Automatic extraction of biomarkers from biopsy images Annotation tool EllipseLabelimg

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Josemaría Tena Martínez

In partial fulfilment

of the requirements for the degree in

Telecommunications Technologies and Services Engineering: Audiovisuals Systems

Advisor: Ferran Marqués Acosta

Barcelona, August 2020

Abstract

This End-of-Grade Project aims to make a system that allows to help in the annotation of tissue biopsy images. Firstly, a program detects the cells in the picture. This annotation must be corrected by a doctor in order to obtain a trustworthy labelling of the image. Therefore, the annotation tool allows to correctly annotate the cells, remove those that have been generated incorrectly and add any missing cells.

We decided to make the tool with the MATLAB GUI, but we verified that it could not be done without the necessary license. We determined to modify an open source program, written in Python, so that it could be used by anyone. Then, I made the connectivity between the annotation made by the program and the annotation tool, so that the program could generate the previous annotation and be modified, in addition performs a score of the program's precision.

Resum

Aquest Treball de Fi de Grau pretén realitzar un sistema que ajudi a l'anotació d'imatges de biòpsia de teixit. Primer de tot cal executar un programa que detecti cèl·lules dins l'imatge. Aquesta anotació prèvia ha de ser corregida per un metge per poder obtenir una anotació fidedigna de la imatge. És per això que el programa permet modificar, eliminar i afegir cèl·lules al primer etiquetatge.

Vam decidir fer l'eina amb la GUI de MATLAB, però vam topar amb què no és podia fer sense la llicència necessària. Per tant, vam optar per modificar un programa de codi obert escrit en Python perquè el pogués utilitzar qualsevol persona.

Posteriorment vaig realitzar la connectivitat entre l'anotació realitzada pel programa i el programa d'anotació, per tal què aquest pogués generar l'anotació prèvia i ser modificada.

Resumen

Este Trabajo de Fin de Grado trata de realizar un sistema que permita ayudar en la anotación de imágenes de biopsia de tejido. Para ello primero se ejecuta un programa que detecta las células en la imagen. Esta anotación previa debe ser corregida por un médico para poder obtener la anotación correcta de la imagen. Por ello el programa permite anotar correctamente las células, eliminar las que se han generado mal y añadir alguna que falte.

Decidimos hacer la herramienta con la GUI de MATLAB, pero comprobamos que no se podía realizar sin la licencia necesaria. Optamos por modificar un programa de código abierto, escrito en Python, para que lo pudiera utilizar cualquier persona. Posteriormente realicé la conectividad entre la anotación realizada por el programa y el programa de anotación, para que este pudiera generar la anotación previa y ser modificada, además de obtener una puntuación de la precisión del programa.

A todos aquellos que creyeron en mí, he llegado hasta el final.

Agradecimientos

Quiero mostrar agradecimiento a mi tutor Ferran Marqués por haberme dado la oportunidad en este proyecto tan interesante y por su supervisión. También dar las gracias al profesor Philippe Salembier y Verónica Vilaplana por dedicarme su tiempo.

A mis amigos, “Puros”, Nira y Anna que me han apoyado durante este largo trayecto, sobre todo, estos meses difíciles de cuarentena. Sin vuestro apoyo no habría llegado hasta aquí. La carrera ha sido más fácil a vuestro lado. Sin su respaldo no habría llegado hasta aquí.

A mis padres por el respaldo, la paciencia y por ir presumiendo de que tienen un hijo teleco.

Por último, agradecer a Luuk por su ayuda en este proyecto.

Historial de revisiones y registro de aprobación

Revision	Date	Purpose
0	10/08/2020	Document creation
1	21/08/2020	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Josemaría Tena Martínez	j.tenamartinez@gmail.com
Ferran Marqués Acosta	Ferran.marques@upc.edu

Written by:		Reviewed and approved by:	
Date	10/08/2020	Date	21/08/2020
Name	Josemaría Tena Martínez	Name	Ferran Marqués Acosta
Position	Project Author	Position	Project Supervisor

Tabla de contenidos

Abstract 2
Resum 3
Resumen 4
Agradecimientos 6
Historial de revisiones y registro de aprobación 7
Tabla de contenidos 8
Lista de figuras 10
Lista de tablas: 11
1. Introducción 12
1.1. Contexto 12
1.2. Objetivos 12
1.3. Requisitos y especificaciones 13
1.4. Métodos y procedimientos 13
1.5. Desviaciones e incidencias 13
1.6. Plan de trabajo 14
1.6.1. Workplan packages 14
1.6.2. Milestones 16
1.6.3. Diagrama de Gantt 17
2. Estado del arte de la tecnología utilizada o aplicada en esta tesis: 18
2.1. Morfología matemática 18
2.1.1. Operadores morfológicos: Dilatación y Erosión 18
2.1.2. Filtros morfológicos: Apertura y Cierre 19
2.1.3. Filtros por reconstrucción 19
2.1.4. Técnicas de segmentación: Watershed 20
2.2. Matlab 22
2.3. Python 23
2.3.1. PyQt5 24
2.4. Programas de anotación 24
2.4.1. LabelImg 25

2.5. F-score 25
2.6. XML 25
3. Metodología / desarrollo del proyecto: 27
3.1. Estudio del programa de detección de células 27
3.2. Programa de anotación con GUI de Matlab 31
3.3. Búsqueda y selección de programas de anotación 32
3.4. Modificación del programa Labelmg 32
3.4.1. Reemplazar la anotación rectangular por una elipse 32
3.4.2. Rotación y traslación de la elipse 34
3.4.3. Guardado en formato XML 36
3.4.4. Anotación a partir de etiqueta 37
3.5. Integración de ambos sistemas 37
3.6. Ejecutable para Windows 38
4. Resultados 39
4.1. Prueba con recorte 39
4.2. Prueba con imagen completa 43
5. Presupuesto 45
6. Conclusiones y future desarrollos: 46
Bibliografía: 47
Apendices: 48
Glosario 70

Lista de figuras

Figure 1. Imagen obtenida por microscopio de un trozo de tejido 12
Figure 2. (a) Diagrama de Gantt de marzo a junio. **(b)** Diagrama de Gantt de junio a agosto. 17
Figure 3. Representación de las señales x e y. En rojo el resultado del supremo entre x e y. En verde, el resultado del ínfimo entre x e y.[3] 18
Figure 4. (a) Imagen original ejemplo. **(b)** Imagen dilatada con elemento estructurante en forma de disco. **(c)** Imagen erosionada de la dilatación con elemento estructurante cuadrado. 19
Figure 5. (a) Imagen con objetos no deseados. **(b)** Imagen erosionada. **(c)** Apertura de la imagen inicial. 19
Figure 6. Ilustración de los pasos empleados en la reconstrucción y su resultado final. El símbolo delta indica la operación de dilatación.[3] 20
Figure 7. Ilustración imagen escala de grises y su relieve. [5] 21
Figure 8. Imagen Lenna **(esquina superior izquierda)**. Dilatación de la imagen **(esquina superior derecha)**. Erosión de la imagen **(esquina inferior izquierda**). Gradiente morfológico de la imagen **(esquina inferior derecha)**. 22
Figure 9. Interfaz de la GUI de Matlab para la generación de interfaces de usuario 23
Figure 10. Muestra de la interfaz de LabelImg y ejemplos de anotación. [10] 25
Figure 11. Recorte del XML empleado en ElipseLabelImg. 26
Figure 12. Recorte imagen ejemplo sin anotar. 27
Figure 13. Representación canal Rojo en escala de grises 27
Figure 14. Imagen simplificada. 28
Figure 15. Representación binaria de los marcadores de núcleo y fondo. 29
Figure 16. Imagen de la segmentación por watershed. 29
Figure 17. Imagen de la segmentación tras la segunda iteración del proceso 30
Figure 18. Imagen anotada 31
Figure 19. Trazado del eje mayor. 33
Figure 20. Trazado del eje menor. 33
Figure 21. Imagen con anotación en forma de elipse 34
Figure 22. Representación de estirar vértice eje menor. 35
Figure 23. Representación de estirar y desplazar vértice eje mayor 35
Figure 24. Imagen con elipse antes de desplazar. 36
Figure 25. Imagen tras desplazar la elipse. 36
Figure 26. Recorte del contenido de una anotación en XML. 37
Figure 27. Pantalla de inicio de EllipseLabelImg 39
Figure 28. Interfaz EllipseLabelImg al abrir una imagen 40
Figure 29. Comparativa entre la anotación realizada en Matlab y la anotación importada en EllipseLabelImg. 40
Figure 30. Muestra al añadir una elipse con etiqueta 'CellN', esta se muestra en rojo y se añade a la lista. 41
Figure 31. Muestra al eliminar una elipse y añadir dos elipses como ejemplo de corrección. 41
Figure 32. a) Máscara de diferencias entre anotaciones. **b)** Máscara de diferencias con elipse eliminada. 42
Figure 33. Captura del programa EllipseLabelImage al cargar una imagen completa 43
Figure 34. Captura de la imagen magnificada. Se realiza un movimiento de uno de los vertices. 43
Figure 35. Máscara de diferencias entre la anotación original y la obtenida por el programa. 44

Lista de tablas:

Table 1. Valores de F-score obtenidos en las imágenes KI67. 44
Table 2. Costes por software 45
Table 3. Costes por Hardware 45
Table 4. Costes por empleado 45
Table 5. Coste total. 45

1. Introducción

1.1. Contexto

Después del cáncer de pulmón, el cáncer de mama es el más común. Según la AECC se diagnosticaron 2.088.849 nuevos casos en todo el mundo en 2018. En España se diagnosticaron 33.307 nuevos casos en 2019. Se estima que la tasa de incidencia en el estado en 2018 es de 138 por cada 100.000 habitantes.[1][2] .

Una de las claves para combatir el cáncer y aumentar su tasa de supervivencia es la detección temprana de la enfermedad. Para ello se requiere un sistema de detección entre la población. La biopsia de tejido permite formular un diagnóstico definitivo. En ella se extrae una porción de tejido para examinar posteriormente en un microscopio por un patólogo.

Este análisis a través del microscopio necesita de recursos médicos. Es por ello que una manera de maximizar la cantidad de diagnósticos y reducir los recursos es la detección de tumores y análisis de las células por imagen. Una vez obtenida la imagen de microscopio, esta se puede tratar para extraer la información necesaria para que posteriormente el especialista pueda realizar un diagnóstico. Este es el principio de este proyecto, análisis de imágenes de tejido para poder ayudar al médico a realizar un diagnóstico.

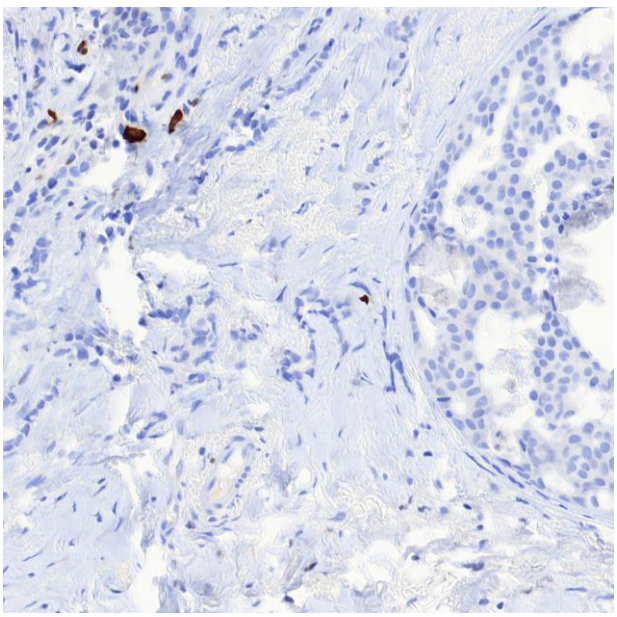


Figure 1. Imagen obtenida por microscopio de un trozo de tejido

1.2. Objetivos

El objetivo de este proyecto es realizar una herramienta de anotación compatible con el sistema de pre-anotación escrito en MATLAB. De esta manera, al tener imágenes anotadas por un doctor, poder mejorar el sistema de anotado de células, a través de parámetros como el F-Score. Esta herramienta de anotación debe ser de código libre para que pueda ser empleada por cualquier persona. Además, debe tener una comunicación de los datos con el entorno de MATLAB.

1.3. Requisitos y especificaciones

Este proyecto requiere:

- Generar datos de la detección de las células en Matlab de tal manera que un programa de anotación pueda interpretarlos.
- Un programa de anotación de código libre que pueda interpretar los datos obtenidos por Matlab y generar una anotación para su posterior corrección.
- El programa debe permitir anotar con la forma de una elipse.
- Es necesario que pueda eliminar anotaciones incorrectas, añadir anotaciones buenas y mantener la compatibilidad.

Las especificaciones de este proyecto:

- Un ordenador con licencia MATLAB, editor de texto e intérprete de Python. Además, debe de poder generar ejecutables.

1.4. Métodos y procedimientos

Para llevar a cabo este proyecto se planteó el siguiente procedimiento:

- Estudio del programa de Matlab para la detección de células.
- Realizar un programa con la GUI de Matlab.
- Encontrar programas de anotación de código libre en Python o Java.
- Modificar el programa de anotación para cumplir las especificaciones.
- Añadir funciones para que el trabajo de anotación sea más cómodo.
- Comprobar el funcionamiento del programa y corregir errores.

Para ello se ha utilizado el programa Matlab y el programa de anotación LabelImg.

1.5. Desviaciones e incidencias

La idea inicial era realizar de forma sencilla una herramienta de anotación basada en Matlab con la GUI del mismo programa. A partir de esta, generar un ejecutable Matlab para que se pudiera utilizar sin necesidad de licencia e instalado de Matlab. Comprobé que no se podía realizar con la herramienta del toolbox de Matlab ya que era necesaria obtener la licencia. Por ello se optó por buscar otros programas de anotación de código libre y modificarlo para que cumpliera con los requisitos. Entre los distintos programas analizados se eligió LabelImg escrito en Python. Debido a mi escasa noción en la librería PyQT5 de Python que emplea este programa, se ha dificultado las tareas a realizar.

1.6. Plan de trabajo

1.6.1. Workplan packages

<table>
<tr><td>Project: Análisis del código Matlab</td><td colspan="2">WP ref: (WP1)</td></tr>
<tr><td>Major constituent: Investigation</td><td colspan="2">Sheet 1 of 1</td></tr>
<tr><td rowspan="2">Short description: Este paquete de trabajo consiste en analizar y comprender el código proporcionado por los tutores.</td><td colspan="2">Planned start date: 23/03/20
Planned end date: 30/03/20</td></tr>
<tr><td colspan="2">Start event: T1
End event: T2</td></tr>
<tr><td>Internal task T1: Depuración del código
Internal task T2: Comprensión de las funciones de morfología empleadas</td><td>Deliverables:

-</td><td>Dates:

-</td></tr>
</table>

<table>
<tr><td>Project: Programa anotador basado en GUI de Matlab</td><td colspan="2">WP ref: (WP2)</td></tr>
<tr><td>Major constituent: Software</td><td colspan="2">Sheet 1 of 1</td></tr>
<tr><td rowspan="2">Short description: Este paquete de trabajo consiste en desarrollar una interfaz gráfica con el uso de la herramienta de GUI de Matlab. Esta interfaz sirve para poder modificar la anotación generada en el código anteriormente mencionado.</td><td colspan="2">Planned start date: 30/03/20

Planned end date: 06/04/20</td></tr>
<tr><td colspan="2">Start event: T1

End event: T2</td></tr>
<tr><td>Internal task T1: Implementación de la App

Internal task T2: Comprobar si se puede generar un ejecutable Matlab</td><td>Deliverables:</td><td>Dates:</td></tr>
</table>

<table>
<tr><td>Project: Alternativas de programas de Anotación openSource</td><td colspan="2">WP ref: (WP3)</td></tr>
<tr><td>Major constituent: Investigation</td><td colspan="2">Sheet 1 of 1</td></tr>
<tr><td rowspan="2">Short description: Este paquete de trabajo consiste en la búsqueda de programas de anotación de código abierto para ser modificados posteriormente.</td><td colspan="2">Planned start date: 06/04/20
Planned end date: 13/04/20</td></tr>
<tr><td colspan="2">Start event: T1
End event: T1</td></tr>
<tr><td>Internal task T1: Búsqueda de programas de anotación</td><td>Deliverables:</td><td>Dates:</td></tr>
</table>

Project: Modificar el programa LabelImg	WP ref: (WP4)	
Major constituent: Software	Sheet 1 of 1	
Short description: Este paquete de trabajo consiste en realizar las modificaciones necesarias del programa LabelImg, para que cumpla con los requisitos del sistema.	Planned start date: 13/04/20 Planned end date: 29/06/20	
	Start event: T1 End event: T7	
Internal task T1: Prueba del programa Internal task T2: Comprensión del funcionamiento Internal task T3: Estudio de la libreria PyQt5 Internal task T4: Anotación en forma de elipse Internal task T5: Modificar guardado en XML Internal task T6: Prueba del programa modificado Internal task T7: Correción de errores	Deliverables:	Dates:

Project: Añadir características al programa modificado	WP ref: (WP5)	
Major constituent: Software	Sheet 1 of 1	
Short description: Este paquete de trabajo consiste en realizar mejoras para el programa de etiquetado.	Planned start date: 29/06/20 Planned end date: 16/07/20	
	Start event: T1 End event: T5	
Internal task T1: Corregir cargado de anotación XML Internal task T2: Mejorar la selección de la elipse Internal task T3: Mejorar movilidad de los vértices Internal task T4: Añadir rotación de la elipse Internal task T5: Añadir color según etiqueta	Deliverables:	Dates:

Project: Conectividad con el programa de pre-anotado	WP ref: (WP6)	
Major constituent: Software	Sheet 1 of 1	
Short description: Este paquete de trabajo consiste en generar un XML con la anotación generada en Matlab, de tal manera que el programa anotador pueda abrir y modificar las anotaciones.	Planned start date: 16/07/20 Planned end date: 23/07/20	
	Start event: T1 End event: T2	
Internal task T1: Implementar función de generado del XML Internal task T2: Comprobar que la anotación es idéntica en ambos lados	Deliverables:	Dates:

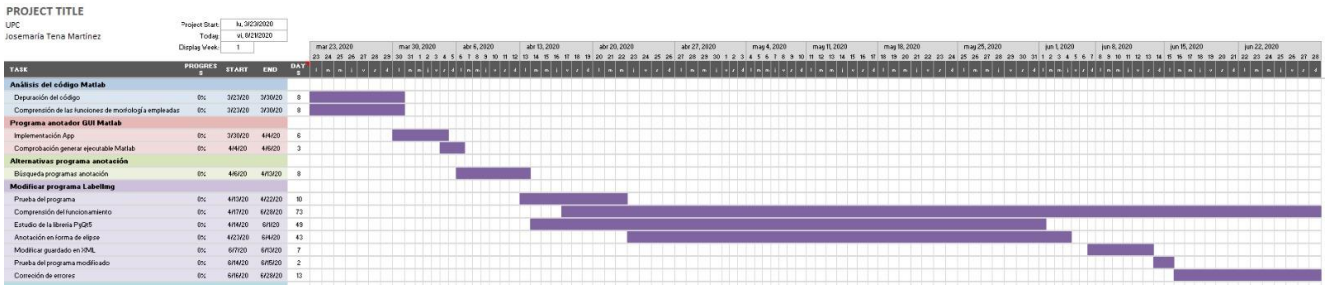
Project: Cargado selectivo de anotaciones	WP ref: (WP7)	
Major constituent: Software	Sheet 1 of 1	
Short description: Este paquete de trabajo consiste en realizar un sistema en el que solo se cargue las elipses visibles en la ventana del programa.	Planned start date: 23/07/20 Planned end date: 06/08/20	
	Start event: T1 End event: T4	
Internal task T1: Añadir botón para carga de anotaciones Internal task T2: Añadir click & drag sobre la ventana Internal task T3: Guardado de la nueva anotación por zona Internal task T4: Marcado de zona anotada y corregida	Deliverables:	Dates:

Project: Documentación	WP ref: (WP8)	
Major constituent:	Sheet 1 of 1	
Short description: Este paquete de trabajo consiste en las tareas de redacción de memoria del TFG.	Planned start date: 23/07/20 Planned end date: 21/08/20	
	Start event: T1 End event: T1	
Internal task T1: Redactar el documento Degree_Thesis.doc	Deliverables:	Dates:

1.6.2. **Milestones**

WP#	Task#	Short title	Milestone / deliverable	Date (week)
2	3	Programa anotación	Interfaz Matlab	06/04/20
4	7	Programa anotación	Programa anotación	29/06/20
5	5	Añadir características	Programa anotación	16/07/20
6	2	Integración sistemas	Programa escritura XML	29/07/20
7	4	Cargado selectivo	Programa anotación	06/08/20
8	1	Documentación	Degree thesis	21/08/20

1.6.3. Diagrama de Gantt



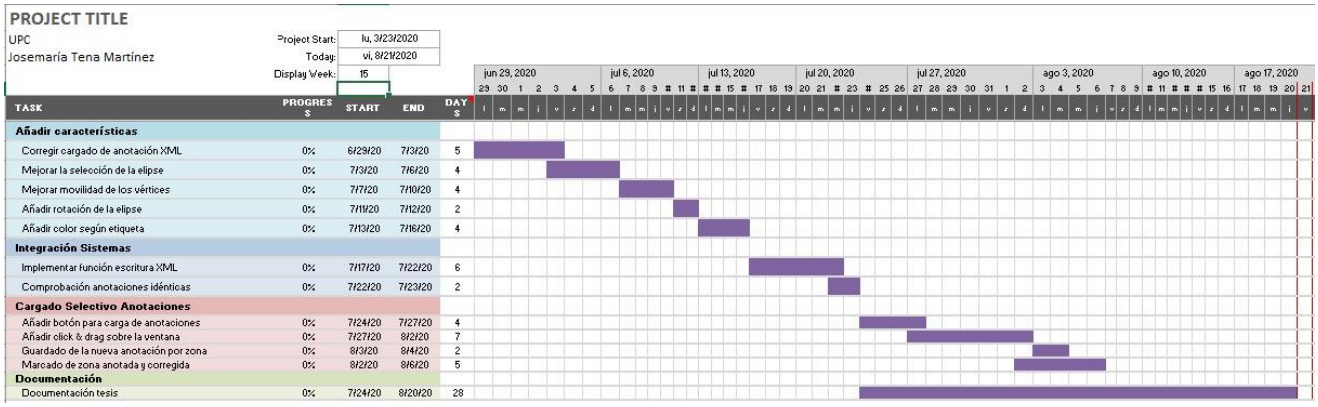


Figure 2. (a) *Diagrama de Gantt de marzo a junio.* ***(b)*** *Diagrama de Gantt de junio a agosto.*

2. Estado del arte de la tecnología utilizada o aplicada en esta tesis:

En este apartado se va a explicar las nociones básicas de procesado de imagen y las herramientas empleadas en este proyecto. Se hace una breve explicación de morfología matemática y segmentación por watershed, además de ver los entornos de programación usados y los programas de anotación.

2.1. Morfología matemática

En el procesado de imagen existe varios modelos de tratamiento de la imagen. La morfología matemática es uno de ellos. Se basa en la comprensión de la imagen como un conjunto de valores en los que se preserva un orden, continuidad, forma, tamaño y conectividad. Dada una superposición de imágenes, observamos que la linealidad no es útil en la composición de estas. No queda claro cuando la imagen está encima de la otra. Este orden parcial da la entrada al uso de las operaciones de supremo e ínfimo. El supremo da como resultado la superposición de elementos que están por encima, pertenecientes a una imagen A si está es mayor que B, o si, dada una zona B es mayor que A, da los valores de B. De forma dual, tenemos la operación del ínfimo. Esta da como resultado la parte menor de la superposición de dos imágenes A y B. En la figura siguiente, para el caso de una dimensión, podemos observar el resultado de aplicar el resultado a dos señales en las que varía el orden. En algunos tramos la señal X es mayor que Y, mientras que en otros X es menor que Y.

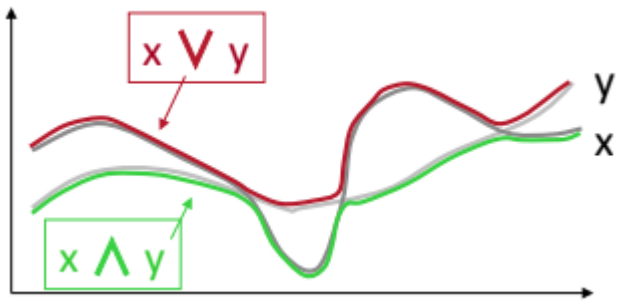


Figure 3. *Representación de las señales x e y. En rojo el resultado del supremo entre x e y. En verde, el resultado del ínfimo entre x e y.[3]*

2.1.1. Operadores morfológicos: Dilatación y Erosión

Una vez visto el concepto de supremo e ínfimo, podemos explicar los operadores morfológicos dilatación y erosión. Una idea intuitiva sobre estos operadores sería imaginar que, dada una imagen, esta es analizada mediante un elemento estructurante, aportándonos así información sobre la imagen.

El elemento estructurante es un conjunto, de forma diversa, en el que dentro de la figura su valor es 0 y fuera del conjunto es -infinito. Estos elementos estructurantes son como unos rastreadores dentro de la imagen.

Dada una imagen y un elemento estructurante, la dilatación es la operación del supremo entre la imagen y el elemento estructurante. Esto quiere decir que, en una imagen, por ejemplo binaria, si el elemento estructurante se sitúa sobre una zona con valores blancos y negros de la imagen, al realizar el supremo, propagará el blanco sobre los valores contenidos dentro del elemento estructurante. De forma dual, la erosión realiza la operación del ínfimo entre la imagen y el elemento estructurante. Siguiendo el ejemplo anterior, el elemento estructurante, propagará el negro sobre los valores de blanco contenidos dentro de la forma del elemento estructurante.

En la siguiente figura observamos como afecta a una imagen estas operaciones. Tenemos una imagen negra con un centro blanco. Al realizar una dilatación con un elemento estructurante en forma de disco, cuando se sitúa sobre el punto blanco, se propaga ese valor sobre en la zona definida por el elemento. Es por ello que en la imagen dilatada podemos observar la forma del elemento estructurante. En el caso de la erosión, ocurre lo contrario, al realizarla sobre la imagen dilatada, con un elemento estructurante cuadrado, vemos que hace retroceder los valores blancos, los reduce. Se da este resultado ya que la erosión propaga el valor más bajo. También se puede apreciar la forma del elemento estructurante, dando como imagen resultante una figura más cuadrada.

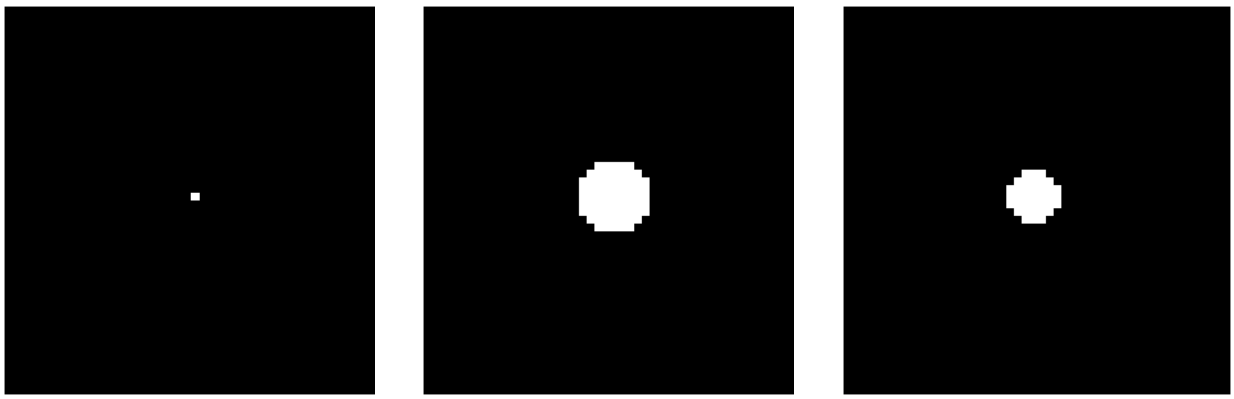


Figure 4. (a) *Imagen original ejemplo.* ***(b)*** *Imagen dilatada con elemento estructurante en forma de disco.* ***(c)*** *Imagen erosionada de la dilatación con elemento estructurante cuadrado.*

A modo de resumen y para un mejor entendimiento de los procedimientos usados, podemos concluir que la dilatación expande los elementos blancos/ contrae los elementos negros y la erosión, expande los negros/ contrae los blancos.

2.1.2. Filtros morfológicos: Apertura y Cierre

Una vez vistos estos dos operadores morfológicos, podemos comprender los filtros morfológicos de apertura y cierre. Una apertura es la dilatación de la erosión. Esto quiere decir que combinamos ambos operadores, realizando primero una erosión con un elemento estructurante y sobre esta imagen resultante, una dilatación con el mismo elemento estructurante. Como resultado obtenemos un filtro, por mucho que se aplique sobre la misma imagen, obtendremos siempre el mismo resultado, sin variar el elemento estructurante. De forma dual tenemos el cierre. Un cierre es la erosión de la dilatación, es decir, se realiza una dilatación y posteriormente se erosiona.

De una forma intuitiva podemos observar que al realizar tanto una apertura como un cierre, las formas pueden preservarse, ya que de la misma manera que se expande los valores en una dilatación, posteriormente se contraen con la erosión. Siempre se preservará aquello que sea mayor que el elemento estructurante. Por lo tanto, las formas menores al elemento estructurante desaparecen. Este es uno de los usos de estos filtros. Veremos que realizaremos estas operaciones para simplificar las imágenes, en el sentido de eliminar objetos que no sean de interés. Si los objetos son más claros que el fondo, con una apertura y un elemento estructurante mayor y con geometría similar al objeto a eliminar, podremos eliminar estos. Con el primer paso de erosión, al expandir los valores más bajos, se eliminarían estos objetos más claros. Al realizar una dilatación posterior, los objetos que han permanecido recuperan sus contornos originales, ya que en el paso anterior han menguado debido a la erosión. En el caso que los objetos que no sean de interés sean más oscuros, podemos realizar un cierre. El procedimiento es dual, al realizar una dilatación, los objetos más pequeños al elemento estructurante desaparecerían al expandir los valores más altos. Posteriormente, con la erosión, se recupera el contorno original de las figuras que han permanecido, ya que estás se han expandido en el paso anterior.

En la siguiente figura podemos ver un ejemplo, donde vemos un objeto que deseamos conservar y unos puntos blancos que queremos eliminar. Para ello, como hemos visto antes, realizamos una apertura con un elemento estructurante en forma de cuadrado, más grande en tamaño que los objetos. Como resultado obtenemos el objeto de interés. En este caso, de análisis de células, en el fondo residen objetos, que no son células y que nos interesa eliminar. Es por ello por lo que usaremos estos filtros, para poder simplificar las imágenes, es decir, eliminar aquello que no nos interese y obtener solo las formas relevantes.

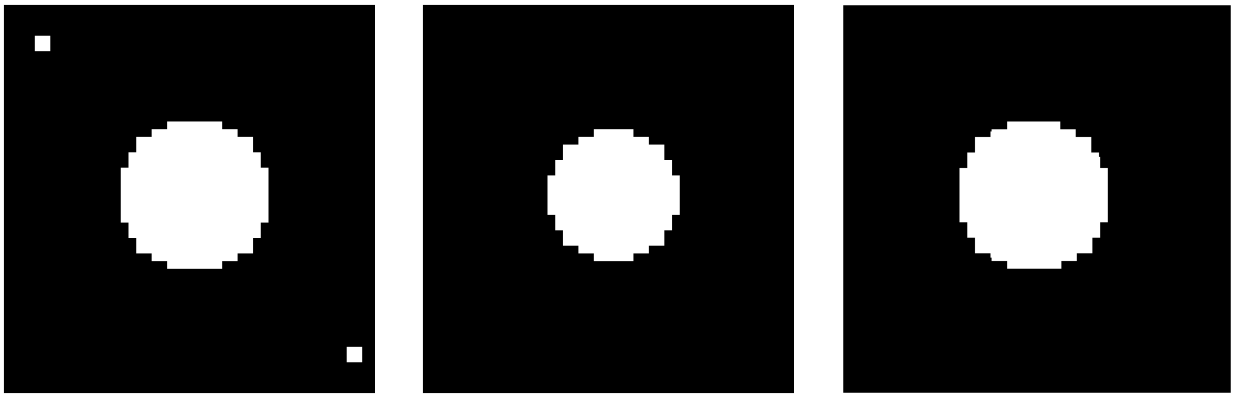


Figure 5. (a) *Imagen con objetos no deseados.* ***(b)*** *Imagen erosionada.* ***(c)*** *Apertura de la imagen inicial.*

2.1.3. Filtros por reconstrucción

En algunos casos, los objetos relevantes y los que queremos eliminar, no se diferencian por su nivel de gris o forma. Una técnica que se puede usar para conservar solo los objetos que nos interesan, es usando una reconstrucción geodésica. Esta nos permite usar una imagen a la que llamaremos *marker*, que usaremos como plantilla de formas que queremos preservar. Sobre la imagen colocamos el marker, haciendo que las formas de este coinciden con la localización de los objetos deseados. A continuación, se realiza un

proceso iterativo en el que el marker se dilata por un elemento estructurante pequeño y se vuelve a colocar sobre la imagen, realizando la operación supremo entre el marker y la imagen. Finalmente, nuestra imagen marker, contiene como resultado la reconstrucción geodésica, solo permanece los objetos que hemos considerado marcar. En la siguiente figura podemos ver la ilustración de estos pasos mencionados. Se tiene una imagen X con objetos blancos y una imagen marker Y con una forma determinada. Esta forma al superponer con la imagen X, "toca" unos determinados objetos y otros no. Observamos que a medida que realizamos dilatación y supremo, el marker toma la forma de los objetos marcados. La imagen Y resultante es estos objetos de interés.

De manera análoga tenemos la reconstrucción dual, erosionando el marcador. Esta técnica se emplea para obtener los elementos conectados. Realizando esta operación, el resultado es imagen en la que solo se tiene en cuenta la forma de los objetos, si estos tienen partes internas de diferente valor, no se conserva, solo se mantiene la forma de estos.

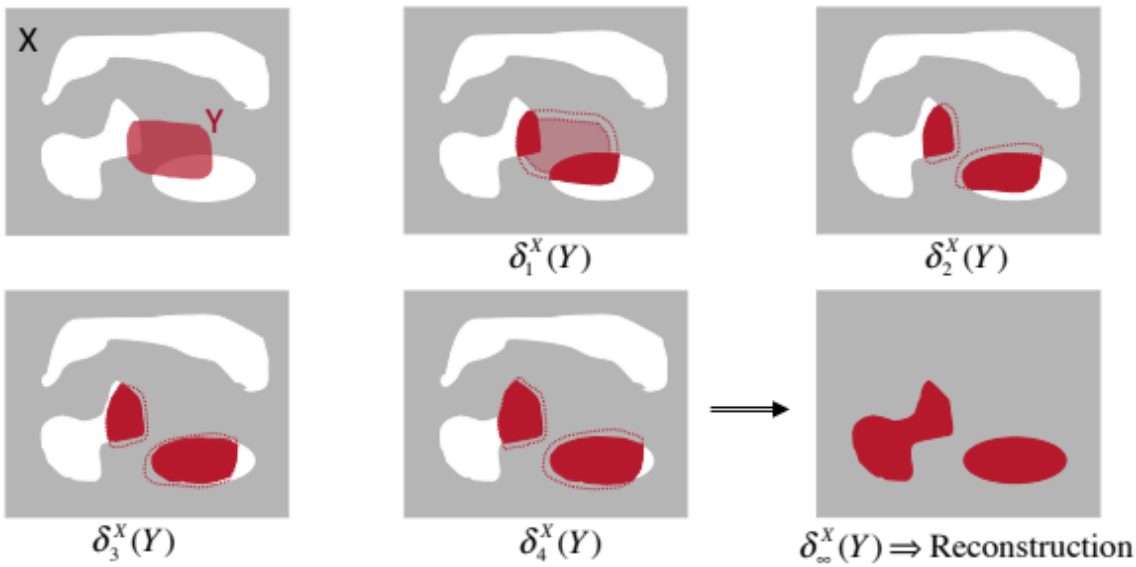


Figure 6. *Ilustración de los pasos empleados en la reconstrucción y su resultado final. El símbolo delta indica la operación de dilatación.[3]*

Como podemos observar, los marcadores tienen que estar relacionados de alguna manera con la imagen a la que se le aplicará la reconstrucción. Una técnica habitual para obtener estos marcadores es utilizar aperturas y cierres de la imagen como marcadores. Este tipo de reconstrucción se denomina reconstrucción por apertura, por erosión, etc. Veremos que en el procedimiento de análisis de las imágenes de células, se emplea este tipo de reconstrucciones, usando como imagen marker una operación morfológica de la imagen original.

2.1.4. Técnicas de segmentación: Watershed

Dentro de las diversas técnicas de segmentación, en este proyecto se hace uso del *watershed*. La segmentación consiste en dividir la imagen en regiones que comparten una característica común, ya sea conjuntos de pixeles con un valor similar u objetos. De esta forma podemos dar como una etiqueta a diferentes regiones de la imagen.

La técnica del watershed, traducido como cuenca, interpreta las imágenes como relieves, donde hay máximos y mínimos, estos últimos serían las cuencas. Si analizamos una imagen en escala de grises por sus valores, podemos visualizarla como un relieve 3D en que los valores de gris indican las diferentes alturas. En la siguiente figura podemos apreciar un ejemplo de la vista de relieve de una imagen en escala de grises.

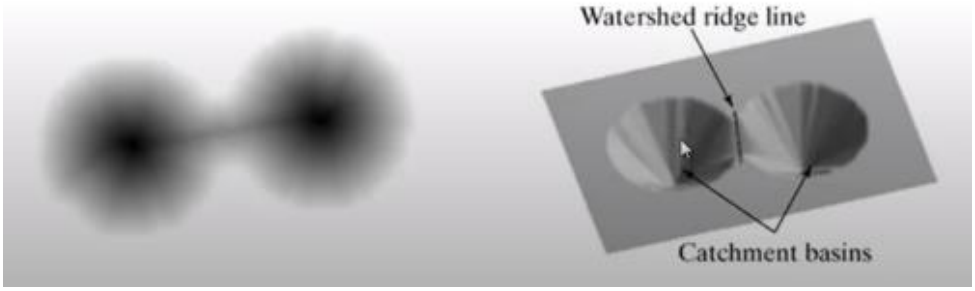


Figure 7. *Ilustración imagen escala de grises y su relieve. [5]*

Dada esta interpretación de la imagen, la técnica consiste en una inundación de este relieve. El agua entra por las cuencas y a medida que inundamos el relieve, esta va creciendo por las diferentes cuencas. Cuando el agua procedente de diferentes cuencas se toca, se forma un dique. Sobre la imagen se realiza el siguiente procedimiento. Se interpreta como origen de la región los mínimos de la imagen. A cada iteración se incrementa el valor del nivel de gris y los píxeles vecinos con ese nivel se unen a la región. De esta manera la región va creciendo hasta que se encuentra con las otras regiones procedentes de otras cuencas. Es por ello esta interpretación de ir inundando, el “agua” entra por los mínimos y va subiendo por el relieve, obteniendo la imagen segmentada por los diques formados.

Para un buen desempeño de esta técnica, es necesario indicar por dónde queremos que se inicie la inundación. Por eso veremos que en este proyecto se usan funciones que imponen los mínimos, o que, siguiendo la interpretación de relieve, realizan agujeros por donde tiene que entrar el agua. El resultado es una imagen segmentada como deseamos al escoger estos mínimos. Como hemos observado, es necesaria una imagen en escala de grises. Es común el uso del gradiente morfológico como imagen a segmentar. Este se obtiene realizando la diferencia entre la dilatación y la erosión de una imagen, usando el mismo elemento estructurante. Así conseguimos una imagen con los contornos de los objetos, ya que la diferencia es el desplazamiento del contorno producido por las operaciones de dilatación y erosión. Como podemos observar en la siguiente imagen, realizando la resta de la dilatación y la erosión, obtenemos los contornos de la imagen original. Esta imagen resultante se puede interpretar como relieve y realizar un watersehd para segmentar diferentes regiones. Por ejemplo, el motivo y el fondo, imponiendo mínimos en el fondo y en la modelo.

***Figure 8**. Imagen Lenna **(esquina superior izquierda)**. Dilatación de la imagen **(esquina superior derecha)**. Erosión de la imagen **(esquina inferior izquierda**). Gradiente morfológico de la imagen **(esquina inferior derecha)**.*

2.2. Matlab

Matlab es un entorno de desarrollo de cálculos, donde se interpretan fácilmente vectores y matrices, junto con un lenguaje de programación. Este programa fue desarrollado por MathWorks y Cleve Moler. Entre sus prestaciones encontramos la fácil manipulación de datos, el desarrollo de algoritmos y una interfaz de usuario llamada GUI.

Dentro de este proyecto, se usa este programa para la implementación de un código para análisis de imagen. En Matlab se pueden hallar diversas funciones relacionadas con el procesado de imagen, entre ellas, funciones pertenecientes a la morfología matemática. La gran variedad de funciones y la sencillez con la que se puede manipular las imágenes, hacen de Matlab un entorno ideal para el desarrollo de programas de procesado de imagen. Dado que el proyecto se basa en la realización de operaciones morfológicas sobre la imagen, es una buena opción el uso de este lenguaje.

La herramienta GUI permite desarrollar una interfaz de usuario, sin necesidad de escribir todo el código. Esta herramienta otorga un entorno en el que se pueden añadir elementos

de forma interactiva, visualizando la interfaz generada. Dada la facilidad con la que se puede realizar estas interfaces, es una buena opción para realizar el programa de anotación. Además, no habría problemas de compatibilidad entre entornos, ya que todo estaría en el mismo.

Matlab permite la generación de ficheros ejecutables .m para que no sea necesario tener el entorno, solo un intérprete que se proporciona de forma gratuita.

Una desventaja de este entorno es la licencia ya que es de pago.

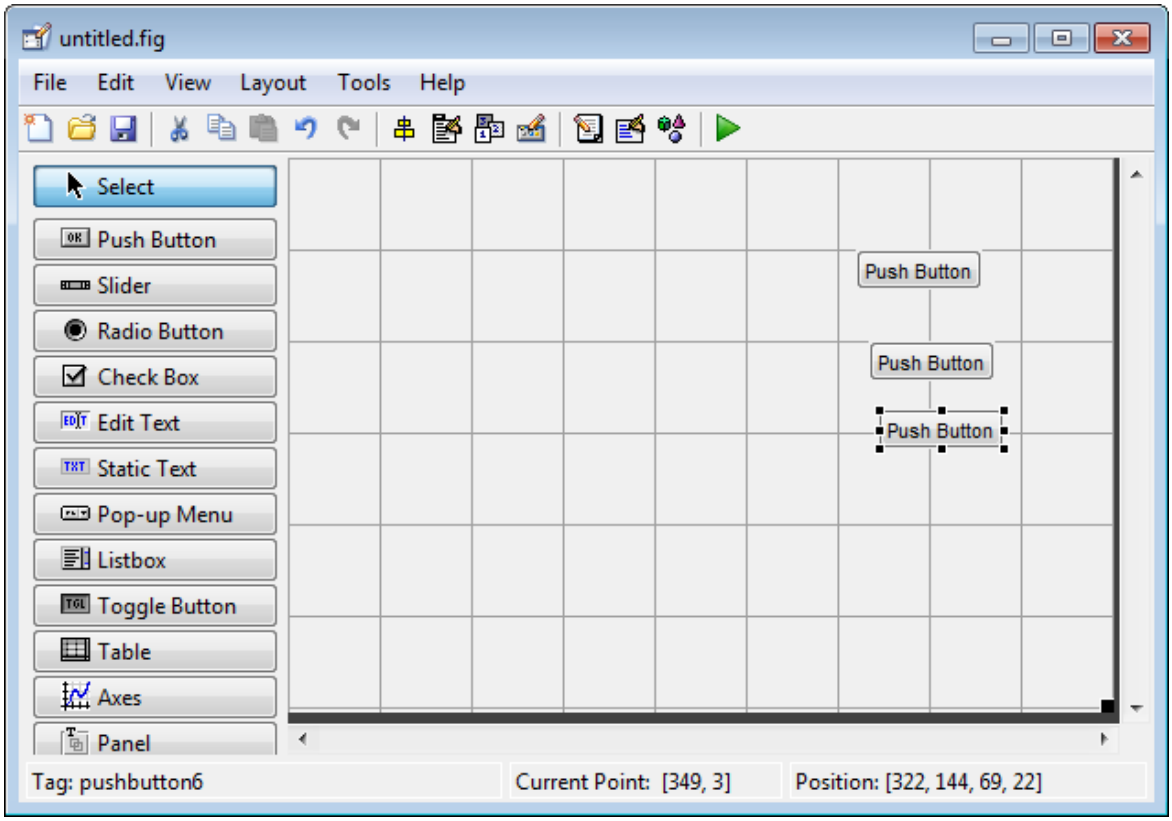


Figure 9. *Interfaz de la GUI de Matlab para la generación de interfaces de usuario.*

2.3. Python

Python es un lenguaje de programación con el objetivo de ser fácil de interpretar por el usuario. Esto lo convierte en un lenguaje de alto nivel, las instrucciones son intuitivas para el usuario, al contrario de un lenguaje como C. Además de esta legibilidad, el lenguaje soporta diferentes tipos de programación, ya que se puede usar como lenguaje estructurado u orientado a objetos. Dada su versatilidad, se han realizado diversos módulos que permiten usar este lenguaje en todo tipo de ámbitos, desde la programación de juegos hasta poder realizar procesado de imagen. De hecho, una alternativa gratuita a Matlab es usar Python y la librería numpy, que contiene funciones para el tratamiento de matrices y vectores, funciones matemáticas, etc.

2.3.1. PyQt5

Una de las librerías más usadas para el desarrollo de interfaces de usuario en Python es la librería PyQt5. La sencillez del código y el uso de funciones para generar interfaces permite un fácil acceso al desarrollo de programas. La librería dispone de muchas funciones para usos muy específicos.

En este proyecto me he centrado en las funciones de la clase QPainter que está destinada al dibujo dentro de las ventanas. Esta clase proporciona las funciones necesarias para dibujar formas sobre las imágenes contenidas en una ventana, en este caso, realizar anotaciones sobre imágenes de células. Las funciones se encargan de toda la parte visible por el usuario, trazado de figuras, colores, sin interferir o modificar la imagen, solo proporcionando una vista de los elementos al usuario.

2.4. Programas de anotación

Los programas de anotación son interfaces que permiten al usuario añadir etiquetas sobre las imágenes. Un caso de uso actual es etiquetar estas imágenes para poder entrenar redes neuronales. De alguna forma debemos establecer qué vemos en la imagen. Por ejemplo, si es una toma de carretera con coches y queremos detectarlos, debemos generar una anotación sobre los coches, indicando que ese conjunto de pixeles dentro de la anotación son un coche. De esta manera generamos lo que se denomina ground truth. Es la forma de saber qué hay en la imagen.

En este proyecto, es necesario anotar las imágenes de células proporcionadas, ya que estas no tienen una anotación. De cara a futuros proyectos en el que se realicen la detección con una red neuronal, es necesario un ground truth. Como he mencionado, este sería el etiquetado de regiones de la imagen como células. Por todo ello, se requiere un programa de anotación, para poder generar esta base de datos.

En internet podemos hallar diversos programas de pago y de código libre. En función de las necesidades, tipos de imagen y presupuesto, escogeremos uno u otro programa. Matlab incorpora una herramienta de etiquetado, pero es necesario tener la licencia. Para el proyecto me he centrado en buscar programas que fuesen de código abierto. Observé que la mayoría estaban escritos en Python. Como requisito se decidió que la forma de la etiqueta debía ser elíptica, dada la forma de las células. Muchos de estos programas encontrados, tenían la posibilidad de anotar con rectángulos y de forma libre, algunos con circunferencias, pero sin llegar a ser elipses. El programa desarrollado en este proyecto permite la selección en forma elíptica, como principal característica y se puede decir que, si no es el único, es uno de los pocos que permite esta anotación.

Programas vistos como SlideRunner [6], orientado a anotación de imágenes de células, no incorpora esta funcionalidad. A favor, es un programa muy completo con muchas funcionalidades, como permitir diferentes anotaciones por usuario o herramientas de selección basadas en watershed para seleccionar regiones de forma libre, respetando los contornos de dichas regiones.

Para la realización de este proyecto, encontré el programa LabelImg y por sus características decidimos modificarlo para que pudiera cumplir con los requisitos. Entre sus atributos encontramos una interfaz sencilla y simple.

2.4.1. LabelImg

Desarrollado por Tzutalin, este programa permite hacer selecciones en forma rectangular. Podemos encontrar y descargar el programa través de GitHub. La interfaz es sencilla de usar y como exportación de los datos, guarda las anotaciones en un fichero XML. En la página de GitHub podemos observar que hay varios contribuidores, aportes, foro con varias dudas resueltas. Todo ello me indicó que podría encontrar todo tipo de ayuda en cuanto comprensión del código, variantes hechas por otros usuarios, solución de errores, etc. Por todo ello se decidió modificar este programa, cambiando la selección rectangular por la elíptica y manteniendo tanto la interfaz como el guardado en XML. [7]

Figure 10. *Muestra de la interfaz de LabelImg y ejemplos de anotación. [10]*

2.5. F-score

El F-score es una medida de exactitud de un test. Este se calcula a partir de la precisión y la exhaustividad. Los verdaderos positivos son los elementos que pertenecen a la hipótesis asignada. Se detectan una serie de elementos bajo una hipótesis, los que no pertenecen a esa hipótesis, son falsos positivos. La precisión es el ratio entre verdaderos positivos y positivos detectados. La exhaustividad es el ratio entre verdaderos positivos y verdaderos. En el ejemplo de este proyecto, supongamos una imagen anotada con 10 células. Tras la corrección, el patólogo ha encontrado 12 células. Además de perder 2 células, una célula que hemos indicado como tal, no lo es. Por lo tanto, de todas las células detectadas, 10, 9 han resultado ser correctas. Esto nos da una precisión del 90%. En cambio, no hemos encontrado todas, sino que hemos encontrado 9 de 12 que hay en realidad, esto nos indica una exhaustividad del 75%. El F-score es una medida combinada de ambas que se calcula como: $2 \times \frac{(Precision \times Recall)}{Precision + Recall}$. En el ejemplo que he expuesto obtenemos un F-score del 82% aproximadamente.

2.6. XML

Es un lenguaje para compartir datos estructurados entre diferentes máquinas. La manera de estructurar estos datos es en forma de árbol, se puede componer por elementos que a

su vez están compuestos por otros. Esto permite implementar de forma sencilla y rápida lectores de este tipo de documento, accediendo a los campos necesarios y exportan la información necesaria. Por ello admite el intercambio de datos entre diferentes plataformas. Dentro de los documentos encontramos las etiquetas, indicadas como <*texto*>, dónde *texto* es el objeto al que tratamos acceder. Este objeto puede contener otras etiquetas, dando la estructura de árbol y accediendo a diferentes objetos. Los datos van entre el inicio de la etiqueta y su final, indicado como </*texto*>. En la siguiente figura podemos observar un ejemplo de fichero XML donde los datos están estructurados. Se trata de una porción del fichero XML generado por el programa ElipseLabelImg, desarrollado en este proyecto. Podemos observar los campos definidos por las etiquetas. Al tratarse de una imagen, tenemos campos, necesarios para el intercambio de datos, como el ancho y largo, el directorio donde está y en *object* encontramos las anotaciones. Estas contienen el nombre de la etiqueta y las coordenadas sobre la imagen en las que está ubicada la anotación.

```
<annotation>
    <folder>ICS</folder>
    <filename>recortePrueba</filename>
    <path>C:\Users\Josemar&#237;a
        7\Documents\UPC\TFG\ICSrecortePrueba</path>
    <source>
        <database>Unkwown</database>
    </source>
    <size>
        <width>189.0</width>
        <height>299.0</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>Cell</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>0</xmin>
            <ymin>44</ymin>
            <xmax>58</xmax>
            <ymax>82</ymax>
            <xpoint3>15</xpoint3>
            <ypoint3>84</ypoint3>
            <xpoint4>41</xpoint4>
            <ypoint4>42</ypoint4>
        </bndbox>
    </object>
```

Figure 11. *Recorte del XML empleado en ElipseLabelImg.*

3. Metodología / desarrollo del proyecto:

3.1. Estudio del programa de detección de células

Para la realización del proyecto, primero analicé el código escrito en Matlab proporcionado por Philipe Salambier, para el análisis de la imagen y detección de las células. Las imágenes empleadas para este programa son las obtenidas por la proteína KI67. Se emplea la técnica de watershed como principal herramienta de análisis.

En estas imágenes podemos observar que las células tienen un tinte azulado sobre el fondo y las células cancerígenas, un tinte marrón. Es por ello que primero se toma el canal rojo de la imagen, de esta manera obtenemos una imagen interpretable como una escala de grises. En ella quedan resaltas las células, más oscuras que el fondo, dado su que su valor RGB predomina más el azul que el rojo, por lo tanto, tiene valores inferiores.

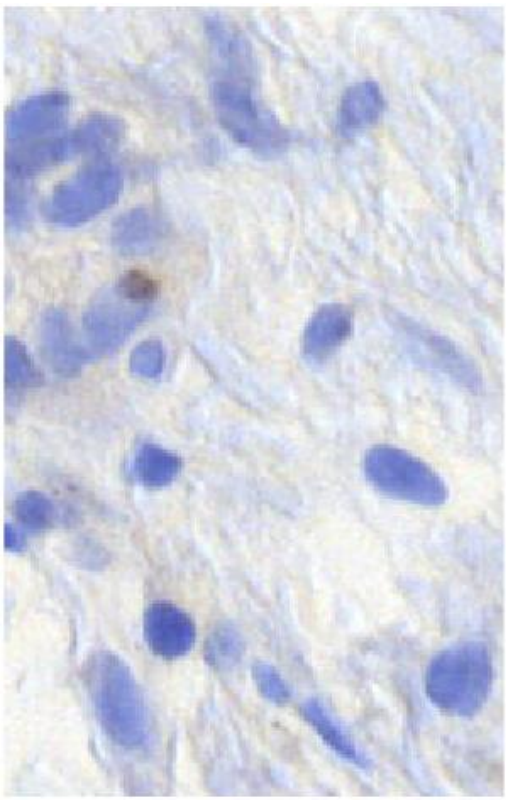


***Figure 12**. Recorte imagen ejemplo sin anotar.*

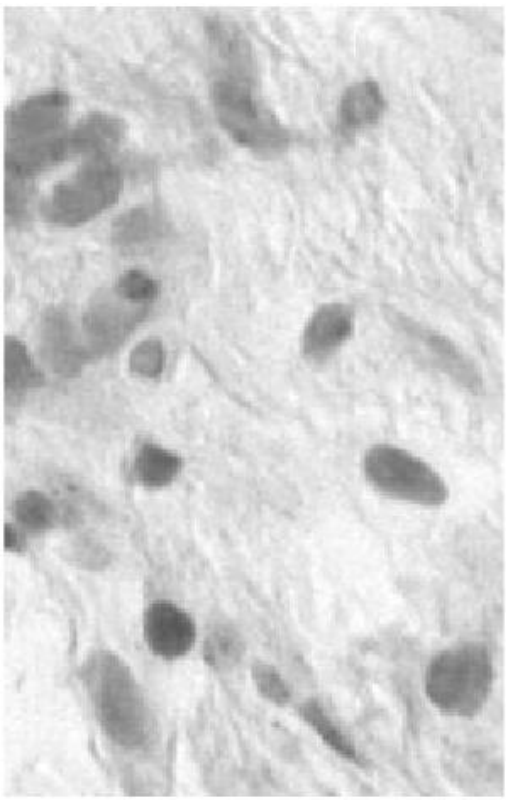


***Figure 13**. Representación canal Rojo en escala de grises*

Seguidamente se realiza una reconstrucción por apertura para simplificar la imagen, eliminar elementos pequeños que no son células. Se realiza una apertura con un elemento estructurante con forma de disco y se hace reconstrucción con la imagen obtenida de la apertura. Después se realiza un cierre con un elemento estructurante más pequeño y se obtiene la reconstrucción usando como marcador el cierre sobre la reconstrucción por apertura. De esta manera obtenemos una imagen simplificada donde se elimina el ruido del fondo y se preservan las células.

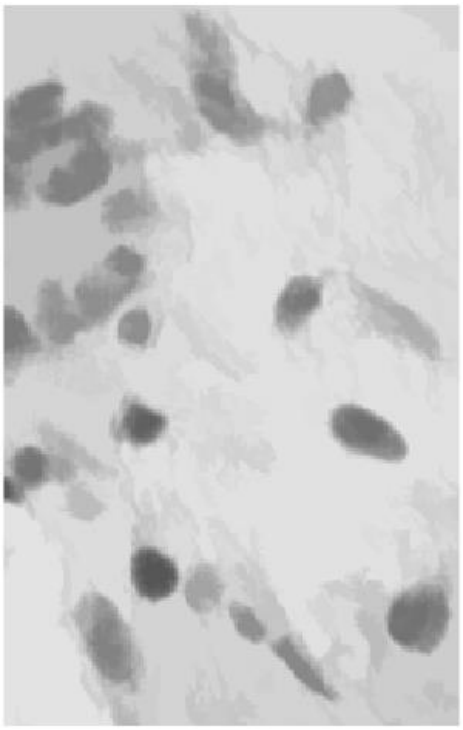


Figure 14. *Imagen simplificada.*

Después de la simplificación, se procede a encontrar los marcadores para el watershed. Primero buscamos los marcadores del núcleo de las células definiendo un umbral de contraste. Si está por encima del umbral, se toma el valor ‘1’ (blanco) y si está por debajo ‘0’ (negro). Con esta operación, al ser el fondo menos contrastado, se queda negro mientras que los objetos con más contraste, las células, quedan blancas. Una vez obtenido el marcador del núcleo, se trata de obtener un marcador del fondo. Para ello se obtiene los mínimos de la imagen a partir de un umbral y se hace un watershed, obteniendo así un marcador de las regiones del fondo. Finalmente se hace un marcador combinado sumando ambas imágenes.

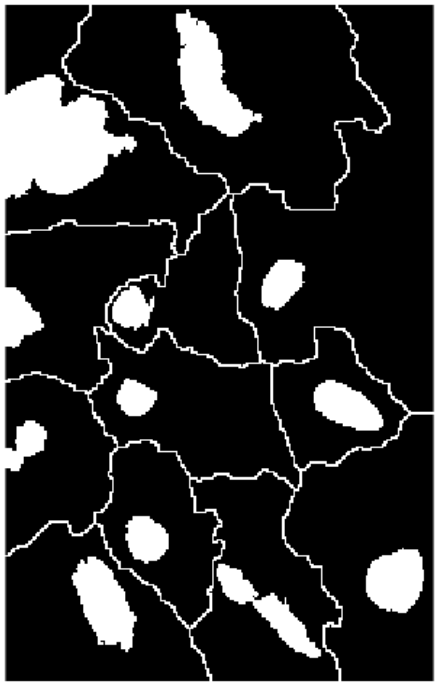


Figure 15. *Representación binaria de los marcadores de núcleo y fondo.*

Una vez llegados a este punto, se procede a realizar el watershed del gradiente de la imagen. El gradiente se obtiene como la diferencia entre la dilatación y la erosión de la imagen simplificada, preservando así los contornos. Se impone los mínimos sobre el gradiente usando la máscara combinada anterior, de esta forma solo se preservan los mínimos del gradiente marcados. Posteriormente se aplica el watershed sobre esta imagen y se eliminan las regiones del fondo, con una reconstrucción usando el marcador de fondo, quedando solo las de las células.

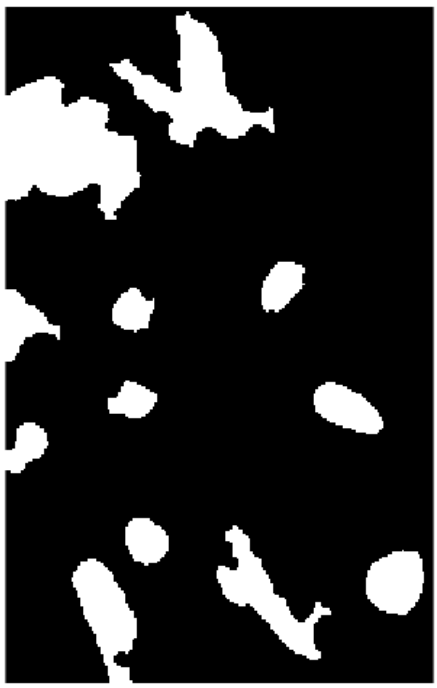


Figure 16. *Imagen de la segmentación por watershed.*

Posteriormente se realiza el mismo procedimiento con elementos estructurantes más pequeños y umbrales inferiores, con el objetivo de recuperar células pequeñas que se hayan eliminado durante el proceso. Se juntan ambas imágenes obteniendo la segmentación definitiva.

Figure 17. *Imagen de la segmentación tras la segunda iteración del proceso*

Finalmente se realiza un análisis sobre los objetos obtenidos, extrayendo características de área, eje mayor, menor, orientación, color... A partir de estos valores se itera sobre los elementos encontrados y se decide si es una célula, además si es una célula que necesita estudio (células marrones). Se calcula la elipse asociada al elemento y se pinta en verde para el caso de células azules y roja para las marrones. Esta es la anotación preliminar obtenida por el programa y que se debe corregir con la herramienta de anotación.

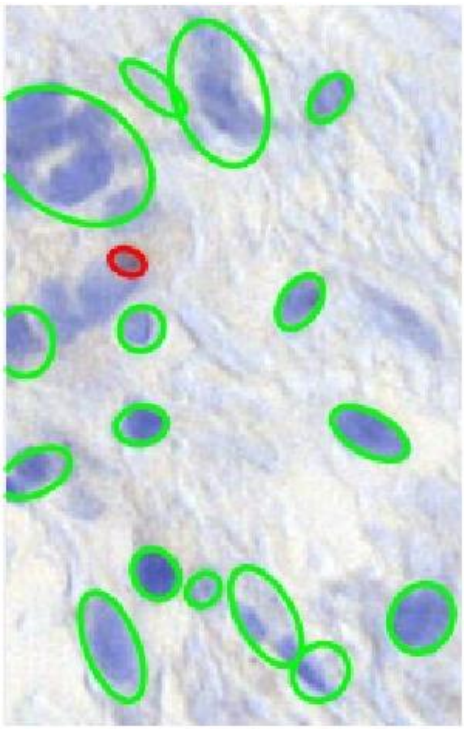


Figure 18. *Imagen anotada*

3.2. Programa de anotación con GUI de Matlab

El objetivo inicial del proyecto era la realización de una herramienta de anotación usando la GUI de Matlab. De esta manera podría hacer un programa con interfaz gráfica de manera sencilla ya que la parte del front-end se realiza con la interfaz de la GUI sin necesidad de escribir mucho código.

Las primeras semanas realicé una GUI que abría la imagen sin anotación y la anotada. Traté de usar diversas funciones de Matlab para poder pintar encima de la imagen anotada y que también pintase la anotación, para poder ser modificada. Observé que no se podía pintar de forma interactiva sobre la imagen debido a un problema de compatibilidad entre las funciones y la GUI.

Los tutores me sugirieron que, dentro de la interfaz, integrase la app “Image Labeler” de la toolbox “Image Processing and Computer Vision”. Esta herramienta permitía anotar imágenes con diferentes formas, excepto una elipse. Por ello me pidieron que modificase el código para que se pudiera hacer elipses, incluir una nueva forma.

Mas tarde comprobé que el código al que intentaba acceder era privado, además de que para incluir Image Labeler en un ejecutable Matlab, era necesaria una licencia por parte del usuario que fuese a utilizar el anotador. Debido a que el médico no tenía Matlab, y con la intención de realizar una herramienta que pudiera utilizar cualquier persona, tomamos la decisión de buscar alternativas. Estas alternativas debían ser programas de código abierto.

3.3. Búsqueda y selección de programas de anotación

Dado el problema con la interfaz realizada en Matlab, buscamos alternativas de código abierto. Entre ellas surgieron los programas ImageJ, SlideRunner y labelImg.

El primero de ellos, ImageJ [8], escrito en Java, admite imágenes en JPG, pero la interfaz no es agradable. No tiene selección de elipse y no genera un fichero con los datos, sino que solo se obtiene una imagen con la anotación pintada. Recordemos que el programa debía tener de alguna manera una comunicación con la anotación realizada en Matlab, poder tomar esta e integrarla.

SlideRunner es un programa de anotación orientado a la anotación de células escrito en Python. Este permite la anotación rectangular, circular y poligonal, indicando los puntos y cerrando la forma. También permite realizar anotaciones hechas por diferentes usuarios. Contiene diversas herramientas, así como una pequeña ventana que guía a través de la imagen de microscopio. El problema inicial de este era que solo permitía imágenes de microscopio, formato .svs y no .jpg. Las anotaciones se guardaban en la misma imagen .svs, no parecía muy adecuado para los requisitos que queríamos cumplir.

Por último, LabelImg, escrito en Python permite solo la anotación rectangular. Admite imágenes JPG, además de tener una interfaz muy sencilla. Los datos de la anotación se guardan en un fichero XML, en él se guardan los dos puntos que forma la diagonal del rectángulo. Los otros dos puntos se calculaban de forma interna al abrir la imagen de nuevo. Este programa fue el que más gustó. Decidimos modificar la anotación rectangular por una elíptica.

3.4. Modificación del programa Labelmg

En este apartado explicaré la modificación realizada en los ficheros que componían el programa LabelImg. Modifiqué los ficheros: labelImg.py canvas.py shape.py pascal_voc_io.py.

3.4.1. Reemplazar la anotación rectangular por una elipse

El primer paso fue reemplazar la anotación en forma de rectángulo por la de una elipse. Para ello primero tuve que analizar el código, ya que no había documentación del mismo. También tuve que mirar cómo se empleaba la librería PyQt5 que es la principal que se usa en este programa. A partir de la comprensión de los diversos ficheros, me dispuse a realizar las modificaciones necesarias, respetando el funcionamiento interno del programa. Cualquier modificación que realizase, podía repercutir negativamente en el funcionamiento de otras partes del programa.

La primera modificación necesaria fue la elipse. La anotación se realiza principalmente en el fichero shape.py, en la función *paint*. Esta función fue reemplaza por *paintEllipse*. En ella modifiqué el código de tal forma que permitiera generar una elipse. Descubrí que la librería QT permitía añadir una forma elíptica al path y dibujar esta, en vez de realizar el path de un rectángulo. No permitía que esta tuviese orientación, es decir, tomaba los puntos de la elipse inscrita en el rectángulo, orientado con los ejes del canvas. La

orientación es totalmente necesaria para poder anotar elipses, ya que permite adecuar mejor la forma y tienen orientaciones diversas. [9]

Además de la dificultad por las funciones de la librería para dibujar una elipse, también había que tener en cuenta todos los eventos del ratón y su gestión. El programa permitía dibujar un trazo de línea recta y a partir de ella se generaba el rectángulo, es decir, se hacía clic en la imagen, se trazaba una recta que seguía el puntero del ratón y finalizaba al hacer clic otra vez. Como solución se planteó como primer paso el poder dibujar el eje mayor de la elipse, una recta, y posteriormente dibujar el eje menor. Una vez obtenidos los 4 puntos, generar la elipse.

El programa permitía hacer 2 clics y entonces generaba la elipse. Como solución modifiqué la función *finalise* en canvas.py. Esta función se ejecuta al terminar la figura, indicando mediante variables de estado que la figura se ha terminado. Añadí líneas y una variable más para indicar que no se ejecutase hasta que hubiese entrado dos veces, es decir, realizar la primera línea y una segunda línea. También era necesario almacenar los puntos para la posterior realización de la elipse. Para ello implementé la función *joinShapes* que une 2 arrays de elementos en uno solo, permitiendo que el atributo points del objeto shape tuviera todos los puntos almacenados.

Una vez permitido el poder realizar dos figuras, el programa generaba dos rectángulos y los interpretaba como una sola figura. Para eliminar su representación modifiqué la función *drawVertex* en shape.py de tal manera que solo generaba los puntos del eje mayor y menor, sin generar el resto de los vértices del rectángulo.

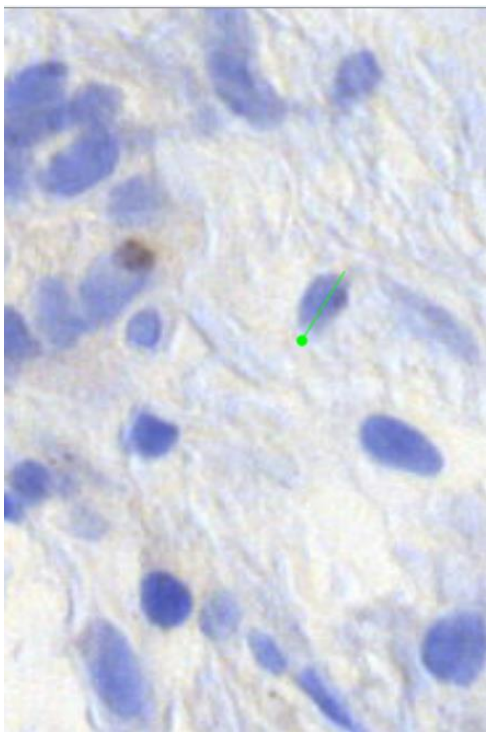


Figure 19. *Trazado del eje mayor.*

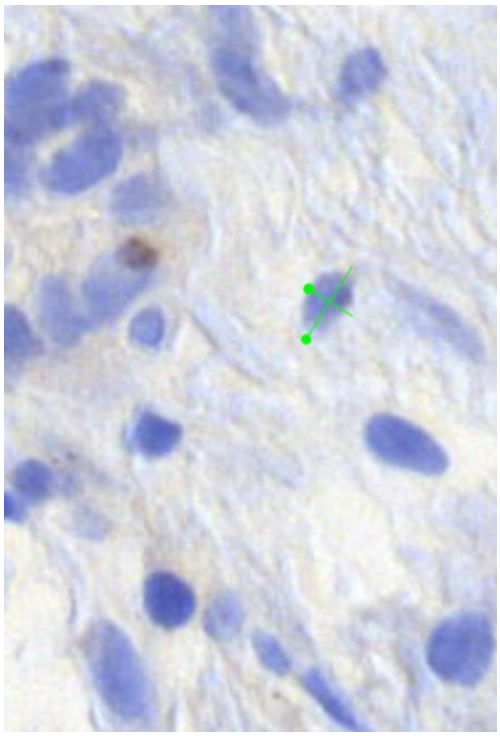


Figure 20. *Trazado del eje menor.*

Después de solucionar los problemas de representación de figuras, realicé la función paintEllipse. En ella se genera la elipse a partir de los puntos indicados. Primero el anotador debe de dibujar el eje mayor y posteriormente el menor. Como es muy difícil acertar la perpendicular, el trazado del eje menor se calcula y dibuja posteriormente de forma correcta. El trazo que realiza el usuario es solo para tener en cuenta el ancho de la elipse, la longitud del eje menor. Por lo tanto, en la función se toma los puntos del eje mayor y a partir de la longitud indicada del eje menor, se calculan los puntos del eje menor realizando una operación matemática de rotación.

Para solucionar el problema de la orientación de la elipse y su dibujado, vi que había unas funciones de la librería que permitían rotar el canvas, es decir, sobre lo que se dibuja. Decidí pintar la elipse en su posición recta, calcular el ángulo entre el eje mayor y el eje de ordenadas y rotar el canvas ese ángulo. Una vez rotado, uso la función de pintado de elipse, como si fuera recta y deshago el cambio de coordenadas, permitiendo así tener las elipses orientadas. [11][12]

Finalmente, con las modificaciones mencionadas, el programa permitía realizar una elipse, primero dibujando su eje mayor y posteriormente el eje menor. Al terminar, haciendo el clic del segundo trazo, se genera una elipse, orientada a partir del eje mayor y con los puntos del eje menor calculado a partir de la longitud determinada por el usuario.

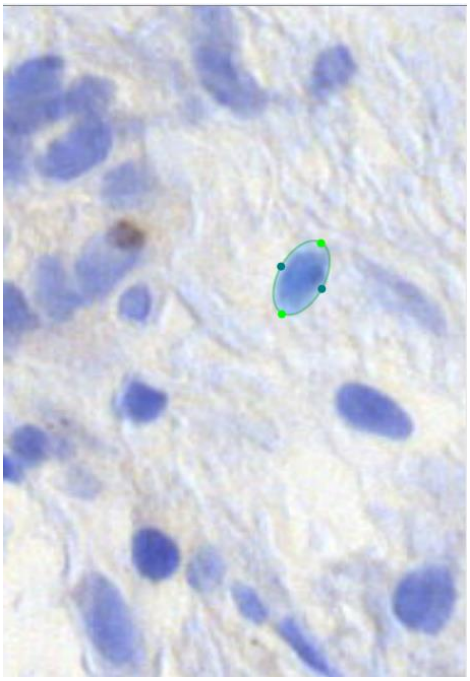


Figure 21. *Imagen con anotación en forma de elipse*

3.4.2. Rotación y traslación de la elipse

Una vez conseguida la elipse, nos pusimos en la perspectiva del anotador y las necesidades que pudiera tener. Al tratar de realizar una anotación de una célula, se

observa que no siempre se acierta a la primera con la forma. Por ello era necesario implementar funciones que permitieran la modificación de la figura.

La primera funcionalidad necesaria era modificar los puntos. El programa permite seleccionar los vértices de la elipse al acercar el puntero a ellos. Modifiqué el código para que esta selección fuese más precisa ya que en el caso de las células, los puntos respecto la imagen son muy pequeños, además de estar juntos. Si seleccionamos cualquiera de los vértices del eje menor, este permite modificar el ancho de la elipse, manteniendo fija su orientación. Sin embargo, al seleccionar un vértice del eje mayor, este permite modificar la longitud de la elipse, además de su orientación. El punto opuesto al seleccionado se mantiene fijo mientras podemos desplazar libremente el extremo seleccionado, modificando así longitud y orientación. Consideramos que este tipo de movimiento era muy útil y cómodo a la hora de anotar células. Para facilitar la distinción entre los vértices pertenecientes al eje mayor y al eje menor, estos están pintados en diferentes colores.

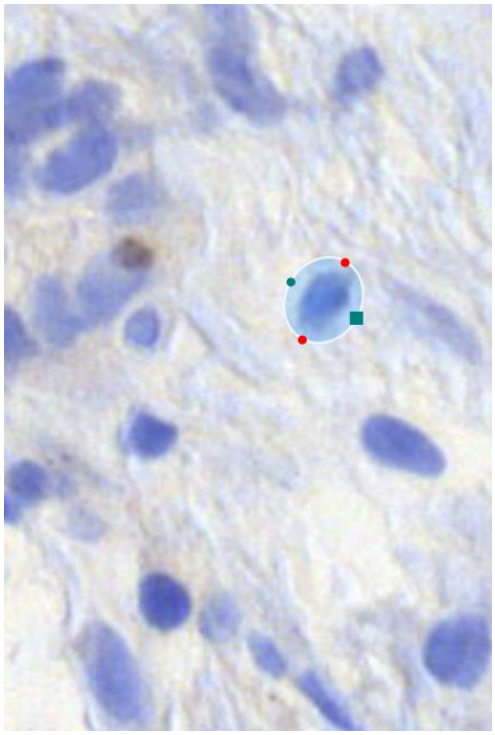


Figure 22. *Representación de estirar vértice eje menor.*

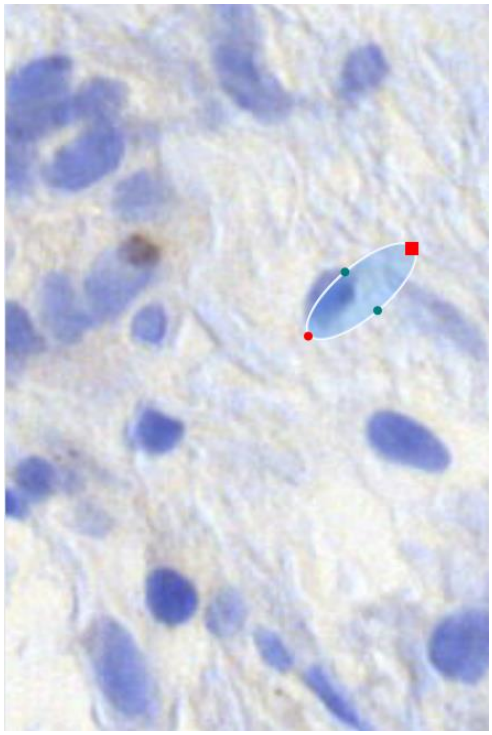


Figure 23. *Representación de estirar y desplazar vértice eje mayor*

El programa permitía coger y arrastrar libremente los rectángulos generados. Al convertirlo en elipse, nos dimos cuenta de que no seleccionaba bien la elipse. Las funciones que calculaban cuando el puntero estaba dentro de la figura fallaban. Teniendo en cuenta como el programa interpretaba el interior, no podía realizar la elipse con las funciones disponibles para que interpretase bien el interior. Como solución decidí emplear una función que permite indicar el path de un polígono, dado los puntos. Modifiqué la función *makePath* en shape.py de tal manera que se genera un rombo en el interior de la elipse, así también hacía más eficiente el funcionamiento del programa, al no tener que realizar

todo el proceso de generación de la elipse descrito antes. El rombo es una buena aproximación de área de selección al anotar, ya que instintivamente colocas el puntero cerca del centro de la figura para desplazarla.

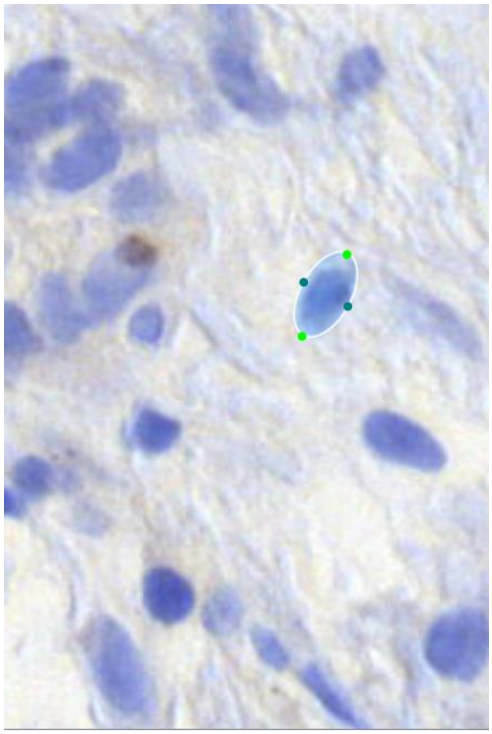


***Figure 24**. Imagen con elipse antes de desplazar.*

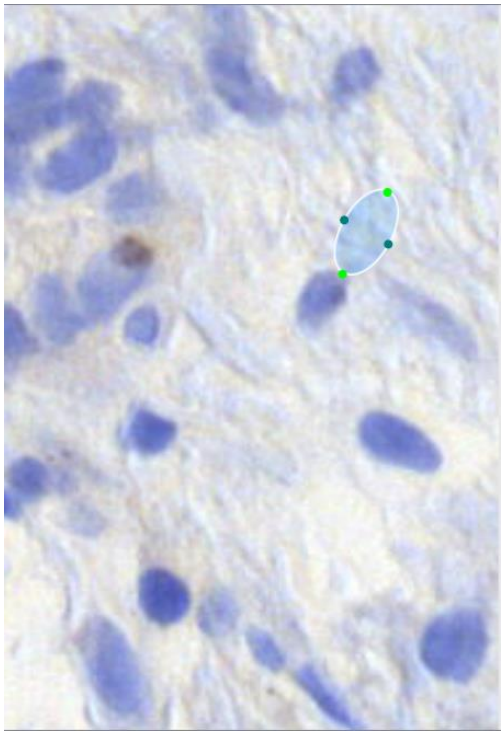


***Figure 25**. Imagen tras desplazar la elipse.*

Como última funcionalidad necesaria, implementé la rotación de las elipses a partir del manejo del teclado. Cuando el usuario selecciona una elipse, esta puede ser rotada, sin mantener uno de los vértices fijo, moviendo ambos. Para rotar modifiqué las funciones *moveBoundedVertex, moveOnePixel y keyPressEvent* en canvas.py. Al pulsar las teclas 'Z' y 'V' estas permiten rotar la elipse en sentido antihorario y horario respectivamente. Además, incluí un movimiento de rotación más suave, más lento, para una mejor precisión en las teclas 'X' y 'C'.[13]

3.4.3. Guardado en formato XML

Como característica importante del programa original, guarda las anotaciones en un fichero XML. Analicé el fichero que se generaba al realizar unas anotaciones, los campos que contenía y el contenido. Observé que guardaba 4 valores de cada figura, al ser un rectángulo, la coordenada x menor, la coordenada y menor y las coordenadas x e y mayor. A partir de estos valores, al volver a abrir la imagen, calculaba los otros puntos y dibujaba el rectángulo. Decidí que, para nuestro caso, tal como estaba implementada la función de dibujo de la elipse, era necesario almacenar las coordenadas x e y de los vértices del eje mayor y del eje menor.

Para ello modifiqué el fichero pascal_voc_io, teniendo en cuenta que debía modificar tanto las funciones de escritura del XML como las funciones de la lectura del fichero. En estas funciones se utiliza la librería XML Tree de Python para generar y leer ficheros XML. Añadí

las variables necesarias para poder almacenar todos los puntos necesarios y en el otro lado, la lectura e interpretación de todos ellos. [14]

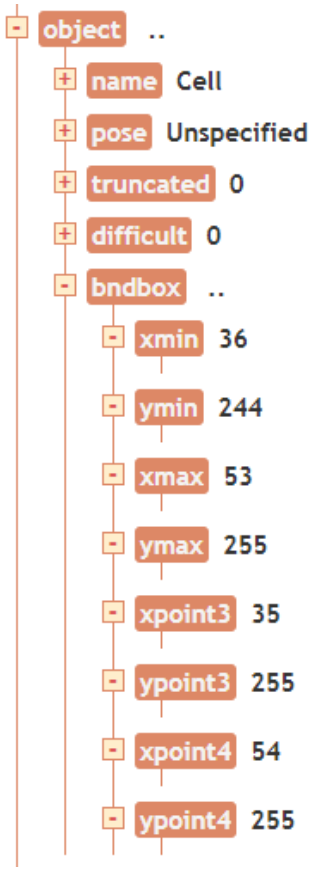


Figure 26. *Recorte del contenido de una anotación en XML.*

3.4.4. Anotación a partir de etiqueta

Dado que en Matlab se generaba la anotación con unos colores determinados y se diferenciaba los tipos de célula, decidí implementar diferentes colores de trazado a partir de la etiqueta. Al realizar una elipse, podemos etiquetarla con una palabra. Cuando la etiqueta es 'Cell' esta se pinta de color verde. Al etiquetar una célula marrón, poniendo de nombre 'CellN' esta se pinta de color rojo, dando concordancia a la anotación realizada en Matlab.

3.5. Integración de ambos sistemas

Una vez terminado el lado del anotador, me dispuse a realizar la integración de ambos sistemas. A partir de la anotación realizada en Matlab, era necesaria la realización de un fichero XML con los puntos de los vértices de la elipse. De esta forma, al abrir el programa anotador y seleccionar la imagen, se cargaría la anotación realizada en Matlab como si se hubiera hecho en el programa de anotación.

Como escribí anteriormente un programa en Python que generaba un XML cumpliendo con los campos necesarios para el programa de anotaciones, aproveché ese código y manejo de la librería para generar el XML. Para ello era necesario que desde Matlab se ejecutase dicha función, realizando una comunicación entre las variables de Matlab y la función escrita en lenguaje Python.

Desde el lado Matlab, a partir de los valores obtenidos, se calculan las coordenadas x e y de cada vértice de la elipse. Estos 8 valores se almacenan en una matriz en la que cada fila es una elipse y las columnas se corresponden con los puntos. Matlab permite el paso de ciertos tipos de variables a funciones escritas en Python, pero no el paso de matrices. Para ello guardo la matriz en un fichero .mat que posteriormente la rutina en Python abre y carga los valores. En la matriz, además de los puntos, hay una columna adicional que indica el tipo de célula, para que al generar el XML se escriba la correspondiente etiqueta. Si la célula es azul se utiliza el valor '0' para indicar a la rutina en Python que debe escribir la etiqueta 'Cell'. Si la célula es marrón, se emplea el valor '1' y se escribe la etiqueta 'CellN'. También es necesario indicar el nombre del fichero de la imagen, ya que tiene que coincidir con el nombre del XML, por ello se pasan como parámetros a la función. Por último, desde Matlab se hace la llamada a la función xmlWritter.py [15]. Esta genera un fichero XML con la anotación realizada en Matlab.

Finalmente, una vez se ha corregido la anotación, se llama a una rutina en Matlab que calcula el F-score. Primero se toman los datos del XML con *xmlReader.py*, este programa carga los ficheros XML y genera un archivo .mat con la matriz de puntos de las elipses. Desde Matlab se toman estos puntos, por cada vector correspondiente a una elipse, se generan los puntos de la elipse. Creando una máscara negra, es decir de valor 1, se pintan estos trazos de elipses. Para poder comparar ambas anotaciones, decidí realizar una máscara binaria de cada anotación, donde el espacio ocupado por la anotación es blanco y el fondo oscuro. Para rellenar las elipses uso la técnica de hole-filling, resolviendo también el problema de que una figura intersecte con un borde y no esté cerrada [16]. Una vez generadas las máscaras correspondientes a la anotación Matlab y la anotación corregida, se calculan los parámetros de precisión, exhaustividad y f-score. Además, se proporciona una máscara que detalla las diferencias entre ambas imágenes. Si una elipse está presente en la anotación original y en la corregida no está, se marca de color amarillo. Si, por el contrario, la elipse se añade en la corrección, esta aparece representada en color azul.

3.6. Ejecutable para Windows

Debido a que es necesario instalar un intérprete de Python y diversas librerías para ejecutar el programa, decidimos que era mejor poder generar un ejecutable Windows (.exe) y facilitar la tarea al anotador. Por ello usé el programa PyInstaller. [17] Este permite generar un archivo ejecutable del proyecto .py indicado, añadiendo previamente unos indicadores de lectura en el archivo principal. De esta manera solo es necesario que el usuario ejecute el fichero .exe, sin necesidad de instalaciones previas, permitiendo un uso muy cómodo.

4. Resultados

En este apartado mostraré los resultados obtenidos del programa EllipseLabelImg. Para probar su funcionamiento he realizado conjuntos de pruebas. La primera prueba con un recorte de una imagen de células y la segunda con la imagen completa. En el primer caso, es muy útil para ver si el programa produce algún error al importar las anotaciones Matlab, comprobar que las posiciones de las anotaciones y sus etiquetas son correctas. En el segundo caso es una simulación de la experiencia real con el programa, anotar imágenes enteras.

4.1. Prueba con recorte

A continuación, mostraré la interfaz gráfica de EllipseLabelImg y su funcionamiento. La interfaz consta de una barra lateral con varios botones de apertura de imagen, guardado, pasar a la siguiente imagen y realizar una nueva anotación. En el centro encontramos el canvas donde se mostrará la imagen que vamos a anotar y en el lateral un listado de las etiquetas en la imagen.

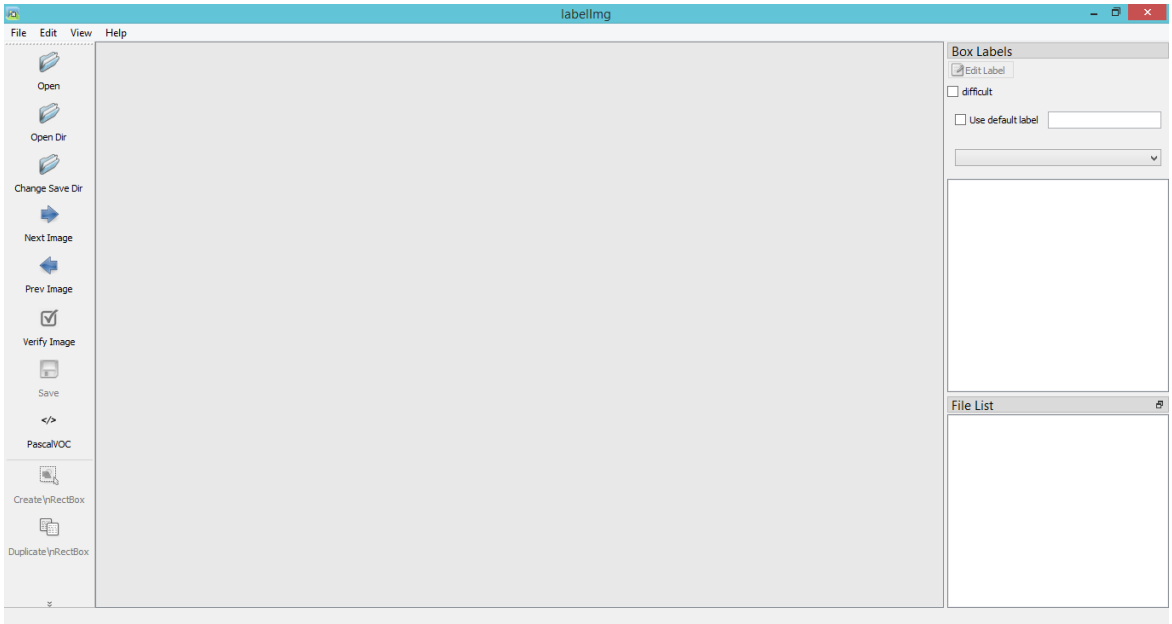


Figure 27. *Pantalla de inicio de EllipseLabelImg*

Para empezar a anotar hacemos en clic en Open o en Open Dir. Open abrirá únicamente la imagen seleccionada, mientras que Open Dir abrirá la primera imagen del directorio y permitirá desplazarse a las siguientes imágenes con el botón Next Image. Al existir un XML con una anotación, el programa lo lee e importa automáticamente la anotación realizada en Matlab. En la barra lateral observamos la lista de etiquetas, las células con tinte marrón tienen una etiqueta de color rojo, para poder resaltar del resto y dar prioridad a su corrección ya que son las más relevantes.

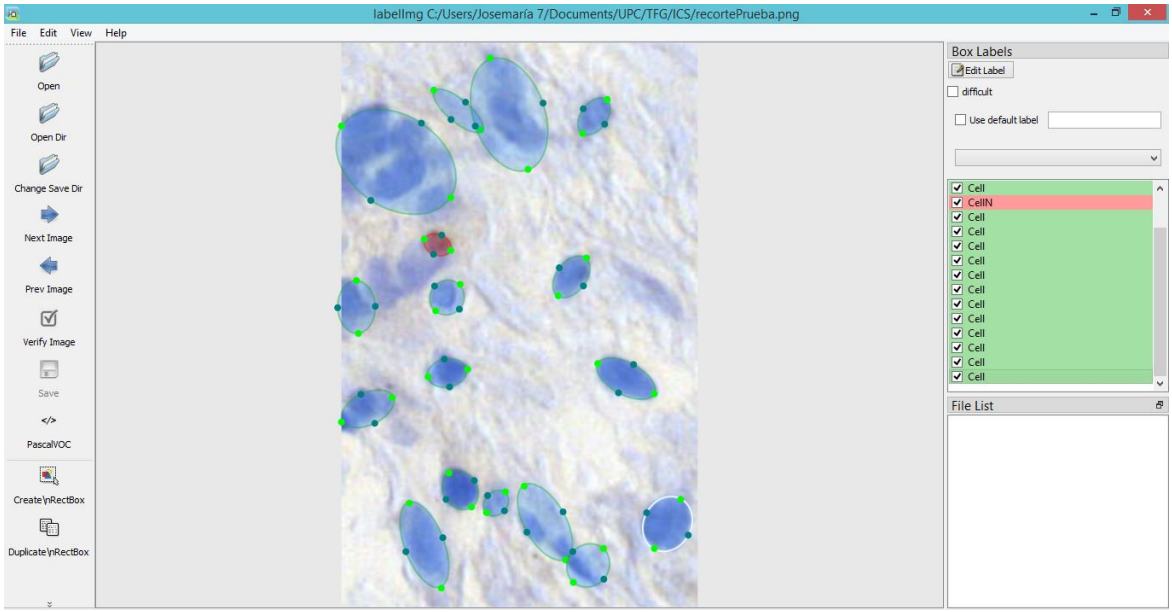


Figure 28. *Interfaz EllipseLabelImg al abrir una imagen*

La anotación generada en Matlab se ha importado de forma correcta. Las células con tinte marrón están anotadas con un sombreado rojo para hacerlas más visibles, además de seguir los mismos colores empleados en Matlab.

El programa permite mover las elipses, modificarlas, eliminar y añadir de nuevas. Al añadir una nueva elipse debemos indicar su etiqueta y se actualiza el listado de la barra lateral. Si hacemos clic a una etiqueta de la barra, su elipse correspondiente queda seleccionada.

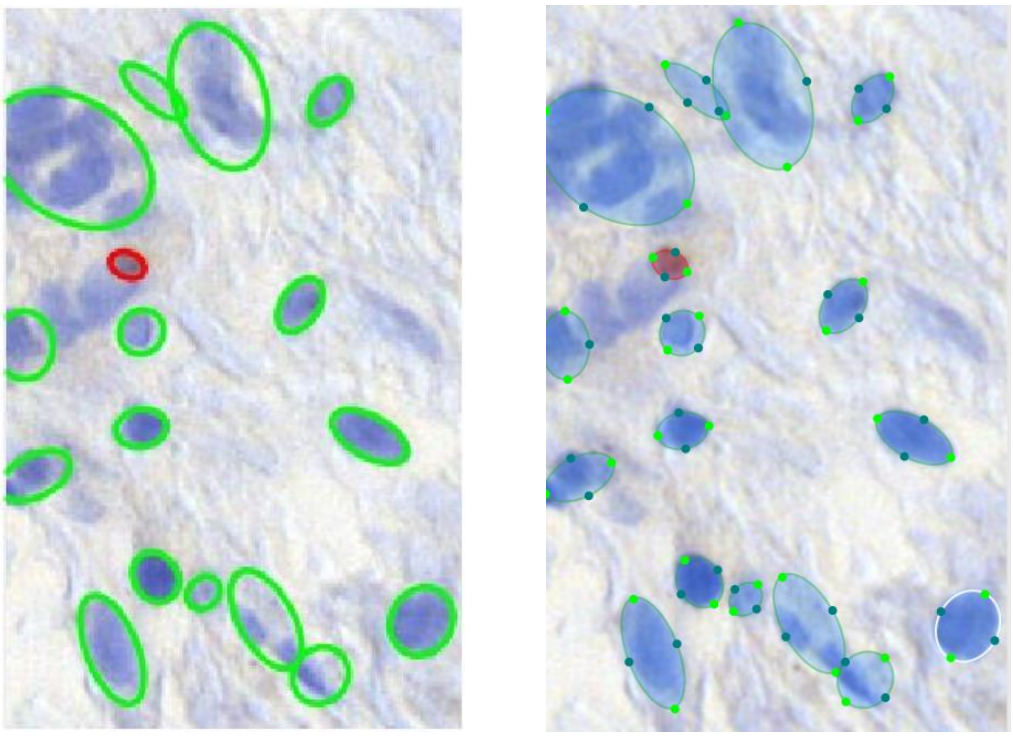


Figure 29. *Comparativa entre la anotación realizada en Matlab y la anotación importada en EllipseLabelImg.*

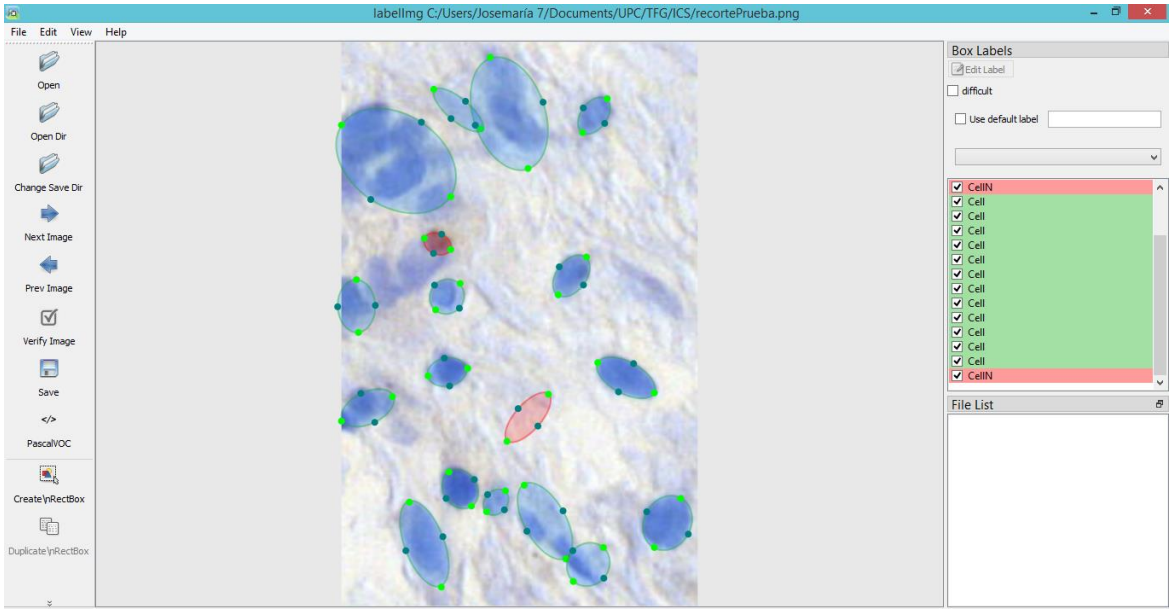


Figure 30. *Muestra al añadir una elipse con etiqueta 'CellN', esta se muestra en rojo y se añade a la lista.*

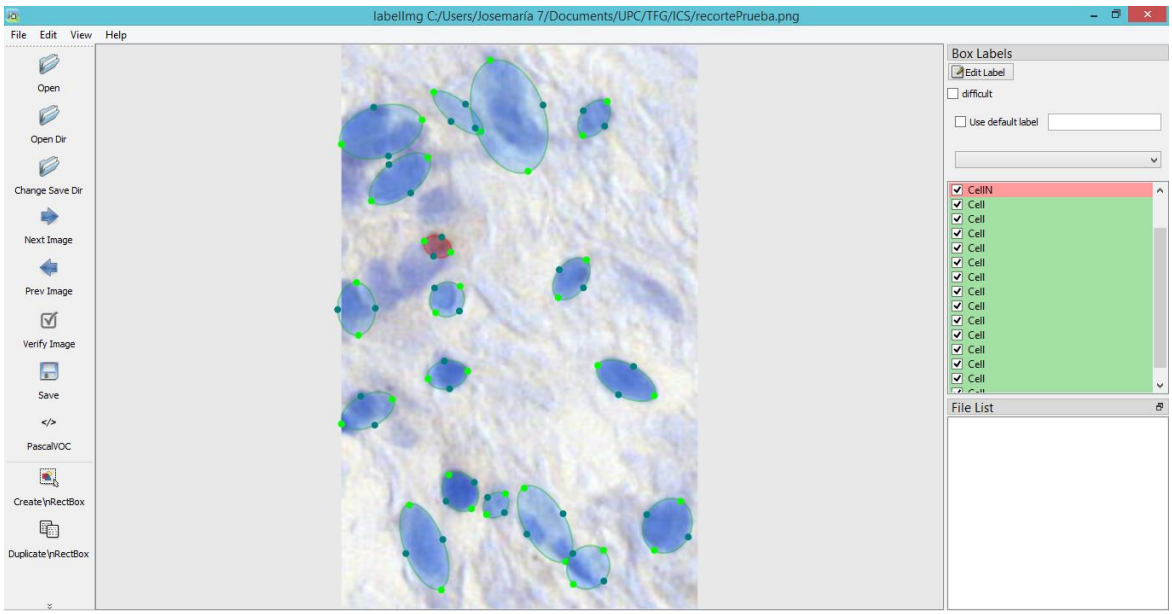


Figure 31. *Muestra al eliminar una elipse y añadir dos elipses como ejemplo de corrección.*

Para comprobar la exactitud en la que el sistema importa y exporta las anotaciones, sin realizar ninguna modificación, he usado el f-score como medida de exactitud, además de visualizar la máscara de diferencia entre ambas anotaciones. En este caso el valor de f-score ha sido de 1. Como se observa en la siguiente figura, no se aprecian diferencias entre la anotación importada y la exportada por el programa. Además, he comprobado el correcto funcionamiento del programa f-score, eliminando una anotación en el programa anotador y observando la máscara diferencia. Se aprecia una elipse amarilla, correspondiente a la anotación eliminada.

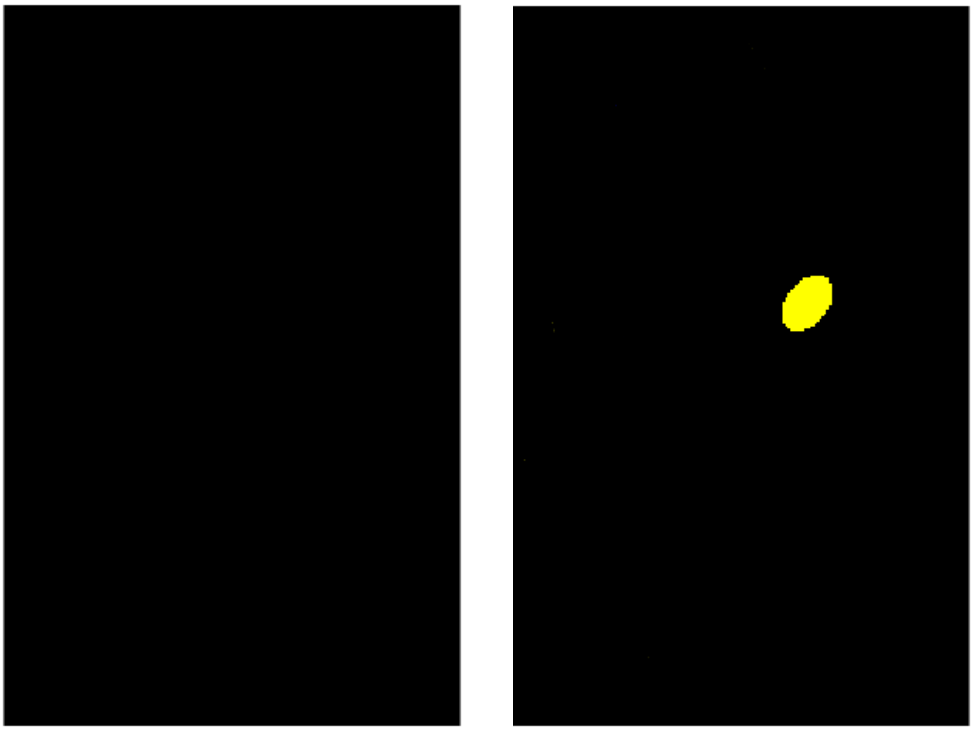


Figure 32. a) *Máscara de diferencias entre anotaciones.* ***b)*** *Máscara de diferencias con elipse eliminada.*

Teniendo en cuenta los resultados en esta prueba, podemos afirmar que el programa es totalmente funcional.

4.2. Prueba con imagen completa

Aquí se ha tratado de ver si el programa funcionaba correctamente al trabajar con una imagen completa, tal como sería en una experiencia real. Se debe notar que, en cuanto a carga del programa, es diferente tener 20 células o tener 1000 células que son las que suelen tener. En la siguiente imagen de la interfaz, podemos comprobar que a vista completa no se puede trabajar las anotaciones y que debemos hacer zoom en diferentes zonas. El zoom se realiza con los comandos Ctrl + +.

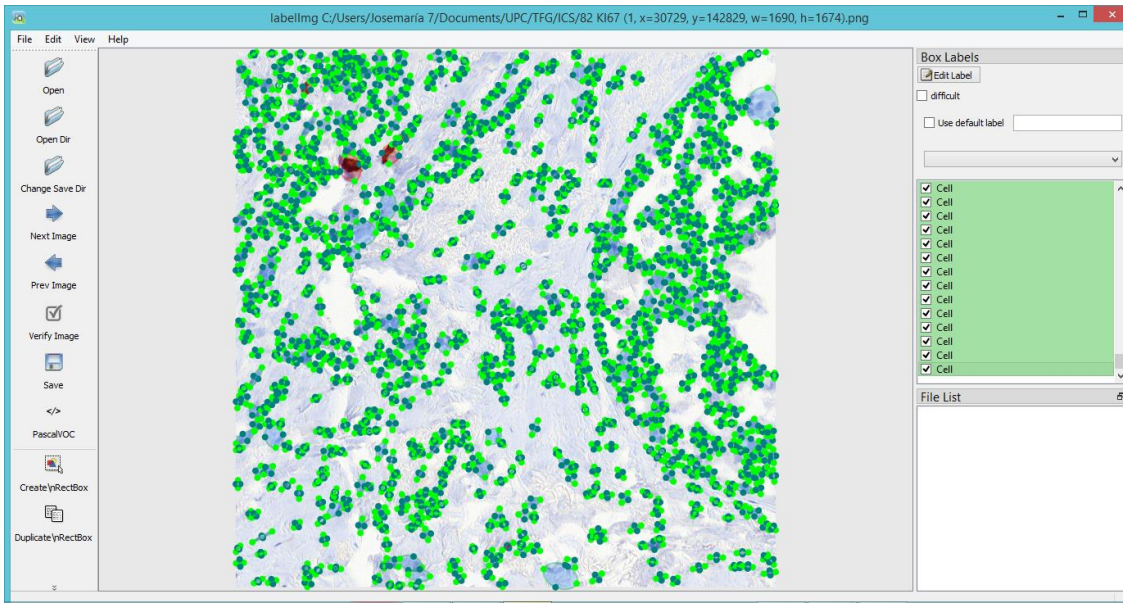


Figure 33. *Captura del programa EllipseLabelImage al cargar una imagen completa*

El programa ha tardado más en mostrar la imagen con las elipses, pero el tiempo está por debajo del minuto. Al realizar zoom, este no es inmediato y tarda alrededor de 5 segundos. Para desplazarnos sobre la imagen, hay que hacer uso de las barras que aparecen alrededor de la imagen. Al hacer uso de ellas, el movimiento no es inmediato, sino que tarda unos segundos.

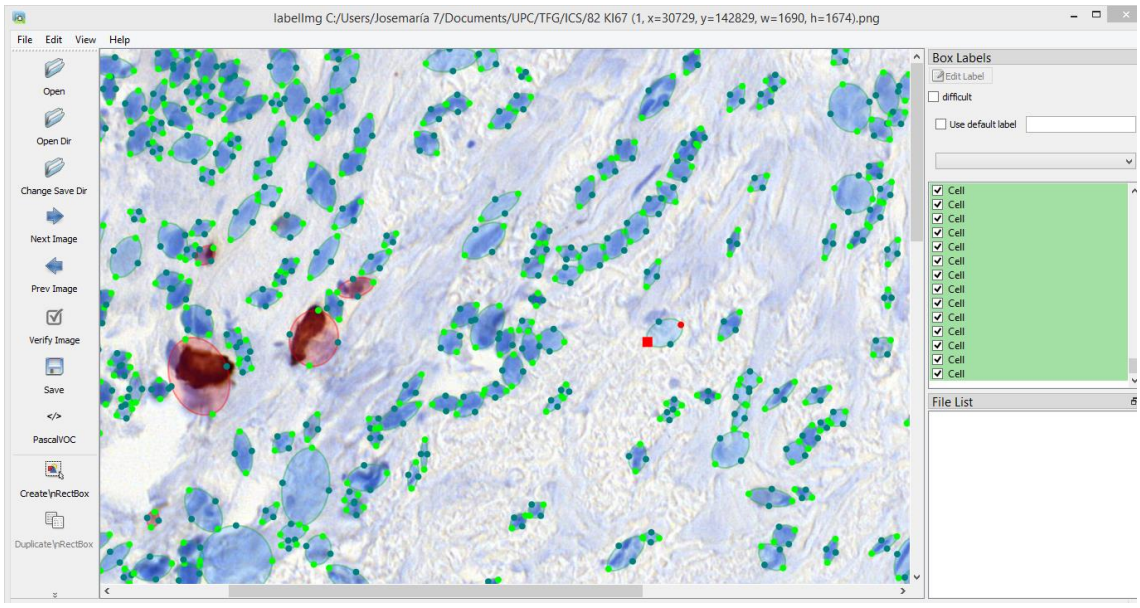


Figure 34. *Captura de la imagen magnificada. Se realiza un movimiento de uno de los vertices.*

Si realizamos las mismas acciones que en la prueba anterior, vemos que el programa tarda unos segundos en realizar las tareas de desplazamiento, selección de elipse, dificultando su uso. Esto lleva a una mala experiencia por parte del usuario, ya que es necesaria la agilidad a la hora de anotar. Traté solucionar el problema realizando una carga selectiva, solo mostrando las elipses de la zona que se visualiza, pero finalmente no pude implementar una solución que funcionase.

Finalmente, he comprobado el valor f-score y la máscara de diferencias.

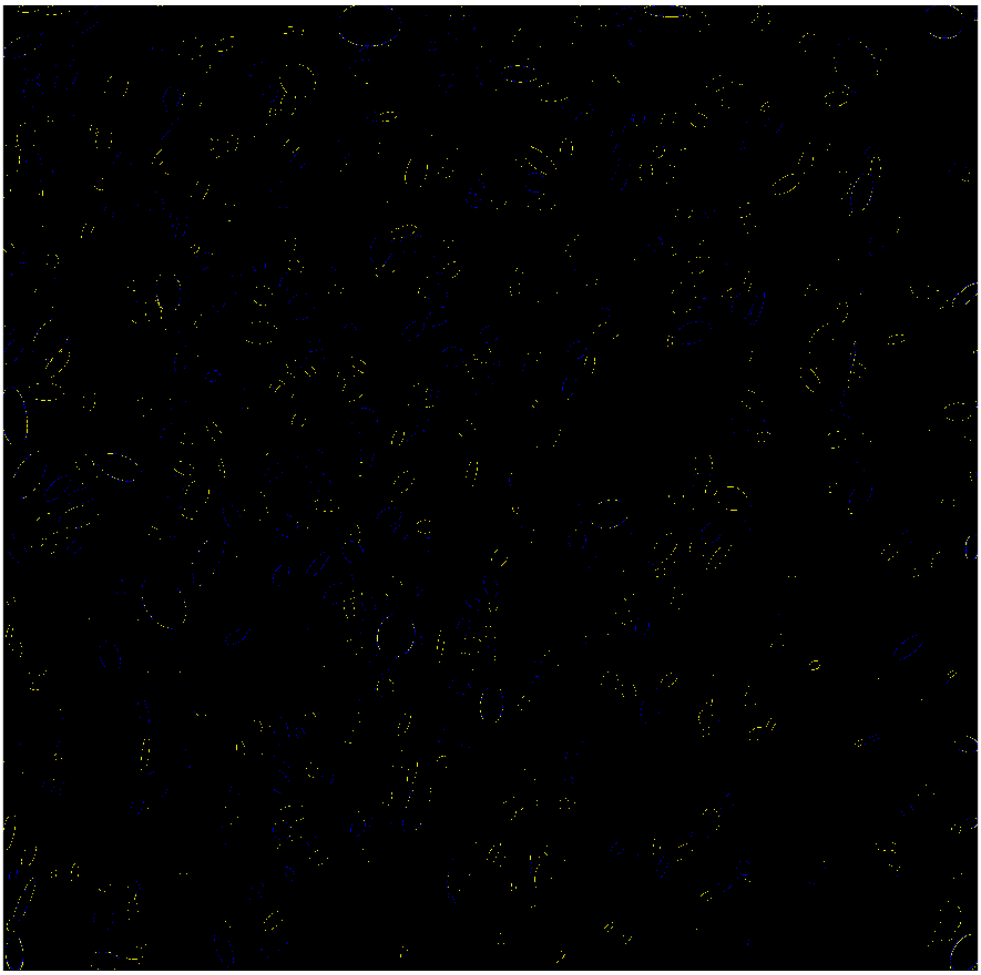


Figure 35. *Máscara de diferencias entre la anotación original y la obtenida por el programa.*

En este caso se pueden apreciar pequeñas diferencias entre la anotación generada y la obtenida tras pasar por el programa de anotación. El valor de f-score obtenido es de 0,9862. Para tener una medida de fscore más independiente de la imagen, he realizado la prueba a todas las imágenes disponibles de la proteína KI67.

	Ima 1	Ima 2	Ima 3	Ima 4	Ima 5	Ima 6	Media
F-score	0.9840	0.9862	0.9827	0.9866	0.9861	0.9821	0.9846

Table 1. *Valores de F-score obtenidos en las imágenes KI67.*

El programa introduce un pequeño error en las imágenes completes y se ralentiza debido a la carga de todas las anotaciones en las variables del sistema.

5. Presupuesto

Este proyecto ha consistido en el desarrollo de software. La gran mayoría de programas empleados son de libre distribución, solo ha sido necesaria la licencia de Matlab. Para el desarrollo solo ha sido necesario un portátil. Por lo tanto, los costes contemplarán únicamente la licencia de Matlab, un portátil y el personal.

Coste software:

La licencia de un año de Matlab tiene un coste de 800 euros. Esta se ha usado durante 5 meses.

Licencias Software	Uso (meses)	Amortización (años)	Coste (€)
Matlab	5	1	333,34

Table 2. *Costes por software*

Coste hardware:

Durante el proyecto se ha hecho uso de un portátil de gama media – alta para poder trabajar de forma cómoda y rápida. El coste aproximado de un portátil de estas características es de 1000 euros.

Hardware	Uso (meses)	Amortización (años)	Coste (€)
Portátil	5	5	83,34

Table 3. *Costes por Hardware*

Coste empleado:

Se considera un empleo de ingeniero Junior y un salario de 10 euros la hora. Además, se tiene en cuenta los costes derivados de la seguridad social que la empresa debe hacerse cargo.

Empleo	Salario/hora (€/h)	Total horas (h)	Seguridad Social (%)	Coste (€)
Ingeniero Junior	10	540	33,4	7203,6

Table 4. *Costes por empleado*

Coste total proyecto:

Descripción	Coste (€)
Software	333,34
Hardware	83,34
Ingeniero Junior	7203,6
Total	7620,28

Table 5. *Coste total.*

6. Conclusiones y future desarrollos:

La idea propuesta al principio era poder participar en un gran proyecto con un impacto social relevante. Tengo particular interés en el ámbito sanitario y el procesado de imagen. Me parecía una gran oportunidad para realizar un trabajo relevante y del que poder ampliar conocimientos vistos en la mención de audiovisuales. Concretamente el procesado por morfología me resulta muy interesante y son técnicas que iba a emplear. Esperaba poder realizar tareas en los ámbitos mencionados.

Al principio del proyecto, no esperaba que tuviera que desarrollar una interfaz gráfica para poder corregir las anotaciones. Sobre el papel sí que era sencillo implementar una interfaz con la GUI de Matlab, aun sin no haber usado demasiado la herramienta. Creo que por parte de mis tutores esperaban que esta tarea fuese mucho más fácil de asumir y realizar, pudiendo así continuar con el resto de las tareas, probablemente relacionadas con lo que quería hacer, procesado de imagen. Debido a las incidencias con Matlab, hubo un cambio de planes, en el que debía modificar un programa de anotación ya existente. Una vez más asumieron que no me llevaría mucho tiempo y podría hacer diversas tareas. Al final ha resultado mucho más complicado de lo que esperábamos, por ambas partes. Nunca había implementado interfaces gráficas y desconocía totalmente la librería PyQt5 empleada en el programa LabelImg que es el que modifiqué. A ello se sumó también que el código no estaba documentado, había una ausencia total de comentarios, dificultando mucho la entrada a la comprensión del código y ver que partes eran necesarias modificar. Debido a diversos factores, complicaciones y problemas personales, el proyecto ha terminado siendo el desarrollo de una herramienta de anotación. Tengo que admitir que a priori, no era la persona más adecuada para realizar esta tarea, dado que mis conocimientos en esos ámbitos eran muy escasos, a diferencia de haber trabajado en el ámbito de procesado de imagen. He aprendido mucho sobre el desarrollo de interfaces gráficas en Python, sin embargo, no es una temática que sea de mi interés. Estoy satisfecho con haber logrado realizar una herramienta de anotación, pero no es lo que esperaba realizar en el proyecto. Me habría gustado más centrarme en otras tareas. Considero que el realizar esta herramienta es relevante, ya que un médico debe anotar esas imágenes y así continuar el proyecto. He aprendido que no se debe trabajar un código sin que esté documentado, dificulta mucho las tareas. También he concluido que antes de aceptar un proyecto, debo conocer lo que voy a realizar de antemano y realizar una buena planificación, para que no haya este tipo de decepciones.

Como he mencionado en el apartado de resultados, el programa funciona correctamente para imágenes pequeñas dónde hay pocas células. Introduce un pequeño error debido a que la generación de las elipses da valores continuos y se discretizan para pintar la imagen, produciendo pequeñas variaciones de 1 o 2 píxeles.

Sobre la herramienta de anotación, el siguiente paso sería hacer que funcionase con las imágenes de células. El programa no funciona si tiene que cargar en sus variables muchas elipses, es por ello por lo que debe cargar solo las que se muestran en pantalla al realizar un zoom. Otra funcionalidad necesaria es la de marcar las zonas de la imagen en las que ya se ha corregido la anotación, facilitando así la tarea del anotador. Una vez implementadas estas funcionalidades, el programa será totalmente funcional. Después de corregir las anotaciones, se puede optimizar el código en Matlab para aumentar su precisión. Dada una precisión suficientemente alta, se podría anotar muchas más imágenes sin necesidad de un anotador y así obtener una gran base de datos. Finalmente, con una gran base de datos, se podría desarrollar una red neuronal capaz de realizar un diagnóstico a través de la imagen de la biopsia del paciente.

Bibliografía:

[1] Pronóstico del Cáncer de Mama: Mortalidad y Esperanza de vida. (s. f.). AECC. Recuperado 8 de agosto de 2020, de https://www.aecc.es/es/todo-sobre-cancer/tipos-cancer/cancer-mama/mas-informacion/evolucion-cancer-mama

[2] American Society of Clinical Oncology. (s. f.). Cáncer de mama - Diagnóstico. Cancer.Net. Recuperado 8 de agosto de 2020, de https://www.cancer.net/es/tipos-de-cancer/cancer-de-mama/diagnostico

[3] Image and Video Procesing Group, TSC, UPC. (2019). 3.2 Mathematical Morphology and Lattice [Diapositivas].

[4] Image and Video Procesing Group, TSC, UPC. (2019). 3.5 Filters by reconstruction [Diapositivas].

[5] Dr. Rashi Argawal. (2015, 27 febrero). Segmentation using Watershed Algorithm in Matlab [Vídeo]. YouTube. https://www.youtube.com/watch?v=K5P5rjDiZzk

[6] M. Aubreville, C. Bertram, R. Klopfleisch and A. Maier (2018) SlideRunner - A Tool for Massive Cell Annotations in Whole Slide Images. In: Bildverarbeitung für die Medizin 2018. Springer Vieweg, Berlin, Heidelberg, 2018. pp. 309-314. link arXiv:1802.02347

[7] Tzutalin. LabelImg. Git code (2015). https://github.com/tzutalin/labelImg

[8] Rasband, W.S., ImageJ, U. S. National Institutes of Health, Bethesda, Maryland, USA, https://imagej.nih.gov/ij/, 1997-2018.

[9] Harwani, B. M. (2012). Introduction to Python programming and developing GUI applications with PyQT. Boston, MA: Course Technology.

[10] tzutalin. (2015, 1 enero). Screenshot LabelImg [Diapositivas]. GitHub. https://raw.githubusercontent.com/tzutalin/labelImg/master/demo/demo3.jpg

[11] Kamalpreet Grewal. (2014, 4 octubre). Set the orientation of an ellipse in Qt. Stack Overflow. https://stackoverflow.com/questions/26194011/set-the-orientation-of-an-ellipse-in-qt.

[12] Hendricks, M. C. (2012, marzo). Rotated Ellipses and their intersections with lines. http://quickcalcbasic.com/ellipse%20line%20intersection.pdf

[13] cgvict. (2017, 12 mayo). roLabelImg. GitHub. https://github.com/cgvict/roLabelImg

[14] Python. (s. f.). The ElementTree XML. GitHub. Recuperado 4 de julio de 2020, de https://github.com/python/cpython/blob/3.8/Doc/library/xml.etree.elementtree.rst

[15] Matlab. (2019, 9 agosto). Using MATLAB with Python [Vídeo]. YouTube. https://www.youtube.com/watch?v=y7NBT6O0fJU&t

[16] Eddins, S. (2013, 5 septiembre). Defining and filling holes on the border of an image. Mathworks. https://blogs.mathworks.com/steve/2013/09/05/defining-and-filling-holes-on-the-border-of-an-image/

[17] Goebel, H. and Jones, B., 2020. Pyinstaller.

Apendices:

En este apartado incluiré las funciones más relevantes de cada fichero modificado para el funcionamiento de EllipseLabelImg. Además incluye el Código que genera el XML desde Matlab y los códigos de test.

Canvas.py:

```
def finalise(self):
        print("he entrado en finalise")
        assert self.current
        if self.current.points[0] == self.current.points[-1]:
            self.current = None
            self.drawingPolygon.emit(False)
            self.update()
            return

        self.current.close()
        self.shapes.append(self.current)
        self.numLines+=1
        self.current = None
        self.setHiding(False)
        if self.numLines == 2:
            self.numLines = 0
            self.joinShapes()
            self.newShape.emit()
            #Coger shape actual y anterior y juntarlos, juntar los points

        #Abre cajita de dialogo
        #self.newShape.emit()
        self.update()
 def joinShapes(self):
        #Coger los ultimos 2 elems del array
        self.shapes[-2].points += self.shapes[-1].points
        self.shapes = self.shapes[:-1]
        self.shapes[-1].isEllipse = True
        print("valores que me han quedado en shape")
        print(self.shapes[-1].points)


def keyPressEvent(self, ev):
        #Poner la tecla y hacer operaciones en moveonepixel desde el centro
        key = ev.key()
        if key == Qt.Key_Escape and self.current:
            print('ESC press')
            self.current = None
            self.drawingPolygon.emit(False)
            self.update()
        elif key == Qt.Key_Return and self.canCloseShape():
```

```
            self.finalise()
        elif key == Qt.Key_Left and self.selectedShape:
            self.moveOnePixel('Left')
        elif key == Qt.Key_Right and self.selectedShape:
            self.moveOnePixel('Right')
        elif key == Qt.Key_Up and self.selectedShape:
            self.moveOnePixel('Up')
        elif key == Qt.Key_Down and self.selectedShape:
            self.moveOnePixel('Down')
        elif key == Qt.Key_Z and self.selectedShape:
            self.moveOnePixel('RotateCCW')
        elif key == Qt.Key_X and self.selectedShape:
            self.moveOnePixel('RotateCCW_less')
        elif key == Qt.Key_C and self.selectedShape:
            self.moveOnePixel('RotateCW_less')
        elif key == Qt.Key_V and self.selectedShape:
            self.moveOnePixel('RotateCW')

    def moveOnePixel(self, direction):
        # print(self.selectedShape.points)
        center = self.selectedShape.centerEllipse
        heightEllipse = self.selectedShape.heightEllipse
        widthEllipse = self.selectedShape.widthEllipse
        self.selectedShape.isXML = False
        if direction == 'Left' and not self.moveOutOfBound(QPointF(-
1.0, 0)):
            # print("move Left one pixel")
            self.selectedShape.points[0] += QPointF(-1.0, 0)
            self.selectedShape.points[1] += QPointF(-1.0, 0)
            self.selectedShape.points[2] += QPointF(-1.0, 0)
            self.selectedShape.points[3] += QPointF(-1.0, 0)
        elif direction == 'Right' and not self.moveOutOfBound(QPointF(1.0,
0)):
            # print("move Right one pixel")
            self.selectedShape.points[0] += QPointF(1.0, 0)
            self.selectedShape.points[1] += QPointF(1.0, 0)
            self.selectedShape.points[2] += QPointF(1.0, 0)
            self.selectedShape.points[3] += QPointF(1.0, 0)
        elif direction == 'Up' and not self.moveOutOfBound(QPointF(0, -
1.0)):
            # print("move Up one pixel")
            self.selectedShape.points[0] += QPointF(0, -1.0)
            self.selectedShape.points[1] += QPointF(0, -1.0)
            self.selectedShape.points[2] += QPointF(0, -1.0)
            self.selectedShape.points[3] += QPointF(0, -1.0)
        elif direction == 'Down' and not self.moveOutOfBound(QPointF(0, 1.0
)):
            # print("move Down one pixel")
```

```
            self.selectedShape.points[0] += QPointF(0, 1.0)
            self.selectedShape.points[1] += QPointF(0, 1.0)
            self.selectedShape.points[2] += QPointF(0, 1.0)
            self.selectedShape.points[3] += QPointF(0, 1.0)
        elif direction == 'RotateCCW':
            rotatePoints = self.selectedShape.points
            angle = -0.1
            i = 0
            for rotP in rotatePoints:
                if i%2 == 0:
                    xrot = ((rotP.x() - center.x()) * math.cos(angle) - (ro
tP.y() - center.y()) * math.sin(angle) + center.x())
                    yrot = ((rotP.y() - center.y()) * math.cos(angle) + (ro
tP.x() - center.x()) * math.sin(angle) + center.y())
                    pt = QPointF(xrot,yrot)
                    self.selectedShape.points[i] = pt
                i += 1
            i = 0
        elif direction == 'RotateCCW_less':
            rotatePoints = self.selectedShape.points
            angle = -0.01
            i = 0
            for rotP in rotatePoints:
                if i%2 == 0:
                    xrot = ((rotP.x() - center.x()) * math.cos(angle) - (ro
tP.y() - center.y()) * math.sin(angle) + center.x())
                    yrot = ((rotP.y() - center.y()) * math.cos(angle) + (ro
tP.x() - center.x()) * math.sin(angle) + center.y())
                    pt = QPointF(xrot,yrot)
                    self.selectedShape.points[i] = pt
                i += 1
            i = 0
        elif direction == 'RotateCW_less':
            rotatePoints = self.selectedShape.points
            angle = 0.01
            i = 0
            for rotP in rotatePoints:
                if i%2 == 0:
                    xrot = ((rotP.x() - center.x()) * math.cos(angle) - (ro
tP.y() - center.y()) * math.sin(angle) + center.x())
                    yrot = ((rotP.y() - center.y()) * math.cos(angle) + (ro
tP.x() - center.x()) * math.sin(angle) + center.y())
                    pt = QPointF(xrot,yrot)
                    self.selectedShape.points[i] = pt
                i += 1
            i = 0
        elif direction == 'RotateCW':
            rotatePoints = self.selectedShape.points
```

```
            angle = 0.1
            i = 0
            for rotP in rotatePoints:
                if i%2 == 0:
                    xrot = ((rotP.x() - center.x()) * math.cos(angle) - (ro
tP.y() - center.y()) * math.sin(angle) + center.x())
                    yrot = ((rotP.y() - center.y()) * math.cos(angle) + (ro
tP.x() - center.x()) * math.sin(angle) + center.y())
                    pt = QPointF(xrot,yrot)
                    self.selectedShape.points[i] = pt
                i += 1
            i = 0
        self.shapeMoved.emit()
        self.repaint()
```

shape.py:

```
class Shape(object):
    P_SQUARE, P_ROUND = range(2)

    MOVE_VERTEX, NEAR_VERTEX = range(2)

    # The following class variables influence the drawing
    # of _all_ shape objects.
    line_color = DEFAULT_LINE_COLOR
    fill_color = DEFAULT_FILL_COLOR
    select_line_color = DEFAULT_SELECT_LINE_COLOR
    select_fill_color = DEFAULT_SELECT_FILL_COLOR
    vertex_fill_color = DEFAULT_VERTEX_FILL_COLOR
    hvertex_fill_color = DEFAULT_HVERTEX_FILL_COLOR
    vertex_fill_color_minor = DEFAULT_VERTEX_FILL_COLOR_MINOR
    diff_cell_fill_color = DIFFERENT_FILL_COLOR
    point_type = P_ROUND
    point_size = 8
    scale = 1.0

    def __init__(self, label=None, line_color=None, difficult=False, paintL
abel=False):
        self.label = label
        self.points = []
        self.fill = False
        self.selected = False
        self.difficult = difficult
        self.paintLabel = paintLabel

        self._highlightIndex = None
        self._highlightMode = self.NEAR_VERTEX
```

```
        self._highlightSettings = {
            self.NEAR_VERTEX: (4, self.P_ROUND),
            self.MOVE_VERTEX: (1.5, self.P_SQUARE),
        }

        self._closed = False
        self.isEllipse = False
        self.centerEllipse = QPointF(0,0)
        self.heightEllipse = 0
        self.widthEllipse = 0
        self.isXML = False
        if line_color is not None:
            # Override the class line_color attribute
            # with an object attribute. Currently this
            # is used for drawing the pending line a different color.
            self.line_color = line_color

def reachMaxPoints(self):
        if len(self.points) >= 8:
            return True
        return False


def paint(self, painter):
        if self.isEllipse:
            return self.paintEllipse(painter)
        else:
            return self.paintRect(painter)
def makePath(self):
        print("he entrado en makePath")
        print(len(self.points))
        pointsDiamond = [self.points[0], self.points[4], self.points[2], se
lf.points[6]]
        poly = QPolygonF(pointsDiamond)
        path = QPainterPath(self.points[0])
        path.addPolygon(poly)
        return path


def paintEllipse(self, painter):
        print("estamos pintando una ellipse")
        if self.points:
            color = self.select_line_color if self.selected else self.line_
color
            pen = QPen(color)
            # Try using integer sizes for smoother drawing(?)
            pen.setWidth(max(1, int(round(2.0 / self.scale))))
            painter.setPen(pen)
```

```
            line_path = QPainterPath()
            vrtx_path = QPainterPath()
            vrtx_path_minor = QPainterPath()
            ellipse_path = QPainterPath()

            line_path.moveTo(self.points[0])
            ellipse_path.moveTo(self.points[0])
            # Uncommenting the following line will draw 2 paths
            # for the 1st vertex, and make it non-filled, which
            # may be desirable.
            #self.drawVertex(vrtx_path, 0)

            for i, p in enumerate(self.points):
                print("dentro del bucle de puntos")
                print(p)
                print(i)
                print('shape paint points (%d, %d)' % (p.x(), p.y()))
                #Aquí se pinta la linea entre 2 puntos
                line_path.lineTo(p)
                #if i == 4 or i == 6 or i == 2 or i == 0:
                if i == 2 or i == 0:
                    self.drawVertex(vrtx_path, i)
            if self.isClosed():
                #line_path.lineTo(self.points[0])
                print("He hecho lineTo de self.points[0] a line_path")
            # Prueba de hacer path eliptico
            #line_path.addEllipse(p, 50, 200, 400)
            #vrtx_path.addEllipse(self.drawVertex)
            print("numero de points", len(self.points))
            if len(self.points) > 1:
            #ellipse path
                print("pathEllipse")
                print(len(self.points))
                print(self.points[0])
                print(self.points[1])
                print(self.points[2])
                print(self.points[3])
                print(self.points[4])
                print(self.points[5])
                print(self.points[6])
                print(self.points[7])

                fstMajorAxis = self.points[0]
                endMajorAxis = self.points[2]
                xCenter = (endMajorAxis.x() - fstMajorAxis.x())/2 + fstMajo
rAxis.x() if endMajorAxis.x() > fstMajorAxis.x() else (fstMajorAxis.x() - e
ndMajorAxis.x())/2 + endMajorAxis.x()
```

```
                yCenter = (endMajorAxis.y() - fstMajorAxis.y())/2 + fstMajo
rAxis.y() if endMajorAxis.y() > fstMajorAxis.x() else (fstMajorAxis.y() - e
ndMajorAxis.y())/2 + endMajorAxis.y()
                widthRect = round(math.sqrt((self.points[2].x()-
self.points[0].x())**2 + (self.points[2].y()-self.points[0].y())**2))
                heightRect = round(math.sqrt((self.points[6].x()-
self.points[4].x())**2 + (self.points[6].y()-self.points[4].y())**2))

                self.heightEllipse = heightRect
                self.widthEllipse = widthRect

                xCenter = round(xCenter)
                yCenter = round(yCenter)
                print('el centro calculado es (%d,%d)', (xCenter,yCenter))
                ellipse_center = QPointF(xCenter, yCenter)
                self.centerEllipse = ellipse_center
                ellipse_path.moveTo(ellipse_center)
                print(ellipse_center.x())
                print(ellipse_center.y())
                rectEllipse = QRectF(ellipse_center.x() - widthRect/2, elli
pse_center.y() - heightRect/2,widthRect,heightRect)
                print(widthRect)
                print(heightRect)
                ellipse_path.addEllipse(rectEllipse)
                #Aqui ya está la ellipse
                #Calculo del ángulo y rotación
                #Dont divide by 0 on slope calc
                slopeMajor = (self.points[2].y() - self.points[0].y())/(sel
f.points[2].x() - self.points[0].x()) if endMajorAxis.x() != fstMajorAxis.x
() else math.inf
                theta_rad = math.atan2(self.points[2].y() - self.points[0].
y(), self.points[2].x() - self.points[0].x())
                theta = theta_rad * (180/math.pi)
                #Pintar eje menor
                #fstMinorAxes = QPointF(xCenter,yCenter - heightRect/2)
                #print(fstMinorAxes)
                print("angulo", theta_rad)
                #endMinorAxes = QPointF(xCenter,yCenter + heightRect/2)


                #fstMinorPoint_x = xCenter - heightRect*math.cos(theta_rad)
                fstMinorPoint_x = (xCenter - xCenter)*math.cos(theta_rad) -
 (yCenter + heightRect/2 - yCenter)*math.sin(theta_rad) + xCenter
                fstMinorPoint_y = (yCenter + heightRect/2 - yCenter)*math.c
os(theta_rad) + (xCenter + - xCenter)*math.sin(theta_rad) + yCenter
                endMinorPoint_x = (xCenter - xCenter)*math.cos(theta_rad) -
 (yCenter - heightRect/2 - yCenter)*math.sin(theta_rad) + xCenter
```

```
                endMinorPoint_y = (yCenter - heightRect/2 - yCenter)*math.c
os(theta_rad) + (xCenter + - xCenter)*math.sin(theta_rad) + yCenter
                #fstMinorPoint_y = yCenter - heightRect*math.sin(theta_rad)
                print(fstMinorPoint_x)
                print(y_o)


                painter.save()
                #painter.translate(ellipse_center.y() - widthRect/2, ellips
e_center.y() + heightRect/2)
                painter.translate(ellipse_center.x(), ellipse_center.y())
                #painter.translate(widthRect/2, heightRect/2)
                painter.rotate(theta)
                #painter.translate(-ellipse_center.x()+ widthRect/2, -
ellipse_center.y() + heightRect/2)
                painter.translate(-ellipse_center.x(), -ellipse_center.y())
                painter.drawPath(ellipse_path)
                #painter.translate(-widthRect/2, -heightRect/2)
                #color = self.select_fill_color if self.selected else self.
fill_color
                color = self.diff_cell_fill_color if self.label == "CellN"
else self.select_fill_color
                #Aqui es el rellenar de shape
                #painter.fillPath(line_path, color)
                painter.fillPath(ellipse_path, color)
                painter.restore()
                if not self.isXML:
                    self.points[4] = QPointF(fstMinorPoint_x, fstMinorPoint
_y) if fstMinorPoint_x < endMinorPoint_x else QPointF(endMinorPoint_x, endM
inorPoint_y)
                    self.points[6] = QPointF(endMinorPoint_x, endMinorPoint
_y) if fstMinorPoint_x < endMinorPoint_x else QPointF(fstMinorPoint_x, fstM
inorPoint_y)
                self.drawVertex(vrtx_path_minor, 4)
                self.drawVertex(vrtx_path_minor, 6)

                # For recalculate points after moving
                self.isXML = False

                #ellipse_path.arcTo(self.points[0].x(), self.points[0].y(),
 self.points[1].x() - self.points[0].x(), self.points[1].y() - self.points[
0].y(), 0.0, 360.0)
            #painter.drawEllipse(((self.points[1].x() - self.points[0].x())
/2)+ self.points[0].x(),((self.points[1].y() - self.points[0].y())/2)+ self
.points[0].y(),self.points[1].x() - self.points[0].x(),self.points[1].y() -
 self.points[0].y())
            #painter.drawPath(line_path)
            # painter.drawPath(ellipse_path)
```

```
            #painter.drawPath(line_path)
            print("He hecho drawPath de line_path")
            #aqui he hecho cambios poniendo ellipse
            #painter.drawPath(vrtx_path)
            #painter.fillPath(ellipse_path, self.vertex_fill_color)
            painter.fillPath(vrtx_path, self.vertex_fill_color)
            painter.fillPath(vrtx_path_minor, self.vertex_fill_color_minor)

            #Try delete the extra-point and store the main 4
            #if len(self.points) == 8:

            # Draw text at the top-left
            if self.paintLabel:
                min_x = sys.maxsize
                min_y = sys.maxsize
                for point in self.points:
                    min_x = min(min_x, point.x())
                    min_y = min(min_y, point.y())
                if min_x != sys.maxsize and min_y != sys.maxsize:
                    font = QFont()
                    font.setPointSize(8)
                    font.setBold(True)
                    painter.setFont(font)
                    if(self.label == None):
                        self.label = ""
                    if(min_y < MIN_Y_LABEL):
                        min_y += MIN_Y_LABEL
                    painter.drawText(min_x, min_y, self.label)

            if self.fill:
                print("rellenar shape")
                # color = self.select_fill_color if self.selected else self
.fill_color
                # #Aqui es el rellenar de shape
                # #painter.fillPath(line_path, color)
                # painter.fillPath(ellipse_path, color)
```

Labelfile.py:

```
def savePascalVocFormat(self, filename, shapes, imagePath, imageData,
                            lineColor=None, fillColor=None, databaseSrc=Non
e):
        imgFolderPath = os.path.dirname(imagePath)
        imgFolderName = os.path.split(imgFolderPath)[-1]
        imgFileName = os.path.basename(imagePath)
        #imgFileNameWithoutExt = os.path.splitext(imgFileName)[0]
```

```
        # Read from file path because self.imageData might be empty if savi
ng to
        # Pascal format
        image = QImage()
        image.load(imagePath)
        imageShape = [image.height(), image.width(),
                      1 if image.isGrayscale() else 3]
        writer = PascalVocWriter(imgFolderName, imgFileName,
                                 imageShape, localImgPath=imagePath)
        writer.verified = self.verified
        print("savePascalVocFormat")
        for shape in shapes:
            points = shape['points']
            p = points
            points[1] = p[2]
            points[2] = p[4]
            points[3] = p[6]
            points.pop(7)
            points.pop(6)
            points.pop(5)
            points.pop(4)
            label = shape['label']
            print(points)
            # Add Chris
            difficult = int(shape['difficult'])
            bndbox = LabelFile.convertPoints2BndBox(points)
            writer.addBndBox(bndbox[0], bndbox[1], bndbox[2], bndbox[3], bn
dbox[4], bndbox[5], bndbox[6], bndbox[7], label, difficult)

        writer.save(targetFile=filename)
        return


def convertPoints2BndBox(points):
        xmin = float('inf')
        ymin = float('inf')
        xmax = float('-inf')
        ymax = float('-inf')
        xPoint3 = float('inf')
        yPoint3 = float('inf')
        xPoint4 = float('inf')
        xPoint4 = float('inf')

        P = points[0]
        xmin = P[0]
        ymin = P[1]

        P = points[1]
```

```
        xmax = P[0]
        ymax = P[1]

        P = points[2]
        xPoint3 = P[0]
        yPoint3 = P[1]

        P = points[3]
        xPoint4 = P[0]
        yPoint4 = P[1]

        if xmin < 1:
            xmin = 1

        if ymin < 1:
            ymin = 1

        return (int(xmin), int(ymin), int(xmax), int(ymax), int(xPoint3), i
nt(yPoint3), int(xPoint4), int(yPoint4))
```

xmlWritter.py:

```
#Recibe los puntos de la ellipse calculados por matlab y genera un XML que
puede reconocer el programa de anotación
from xml.dom import minidom
import xml.etree.ElementTree as ET
from xml.dom import minidom
import xml.etree.ElementTree as ET
import scipy.io as sc

def xmlWritter(filename_ima, path_file, height_ima, width_ima):
    annotation = ET.Element('annotation')
    #items = ET.SubElement(data, 'items')
    folder = ET.SubElement(annotation, 'folder')
    folder.text = 'ICS'
    filename = ET.SubElement(annotation, 'filename')
    filename.text = filename_ima
    path = ET.SubElement(annotation, 'path')
    path.text = path_file + filename_ima
    source = ET.SubElement(annotation, 'source')
    database = ET.SubElement(source, 'database')
    database.text = 'Unkwown'
    size = ET.SubElement(annotation, 'size')
    width = ET.SubElement(size, 'width')
    width.text = str(width_ima)
    height = ET.SubElement(size, 'height')
    height.text = str(height_ima)
```

```
depth = ET.SubElement(size, 'depth')
depth.text = '3'
segmented = ET.SubElement(annotation, 'segmented')
segmented.text = '0'

#Load mat.file
points = sc.loadmat(filename_ima + '.mat')
points_array = points['points']
#Iterate over the matrix to get the points
#Object elemens --> annotations


for i in range(len(points_array)):

    object = ET.SubElement(annotation, 'object')
    name = ET.SubElement(object, 'name')
    if (points_array[i][8] == 0):
        name.text = 'Cell'
    elif (points_array[i][8] == 1):
        name.text = 'CellN'
    pose = ET.SubElement(object, 'pose')
    pose.text = 'Unspecified'
    truncated = ET.SubElement(object, 'truncated')
    truncated.text = '0'
    difficult = ET.SubElement(object, 'difficult')
    difficult.text = '0'
    bndbox = ET.SubElement(object, 'bndbox')
    xmin = ET.SubElement(bndbox, 'xmin')
    ymin = ET.SubElement(bndbox, 'ymin')
    xmax = ET.SubElement(bndbox, 'xmax')
    ymax = ET.SubElement(bndbox, 'ymax')
    xpoint3 = ET.SubElement(bndbox, 'xpoint3')
    ypoint3 = ET.SubElement(bndbox, 'ypoint3')
    xpoint4 = ET.SubElement(bndbox, 'xpoint4')
    ypoint4 = ET.SubElement(bndbox, 'ypoint4')

    #Write point values
    xmin.text = str(points_array[i][0])
    xmax.text = str(points_array[i][2])
    xpoint3.text = str(points_array[i][4])
    xpoint4.text = str(points_array[i][6])
    ymin.text = str(points_array[i][1])
    ymax.text = str(points_array[i][3])
    ypoint3.text = str(points_array[i][5])
    ypoint4.text = str(points_array[i][7])
    mydata = ET.tostring(annotation)
    myfile = open(filename_ima + '.xml', "wb")
    myfile.write(mydata)
```

```
def xmlReader(filename_ima):

    #Parse xml file and get the points of the ellipses
    #Generate a .mat file and store them

    tree = ET.parse(filename_ima + '.xml')
    points = np.zeros(8)
    elipses = []
    for object in tree.findall('object'): #Getting all bnbbox of object
        points = {}
        bndbox = object.find('bndbox')
        x1 = bndbox.find('xmin').text
        x2 = bndbox.find('ymin').text
        x3 = bndbox.find('xmax').text
        x4 = bndbox.find('ymax').text
        x5 = bndbox.find('xpoint3').text
        x6 = bndbox.find('ypoint3').text
        x7 = bndbox.find('xpoint4').text
        x8 = bndbox.find('ypoint4').text

        points['bndbox'] = [int(x1),int(x2),int(x3),int(x4),int(x5),int(x6)
,int(x7),int(x8)]
        elipses.append(points)

    sc.savemat(filename_ima + '.mat',{'elipses':elipses})
```

ellipseGenerator.py:

```
#ellipseGenerator
from xml.dom import minidom
import xml.etree.ElementTree as ET

annotation = ET.Element('annotation')
#items = ET.SubElement(data, 'items')
folder = ET.SubElement(annotation, 'folder')
folder.text = 'ICS'
filename = ET.SubElement(annotation, 'filename')
filename.text = '82 KI67 (1, x=36173, y=138412, w=1690, h=1674).png'
path = ET.SubElement(annotation, 'path')
path.text = r'C:\Users\Josemaría 7\Documents\UPC\TFG\ICS\82 KI67 (1, x=3617
3, y=138412, w=1690, h=1674).png'
source = ET.SubElement(annotation, 'source')
database = ET.SubElement(source, 'database')
database.text = 'Unkwown'
size = ET.SubElement(annotation, 'size')
width = ET.SubElement(size, 'width')
width.text = '1690'
height = ET.SubElement(size, 'height')
```

```
height.text = '1674'
depth = ET.SubElement(size, 'depth')
depth.text = '3'
segmented = ET.SubElement(annotation, 'segmented')
segmented.text = '0'

#Object elemens --> annotations
#Generate fake ellipses
numEllipses = 1000
count_x = 0
count_y = 0
for ellipse in range(numEllipses):
    object = ET.SubElement(annotation, 'object')
    name = ET.SubElement(object, 'name')
    name.text = 'Cell'
    pose = ET.SubElement(object, 'pose')
    pose.text = 'Unspecified'
    truncated = ET.SubElement(object, 'truncated')
    truncated.text = '0'
    difficult = ET.SubElement(object, 'difficult')
    difficult.text = '0'
    bndbox = ET.SubElement(object, 'bndbox')
    xmin = ET.SubElement(bndbox, 'xmin')
    ymin = ET.SubElement(bndbox, 'ymin')
    xmax = ET.SubElement(bndbox, 'xmax')
    ymax = ET.SubElement(bndbox, 'ymax')
    xpoint3 = ET.SubElement(bndbox, 'xpoint3')
    ypoint3 = ET.SubElement(bndbox, 'ypoint3')
    xpoint4 = ET.SubElement(bndbox, 'xpoint4')
    ypoint4 = ET.SubElement(bndbox, 'ypoint4')

    widthE = 60
    heightE = 70
    avance_x = count_x*widthE
    avance_y = count_y*heightE
    if avance_x + widthE > 1690:
        count_x = 0
        count_y += 1
        avance_y = count_y*heightE
    count_x += 1

    if avance_y + heightE > 1674 and avance_x + widthE >1690:
        break

    xmin.text = str(10 + avance_x)
    xmax.text = str(24 + avance_x)
    xpoint3.text = str(5 + avance_x)
    xpoint4.text = str(29 + avance_x)
```

```
    ymin.text = str(46 + avance_y)
    ymax.text = str(10 + avance_y)
    ypoint3.text = str(23 + avance_y)
    ypoint4.text = str(32 + avance_y)
print(ellipse)
mydata = ET.tostring(annotation)
myfile = open("82 KI67 (1, x=30729, y=142829, w=1690, h=1674).xml", "wb")
myfile.write(mydata)
```

Matlab:

Annotate.m

```
close all
clear

List = dir('*KI67*');

%Loop on images
fsco = zeros(1,length(List));
for nima = 1:length(List)
    filename = List(nima).name;
    [filepath,filename_ima,ext] = fileparts(filename);
    if ismember(ext,{'.png'})
        ima_analize = demo2_k67_Watershed_02_function(filename);
        fsco(nima) = fscore(filename_ima);
    end
end

Fscore_mean = mean(fsco);
```

Demo2_k67_Watershed_02_function.m

```
function ima_analyze = demo2_k67_Watershed_02_function(filename)
```

```
% Read images
    ima = imread(filename);
    imaGr = ima(:,:,1);

    height = size(ima,1);
    width = size(ima,2);
  %% Simplification (opening and closing by reconstruction)
    size1 = 15;
    SE = strel('disk',size1,4);
    tmp = imopen(imaGr,SE);
    ima2 = imreconstruct(tmp,imaGr);

    size2 = 3;
    SE = strel('disk',size2,4);
    tmp = imclose(ima2,SE);
    imaSimple = 255-imreconstruct(255-tmp,255-ima2);
  %% Marker Nucleus
    contrast = 55;
    MaskN = imextendedmin(imaSimple,contrast,4);
  %% Marker Background (watershed of Nucleus)
    tmp = imimposemin(imaSimple,MaskN,8);
    tmp1 = watershed(tmp,8);
    MaskB = zeros(size(tmp1));
    MaskB(tmp1==0) = 1;
  %% Combined Marker
    tmp = imdilate(MaskB,ones(3,3)); % Prevent the background marker to
touch the Nucleus marker
    MaskN(tmp==1) = 0;
    Mask = MaskB + MaskN;
  %% Watershed of the iamge gradient
    grad = imdilate(imaSimple,ones(3,3)) - imerode(imaSimple,ones(3,3));
    tmp  = imimposemin(grad,Mask,8);
    imaLab = watershed(tmp,8);
    % Remove background regions
    tmp1 = imreconstruct(uint8(MaskB),uint8(imaLab),4);
    imaLab(tmp1>0)= 0;
  %% Clean the segmentation
    imaPred = zeros(size(imaLab));
    imaPred(imaLab>0) = 1;
    size3 = 1;
    SE = strel('disk',size3,4);
    imaPred = imclose(imaPred,SE);
     %% Second iteration to extract possible cells that were missed in
the first iteration
    if (1)
        imaGr1 = imaGr;
        imaGr1(imaPred>0) = 255;
        %% Simplification (opening and closing by reconstruction)
        size1 = 45;
        SE = strel('disk',size1,4);
        tmp = imopen(imaGr1,SE);
        ima2 = imreconstruct(tmp,imaGr1,4);

        size2 = 3;
        SE = strel('disk',size2,4);
        tmp = imclose(ima2,SE);
        imaSimple = 255-imreconstruct(255-tmp,255-ima2);
        %% Marker Nucleus
        contrast1 = 25;
```

```
        if (0)
            MaskN = imextendedmin(imaSimple,contrast1);
        else % Better to be precise and see where the marker and
reconstruction are the same
            tmp = imhmin(imaSimple,contrast1,4);
            MaskN = zeros(size(imaGr));
            MaskN(tmp == (imaSimple + contrast1)) = 1;
        end
        BW = imregionalmin(imaSimple,4);
        MaskN(BW==0) = 0;
        %% Marker Background
        tmp = imimposemin(imaSimple,MaskN,8);
        tmp1 = watershed(tmp,8);
        MaskB = zeros(size(tmp1));
        MaskB(tmp1==0) = 1;

        %% Combined Marker
        tmp = imdilate(MaskB,ones(3,3));
        MaskN(tmp==1) = 0;
        Mask = MaskB + MaskN;

        %% Watershed
        tmp = imaSimple;
        tmp(imaPred>0) = 255;
        grad = imdilate(tmp,ones(3,3)) - imerode(tmp,ones(3,3));
        tmp  = imimposemin(grad,Mask,8);
        imaLab1 = watershed(tmp,8);
        % Remove background regions
        tmp1 = imreconstruct(uint8(MaskB),uint8(imaLab1),4);
        imaLab1(tmp1>0)= 0;

        %% Clean the segmentation
        imaPred1 = zeros(size(imaLab1));
        imaPred1(imaLab1>0) = 1;
        imaPred1 = imclose(imaPred1,ones(3,3));
        imaPred = 2*imaPred + imaPred1;
        %imaPred(imaPred1==1) = 1;
        linkaxes
    end
    N1 = max(max(bwlabel(imaPred)));
    imaCont = ima;
    imaLabGr = imdilate(imaPred,ones(3,3)) - imerode(imaPred,ones(3,3));
    for i=1:size(imaCont,1)
    for j=1:size(imaCont,2)
        if(imaLabGr(i,j)~=0)
            imaCont(i,j,1)=0;
            imaCont(i,j,2)=255;
            imaCont(i,j,3)=0;
        end
    end
    end
    %% Analysis
    imaPred(imaPred>0) = 1;
    imaDist = bwdist(1-imaPred);
    imaDist = - imaDist;
    contrastDist = 0.8;
    imaDist1 = imaDist;
    imaDist1(imaDist>-3.0) = 0;
    tmp = imextendedmin(imaDist1,contrastDist,8);
```

```
    %imaDist(tmp>0) = 0;
    imaDist = imimposemin(imaDist,tmp,8);
    tmp1 = watershed(imaDist,8);
    imaPred1 = imaPred;
    imaPred1(tmp1==0) = 0;
    imaLab = bwlabel(imaPred1);

    stats = regionprops(imaLab,'Area','Centroid', 'MajorAxisLength',...
        'MinorAxisLength','Orientation', 'Eccentricity');
    areaVal = cat(1,stats.Area);
    % Compute average color
    for k = 1:length(stats)
            stats(k).AverageColor = [0;0;0];
    end
    for i=1:size(imaCont,1)
    for j=1:size(imaCont,2)
        if imaLab(i,j) ~= 0
            stats(imaLab(i,j)).AverageColor(1) =
stats(imaLab(i,j)).AverageColor(1) + double(ima(i,j,1));
            stats(imaLab(i,j)).AverageColor(2) =
stats(imaLab(i,j)).AverageColor(2) + double(ima(i,j,2));
            stats(imaLab(i,j)).AverageColor(3) =
stats(imaLab(i,j)).AverageColor(3) + double(ima(i,j,3));
        end
    end
    end
    for k = 1:length(stats)
            stats(k).AverageColor = stats(k).AverageColor/stats(k).Area;
    end

    % Select cells
    for k = 1:length(stats)
        stats(k).Status = 1;
        if stats(k).Area < 90
            stats(k).Status = 0;
        end
    end
    k1 = 1;
    statsF = stats(1);
    for k = 1:length(stats)
        if stats(k).Status == 1
            statsF(k1) = stats(k);
            k1 = k1+1;
        end
    end
    averageSize = median(cat(1,statsF.Area));

%     [idx,C]= kmeans(cat(2,statsF(:).AverageColor)',2);
%     C

    brownVal = zeros(3,length(statsF));
    areaVal  = zeros(1,length(statsF));
    eccentVal= zeros(1,length(statsF));
    meanVal  = zeros(1,length(statsF));
    for k = 1:length(statsF)
        brownVal(:,k) = statsF(k).AverageColor;
        areaVal(k)    = statsF(k).Area;
        eccentVal(k) = statsF(k).Eccentricity;
        meanVal(k)    = (brownVal(1,k)+brownVal(2,k)+brownVal(3,k))/3;
```

```
    end

    R1 = 200; B1 = 200;
    R2 = 100; B2 = 150;
    Alpha = - 1 * (B1-B2)/(R1-R2);
    Beta  = (R1 * B2 - R2 * B1)/(R1 - R2);
    F1 = figure('visible','off');
    set(F1,'Resize','off')
    t = linspace(0,2*pi,50);
    imshow(ima)
    hold on

    points = zeros(length(statsF),9);
    for k = 1:length(statsF)
        a = statsF(k).MajorAxisLength/2;
        b = statsF(k).MinorAxisLength/2;
        Xc = statsF(k).Centroid(1);
        Yc = statsF(k).Centroid(2);
        phi = deg2rad(-statsF(k).Orientation);
        x = Xc + a*cos(t)*cos(phi) - b*sin(t)*sin(phi);
        y = Yc + a*cos(t)*sin(phi) + b*sin(t)*cos(phi);

        %Calculo vertices para xml

        x1 = round(Xc + a*cos(pi)*cos(phi) - b*sin(pi)*sin(phi));
        y1 = round(Yc + a*cos(pi)*sin(phi) + b*sin(pi)*cos(phi));

        x2 = round(Xc + a*cos(0)*cos(phi) - b*sin(0)*sin(phi));
        y2 = round(Yc + a*cos(0)*sin(phi) + b*sin(0)*cos(phi));

        x3 = round(Xc + a*cos(pi/2)*cos(phi) - b*sin(pi/2)*sin(phi));
        y3 = round(Yc + a*cos(pi/2)*sin(phi) + b*sin(pi/2)*cos(phi));

        x4 = round(Xc + a*cos(3*pi/2)*cos(phi) -
b*sin(3*pi/2)*sin(phi));
        y4 = round(Yc + a*cos(3*pi/2)*sin(phi) +
b*sin(3*pi/2)*cos(phi));

        vec = [x1 y1 x2 y2 x3 y3 x4 y4];
        vec(vec < 0) = 0; %Really necesary?
        points(k,(1:8)) = vec;

        if brownVal(3,k) + Alpha * brownVal(1,k) > Beta
            plot(x,y,'g','Linewidth',2)
            points(k,9) = 0;
        else
            plot(x,y,'r','Linewidth',2)
            points(k,9) = 1;
        end
    end
    %Get Current figure and take it as a frame
    %F = getframe(F1);
    export_fig test.jpg -native
    hold off
    ima_analyze = imread('test.jpg');

    %Call py function to generate XML annotation
```

```
    points = uint16(points); %wih 16 bits we can afford pixel values and
be efficient
    [filepath,filename_ima,ext] = fileparts(filename);
    mat_file_name = strcat(filename_ima,'.mat');
    save(strcat(mat_file_name),'points','-v7')
    path_file = pwd; %se toma el actual pero puede cambiar al del
archivo
    py.xmlWritter.xmlWritter(filename_ima,path_file,height,width);
    delete(fullfile(path_file,mat_file_name));

end
```

fscore.m

```
function Fscore = fscore(filename_ima)
[points_orig, points] = xmlParser2points(filename_ima);
mask_orig = printcells(filename_ima,points_orig);
mask_corregida = printcells(filename_ima,points);

%fscore
m = find(mask_corregida);
[rows, cols] = size(m);
T = rows;
l = find(mask_orig);
[rows2, cols2] = size(l);
P = rows2;
TP = 0;
mask_dif = zeros(size(mask_orig,1),size(mask_orig,2),3);
mask_dif = uint8(mask_dif);

for i = 1:size(mask_dif,1)
    for j = 1:size(mask_dif,2)
        if and(mask_orig(i,j) == 1, mask_corregida(i,j) == 1)
            TP = TP + 1;
        elseif and(mask_orig(i,j) == 0, mask_corregida(i,j) == 1)
            blue = [0 0 255];
            mask_dif(i,j,:) = blue;
        elseif and(mask_orig(i,j) == 1, mask_corregida(i,j) == 0)
            yellow = [255 255 0];
            mask_dif(i,j,:) = yellow;
        end
    end
end
figure
imshow(mask_dif)
Precision = TP/P;
Recall = TP/T;
Fscore = 2*(Precision*Recall)/(Precision + Recall);
end
```

xmlParser2Points.m

```
function [points_orig,points] = xmlParser2points(filename_ima)

    %First read the xml's
    filename_xml_orig = strcat(filename_ima,'_orig');
    mat_file_name_orig = strcat(filename_xml_orig,'.mat');
```

```
    mat_file_name = strcat(filename_ima,'.mat');
    py.xmlWritter.xmlReader(filename_xml_orig);
    py.xmlWritter.xmlReader(filename_ima);

    %load the .mat generated
    dict_py_orig = load(mat_file_name_orig);
    dict_py = load(mat_file_name);

    points_orig = zeros(size(dict_py_orig.elipses,2),8);
    points = zeros(size(dict_py.elipses,2),8);
    %Parse the struct into matrix
    for i = 1:size(dict_py_orig.elipses,2)
        vec = dict_py_orig.elipses{1,i}.bndbox;
        points_orig(i,:) = vec;
    end

    for i = 1:size(dict_py.elipses,2)
        vec = dict_py.elipses{1,i}.bndbox;
        points(i,:) = vec;
    end

end
```

printcells.m

```
function result = printcells(filename_ima,points)
    %%Generate mask of annotation

    ima=imread(strcat(filename_ima,'.png'));

    h=size(ima,1);
    w=size(ima,2);

    mask=zeros(h,w);
    for el = 1:size(points,1)

        m =(points(el,4)-points(el,2))/(points(el,3)-points(el,1));
        phi=atan(double(m));

        Xc=(points(el,3)-points(el,1))/2 + points(el,1);
        Yc=(points(el,4)-points(el,2))/2 + points(el,2);

        t = linspace(0,2*pi,200);

        rx=(points(el,1) - points(el,3))^2;
        ry=(points(el,2) - points(el,4))^2;
        lx=(points(el,5) - points(el,7))^2;
        ly=(points(el,6) - points(el,8))^2;

        a=sqrt(double(rx+ry))/2;
        b=sqrt(double(lx+ly))/2;

        x = Xc + a*cos(t)*cos(phi) - b*sin(t)*sin(phi);
        y = Yc + a*cos(t)*sin(phi) + b*sin(t)*cos(phi);
        x=uint16(x);
        y=uint16(y);
```

```
        for i = 1:200
            if and(and(x(i)<w,x(i)>0),and(y(i)<h,y(i)>0))
                mask(y(i),x(i))=1;
            end
        end
    end

    mask1 = padarray(mask,[1 1],1,'pre');
    mask1f = imfill (mask1, 'holes');
    mask1f = mask1f(2:end,2:end);
    mask2 = padarray(padarray(mask,[1 0],1,'pre'),[0 1],1,'post');
    mask2f = imfill (mask2, 'holes');
    mask2f = mask2f(2:end,1:end-1);
    mask3 = padarray(mask,[1 1],1,'post');
    mask3f = imfill (mask3, 'holes');
    mask3f = mask3f(1:end-1,1:end-1);
    mask4 = padarray(padarray(mask,[1 0],1,'post'),[0 1],1,'pre');
    mask4f = imfill (mask4, 'holes');
    mask4f = mask4f(1:end-1,2:end);
    result = mask1f | mask2f | mask3f | mask4f;
end
```

Glosario

- GUI: Graphical User Interfaces
- AECC: Asociación Española Contra el Cáncer
- 3D: Tri-dimensional
- RGB: Red Green and Blue; hace referencia al espacio de color de la imagen
- JPG: Joint Photographic Experts Group; hace referencia al estándar de compresión
- XML: eXtensible Markup Language