Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Reconocimiento de parásitos de leishmania en imágenes de microscopio

Leishmania parasite detection in microscopic images

Jorge Acevedo de León

La Laguna, 8 de julio de 2022

D. **D. Christopher Expósito Izquierdo**, con N.I.F. 78.851.649-J profesor Ayudante Doctor de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

D. **D. Rafael Arnay del Arco**, con N.I.F. 78.569.591-G profesor Contratado Doctor de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de La Laguna, como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

”*Reconocimiento de parásitos de leishmania en imágenes de microscopio*”

ha sido realizada bajo su dirección por D. **Jorge Acevedo de León**, con N.I.F. 42.226.917-R.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 8 de julio de 2022

Agradecimientos

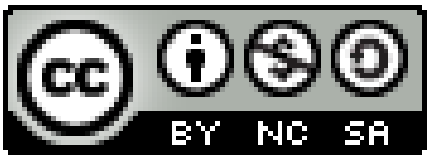
Agradezco a mis padres y familia por su confianza, apoyo y amor continuo durante toda mi vida académica.

A mis profesores por haberme permitido nutrirme de su conocimiento.

A mis amigos por compartir conmigo tantas, tan variopintas y valiosas experiencias vitales.

Y a todos por conformar los recuerdos y vivencias que han moldeado mi persona, fragmentos del pasado que nos mantendrán unidos hasta que el paso del tiempo nos transforme en polvo.

Licencia



Resumen

Leishmania es un género de protistas responsables de la enfermedad conocida como leishmaniasis o leishmaniosis. El principal vector de infección son los mosquitos de los géneros Phlebotomus (en Eurasia y África) y Lutzomyia (en América).

La leishmaniasis afecta a varios grupos de vertebrados, entre ellos, marsupiales, cánidos, roedores y primates, y se estima que afecta actualmente a 6 millones de personas en 98 países. Cada año se producen entre 0,9 y 1,6 millones de nuevos casos, y se conocen 21 especies que causan enfermedades en los seres humanos.

En el presente trabajo se propone un sistema de detección de una serie de características de estos organismos, como su cantidad, posición en la imagen y tamaño mediante técnicas de visión por computador. Las imágenes a tratar son tomadas mediante el uso de un microscopio óptico, en condiciones de aumento y luminosidad variables pero registradas en el nombre del fichero contenedor de la imagen.

Las imágenes de laboratorios son analizadas con éxito por el software, ya que los parásitos son detectados y medidos, las imágenes son marcadas para que las caracteristicas relevantes puedan ser discernidas a primera vista y se genera un documento con datos relevantes al análisis.

Palabras clave: Leishmania, Reconocimiento de patrones, Visión por computador, Python, OpenCV

Abstract

Leishmania is a genus of protists responsible for the disease known as leishmaniasis or leishmaniasis. The main vector of infection is mosquitoes of the genera Phlebotomus (in Eurasia and Africa) and Lutzomyia (in the Americas).

Leishmaniasis affects several vertebrate groups, including marsupials, canids, rodents and primates, and is currently estimated to affect 6 million people in 98 countries. Between 0.9 and 1.6 million new cases occur each year, and 21 species are known to cause disease in humans.

This work proposes a system for detecting a series of characteristics of these organisms, such as their number, position in the image and size, by means of computer vision techniques. The images to be processed are taken using an optical microscope, under variable magnification and luminosity conditions, but registered in the name of the file containing the image.

The laboratory images are successfully analyzed by the software, as the parasites are detected and measured, the images are marked so that the relevant features can be discerned at first glance and a document with data relevant to the analysis is generated.

Keywords: Leishmania, Pattern Recognition, Computer Vision, Python, OpenCV

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. ¿Que es la Leishmania?

La leishmaniasis es una enfermedad tropical y subtropical causada por un parásito intracelular que se transmite al ser humano por la picadura de mosca de la arena, principalmente Phlebotomus y Lutzomyia (Europa, norte de África, Oriente Medio, Asia y parte de Sudamérica) [2].

Según la Organización Mundial de la Salud (OMS), la leishmaniasis es una de las siete enfermedades tropicales más importantes y representa un grave problema sanitario mundial que presenta un amplio espectro de manifestaciones clínicas con un resultado potencialmente mortal.

Se encuentra en todos los continentes excepto en Oceanía y es endémica en zonas geográficas circunscritas en el noreste de África, el sur de Europa, Oriente Medio, el sureste de México y América Central y del Sur y América del Sur. Las características clínicas incluyen una amplia gama de manifestaciones con diferentes grados de gravedad que dependen de la especie de Leishmania implicada y de la respuesta inmunitaria del huésped.

1.2. Contexto histórico

La leishmaniasis tiene una larga historia que se remonta al siglo I de nuestra era. Ya en este periodo, la cerámica preincaica de Ecuador y Perú mostraba representaciones de lesiones cutáneas y deformidades faciales típicas de la leishmaniasis cutánea y mucocutánea. Los textos incas de los siglos XV y XVI y los relatos de los conquistadores españoles señalaban la presencia de lesiones cutáneas en los trabajadores agrícolas que regresaban de los Andes.

Estas úlceras se asemejaban a las lesiones de la lepra y fueron etiquetadas como "lepra blanca", "enfermedad de los Andes" o "enfermedad del valle". En África y en la India, los informes de mediados del siglo XVIII describen la enfermedad hoy conocida como leishmaniasis visceral, como "kal-azar" o "fiebre negra". En 1756, Alexander Russell realizó un importante avance en el descubrimiento de la leishmaniasis tras examinar a un paciente turco. Según Russell, " una vez cicatrizada, deja una fea cicatriz, que permanece toda la vida, y durante muchos meses tiene un color lívido. Cuando no se irritan, rara vez dan mucho dolor". Russell llamó a esta enfermedad "forúnculo de Alepo".

La enfermedad se conoció como Leishmaniasis después de que William Leishman, un médico de Glasgow que trabajaba con el ejército británico en la India describió una

de las primeras manchas de Leishmania en 1901. En Dum Dum, una ciudad cercana a Calcuta, Leishman descubrió cuerpos ovoides en el bazo de un soldado británico que sufría ataques de fiebre, anemia, atrofia muscular e hinchazón del bazo. Leishman describió esta enfermedad como "fiebre dum dumz publicó sus hallazgos en 1903. Charles Donovan también reconoció estos síntomas en otros pacientes de kal-azar y publicó su descubrimiento unas semanas después de Leishman. Tras examinar el parásito con la tinción de Leishman, estos amastigotes se conocieron como cuerpos de Leishman-Donovan y, oficialmente, esta especie pasó a denominarse L. Donovani. Al relacionar este protozoo con el kal-azar, Leishman y Donovan descubrieron el género Leishmanias[1].

1.3. Motivación

La investigación sobre Leishmania y parásitos afines producen grandes cantidades de imágenes de microscopio que, a su vez, requieren de gran cantidad de tiempo para ser analizadas. En un solo laboratorio, el número puede ascender fácilmente a miles de imágenes con sólo una docena de experimentos diferentes (ya que cada experimento produce normalmente más de un centenar de imágenes.

Esto no sólo impide a los investigadores explorar nuevas alternativas, ya que les roba tiempo útil, sino que también es propenso a la variación entre personas. Esto ocurre porque, aunque existen convenciones sobre el proceso de anotación, muchas imágenes son extremadamente complejas[2] y esto da lugar a recuentos finales diferentes.

Además, por la naturaleza del proceso de anotación, que consume mucho tiempo (y por tanto supone un esfuerzo mental), hace que exista una varianza intrapersonal, que se expresa como una función decreciente a lo largo del tiempo a medida que el sujeto se cansa, frustrado, aburrido o presionado para terminar cada anotación [3].

Todas estas razones llevan a la necesidad de métodos automáticos o semiautomáticos de clasificación y de imágenes, necesidad que se dispone a satisfacer el proyecto objeto de este documento.

1.4. Objetivos

El objetivo principal de este proyecto es el desarrollo de una aplicación Full-Stack para la gestión y análisis de imágenes de microscopio, una herramienta que los investigadores del Centro de Enfermedades Tropicales puedan usar en su día a día para acelerar su flujo de trabajo, reduciendo el tiempo que les toman dichas tareas. El análisis consiste en la detección de la posición de los parásitos en la imagen, su tamaño, morfología y agrupamientos.

La salida esperada de este procedimiento consiste en una copia de la imagen original en la que se encuentren resaltados los parásitos, además de un fichero contenedor de los parámetros de ejecución del algoritmo y los datos poblacionales e individuales de los parásitos hallados.

[1] Información extraída de la página oficial de la universidad de Standford: https://web.stanford.edu/group/parasites/ParaSites2006/Leishmaniasis/history.html:%7E:text=Leishmaniasis%20has%2

[2] Una imagen es compleja cuando el número de parásitos o la visibilidad de los mismos resultan en un alto margen de error en el juicio del investigador.

Capítulo 2

Antecedentes y estado del arte

2.1. Visión por computador

La visión por computador[1] es un campo de la inteligencia artificial (IA) que permite a los ordenadores y sistemas obtener información significativa a partir de imágenes digitales, vídeos y otras entradas visuales, y emprender acciones o hacer recomendaciones basadas en esa información. Si la IA permite a los ordenadores pensar, la visión por ordenador les permite ver, observar y comprender.

Pese a que la visión por computador funciona de manera similar a la humana, nosotros contamos con la ventaja de una vida entera de experiencia y entrenamiento en diferenciar objetos, distancias entre los mismos, velocidades y en encontrar detalles que nos indiquen que algo está mal en lo que vemos.

La visión por computador se ha expandido al vasto área de conocimiento que comprende desde la recopilación de datos en crudo hasta la extracción de patrones en imágenes y la posterior interpretación de la información.

El objetivo principal de la visión por computador es crear modelos y extractos de datos e información de las imágenes, mientras que el procesamiento de imágenes trata de implementar transformaciones computacionales para las mismas, como la nitidez o el contraste, entre otros. Funcionalmente, la visión por computador y la visión humana son lo mismo, tienen el objetivo de interpretar datos espaciales, es decir, datos indexados por más de una dimensión.

Tal y como los humanos usamos nuestros ojos y cerebros para comprender el mundo que nos rodea, la visión artificial trata de producir el mismo efecto para que los ordenadores puedan percibir y comprender una imagen o secuencia de imágenes y actuar según convenga en una determinada situación. Esta comprensión se consigue gracias a distintos campos como la geometría, la estadística, la física y otras disciplinas.

2.2. Estado del arte

El campo de la visión por computador ha vivido un auge en su expansión y desarrollo en los últimos años gracias a los avances en técnicas modernas de inteligencia artificial. En este apartado, se hará una breve revisión de las últimas aportaciones al campo específico de la detección de cuerpos en imágenes de microscopio.

En la publicación *Malaria Parasite Detection Using Deep Learning Methods* [4] por Kaustubh Chakradeo, Michael Delves y Sofya Titarenko, se lleva a cabo la implementación

[1] Whats Computer Vision?: https://www.ibm.com/topics/computer-vision

de un modelo de aprendizaje automático construido con un número relativamente pequeño de capas, lo que permite su ejecución en equipos con recursos limitados.

Si se compara, con modelos desarrollados anteriormente, este destaca por su mejoría en una gran amplitud de métricas de precisión. Utilizaron redes neuronales basadas en el modelo VGG[2], una estructura que demuestra buenos resultados para la tarea de clasificación de imágenes.

Otro trabajo que merece ser citado es *An Effective Approach for Robust Lung Cancer Cell Detection*[1] por Hao Pan, Zheng Xu, y Junzhou Huang. El marco propuesto está basado en la detección de células de cáncer de pulmón por píxeles sobre CNN[3]. Los portaobjetos de tejido de biopsia de pulmón teñido con hematoxilina y eosina se escanean con un aumento de 40x.

La red neuronal convolucional se entrena utilizando los datos de entrenamiento. En la parte de pruebas de detección de células la imagen de prueba se introduce en la CNN entrenada para obtener el mapa de salida y, a continuación, se aplica el método de análisis del momento de la imagen para obtener los centroides de células de cáncer de pulmón.

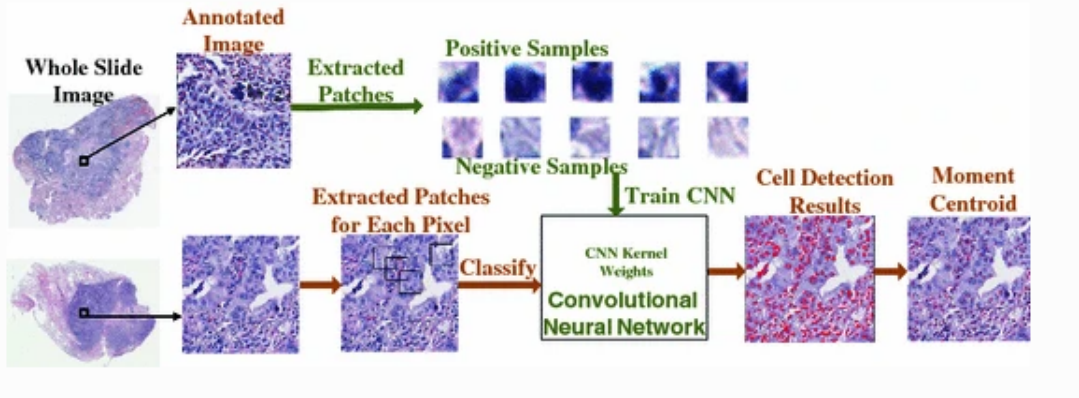


Figura 2.1: Esquema de funcionamiento del proyecto de detección cáncer de pulmón [1]

Investigadores de la universidad Shiraz de ciencias médicas en Irán han desarrollado un sistema capaz de detectar leishmaniasis en imágenes de microscopio mediante técnicas de machine learning, usando el algoritmo de Viola-Jones[4], consiguiendo un 65 % de memorización y un 50 % de precisión en la detección de macrófagos infectados con el parásito de leishmania [6].

En la Universidad Politécnica de Cataluña se han propuesto técnicas de clasificación y segmentación de parásitos utilizando redes convolucionales (U-net[5] en concreto) [7].

[2]El modelo VGG, o VGGNet, es un modelo de red neuronal convolucional propuesto por A. Zisserman y K. Simonyan de la Universidad de Oxford. Estos investigadores publicaron su modelo en el artículo de investigación titulado "Very Deep Convolutional Networks for Large-Scale Image Recognition"[5].

[3]Una red neuronal convolucional (CNN) es un tipo de red neuronal artificial que se utiliza principalmente para el reconocimiento y procesamiento de imágenes, debido a su capacidad para reconocer patrones en ellas. Una CNN es una herramienta potente, pero requiere millones de puntos de datos etiquetados para su entrenamiento.

[4]El algoritmo de Viola-Jones es un framework de detección facial (aunque adaptable a otras clases de objetos) robusto y eficaz, además de útil, ya que puede ser utilizado en casos en los que se disponga de poca potencia computacional, propuesto en 2001 por Paul Viola y Michael Jones

[5]U-NET es un modelo de red neuronal dedicado a tareas de visión artificial y más concretamente a problemas de segmentación semántica.

Para el entrenamiento de los modelos hicieron uso de imágenes previamente segmentadas (un recurso del cual no se ha dispuesto para la realización de este proyecto).

Capítulo 3

Desarrollo

3.1. Recursos físicos

Para la búsqueda de información y la redacción de la memoria se ha utilizado mi ordenador portátil personal: Acer aspire V3-571G con procesador Intel® Core™ i7-3632QM CPU @ 2.20GHz × 8, 8GB DDR3 RAM, y gráficos GeForce 710M/PCIe/SSE2.

Las imágenes que ha provisto el Instituto Universitario de Enfermedades Tropicales y Salud Pública de Canarias han sido tomadas con un microscopio Evos M5000[1].

Figura 3.1: Microscopio EVOS M5000.

El sistema de adquisición de imágenes EVOS M5000 es un microscopio digital de sobremesa invertido totalmente integrado para adquisición de imágenes en color, con fluorescencia de cuatro colores y luz transmitida. Características del microscopio:

- Fuente de luz LED
- Aumento 1,25X a 100X
- Objetivos Torreta de 5 posiciones
- Resolución 1920 x 1080 píxeles
- Pantalla LCD articulada de 47 cm (18,5 pulg)
- Dimensiones 33,02 cm x 53,34 cm x 40,64 cm (18 pulg. x 23 pulg. x 18 pulg.) [An × Al× Pr]

[1] Microscopio: https://www.thermofisher.com/order/catalog/product/AMF5000

3.2. Análisis de imágenes de microscopio

En esta sección, se llevará a cabo una descripción de la arquitectura del sistema implementado y el flujo de trabajo que se ha seguido para conseguir los resultados a los que se ha llegado. Para ello, se dividirá en sub secciones, una para cada paso del procesamiento en las qué se hará un análisis en profundidad de cada uno de ellos.

3.2.1. Estructura del procesamiento

Esta sección provee una descripción general de la estructura del programa desarrollado, apoyada en la figura que se expone a continuación.

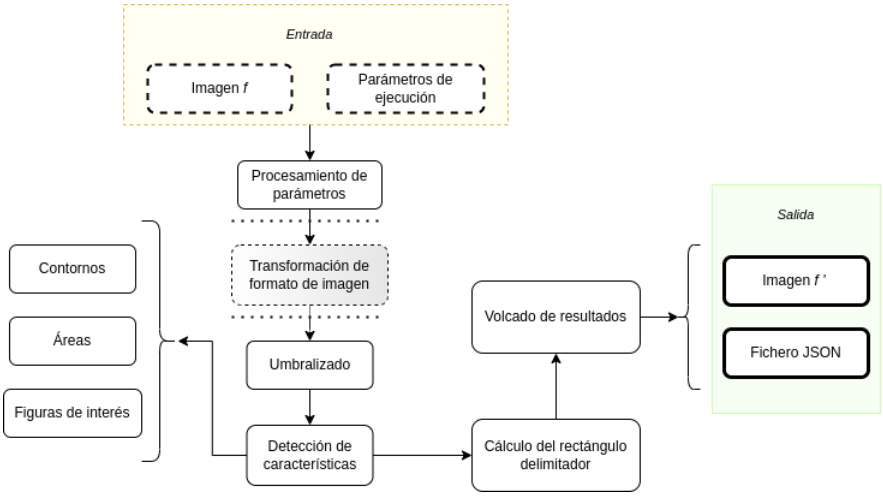


Figura 3.2: Esquema de flujo de trabajo del programa principal

El programa recibe una imagen f y comprueba si se encuentra en un formato soportado (TIFF, PNG) y a continuación se lleva a cabo la revisión de los parámetros de lanzamiento. A continuación, en función de los parámetros de lanzamiento, el formato de la imagen será transformado o no, ya que existe la opción de lanzar exclusivamente el script de transformación sobre los ficheros de un directorio, para así ahorrar tiempo de ejecución en caso de que se quieran llevar a cabo múltiples iteraciones del analizador con distintos parámetros para una misma imagen.

El umbralizado es el paso que sigue, procedimiento por el cual la información de la imagen se polariza a blanco y negro (valores de los pixeles de 0-255), utilizando los métodos de desenfoque gaussiano y umbralizado adaptativo implementados por OpenCV. Con la imagen umbralizada, entramos en la fase de detección de características, que comienza con la detección de contornos, la cual se encuentra también en la librería recién mencionada. Posteriormente se calculan las áreas de los contornos o figuras encontradas, áreas que se utilizan para el proceso de diferenciación pues los parásitos deben de tener un tamaño superior a un valor definido como parámetro para ser considerado como tal.

El último paso es el cálculo del bounding rectangle o rectángulo delimitador de cada uno de los cuerpos de interés hallados por los procedimientos anteriores. La imagen umbralizada es editada siendo añadidos colores que diferencian los contornos en función de su tamaño y los bounding rects calculados. Para finalizar, se vuelcan los datos obtenidos a lo largo en un fichero en formato JSON qué contiene datos estadísticos medios además de los valores de los parámetros de lanzamiento e información de cada uno de los parásitos encontrados.

3.2.2. Software utilizado

En cuanto a las herramientas software utilizadas, estas han sido diversas en las diferentes etapas del trabajo:

El lenguaje en el qué se ha desarrollado el programa es Python[2] en su versión 3.8, un lenguaje de programación de alto nivel multiparadigma, interpretado, dinámico y multiplataforma. Tiene una gran extensión de uso en el ámbito científico por su agradable curva de aprendizaje y la gran cantidad de herramientas disponibles publicadas por la comunidad.

Este proyecto ha sido llevado a cabo usando como núcleo OpenCV[3]. OpenCV (Open Source Computer Vision Library) es una librería de software de visión por ordenador y aprendizaje automático de código abierto. OpenCV se construyó para proporcionar una infraestructura común para las aplicaciones de visión por ordenador y para acelerar el desarrollo de sistemas dotados de percepción.

Para la carga inicial y conversión de las imágenes al formato con el qué serán posteriormente tratadas hago uso de dos librerías concebidas con este fin: en primer lugar Python Imaging Library (PIL[4]). PIL añade capacidades de procesamiento de imágenes a su intérprete de Python. Esta librería proporciona un amplio soporte de formatos de archivo, una representación interna eficiente y capacidades de procesamiento de imágenes bastante potentes. El núcleo de la biblioteca de imágenes está diseñado para un acceso rápido a los datos almacenados en unos pocos formatos básicos de píxeles y proporciona una base sólida para una herramienta general de procesamiento de imágenes.

El segundo módulo se trata de NumPy (Numerical Python)[5], un proyecto de código abierto cuyo objetivo es facilitar la computación numérica con Python. Fue creado en 2005, basándose en los primeros trabajos de las librerías Numeric y Numarray. NumPy siempre será un software 100 % de código abierto, de uso libre para todos y liberado bajo los términos liberales de la licencia BSD modificada.

Mientras estimaba los valores óptimos para el preprocesamiento de las imágenes, también he hecho uso de Matplotlib[6] una librería para crear visualizaciones estáticas, animadas e interactivas en Python.

En el lado del servidor, he hecho uso de Maven junto a Spring Boot utilizando Java como lenguaje de programación y la base de datos es de MongoDB. Para el cliente he hecho uso de Vue acompañado de Vuetify. Todas estás tecnologías se describen mejor en el siguiente capítulo.

He utilizado VSCode como entorno de desarrollo, debido a su versatilidad y la familiaridad que ya tenía con el.

Por último cabe mencionar el uso de cuatro paquetes integrados en Python: Statistics[7], para el cálculo de valores como la media o desviación típica, JSON[8], para trabajar con el formato de notación objeto de JavaScript (el cual será descrito en mayor profundidad posteriormente), OS[9] para la gestión de ficheros mediante llamadas al sistema y argparse[10]

[2] Python: https://www.python.org/
[3] OpenCV: https://opencv.org/about/
[4] PIL: https://pillow.readthedocs.io/en/stable/
[5] NumPy: https://numpy.org/about/
[6] Matplotlib: https://matplotlib.org/
[7] Statistics: https://docs.python.org/3/library/statistics.html
[8] JSON: https://www.mclibre.org/consultar/informatica/lecciones/formato-json.html
[9] OS: https://docs.python.org/3/library/os.html?highlight=os#module-os
[10] Argparse: https://docs.python.org/3/library/argparse.html?highlight=argparse#module-argparse

que facilita la escritura de interfaces de línea de comandos.

3.2.3. Procesamiento de parámetros

El procesamiento de los parámetros se lleva a cabo en el programa principal mediante el paquete integrado de Python argparse. Se trata de una gran alternativa a escribir un CLI (Command Line Interface) propio por su accesibilidad y sencillez, con pocas líneas de código se consiguen resultados que de otra forma requerirían de un mayor esfuerzo.

El programa cuenta con la siguiente lista de parámetros:

- -h, –help: Muestra un mensaje de ayuda y sale del programa.
- -i, –input: Ruta a la imagen que se desea procesar. Obligatorio.
- -o, –output: Ruta en la que guardar el fichero JSON resultante. Obligatorio.
- -a, –algorithm: Nombre del algoritmo de umbralización seleccionado, los disponibles ahora son: [adaptativo, simple, doble]. Obligatorio.
- -th, –threshold-value: Valor umbral para el procedimiento de umbralizado. (por defecto 100).
- -th2, –threshold-value-2: Segundo valor umbral para el procedimiento de umbralizado doble. (por defecto 120).
- -gk, –gaussian_kernel: Tamaño del núcleo gaussiano para el método de umbralizado. La anchura y la altura pueden ser diferentes, pero ambas deben ser positivas e impares. O bien, pueden ser ceros y entonces se calculan a partir de sigma. (por defecto: [11, 11]).
- -bs, –block-size: Tamaño de la vecindad de píxeles que se utiliza para calcular un valor de umbral para el píxel: 3, 5, 7, etc. (por defecto: 11)
- -c, –constant: Constante que se resta a la media o a la media ponderada (ver los detalles más abajo). Normalmente, es positivo, pero también puede ser cero o negativo. (por defecto: 2).
- -ms, –minimum-size: Valor mínimo de tamaño (área) para que los contornos detectados se consideran parásitos. Con el conjunto de imágenes actual, 40.0 parece ser un valor adecuado. (por defecto: 40.0).
- -mnd, –max-neighbourhood-distance: Valor máximo de distancia entre un centro de cluter potencial y sus vecinos. Con el conjunto de imágenes actual, 105.0 parece ser un valor muy adecuado. (por defecto: 105.0).
- -mcs, –min-cluster-size: Valor mínimo de tamaño de un cluster para ser considerado como tal. Con el conjunto de imágenes actual, 13 parece ser un valor adecuado. (por defecto: 13).
- –version: Muestra la versión del programa y sale del mismo.

Para asegurar que cualquier usuario hábil puede usar el programa, se implementa una bastante completa opción -h que dicta todos los argumentos, cuales son obligatorios, sus valores por defecto y una descripción breve y concisa de cada uno de ellos.

3.2.4. Transformación de formato

Las imágenes recibidas originalmente se encontraban en un formato de representación en 16 bits que ninguno de los métodos tradicionales de tratamiento de imágenes en Python eran capaces de abrir (PIL, OpenCV) pese a que los visores de imágenes eran capaces de visualizarlas sin ninguna queja. Por ello, a dichas imágenes se les ha aplicado una reconversión a 8 bits, mediante la cual la pérdida de información, pese a que existente, es prácticamente obviable.

3.2.5. Umbralizado

El umbralizado o *thresholding* consiste en un proceso mediante el cual los valores de los pixeles de una imagen en escala de grises son transformados o bien en blanco o en negro (valor 0 o 255). Este procedimiento, en su forma más simple requiere de un parámatro, el valor *umbral*. Todos los valores de los pixeles de la imágen serán comparados con el umbral: si son inferiores se convierten en 0 y son superiores en 255. A este procedimiento, se le conoce como umbralizado simple.

Sin embargo, cuando la iluminación no es completamente uniforme a lo largo de la imagen, contar con un criterio adicional para el umbralizado podría ser más beneficioso. Aquí entra el umbralizado adaptativo. En este tipo de umbralizado además se hace uso de un filtro de filtrado de ruido, en este caso, se han probado a usar el filtrado de la mediana y el filtrado gaussiano. A continuación, se describe como se ha hecho uso de las implementaciones de OpenCV de ambas aproximaciones en el proyecto.

Umbralizado simple

Con cierto tipo de imágenes de las que se ha dispuesto, se han conseguido resultados satisfactorios mediante esta aproximación, sobre todo en aquellas en las que la intesidad del color del interior del parásito es distinta a la del medio, como en el siguiente ejemplo:

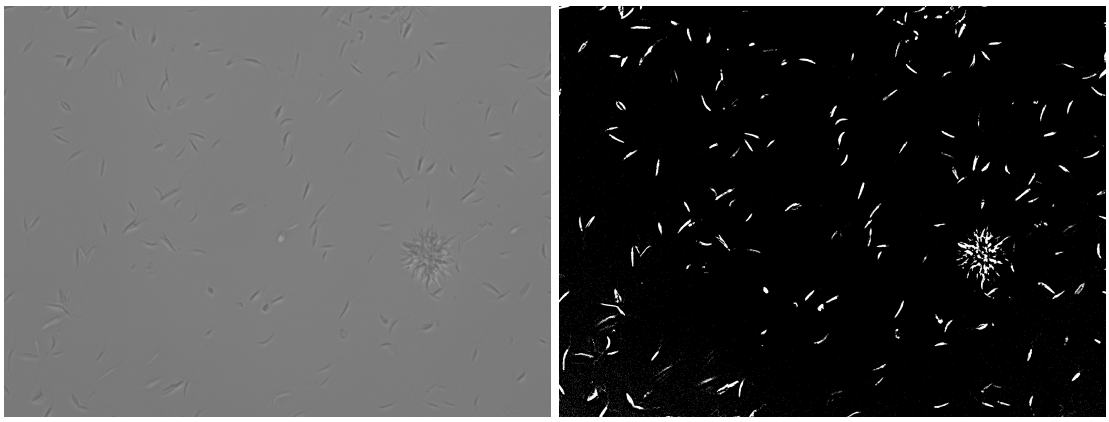


Figura 3.3: Umbralizado simple con umbral v = 118

Sin embargo y como se puede apreciar, aunque parece que el resultado es aceptable, tiene margen de mejora, aunque la iluminación parece uniforme en la imagen original, cuenta con una mayor intensidad en las esquinas inferiores, lo que provoca que en el umbralizado aparezcan ligeramente sombreadas. Además de esto, no se consigue aislar perfectamente a los parásitos y el umbral podría ser demasiado alto (o bajo) en algunas partes.

También hay que tener en cuenta que el valor de umbral utilizado no ha sido para nada arbitrario, si el valor que se toma no es preciso, los resultados varían su calidad entre poco acertados y mediocres:

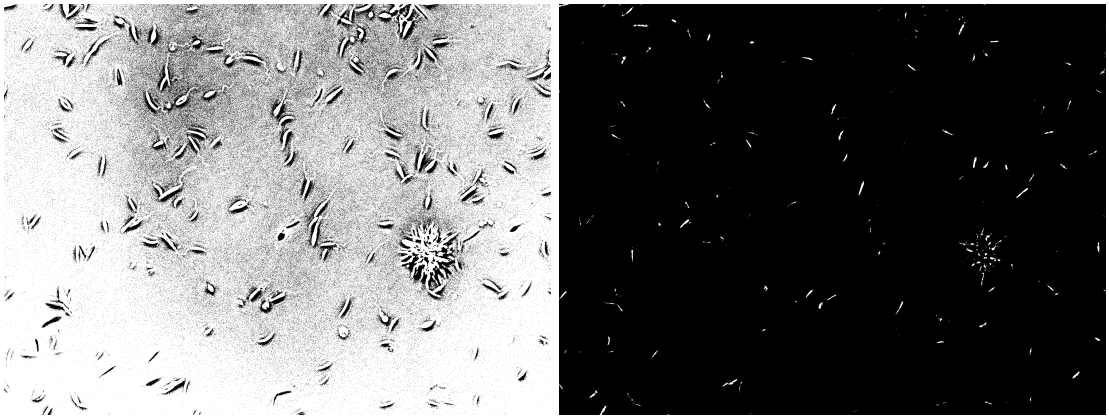


Figura 3.4: Umbralizado simple con umbrales v = 110, v = 128

Umbralizado doble

El umbralizado doble consiste en el aplicado de dos procedimientos de umbralización simple consecutivos y en sumar las imágenes resultantes de los mismos. Este método funciona especialmente bien en aquellas imágenes que cuentan con parásitos en dos tonalidades, una más clara y una más oscura. El resto de métodos fallan en detectar ambas ya que se acaban quedando con los parásitos de una u otra tonalidad pero no de ambas.

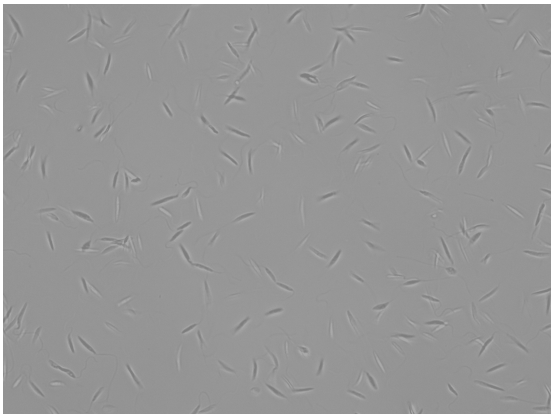


Figura 3.5: Imagen con parásitos en dos tonalidades de gris

Haciendo una fina selección de los parámetros de lanzamiento del algoritmo (en concreto de los valores umbral 1 y 2) podemos conseguir una imagen umbralizada que contemple a todos los parásitos presentes en la imagen original.

Umbralizado adaptativo

El umbralizado adaptativo se fundamenta en la idea de que se puede conseguir un mejor umbralizado en imágenes de iluminación variable si se determina el valor umbral con el que se compara cada píxel en función de los valores de intensidad de color de sus

píxeles vecinos. Como paso previo a la ejecución de este umbralizado y por recomendación de la documentación de OpenCV, se le aplica a la imagen en cuestión un filtrado gaussiano para reducir el ruido y ecualizar la imagen.

El filtrado gaussiano es un efecto muy utilizado en el software de gráficos, normalmente para reducir el ruido de la imagen y reducir los detalles. El efecto visual de esta técnica de desenfoque que recuerda al de ver la imagen a través de una pantalla translúcida, claramente diferente del efecto bokeh producido por una lente desenfocada o la sombra de un objeto bajo la iluminación habitual.

La implementación del umbralizado adaptativo de OpenCV cuenta con dos argumentos que nos serán de especial interés *blockSize* o tamaño de bloque que define el tamaño de la vecindad de píxeles que se contarán como vecinos y *C* que se trata de una constante que se resta a la media o a la suma ponderada de los píxeles de la vecindad.

Para el conjunto de datos con el se ha trabajado, la experimentación con estos parámatros lleva a la idea de que los valores *blockSize = 11* y *C = 3* llevan a buenos resultados iniciales.

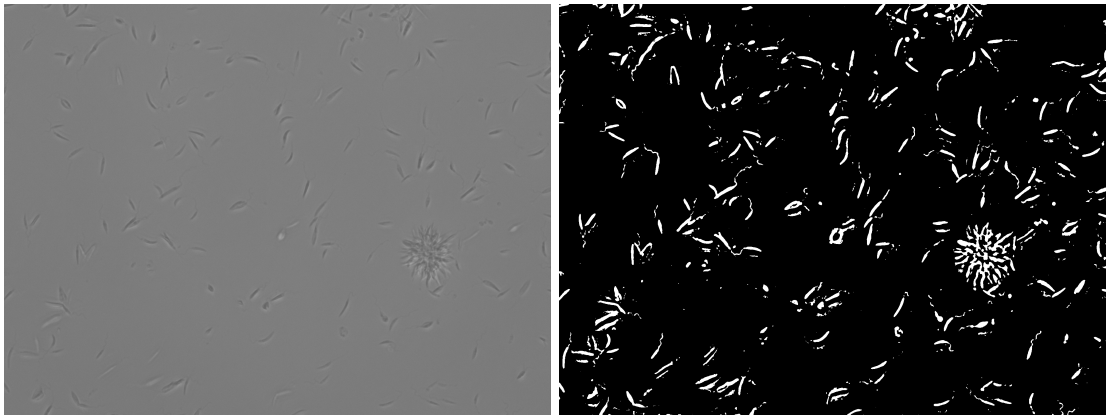


Figura 3.6: Umbralizado adaptativo con blockSize = 11, C = 3

3.2.6. Detección de características

El proceso de detección de carácterísticas se basa, mayoritariamente en el análisis de la imagen en busca de cuerpos cerrados, definidos por su contorno. El contorno de un cuerpo es el conjunto de píxeles que delimitan sus bordes.

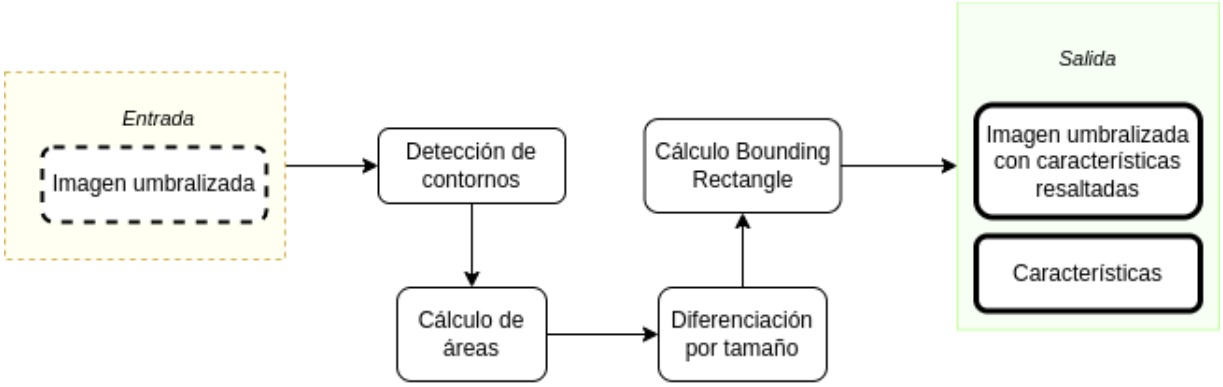


Figura 3.7: Esquema de proceso de detección de características

En OpenCV, encontrar contornos es como encontrar un objeto blanco en un fondo negro. Así que, el objeto a encontrar debe ser blanco y el fondo debe ser negro.

Para cumplir esta labor se hace uso del método *findContours* de OpenCV. Hay tres argumentos en la función, el primero es la imagen de la que se quiere encontrar los contornos, el segundo es el modo de recuperación del contorno, el tercero es el método de aproximación del contorno. Y da como resultado los contornos y la jerarquía. Una vez se cuenta con el array de contornos resultantes, se calcula el área de los mismos.

Se toman como características los contornos de interés, es decir, aquellos cuya área superen un valor umbral que se recibe como parámetro en tiempo de ejecución (40 por defecto). Por último, se calcula el *bounding rectangle* o rectángulo delimitador de cada característica mediante el método *minAreaRect* también perteneciente a OpenCV.

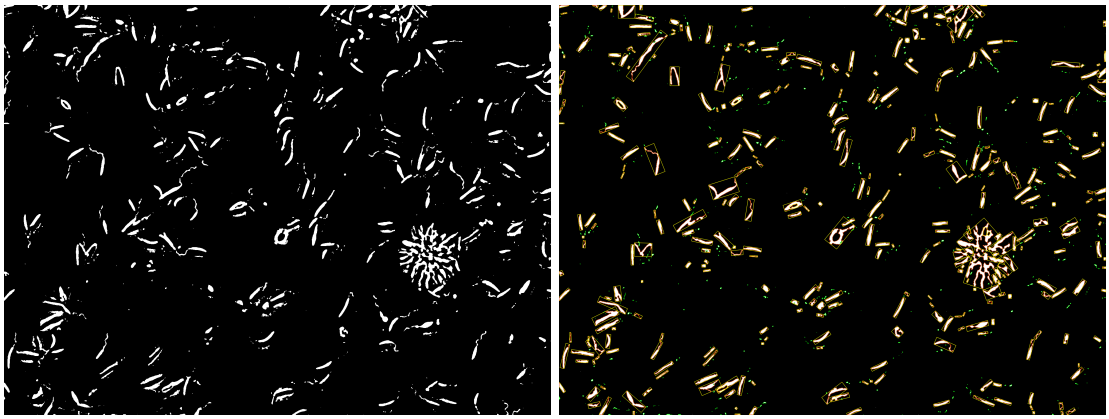


Figura 3.8: Imagen con características destacadas y bounding rect

3.2.7. Detección de agrupamientos

En las imágenes que han sido recibidas, existen regiones en las que los que existe una gran densidad de parásitos, aparentemente apilados unos sobre otros formando lo que se denomina agrupamiento o cluster.

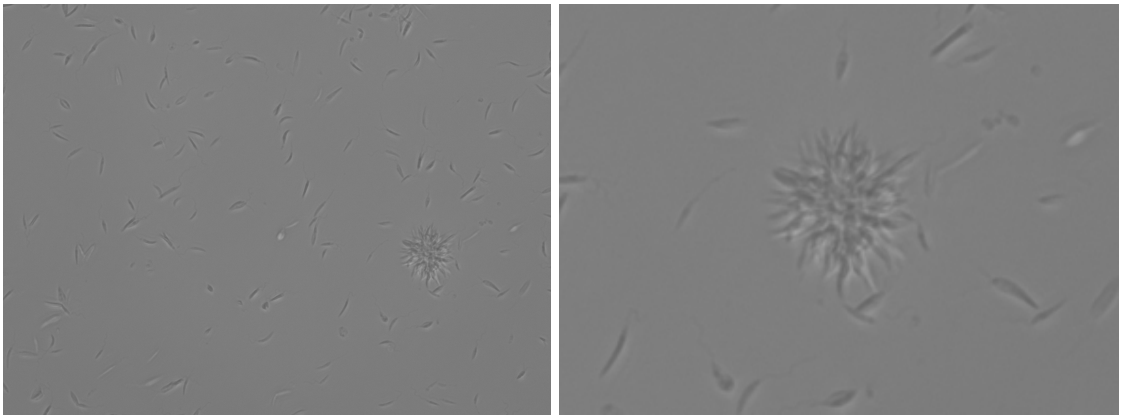


Figura 3.9: Ejemplo de agrupamiento

Agrupamiento jerárquico

El primer tipo de técnica de agrupamiento que se intentó implementar fue el agrupamiento jerárquico, un tipo de algoritmo de aprendizaje automático no supervisado que se

utiliza para agrupar puntos de datos no etiquetados con características similares.

Existen dos tipos de agrupación jerárquica: Aglomerativo y Divisivo. En el primero, los puntos de datos se agrupan utilizando un enfoque ascendente que comienza con puntos de datos individuales, mientras que en el segundo se sigue un enfoque descendente en el que todos los puntos de datos se tratan como un gran conjunto y el proceso de agrupación implica dividir el gran conjunto en varios conjuntos pequeños. Para el presente proyecto se ha intentado un enfoque aglomerativo y la implementación de la librería sklearn[11].

Sin embargo, mediante esta técnica no se ha conseguido generar los agrupamientos deseados. Seleccionar el número de agrupamientos como parámetro no era práctico, ya que el técnico tendría que hacer una observación previa de la imagen y detectar el mismo los agrupamientos en primer lugar. Pese a que esta operación podría ser aceptable (aunque incómoda), incluso habiendo probado todos los criterios de agrupamiento posibles ('ward', 'complete', 'average', 'single'), los grupos resultantes no eran satisfactorios. La figura a continuación muestra resultados de ejecución sobre la imagen de la figura 3.9.

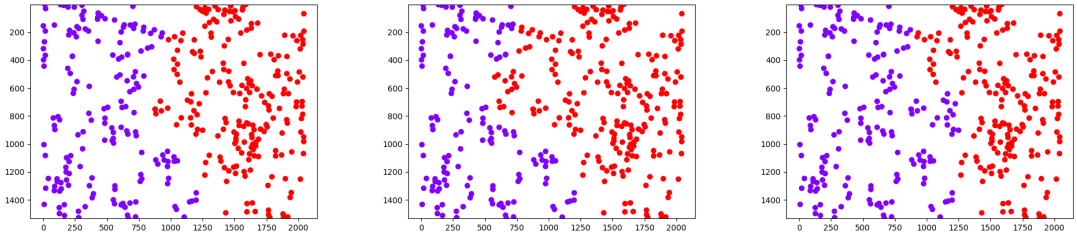


Figura 3.10: Resultados agrupamiento aglomerativo con criterio 'ward', 'complete' y 'average' y distancia euclídea.

También se ha probado la otra aproximación. En las ejecuciones anteriores se ajustaba el número de agrupaciones a 2, una para la agrupación existente y otra para el resto de puntos. En las figura que se muestran a continuación, se utiliza el criterio 'ward' y se da valores al parámetro `distance_threshold`[12] sin introducir el número de agrupamientos.

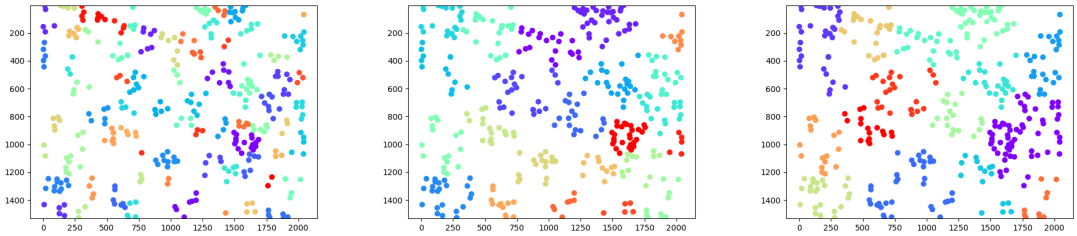


Figura 3.11: Resultados agrupamiento aglomerativo con distancia umbral de 200, 500, 700.

Pese a que detecta la agrupación (gráfico central, distancia umbral de 500) presente en la imagen original, agrupa en otros conjuntos al resto de parásitos por lo que habría que descartar manualmente los grupos que realmente se consideran como tal.

[11] Más información sobre la implementación de sklearn de agrupamiento aglomerativo y los criterios mencionados: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html

[12] El umbral de distancia de enlace por encima del cual, los agrupamientos no se fusionarán.

Agrupamiento basado en densidad

El agrupamiento espacial basado en densidad de o Density-based spatial clustering of applications with noise (DBSCAN) es un algoritmo de agrupamiento de datos basado en densidad (density-based clustering) porque encuentra un número de grupos (clusters) comenzando por una estimación de la distribución de densidad de los nodos correspondientes. Se ha hecho uso de la implementación de la librería sklearn[13].

Cuenta con dos parámetros principales: `eps` y `min_samples`

- `eps`: La distancia máxima entre dos puntos para que una se consideren vecinos. No se trata de un límite máximo en las distancias de los puntos dentro de un cluster. Este es el parámetro más importante de DBSCAN.
- `min_samples`: El número de puntos (o el peso total) en una vecindad para que un punto sea considerado como punto central. Esto incluye el propio punto.

Despues de una serie de ejecuciones alterando estos parámetros, se han conseguido los resultados esperados en las imágenes de las que se dispone. Tomando otra vez como ejemplo la figura 3.9:

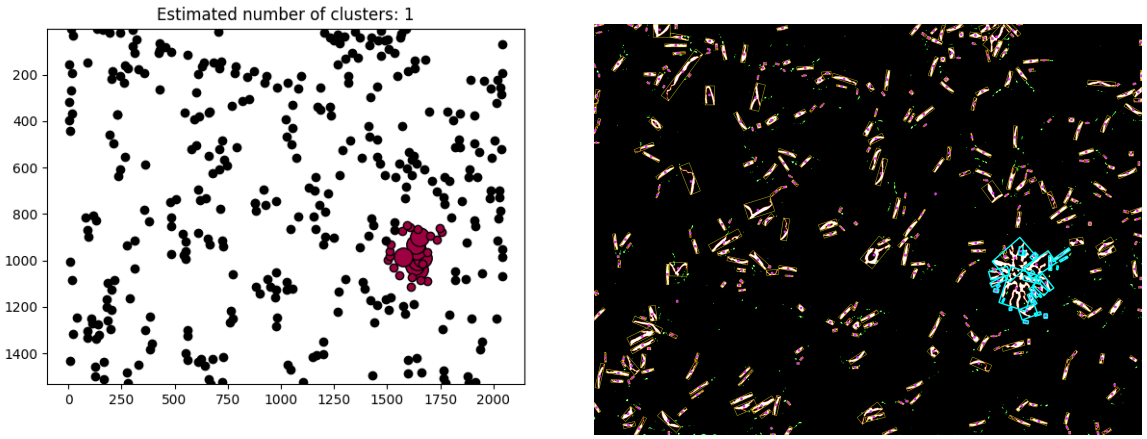


Figura 3.12: Resultados agrupamiento DBSCAN con eps = 105 y min_samples = 13.

3.2.8. Volcado de datos

Para habilitar una inferencia más profunda de la detección de parásitos en las imágenes de microscopio, es conveniente que los datos resultantes del análisis descrito sean volcados en un fichero, para que puedan ser estudiados o tratados por otra herramienta software.

El formato seleccionado ha sido el de notación de objetos de JavaScript, JSON. En este fichero se imprime información como los valores que se han utilizado para los parámetros de lanzamiento, el número de características (parásitos) encontrados, tamaño medio de los parásitos encontrados y, para cada parásito, su posición, descrita por las cuatro esquinas de su rectángulo delimitador, su área y su identificador.

Para ejemplificar la descripción anterior, a continuación se encuentra un fragmento de un fichero JSON de resultados:

[13]Más información sobre la implementación de sklearn de DBSCAN y los criterios mencionados: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

```
{
  "image_name": "8bit_Lamazo40x_3.366light_017.png",
  "gaussian_kernel": [
    11,
    11
  ],
  "block_size": 11,
  "constant": 2,
  "threshold_size": 40.0,
  "parasite_number": 149,
  "average_size(px)": 216.78187919463087,
  "size_variance(px)": 4664.136586452763,
  "parasites": [
    {
      "parasite_id": 0,
      "parasite_size": 314.0,
      "parasite_location": [
        [
          1210,
          1521
        ],
        [
          1213,
          1511
        ],
        [
          1279,
          1532
        ],
        [
          1275,
          1543
        ]
      ]
    }, ...
```


Fragmento de fichero JSON resultante de la ejecución del análisis

3.3. Aplicación web

La concepción de esta herramienta tiene como fin facilitar el trabajo de los investigadores del Instito de Enfermedades Tropicales y Salud Pública de Canarias (IUETSPC), por lo que no es sino razonable desarrollar una aplicación full stack con la que puedan interactuar directamente desde su navegador, sin necesidad de instalación ni de procedimientos o requisitos de ningún tipo por parte del usuario (tener una aplicación fullstack requiere que se cuente con una máquina en la que se ejecute el servidor y conexión al mismo como mínimo, sin embargo, para el usuario es la interfaz más cómoda y accesible).

3.3.1. Estructura de la aplicación web

En esta sección se lleva a cabo una descripción general de la estructura de la aplicación web desarrollada, apoyada en la siguiente figura:

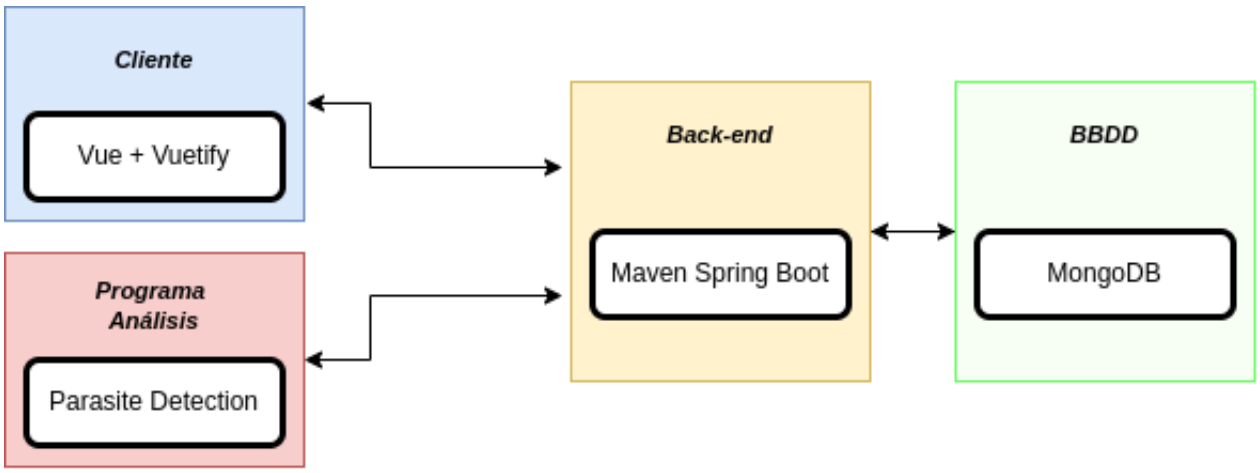


Figura 3.13: Esquema de elementos de la aplicación web

Se cuenta con una base de datos no relacional MongoDB[14] que permite la persistencia de los datos (en este caso las imágenes y los ficheros de datos resultantes) en la herramienta. El siguiente componente de la infraestructura es el back-end, con Maven y Spring Boot.

En el lado del cliente se utiliza el framework de desarrollo Vue.js[15], ya que ofrece gran versatilidad y cuenta con una curva de aprendizaje suave para usuarios no avezados en el desarrollo web. Además se ha hecho uso de Vuetify[16], una librería de componentes de Vue.js. El último elemento es el programa de análisis, que ya ha sido descrito en profundidad.

3.3.2. Base de datos

La base de datos se ejecuta mediante Docker[17], lo que habilita desplegar la base de datos sin la necesidad de tener MongoDB instalado en el equipo.

MongoDB es una base de datos NoSQL orientada a documentos que se utiliza para el almacenamiento de grandes volúmenes de datos. En lugar de utilizar tablas y filas como en las bases de datos relacionales tradicionales, MongoDB hace uso de colecciones y documentos. Los documentos consisten en pares clave-valor que son la unidad básica de datos en MongoDB. Las colecciones contienen conjuntos de documentos y funciones que son el equivalente a las tablas de las bases de datos relacionales. Estas características descritas hicieron que MongoDB fuera la elección finalmente tomada.

En la base datos se almacenarán las imágenes pre y post análisis junto a sus ficheros JSON resultantes, para ello cuentan con una clase repositorio que sirve como representación en la bbdd.

[14] MongoDB (del inglés humongous, "enorme") es un sistema de base de datos NoSQL, orientado a documentos y de código abierto.

[15] Para más información sobre Vue.js: https://vuejs.org/

[16] Para más información sobre Vuetify: https://vuetifyjs.com/en/

[17] Docker es una plataforma de software abierta diseñada para crear y ejecutar aplicaciones de un modo ágil y versátil. Entre sus ventajas está el hecho de que se puede trabajar con esta herramienta tanto en un ordenador como en la propia nube.

3.3.3. Back-end

En el lado del servidor se utiliza Maven para la construcción del proyecto, con el código fuente escrito en Java 1.8. Está acompañado de Spring Boot que proporciona una buena plataforma para desarrollar una aplicación Spring autónoma y de nivel de producción que se puede ejecutar directamente, ya que se puede empezar con configuraciones mínimas sin necesidad de una configuración completa de Spring. La API implementada mantiene una arquitectura REST[18]. En el servidor se definen tres tipos de elementos: analizadores, imágenes y ficheros JSON.

Para acceder a estos tres elementos cada uno cuenta con un endpoint[19] particular:

- Para las imágenes: IP:PORT/images
- Para el análisis: IP:PORT/analyzer
- Para los JSON: IP:PORT/json

La figura 4.11 proporciona una visión simplificada de las peticiones a las que responde cada endpoint implementado[20].

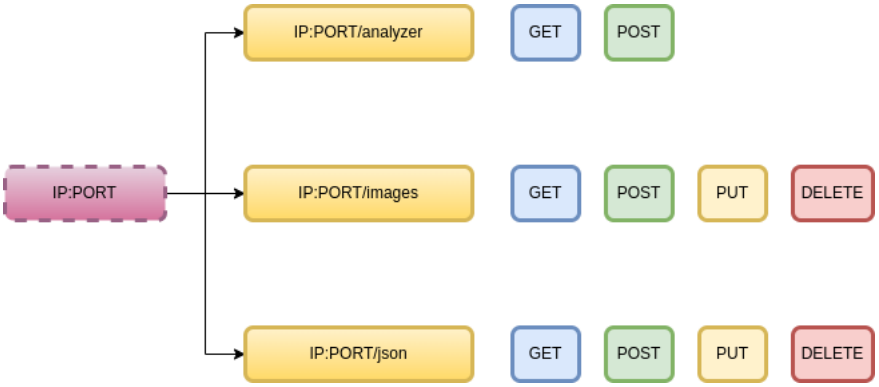


Figura 3.14: Esquema simplificado de la API del servidor

Analizador

El analizador es representado mediante una clase simple, definido por tres cadenas: su identificador, la ruta de la imagen a ser analizada y los parámetros de lanzamiento del algoritmo. Además, se define su comportamiento para las peticiones `GET, POST`.

- La petición `GET IP:PORT/analyzer` devuelve una descripción sobre como ejecutar el análisis mediante la API: que necesita para funcionar y que devuelve.

[18]Cuando el cliente envía una solicitud a través de una API de RESTful, esta transfiere una representación del estado del recurso requerido a quien lo haya solicitado o al extremo. La información se entrega por medio de HTTP en uno de estos formatos: JSON (JavaScript Object Notation), HTML, XLT, Python, PHP o texto sin formato. JSON es el lenguaje de programación más popular, ya que tanto las máquinas como las personas lo pueden comprender y no depende de ningún lenguaje, a pesar de que su nombre indique lo contrario.

[19]Un endpoint de la API es un punto en el que una API -el código que permite que dos programas software se comuniquen entre sí- se conecta con el programa software. Las APIs funcionan enviando solicitudes de información desde una aplicación o servidor web y recibiendo una respuesta.

[20]Se utilizará la nomenclatura IP:PORT para generalizar las descripciones, ya que la dirección y el puerto con el que comunicarse dependerán de la máquina en la que se despliegue la herramienta.

- La petición `POST -F exec_params = params IP:PORT/analyzer/id` ejecuta el análisis con los parámetros que se encuentren en params sobre la imagen con el identificador id. Los parámetros aceptados son los siguientes:
    - -h, –help: Muestra un mensaje de ayuda y sale del programa.
    - -i, –input: Ruta a la imagen que se desea procesar. Obligatorio.
    - -o, –output: Ruta en la que guardar el fichero JSON resultante. Obligatorio.
    - -a, –algorithm: Nombre del algoritmo de umbralización seleccionado, los disponibles ahora son: [adaptativo, simple, doble]. Obligatorio.
    - -th, –threshold-value: Valor umbral para el procedimiento de umbralizado. (por defecto 100).
    - -th2, –threshold-value-2: Segundo valor umbral para el procedimiento de umbralizado doble. (por defecto 120).
    - -gk, –gaussian_kernel: Tamaño del núcleo gaussiano para el método de umbralizado. La anchura y la altura pueden ser diferentes, pero ambas deben ser positivas e impares. O bien, pueden ser ceros y entonces se calculan a partir de sigma. (por defecto: [11, 11]).
    - -bs, –block-size: Tamaño de la vecindad de píxeles que se utiliza para calcular un valor de umbral para el píxel: 3, 5, 7, etc. (por defecto: 11)
    - -c, –constant: Constante que se resta a la media o a la media ponderada (ver los detalles más abajo). Normalmente, es positivo, pero también puede ser cero o negativo. (por defecto: 2).
    - -ms, –minimum-size: Valor mínimo de tamaño (área) para que los contornos detectados se consideran parásitos. Con el conjunto de imágenes actual, 40.0 parece ser un valor adecuado. (por defecto: 40.0).
    - -mnd, –max-neighbourhood-distance: Valor máximo de distancia entre un centro de cluter potencial y sus vecinos. Con el conjunto de imágenes actual, 105.0 parece ser un valor muy adecuado. (por defecto: 105.0).
    - -mcs, –min-cluster-size: Valor mínimo de tamaño de un cluster para ser considerado como tal. Con el conjunto de imágenes actual, 13 parece ser un valor adecuado. (por defecto: 13).
    - –version: Muestra la versión del programa y sale del mismo.

Imagen

Las imágenes se alamecenan con un identificador individual, un nombre y un tipo de dato `binary` que contiene los datos del fichero binario de la imagen.

- La petición `GET localhost:3000/images` devuelve una lista de los identificadores y nombres de todas las imágenes que se encuentran actualmente en el almacenamiento de la base de datos.
- La petición `GET localhost:3000/images/id` devuelve la imagen con identificador `id`.

- La petición `POST -F name = img_name -F image = img_route localhost:3000/images` sube el fichero contenedor de la imagen encontrada en la ruta `img_route` con el nombre `img_name` a la base de datos.
- La petición `PUT -F name = img_name -F image = img_route localhost:3000/images/id` sustituye la imagen con identificador `id` con la imagen encontrada en la ruta `img_route` con el nombre `img_name`.
- La petición `DELETE localhost:3000/images/id` elimina la imagen con identificador `id` del repositorio.

JSON

Los ficheros de resultados en formato JSON se almacenan con tres cadenas: un identificador individual, un nombre y su contenido.

- La petición `GET localhost:3000/json` devuelve una lista de los identificadores y nombres de todos los ficheros JSON que se encuentran actualmente en el almacenamiento de la base de datos.
- La petición `GET localhost:3000/json/id` devuelve el fichero JSON con identificador `id`.
- La petición `POST -F name = json_name -F json = json_route localhost:3000/json` sube el fichero JSON encontrado en la ruta `json_route` con el nombre `json_name`.
- La petición `PUT -F name = json_name -F json = json_route localhost:3000/json/id` sustituye el JSON con identificador `id` con el fichero JSON encontrado en la ruta `json_route` con el nombre `json_name`.
- La petición `DELETE localhost:3000/json/id` elimina el fichero JSON con identificador `id` del repositorio.

3.3.4. Cliente

El último componente de la herramienta a ser descrito es el cliente, encargado de ofrecer al usuario un medio de comunicación con la API descrita en el punto anterior. Ha sido desarrollada con el objetivo de ser lo más amigable, sencilla y directa posible, permitiendo aún así una muy buena velocidad en el flujo de trabajo.

Se ha hecho uso del framework de desarrollo de JavaScript para la creación de interfaces de usuario. Se basa en el HTML, CSS y JavaScript estándar, y proporciona un modelo de programación declarativo y basado en componentes que ayuda a desarrollar eficazmente interfaces de usuario, ya sean sencillas o complejas. Encima de Vue.js se utiliza Vuetify, una librería escrita encima de Vue.js que aporta gran variedad de componentes hechos a mano, lo que ha ahorrado valioso tiempo de desarrollo.

Subir imágenes

Para poder ejecutar un algoritmo de análisis como el desarrollado sobre una imagen, el primer paso es dar acceso a dicha imagen a la entidad que ejecuta el algoritmo, en este

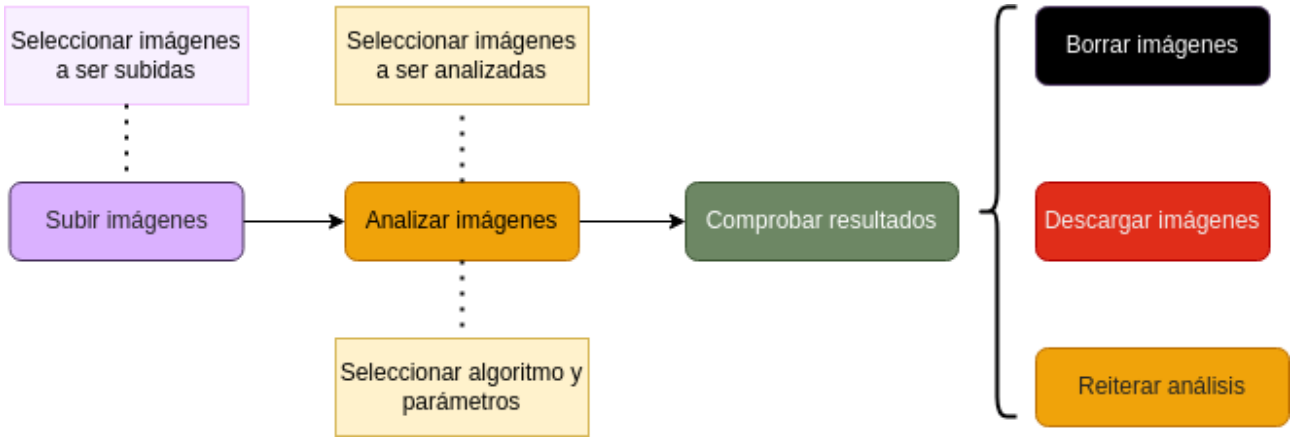


Figura 3.15: Esquema simplificado del flujo de trabajo

caso, el servidor descrito. Para ello, se hace uso del componente de selección de ficheros junto al botón "Subir imágenes".

Desde el cliente se lanza una petición HTTP de tipo POST con el nombre de la imagen y su contenido por cada imagen que se haya subido. Una vez se el servidor a recibido la petición y actuado, almacenando dichos ficheros en la base de datos, devuelve un mensaje de OK si todo ha salido bien. Por último se ejecuta una petición GET que se trae la lista actualizada de imágenes en la base de datos para que el cliente lo refleje así en la tabla de imágenes base.

Figura 3.16: Elementos que permiten la subida de imágenes

Analizar imágenes

Una vez las imágenes han sido subidas, una tabla con una fila por cada una de ellas se muestra en la herramienta:

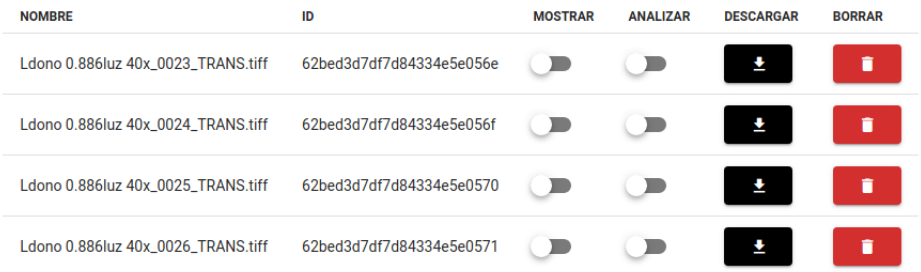


Figura 3.17: Tabla de imágenes subidas

Además de la posibilidad de descargar o borrar las imágenes de forma individual desde la tabla, existe la opción de activar los interruptores de ’Mostrar’ y ’Analizar’. El primero ’Mostrar’, muestra la imagen en la parte inferior y permite hacerle click para ser visualizada en grande además de aplicarle transformaciones simples como zoom o rotaciones. El segundo, ’Analizar’ coloca la imágen en la cola de análisis.

El siguiente paso es la selección del algoritmo de umbralizado (simple, doble o adaptativo) y los parámetros de lanzamiento necesarios para el correcto funcionamiento del programa de análisis. Dichos parámetros cambian en función del algoritmo seleccionado.

Figura 3.18: Parámetros de lanzamiento para algoritmo adaptativo

Resultados del análisis

Las imágenes resultantes del análisis se encontrarán reflejadas en la tabla de imágenes resultantes. En dicha tabla aparecen todos los resultados de las ejecuciones de la sesión actual. La tabla de resultados carece de la columna .Analizarza que los resultados del análisis no pueden ser analizados de vuelta. Tanto las imágenes resultantes como sus ficheros de volcado de datos se nombran de forma iterativa, es decir, después de ejecutar el análisis por primera vez sobre, por ejemplo, `imagen.png` se generaran `imagen_result0.png` y `imagen_data0.json`. El número irá aumentando a medida que se vayan efectuando ejecuciones del análisis. Cuenta con una nueva columna Resultados", que ofrece un botón de descarga azul, para descargar el fichero de resultados generado en el análisis de la imagen. En la siguiente figura se observa un ejemplo de tabla de resultados:

Figura 3.19: Tabla de resultados

Guía de uso

Para reducir la dificultad inicial de la curva de aprendizaje de la herramienta, los investigadores pueden acceder a la guía de usuario desde el menú lateral de la aplicación. Este menú se puede abrir haciendo click en el icono que muestra tres líneas blancas apiladas en la esquina superior izquierda.

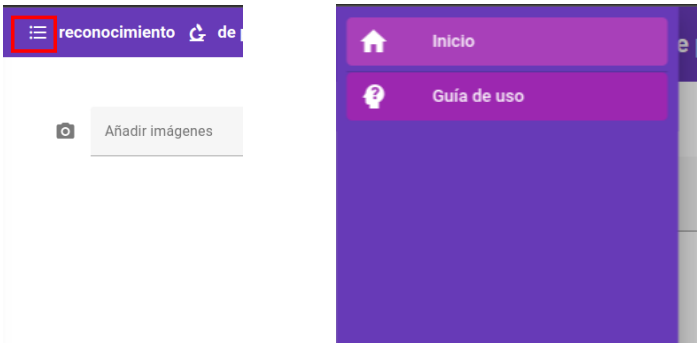


Figura 3.20: Acceso a página de ayuda

Capítulo 4

Resultados obtenidos

El desarrollo de este trabajo ha producido en última instancia imágenes de microscopio en la que los parásitos presentes se encuentran *marcados* mediante un rectángulo delimitador. A continuación se adjuntan una colección de dichos resultados, para que su precisión quede demostrada.

Las imágenes recabadas han sido tomadas de tres muestras distintas. Por las diferencias tanto en las muestras como en las condiciones en las que se tomaron para cada tipo de imagen funcionan mejor determinados algoritmos de umbralizado y parámetros. A continuación se describen cuales son.

4.1. Muestra I

La primera carpeta de imágenes que se recibió para comenzar a trabajar comparte condiciones de aumento y luminosidad entre todos sus elementos (luminosidad 3.366 y aumento 40x).

El algoritmo de umbralizado con el que mejores resultados se han conseguido es el adaptativo, ya que cuenta con regiones en las que la iluminación varía levemente. Han sido analizadas con los siguientes parámetros: "gaussian_kernel": [11,11], "block_size": 11, çonstant": 2, "minimum_size": 50.0, "max_neighbour_distance(px)": 105.0, "min_cluster_-size": 13.

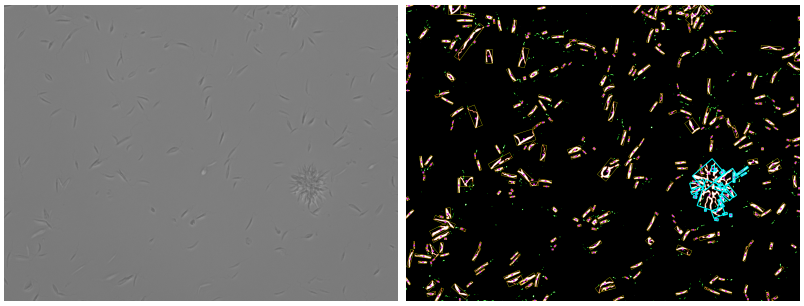


Figura 4.1: Imagen 8bit_Lamazo40x_3.366light_007.png, 346 parásitos detectados y 1 agrupamiento.

4.2. Muestra II

En la segunda muestra recibida, pese a que todos los elementos comparten el aumento 40x, existen imágenes con dos condiciones de iluminación distinta por lo que serán

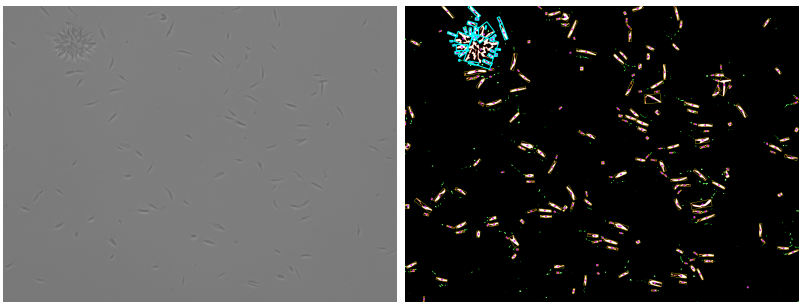


Figura 4.2: Imagen 8bit_Lamazo40x_3.366light_010.png, 224 parásitos detectados y 1 agrupamiento.

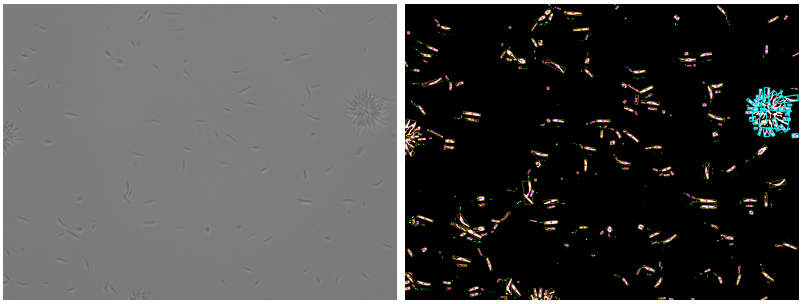


Figura 4.3: Imagen 8bit_Lamazo40x_3.366light_011.png, 241 parásitos detectados y 1 agrupamiento.

mostrados los resultados en subsecciones distintas.

4.2.1. Iluminación 0.886

Para este tipo de imágenes el algoritmo de umbralizado simple demuestra buenos resultados. Los parámetros utilizados han sido los siguientes: "threshold_value": 78.0, "minimum_size": 60.0, "max_neighbour_distance(px)": 105.0, "min_cluster_size": 13.

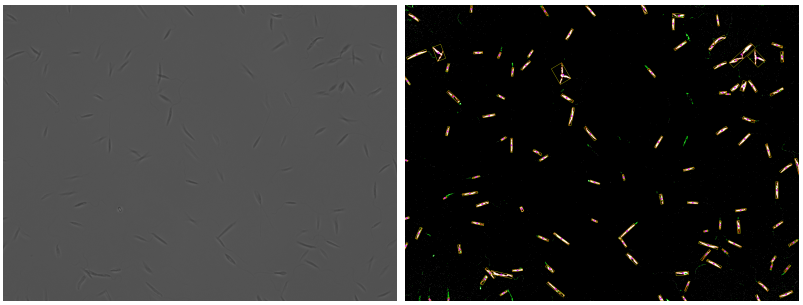


Figura 4.4: Imagen Ldono 0.886luz 40x_0037_TRANS.tiff, 106 parásitos detectados y 0 agrupamiento.

4.2.2. Iluminación 1.596

Este subconjunto de imágenes cuenta con la peculiaridad de que pueden aparecer parásitos en dos tonalidades de gris en la misma imagen. Para su correcto análisis, el algoritmo de umbralizado doble es el que mejores resultados demuestra. Sin embargo, hay que ajustar muy bien los valores umbral para conseguir un producto óptimo.

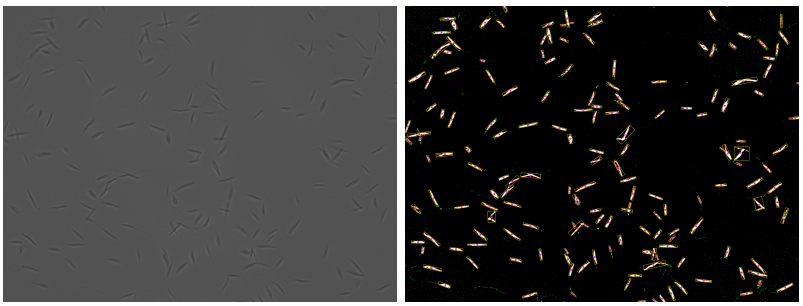


Figura 4.5: Imagen Ldono 0.886luz 40x_0041_TRANS.tiff, 170 parásitos detectados y 0 agrupamiento.

Los parámetros utilizados han sido los siguientes: "threshold_value": 160.0, "threshold_value_2": 120.0, "minimum_size": 60.0, "max_neighbour_distance(px)": 105.0, "min_cluster_size": 13.

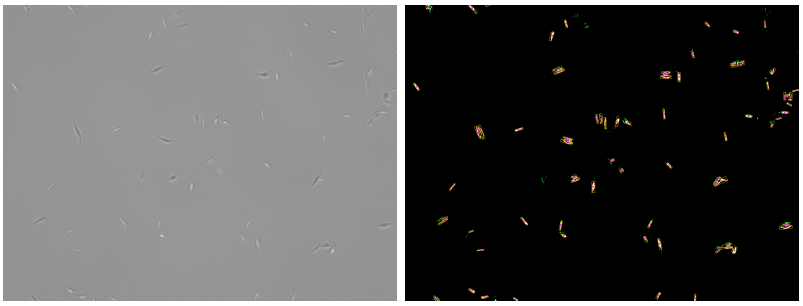


Figura 4.6: Imagen Ldono 1.596luz 40X_0001_TRANS.tiff, 62 parásitos detectados y 0 agrupamientos.

Además y a modo de comparación se adjuntan los resultados de la ejecución del análisis sobre la misma imagen pero con el algoritmo de umbralizado adaptativo y los siguientes parámetros: "gaussian_kernel": [21,21], "block_size": 11, çonstant": 3, "minimum_size": 60.0, "max_neighbour_distance(px)": 105.0, "min_cluster_size": 13.

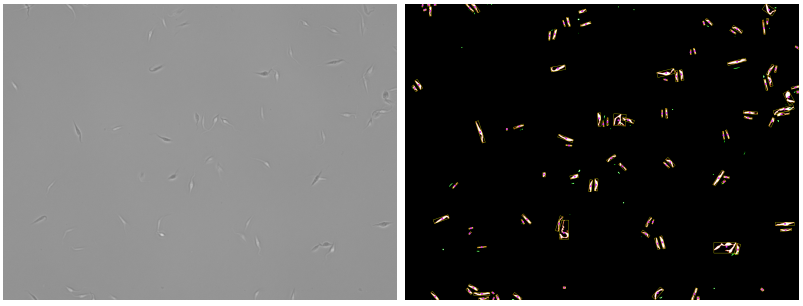


Figura 4.7: Imagen Ldono 1.596luz 40X_0001_TRANS.tiff, 70 parásitos detectados y 0 agrupamientos.

4.3. Muestra III

La tercera carpeta de imágenes comparte condiciones de aumento y luminosidad entre todos sus elementos (luminosidad 0.713 y aumento 40x). Este subconjunto de imágenes también cuenta con la peculiaridad de que pueden aparecer parásitos en dos tonalidades

de gris en la misma imagen. Para su correcto análisis, el algoritmo de umbralizado doble es el que mejores resultados demuestra. Sin embargo, hay que ajustar muy bien los valores umbral para conseguir un producto óptimo.

Los parámetros utilizados han sido los siguientes: "threshold_value": 72.0, "threshold_value_2": 60.0, "minimum_size": 60.0, "max_neighbour_distance(px)": 105.0, "min_cluster_size": 13.

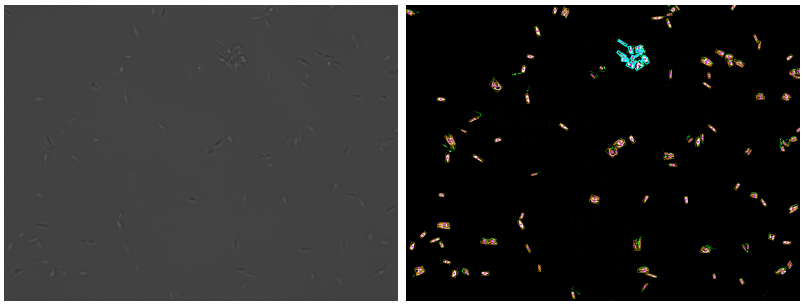


Figura 4.8: Imagen Trypa 0.713luz 40X_0005_TRANS.tiff, 147 parásitos detectados y 1 agrupamiento.

Capítulo 5

Presupuesto

En la Tabla 5.1 se detallan las tareas realizadas y el tiempo invertido en cada una en horas, además de los recursos necesarios para llevar a cabo este proyecto, todo ello acompañado de su precio correspondiente.

El salario medio de un Ingeniero Informático es de unos 36.500,00 euros brutos anuales, alrededor de 3.000,00 euros brutos mensuales[1]. Según esto, y teniendo en cuenta que un mes tiene 20 días laborales y suponiendo una jornada de 8 horas, el coste de cada hora de trabajo es de 19,00 euros brutos. Las horas trabajadas estipuladas por la guía docente de la asignatura del Trabajo Fin de Grado, ascienden a un total de 300 horas.

Todo esto, aplicado a las horas de trabajo dedicadas a cada fase del proyecto, y sumado a los recursos necesarios para su realización, ofrece el presupuesto total en bruto del proyecto.

Tabla 5.1: Presupuesto del proyecto

Descripción	**Cantidad(h)**	**Precio (€)**	**Total (€)**
Salario por hora Ingeniero Informático	300	19,00	5700,00

[1] Salario Ingeniero Informático en España: https://www.jobted.es/salario/ingeniero-informatico

Capítulo 6

Conclusiones y líneas futuras

El proyecto de reconocimiento de parásitos en imágenes de microscopio consiste en un problema real al que se enfrentan los investigadores del Instituto Universitario de Enfermedades Tropicales y Salud Pública de Canarias, por el presenta trabajo, representa o al menos trata de representar una solución real a un problema real.

Partiendo de esta premisa, considero que los resultados obtenidos son satisfactorios: las imágenes de laboratorios son analizadas con éxito por el software, ya que los parásitos son detectados y medidos, las imágenes son marcadas para que las características relevantes puedan ser discernidas a primera vista y se genera un documento con datos relevantes al análisis. Además se ha llevado a cabo una simple aplicación web para que los investigadores puedan hacer uso de la herramienta con mayor facilidad, sin necesidad de instalaciones (por parte del usuario) ni de trabajar con una terminal.

Sin embargo, la herramienta aún esta lejos de ser perfecta; si en algún punto se disponen de las suficientes[1] imágenes (y sobre todo imágenes etiquetadas), se podrían implementar otros mecanismos de detección, mediante técnicas modernas de aprendizaje automático deep learning o redes convolucionales, mecanismos que podrían ser más flexibles y polivalentes que los implementados. Utilizando aproximaciones como las mencionadas se podría determinar el estado de los parásitos, estudiando cuanto difiere su morfología de la de un parásito sano.

Existe margen de desarrollo para la aplicación web. Una pantalla que muestre un análisis interactivo de la imagen resultante, en la que se puedan seleccionar con el ratón los parásitos de forma individual, acompañado de información relevante al análisis (como número de parásitos y su tamaño, por ejemplo). En última instancia sería de gran utilidad el desarrollo de un componente que permita la edición de la imagen resultante de manera interactiva, para que el técnico pueda hacer retoques en la detección final de los parásitos.

Los ficheros de datos resultantes también generan muchos datos que si fueran procesados, podrían resultar útiles para entrenar un modelo que después pudiera hacer predicciones acerca de las características de otras imágenes nuevas, por lo que por ahí también se podría seguir evolucionando el proyecto.

[1] Se cuenta con un total de 75 imágenes, ninguna de ellas profesionalmente etiquetada. Según que aproximación se desee implementar el número mínimo de imágenes etiquetadas necesarias podría aproximarse a las 200.

Capítulo 7

Summary and Conclusions

The project of parasite recognition in microscope images consists of a real problem faced by the researchers of the Instituto Universitario de Enfermedades Tropicales y Salud Pública de Canarias, by the present work, represents or at least tries to represent a real solution to a real problem.

Based on this premise, I consider that the results obtained are satisfactory: the laboratory images are successfully analyzed by the software, since the parasites are detected and measured, the images are marked so that the relevant features can be discerned at first glance and a document with data relevant to the analysis is generated. In addition, a simple web application has been developed so that researchers can make use of the tool more easily, without the need for installation (by the user) and without the need to work with a terminal.

However, the tool is still far from perfect; if at any point there are enough images[1] (and especially labeled images) available, it will be possible to use the tool in a very short time. images (and especially labeled images), other detection mechanisms could be implemented, using modern deep learning machine learning techniques or convolutional networks, mechanisms that could be more flexible and multipurpose than the ones implemented. Using approaches such as those mentioned above, the state of the parasites could be determined by studying how much their morphology differs from that of a healthy parasite.

There is scope for development of the web application. A screen displaying an interactive analysis of the resulting image, in which individual parasites can be selected with the mouse, accompanied by information relevant to the analysis (such as number of parasites and their size, for example). Ultimately, it would be very useful to develop a component that allows interactive editing of the resulting image, so that the technician can make adjustments in the final detection of the parasites.

The resulting data files also generate a lot of data that, if processed, could be useful for training a model that could then make predictions about the characteristics of other new images, so the project could also continue to evolve.

[1] There are a total of 75 images, none of which are professionally labeled. Depending on the approach to be implemented, the minimum number of labeled images required could be closer to 200.

Bibliografía

[1] Hao Pan, Zheng Xu, and Junzhou Huang. An effective approach for robust lung cancer cell detection. In *International Workshop on Patch-based Techniques in Medical Imaging*, pages 87–94. Springer, 2015.

[2] Emanuela Handman. Cell Biology of Leishmania. *Advances in Parasitology*, pages 1–39, 1999.

[3] PAÑogueira. Determining leishmania infection levels by automatic analysis of microscopy images. *arXiv preprint arXiv:1311.2621*, 2013.

[4] Kaustubh Chakradeo, Michael Delves, and Sofya Titarenko. Malaria parasite detection using deep learning methods. *International Journal of Computer and Information Engineering*, 15(2):175–182, 2021.

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[6] Mojtaba Zare, Hossein Akbarialiabad, Hossein Parsaei, Qasem Asgari, Ali Alinejad, Mohammad Saleh Bahreini, Seyed Hossein Hosseini, Mohsen Ghofrani-Jahromi, Reza Shahriarirad, Yalda Amirmoezzi, Sepehr Shahriarirad, Ali Zeighami, and Gholamreza Abdollahifard. A machine learning-based system for detecting leishmaniasis in microscopic images. *BMC Infectious Diseases*, 22(1), 2022.

[7] Marc Górriz, Albert Aparicio, Berta Raventós, Verónica Vilaplana, Elisa Sayrol, and Daniel López-Codina. Leishmaniasis Parasite Segmentation and Classification Using Deep Learning. *Articulated Motion and Deformable Objects*, pages 53–62, 2018.