtenidos del marcado.
	2	A lo largo de cada etapa, se guardan en disco las imágenes producidas.
Postcondiciones	En cada carpeta de proyecto tenemos las imágenes generadas y en otra carpeta un *csv* con los datos de perikymata extraídos.	
Excepciones	Paso	Acción
Rendimiento		
Frecuencia	Alta	
Importancia	Alta	
Urgencia	Alta	
Comentarios		

Figura B.7: Caso de uso 6

Apéndice C

Especificación de diseño

C.1. Introducción

En esta parte de los anexos se mostrará el diseño de la aplicación Java y el servidor desarrollado en Python encargado de darle el servicio de filtrado. Para ello, se incluirán elementos representativos como los diagramas de clases y de secuencia.

Para su elaboración se ha hecho uso del programa de modelado Astah y de Object Aid, un plugin para Eclipse que permite realizar diagramas de clases y de secuencia.

Las clases y operaciones desarrolladas son muy extensas para mostrarlas detalladamente, por los que se omitirán muchos detalles para ver lo esencial.

C.2. Diseño de datos

Para explicar el diseño de los datos, comenzaremos explicando la estructura exterior de la aplicación Java, que podemos ver en la figura C.1. En ella encontraremos los siguientes elementos:

- Paquete *es.ubu.lsi.perikymata*: contiene la aplicación principal, el modelo, las vistas y todas las clases utilizadas para desarrollar la funcionalidad de la aplicación.
- Paquete *rsc*: contiene las imágenes que usa la aplicación y los ejecutables para realizar la operación de *stitching* en múltiples plataformas.
- Paquete *test.es.ubu.lsi.perikymata*: contiene las pruebas unitarias de la aplicación.

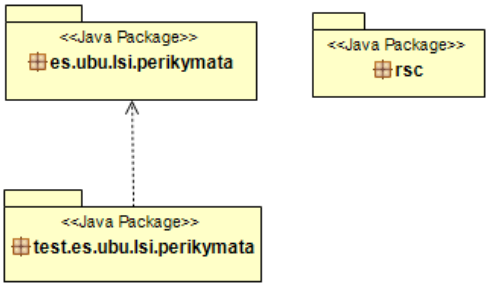


Figura C.1: Estructura exterior

Nos centraremos en el paquete que contiene la aplicación. En el encontraremos los paquetes y clases que se han continuado desarrollando según el patrón Modelo-Vista-Controlador (MVC) que heredamos de la aplicación anterior [1]. Los elementos que contiene son (figura C.2):

- Paquete *modelo*: están contenidas las clases necesarias para hacer persistentes los datos del proyecto.
- Paquete *vista*: contiene los archivos FXML que forman la interfaz de la aplicación y las clases que hacen de controlador para cada uno de ellos.
- Paquete *util*: Contiene las clases de utilidad para distintas operaciones.
- *MainApp.java*: Esta clase Java es el controlador principal de la aplicación. Permite lanzarla y se relaciona con los controladores de cada vista proporcionándoles acceso a elementos comunes y necesarios.

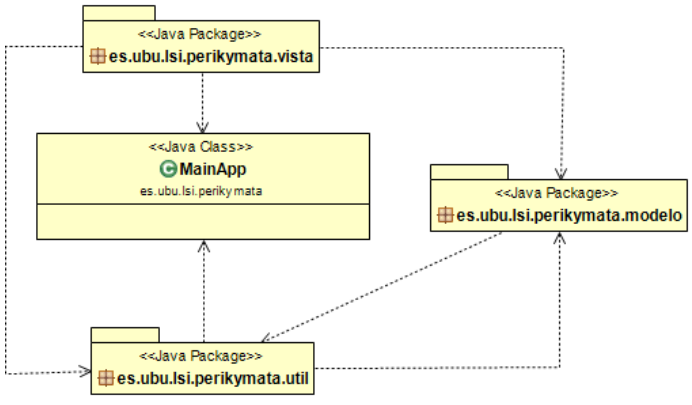


Figura C.2: Modelo-Vista-Controlador

En la figura C.3 se muestran todas las clases y relaciones más importantes que intervienen en la aplicación. Se encuentran agrupadas por colores para facilitar su entendimiento. las agrupaciones se corresponden con los paquetes MCV que hemos descrito anteriormente.

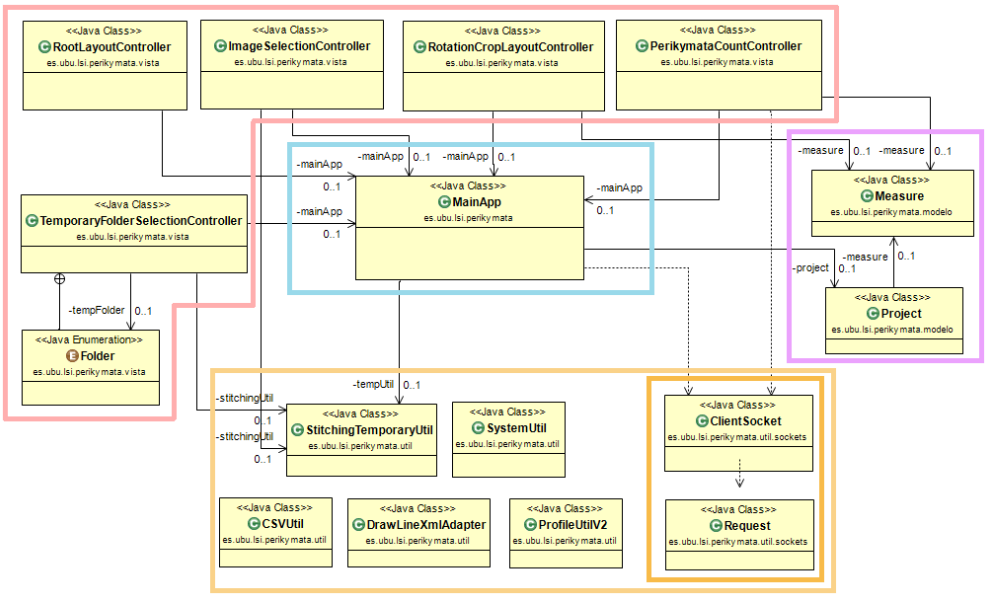


Figura C.3: Diagrama general de clases

- La agrupación en color rojo contiene las clases del paquete *vista* y son los controladores de los archivos FXML. Cada una de estas clases tiene como atributos los componentes definidos en los archivos FXML. Además, los métodos contienen el comportamiento de cada componente; estos son utilizados cuando el usuario interactúa con la aplicación (pulsar un botón, mover el ratón, desencadenar eventos, etc).
- La agrupación azul muestra el controlador principal *MainApp.java* que lanza aplicación principal entre otras cosas, como comentábamos anteriormente.
- La agrupación en color morado se corresponde con el paquete *modelo*. Las clases *Project.java* y *Measure.java* son las utilizadas para la persistencia de los datos de la aplicación.
- La agrupación en color naranja se corresponde con el paquete *util* y contiene clases relacionadas con la exportación a *csv*, la operación de *stitching* y el cálculo de perikymata . Encontramos además, un subpaquete *sockets* que contiene las clases usadas para realizar las peticiones al servidor de Python.

La aplicación de Python tiene únicamente dos ficheros contenidos en la carpeta *src*, uno encargado de lanzar el servidor y otro que contiene el procedimiento necesario para filtrar las imágenes, como vemos en la figura C.4.

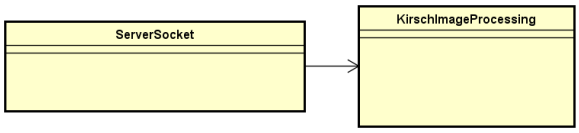


Figura C.4: Clases del servidor Python

C.3. Diseño procedimental

En este apartado comentaremos, de forma sencilla, el diseño de los procedimientos más importantes que se llevan a cabo en la aplicación. Para ello nos apoyaremos en los diagramas de secuencia que veremos a continuación.

Stitching

En esta fase el usuario selecciona (mediante un botón) una imagen completa de un diente o varias imágenes que se unen en una usando el ejecutable de *stitching* adecuado al sistema operativo y la arquitectura. Estos ejecutables los encontramos en el paquete *rsc.stitching.bin*. La aplicación se encarga copiar las imágenes a la carpeta temporal definida y de crear los argumentos para el ejecutable con la ruta de las imágenes a juntar. Como resultado obtenemos una imagen completa del diente cargada en la aplicación.

Podemos observar este procedimiento en la figura C.5.

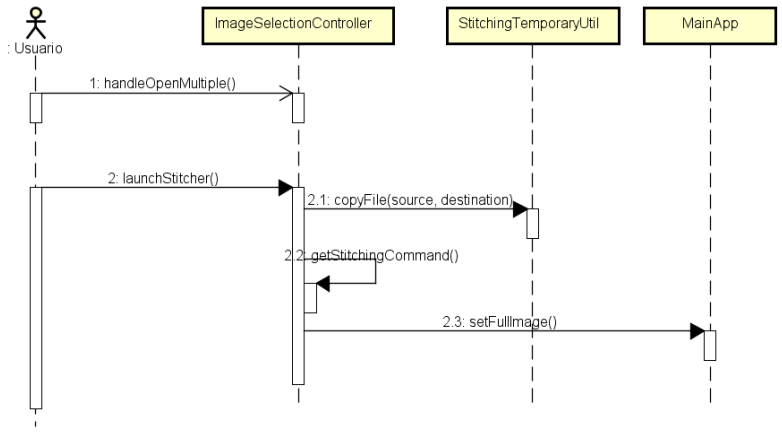


Figura C.5: Diagrama de secuencia para la operación de *stitching*

Rotación y recortado de imagen

Esta fase sirve para seleccionar la corona dental en la imagen y rotarla para hacer que las perikymata queden orientadas de forma vertical, así queda preparada la imagen para la fase de filtrado. Además, también se guarda la medida que servirá para saber la escala de la imagen y poder hacer bien el cálculo de la distancia de las perikymata.

Este proceso se describe de forma sencilla en la figura C.6

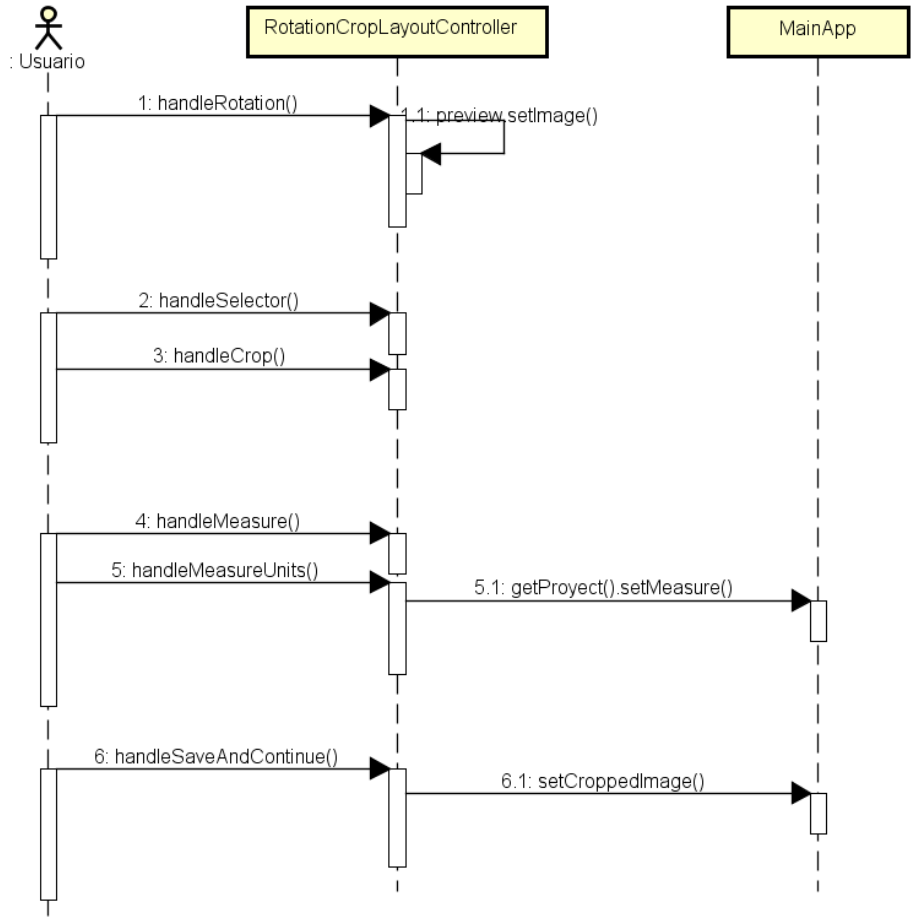


Figura C.6: Diagrama de secuencia para rotación, recortado y medida de escala

Filtrado de la imagen

Este es el principio de la última fase de la aplicación (figura C.7). El usuario usa el botón de filtrado por defecto o usa el filtrado avanzado modificando los parámetros que desee.

La aplicación Java se encarga de mandar una petición al servidor de Python vía sockets para filtrar la imagen. Cuando acaba de filtrar se lo comunica de nuevo a la aplicación principal.

Luego, se cargan en la aplicación dos imágenes filtradas, a una solo se la ha aplicado el filtro y a la otra además se la ha superpuesto la original. Así el usuario puede ver las perikymata detectadas con el proceso de filtrado sobre la original.

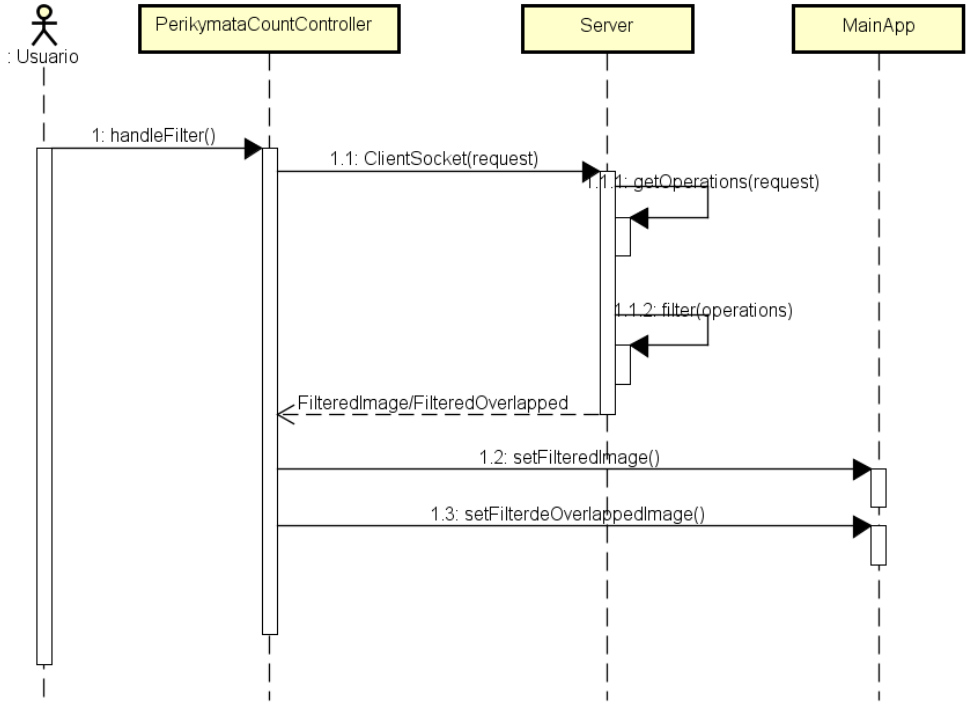


Figura C.7: Diagrama de secuencia para el filtrado

Recuento de perikymata y exportación de datos

Este paso pone fin a la última fase de la aplicación (figura C.8).
A partir de las imágenes filtradas que hemos comentado, el usuario pulsa un botón para dibujar una línea donde mejor se vean las perikymata y después pulsa otro botón para que la aplicación marque las que encuentre sobre esa línea y se las muestre. Después, con un botón, se exportan los resultados a un archivo *csv*.

Los datos de donde se ha dibujado la línea y las perikymata marcadas también se guardan en el fichero XML del proyecto. También se permite al usuario que borre o añada perikymata donde sea necesario y para ello tenemos dos botones.

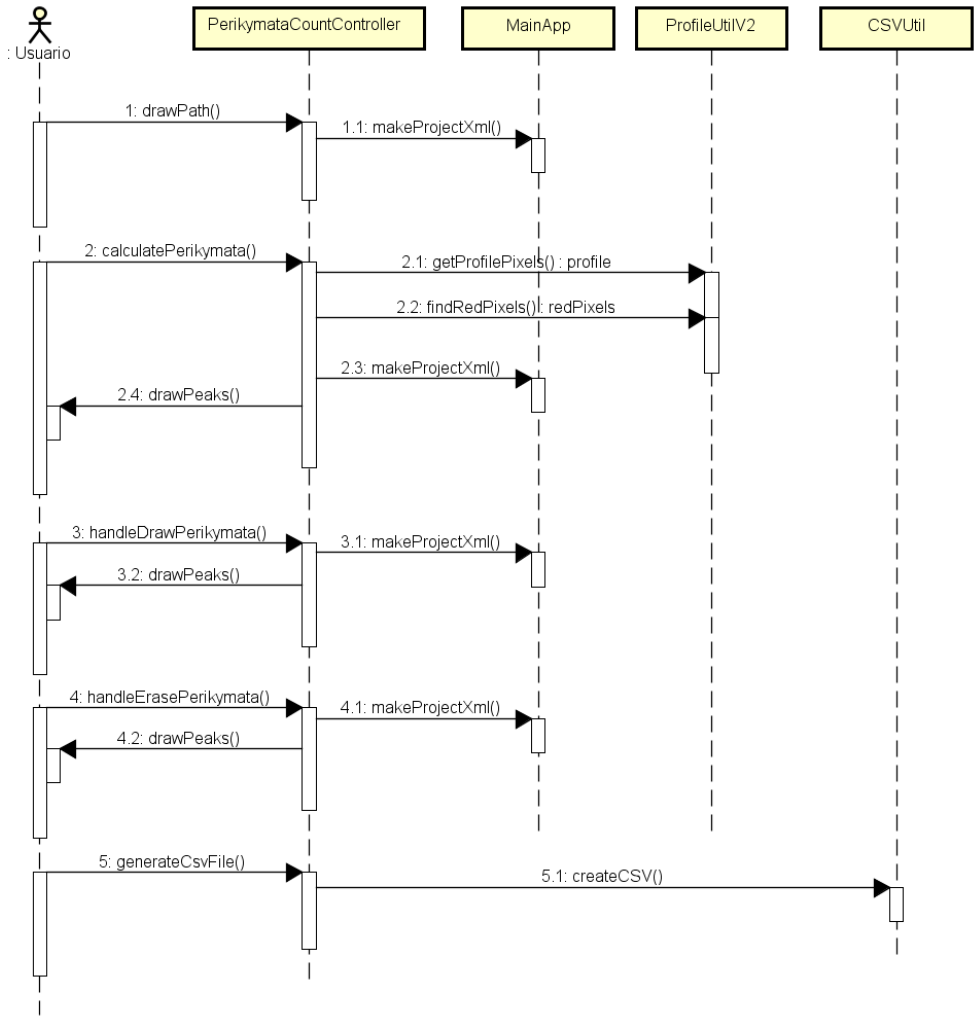


Figura C.8: Diagrama de secuencia para el filtrado

Apéndice D

Documentación técnica de programación

D.1. Introducción

En este apartado se describirán la estructura de directorios, el manual del programador, las pruebas del sistema y todos los elementos necesarios para poder compilar, instalar y ejecutar el proyecto.

D.2. Estructura de directorios

Tenemos dos aplicaciones, la aplicación principal en Java y el servidor en Python. Empezaremos por la estructura de directorios de la aplicación Java.

- *src.es.ubu.lsi.perikymata*: Contiene el controlador principal *MainApp.java* y los paquetes que desarrollan la aplicación siguiendo el patrón Modelo-Vista-Controlador.
    - *modelo*: contiene las clases del modelo, que permiten tener los datos del proyectos de forma persistente.
    - *vista*: contiene los controladores y los archivos FXML que son las vistas de la aplicación Java.
    - *util*: contiene el paquete *sockets* y clases de utilidad que se usan para exportar los datos a *csv*, obtener las perikymata de la imagen, leer datos de ficheros XML y hacer comprobaciones para la operación de *stitching*.
        - *sockets*: Contiene la clase socket que hace de cliente y la clase que representa una petición al servidor.

- *src.rsc*: contiene el paquete *stitching* y los iconos que utiliza la aplicación.
  - *stitching*: contiene los archivos relacionados con la operación de *stitching*.
    - *bin*: contiene los ejecutable de *stitching* para Windows[1] y Linux (ficheros *.ubu*).
    - *source*: contiene el código fuente de la aplicación de *stitching*. Está escrito en C++.
- *src.test.es.ubu.lsi.perikymata*: contiene la pruebas unitarias sobre las clases de utilidad.

La aplicación en Python se encuentra en la carpeta *PythonApp* y dentro encontramos la siguiente estructura:

- *src*:
- *Installation*:

D.3. Manual del programador

A continuación, hablaremos de los aspectos relevantes que puedan servir de ayuda al programador y de las herramientas necesarias para continuar el desarrollo del proyecto.

Java 8, Eclipse y la aplicación Java

En primer lugar, será necesario descargar el kit de desarrollo de Java 8 (JDK[2]). Se han utilizado expresiones *lambda*, por lo que el uso de versiones anteriores queda descartado pues no tienen esa función. No deberían existir problemas en utilizar versiones superiores.

En el JDK ya encontramos lo necesario para poder usar JavaFX, pero para que sea más fácil el diseño de las vistas, se recomienda descargar JavaFX Scene Builder 2.0[3], que es el que se ha usado en este proyecto.

La aplicación en Java ha sido desarrollada usando Eclipse y para cargarla solamente será necesario importar el proyecto. Podemos descargar Eclipse desde:
https://eclipse.org/

[1] Para Windows solo tiene el ejecutable de 32 bits porque los sistemas de 64 si que pueden ejecutarlo.
[2] JDK 8: https://goo.gl/KJ3ttC.
[3] Scene Builder: https://goo.gl/X1vErn.

Python 3, PyCharm y la aplicación servidor

Con Python 3, en concreto la versión 3.5, se ha desarrollado el servidor que atiende las peticiones de la aplicación Java para filtrar la imagen de la pieza dental. En las últimas distribuciones Linux viene incluido por defecto, pero en Windows es necesario instalarlo.

Para el filtrado se ha utilizado *Scikit-Image* 0.13.0, que es un paquete que contiene una gran cantidad de funciones orientadas al procesamiento de imagen para Python.

Tiene dependencias con otros paquetes como *Numpy*, *Sci-Py* y *Matplotlib* por lo que para poder usarlo será necesario tenerlos instalados.

En Windows, la forma más rápida y cómoda de instalar todo es descargar e instalar Anaconda3, que incluye Python 3 y los paquetes que necesitamos, aunque también incluye muchos otros paquetes que no se utilizan.

Encontramos Anaconda en el siguiente enlace:
https://www.continuum.io/downloads.

Otra opción es instalar Miniconda3 e instalar manualmente *Scikit-Image* y sus dependencias, ya que Miniconda únicamente contiene el intérprete de Python y alguna funcionalidad muy básica.

El IDE con el que se ha desarrollado todo el código Python, es PyCharm y se puede descargar desde aquí:
https://www.jetbrains.com/pycharm/download/

Para cargar la aplicación de Python en PyCharm, únicamente importaremos la carpeta *PythonApp*, que contiene el fichero que lanza el servidor y el fichero encargado del filtrado.

Stitching, OpenCV y Visual Studio

La versión anterior del proyecto [1] utilizaba OpenCV 3.1 para compilar la aplicación de *stitching*, dando como resultado un ejecutable que solo corría en sistemas de 64 bits.

Para poder tener un ejecutable de 32 bits se ha utilizado la versión 2.4.11 porque es la última que tiene compiladas las librerías de 32 bits para usar en Visual Studio. Tiene la restricción de que el último IDE compatible es Visual Studio 2013. Para que Visual Studio utilice las librerías estáticas, es necesario indicárselo en el apartado de propiedades del proyecto de *stitching* que hayamos creado, como se muestra en la figura D.1

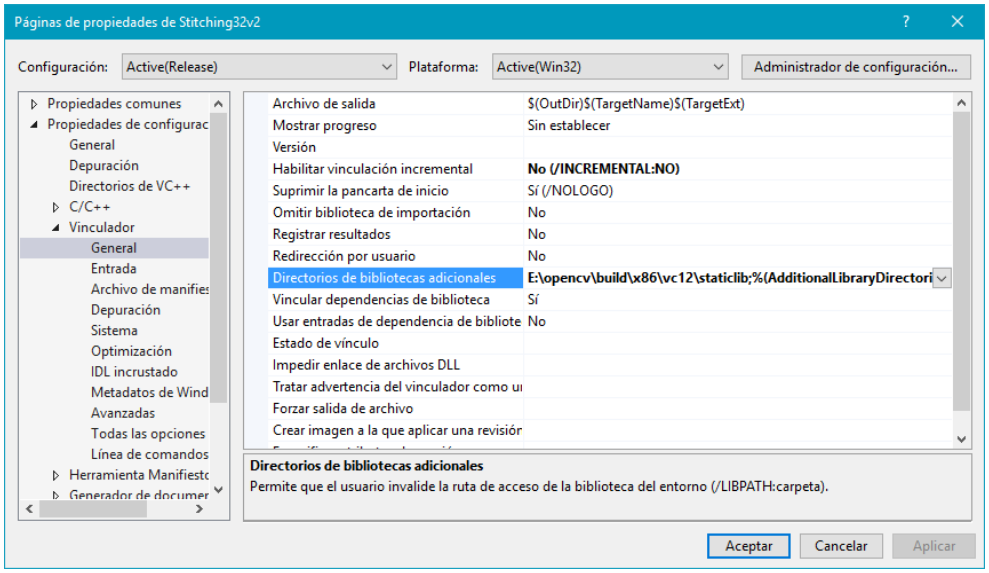


Figura D.1: Configuración de librerías estáticas

OpenCV 2.4.11 lo descargaremos de aquí:
https://goo.gl/9W8Mpr

Y Visual Studio 2013 desde aquí:
https://www.visualstudio.com/es/vs/older-downloads/

Compilar la aplicación de *stitching* en Ubuntu

Para compilar la aplicación de *stitching* para Linux[4] se tiene que descargar OpenCV 2.4.11, compilar el código fuente usando *CMake* para generar las librerías y después compilar la aplicación C++ configurando en modo estático las librerías de OpenCV mediante un archivo *CMakeLists.txt*.

Existen multitud de tutoriales[5] y *scripts* que instalan OpenCV, uno de ellos que puede servir para instalar la versión 2.4.11 es el que hay en el siguiente repositorio:
https://gist.github.com/ceefour/9a54109d9e16050e6742

El fichero *CMakeLists.txt* quedaría como en la figura D.2 y deberá estar situado en la misma carpeta que el código C++ de *stitching*.

[4]En este proyecto, los desarrollos en Linux se han hecho utilizando Ubuntu 16.04 LTS.
[5]Manual de instalación en Linux de OpenCV: https://goo.gl/L4go7Y.

```
cmake_minimum_required(VERSION 2.8)
project( Stitching )
find_package( OpenCV REQUIRED )
add_executable( Stitching Stitching.cpp )
target_link_libraries( Stitching ${OpenCV_LIBS} )
```

Figura D.2: Fichero *CMakeLists.txt*

Después, únicamente abriríamos una terminal y ejecutaríamos los siguientes comandos:

$ cmake .
$ make

El filtrado

Comentaremos brevemente el proceso de filtrado de imágenes elaborado en Python, las finalidad de las operaciones más relevantes puede encontrarse en la memoria:

1. Lectura de la imagen.
2. Conversión de la imagen a escala de grises.
3. Realizar la ecualización adaptativa del histograma de la imagen.
4. Reducción de ruido mediante una función que usa el Algoritmo Chambolle.
5. Realizar la operación de convolución con el kernel de Kirsch seleccionado.
6. Binarizar la imagen.
7. Eliminar objetos pequeños.
8. Esqueletonizar la imagen.
9. Detectar las líneas mediante la transformada probabilística de Hough.
10. Guardar la imagen con la líneas detectadas
11. Guardar la imagen original superponiéndola las líneas detectadas.

D.4. Compilación, instalación y ejecución del proyecto

En primer lugar, se deberá descargar o clonar el proyecto desde el repositorio de GitHub:
https://github.com/amtBurgos/Perikymata2017

En el repositorio no se encuentran las imágenes de las pruebas con *Jupyter Notebooks*, los proyectos de prueba de la aplicación, ni los ficheros fuente de la documentación. Estos archivos vienen incluidos en el DVD que acompaña a la documentación.

Para clonarlo (figura D.3), abrimos Eclipse y seleccionamos *Importar*, *Project from Git*, *Clone URI* y pulsaremos los sucesivos *siguientes* que aparezcan de modo que al final se habrá importado el proyecto.

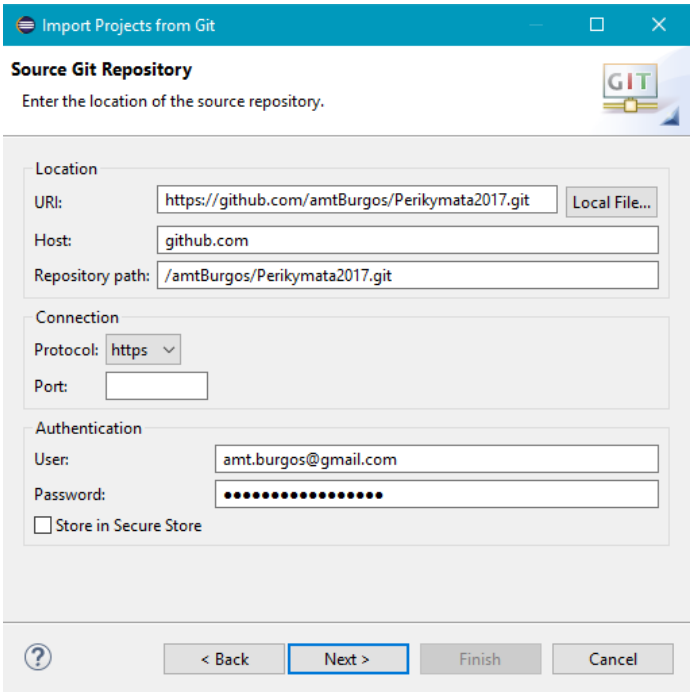


Figura D.3: Importar proyecto

Una vez importado, tendremos una estructura como la figura D.4

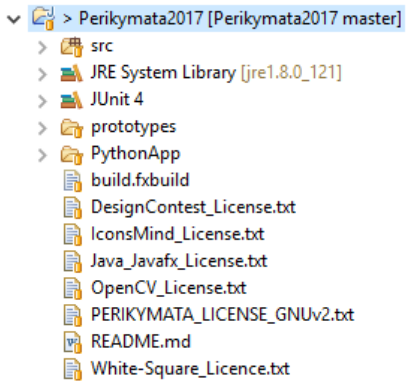


Figura D.4: Proyecto importado

Importar la aplicación Java

Para que la aplicación Java funcione, solo es necesario que esté la carpeta *src*, aunque aquí nos aparezcan más.

Para compilarlo

La carpeta *PythonApp* contiene los archivos necesarios para el funcionamiento del servidor de Python. Tiene que estar en la misma ruta del proyecto. Al igual que la carpeta *prototypes*, no es necesaria que esté como carpeta en Eclipse (aunque sí tiene que estar físicamente), por lo que queda a elección del programador añadir un filtro de Eclipse para que la ignore.

Importar aplicación Python

Este paso se realiza después de descargabar el proyecto y configurarlo en Eclipse. Para poder importar la aplicación de Python, abriremos PyCharm y haremos clic en *File*, *Open* y seleccionaremos la carpeta *PythonApp*.

Ejecución del proyecto

Para ejecutar el proyecto deberemos pulsar el icono de *play* de Eclipse y seleccionar la aplicación principal *MainApp* como en la figura D.5.

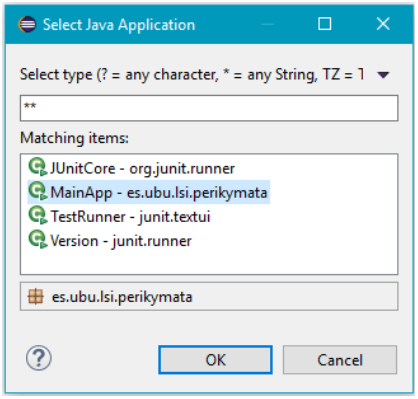


Figura D.5: Ejecutar aplicación

En Windows, esto hará que se habrá una ventana de comandos donde se ejecute el servidor, haciendo uso del fichero *StartServerWindows.bat* que hay en la carpeta *PythonApp* (por eso la aplicación Java y el servidor Python deben ir juntos). En Linux, el servidor corre en segundo plano, de modo que no se abrirá ninguna ventana.

Se ha encontrado que, en algunas ocasiones, la aplicación Java no ha podido ejecutar el fichero *.bat* por lo que si estamos en esa situación, habrá que ejecutar el servidor desde PyCharm primero y luego la aplicación Java desde Eclipse. En PyCharm solo hay que pulsar el botón de *play* para arrancarlo.

La aplicación Python no se compila porque es un lenguaje interpretado, sin embargo, la primera vez que se ejecute, tardará un rato en arrancar porque necesita crear algunos archivos en caché.

D.5. Pruebas del sistema

Disponemos de pruebas unitarias sobre las clases de utilidad en la aplicación Java. Para ello se ha seguido usando JUnit. Como nuevas pruebas respecto a la versión, encontramos la que prueba que la lanza un servidor y prueba que la clase *ClientSocket.java* se conecte adecuadamente y mande una petición de filtrado válida al servido. También se prueba la clase *StitchingUtil.java* que se encarga de validar y copiar los ficheros en la carpeta temporal para la operación de unión de imágenes. Podemos ver todas las pruebas unitarias en la figura D.6.

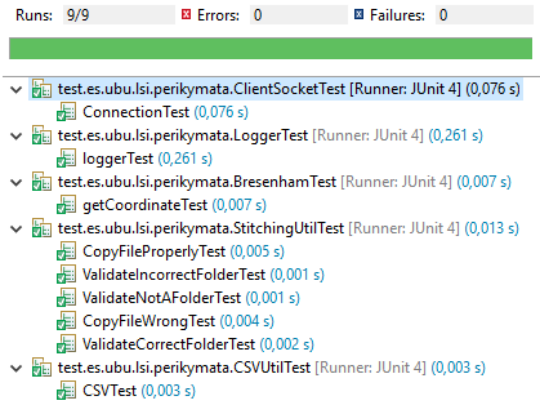
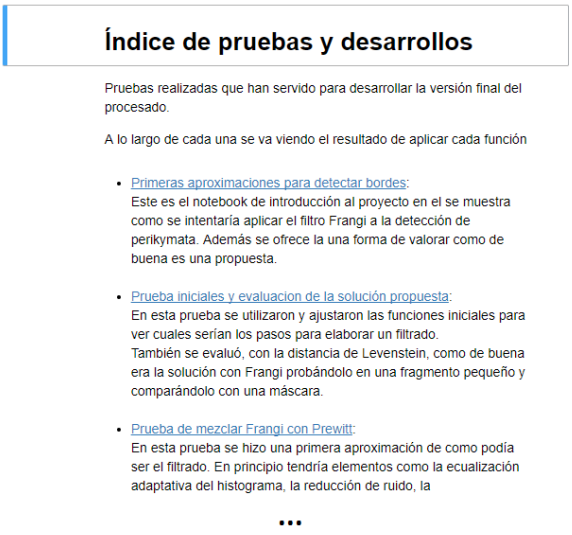


Figura D.6: Pruebas unitarias

En la carpeta *prototypes* se incluyen los *Jupyter notebooks* que han servido para realizar pruebas y desarrollar el filtrado final. Podemos ver todas las pruebas organizadas a partir del archivo *Índice de pruebas y desarrollos.ipynb* (figura D.7[6]). Cada prueba muestra el filtrado utilizado y como queda al aplicarlo a las imágenes de piezas dentales.

[6]En la figura se muestran las tres primeras.

En el DVD aportado, esta carpeta contiene además todas las imágenes resultantes de los distintos filtrados. En cada subcarpeta se ha incluido un fichero *info.txt* que explica el contenido



Índice de pruebas y desarrollos

Pruebas realizadas que han servido para desarrollar la versión final del procesado.

A lo largo de cada una se va viendo el resultado de aplicar cada función

- Primeras aproximaciones para detectar bordes:
  Este es el notebook de introducción al proyecto en el se muestra como se intentaría aplicar el filtro Frangi a la detección de perikymata. Además se ofrece la una forma de valorar como de buena es una propuesta.
- Prueba iniciales y evaluacion de la solución propuesta:
  En esta prueba se utilizaron y ajustaron las funciones iniciales para ver cuales serían los pasos para elaborar un filtrado.
  También se evaluó, con la distancia de Levenstein, como de buena era la solución con Frangi probándolo en una fragmento pequeño y comparándolo con una máscara.
- Prueba de mezclar Frangi con Prewitt:
  En esta prueba se hizo una primera aproximación de como podía ser el filtrado. En principio tendría elementos como la ecualización adaptativa del histograma, la reducción de ruido, la

...

Figura D.7: Índice de pruebas

Las pruebas sobre la interfaz se han realizado supervisando su correcto funcionamiento cada vez que se incluían cambios en los componentes.

Apéndice E

Documentación de usuario

E.1. Introducción

En este apartado se explicarán los pasos necesarios para poder instalar la aplicación y usarla correctamente.

E.2. Requisitos de usuarios

- Es necesario disponer de un sistema operativo Windows 7 o superior o Ubuntu 16.04 LTS[1].
- El procesador del equipo puede ser de 32 o 64 bits.
- En cuanto al software, será necesario tener instalado en el equipo Java 8 y Python 3.

Sistemas Windows

Para Windows descargaremos Java 8 desde:
http://www.oracle.com/technetwork/java/javase/downloads/

Para obtener Python 3, descargaremos la siguiente versión de Miniconda3[2]:

- Si nuestro sistema es de 32 bits:
  https://repo.continuum.io/miniconda/Miniconda3-3.19.0-Windows-x86.exe

[1] Pueden usarse otras distribuciones Linux, pero los ficheros de instalación proporcionados pueden no funcionar

[2] Actualmente existen errores en los repositorios de *conda* al instalar la última versión de Miniconda3, pero en la versión que aquí se facilita, la instalación es correcta.

- Si nuestro sistema es de 64 bits:
  https://repo.continuum.io/miniconda/Miniconda3-3.19.0-Windows-x86_64.exe

Al instalar Miniconda3 hay que marcar la opción de añadir Anaconda al *path* del sistema como en la figura E.1.

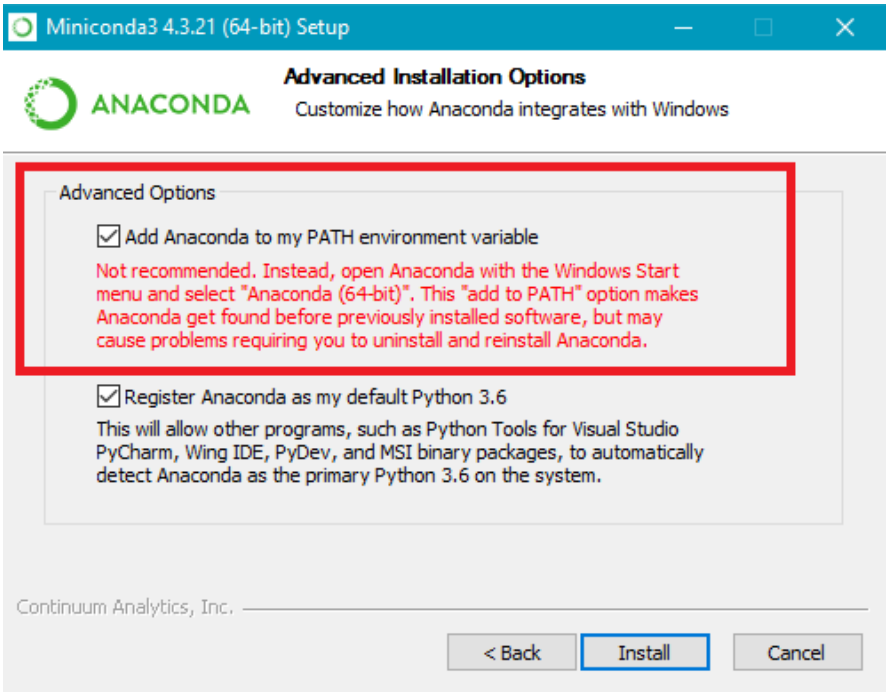


Figura E.1: Instalación de Miniconda

Sistemas Linux

La instalación de Java 8 para sistemas Linux se debe llevar a cabo desde una terminal mediante el siguiente comando:

$ sudo apt-get install oracle-java8-installer

Python 3 viene incluido desde hace tiempo en las distribuciones Linux ya que, en un futuro, será la versión por defecto. En caso de que no estuviese instalado utilizaríamos el siguiente comando desde una terminal:

$ sudo apt-get install python3

E.3. Instalación

Una vez cumplidos con los requisitos, descargaremos el fichero *Perikymata2.0.zip*, que contiene la última versión disponible de la aplicación. Podemos encontrarlo en el siguiente enlace:
https://github.com/amtBurgos/Perikymata2017/releases

En el encontraremos cuatro elementos:

- La aplicación Java llamada *Perikymata2.0.jar*.
- La carpeta con el servidor en Python llamada *PythonApp*.
- Un script Linux llamado *StartPerikymata.sh*[3] para arrancar la aplicación Java.
- Una carpeta llamada *Installation* con los scripts para poder instalar *Scikit-Image* en Windows y Linux.

Será necesario instalar *Scikit-Image* para que funcione el servidor de Python por lo que se han incluido en la carpeta *Installation* dos ficheros:

- Para instalar *Scikit-Image* en Windows, haremos doble clic sobre *ServerInstallation.bat* y esperaremos, ya que el proceso tarda un rato.
- Para instalarlo en Linux, abriremos una terminal desde la carpeta *Installation* y ejecutaremos los siguientes comandos:

  $ chmod +x ServerInstallationLinux.sh[4]

  $ ./ServerInstallationLinux.sh[5]

  La distribución donde se han probado estos *scripts* es Ubuntu 16.04 LTS. Es posible que no funcionen en otras distribuciones, en tal caso, correrá a cuenta del usuario instalar *Scikit-Image* y sus dependencias. Podemos encontrar los pasos desde su página oficial:
  http://scikit-image.org/docs/0.13.x/install.html

  Dejaremos preparado también, los permisos del fichero *StartPerikymata.sh*, abriendo una terminal desde la carpeta principal *Perikymata2.0* e introduciendo el siguiente comando:

  $ chmod +x StartPerikymata.sh

[3] Ha sido necesario incluir este fichero porque, mediante el doble clic, el arranque de la aplicación no respondía adecuadamente.
[4] Este comando da permisos de ejecución al script.
[5] Es posible que nos pida alguna confirmación para realizar la instalación.

E.4. Manual del usuario

A continuación se mostrará el modo de usar la aplicación, explicando cada una de sus fases, opciones y el flujo normal de ejecución. Las fases que contiene son las siguiente:

- Arranque de la aplicación. En esta fase se crea o se abre un proyecto.
- Selección y unión de imágenes (*stitching*). Se juntan las imágenes para conseguir una imagen completa de la pieza dental.
- Rotación y recortado de la imagen. Se prepara la imagen para el filtrado y se consigue la medida.
- Filtrado de la imagen. Se filtra la imagen para marcar las perikymata.
- Marcado de perikymata y exportación de datos.

Arranque de la aplicación

En primer lugar, es necesario recordar que para que la aplicación funcione, deben estar en la misma carpeta el ejecutable Java y la carpeta con el servidor de Python.

Como veíamos en la sección anterior, la carpeta *Perikymata2.0* tiene dos elementos, el servidor en Python (carpeta *PythonApp*) y el ejecutable Java de la aplicación principal (*Perikymata2.0.jar*).

Desde esa carpeta, en Windows haremos doble clic sobre el fichero *Perikymata2.0.jar* y en Linux abriremos una terminal y pondremos el siguiente comando:

$ ./StartPerikymata.sh

Este script ejecuta el comando: *$ java -jar Perikymata2.0.jar*. Ha sido necesario incluir este fichero en Linux debido a que el doble clic no funcionaba en algunas ocasiones, así nos aseguramos de que la aplicación se inicia correctamente siempre[6].

Automáticamente se inicia el servidor, la primera vez que se ejecute todo el proyecto puede tardar un poco en arrancar porque necesita crear unos archivos en caché, por lo que esperaremos. En Linux corre en segundo plano, pero en Windows se nos abre una ventana de comandos donde veremos las operaciones que van sucediendo.

[6]La aplicación en Linux puede dar a veces errores que causan el mal funcionamiento de la botonera superior y el redimensionado de la aplicación

Si no teníamos ningún proyecto cargado, aparecerá la primera ventana de la aplicación como la de la figura E.2. En caso contrario, se abrirá el último proyecto con el que se ha trabajado desde la última etapa de la aplicación donde se tengan datos.

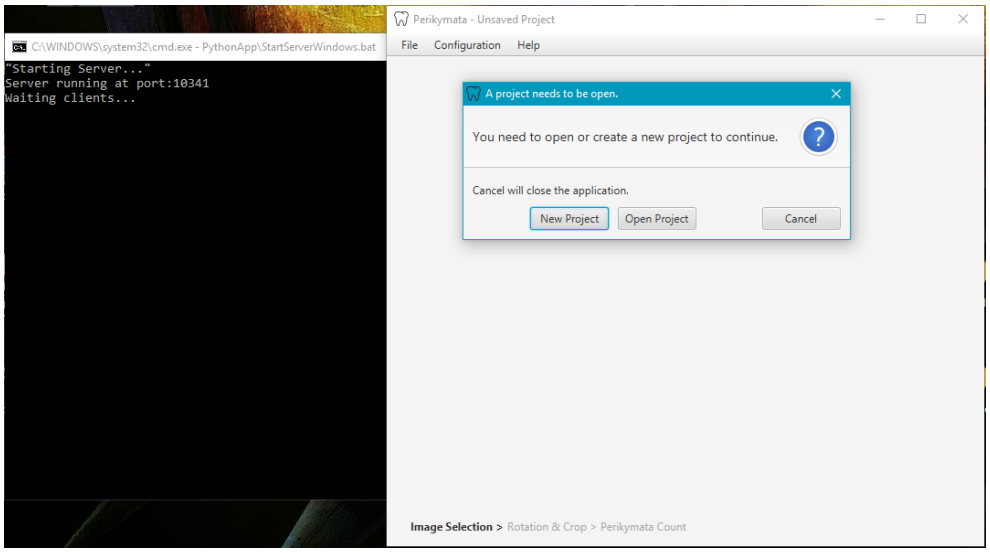


Figura E.2: Fase de inicio

En esta ventana encontraremos tres botones:

- *New Project*: crea un nuevo proyecto de perikymata, para ello nos pide el nombre y el lugar donde queremos guardarlo.
- *Open Project*: Abre un proyecto. Tenemos que buscar la carpeta del proyecto que queremos abrir y en ella, el fichero XML que contiene la configuración. El nombre de este fichero es del tipo *¡NombreProyecto¿.xml*.
- *Cancel*: Cierra el servidor y la aplicación.

Pulsaremos *New Project* y crearemos un proyecto en un sitio de nuestra preferencia. Dentro del proyecto, se crearán los siguientes elementos:

- Carpeta *Fragments*: contiene los fragmentos que se han seleccionado para juntar las imágenes en la fase de selección y unión de imágenes.
- Carpeta *Full_Image*: contiene la imagen completa del diente, que ha introducido el usuario o que es el resultado de la unión de los fragmentos
- Carpeta *Cropped_Image*: Contiene la imagen original recortada de la fase de recorte, la imagen filtrada con líneas detectadas y la imagen original con las líneas detectadas superpuestas.

- Carpeta *Perikymata_Outputs*: contiene el fichero *csv* con los datos exportados.
- Fichero *¡NombreProyecto¿.xml*: es el fichero de configuración del proyecto. Contiene información sobre la carpeta temporal elegida, las líneas dibujadas y las perikymata marcadas.

Selección y unión de imágenes

La siguiente fase es la selección y unión de imágenes. La pantalla que nos aparecerá es como la de la figura E.3

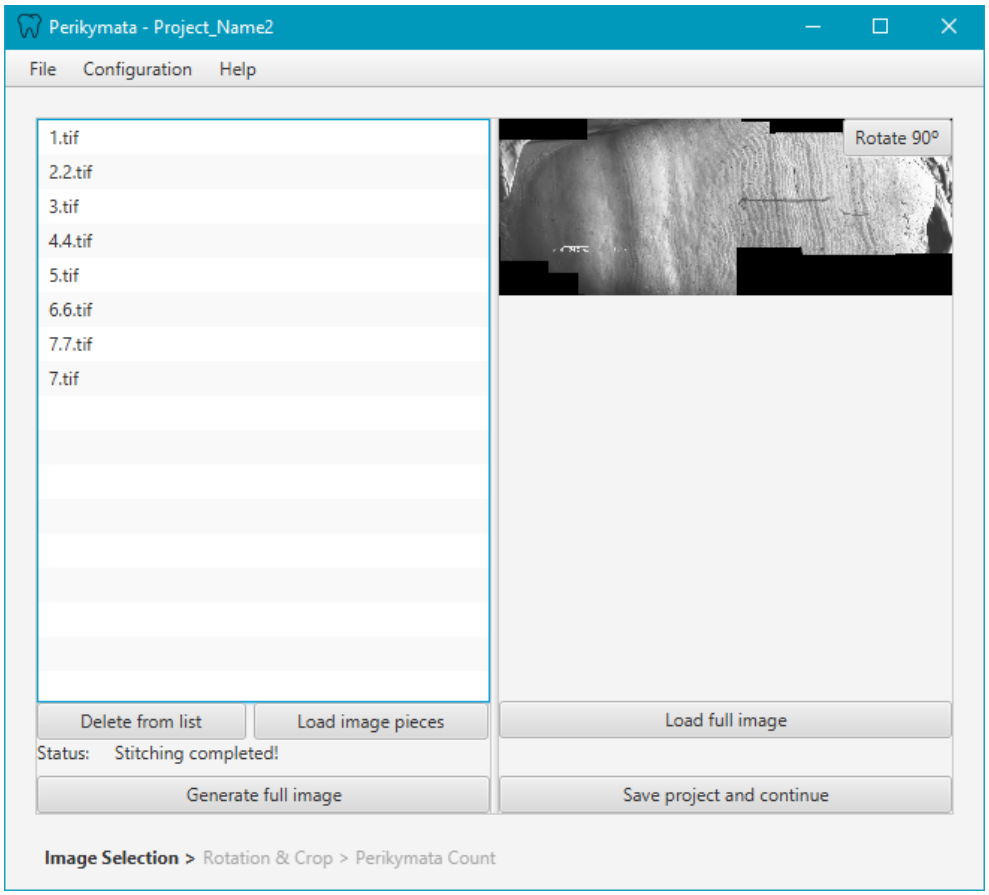


Figura E.3: Fase de selección y unión de imágenes

El objetivo es tener la imagen completa del diente. Los pasos para obtenerla serán los siguientes:

- Pulsar *Load image pieces*: nos permitirá cargar las imágenes de fragmentos del diente.

- Pulsar *Delete from list*: si nos hemos equivocado, seleccionaremos la imagen a borrar de la lista y luego pulsaremos este botón.
- Pulsar *Generate full image*: inicia el proceso de unión de imágenes *stitching*. Tarda más tiempo o menos dependiendo de la cantidad de imágenes y de la potencia del ordenador.
- Pulsar *Load full image*: con este botón cargamos una imagen completa del diente que tengamos. En este caso no haría falta cargar y unir los fragmentos pequeños.
- Pulsar *Rotate 90º*: la imagen tiene que estar con las perikymata orientadas verticalmente. Este botón nos permite rotar la imagen de forma rápida.
- Pulsar *Save project and continue*: guarda las imágenes y pasa a la siguiente fase.

En la figura E.3 también vemos un elemento en la parte de abajo, las *migas de pan*. Nos permiten ver en que fase de la aplicación nos encontramos y retroceder a otras anteriores.

Rotación y recortado de la imagen

En esta fase, el objetivo es rotar y recortar en la imagen la zona de la corona, para que la aplicación pueda más adelante delimitar bien los deciles donde se encuentran las perikymata.

También se busca dejar las perikymata orientadas los más verticalmente posible y para ello se permite rotar la imagen en un ángulo de 40º hacia cada lado. Además en esta fase, también se guarda la medida de la imagen que permite saber la escala.

Los botones tienen *tooltips*, de modo que si dejamos el ratón encima, nos aparecerá un mensaje explicando para que vale cada botón.

Podemos ver la rotación y recortado en la figura E.4.

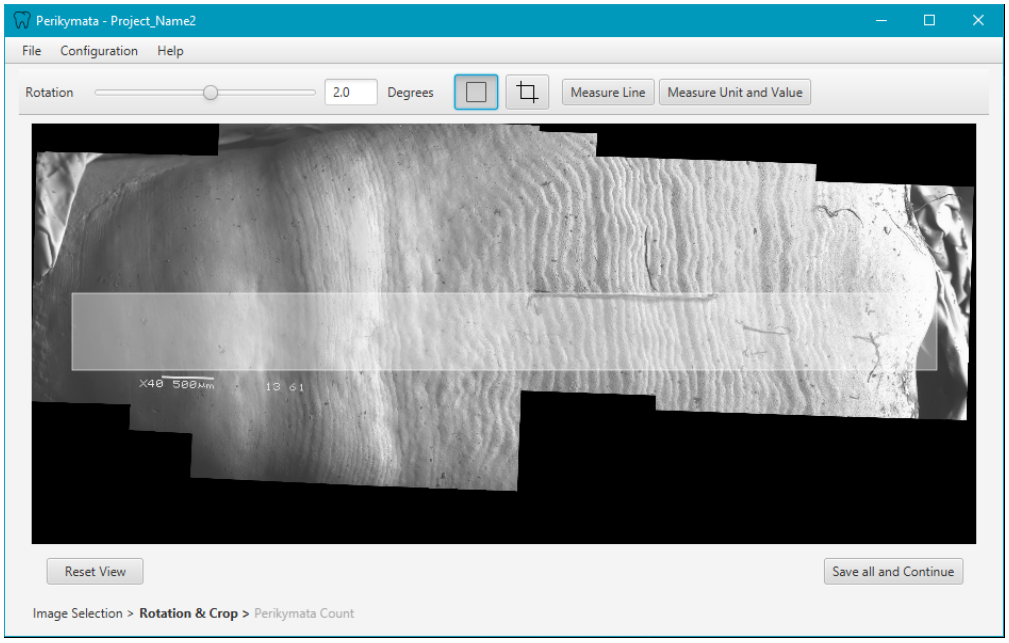


Figura E.4: Fase de rotación y recortado de la imagen

En primer lugar, usaremos el *slider* para rotar la imagen. También podemos introducir los grados que queremos que gire; para girar hacia la derecha introduciremos números positivos y hacia la izquierda, negativos.

Después con el botón *cuadrado* seleccionaremos la zona de la corona en la imagen y con el botón *recorte*[7] recortaremos la imagen. Esa imagen recortada será la que filtremos y sobre la que marquemos las perikymata. Es importante que el recorte abarque la anchura de la corona porque es donde se marcarán los deciles. La altura que debe tener el recorte queda a criterio del usuario; si damos más altura puede que incluyamos zonas de las perikymata que se vean y detecten mejor. Cuando más grande sea el recorte, más tiempo tardará en filtrarse (dentro de un tiempo razonado).

Si no estamos satisfechos con el recorte pulsaremos sobre *Reset View* para desechar los cambios.

Para obtener la medida pulsaremos en *Measure Line*. Después, pulsaremos y arrastraremos sobre la imagen en la zona de la leyenda para dibujar una línea roja que abarque la medida. Cuando acabemos, aparecerá una ventana donde nos pide ingresar la unidad de la medida (cm, mm, nm, etc) y el valor (figura E.5). Estos datos podemos cambiarlos también con el botón *Measure Unit and Value*.

[7]El recorte debe hacerse desde la parte izquierda de la imagen hacia la derecha.

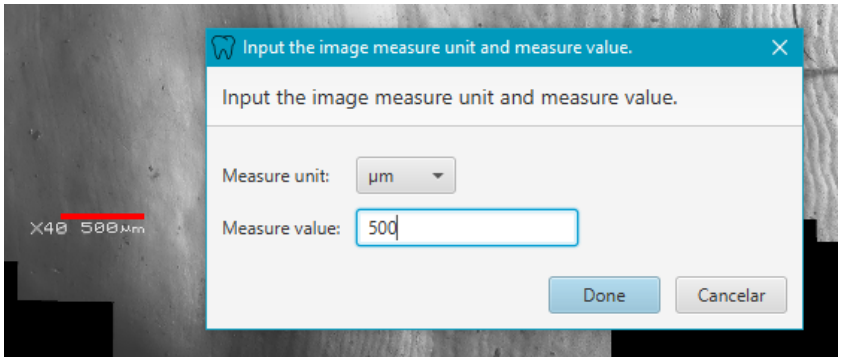


Figura E.5: Medida de la imagen

Para concluir esta fase, pulsaremos sobre *Save all and Continue*, se guardaran todos los datos y se pasará a la fase de filtrado y recuento de perikymata.

Filtrado de la imagen

El servidor en Python nos da el servicio de filtrado, en Windows se puede ver en la ventana de comandos que tenemos junto a la aplicación y en Linux corre en segundo plano.

En esta fase filtraremos la imagen[8] para marcar las perikymata y que después se puedan marcar fácilmente. Para ello contamos con el modo por defecto y modo avanzado. El estado inicial de la ventana será como el de la figura E.6

[8]El servidor tiene que estar arrancado, en caso contrario aparecerán mensajes de que no se puede filtrar

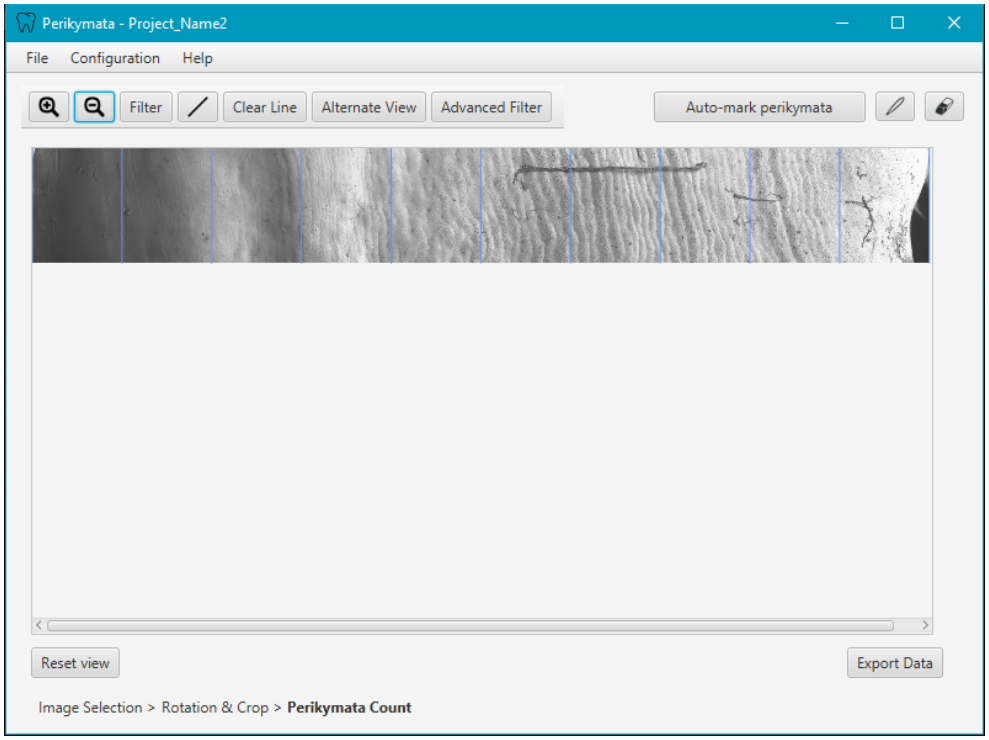


Figura E.6: Estado inicial de la ventana de filtrado y recuento

Para usar el modo por defecto, solamente pulsaremos el botón *Filter* y esperaremos hasta que la imagen cambie a una como en la figura E.7. Esta imagen está filtrada, con las perikymata encontradas reducidas a una anchura de un píxel, en color blanco, y las partes detectadas con línea roja.

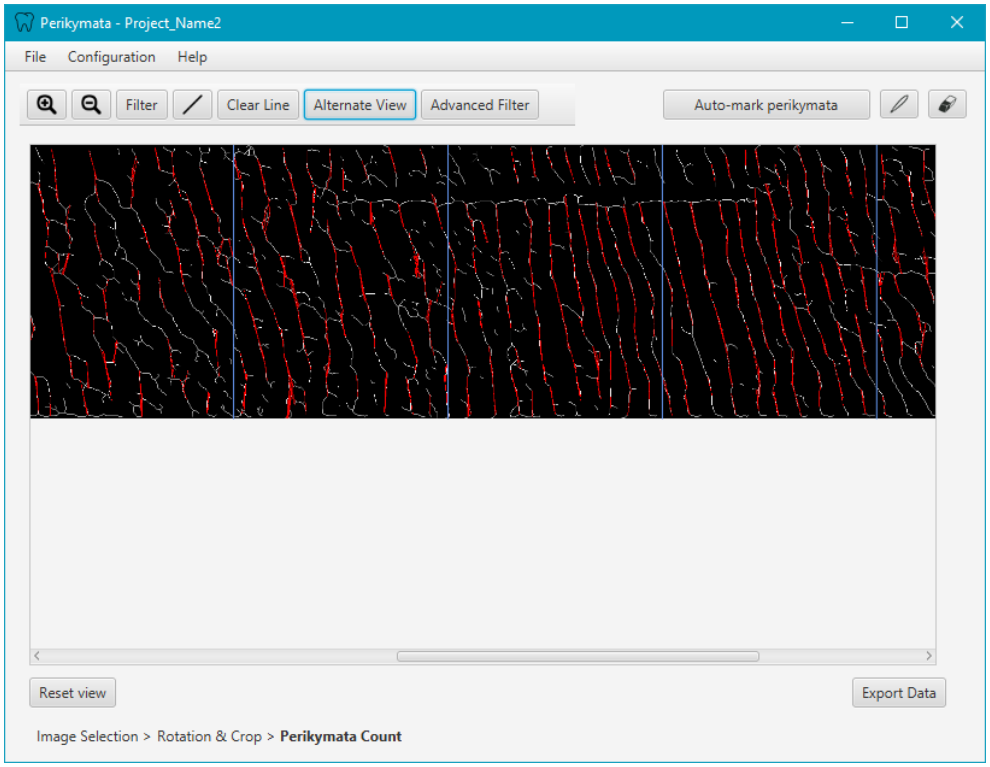


Figura E.7: Imagen filtrada por defecto

Disponemos de un botón *Alternate View* para cambiar a la imagen original con las líneas detectadas superpuestas, como en la figura E.8.

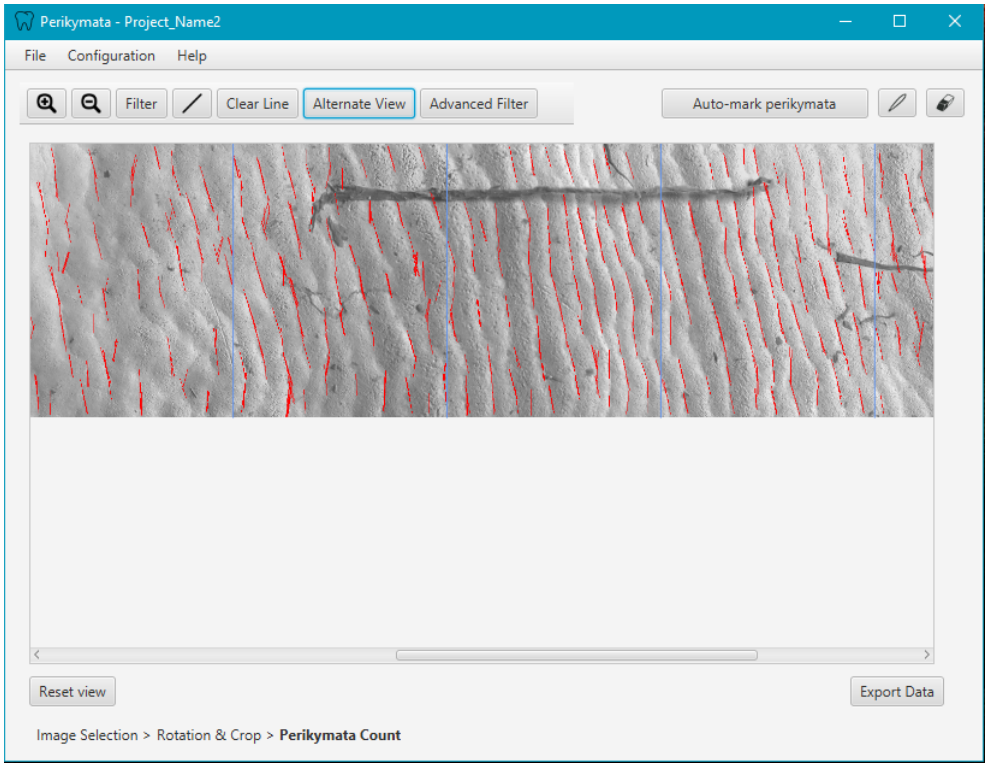


Figura E.8: Imagen original con líneas superpuestas

Aunque el filtrado por defecto es efectivo, con el filtrado avanzado tendremos más control para que se detecten mejor las perikymata. Pulsaremos el botón *Advanced Fiilter* y se extenderá la parte izquierda mostrando los parámetros del filtrado que podemos modificar (figura E.9).

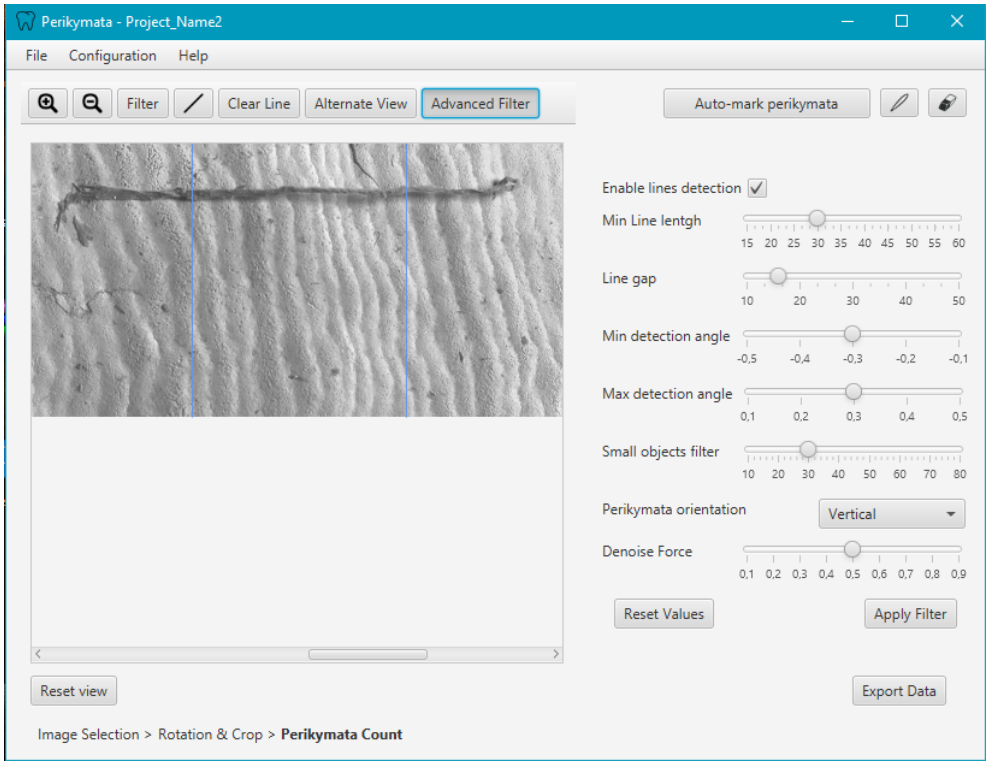


Figura E.9: Filtrado avanzado

Explicaremos a continuación para que sirven cada uno de estos parámetros:

- *Enable lines detection*: si lo dejamos marcado, podremos modificar parámetros hasta la opción *Max detection angle* (incluido) y en la imagen se reducirán perikymata a una anchura de un píxel (color blanco) y luego se detectarán las líneas (color rojo). Si lo desmarcamos, solo podremos modificar las dos últimas opciones y el resultado será una imagen con las perikymata reducidas a un píxel marcadas de color rojo, sin detección de líneas.
  Se recomienda desactivarlo si en la imagen original, las perikymata, se ven muy bien, pero si no se aprecian bien es mejor marcarlo. Podemos ver el resultado sin marcar en la figura E.10.

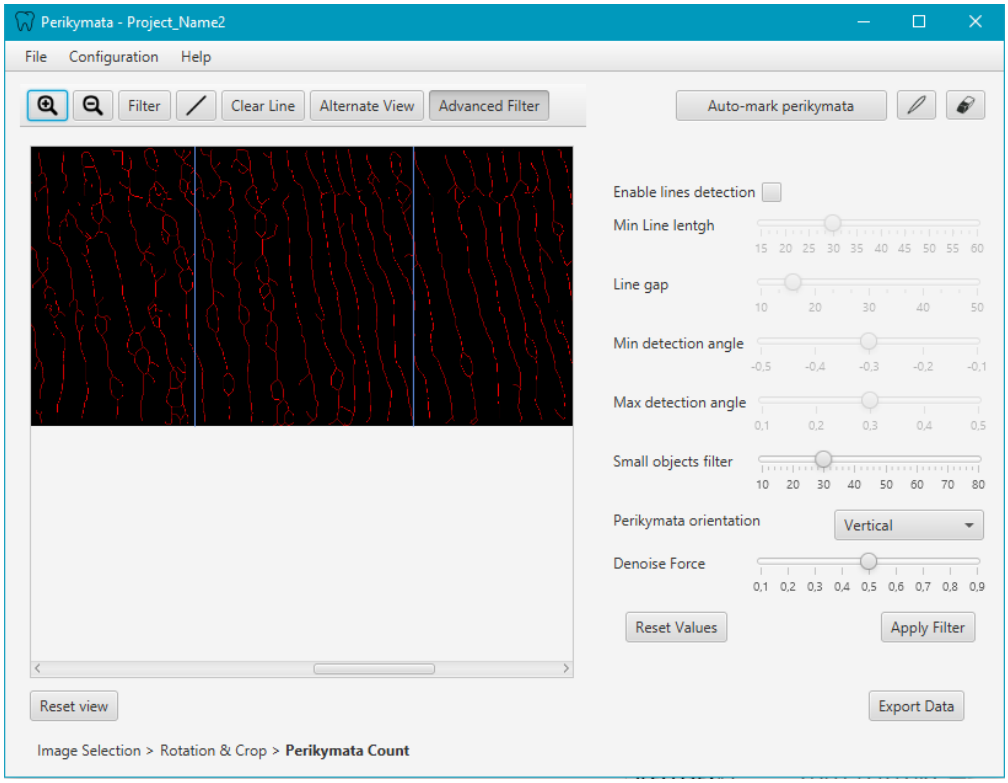


Figura E.10: Filtrado sin detección de líneas

- *Line gap*: esta opción indica la distancia máxima (en píxeles) entre dos puntos detectados para formar un línea roja. Si ponemos un valor pequeño, se formarán líneas rojas muy pequeñas, y si ponemos un valor demasiado grande, las líneas serán rectas muy largas. Recomendamos usar valores pequeños (razonables), para que se adapten a la forma de las perikymata.

- *Min detection angle*: como las perikymata no son puramente verticales, con este parámetro decimos el rango en el que buscar perikymata. Por defecto tiene valor -0.3. Si tomamos una línea vertical, este parámetro indica el límite en radianes hacia la izquierda en los que buscar.

- *Max detection angle*: indica el ángulo máximo en el que buscar perikymata. Al igual que el anterior parámetro, si tomamos un línea vertical, este parámetro indica el límite hacia la derecha en el que buscar perikymata.

- *Small object filter*: debido al ruido de la imagen, a veces se reducen al tamaño de un píxel objetos pequeños. Este sirve para comunicar que los objetos con una longitud menor que ese valor, serán eliminados.

- *Denoise Force*: A la imagen se le aplica primero un filtro de reducción de ruido. Si ponemos este parámetro muy alto, la imagen se difuminará, se eliminará ruido, pero las perikymata se diferenciarán peor. Se recomienda que si no se aprecia mucho ruido, se utilice con el valor *0.5* que aparece por defecto.

- *Perikymata orientation*: Este es el parámetro más importante de las opciones avanzadas. El proceso de filtrado resalta más las líneas según su orientación: verticales, horizontales, inclinadas. Este parámetro tres opciones y sus complementarias, seis en total:

  - *Vertical* - *Vertical 2*: Si indicamos este parámetro es porque las perikymata que vemos en la imagen original recortada son razonablemente verticales. Se ha incluido la opción dos porque, a veces, en vez de marcar la zona del *valle* de las perikymata[9], se marca la zona de la *cordillera*[10]. Nos interesa marcar el valle de las perikymata por lo que si una opción marca la cordillera, deberemos usar su complementaria. Podemos ver un ejemplo de este comportamiento en la figura E.11.
  - *Vertical Right* - *Vertical Right 2*: Sirve para resaltar las perikymata que tiene zonas inclinadas ligeramente hacia la derecha.
  - *Vertical Left* - *Vertical Left 2*: Sirve para resaltar las perikymata que tiene zonas inclinadas ligeramente hacia la izquierda.

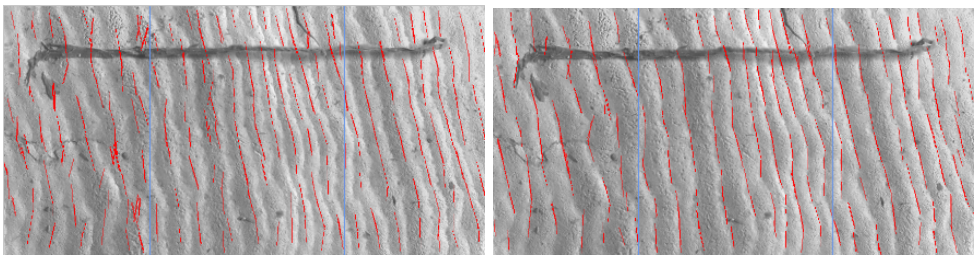


Figura E.11: Opciones Vertical (izda.) y Vertical 2 (dcha.)

Para terminar esta fase, comentaremos que también tenemos un botón *Reset View* para volver a comenzar el proceso si los resultados no son los esperados. Este botón cargará la imagen recortada original.

[9] El valle es la zona donde realmente se ve la línea de la perikyma
[10] La cordillera es la zona del esmalte del diente entre dos perikymata

Marcado de perikymata y exportación de datos

Con la imagen ya filtrada pulsaremos el botón representado por un *icono* de una línea. Pulsaremos y arrastraremos sobre la imagen pasando por las perikymata detectadas (color rojo) para que la aplicación pueda marcarlas posteriormente. También podemos ir pulsando sobre la imagen para que la línea se vaya creando. Si la línea excede los límites de la imagen, aparecerá un mensaje de error. El botón *Clear Line* permite borrar la línea si no nos hemos equivocado.

Al pulsar el botón *Auto-mark perikymata* se marcarán en verde las perikymata rojas por donde hemos pasado la línea (figura E.12). Es posible que algunos puntos verdes no sean verdaderamente una perikymata y otros puntos que se han quedado sin marcar sí; para ello disponemos en la esquina superior derecha de dos botones que nos permitirán añadir y quitar perikymata a la imagen.

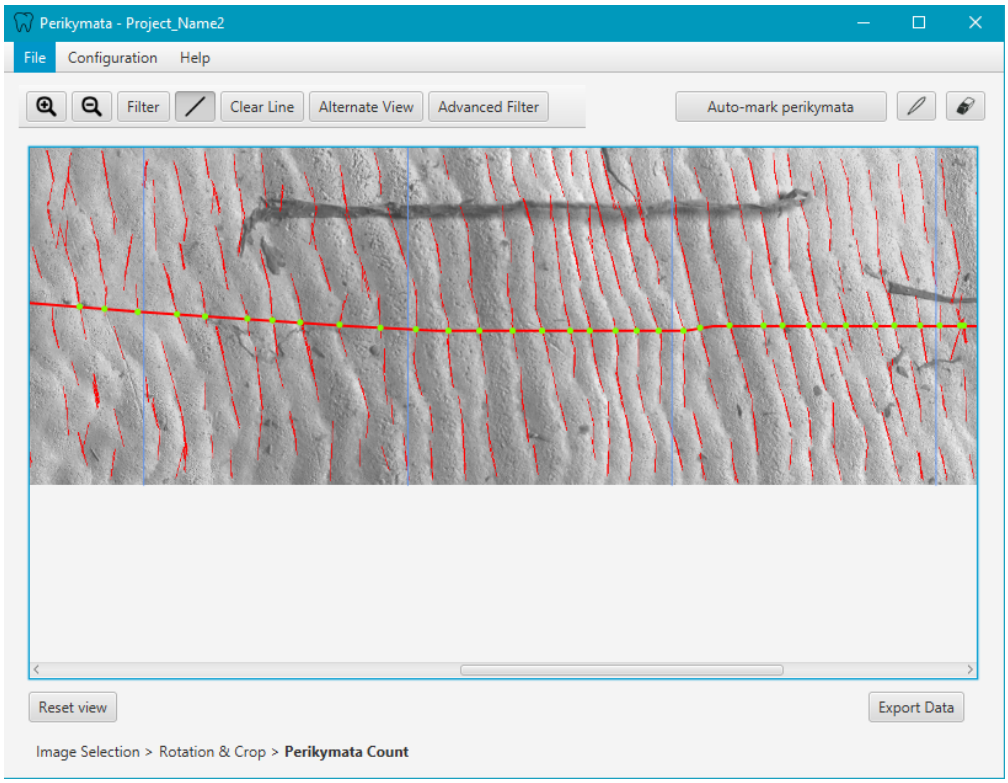


Figura E.12: Marcado de perikymata

Una vez todo esté de nuestro gusto, pulsaremos el botón *Export Data* y se guardará un fichero *csv* en la carpeta del proyecto *Perikymata_Outputs*. Este archivo contendrá las perikymata marcadas, el decil en el que se encuentran, sus coordenadas en píxeles, y la distancia a la perikyma anterior.

Otras funcionalidades

En la parte superior de la ventana (figura E.13), tenemos tres menús con distintos botones, explicaremos los más relevantes.

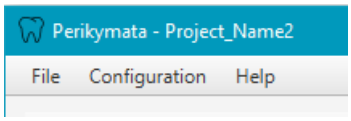


Figura E.13: Funcionalidades comunes

En el menú *File* tenemos los siguientes botones:

- *New Proyect*: crea un nuevo proyecto en el que poder trabajar.
- *Open Project*: permite abrir otros proyectos que tengamos.
- *Save*: guarda el estado del proyecto.
- *Close*: cierra la aplicación.

En el menú *Configuration* tenemos:

- *Full Screen*: hace que la ventana de la aplicación se maximice. Es útil en etapas como la del filtrado y recuento de perikymata, pues permite ver mejor toda la interfaz.
- *Set temporary folder*: para la unión de imágenes se necesita utilizar una carpeta sin rutas con espacios en blanco como carpeta temporal. Esta opción abre una ventana como la de la figura E.14 para permitirnos seleccionar una carpeta temporal de nuestra elección. La aplicación mostrará un mensaje de error si se intenta seleccionar una carpeta con espacios en blanco en su ruta.

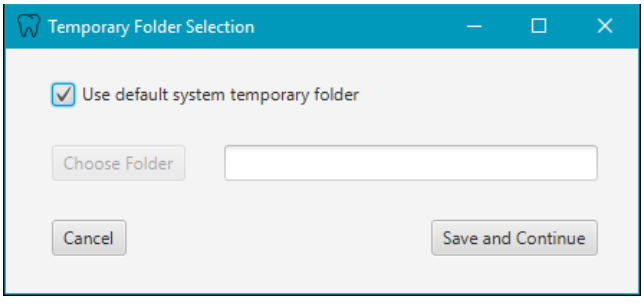


Figura E.14: Elección de carpeta temporal

En relación con este apartado, hay que explicar uno de los errores que se han encontrado al usar la aplicación en Linux. Puede darse la ocasión en que los botones de menú de la parte superior no respondan y que tampoco se pueda redimensionar la ventana. Salvo eso, la aplicación funciona con normalidad. Se desconoce el origen del error.

Observaciones

Sobre las imágenes a filtrar:
Las imágenes de fragmentos que se han proporcionado para este proyecto suelen venir incluidas en una carpeta. En ella encontramos imágenes con la leyenda que indica la medida y las mismas imágenes pero sin la leyenda.

Para que las imágenes se puedan unir correctamente, es necesario que se escojan todas las imágenes sin leyenda menos una, para poder tomar la medida en la etapa correspondiente de la aplicación. Se recomienda que la imagen con la medida sea una de las esquinas, para interferir lo más mínimo con la operación de unión. Además, es labor del usuario que estas estén orientadas en el mismo sentido, de modo que las perikymata queden de forma vertical. Si al unir las imágenes obtenemos una imagen completamente negra, es posible que debamos eliminar de la lista alguna imagen muy similar a otra, porque esta puede comprometer la operación.

Sobre la aplicación en Linux y los scripts proporcionados:
Los usuarios de Windows pueden ver si funciona el servidor en una ventana de comandos, pero los usuarios de Linux no. Por eso en la carpeta *PythonApp*, que contiene el servidor, se proporcionan los siguientes *scripts* por si los usuarios encuentran dificultades para usar la aplicación por problemas del servidor:

- *StartServerLinux.sh*: enciende el servidor.
- *StopServerLinux.sh*: detiene el proceso *python3* que usa el servidor.

Para poder usarlos, habría que abrir una terminal dentro de la carpeta *PythonApp* y escribir los siguientes comandos para cada *script*. El primer comando solo es necesario la primera vez:

$ chmod +x script.sh
$ ./script.sh

Hay que comentar también, que los scripts están hechos para ejecutarse en una *shell* de *bash*, como la de Ubuntu, por ejemplo. Si se utiliza otra *shell*, los usuarios pueden modificar los *scripts* para adaptarlos.

Bibliografía

[1] Sergio Chico Carrancio. Análisis paleontológico de piezas dentales. Trabajo de fin de grado, Universidad de Burgos, 2016.

[2] Ismael Tobar García. Estimación de la dieta por análisis de marcas dentales. Trabajo de fin de grado, Universidad de Burgos, 2017.

[3] 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.

[4] Wikipedia. Gnu general public license — wikipedia, la enciclopedia libre, 2017. (Accedido 24/06/2017).

[5] Wikipedia. Hsl and hsv — wikipedia, the free encyclopedia, 2017. (Accedido 11/06/2017).

[6] Wikipedia. Rgb color model — wikipedia, the free encyclopedia, 2017. (Accedido 11/06/2017).

[7] Wikipedia. Sobel operator — wikipedia, the free encyclopedia, 2017. (Accedido 09/06/2017).