



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Cuantificación de la carga viral de SARS-CoV-2 mediante el procesamiento de imágenes de microgotas fluorescentes

TESIS

Que para obtener el título de

Ingeniero Eléctrico Electrónico

P R E S E N T A N

Zuleima Hidalgo Jaime

Isaí Sánchez Jiménez

DIRECTORA DE TESIS

Dra. Flor Lizeth Torres Ortiz



Ciudad Universitaria, Cd. Mx., 2025



**PROTESTA UNIVERSITARIA DE INTEGRIDAD Y
HONESTIDAD ACADÉMICA Y PROFESIONAL**
(Titulación con trabajo escrito)



De conformidad con lo dispuesto en los artículos 87, fracción V, del Estatuto General, 68, primer párrafo, del Reglamento General de Estudios Universitarios y 26, fracción I, y 35 del Reglamento General de Exámenes, me comprometo en todo tiempo a honrar a la institución y a cumplir con los principios establecidos en el Código de Ética de la Universidad Nacional Autónoma de México, especialmente con los de integridad y honestidad académica.

De acuerdo con lo anterior, manifiesto que el trabajo escrito titulado CUANTIFICACION DE LA CARGA VIRAL DE SARS-COV-2 MEDIANTE EL PROCESAMIENTO DE IMAGENES DE MICROGOTAS FLUORESCENTES, que presenté para obtener el título de INGENIERO ELÉCTRICO ELECTRÓNICO es original, de mi autoría y lo realicé con el rigor metodológico exigido por mi Entidad Académica, citando las fuentes de ideas, textos, imágenes, gráficos u otro tipo de obras empleadas para su desarrollo.

En consecuencia, acepto que la falta de cumplimiento de las disposiciones reglamentarias y normativas de la Universidad, en particular las ya referidas en el Código de Ética, llevará a la nulidad de los actos de carácter académico administrativo del proceso de titulación.

ZULEIMA HIDALGO JAIME
Número de cuenta: 314016395



**PROTESTA UNIVERSITARIA DE INTEGRIDAD Y
HONESTIDAD ACADÉMICA Y PROFESIONAL**
(Titulación con trabajo escrito)



De conformidad con lo dispuesto en los artículos 87, fracción V, del Estatuto General, 68, primer párrafo, del Reglamento General de Estudios Universitarios y 26, fracción I, y 35 del Reglamento General de Exámenes, me comprometo en todo tiempo a honrar a la institución y a cumplir con los principios establecidos en el Código de Ética de la Universidad Nacional Autónoma de México, especialmente con los de integridad y honestidad académica.

De acuerdo con lo anterior, manifiesto que el trabajo escrito titulado CUANTIFICACION DE LA CARGA VIRAL DE SARS-COV-2 MEDIANTE EL PROCESAMIENTO DE IMAGENES DE MICROGOTAS FLUORESCENTES, que presenté para obtener el título de INGENIERO ELÉCTRICO ELECTRÓNICO es original, de mi autoría y lo realicé con el rigor metodológico exigido por mi Entidad Académica, citando las fuentes de ideas, textos, imágenes, gráficos u otro tipo de obras empleadas para su desarrollo.

En consecuencia, acepto que la falta de cumplimiento de las disposiciones reglamentarias y normativas de la Universidad, en particular las ya referidas en el Código de Ética, llevará a la nulidad de los actos de carácter académico administrativo del proceso de titulación.

ISAI SANCHEZ JIMENEZ
Número de cuenta: 314295293

Agradecimientos

Zuleima Hidalgo Jaime:

Antes que nada, quiero agradecer con todo mi amor y cariño a mis padres, Yazmín Jaime García y Guillermo Hidalgo Ruiz, quienes fueron la base de mi esfuerzo, mis guías, mi motivación y mi apoyo incondicional, entre muchas otras cosas que me ayudaron a culminar esta etapa de mi vida. Gracias por todo su amor y paciencia. Es en gran medida por ustedes, por sus consejos y enseñanzas, que soy quien soy hoy en día, y estoy orgullosa de la persona en la que me he convertido. Nunca me cansaré de repetir lo afortunada que soy de que me acompañen y me vean crecer en todos los ámbitos, y estaré muy contenta de poder regresarles con cariño un poco de todo lo que me han brindado. Los amo con todo mi corazón.

También agradezco infinitamente al Ing. Ricardo Estrada Salgado quien fue parte fundamental para no rendirme en este trayecto y que me dio muchas herramientas y apoyo para no desistir de la carrera, es algo que atesoraré siempre. Asimismo, quiero agradecer a mi mejor amiga, Mariel Murillo Rodríguez, por todas las risas, pláticas, viajes, salidas y por supuesto, el apoyo que me brindó a lo largo de toda la carrera, especialmente en los momentos en que el estrés me sobrepasaba y, en particular, por estar ahí cuando más necesité una mano para seguir.

Agradezco a todas las personas que conocí durante todo este tiempo y que me ayudaron y acompañaron en las noches de desvelo haciendo tareas, trabajos y proyectos. En especial agradezco al Mat. Osvaldo Emmanuel Palma Cabrera por motivarme y hacer del último tramo de la carrera un lapso muy bonito y divertido, también por extenderme su sincero apoyo en lo que yo necesitase.

Quiero agradecer a mi directora de tesis la Dra. Flor Lizeth Torres Ortiz por todo su apoyo, paciencia, observaciones y estar siempre al pendiente de cualquier duda al igual que la Dra. Laura A. Oropeza Ramos por toda la comprensión y apoyo que nos brindó de manera muy amable y con su buen humor que la caracteriza.

Finalmente agradezco a la Universidad Nacional Autónoma de México y a la Facultad de Ingeniería por darme la oportunidad de desarrollarme profesionalmente y otorgarme todas las herramientas para convertirme en ingeniera.

Investigación realizada gracias al Programa UNAM-PAPIIT IT100922, así como la beca otorgada a Zuleima Hidalgo Jaime por su participación en el mismo.

Contenido

Índice de Figuras.....	3
Resumen	5
1 Introducción.....	6
1.1 Antecedentes	6
1.2 Objetivos	8
1.3 Justificación.....	8
1.4 Descripción general de la estructura de la tesis.....	9
2 Fundamentos tecnológicos.....	10
2.1 Microfluídica, creación de microgotas y el método ddLamp	10
2.2 Lenguaje de programación Python	12
2.2.1 OpenCV y NumPy	12
2.2.2 Numpy	13
2.2.3 TKinter	13
3 Metodología.....	14
3.1 Fase 1: Procesamiento de las imágenes	14
3.1.1 Transformar la imagen.....	16
3.1.2 Importar	16
3.1.3 Ajuste de brillo y contraste	20
3.1.4 Suavizar (Filtro Gaussiano).....	22
3.1.5 Detección de Contornos (Filtro de Canny).....	24
3.1.6 Erosión y Dilatación	27
3.2 Fase 2: Conteo de las microgotas	30
3.2.1 La transformada de Hough	30
3.2.2 K-Means	35
3.2.3 Control Negativo	40
4 Aplicación en Python	40
4.1 Inicio	41
4.2 Interfaz Gráfica	42
4.3 Detección de microgotas	54
5 Resultados.....	71
5.1 Casos de estudio.....	71
5.2 Análisis de resultados.....	71
5.2.1 Muestras de pacientes con baja carga viral	72

5.2.2	Muestras de pacientes con carga viral media	74
5.2.3	Muestras de pacientes con carga viral alta	77
6	Conclusiones.....	80
	Anexos	81
A.1	Visual Studio Code.....	81
A.1.1	Preparación del entorno de trabajo de VSC para la creación de la interfaz.....	83
A.2	Guía de funcionamiento de la interfaz de usuario.....	84
	Referencias	84

Índice de Figuras

Figura 1.1.1	Procesos biológicos en microgotas.	6
Figura 1.1.2	Imágenes de análisis de microgotas obtenidas con microscopía.....	7
Figura 2.1	Microcanales en un microdispositivo de PDMS [9].	10
Figura 2.2	Ejemplo de microcanales y generación de microgotas. Extraído de [9].	11
Figura 2.3	Ejemplo de aplicación desarrollada con Tkinter en Python.....	14
Figura 3.1	Diagrama de flujo para el procesamiento de las imágenes de gotas.....	15
Figura 3.2	Captura de una imagen con un dispositivo pasivo digital. Modificado de [15].....	16
Figura 3.3	Representación matricial de una imagen en escala de grises	19
Figura 3.4	Ajuste de brillo y contraste de una imagen.	21
Figura 3.5	Ejemplos de Kernels Gaussianos.....	22
Figura 3.6	Representación de matrices usando filtro Gaussiano. Modificado de [18].....	23
Figura 3.7	Imagen suavizada después de utilizar el filtro Gaussiano. Extraído de [19].	24
Figura 3.8	Ejemplo gráfico de supresión de no máximos. Modificado de [20].....	25
Figura 3.9	Etapa final del filtro Canny: umbralización. Modificado de [20]	26
Figura 3.10	. Ejemplo de utilizar filtro Canny en una imagen.	27
Figura 3.11	Ejemplo de matriz 8x8 y kernel 3x3.	28
Figura 3.12	Erosión de una matriz 8x8 con un kernel 3x3.	28
Figura 3.13	Ejemplo de erosión.....	29
Figura 3.14	Ejemplo de dilatación de una imagen.....	30
Figura 3.15	Representación gráfica de la transformada de Hough	31
Figura 3.16	Representación gráfica de la transformada de Hough para circunferencias	32
Figura 3.17	Representación gráfica de la transformada de Hough	33
Figura 3.18	Localización de centroides al azar, donde se supone una k = 3	36
Figura 3.19	Puntos cercanos al centroide Azul son calculados.	36
Figura 3.20	Cálculo de puntos cercanos al centroide Azul.....	37
Figura 3.21	Para los puntos cercanos a los centroides morado	37
Figura 3.22	En una iteración n	38
Figura 3.23	Los colores se actualizan y el centroide se mueve.	38
Figura 3.24	Cuando la delta de posición es mínima, consideramos terminado el algoritmo.....	39
Figura 3.25	Con k = 3 encontramos 3 grupos.	39
Figura 3.26.	Con k = 4 obtenemos grupos que a primera vista no son evidentes.....	40

Figura 4.1 Diagrama de funciones que componen la interfaz gráfica.	42
Figura 4.2 Diagrama de funciones que componen la detección de microgotas.	55
Figura 5.1 Tabla con resultados del análisis de imágenes con baja carga viral	72
Figura 5.2 Análisis de muestras con carga viral baja	73
Figura 5.3 Resultado de imagen No. 1 con baja carga viral usando K-means.	74
Figura 5.4 Imagen, histograma y gráfica de dispersión de muestra con carga viral	74
Figura 5.5 Tabla con resultados del análisis de imágenes con carga viral media.	75
Figura 5.6 Resultado de imagen 1 con carga viral media usando k-means	75
Figura 5.7 Imagen, histograma y gráfica de dispersión	75
Figura 5.8 Análisis de muestras con carga viral media.	76
Figura 5.9 Tabla con resultados del análisis de imágenes con carga viral alta	77
Figura 5.100 Resultado de imagen 1 con carga viral alta usando k-means	77
Figura 5.111 Imagen, histograma y gráfica de dispersión de muestra con carga viral	77
Figura 5.122 Análisis de muestras con carga viral alta	79
Figura A.0.1 Pantalla de inicio de Visual Studio Code	82
Figura A.0.2 Entorno de Visual Studio Code	82

Cuantificación de la carga viral de SARS-CoV-2 mediante el procesamiento de imágenes de microgotas fluorescentes

Resumen

Esta tesis presenta el desarrollo de un software con una interfaz intuitiva, diseñado para el conteo y análisis de microgotas fluorescentes a partir de imágenes, con el fin de cuantificar la carga viral de SARS-CoV-2 mediante procesamiento de imágenes. El software permite clasificar las microgotas como positivas o negativas en función de su "nivel de fluorescencia", el cual indica si han reaccionado a marcadores fluorescentes asociados a la presencia de carga viral.

La interfaz gráfica permite sintonizar parámetros de búsqueda clave, como el radio de las microgotas y los umbrales de fluorescencia. De esta manera, se optimiza la precisión del conteo y se facilita el análisis detallado de las imágenes de las muestras de pacientes reales.

El software se basa en una metodología que consta de dos fases: la primera consiste en el procesamiento de las imágenes de las microgotas, y la segunda en el conteo y clasificación de éstas. Para ello, se implementa un conjunto de algoritmos programados en Python, cuyo propósito es realizar las siguientes acciones: (1) Detectar los contornos de las microgotas cuyo diámetro se encuentre dentro de un rango prescrito. (2) Cuantificar la fluorescencia de cada microgota. (3) Clasificar las microgotas dentro de un rango de fluorescencia como brillantes (o fluorescentes), y aquellas fuera de este rango como opacas. (4) Cuantificar las microgotas fluorescentes y las microgotas opacas para después clasificarlas como positivas o negativas, respectivamente. (5) Integrar *sliders* (barras deslizantes) en una interfaz gráfica de usuario (GUI) para ajustar intuitivamente el diámetro de las microgotas y el rango de su fluorescencia.

Para calibrar los parámetros predeterminados que permiten un conteo preciso de microgotas en cada imagen, se estudiaron metodologías previamente empleadas y se llevó a cabo una optimización heurística utilizando diversos tipos de imágenes. No obstante, el software permite ajustar estos parámetros en tiempo real, brindando al usuario la flexibilidad de refinar los ajustes con base en su experiencia, lo cual contribuye a mejorar la precisión del análisis de manera personalizada.

El software desarrollado incluye la opción de emplear una imagen de control para mejorar la precisión del conteo de las microgotas, particularmente en condiciones donde la carga viral es baja o las imágenes dificultan una detección automática. Esta imagen de control permite reducir el sesgo en los resultados al proporcionar una referencia adicional que mejora la clasificación y el conteo de las microgotas.

Además, el software ofrece herramientas que permiten al usuario agregar microgotas que no se detectaron con los parámetros predeterminados, así como eliminar aquellas que se clasificaron erróneamente como positivas. Estas herramientas de ajuste manual son fundamentales, ya que permiten a los usuarios ajustar los resultados según su criterio y experiencia, minimizando errores y optimizando la precisión del análisis de muestras de pacientes.

1 Introducción

1.1 Antecedentes

Los avances en la tecnología MEMS (acrónimo de *Micro Electro Mechanical Systems*) han posibilitado la miniaturización de procesos a escalas micro y nano en diversas áreas de investigación y tecnología, por ejemplo, en instrumentación, telecomunicaciones, metrología e incluso en áreas médicas [1]. La tecnología MEMS también ha permitido el surgimiento de nuevas áreas científicas y tecnológicas, por ejemplo, la microfluídica, engloba la ciencia y tecnología para la manipulación y el control de fluidos en canales y redes fluídicas con dimensiones micrométricas [2], y que ha revolucionado la integración de procesos biológicos y químicos en microdispositivos. En particular, la microfluídica ha permitido crear gotas cuyos volúmenes abarcan de los microlitros hasta los picolitros, que pueden emplearse como biorreactores miniaturizados donde ocurren ensayos bioquímicos. La microfluídica de microgotas ofrece la ventaja de trabajar con volúmenes extremadamente pequeños, lo que permite ahorrar varias órdenes de magnitud en insumos. Además, posibilita la generación de microgotas a tasas muy elevadas, del orden de miles por segundo, incrementando así el rendimiento de los ensayos. Estas microgotas pueden ser controladas, almacenadas, incubadas y reinyectadas directamente en el chip, lo que mejora la sensibilidad, especificidad y precisión en la cuantificación de los procesos celulares [3].

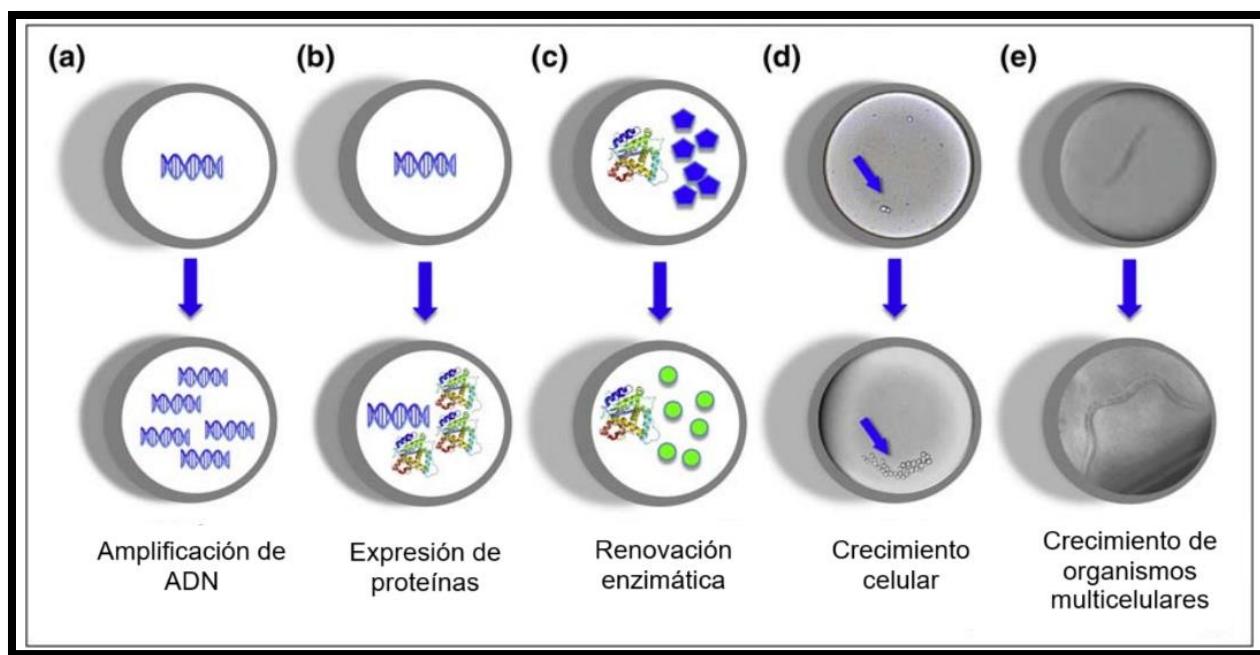


Figura 1.1.1 Procesos biológicos en microgotas. (a) Amplificación de ADN; (b) expresión de proteínas; (c) recambio enzimático que produce un producto fluorescente; (d) encapsulación y crecimiento celular; y (e) crecimiento de organismos multicelulares. Modificado de [4].

Para cuantificar lo que ocurre dentro de cada microgota, los sistemas que detectan la fluorescencia en microgotas han demostrado tener una ventaja sobre las técnicas convencionales gracias a la separación y el aislamiento de cada ensayo. Existen diferentes técnicas para detectar la fluorescencia en microgotas mediante imágenes, como la microscopía

de campo brillante, donde al agregar tintes o trazadores se crean reacciones dentro de las microgotas las cuales son apreciables por cambios colorimétricos, por ejemplo, en la Universidad de Cambridge, se propuso un método basado en microgotas para identificar catalizadores homogéneos nuevos al emplear la oxidación de metano a metanol con oxígeno molecular. Para detectar el metanol producido, idearon un método colorimétrico usando microgotas indicadoras que alteraban su color según la concentración de metanol [5].

Ejemplo de imágenes obtenidas a través de microscopía de campo brillante pueden observarse en la Figura 1.2.

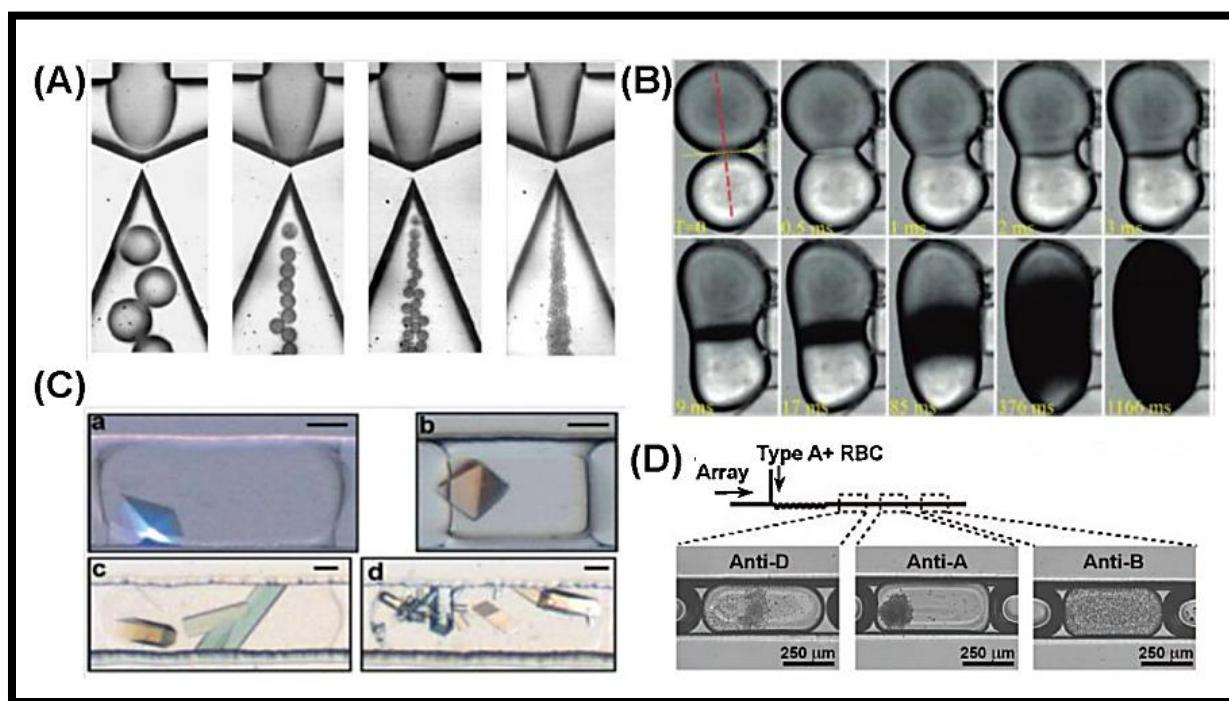


Figura 1.1.2 Imágenes de análisis de microgotas obtenidas con microscopía de campo brillante. (A) Imágenes de alta velocidad del proceso de generación de gotas con geometría de enfoque de flujo. (B) Un sistema de gotas para estudiar la cinética ultrarrápida con resolución submilisegundo utilizando una reacción cromogénica de Fe³⁺ + con SCN⁻. (C) Detección de las condiciones de cristalización de proteínas en microgotas con iluminación de luz polarizada. (D) Tipificación y subtipificación de sangre mediante ensayo de aglutinación celular dentro de gotas. Extraído y traducido de [5].

Por otro lado, el departamento de química de la Universidad de Washington separó una célula en una microgota de aproximadamente un picolitro de volumen a la cual se le provocó la liberación de una enzima al serle aplicada la radiación láser. La actividad enzimática fue visualizada y analizada gracias a la intensidad de fluorescencia [6].

Se han reportado diversas técnicas de biología molecular dentro de microgotas para detectar patógenos, incluidos los virus. Un ejemplo de estas técnicas es el ensayo de amplificación isotérmica mediada por bucle (LAMP, por sus siglas en inglés *loop-mediated isothermal amplification*), que se utiliza para amplificar ADN o ARN específicos. Este proceso requiere una polimerasa denominada Bst, que tiene la capacidad de desplazar las cadenas de ADN, junto con un conjunto de cuatro a seis iniciadores diseñados específicamente para reconocer secuencias

únicas en el ADN objetivo [3]. Ya que esta tecnología produce una cantidad significativa y detectable de ADN se facilita observar el resultado con indicadores fluorescentes o colorimétricos.

Esta tesis surge de la colaboración en una metodología previamente diseñada por un grupo de personas investigadoras de Facultad de Ingeniería, Instituto de Ingeniería y Facultad de Química de la UNAM. Una muestra del paciente se mezcla con componentes para la amplificación genética y se distribuye en miles de microgotas mediante un dispositivo de microfluídica. Las gotas que fluorescen a partir de cierto nivel se consideran positivas y por debajo negativas. Esto permite realizar muchas pruebas de Covid de un mismo paciente en el mismo experimento, aumentando la robustez del resultado en un tiempo de 30 minutos.

Para mejorar y optimizar el conteo de microgotas, se ha desarrollado software que logra identificar, contar y graficar las microgotas a partir de imágenes donde se tienen las células o moléculas de interés iluminadas. Un ejemplo de ello es el algoritmo creado por la Universidad Estatal de Luisiana que lleva por nombre “FluoroCellTrack”. Este algoritmo puede encontrar automáticamente las gotas de las células, determinar el número de células vivas y muertas, cuantificar la fluorescencia intracelular de las células intactas, entre otras tareas vinculadas a la cuantificación. [7]. También existe un software online llamado Biodock el cual se basa en inteligencia artificial para la identificación de material biológico [8].

En concreto, en las áreas médica, biológica y química pueden ser de mucha ayuda estas tecnologías para determinar, no solo si un paciente tiene SARS-CoV 2 sino también permite cuantificar la carga viral, y así, se puedan tomar las medidas clínicas necesarias.

1.2 Objetivos

1. Desarrollar un algoritmo en lenguaje PYTHON que analice imágenes de microgotas tomadas con un microscopio óptico y una cámara Canon Ti6, para diferenciar las microgotas que reaccionan con los marcadores fluorescentes (positivas) de las que no se iluminan (negativas)
2. Diseñar una interfaz para que la persona usuaria pueda ajustar los parámetros de interés (diámetros de gotas y umbrales de intensidad de fluorescencia) y analizar los resultados cuantitativos.

1.3 Justificación

La pandemia de COVID-19 ha acentuado la necesidad de contar con herramientas precisas y eficientes para la detección y cuantificación de la carga viral del SARS-CoV-2. Por ello, el desarrollo de métodos innovadores y accesibles es muy importante. El uso de técnicas de conteo de microgotas fluorescentes ofrece una oportunidad prometedora para detectar rápida y específicamente el virus. Este enfoque, combinado con la versatilidad y potencia de la programación en Python, permite la automatización, análisis de datos y creación de algoritmos para la cuantificación de la carga viral. En este trabajo se propone un método basado en el conteo

de microgotas fluorescentes utilizando Python como herramienta principal de análisis. El software podrá eventualmente utilizarse para detectar fluorescencia en microgotas relacionadas con cualquier otro patógeno de interés.

1.4 Descripción general de la estructura de la tesis

El presente trabajo está dividido en 6 capítulos. Para tener claridad en la estructura de la tesis, a continuación, y de manera breve, se describe en qué consiste cada uno de estos capítulos.

Capítulo 1, Introducción: en esta sección se mencionan los antecedentes relevantes, incluyendo trabajos de colegas, investigadores y desarrolladores que fundamentan y justifican este proyecto. Además, se presentan los objetivos y el propósito del desarrollo de este trabajo.

Capítulo 2, Fundamentos tecnológicos: en este capítulo se explican las herramientas tecnológicas indispensables para el desarrollo del proyecto, ya que sin ellas no podría ser posible dar inicio al trabajo, siendo la creación de microgotas, el lenguaje de programación Python y sus bibliotecas parte de dichas herramientas.

Capítulo 3, Metodología: aquí se explican de manera detallada todas las herramientas que fueron elegidas por su conveniencia y su importancia para la precisión y confiabilidad del proyecto, desde el procesamiento de imágenes y todos los pasos que conlleva, hasta los algoritmos para la detección de las microgotas.

Capítulo 4. Aplicación en Python: esta sección proporciona una descripción detallada del código del software para la detección de microgotas.

Capítulo 5. Resultados: Este capítulo presenta los resultados de la evaluación del software. Esta evaluación incluye el análisis de imágenes de casos de estudios reales. Se presenta la examinación detallada de los hallazgos y se proporcionando una interpretación de los resultados.

Capítulo 6. Conclusiones: en este capítulo se analizan los resultados obtenidos y se presentan conclusiones fundamentadas en ellos. Además, se discuten posibles mejoras.

2 Fundamentos tecnológicos

2.1 Microfluídica, creación de microgotas y el método ddLamp

La microfluídica es una disciplina científica y tecnológica dedicada al control y manejo de fluidos en volúmenes muy reducidos, generalmente en el rango de microlitros (1×10^{-6} litros) incluso nano (1×10^{-9} litros) y pico litros (1×10^{-12} litros). Se utiliza en diversos campos como la biología, química, medicina, física e ingeniería. Un ejemplo de tales dispositivos se ilustra en la Figura 2.1.

En la microfluídica, los fluidos se manipulan en canales y estructuras de dimensiones microscópicas, lo que permite realizar experimentos y análisis con volúmenes de muestra muy pequeños. Esto conlleva ventajas como la reducción de costos, la mejora en la eficiencia de los procesos, la posibilidad de realizar experimentos a alta velocidad y la capacidad de realizar análisis a nivel celular o molecular.

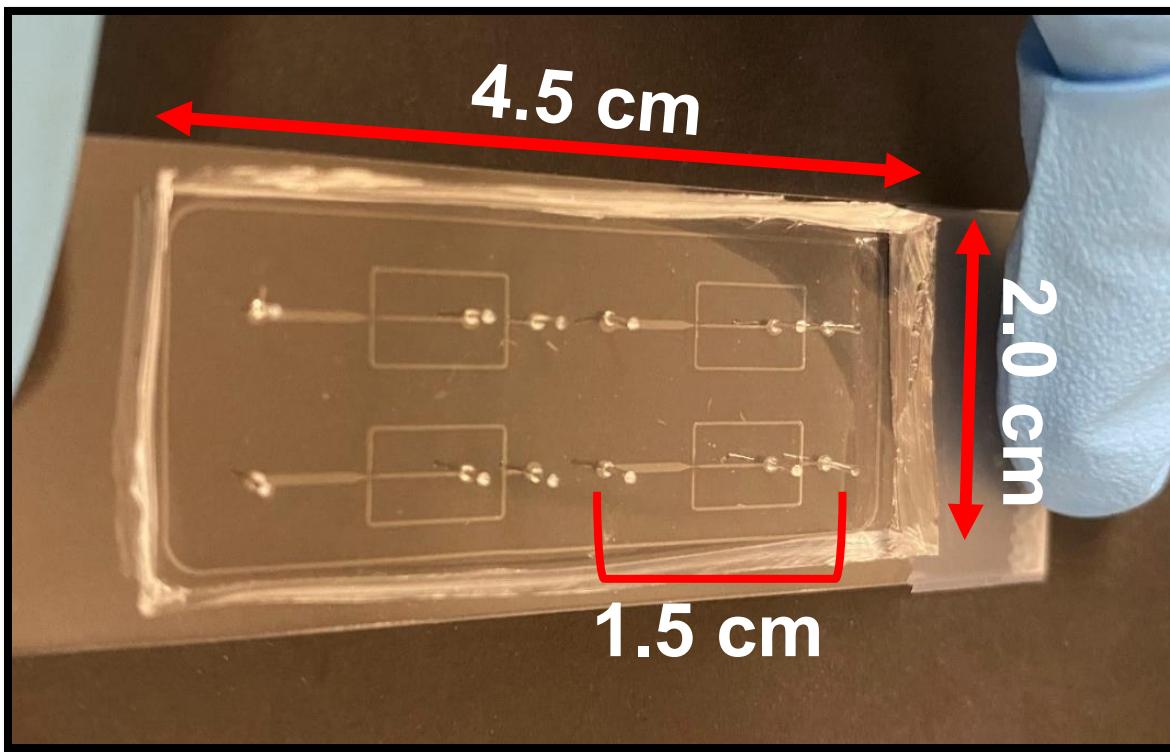


Figura 2.1 Microcanales en un microdispositivo de PDMS. Extraído de [9].

La partición de microgotas en microfluídica se realiza utilizando herramientas, como bombas de jeringa para controlar el flujo o microscopios ópticos para observar las microgotas y microdispositivos especializados como chips microfluídicos hechos de PDMS. Ejemplo de dichos dispositivos se encuentra en la Figura 2.1. Básicamente, implica la creación de microgotas líquidas dentro de otro líquido que no puedan ser mezclados entre sí. Estas microgotas pueden contener muestras, reactivos u otros componentes de interés para llevar a cabo experimentos o análisis específicos. El proceso de partición de microgotas puede variar según el diseño del sistema microfluídico utilizado, pero generalmente involucra la formación controlada de gotas a

partir de un flujo continuo de líquido que se divide en gotas discretas. Esto se logra mediante la aplicación de fuerzas físicas, como la presión y el movimiento dentro de un canal microfluídico. Ejemplo de esto se observa en la Figura 2.2. Cabe resaltar que esta técnica de generación de microgotas no es la única, pero es de las más utilizadas.

Una vez formadas las gotas individuales, estas pueden ser transportadas, manipuladas y procesadas de manera independiente en el sistema microfluídico para llevar a cabo diversas operaciones, como reacciones químicas, análisis biológicos, separaciones, entre otras. La capacidad de dividir gotas en volúmenes muy pequeños y controlar su manipulación en sistemas microfluídicos ha permitido avances significativos en la experimentación de alta velocidad y alto rendimiento en diversos campos científicos como la biología molecular y el método LAMP en gotas digitales (ddLAMP) [10] explicado a continuación.

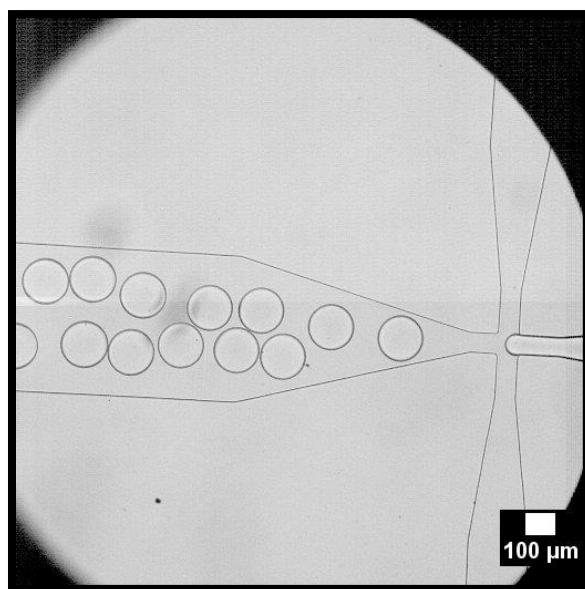


Figura 2.2 Ejemplo de microcanales y generación de microgotas. Extraído de [9].

Han surgido nuevas técnicas de biología molecular para la detección de patógenos, como LAMP en gotas digitales. La reacción LAMP es basada en una amplificación isotérmica mediada por bucles, utiliza un conjunto específico de cebadores para amplificar ADN incluso en cantidades mínimas. En la ddLAMP, la amplificación del ADN se lleva a cabo dentro de gotas individuales (discretas), por eso se considera un entorno digital. Estas gotas contienen todos los componentes necesarios para la reacción de amplificación, incluidos los cebadores específicos para la secuencia de ADN que se desea amplificar. Durante el proceso, la presencia de ADN objetivo genera un producto amplificado que, al contener tintes fluorescentes, emite una señal cuya intensidad varía según la cantidad de ADN presente. Esto explica por qué, en las imágenes de detección viral, un brillo tenue indica una carga viral baja o ausencia del virus, mientras que un brillo intenso señala una carga viral alta. Al realizar la amplificación en gotas individuales, se logra una cuantificación absoluta del ADN sin necesidad de utilizar curvas estándar de calibración. Esto significa que se puede determinar la cantidad exacta de ADN presente en una muestra con una buena precisión y sensibilidad. Básicamente la ddLAMP es una técnica avanzada que combina la alta especificidad y eficiencia de la LAMP con la capacidad de cuantificación absoluta y la facilidad de manejo proporcionada por el formato de gotas digitales. Esto la convierte en gran

herramienta para la detección y cuantificación de ADN en una variedad de aplicaciones científicas y médicas [11].

2.2 Lenguaje de programación Python

Python es un lenguaje de programación creado por Guido van Rossum en 1991, inicialmente desarrollado en el sistema operativo AmoebaOS. Python se utiliza principalmente para automatizar tareas. Aunque comenzó como un lenguaje de scripting, hoy en día es un lenguaje de propósito general, conocido por su sintaxis clara y flexibilidad en aplicaciones.

Una de las características principales de este lenguaje de programación es que, a diferencia de C, trabaja con un intérprete donde una máquina virtual transforma parte del código y lo ejecuta. Otra de las características del lenguaje es su sintaxis similar al lenguaje natural donde la mayoría de las palabras reservadas e instrucciones asemejan a una oración en inglés lo que facilita su aprendizaje para principiantes y su uso eficiente para programadores más experimentados. A pesar de su simplicidad aparente, Python es un lenguaje poderoso que puede utilizarse para una amplia gama de aplicaciones, desde desarrollo web hasta inteligencia artificial. Python ofrece una variedad de estructuras de datos integradas, como listas, matrices, conjuntos y diccionarios, que son fáciles de usar y bastante eficientes (hablando de los recursos del equipo que lo ejecuta). Estas estructuras de datos permiten a los programadores manipular datos efectivamente y realizar operaciones complejas [12]. Además, el ecosistema cuenta con una amplia variedad de módulos de terceros disponibles para su instalación los cuales amplían las capacidades de Python en áreas específicas como el procesamiento de datos, la creación de interfaces gráficas de usuario, desarrollo web, software de diseño gráfico, software de modelado 3D, programas de análisis de datos, entre otros. Algunos de estos módulos son OpenCV (*Open Source Computer Vision Library*) junto con NumPy (*Numerical Python*) y Tkinter, los cuales se describen a continuación.

2.2.1 OpenCV

OpenCV es una biblioteca de programación de código abierto enfocada en la visión por computadora. Esta biblioteca la lanzó Intel en 1999 para ofrecer una colección de funciones de programación optimizada y portátil para facilitar la investigación y el intercambio de conocimientos en el ámbito de la visión por computadora. OpenCV permite que las computadoras procesen y entiendan imágenes y videos, lo que equivale a hacer que una computadora pueda "ver" [13].

Algunos de los propósitos de OpenCV son el procesamiento de imágenes, como aplicar filtros, mejorar la calidad de imagen, transformarlas y realizar operaciones de edición con distintas funciones como el filtro gaussiano para la suavización de la imagen o el filtro Canny para detección de bordes, que, en caso particular de este trabajo, es de mucha importancia para detectar microgotas. Incluso OpenCV incluye una biblioteca de aprendizaje automático que ofrece reconocimiento de patrones, *clustering* y otras tareas de machine learning aplicadas a la visión por computadora. OpenCV puede utilizarse para distintas áreas y aplicaciones aparte de las mencionadas anteriormente, lo que la hace una herramienta fundamental para este trabajo.

2.2.2 Numpy

Numpy es una biblioteca de Python que facilita la estructuración de datos en forma de arreglos (*arrays*) multidimensionales. Las imágenes y matrices de OpenCV pueden convertirse fácilmente en *arrays* de NumPy y viceversa, lo que permite una integración fluida entre ambas bibliotecas. Esto significa que los resultados de procesamiento de OpenCV pueden ser utilizados de manera sencilla como datos de entrada para operaciones en NumPy. Así, NumPy facilita considerablemente las operaciones numéricas en OpenCV, simplificando el trabajo con imágenes y datos relacionados con la visión por computadora [14]. Además de NumPy, OpenCV se puede utilizar con otras bibliotecas, como matplotlib, ampliando el conjunto de herramientas para abordar problemas con imágenes digitales.

2.2.3 Tkinter

Tkinter es la biblioteca estándar de Python para crear interfaces gráficas de usuario (GUI, *Graphic User Interface* por sus siglas en inglés). Esta biblioteca es un *wrapper* o envoltorio de la biblioteca Tcl/Tk, que permite a los desarrolladores crear aplicaciones de escritorio de manera sencilla y rápida. Tkinter viene incluida en la instalación estándar de Python, lo que la hace fácilmente accesible sin necesidad de instalaciones adicionales, aunque si no se encontrara instalada se puede hacer uso del instalador de paquetes ‘pip’ para instalar la biblioteca. Con Tkinter, los desarrolladores pueden diseñar y construir interfaces gráficas de usuario interactivas para sus programas de Python. La biblioteca ofrece una gran variedad de *widgets*, componentes de interfaz gráfica, que se pueden usar para crear aplicaciones con botones, cajas de texto, menús desplegables, listas y mucho más, y también se puede personalizar la apariencia de los *widgets* y responder a eventos, como clic en un botón o escribir en un cuadro de texto. Un ejemplo de una interfaz gráfica creada con esta herramienta se muestra en la Figura 2.3. Algunas características específicas de esta biblioteca son la portabilidad entre diferentes sistemas operativos, su fácil utilización y personalización de *widgets* y finalmente su extensa documentación.

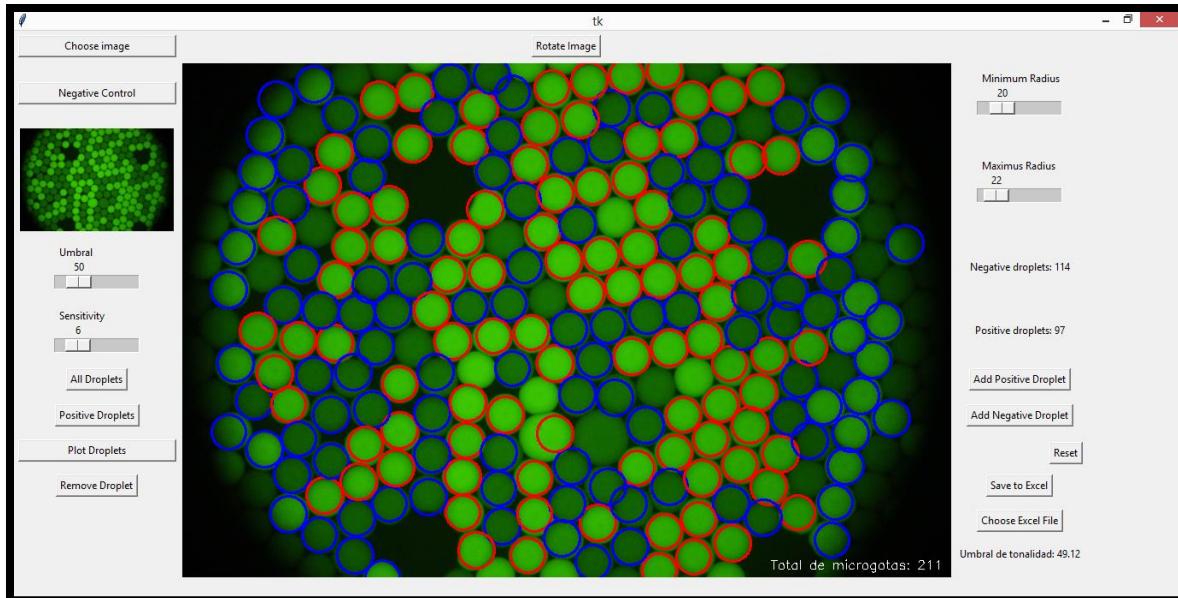


Figura 2.3 Ejemplo de aplicación desarrollada con Tkinter en Python.

3 Metodología

En este capítulo se describe la metodología para cuantificar la carga viral de SARS-CoV-2 mediante el procesamiento de imágenes de microgotas fluorescentes, las cuales, en el contexto de este proyecto, presentan una tonalidad verde. Esta metodología consta de dos fases: la primera consiste en el procesamiento de las imágenes de las microgotas, y la segunda en el conteo de éstas. Para ello, se implementa una serie de procesos algorítmicos programados en Python, cuyo propósito es realizar las siguientes acciones:

- Detectar los contornos de las microgotas cuyo diámetro se encuentre dentro de un rango prescrito.
- Cuantificar la fluorescencia de cada microgota.
- Clasificar las microgotas dentro de un rango de fluorescencia como brillantes (o fluorescentes), y aquellas fuera de este rango como opacas.
- Cuantificar las microgotas fluorescentes y las microgotas opacas.
- Integrar *sliders* en una GUI para ajustar de manera intuitiva el diámetro de las microgotas y el rango de su fluorescencia.

3.1 Fase 1: Procesamiento de las imágenes

La Figura 3.1 muestra un diagrama de flujo que detalla la secuencia ordenada de los procesos algorítmicos, empezando por la transformación de la imagen y culminando en la erosión y dilatación, correspondientes a la primera fase de la cuantificación de la carga viral de SARS-CoV-2 utilizando imágenes de microgotas fluorescentes: el procesamiento de las imágenes. Cada proceso algorítmico fue implementado computacionalmente en el lenguaje de programación Python. Se utilizó en particular la biblioteca OpenCV desarrollada para programar algoritmos de visión computacional. Los procesos algorítmicos de la primera fase se detallan a continuación.

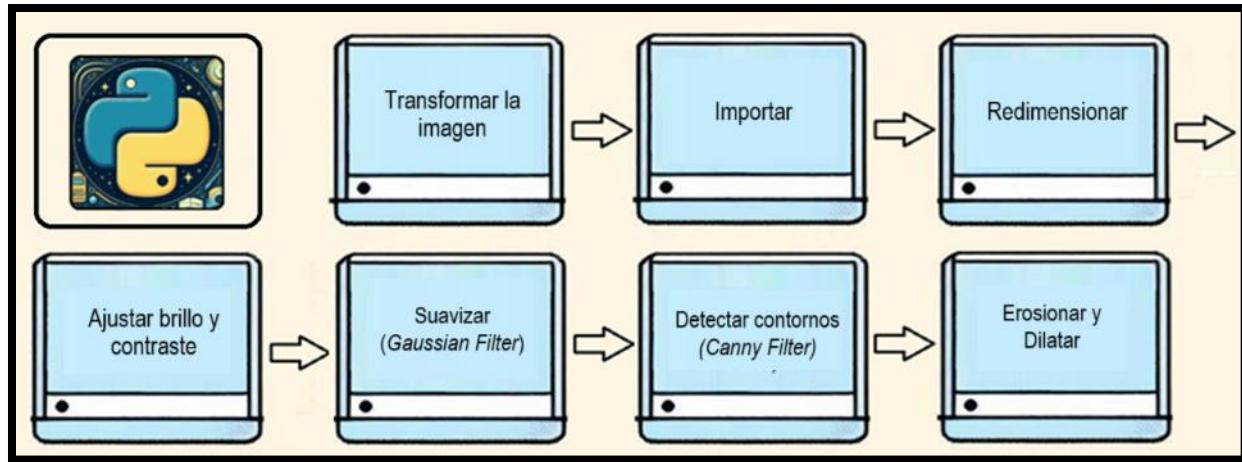


Figura 3.1 Diagrama de flujo para el procesamiento de las imágenes de microgotas.

El objetivo de procesar la imagen es afinar la calidad de los datos de las imágenes de las microgotas. Estas imágenes, al ser capturadas, pueden presentar problemas como ruido, baja definición, contrastes insuficientes o bordes poco definidos, lo que dificulta su análisis. Por ello, para solucionar estos inconvenientes, se emplean los pasos ordenadamente del diagrama de la figura 3.1 y así, asegurar que las microgotas se detecten de manera más confiable.

- 1- Transformar la imagen: Antes de poder manipular los elementos de una imagen a nivel computacional, es necesario convertir el objeto físico, en este caso, microgotas fluorescentes, en una representación numérica (matrices) que pueda ser manipulada mediante software. Este paso es independiente de la aplicación de filtros o herramientas para modificar los píxeles una vez que la imagen ha sido capturada.
- 2- Importar: Después de obtener la imagen de las microgotas fluorescentes, es necesario cargarla en un entorno de trabajo que cuente con las herramientas adecuadas para aplicar los filtros necesarios y resaltar los datos de interés. En particular, se utilizó Visual Studio Code, explicado a detalle en la sección de anexos A.1.
- 3- Redimensionar: La redimensión en este trabajo tiene como propósito ajustar adecuadamente la imagen en el panel de la interfaz gráfica evitando que haya pérdida de datos y también, para asegurar una visualización clara y cómoda para el usuario.
- 4- Ajustar brillo y contraste: El ajuste de brillo y contraste se hace para resaltar los detalles importantes de la imagen y asegurar una buena visibilidad de los bordes, por ejemplo, ya que, si es demasiado oscura, las microgotas pueden resultar difíciles de distinguir.
- 5- Suavizar (filtro Gaussiano): El filtro Gaussiano es necesario para reducir el ruido o variaciones bruscas en la intensidad de los píxeles, ya que pueden ser tomados como falsos bordes.
- 6- Detectar contornos (Filtro Canny): El filtro Canny es esencial debido a que con esto se identifican los bordes más significativos de la imagen al encontrar cambios abruptos en

la intensidad y que son necesarios para la detección de círculos por medio de la función HoughCircles explicada más adelante, en la sección 3.1.

- 7- Erosionar y Dilatar: Mejoran la definición de los bordes eliminando pequeñas irregularidades, La erosión elimina detalles débiles o no deseados, mientras que la dilatación fortalece y resalta los bordes principales, permitiendo una mejor detección de círculos.

3.1.1 Transformar la imagen

Cuando se digitaliza una imagen, esta se convierte en una representación numérica, donde cada píxel corresponde a un valor en una matriz. El proceso comienza con la captura de la imagen a través de una lente que la proyecta sobre un sensor digital o dispositivo sensible a la luz. Esta información se muestrea y cuantifica, resultando en una imagen que se describe mediante una matriz numérica, con píxeles ubicados según coordenadas (x, y). En función del brillo de cada píxel, las imágenes pueden clasificarse de diversas maneras [15]:

- 1) Imágenes binarias: Tienen un rango de dos valores, 0 para negro y 255 para blanco (específico de imágenes en formato de 8 bits por píxel).
- 2) Imágenes en escala de grises: Poseen un rango de 256 niveles de grises.
- 3) Imágenes a color en formato de 8 bits: Cada una de las tres matrices (rojo, verde y azul) tiene un rango de 256 niveles de intensidad. Aunque existen otros formatos como CMYK (el inglés: Cyan, Magenta, Yellow, Black), el uso depende de las necesidades del proyecto.

En la Figura 3.2 se ilustra de manera básica cómo se transforma una imagen a una representación numérica para ser procesada en una computadora.

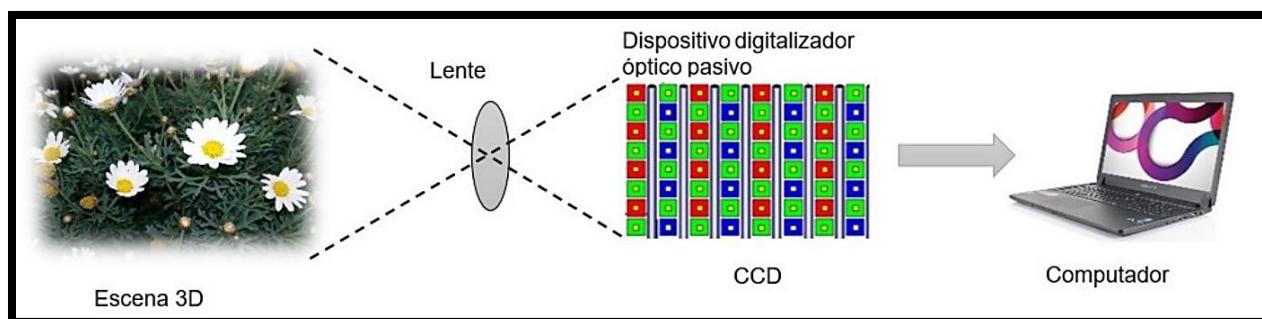


Figura 3.2 Captura de una imagen con un dispositivo pasivo digital. Modificado de [15]

3.1.2 Importar

Para comenzar el procesamiento de la imagen en Python, se requiere importarla al entorno mediante la función `cv2.imread()`. Esta sentencia permite cargar una imagen en varios formatos, como BMP, JPG, PNG o JPEG, entre otros. Una ventaja adicional de esta función es que ofrece dos parámetros para obtener la imagen en diferentes modalidades: en escala de grises con `cv2.IMREAD_GRAYSCALE`, o en colores RGB con `cv2.IMREAD_COLOR` [15]. Esto resulta

especialmente útil, dado que las imágenes con las que se trabajó en este proyecto tienen tonos verdes.

Dos de los argumentos requeridos en esta función son la imagen y la especificación de la bandera que determina si se desea la imagen en escala de grises o a color. Para ilustrar de manera simple, se presenta el siguiente ejemplo práctico de cómo cargar una imagen a color.

Ejemplo:

```
import cv2
img = cv2.imread('Image.jpg', cv2.IMREAD_COLOR)
```

Así, la variable *img* ahora contiene la imagen guardada en 3 matrices correspondientes a los canales RGB, que ahora pueden ser manipuladas, y se procede con la configuración de la imagen. Antes de ajustar la imagen a un tamaño adecuado, es importante determinar la orientación más cómoda para el usuario. Para ello, se rota la imagen 90° antes de redimensionarla.

Rotación

Una vez que la imagen se cargó como una matriz numérica, puede ajustarse según las preferencias del usuario. Esto se logra con la función *cv2.rotate()*, que requiere como entrada la imagen y una bandera que indica el tipo de rotación deseada [16]. Existen tres funciones en la biblioteca OpenCV para rotar las imágenes:

- *cv2.ROTATE_90_CLOCKWISE*: Realiza una rotación de 90 grados en el sentido de las agujas del reloj a la imagen.
- *cv2.ROTATE_90_COUNTERCLOCKWISE*: Ejecuta una rotación de 90 grados en sentido antihorario a la imagen.
- *cv2.ROTATE_180*: Aplica una rotación de 180 grados en el sentido de las agujas del reloj a la imagen.

Ejemplo:

```
imgRot = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)
```

En la línea anterior, se genera una nueva imagen “*imgRot*” a partir de la imagen original “*img*” rotada a 90° a la derecha.

Redimensión

Para ajustar las dimensiones de una imagen se puede usar la función *cv2.resize()*, que permite establecer tanto el tamaño deseado como el método de interpolación. Para definir el tamaño, es necesario indicar el ancho y largo dentro de la misma función. Cuando se disminuye el tamaño de una imagen, algunos píxeles deben eliminarse. Cuando se aumenta el tamaño de una imagen,

se deben generar nuevos píxeles y los píxeles originales se deben distribuir de alguna manera en la nueva configuración. La interpolación es el método que se utiliza para estimar los valores de los nuevos píxeles, basándose en los píxeles originales circundantes.

Existen varios métodos de interpolación, y el elegido afecta cómo se calculan los valores de los nuevos píxeles. Algunos métodos comunes de interpolación incluyen:

- `cv2.INTER_NEAREST` (píxel vecino más cercano)
- `cv2.INTER_LINEAR` (Interpolación Bilineal)
- `cv2.INTER_CUBIC` (Interpolación Bicúbica)
- `cv2.INTER_LANCZOS4` (Interpolación Lanczos)
- `cv2.INTER_AREA` (Relación de Área de Píxeles)

Este último fue el que se utilizó en este trabajo de tesis. El método emplea una relación de área de píxeles para redistribuir los valores. Es más efectivo al reducir el tamaño de la imagen, pero al aumentar el tamaño, puede generar resultados borrosos [17].

En concreto, para ajustar las dimensiones de una imagen utilizando esta función, es necesario proporcionar la imagen y especificar el tamaño deseado (en píxeles), seguido del método de interpolación que se va a emplear. Así, el ejemplo de esta instrucción quedaría de la siguiente manera:

Ejemplo:

```
import cv2
img_path = 'ruta_de_la_imagen.jpg'
img = cv2.imread(img_path)
imgResized = cv2.resize(img, (1000, 700), interpolation=cv2.INTER_AREA)
```

En este caso, se puede observar que la nueva imagen es denominada como imgResized, adquiere las dimensiones de 1000x700 píxeles y utiliza el método de interpolación `cv2.INTER_AREA`.

Separación de componentes de color

Cuando se importa una imagen con OpenCV, la imagen se representa como una matriz Numpy. Comúnmente para imágenes de color, esta matriz es tridimensional y tiene dimensiones (*alto, ancho, canal*). Cada elemento de la matriz es un valor entero que representa la intensidad del color en un píxel específico. Para una imagen en color, los tres canales representan los componentes de color BGR (*azul, verde, rojo*). Cada canal es una matriz bidimensional (*alto, ancho*) de valores enteros, donde cada valor puede variar de 0 a 255.

Donde:

Canal 0: Azul (B)

Canal 1: Verde (G)

Canal 2: Rojo (R)

Cada valor en las matrices individuales de los canales representa la intensidad del color en un píxel específico en la escala de 0 a 255. Para entenderlo mejor, en la Figura 3.3 se muestra una comparación entre la matriz de una imagen en escala de grises y las matrices correspondientes a una imagen a color.

$I(x, y) = \begin{bmatrix} 2 & 4 & 3 \\ 3 & 4 & 5 \\ 6 & 2 & 3 \end{bmatrix}$	$R(x, y) = \begin{bmatrix} 3 & 4 & 3 \\ 2 & 3 & 5 \\ 6 & 7 & 3 \end{bmatrix}$
$G(x, y) = \begin{bmatrix} 7 & 5 & 3 \\ 3 & 4 & 5 \\ 5 & 3 & 4 \end{bmatrix}$	$B(x, y) = \begin{bmatrix} 3 & 4 & 4 \\ 4 & 5 & 5 \\ 4 & 2 & 7 \end{bmatrix}$

Figura 3.3 Representación matricial de una imagen en escala de grises y de una imagen a color. Extraído de [15].

La función `cv2.split()` entrega una lista de canales, donde cada canal es una matriz Numpy bidimensional que representa un solo color. En el caso de una imagen en color con tres canales (BGR), cada canal individual es una matriz bidimensional. Un ejemplo de la redimensión y el uso de la función `cv2.split()` se muestra a continuación.

Ejemplo:

```
# Importar La biblioteca cv2
import cv2
# Ruta de la imagen
img_path = 'ruta_de_la_imagen.jpg'
# Cargar La imagen
img = cv2.imread(img_path)
# Redimensionar La imagen a 1000x700 píxeles
img_resized = cv2.resize(img, (1000, 700), interpolation=cv2.INTER_AREA)
# Mostrar La imagen redimensionada
cv2.imshow("Imagen Redimensionada", img_resized)
cv2.waitKey(0)
# Separar Los canales RGB
b, g, r = cv2.split(img_resized)
# Mejorar el contraste del canal verde
enhanced_green = cv2.equalizeHist(g)
# Combinar Los canales de nuevo
enhanced_image = cv2.merge([b, enhanced_green, r])
# Mostrar La imagen con el canal verde mejorado
cv2.imshow("Imagen con canal verde mejorado", enhanced_image)
cv2.waitKey(0)
# Cerrar Las ventanas cuando se presiona cualquier tecla
cv2.destroyAllWindows()
```

En el ejemplo anterior, se carga y redimensiona una imagen a 1000x700 píxeles, mostrándola en una ventana. Luego, separa los canales de color, mejora el contraste del canal verde, y vuelve a combinar los canales. Después muestra la imagen con el canal verde mejorado y cierra las ventanas al presionar una tecla.

3.1.3 Ajuste de brillo y contraste

La función `convertScaleAbs()` en OpenCV ajusta los valores de una imagen multiplicándolos por un número (`alpha`) y sumándoles un valor (`beta`). Luego toma el valor absoluto y lo convierte a un rango entre 0 y 255, que es necesario para trabajar con imágenes de 8 bits y asegurar que la imagen tiene valores válidos después de realizar operaciones (como el ajuste de brillo y contraste) que pueden generar números negativos o fuera de rango.

El parámetro `alpha` se utiliza para escalar los valores de los píxeles de la imagen de entrada; si `alpha` es mayor que 1, los valores de los píxeles aumentan, lo que podría incrementar el brillo o el contraste de la imagen, mientras que, si es menor que 1, los valores disminuyen, oscureciendo la imagen o reduciendo el contraste. El parámetro `beta` es una constante que se suma a los valores de los píxeles después de haber sido escalados por `alpha`; se utiliza para ajustar los valores de los píxeles en la imagen, añadiendo un valor constante que puede aumentar o

disminuir el brillo de la imagen de manera uniforme; si beta es positivo, los valores de los píxeles se incrementan, y si es negativo, los valores se reducen.

Su sintáxis es: `convertScaleAbs (src, dst, alpha = 1, beta = 0)`

donde:

src: matriz de entrada.

dst: matriz de salida.

alpha: factor de escala.

beta: incremento que se añade a los valores escalados.

Ejemplo:

```
import cv2
file_path = r"C:\Users\zuli_\OneDrive\Escritorio\YinYang.jpg"
image = cv2.imread(file_path)
imageResize= cv2.resize(image,(600,700))
alpha = 1.9
beta = 0.5
Brillo_Contrast = cv2.convertScaleAbs(imageResize, alpha = alpha,
beta=beta)
cv2.imshow('Image ajuste Brillo', Brillo_Contrast)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Este código carga la imagen, la redimensiona a 600x700 píxeles y ajusta su brillo y contraste usando los factores `alpha` (contraste) y `beta` (brillo). Muestra la imagen ajustada en una ventana hasta que se presione una tecla. El resultado del código anterior se puede apreciar en la Figura 3.4. En esta figura, la imagen de la izquierda es la original, mientras que la de la derecha tiene un ajuste de brillo y contraste con los valores `alpha` 1.9 y `beta` 0.5.



Figura 3.4 Ajuste de brillo y contraste de una imagen.

3.1.4 Suavizar (Filtro Gaussiano)

Para entender el proceso de suavización de una imagen utilizando *Gaussian Filter*, es importante tener en cuenta el concepto de *kernel*. En procesamiento de imágenes, un *kernel* se define como un conjunto de ponderaciones que se aplican a una región específica de la imagen original con el fin de generar un solo píxel en la imagen resultante. Por ejemplo, si el tamaño del *kernel* es de 3x3 significa que se utilizan 9 píxeles (3 píxeles de ancho por 3 píxeles de alto) de la imagen original para calcular el valor de un solo píxel en la imagen resultante [17]. Se puede visualizar el kernel como un pequeño recuadro que se desplaza gradualmente sobre la imagen de origen. A medida que se desplaza, este "recuadro" difumina la luz proveniente de la fuente, suavizando así la transición entre los píxeles y reduciendo la presencia de detalles no deseados. En la Figura 3.5 se muestra el ejemplo de kernels gaussianos de diferente tamaño.

Ejemplos de Kernels Gaussianos							
$\left[\frac{1}{16}\right]$	$\begin{vmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{vmatrix}$	$\left[\frac{1}{273}\right]$	$\begin{vmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{vmatrix}$	$\left[\frac{1}{140}\right]$	$\begin{matrix} 1 & 1 & 2 & 2 & 2 & 1 & 1 \\ 1 & 2 & 2 & 4 & 2 & 2 & 1 \\ 2 & 2 & 4 & 8 & 4 & 2 & 2 \\ 2 & 4 & 8 & 16 & 8 & 4 & 2 \\ 2 & 2 & 4 & 8 & 4 & 2 & 2 \\ 1 & 2 & 2 & 4 & 2 & 2 & 1 \\ 1 & 1 & 2 & 2 & 2 & 1 & 1 \end{matrix}$	7×7	
	3×3		5×5				

Figura 3.5 Ejemplos de Kernels Gaussianos

El filtro gaussiano sustituye cada píxel con una combinación ponderada de los píxeles en una región rectangular que rodea al píxel central. A diferencia del filtro de mediana, que usa el valor mediano de la región, el filtro gaussiano aplica una función matemática, la distribución gaussiana, para calcular una ponderación de los píxeles vecinos. La imagen se recorre y cada píxel se reemplaza por un valor que es el resultado de multiplicar y sumar el kernel con la región de la imagen que lo rodea. Esto se realiza para cada píxel de la imagen [18]. Ejemplo de ello se muestra en la Figura 3.6.

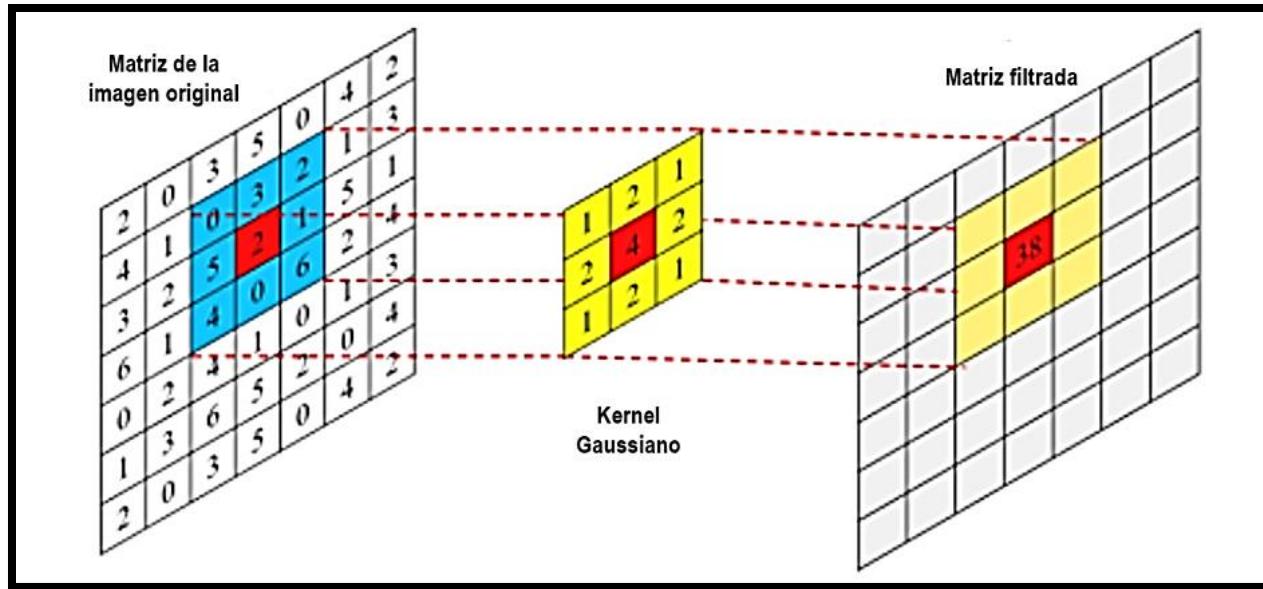


Figura 3.6 Representación de matrices usando filtro Gaussiano. Modificado de [18]

En el filtrado gaussiano, los píxeles más cercanos al píxel central tienen mayor peso en el cálculo del nuevo valor del píxel, mientras que los píxeles más alejados tienen menor peso. Este proceso suaviza la imagen y reduce el ruido, produciendo un desenfoque más natural y gradual que el filtro de mediana. Para utilizar un filtro gaussiano en Python con el apoyo de la biblioteca OpenCV, se emplea la función `cv2.GaussianBlur('imagen', 'tamaño_kernel', 'sigma')`. Esta función toma como entrada la imagen a la que se aplicará el filtro gaussiano, el tamaño del kernel (que debe ser impar), y el valor de sigma, que controla la cantidad de suavizado. Es importante elegir un tamaño de kernel impar para asegurar que haya un píxel central y que el filtro se aplique de manera uniforme alrededor de cada píxel. Para exemplificar se tiene el siguiente código.

Ejemplo:

```

import cv2
# Cargar la imagen
imagen = cv2.imread('nombre_imagen.jpg', cv2.IMREAD_GRAYSCALE)
# Especificar el tamaño del kernel (debe ser un número impar)
tamaño_kernel = (5, 5) # Por ejemplo, un kernel de 5x5
sigma = 1.0 # Especificar el valor de sigma
# Aplicar el filtro gaussiano
imagen_filtrada = cv2.GaussianBlur(imagen, tamaño_kernel, sigma)
# Mostrar La imagen filtrada
cv2.imshow('Imagen Filtrada con Gaussian Blur', imagen_filtrada)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

En la Figura 3.7, se ilustra el efecto de aplicar un filtrado Gaussiano a una imagen.



Figura 3.7 Imagen suavizada después de utilizar el filtro Gaussiano. Extraído de [19].

3.1.5 Detección de Contornos (Filtro de Canny)

El filtro de Canny se utiliza para la detección de bordes, este transforma una imagen de entrada en una imagen de contornos (*edge image*). Este proceso implica identificar y resaltar las regiones de la imagen donde hay cambios significativos en la intensidad, lo que permite visualizar de manera clara las transiciones entre distintas áreas de la imagen. En otras palabras, la aplicación del filtro de Canny ayuda a destacar los contornos y bordes presentes en la imagen original [19]. Debido a que este algoritmo es sensible al ruido, previamente debe someterse a una fase de suavización, como se hace con `cv2.gaussianBlur()` para posteriormente aplicarse dos filtros Sobel a la imagen ya suavizada, uno en dirección horizontal y otro en dirección vertical

El filtro Sobel es un operador que resalta las áreas de cambio de intensidad en una imagen, en este caso, se utiliza para calcular las derivadas parciales en las direcciones x e y . Después de aplicar el filtro Sobel en dirección horizontal, obtenemos una imagen que representa la primera derivada en dirección horizontal para cada píxel (G_x). De manera similar, al aplicar el operador Sobel en dirección vertical, obtenemos otra imagen que representa la primera derivada en dirección vertical (G_y). El gradiente del borde en cada píxel se calcula utilizando las derivadas obtenidas en las direcciones horizontal y vertical [20]:

$$\text{Gradiente} = \sqrt{(G_x)^2 + (G_y)^2}$$

La dirección del borde en cada píxel se puede obtener con:

$$\text{Dirección} = \tan^{-1} \frac{(G_y)}{(G_x)}$$

Después de calcular la magnitud y dirección del cambio en la intensidad de los píxeles (gradiente), se recorre toda la imagen para eliminar los píxeles que no son parte del borde. Esto se hace revisando cada píxel y comprobando si es el más “fuerte” de los píxeles más cercanos en la dirección del cambio de intensidad. Es decir, se eliminan los píxeles que no son el punto más destacado a lo largo del borde. A esto se le conoce como supresión de no máximos.

Para una mejor comprensión se tiene que:

- Si la magnitud del gradiente en el píxel actual es mayor que la de sus dos vecinos en la dirección del gradiente, se conserva el valor del píxel actual.
- Si la magnitud del gradiente en el píxel actual es menor que al menos uno de sus píxeles vecinos, se suprime (se establece su magnitud a cero).

Esto puede visualizarse en la Figura 3.8.

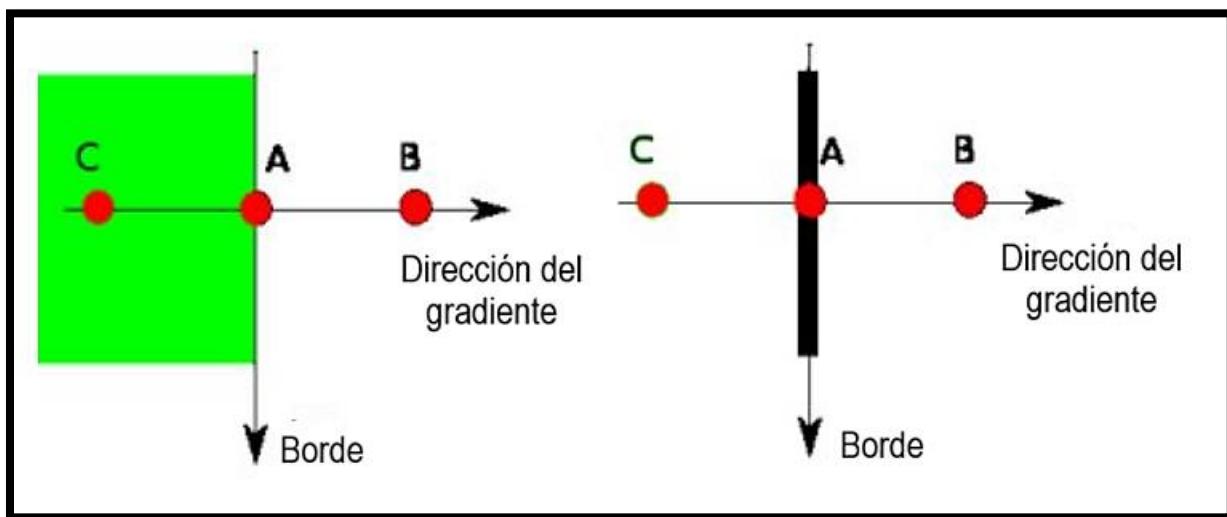


Figura 3.8 Ejemplo gráfico de supresión de no máximos. Modificado de [20]

El punto A se encuentra en el borde en una dirección vertical. La dirección del cambio de intensidad es perpendicular al borde. Ahora, se compara el punto A con los puntos B y C, que están en direcciones de gradiente. Se verifica si el punto A forma un máximo local en comparación con B y C. Si es así, se conserva para la siguiente etapa; de lo contrario, se elimina (se establece en cero). Posteriormente, se aplican umbrales para determinar qué píxeles se consideran bordes. Se utilizan dos umbrales: un umbral inferior y un umbral superior. Los píxeles con gradientes superiores al umbral superior se consideran píxeles de borde fuertes, y aquellos entre los umbrales inferior y superior se consideran píxeles débiles. Los píxeles débiles que están

conectados a píxeles fuertes se mantienen, y los demás se eliminan. Para entender mejor el concepto, se tiene la siguiente representación gráfica (Figura 3.9).

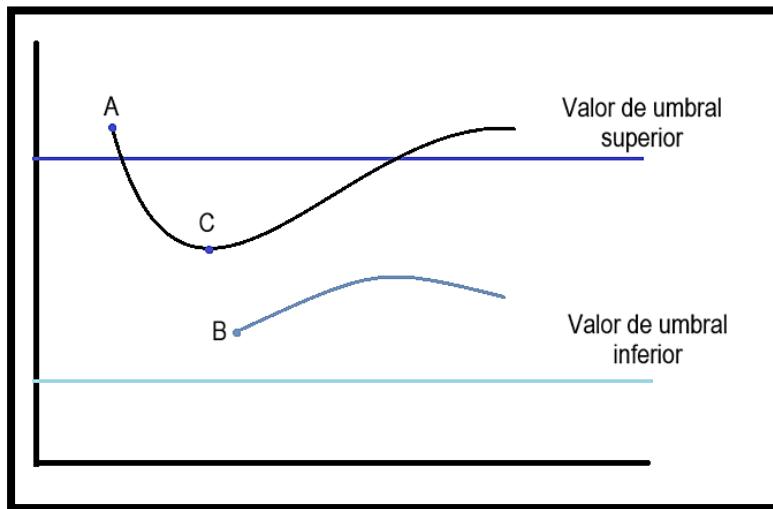


Figura 3.9 Etapa final del filtro Canny: umbralización. Modificado de [20]

El borde A se considera "seguro" debido a que su intensidad supera el umbral superior. Aunque el borde C no alcanza ese umbral, se conecta con el borde A y se incluye como un borde válido, formando una curva continua. En contraste, el borde B, aunque supera el umbral inferior y está cerca del borde C, no se conecta con ningún "borde seguro" y por lo tanto se descarta [20]. Para resumir cómo funciona el filtro Canny, se puede decir que existen 4 pasos: (1) reducción de ruido (suavización), (2) cálculo de gradientes, (3) supresión de no máximos. (4), umbralización.

Ejemplo:

```
import cv2
file_path = r"C:\Users\zuli_\OneDrive\Escritorio\YinYang.jpg"
image = cv2.imread(file_path)
Canny = cv2.Canny(image, 80, 80*2)
cv2.imshow('Canny', Canny)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Este código carga una imagen (*image*) y aplica el detector de bordes de Canny con umbrales de 80 y 160 (80*2). Como resultado, se tiene la Figura 3.10 donde la imagen del lado derecho es a la cual le fue aplicado el filtro Canny y a la izquierda la imagen original.



Figura 3.10 . Ejemplo de utilizar filtro Canny en una imagen. En el lado izquierdo se encuentra la imagen original, mientras la imagen derecha se encuentra con el filtro de Canny aplicado.

3.1.6 Erosión y Dilatación

La dilatación y la erosión son dos operaciones básicas en el procesamiento de imágenes utilizadas en una amplia gama de aplicaciones como reducción de ruido, segmentación de objetos y mejoramiento de características de imagen. Estas operaciones también pueden emplearse para identificar puntos destacados o huecos en una imagen y para definir un gradiente de imagen específico. En general, cuando se aplica la dilatación a una región luminosa en una imagen, esta área se expande, mientras que la erosión la reduce. Además, la dilatación tiende a llenar espacios vacíos dentro de los objetos, mientras que la erosión suele eliminar protuberancias o partes sobresalientes. Estas transformaciones se realizan mediante las funciones `cv2.erode()` y `cv2.dilate()` [19].

Erosión (erode)

La función `cv2.erode()` toma tres argumentos principales:

- La imagen de entrada.
- El kernel, también conocido como elemento estructurante que se utilizará para la operación de erosión.
- El número de iteraciones, que especifica cuántas veces se aplicará la erosión. En la mayoría de los casos, se utiliza 1 iteración.

La operación de erosión se realiza deslizando el kernel sobre la imagen. En cada paso, el valor del píxel central se ajusta al valor mínimo encontrado dentro del área cubierta por el kernel. Esto resulta en la reducción del tamaño de los objetos y la eliminación de pequeños detalles o ruido en la imagen. La idea clave es "erosionar" los bordes de los objetos en la imagen, lo que implica reducir su tamaño. Para lograr esto, se busca encontrar el valor más bajo (más oscuro) dentro del área cubierta por el kernel. Esto se debe a que los píxeles más oscuros suelen representar

los límites de los objetos. A continuación, se muestra un ejemplo sencillo (Figura 3.11). Dada la matriz original y el kernel para la erosión:

Matriz original de 8x8							
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Kernel 3x3		
1	1	1
1	1	1
1	1	1

Figura 3.11 Ejemplo de matriz 8x8 y kernel 3x3.

Al aplicar la erosión, se coloca el kernel en cada posición de la matriz original y se verifica si todos los elementos coinciden con los elementos correspondientes de la matriz original. Si todos los elementos coinciden, se mantiene el valor de 1 en la posición central; de lo contrario, se coloca un 0.

Kernel		
1	1	1
1	(1)	1
1	1	1
Matriz original		
0	0	0
0	0	0
0	0	(1)
0	0	1
0	0	1
0	0	1
0	0	0
Matriz erosionada		
0	0	0
0	0	0
0	0	0
0	0	1
0	0	1
0	0	0
0	0	0
0	0	0

Figura 3.12 Erosión de una matriz 8x8 con un kernel 3x3.

Por lo tanto, en cada posición del kernel, se establece el valor del píxel central en la imagen erosionada como el valor mínimo encontrado dentro del área cubierta por el kernel. Así, se asegura que los objetos se reduzcan gradualmente en tamaño según se aplica la erosión. El resultado de la Figura 3.12 es el siguiente:

Matriz original de 8x8	Matriz erosionada
$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$



Figura 3.13 Ejemplo de erosión.

En la Figura 3.13 se muestra el ejemplo de una matriz a la que se le ha aplicado erosión.

Dilatación (dilate)

La función `cv2.dilate()` se utiliza en el procesamiento de imágenes y visión por computadora para aumentar el tamaño de los objetos en primer plano y reducir el tamaño de huecos y áreas oscuras. Esta operación es esencialmente lo opuesto a la erosión. Tomando como referencia los puntos clave de la función `cv2.erode()`, se muestra a continuación cómo funciona la dilatación con `cv2.dilate()`:

La función `cv2.dilate()` también toma tres argumentos principales:

- La imagen de entrada: imagen sobre la que se aplicará la dilatación.
- El kernel: Se utiliza para decidir cómo se dilata la imagen (la forma y el tamaño del kernel afectan directamente la cantidad y la forma de la dilatación).
- El número de iteraciones: Especifica cuántas veces se aplicará la dilatación a la imagen. Una sola iteración es común, pero se pueden aplicar múltiples iteraciones para una dilatación más pronunciada.

La operación de dilatación, al igual que la erosión, desliza el kernel sobre la imagen. Sin embargo, en lugar de ajustar el valor del píxel central al valor mínimo dentro del área cubierta por el kernel (como en la erosión), la dilatación ajusta el valor del píxel central al valor máximo encontrado. Esto tiene el efecto de "engrosar" o expandir los objetos en la imagen y cerrar pequeños huecos.

Ejemplo:

Retomando el ejemplo de la Figura 3.13. Al aplicar la dilatación, se coloca el kernel en cada posición posible sobre la matriz original. Si al menos un elemento del kernel se sobreponer a un elemento del objeto (valor 1) en la imagen, entonces el píxel central bajo el kernel se establece a 1 (blanco) en la imagen resultante. Esto hace que los objetos en la imagen se expandan. La imagen resultante (Figura 3.14) tendrá los objetos de primer plano más grandes y los huecos y áreas oscuras más pequeños.

Matriz original de 8x8	Matriz dilatada
$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$



Figura 3.14 Ejemplo de dilatación de una imagen.

3.2 Fase 2: Conteo de las microgotas

En esta sección se presentan los procesos algorítmicos utilizados para realizar el conteo de microgotas una vez que la imagen ha sido procesada y está lista para la aplicación de funciones de detección y conteo de círculos, que son las microgotas de interés en este estudio. Es importante mencionar que esta es una aproximación, ya que las gotas parecen círculos debido al ángulo o perspectiva con el que se captura la imagen. Además, no todas las gotas generan círculos perfectos, lo que puede influir en la interpretación de los resultados ya que el algoritmo como se explica en esta sección trabaja con base en suposiciones de figuras geométricas circulares.

3.2.1 La transformada de Hough

La Transformada de Hough es una técnica para procesar imágenes utilizada para detectar formas geométricas como líneas, círculos o elipses. Esta técnica convierte la detección de una forma en

la imagen en un problema de búsqueda en un espacio de parámetros. Para círculos, cada punto de la imagen puede corresponder a varios círculos con distintos centros y radios, lo que se resuelve mediante la acumulación en un espacio tridimensional de parámetros (centro en x, centro en y, y radio). El método de Hough para la detección de círculos trabaja con imágenes de bordes, lo que permite identificar los contornos de los círculos de manera más eficiente. En este espacio de parámetros, se proyectan los posibles círculos presentes en la imagen. La ecuación que describe un círculo es $(x - a)^2 + (y - b)^2 = r^2$, donde x e y son las coordenadas de un punto en la circunferencia, a y b son las coordenadas del centro, y r es el radio. La transformada circular de Hough utiliza esta ecuación para buscar combinaciones de a, b y r que formen círculos en la imagen original. Cada píxel de borde detectado en la imagen se proyecta en el espacio de parámetros como un conjunto de posibles centros representando todas las ubicaciones potenciales del centro del círculo.

El algoritmo guarda estas proyecciones en una matriz acumuladora tridimensional registrando cuántas curvas se intersecan en cada punto del espacio de parámetros. Los valores numéricos más altos, "picos" o también llamados "votos" dentro de esta matriz indican la presencia de un círculo, ya que un alto número de intersecciones sugiere que muchos puntos de borde coinciden en un centro común.

Para visualizar este método, la Figura 3.15 muestra en dos dimensiones (a y b, asumiendo un valor conocido para r) los espacios de imagen y de parámetros. A la izquierda se observa el círculo original en el espacio de la imagen y a la derecha el espacio de proyección donde se "generan" las circunferencias (potenciales centros) que permiten identificar el centro del círculo, en nuestro caso, proyectado por una posible microgota

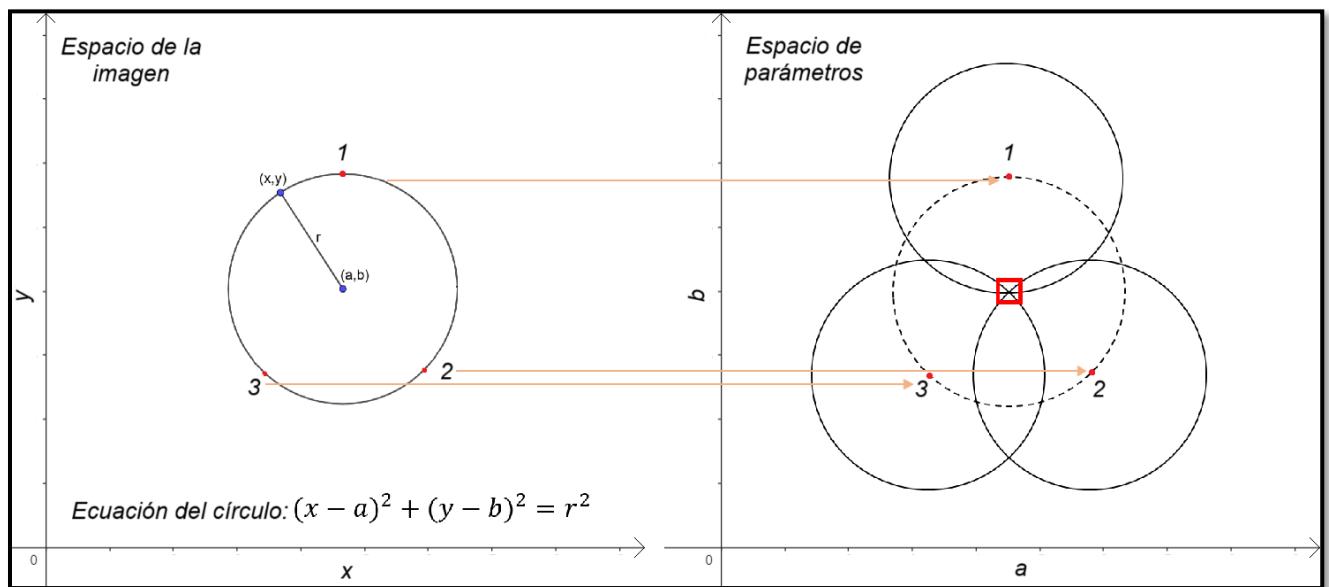


Figura 3.15 Representación gráfica de la transformada de Hough para circunferencias asumiendo un valor para r (bidimensional).

Sin embargo, dado que el acumulador está en tres dimensiones porque r es también una incógnita, el número de valores a, b y r a analizar es mayor, lo que incrementa el consumo de recursos computacionales y reduce la velocidad [19], haciendo que este método sea menos

eficiente en entornos como Python. Gráficamente el acumulador tridimensional se puede visualizar de la siguiente manera (Figura 3.16).

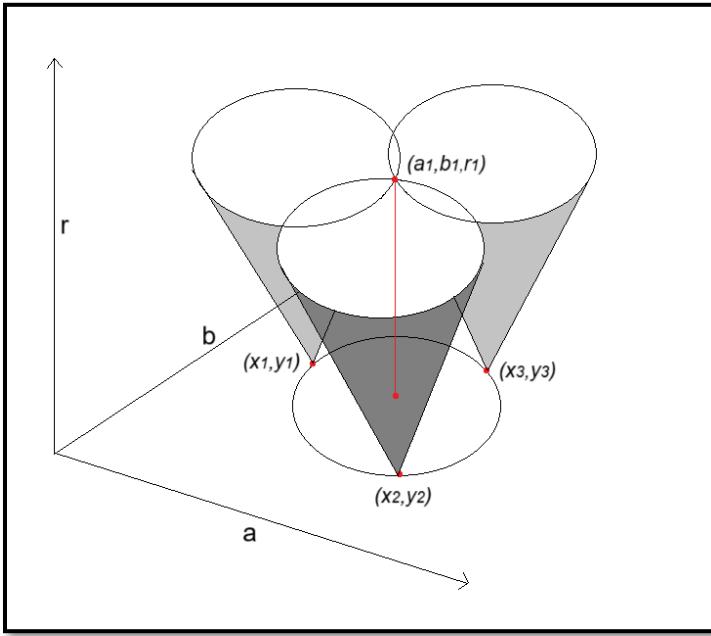


Figura 3.16 Representación gráfica de la transformada de Hough para circunferencias (tridimensional).

Si (a_1, b_1, r_1) son puntos que pertenecen al círculo del espacio de la imagen, las superficies cónicas generadas se unirán en ese punto en el espacio de parámetros. En la imagen original, este punto representa tanto el centro del círculo como su radio [21].

Como se mencionó antes, el uso de un acumulador tridimensional es inefficiente, por lo que OpenCV ha desarrollado distintos métodos que se adaptan a las necesidades del programador, ya sea para encontrar líneas o círculos y elipses. En particular, para el desarrollo de este trabajo el método utilizado fue *Hough Gradient* (`cv.HOUGH_GRADIENT`). A continuación, se describe el funcionamiento de dicho método.

Hough Gradient (`cv.HOUGH_GRADIENT`)

Este método se utiliza específicamente para detectar círculos. En lugar de analizar todas las posibles ubicaciones y radios de los círculos, este método aprovecha la información de los gradientes en los bordes de la imagen. Funciona calculando las direcciones del gradiente en cada píxel del borde para identificar los puntos con alta probabilidad de ser centros de círculos. Posteriormente, estos puntos se guardan en la matriz acumuladora, donde aquellos con mayor número de “votos” o coincidencias se consideran como posibles centros de círculos. En la Figura 3.17, se muestra a la izquierda la imagen de bordes en el plano x, y . A la derecha, se presenta la proyección en el espacio de parámetros, donde se visualizan como “líneas” los puntos que tienen probabilidad de ser centros. La mayor acumulación de estos puntos (o intersecciones), representan los centros a, b de los círculos detectados. Estas líneas son solo una representación visual para entender mejor el concepto. El rango del radio r especificado previamente ayuda a

concentrar la búsqueda, asegurando que los círculos detectados correspondan a los tamaños esperados.

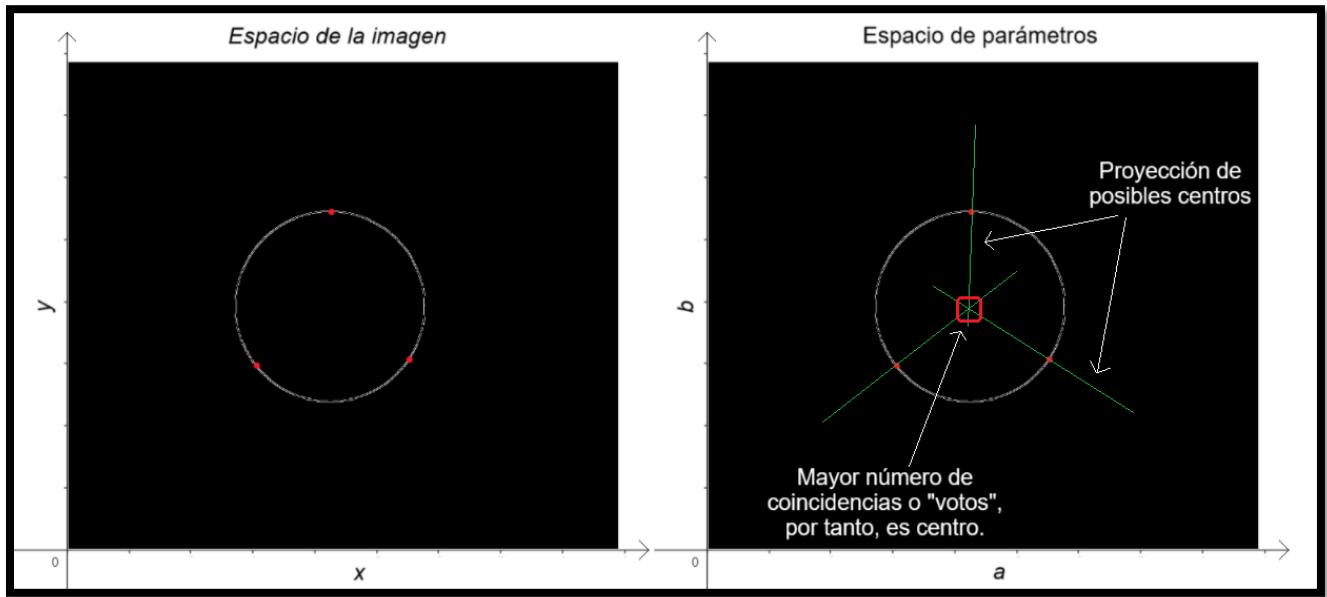


Figura 3.17 Representación gráfica de la transformada de Hough para circunferencias utilizando HOUGH_GRADIENT.

Cuando estas “líneas” se intersecan en el espacio de parámetros, significa que los valores numéricos en ciertas celdas de la matriz acumuladora se incrementan más que en otras, indicando una alta probabilidad de que en ese punto esté el centro de un círculo. Este método se implementa en OpenCV a través de la función `cv2.HoughCircles`, que permite detectar círculos en una imagen mediante la especificación de varios parámetros, como el umbral para el detector de bordes, el rango de radios de los círculos, y la distancia mínima entre los centros de los círculos detectados [22].

Ejemplo:

```
import cv2
import numpy as np
# Cargar la imagen en escala de grises
image = cv2.imread('imagen.jpg', cv2.IMREAD_GRAYSCALE)
# Aplicar desenfoque para suavizar la imagen
blurred_image = cv2.GaussianBlur(image, (7, 7), 1.5)
# Detectar los bordes en la imagen
# Detectar círculos usando HoughCircles
detected_circles = cv2.HoughCircles(edges, cv2.HOUGH_GRADIENT, dp=1,
minDist=20, param1=50, param2=30, minRadius=15, maxRadius=50)
# Verificar si se detectaron círculos
if detected_circles is not None:
    detected_circles = np.uint16(np.around(detected_circles))
    for circle in detected_circles[0, :]:
        # Dibujar el círculo en la imagen original
edges = cv2.Canny(blurred_image, 50, 150)
cv2.circle(image, (circle[0], circle[1]), circle[2], (0, 255, 0),
2)
        # Dibujar el centro del círculo
cv2.circle(image, (circle[0], circle[1]), 2, (0, 0, 255), 3)
# Mostrar la imagen con los círculos detectados
cv2.imshow('Detected Circles', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Donde los parámetros de la función `cv2.HoughCircles` significan lo siguiente:

- edges: Es la imagen que se utiliza para identificar círculos, normalmente obtenida después de aplicar un detector de bordes como el método Canny.
- cv2.HOUGH_GRADIENT: Es un enfoque que utiliza gradientes para la detección de círculos.
- dp: Es el factor de escala para la resolución del acumulador. Un valor de 1 significa que la resolución del acumulador coincide con la de la imagen original. Valores más pequeños incrementan la resolución del acumulador a costa de más tiempo de procesamiento y mayor consumo de memoria.
- minDist: Define la distancia mínima entre los centros de los círculos detectados. Un valor más alto separa más los círculos, mientras que un valor más bajo permite que los círculos estén más cerca entre sí.

- param1: Es el umbral superior utilizado por el algoritmo Canny para la detección de bordes. Un valor más elevado disminuirá la cantidad de bordes detectados.
- param2: Es el umbral del acumulador que determina si un círculo es detectado. Un valor más alto filtra los círculos con mayor rigor, mejorando la precisión, mientras que un valor más bajo puede detectar más círculos, aunque con mayor riesgo de errores.
- minRadius: Establece el radio más pequeño que pueden tener los círculos para ser detectados. Los círculos con un radio inferior a este valor no se detectarán.
- maxRadius: Establece el radio máximo de los círculos a detectar. Los círculos que superen este tamaño no serán detectados [22].

3.2.2 K-Means

Para poder reconocer elementos específicos del experimento de detección desde los datos analizados, en este caso píxeles de microgotas, es importante tener un método para agruparlos y determinar la naturaleza de cada grupo presente en el conjunto de datos. Existen varias formas de hacerlo y cada una tiene ventajas y desventajas, que se analizaron antes de decidir el método utilizado en este trabajo. *K-means* o K-medias es un algoritmo de aprendizaje de maquina (*machine learning*) no supervisado, es decir que no necesita de información previa sobre lo que hay que encontrar y que en principio nos ayuda a agrupar datos o cúmulos que se encuentran de manera natural en nuestros datos pero que en primera instancia pueden no ser tan evidentes, para agruparlos finalmente en alguno de los grupos encontrados. La K proviene de una suposición que se realiza en el algoritmo, el cual es suponer que existe un determinado número K de grupos dentro de nuestros datos, la desventaja principal de este algoritmo es que la K debe estar determinada previo al inicio de los ciclos de iteración, en el caso del programa realizado serian “Gotas positivas” y “Gotas negativas”.

Hay otras consideraciones que podrían ser una desventaja y conducir a un resultado erróneo o no concluyente, que se debe cuidar como los son el número de iteraciones por la posición inicial de los centroides, que son aleatoriamente y que se usarán para encontrar un resultado o el tipo de métrica usada. Estos puntos deberían de quedar más claros con la representación visual de este algoritmo.

Ejemplo:

En el siguiente ejemplo, se presenta un conjunto de datos, que a primera vista, parecen dividirse en tres grupos y por ende se introduce una $K = 3$ que son el número de centroides sobre los cuales al final del algoritmo deberían agruparse los datos si el algoritmo es convergente. La localización de estos centroides se hará de manera al azar, por lo que en iteraciones posteriores es probable que se ubiquen en puntos diferentes a los iniciales.

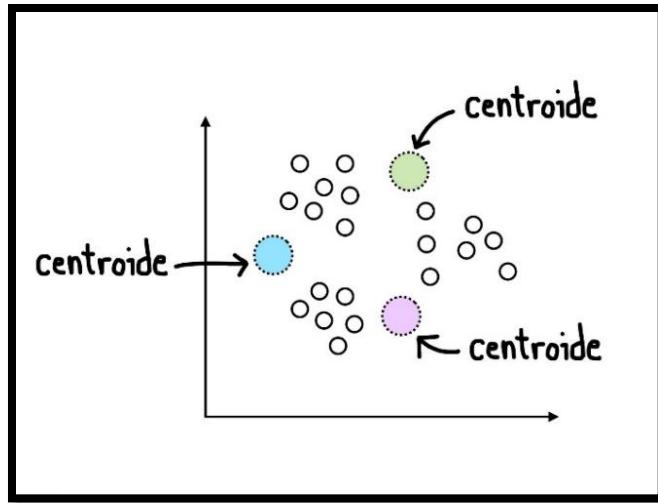


Figura 3.18 Localización de centroides al azar, donde se supone una $k = 3$

Una vez localizados los puntos se procede a calcular distancias entre cada centroide a cada uno de los puntos, para visualizar mejor los centroides y los puntos se representaron en forma de círculos, pero evidentemente tanto los centroides como los círculos que representan datos son puntos en un plano cartesiano. Otra de las cuestiones es la métrica que se usará para el siguiente paso dado, en el caso presentado se supone una métrica de tipo euclidiana. Para los puntos más cercanos al centroide azul, los pintamos de azul y lo mismo sucede para los otros centroides verde y morado como se muestra en las figuras 3.18, 3.19 y 3.20.

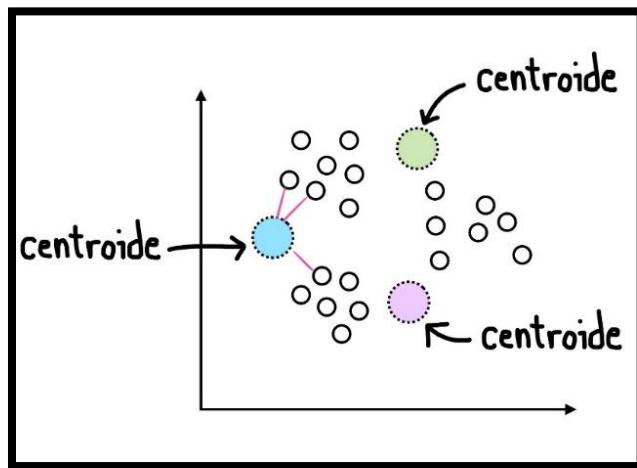


Figura 3.19 Puntos cercanos al centroide Azul son calculados.

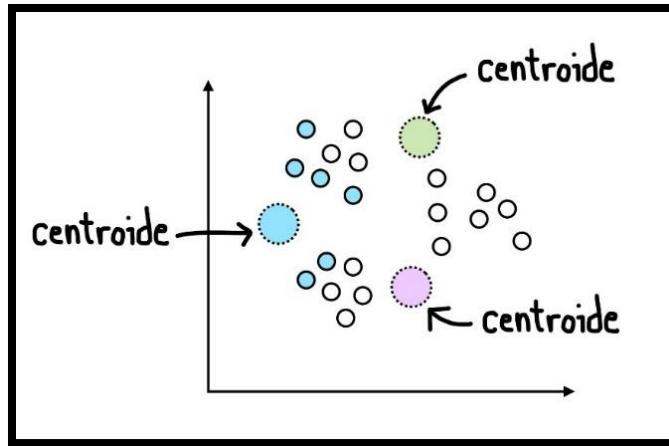


Figura 3.20 Cálculo de puntos cercanos al centroide Azul

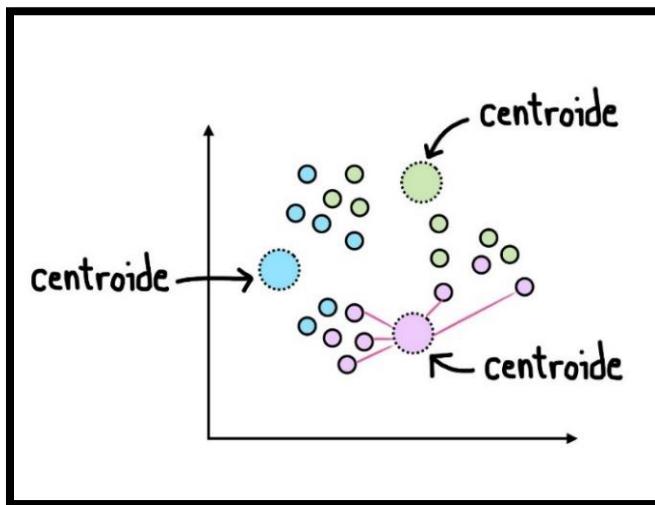


Figura 3.21 Para los puntos cercanos a los centroides morado y verde el procedimiento es igual.

Una vez que se han localizado los puntos más cercanos a los centroides, y se les ha etiquetado con un color, el paso siguiente es mover cada centroide al centro de cada grupo, que puede obtenerse como el promedio de cada grupo como nuevo centroide como se puede ver en las figuras 3.21, 3.22 y 3.23. Este proceso se repetirá varias veces hasta que la delta entre la posición anterior y la actual de los centroides sea muy pequeña y ya no representen cambios significativos convergiendo en un punto específico.

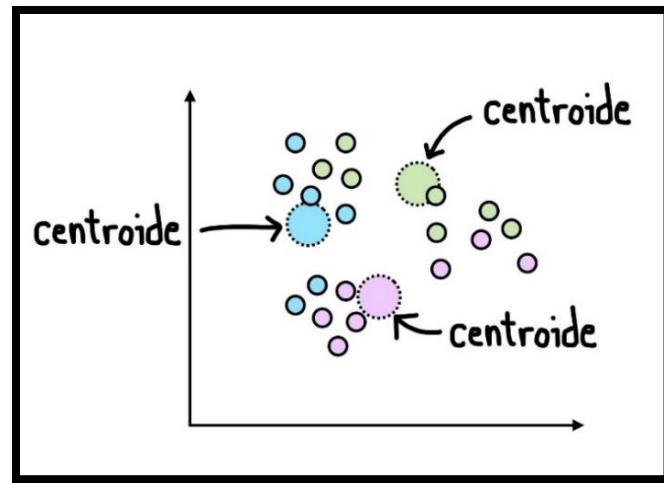


Figura 3.22 En una iteración n , se mueven los centroides a un punto que es el promedio de los elementos.

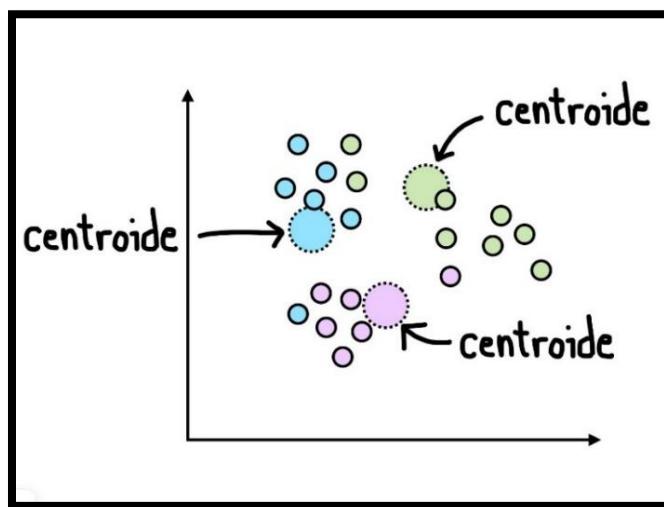


Figura 3.23 Los colores se actualizan y el centroide se mueve.

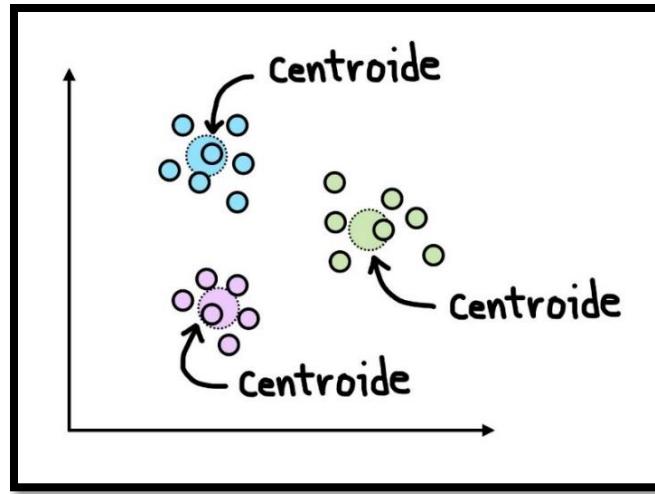


Figura 3.24 Cuando la delta de posición es mínima, consideramos terminado el algoritmo.

Una vez finalizado el algoritmo, los centroides deberían estar posicionados al centro de cada grupo que existe en los datos. Es importante recalcar nuevamente que tanto la K como el número de iteraciones afectan el resultado. Por ejemplo, puede suceder que los centroides tengan una posición aleatoria inicial muy distante de un grupo de datos, por lo cual lo más probable es que nunca converjan. Igualmente, si K fuera 2 solamente encontraríamos 2 grupos que puede que no sean significativos, así mismo si K es igual a 4 se podrían encontrar otros grupos que a primera vista no son evidentes como se muestra en la Figura 3.25. Por lo tanto, entre más alta sea la k , más grupos se podrán encontrar.

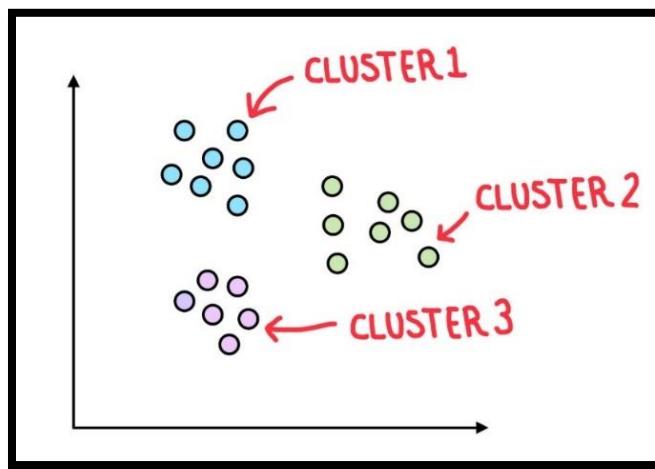


Figura 3.25 Con $k = 3$ encontramos 3 grupos.

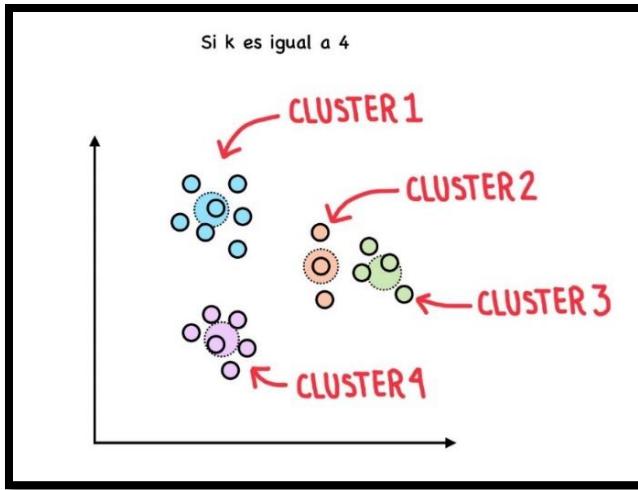


Figura 3.26. Con $k = 4$ obtenemos grupos que a primera vista no son evidentes.

3.2.3 Control Negativo

El control negativo es una prueba inicial que se lleva a cabo sin la inclusión de la variable o del objeto que se está investigando. Su objetivo es establecer una referencia o punto de comparación para que los resultados obtenidos se puedan evaluar cuando se introduce el elemento a medir. Este tipo de control es importante para asegurar que cualquier cambio observado en el experimento se deba a la variable de estudio y no a influencias externas. En el contexto de las microgotas fluorescentes, un control negativo se lleva a cabo sin incluir el patógeno que se busca detectar. Esto sirve para confirmar que no existen rastros de fluorescencia causados por la contaminación del medio o del material. Si no se detecta fluorescencia en el control negativo, se puede concluir que los rastros observados en las muestras experimentales son correctos, evitando la posibilidad de obtener resultados con falsos positivos.

4 Aplicación en Python

En este capítulo se recuerdan los objetivos planteados en la sección 1.2, el desarrollo de una interfaz gráfica cuyo objetivo es detectar microgotas de imágenes previamente ajustadas para contar, clasificar, graficar y guardar los datos de dichas microgotas. Esta aplicación permite a la persona usuaria cargar imágenes, detectar círculos que representan las microgotas, y posteriormente clasificar las microgotas como positivas o negativas de acuerdo con su rango de fluorescencia. La aplicación además incluye opciones para agregar o eliminar microgotas de forma manual y con ello tener resultados con mayor precisión. Debido a la complejidad del código su explicación se ha organizado con base a las funciones que lo integran. El código se compone esencialmente de dos partes principales: la primera es la interfaz gráfica con la que interactúa el usuario y la segunda es el algoritmo de detección de las microgotas, que contiene toda la metodología necesaria para llevar a cabo esta tarea. Cabe mencionar que no todas las funciones que se presentan están ordenadas secuencialmente, ya que algunas se ejecutan por evento, es decir, cuando la persona que interactúa con los elementos cambia los parámetros en tiempo real usando los botones o *sliders*.

4.1 Inicio

Para cualquier desarrollo en Python, primero se debe identificar cuáles son las bibliotecas básicas que se utilizarán según las necesidades del proyecto. Posteriormente, se pueden anexar o descartar bibliotecas según sea el caso. Para este trabajo, las bibliotecas que se utilizaron son las siguientes: OpenCV, NumPy y Tkinter, las cuales se han descrito en la sección 2.2.1 y 2.2.2.

A continuación, se describe otras bibliotecas también utilizadas.

Matplotlib: es una biblioteca que facilita la generación de gráficos en Python. Produce figuras con amplia variedad de formatos y entornos interactivos a través de otras bibliotecas de GUI. [23]

Imutils: consiste en un conjunto de funciones diseñadas para facilitar el procesamiento de imágenes. Estas funciones permiten realizar tareas como rotación, redimensionamiento, recorte, clasificación de contornos, detección de bordes y otras operaciones relacionadas con imágenes [24].

Pillow: es una rama de la biblioteca *Python Imaging Library* (PIL) que ofrece capacidades para abrir, editar y guardar imágenes. Es compatible con muchos tipos de formatos de imagen y cuenta con herramientas de altas capacidades para realizar diversas operaciones de procesamiento de imágenes [25].

Openpyxl: es una biblioteca de Python que permite leer y escribir archivos de Excel XML (xlsx/xlsm/xltx/xltm). Admite tablas, gráficos, dibujos, comentarios y hojas de cálculo con protección [26].

Las características anteriores se importan al espacio de trabajo de la siguiente forma:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
import tkinter as tk
from tkinter import filedialog as fd
from tkinter.messagebox import showinfo, showerror
import imutils
from PIL import Image, ImageTk
import openpyxl
from openpyxl import Workbook
```

4.2 Interfaz Gráfica

La interfaz de usuario está diseñada para facilitar la manipulación de imágenes y el análisis de microgotas. Se utiliza Tkinter para proporcionar una ventana gráfica que incluye diversos controles interactivos (también conocidos como widgets) como botones, etiquetas, y sliders. Los usuarios pueden cargar imágenes, ajustar parámetros de procesamiento como el umbral y la sensibilidad, y visualizar resultados en tiempo real. Los botones permiten realizar acciones específicas, como rotar la imagen, seleccionar hojas de datos como archivos de Excel para guardar resultados, añadir o eliminar manualmente círculos (que representan a las microgotas), entre otras opciones. La interfaz también ofrece retroalimentación mediante etiquetas que muestran el número de gotas detectadas y el umbral de tonalidad actual. Este diseño permite una interacción directa con las funciones de procesamiento de imágenes. Para una mejor comprensión, se presenta en la Figura 4.1 un diagrama que ilustra cuáles son las funciones de la interfaz gráfica. Después, se detallan con mayor profundidad.

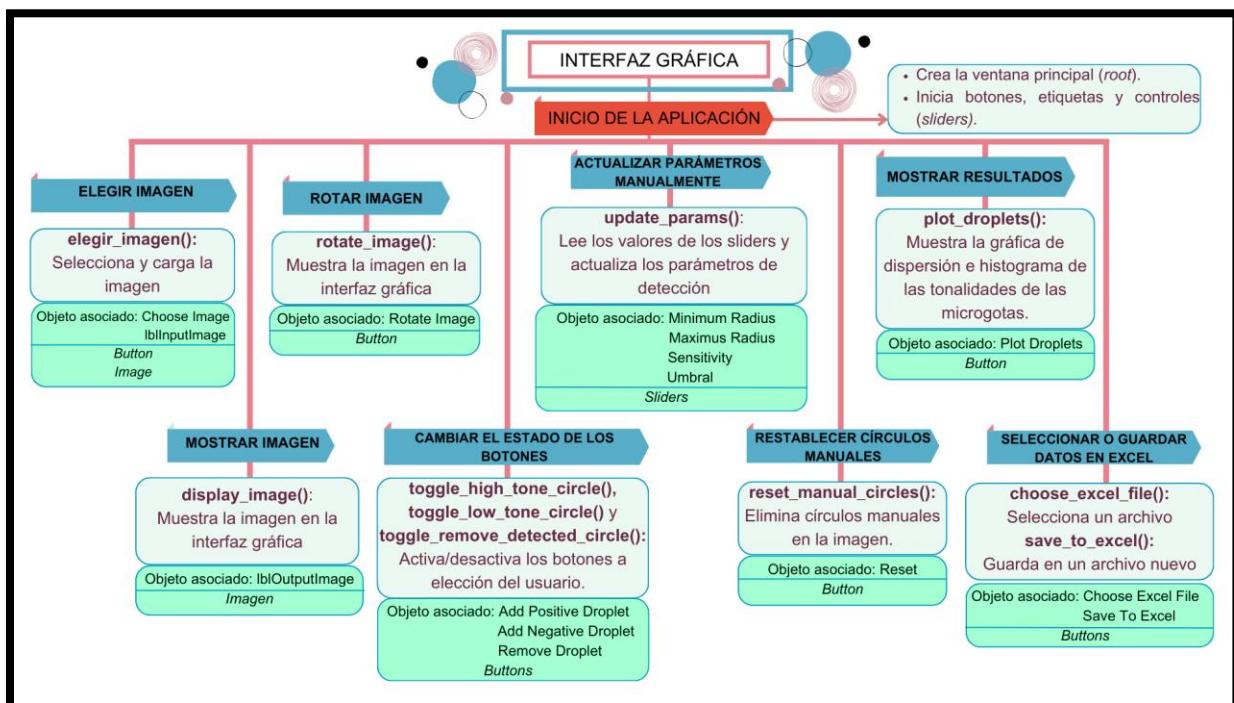


Figura 4.1 Diagrama de funciones que componen la interfaz gráfica.

A continuación, se describe cada función relacionada con la creación de la interfaz.

1. Función para elegir, importar y redimensionar la imagen

```
def elegir_imagen():
    # Abrir un cuadro de diálogo para seleccionar un archivo de imagen
    path_image = fd.askopenfilename(filetypes=[("image", ".jpg"), ("image", ".jpeg"), ("image", ".png")])
    # Verificar si se ha seleccionado una imagen (es decir, que la ruta no esté vacía)
    if Len(path_image) > 0:
        global image
    # Leer la imagen seleccionada utilizando OpenCV
    image = cv2.imread(path_image)
    # Redimensionar la imagen para que su altura sea de 600 píxeles, manteniendo la proporción
    image = imutils.resize(image, height=600)
    # Llamar a una función para mostrar la imagen en la interfaz gráfica
    display_image()
    # Redimensionar la imagen para mostrar una vista previa con un ancho de 180 píxeles, manteniendo la proporción
    imageToShow = imutils.resize(image, width=180)
    # Convertir la imagen a un formato compatible con ImageTk
    im = Image.fromarray(imageToShow)
    img = ImageTk.PhotoImage(image=im)
    # Configurar el widget de etiqueta para mostrar la imagen
    LblInputImage.configure(image=img)
    LblInputImage.image = img
    Aplicar el filtro Canny a la imagen
    apply_canny()
```

La función `elegir_imagen()` permite al usuario seleccionar un archivo de imagen mediante un cuadro de diálogo, verifica si se ha seleccionado un archivo, y luego lee la imagen utilizando OpenCV. Después, redimensiona la imagen a 600 píxeles y la muestra en la interfaz gráfica. Además, crea una vista previa redimensionada de la imagen a un ancho de 180 píxeles, la convierte a un formato compatible con `ImageTk` y actualiza un widget de etiqueta para mostrar esta vista previa. Después, aplica el filtro Canny a la imagen para la detección de bordes.

2. Función para mostrar la imagen en la interfaz

```
def display_image():
    if image is not None: # Verifica que haya una imagen cargada antes de continuar.
        # Redimensiona la imagen para que tenga un ancho de 180 píxeles.
        imageToShow = imutils.resize(image, width=180)
        # Convierte la imagen redimensionada de un array de numpy a un objeto de imagen PIL.
        im = Image.fromarray(imageToShow)
        # Convierte el objeto PIL a un objeto de imagen compatible con Tkinter.
        img = ImageTk.PhotoImage(image=im)
        # Actualiza la etiqueta de imagen en la interfaz con la nueva imagen.
        lblOutputImage.configure(image=img)
        # Necesario para evitar que la imagen sea recolectada por el recolector de basura.
        lblOutputImage.image = img
        # Aplica el filtro Canny a la imagen actual.
        apply_canny()
```

La función `display_image` muestra la imagen cargada en la interfaz de usuario. Si hay una imagen disponible, la función la redimensiona para ajustarla a un tamaño predefinido, la convierte a un formato compatible con Tkinter, y la presenta en la interfaz gráfica. Además, después de mostrar la imagen, se llama a la función `apply_canny` para aplicar el filtro Canny a la imagen actual.

3. Función para girar la imagen

```
def rotate_image():
    global image # Utiliza la variable global 'image' para acceder a la imagen cargada.
    if image is not None: # Verifica si existe una imagen cargada antes de realizar cualquier operación.
        # Rota la imagen 90 grados en el sentido de las agujas del reloj.
        image = cv2.rotate(image, cv2.ROTATE_90_CLOCKWISE)
    # Llama a la función display_image() para actualizar la imagen mostrada en la interfaz.
    display_image()
```

Esta función es responsable de rotar la imagen cargada 90 grados hacia la derecha. Permite a los usuarios ajustar la orientación de la imagen antes de realizar análisis adicionales. Una vez rotada la imagen, la función llama a otra función (`display_image`) para actualizar la imagen mostrada en la interfaz.

4. y 5. Funciones para activar o desactivar los modos de alta y baja tonalidad

```
def toggle_high_tone_circle():
    # Declaramos las variables globales que indican el modo actual de selección
    # de gotas
        global add_high_tone_circle, add_low_tone_circle
    # Establece el modo de alta tonalidad como activo
        add_high_tone_circle = True
    # Desactiva el modo de baja tonalidad
        add_low_tone_circle = False
    # Cambia el botón de alta tonalidad a hundido para mostrar que está activo
        btn_high_tone_circle.config(relief=tk.SUNKEN)
    # Cambia el botón de baja tonalidad a elevado para mostrar que está inactivo
        btn_low_tone_circle.config(relief=tk.RAISED)
    # Habilita el botón para eliminar círculos detectados, permitiendo la
    # interacción
        btn_remove_detected_circle.config(state=tk.NORMAL)

def toggle_low_tone_circle():
    # Declaramos las variables globales que indican el modo actual de selección
    # de gotas
        global add_high_tone_circle, add_low_tone_circle
    # Establece el modo de baja tonalidad como activo
        add_high_tone_circle = False
    # Desactiva el modo de alta tonalidad
        add_low_tone_circle = True
    # Cambia el botón de alta tonalidad a elevado para mostrar que está inactivo
        btn_high_tone_circle.config(relief=tk.RAISED)
    # Cambia el botón de baja tonalidad a hundido para mostrar que está activo
        btn_low_tone_circle.config(relief=tk.SUNKEN)
    # Habilita el botón para eliminar círculos detectados, permitiendo la
    # interacción
        btn_remove_detected_circle.config(state=tk.NORMAL)
```

Las funciones `toggle_high_tone_circle` y `toggle_low_tone_circle` permiten al usuario transitar entre los modos de selección de gotas de alta y baja tonalidad en la interfaz de usuario. Ambas funciones modifican las variables globales `add_high_tone_circle` y `add_low_tone_circle` para activar el modo de selección correspondiente. Al activar el modo de alta tonalidad, `toggle_high_tone_circle` establece `add_high_tone_circle` en `True` y `add_low_tone_circle` en `False`, mientras que `toggle_low_tone_circle` hace lo opuesto para activar el modo de baja tonalidad. Además, cada función modifica la apariencia de los botones en la interfaz: el botón correspondiente al modo seleccionado aparece hundido (`relief=tk.SUNKEN`) para indicar que está activo, mientras que el otro se muestra levantado (`relief=tk.RAISED`) para señalar que está inactivo. Ambas funciones también habilitan el botón de eliminación de círculos detectados (`btn_remove_detected_circle`) para que el usuario

pueda interactuar con los círculos detectados sin importar el modo activo. Esta configuración asegura que el usuario tenga un control claro y visual sobre el estado actual del sistema y pueda cambiar entre modos de manera intuitiva y efectiva.

6. Función para activar el modo de eliminación de círculos

```
def toggle_remove_detected_circle():
    global remove_detected_circle
    # Cambia el estado de la variable 'remove_detected_circle' a su opuesto
    remove_detected_circle = not remove_detected_circle
    if remove_detected_circle:
        # Si está activado, hunde el botón para indicar que la opción está activa
        btn_remove_detected_circle.config(relief=tk.SUNKEN)
        # Deshabilita los botones de alta y baja tonalidad para evitar interacciones
        # conflictivas
        btn_high_tone_circle.config(state=tk.DISABLED)
        btn_low_tone_circle.config(state=tk.DISABLED)
    else:
        # Si está desactivado, eleva el botón para indicar que la opción está
        # inactiva
        btn_remove_detected_circle.config(relief=tk.RAISED)
        # Habilita los botones de alta y baja tonalidad para que puedan ser usados
        # nuevamente
        btn_high_tone_circle.config(state=tk.NORMAL)
        btn_low_tone_circle.config(state=tk.NORMAL)
```

La función `toggle_remove_detected_circle` alterna el estado de eliminación de círculos detectados. Cuando se activa (`remove_detected_circle` es `True`), el botón de eliminación se muestra hundido, indicando su activación, y los botones para agregar círculos de alta y baja tonalidad se desactivan, impidiendo su uso simultáneo. Cuando se desactiva (`remove_detected_circle` es `False`), el botón se muestra elevado, y los botones de alta y baja tonalidad se reactivan, permitiendo al usuario volver a interactuar con ellos. Esta función gestiona visualmente la interfaz para asegurar que solo una operación esté activa a la vez, evitando errores del usuario.

7. Función para actualizar los parámetros del umbral, sensibilidad de detección, radio mínimo y máximo en tiempo real.

```
def update_params(value):
    # Declara que se utilizarán variables globales para poder modificarlas dentro
    # de la función
        global low_threshold_active, param2_active, min_radius_active,
    max_radius_active
    # Actualiza el valor de 'low_threshold_active' con el valor obtenido del
    # widget 'w'
        low_threshold_active = w.get()
    # Actualiza el valor de 'param2_active' con el valor obtenido del widget 'x'
        param2_active = x.get()
    # Actualiza el valor de 'min_radius_active' con el valor obtenido del slider
    'y_slider'
        min_radius_active = y_slider.get()
    # Actualiza el valor de 'max_radius_active' con el valor obtenido del slider
    'z_slider'
        max_radius_active = z_slider.get()
    # Llama a la función 'apply_canny()', que utiliza los valores actualizados
    # para
    # aplicar el algoritmo de detección de bordes Canny con los nuevos parámetros
    apply_canny()
```

La función `update_params(value)` actualiza varios parámetros usados para la detección de bordes mediante el algoritmo de Canny. Usa variables globales para almacenar los valores de los *sliders* de la interfaz gráfica. Primeramente, se declaran las variables globales para que puedan modificarse fuera de la función. Después, se actualizan los valores de `low_threshold_active`, `param2_active`, `min_radius_active`, y `max_radius_active` obteniendo los valores actuales de los controles `w`, `x`, `y_slider`, y `z_slider`. Estas variables representan parámetros clave para la detección de bordes, como los umbrales de detección y los radios mínimo y máximo. La función llama a `apply_canny()`, que aplica el algoritmo de Canny con los nuevos parámetros, permitiendo al usuario ver inmediatamente cómo los cambios afectan la imagen procesada. `Update_params(value)` sincroniza los controles de la interfaz con el procesamiento de la imagen, permitiendo ajustes en tiempo real de los parámetros de detección.

8. Función para remover todos los círculos manuales

```
def reset_manual_circles():
    global manual_circles_high, manual_circles_low
    # Restablece las listas que contienen los círculos manuales de alta y baja
    # tonalidad
    manual_circles_high = []
    manual_circles_low = []
    # Llama a la función 'apply_canny' para actualizar la imagen después del
    # restablecimiento
    apply_canny()
```

La función `reset_manual_circles` elimina todos los círculos que el usuario haya añadido manualmente en las listas `manual_circles_high` y `manual_circles_low`. Al limpiar estas listas, se elimina cualquier información almacenada sobre los círculos de alta y baja tonalidad que se haya añadido manualmente a la imagen. Después de restablecer estas listas, la función llama a `apply_canny()`, que se utiliza para volver a aplicar el procesamiento de imágenes de Canny y actualizar la visualización de la imagen, asegurando que los círculos eliminados ya no aparezcan en la interfaz.

9. Función para generar una gráfica de dispersión e histograma

```
def plot_droplets():
    global tonalidades_bajas, tonalidades_altas # Usa las listas globales
    para almacenar tonalidades.

    # Verifica si hay datos en las listas de tonalidades.
    if tonalidades_bajas or tonalidades_altas:
        # Crear una nueva figura para el gráfico de dispersión.
        plt.figure()
    # Crea un arreglo de índices para las tonalidades bajas.
    y_bajas = np.arange(len(tonalidades_bajas))
    # Crea un arreglo de índices para las tonalidades altas.
    y_altas = np.arange(len(tonalidades_altas))
    # Combina ambos arreglos de índices.
    y = np.concatenate((y_bajas, y_altas))
    # Combina ambas listas de tonalidades.
    x = np.concatenate((tonalidades_bajas, tonalidades_altas))
    # Crea una lista de colores, azul para bajas y rojo para altas.
    colors = ['blue'] * len(tonalidades_bajas) + ['red'] *
    len(tonalidades_altas)
    # Genera el gráfico de dispersión usando los datos combinados.
    plt.scatter(y, x, color=colors)
    plt.xlabel('Número de Gota') # Etiqueta del eje X.
    plt.ylabel('Tonalidad')      # Etiqueta del eje Y.
    plt.title('Distribución de Tonalidad de Las Gotas Detectadas') # Título del gráfico.
    plt.show() # Muestra el gráfico de dispersión.
# Genera el histograma para las tonalidades bajas.
    plt.hist(tonalidades_bajas, bins=30, color='blue', alpha=0.5,
    Label='Tonalidades Bajas')
# Genera el histograma para las tonalidades altas.
    plt.hist(tonalidades_altas, bins=30, color='red', alpha=0.5,
    Label='Tonalidades Altas')
    plt.title('Histograma de Tonalidades de Las Gotas') # Título del histograma.
    plt.xlabel('Tonalidad') # Etiqueta del eje X del histograma.
    plt.ylabel('Número de Gotas') # Etiqueta del eje Y del histograma.
    plt.legend() # Añade una leyenda para diferenciar los colores.
    plt.grid(True) # Añade una cuadrícula al gráfico para mayor claridad.
    plt.show() # Muestra el histograma.
else:
    # Muestra un mensaje informando al usuario que no hay datos disponibles.
    showinfo("Info", "No hay datos de tonalidades para mostrar.")
```

La función *plot_droplets* se encarga de visualizar las tonalidades de las gotas detectadas utilizando un gráfico de dispersión y un histograma. Primero, verifica si existen tonalidades bajas o altas; de lo contrario, muestra un mensaje informativo al usuario indicando que no hay datos para mostrar. Si hay datos disponibles, crea un gráfico de dispersión combinando las tonalidades bajas y altas, asignando el color azul a las gotas con tonalidades bajas y el color rojo a las gotas con tonalidades altas. El gráfico muestra la distribución de las tonalidades a lo largo de las gotas detectadas. Luego, genera un histograma que presenta la frecuencia de las tonalidades bajas y altas, utilizando transparencias para facilitar la comparación visual entre las dos distribuciones. El histograma y el gráfico de dispersión incluyen etiquetas, una leyenda, y una cuadrícula para mejorar la comprensión visual, y ambos gráficos se muestran al usuario para que pueda analizar la distribución y frecuencia de las tonalidades de las gotas.

10. Función para elegir un archivo de Excel

```
def choose_excel_file():
    global file_path
    # Abre un cuadro de diálogo para que el usuario seleccione un archivo Excel (.xlsx).
    path = fd.askopenfilename(filetypes=[("Excel files", ".xlsx")])
    # Si el usuario selecciona un archivo, se guarda la ruta del archivo en la variable global file_path.
    if path:
        file_path = path
    # Muestra un mensaje informativo con la ruta del archivo seleccionado.
        showinfo("Archivo seleccionado", f"Archivo seleccionado: {file_path}")
```

Esta función *choose_excel_file* permite al usuario seleccionar un archivo Excel mediante un cuadro de diálogo de selección de archivos. Al abrir el cuadro de diálogo, el usuario puede navegar por el sistema de archivos y elegir un archivo Excel existente. Si el usuario selecciona un archivo, la ruta del archivo se guarda en la variable global *file_path*. Después, la función muestra un mensaje informativo que indica al usuario el archivo seleccionado con éxito. Esta función es útil para abrir archivos Excel previamente guardados y para que el programa trabaje con los datos contenidos en ellos.

11. Función para guardar los parámetros de las microgotas en Excel

```
def save_to_excel():
    global file_path
# Si file_path es None, significa que el usuario aún no ha seleccionado un archivo
# para guardar.
    if file_path is None:
# Abre un cuadro de diálogo para pedir al usuario que seleccione un nombre de
# archivo para guardar.
        file_path = fd.asksaveasfilename(defaultextension=".xlsx",
filetypes=[("Excel files", ".xlsx")])
# Si el usuario cancela la selección de archivo, se termina la ejecución de la
# función.
    if not file_path:
        return
# Crea un nuevo libro de Excel utilizando la clase Workbook.
    workbook = Workbook()
# Obtiene la hoja activa del libro de Excel.
    sheet = workbook.active
# Añade una fila de encabezado a la hoja de cálculo.
    sheet.append(["Total de microgotas", "Microgotas Positivas", "Microgotas
Negativas", "Umbral de Tonalidad"])
else:
# Intenta abrir el archivo existente especificado en file_path.
    try:
        workbook = openpyxl.load_workbook(file_path)
        sheet = workbook.active
# Muestra un mensaje de error si ocurre un problema al abrir el archivo.
    except Exception as e:
        showerror("Error", f"Error al abrir el archivo: {e}")
        return
# Calcula el total de gotas y las clasifica en positivas y negativas.
    total_droplets = len(tonalidades_bajas) + len(tonalidades_altas)
    positive_droplets = len(tonalidades_altas)
    negative_droplets = len(tonalidades_bajas)
# Añade una fila de datos con el total de gotas, las gotas positivas, negativas y
el umbral de tonalidad.
    sheet.append([total_droplets, positive_droplets, negative_droplets,
umbral_tonalidad])
# Intenta guardar el libro de Excel en el archivo especificado en file_path.
    try:
        workbook.save(file_path)
# Muestra un mensaje de confirmación indicando que los datos se han guardado con
éxito.
        showinfo("Información guardada", "Los datos han sido guardados
exitosamente.")
# Muestra un mensaje de error si ocurre un problema
    except PermissionError:
        showerror("Error", "No se pudo guardar el archivo.")
```

La función `save_to_excel` permite al usuario guardar los datos de las microgotas detectadas en un archivo de Excel. Si el usuario no ha seleccionado un archivo previamente (`file_path` es `None`), la función abre un cuadro de diálogo para que el usuario elija un nombre y ubicación para el nuevo archivo Excel. Luego, crea un nuevo libro de Excel con encabezados para los datos de las microgotas. Si el archivo ya ha sido seleccionado, la función intenta abrirlo y acceder a la hoja de cálculo activa. La función calcula el total de microgotas, así como la cantidad de microgotas positivas y negativas, y agrega estos datos junto con el umbral de tonalidad a la hoja de cálculo.

Configuración de la Interfaz Gráfica de Usuario

Una vez detalladas las funciones individuales de la lógica de la aplicación, se explica la configuración de la interfaz gráfica. Esta sección del código establece y organiza todos los elementos visuales que el usuario emplea para interactuar con la aplicación. A continuación, se describen los principales componentes:

```
# Inicializa variables globales
image = None # Variable para almacenar la imagen cargada
detected_circles = None # Variable para almacenar los círculos detectados en
la imagen
tonalidades_bajas = [] # Lista para almacenar tonalidades bajas
tonalidades_altas = [] # Lista para almacenar tonalidades altas
file_path = None # Variable para almacenar la ruta del archivo
# Crea la ventana principal de la aplicación
root = tk.Tk()
# Crea una etiqueta para mostrar la imagen de entrada
LblInputImage = tk.Label(root)
LblInputImage.grid(column=0, row=2)
# Crea una etiqueta para mostrar la imagen de salida, con capacidad para manejar
eventos de clic
LblOutputImage = tk.Label(root)
LblOutputImage.grid(column=1, row=1, rowspan=12)
LblOutputImage.bind("<Button-1>", add_or_remove_circle) # Asocia clic
izquierdo con la función add_or_remove_circle
LblOutputImage.bind("<Button-3>", add_or_remove_circle) # Asocia clic derecho
con la función add_or_remove_circle
# Botón para rotar la imagen
btn_rotate = tk.Button(root, text="Rotate Image", command=rotate_image)
btn_rotate.grid(column=1, row=0)
# Botón para elegir una imagen
btn = tk.Button(root, text="Choose image", width=25, command=elegir_imagen)
btn.grid(column=0, row=0, padx=5, pady=5)
# Botón para controlar imágenes negativas
btn_control = tk.Button(root, text="Negative Control", width=25,
command=elegir_imagen_control)
```

Continúa...

```

btn_control.grid(column=0, row=1, padx=5, pady=5)
# Escala para ajustar el umbral, con un rango de 0 a 254
w = tk.Scale(root, from_=0, to=254, resolution=1, orient=tk.HORIZONTAL,
Label="Umbral", command=update_params)
w.set(low_threshold_active) # Ajusta el valor inicial de la escala
w.grid(column=0, row=3, padx=5, pady=5)
# Escala para ajustar la sensibilidad, con un rango de 1 a 30
x = tk.Scale(root, from_=1, to=30, resolution=1, orient=tk.HORIZONTAL,
Label="Sensitivity", command=update_params)
x.set(param2_active) # Ajusta el valor inicial de la escala
x.grid(column=0, row=4, padx=5, pady=5)
# Escala para ajustar el radio mínimo, con un rango de 0 a 100
y_slider = tk.Scale(root, from_=0, to=100, resolution=1,
orient=tk.HORIZONTAL, Label="Minimum Radius", command=update_params)
y_slider.set(min_radius_active) # Ajusta el valor inicial del slider
y_slider.grid(column=2, row=1, padx=5, pady=5)
# Escala para ajustar el radio máximo, con un rango de 0 a 200
z_slider = tk.Scale(root, from_=0, to=200, resolution=1,
orient=tk.HORIZONTAL, Label="Maximus Radius", command=update_params)
z_slider.set(max_radius_active) # Ajusta el valor inicial del slider
z_slider.grid(column=2, row=2, padx=5, pady=5)
# Botón para configurar todos los círculos
btn_all = tk.Button(root, text="All Droplets", command=set_all_droplets)
btn_all.grid(column=0, row=5, padx=5, pady=5)
# Botón para configurar gotas positivas
btn_positive = tk.Button(root, text="Positive Droplets",
command=set_positive_droplets)
btn_positive.grid(column=0, row=6, padx=5, pady=5)
# Botón para graficar las gotas
btn_plot = tk.Button(root, text="Plot Droplets", width=25,
command=plot_droplets)
btn_plot.grid(column=0, row=7, padx=5, pady=5)
# Botón para guardar los datos en un archivo Excel
btn_save = tk.Button(root, text="Save to Excel", command=save_to_excel)
btn_save.grid(column=2, row=8, padx=5, pady=5)
# Botón para elegir un archivo Excel
btn_choose_excel = tk.Button(root, text="Choose Excel File",
command=choose_excel_file)
btn_choose_excel.grid(column=2, row=9, padx=5, pady=5)
# Etiqueta para mostrar la cantidad de gotas negativas
lblTonalidadesBajas = tk.Label(root, text="Negative Droplets: 0")
lblTonalidadesBajas.grid(column=2, row=3, padx=5, pady=5)
# Etiqueta para mostrar la cantidad de gotas positivas
lblTonalidadesAltas = tk.Label(root, text="Positive Droplets: 0")
lblTonalidadesAltas.grid(column=2, row=4, padx=5, pady=5)

```

Continúa...

```

# Etiqueta para mostrar el umbral de tonalidad
lblUmbralTonalidad = tk.Label(root, text="Umbral de tonalidad: 0.00")
lblUmbralTonalidad.grid(column=2, row=10, padx=5, pady=5)
# Botón para agregar un círculo positivo manualmente
btn_high_tone_circle = tk.Button(root, text="Add Positive DropLet",
command=toggle_high_tone_circle)
btn_high_tone_circle.grid(column=2, row=5, padx=5, pady=5)
# Botón para agregar un círculo negativo manualmente
btn_low_tone_circle = tk.Button(root, text="Add Negative DropLet",
command=toggle_low_tone_circle)
btn_low_tone_circle.grid(column=2, row=6, padx=5, pady=5)
# Botón para reiniciar los círculos manuales
btn_reset = tk.Button(root, text="Reset", command=reset_manual_circles)
btn_reset.grid(column=2, row=7, padx=5, pady=5, sticky="se")
# Botón para eliminar círculos detectados
btn_remove_detected_circle = tk.Button(root, text="Remove DropLet",
command=toggle_remove_detected_circle)
btn_remove_detected_circle.grid(column=0, row=8, padx=5, pady=5)
# Inicia el bucle principal de la GUI
root.mainloop()

```

El objetivo de esta sección del código es posicionar cada botón y elemento de la interfaz dentro del panel principal. Los componentes se organizan en una cuadrícula, asignándoles ubicaciones específicas en la ventana. Se crean etiquetas para mostrar la imagen de entrada y la imagen de salida, esta última con capacidad para manejar eventos de clic. Se añaden botones con diferentes funciones, como rotar la imagen, elegir una imagen nueva, y controlar imágenes negativas. Se implementan deslizadores para ajustar parámetros críticos de detección, como el umbral, la sensibilidad y los radios mínimo y máximo de las gotas detectadas. Además, se incluyen botones para configuraciones específicas, como configurar todas las gotas detectadas, especificar gotas positivas, graficar las gotas, y guardar o seleccionar archivos Excel. También se agregan etiquetas informativas para mostrar la cantidad de gotas negativas y positivas detectadas, así como el umbral de tonalidad actual. Hay botones que permiten agregar manualmente gotas positivas y negativas, reiniciar los ajustes manuales y eliminar círculos detectados. Esto asegura una interfaz ordenada y accesible, permitiendo al usuario interactuar con la aplicación y realizar ajustes en la detección de gotas.

4.3 Detección de microgotas

En esta parte, se describen las funciones ligadas al algoritmo de detección de microgotas. En la Figura 4.2, se muestra el diagrama con cada función involucrada en esta tarea.

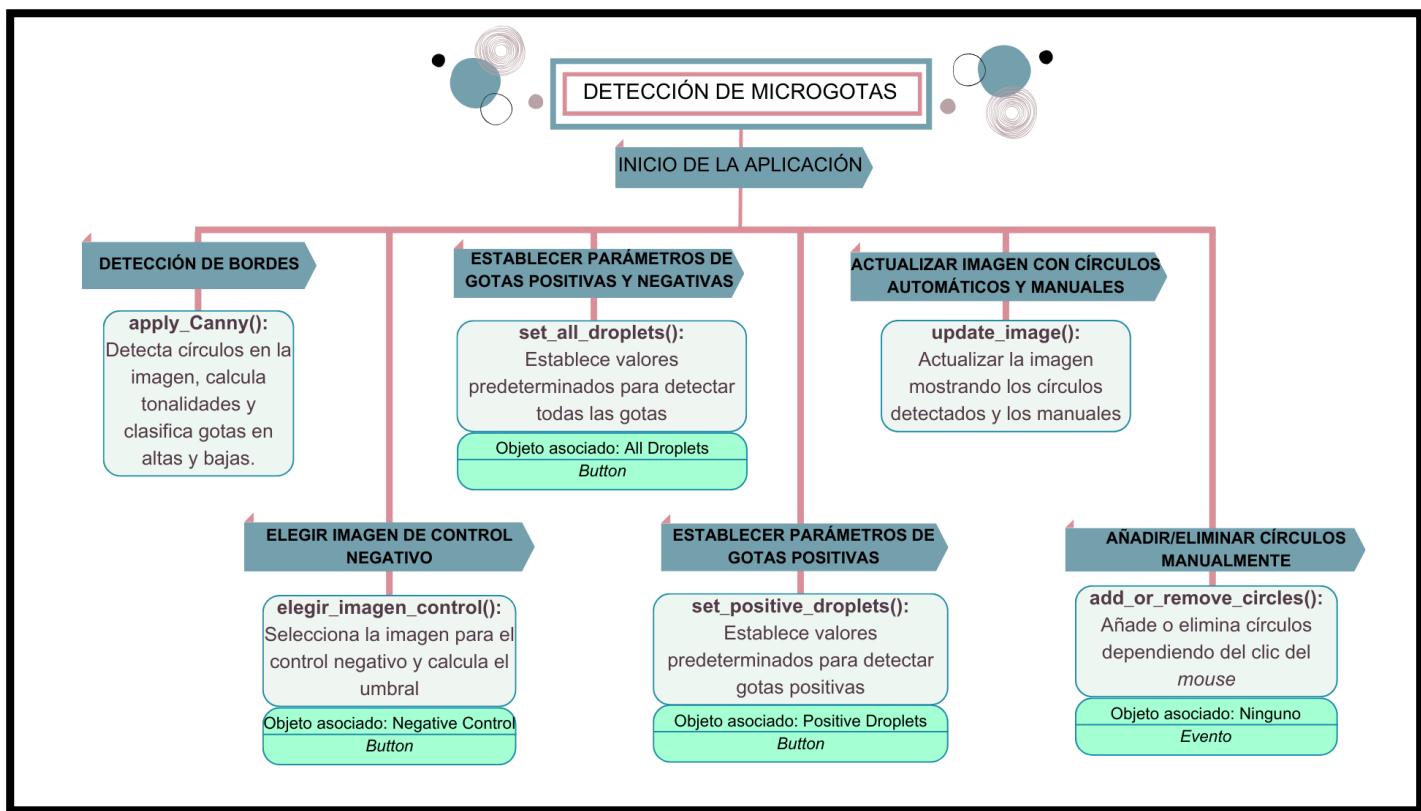


Figura 4.2 Diagrama de funciones que componen la detección de microgotas.

Para la detección de microgotas debe procesarse la imagen (como se vio anteriormente) de acuerdo con los pasos del diagrama de la Figura 3.1. A continuación, se explican detalladamente las funciones que forman parte del diagrama de la Figura 4.2.

1- Función para agregar filtro, suavizar, detectar contornos, erosionar, dilatar y realizar el conteo de microgotas.

```
def apply_canny():
    global tonalidades
    global umbral_tonalidad
    global tonalidades_bajas, tonalidades_altas
    global detected_circles
    if image is not None:
        b, g, r = cv2.split(image)
        alpha = 1.9
        beta = 0
        enhanced_green = cv2.convertScaleAbs(g, alpha=alpha, beta=beta)
        # Suavizar la imagen para reducir el ruido
        blurred = cv2.GaussianBlur(enhanced_green, (9, 9), 2)
        # Aplicar la detección de bordes de Canny
        edges      = cv2.Canny(blurred, low_threshold_active,
        Low_threshold_active*2)
        # Mejorar los bordes mediante dilatación y erosión
        edges = cv2.dilate(edges, None, iterations=2)
        edges = cv2.erode(edges, None, iterations=1)

        # (Continúa con la detección de círculos...)
```

En la función nombrada *apply_canny* la imagen ya está disponible dentro de la variable global llamada *imagen*. Es importante mencionar que dentro de esta función se encuentran diversas acciones que se utilizan para el resto del código, pero en esta parte solo se describe lo relacionado al ajuste de la imagen. La función procesa la imagen dividiéndola en sus canales de color y mejorando el canal verde mediante un ajuste de contraste. Luego, suaviza la imagen utilizando un filtro Gaussiano para reducir el ruido antes de aplicar la detección de bordes de Canny. Los bordes detectados se mejoran mediante dilatación y erosión para destacarlos mejor. Con esta función se prepara para continuar con la detección de círculos. Una vez que la imagen se importó y se ajustó correctamente para la detección de las microgotas, se comienza a hacer el conteo individual dentro de la misma función (*apply_canny*).

```

detected_circles = cv2.HoughCircles(edges, cv2.HOUGH_GRADIENT, 1, 20,
param1=50,param2=int(param2_active),minRadius=int(min_radius_active),maxRadius=int(max_radius_active))
# Lista para almacenar las tonalidades de los círculos detectados.
tonalidades = []
# Copia de la imagen original para dibujar los resultados.
result_image = image.copy()
# Lista para almacenar todos los círculos detectados.
all_circles = []
# Si se detectaron círculos, los procesa.
if detected_circles is not None:
# Redondea los valores de los círculos detectados y los convierte a enteros
de 16 bits.
    detected_circles = np.uint16(np.around(detectected_circles))
# Itera sobre cada círculo detectado y lo agrega a la lista 'all_circles'.
    for pt in detected_circles[0, :]:
        all_circles.append((pt[0], pt[1], pt[2])) # pt[0]: centro x, pt[1]:
centro y, pt[2]: radio
# Para cada círculo en la lista 'all_circles', se calcula la tonalidad.
    for (a, b, r) in all_circles:
# Calcula los límites de la región del círculo en la imagen, asegurándose de
no salir de los bordes de la imagen.
        y1, y2 = max(int(b) - int(r), 0), min(int(b) + int(r), image.shape[0])
        x1, x2 = max(int(a) - int(r), 0), min(int(a) + int(r), image.shape[1])
# Si los límites son válidos (y2 es mayor que y1 y x2 es mayor que x1),
procede.
        if y2 > y1 and x2 > x1:
# Extrae la subimagen correspondiente al área del círculo.
            sub_image = image[y1:y2, x1:x2]
# Si la subimagen no está vacía, calcula la tonalidad promedio.
            if sub_image.size > 0:
                tonalidad = np.mean(sub_image)
# Si la tonalidad calculada no es NaN, la agrega a la lista 'tonalidades'.
                if not np.isnan(tonalidad):
                    tonalidades.append(tonalidad)

```

Después de aplicar el filtro de Canny para la detección de bordes, se emplea la transformación de Hough para detectar círculos en la imagen. Utilizando `cv2.HoughCircles()`, la función busca círculos en la imagen de bordes (`edges`). Los parámetros de esta función incluyen el método de detección, la escala mínima de la imagen, y umbrales para la detección de bordes y el radio de los círculos. Si se detectan círculos, estos se almacenan como coordenadas (x, y, r) en la lista `detected_circles`. Para cada círculo detectado, la función calcula el área rectangular que rodea el círculo en la imagen original. Esta área se extrae como una sub-imagen (`sub_image`). En cada sub-imagen, se calcula la tonalidad promedio de los píxeles utilizando `np.mean(sub_image)`. Este valor de tonalidad se añade a la lista global `tonalidades`, siempre y cuando no sea un valor nulo.

Este proceso asegura que solo se consideren tonalidades válidas y permite analizar las características de los círculos detectados en la imagen original.

El resto del código incluye funciones necesarias para aumentar la precisión del conteo de microgotas positivas, negativas y totales. Incluyendo las herramientas que permiten al usuario agregar o eliminar círculos de forma manual. A continuación, se detallan cada una de estas funciones, comenzando con una descripción completa de la función `apply_canny`, de la cual ya se ha revisado parcialmente.

```
for (x, y, r) in manual_circles_high + manual_circles_low:  
    # Calcula los límites de la región del círculo en la imagen, asegurándose de  
    # no salir de los bordes de la imagen.  
    y1, y2 = max(int(y) - int(r), 0), min(int(y) + int(r), image.shape[0])  
    x1, x2 = max(int(x) - int(r), 0), min(int(x) + int(r), image.shape[1])  
    # Si los límites son válidos (y2 es mayor que y1 y x2 es mayor que x1),  
    # procede.  
    if y2 > y1 and x2 > x1:  
        # Extrae la subimagen correspondiente al área del círculo.  
        sub_image = image[y1:y2, x1:x2]  
        # Si la subimagen no está vacía, calcula la tonalidad promedio.  
        if sub_image.size > 0:  
            tonalidad = np.mean(sub_image)  
        # Si la tonalidad calculada no es NaN, la agrega a la lista 'tonalidades'.  
        if not np.isnan(tonalidad):  
            tonalidades.append(tonalidad)
```

En esta sección del código, se calculan y se agregan las tonalidades para círculos manuales definidos por el usuario. Primero, se itera sobre las listas de círculos manuales `manual_circles_high` y `manual_circles_low`, que contienen las coordenadas (x, y, r) de los círculos. Para cada círculo, se determinan los límites del área rectangular que rodea el círculo en la imagen original, garantizando que las coordenadas no excedan los límites de la imagen. Esto se hace para descartar gotas incompletas principalmente cerca de los bordes.

Se extrae la sub-imagen correspondiente a esta área rectangular y, si la sub-imagen no está vacía, se calcula la tonalidad promedio de los píxeles dentro de ella. Este valor de tonalidad se agrega a la lista global `tonalidades`, siempre que sea un valor válido y no `Nan`. Este proceso permite capturar y analizar las tonalidades en las regiones de interés especificadas manualmente en la imagen.

```

# Utilizar el umbral del control negativo
# Si hay un umbral de control proporcionado
if umbral_control is not None:
    # Ajustar el umbral utilizando la desviación estándar del control
    umbral_tonalidad = umbral_control + 3 * desviacion_estandar_control
# Si no hay un umbral de control pero hay tonalidades calculadas
elif Len(tonalidades) > 0:
    # Si hay al menos 2 tonalidades, aplicar K-means para encontrar el umbral
    if Len(tonalidades) >= 2:
        # Convertir la lista de tonalidades a un arreglo de numpy y darle la forma
        # adecuada para K-means
        tonalidades = np.array(tonalidades).reshape(-1, 1).astype(np.float32)
    # Definir los criterios de terminación para el algoritmo K-means
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100,
    0.2)
    # Número de clusters
    k = 2
    # Aplicar K-means
    _, labels, centers = cv2.kmeans(tonalidades, k, None, criteria, 10,
    cv2.KMEANS_RANDOM_CENTERS)
    # Ordenar los centros y calcular el umbral como el promedio de los centros
    centers = sorted(centers.flatten())
    umbral_tonalidad = sum(centers) / 2
    # Si hay menos de 2 tonalidades, calcular el umbral como la media de las
    tonalidades más 10
    else:
        umbral_tonalidad = np.mean(tonalidades) + 10
# Si no hay un umbral de control ni tonalidades calculadas, establecer el
umbral a 0
else:
    umbral_tonalidad = 0

```

En esta parte del código se calcula el umbral de tonalidad (*umbral_tonalidad*) en función de los valores obtenidos y de si se ha definido un umbral de control negativo (*umbral_control*). Si *umbral_control* está disponible, se ajusta el umbral sumando tres veces la desviación estándar del control al valor del umbral de control. Si *umbral_control* no está definido y hay tonalidades disponibles, se evalúa el número de tonalidades. Si hay al menos dos tonalidades, se utiliza el algoritmo K-means para agrupar las tonalidades en dos clusters y determinar el umbral como el promedio de los centros de estos *clusters*. Si solo hay una tonalidad o ninguna, se usa el promedio de las tonalidades más un valor adicional de 10 como umbral alternativo. Si no hay tonalidades disponibles, el umbral se establece en 0. Este enfoque asegura que el umbral de tonalidad sea calculado de manera adecuada según la cantidad de datos disponibles.

```

# Convertir la lista de tonalidades a un arreglo de numpy y aplanar el arreglo
tonalidades = np.array(tonalidades).flatten()

# Separar las tonalidades en dos categorías según el umbral de tonalidad
tonalidades_bajas = [tono for tono in tonalidades if tono <= umbral_tonalidad] # Tonalidades por debajo o iguales al umbral
tonalidades_altas = [tono for tono in tonalidades if tono > umbral_tonalidad] # Tonalidades por encima del umbral

# Dibujar círculos detectados automáticamente en la imagen de resultado
for (a, b, r) in all_circles:
    # Calcular la tonalidad promedio dentro del área del círculo en la imagen original
    tonalidad = np.mean(image[max(int(b) - int(r), 0):min(int(b) + int(r), image.shape[0]), max(int(a) - int(r), 0):min(int(a) + int(r), image.shape[1])])

    # Dibujar el círculo en la imagen de resultado, con color basado en la tonalidad
    if tonalidad > umbral_tonalidad:
        cv2.circle(result_image, (a, b), r, (255, 0, 0), 2) # Rojo para círculos con tonalidad alta
    else:
        cv2.circle(result_image, (a, b), r, (0, 0, 255), 2) # Azul para círculos con tonalidad baja
    # Dibujar círculos manuales con tonalidades altas en la imagen de resultado
    for (x, y, r) in manual_circles_high:
        cv2.circle(result_image, (x, y), r, (255, 0, 0), 2) # Rojo para círculos manuales de tonalidad alta
    # Dibujar círculos manuales con tonalidades bajas en la imagen de resultado
    for (x, y, r) in manual_circles_low:
        cv2.circle(result_image, (x, y), r, (0, 0, 255), 2) # Azul para círculos manuales de tonalidad baja

```

En esta parte se realiza el procesamiento de las tonalidades extraídas de los círculos detectados y manuales en la imagen. Primero, la lista de tonalidades se convierte en un arreglo de numpy y se aplana para facilitar el manejo de datos. Luego, se clasifica cada tonalidad en dos categorías: *tonalidades_bajas*, para tonalidades menores o iguales a un umbral definido, y *tonalidades_altas*, para tonalidades mayores que el umbral. Despues, se dibujan los círculos detectados automáticamente en la imagen de resultado (*result_image*). Para cada círculo, se calcula la tonalidad promedio en su región correspondiente. Los círculos se dibujan con color rojo si su tonalidad es alta (por encima del umbral) y con color azul si es baja (por debajo o igual al umbral). Finalmente, se dibujan los círculos manuales en la imagen de resultado, diferenciando entre tonalidades altas y bajas con los mismos colores: rojo para tonalidades altas y azul para bajas. Este proceso visualiza claramente los círculos según sus tonalidades en la imagen final.

```

# Actualizar etiquetas en la interfaz gráfica para mostrar el número de gotas
negativas y positivas
lblTonalidadesBajas.config(text=f"Negativedroplets:
Len(tonalidades_bajas)}")
lblTonalidadesAltas.config(text=f"Positivedroplets:
Len(tonalidades_altas)}")

#Añadir un texto en la imagen de resultado con el total de microgotas
detectadas
total_circles = Len(tonalidades_bajas) + Len(tonalidades_altas) # Calcular
el número total de microgotas
text = f'Total de microgotas: {total_circles}' # Texto que se añadirá a la
imagen
font_scale = 0.5 # Escala de la fuente para el texto
thickness = 1 # Grosor del texto
color = (255, 255, 255) # Color del texto (blanco)
font = cv2.FONT_HERSHEY_SIMPLEX # Tipo de fuente
# Obtener el tamaño del texto para posicionarlo adecuadamente en la imagen
text_size, _ = cv2.getTextSize(text, font, font_scale, thickness)
text_w, text_h = text_size
pos_x = result_image.shape[1] - text_w - 10 # Posición X del texto (margen
derecho)
pos_y = result_image.shape[0] - 10 # Posición Y del texto (margen inferior)

# Añadir el texto a la imagen en la posición calculada
cv2.putText(result_image, text, (pos_x, pos_y), font, font_scale, color,
thickness)
# Convertir la imagen de resultado a un formato compatible con Tkinter para
su visualización
im = Image.fromarray(result_image)
img = ImageTk.PhotoImage(image=im)
lblOutputImage.configure(image=img) # Actualizar el widget de la imagen con
la imagen modificada
lblOutputImage.image = img # Guardar la referencia de la imagen en el widget
# Actualizar la etiqueta del umbral de tonalidad en la interfaz gráfica
lblUmbralTonalidad.config(text=f"Umbral de tonalidad:
{umbral_tonalidad:.2f}")
# Imprimir en la consola información sobre el umbral de tonalidad y las
cantidades de gotas
print("EL umbral de la tonalidad está en ", umbral_tonalidad)
print("El número total de microgotas es: ", total_circles)
print("Tonalidades bajas: ", Len(tonalidades_bajas))
print("Tonalidades altas: ", Len(tonalidades_altas))

```

En la última parte de la función `apply_canny` se actualizan varios elementos de la interfaz gráfica y se muestran estadísticas sobre los resultados del procesamiento de imágenes. Primero, se actualizan las etiquetas `lblTonalidadesBajas` y `lblTonalidadesAltas` para mostrar el número de gotas negativas y positivas detectadas, respectivamente, utilizando las listas `tonalidades_bajas` y `tonalidades_altas`. A continuación, se añade un texto a la imagen de resultado (`result_image`)

que indica el total de microgotas detectadas. Para esto, se calcula la posición del texto en la esquina inferior derecha de la imagen, se define el estilo de fuente y color, y se utiliza `cv2.putText()` para dibujar el texto en la imagen. El resultado se convierte en un formato compatible con Tkinter (`ImageTk.PhotoImage`) y se actualiza el widget `lblOutputImage` para mostrar la imagen modificada. Se actualiza la etiqueta `lblUmbralTonalidad` para mostrar el umbral de tonalidad actual y se imprimen en la consola estadísticas sobre el umbral de tonalidad y el número total de microgotas, así como la cantidad de tonalidades bajas y altas. Esto proporciona una visión clara de los resultados del análisis y facilita la interpretación de los datos.

2- Función para seleccionar la imagen de control negativo

```
def elegir_imagen_control():
    # Abre un cuadro de diálogo para seleccionar un archivo de imagen.
    # Se filtran los archivos para que solo se puedan elegir imágenes con
    # extensiones .jpg, .jpeg, .png.
    path_image = fd.askopenfilename(filetypes=[("image", ".jpg"), ("image",
    ".jpeg"), ("image", ".png")])

    # Verifica que se haya seleccionado una imagen.
    if Len(path_image) > 0:
        # Usa la ruta de la imagen seleccionada para leerla con OpenCV.
        global negative_control_image, negative_control_tonalidades,
        umbral_control, desviacion_estandar_control
        negative_control_image = cv2.imread(path_image)
        # Redimensiona la imagen a una altura de 600 píxeles manteniendo la relación
        # de aspecto.
        negative_control_image=imutils.resize(negative_control_image,
        height=600)
        # Calcula tonalidades del control negativo.
        # Separa los canales de color de la imagen.
        b, g, r = cv2.split(negative_control_image)
        alpha = 1.9 # Factor de escala para ajustar el contraste.
        beta = 0     # Valor de desplazamiento para ajustar el brillo.
        # Mejora el canal verde de la imagen usando la conversión de escala.
        enhanced_green = cv2.convertScaleAbs(g, alpha=alpha, beta=beta)
        # Aplica un desenfoque gaussiano para reducir el ruido.
        blurred = cv2.GaussianBlur(enhanced_green, (9, 9), 2)
        # Detecta bordes en la imagen usando el algoritmo de Canny.
        edges = cv2.Canny(blurred, low_threshold_all, low_threshold_all * 2)
```

Continúa...

```

# Realiza una dilatación para ampliar los bordes detectados.
edges = cv2.dilate(edges, None, iterations=2)
# Realiza una erosión para reducir el tamaño de los bordes detectados.
edges = cv2.erode(edges, None, iterations=1)
# Detecta círculos en la imagen usando el algoritmo de Hough.
detected_circles = cv2.HoughCircles(edges, cv2.HOUGH_GRADIENT, 1, 20,
param1=50, param2=int(param2_all), minRadius=int(min_radius_active), maxRadius
= int(max_radius_active))
# Inicializa la lista para almacenar las tonalidades de las gotas detectadas.
negative_control_tonalidades = []
if detected_circles is not None:
# Redondea las coordenadas de los círculos detectados.
detected_circles = np.uint16(np.around(detected_circles))
# Recorre cada círculo detectado.
for pt in detected_circles[0, :]:
    a, b, r = pt[0], pt[1], pt[2]
# Define los límites de la subimagen basada en la posición y el radio del
círculo.
    y1, y2 = max(int(b) - int(r), 0), min(int(b) + int(r),
negative_control_image.shape[0])
    x1, x2 = max(int(a) - int(r), 0), min(int(a) + int(r),
negative_control_image.shape[1])
    if y2 > y1 and x2 > x1:
# Extrae la subimagen dentro del círculo.
    sub_image = negative_control_image[y1:y2, x1:x2]
    if sub_image.size > 0:
# Calcula la tonalidad promedio de la subimagen.
    tonalidad = np.mean(sub_image)
    if not np.isnan(tonalidad):
# Agrega la tonalidad a la lista si no es NaN.
    negative_control_tonalidades.append(tonalidad)
# Si se detectaron tonalidades, calcula el umbral y la desviación estándar.
if len(negative_control_tonalidades) > 0:
    umbral_control = np.mean(negative_control_tonalidades)
    desviacion_estandar_control = np.std(negative_control_tonalidades)
# Muestra un mensaje de éxito con el número de tonalidades detectadas.
showinfo("Control Negativo", f"Control negativo cargado con {len(negative_control_tonalidades)} tonalidades.")
else:
# Si no se detectaron tonalidades, asigna None a los parámetros y muestra un
mensaje de error.
    umbral_control = None
    desviacion_estandar_control = None
    showerror("Error", "No se detectaron gotas en el control
negativo.")

```

La función `elegir_imagen_control()` permite al usuario seleccionar una imagen desde su sistema de archivos mediante un cuadro de diálogo. Este diálogo filtra los archivos para que solo se puedan elegir imágenes con extensiones **.jpg**, **.jpeg**, y **.png**. Esta función repite el mismo procedimiento que la función antes descrita “`elegir_imagen()`” y parte de “`apply_canny()`” respecto a la importación y detección de las microgotas, a diferencia que esta función tiene como finalidad encontrar el umbral de control negativo para la clasificación correcta de las microgotas. Por tanto, su funcionamiento se describe de la siguiente manera:

La imagen seleccionada es leída usando la librería OpenCV, y se redimensiona a una altura de 600 píxeles mientras se mantiene la relación de aspecto original.

Después, la imagen se divide en 3 canales RGB (red, green y blue respectivamente) mejorando en específico el canal verde. Posteriormente, a este canal se le aplica un aumento en el contraste usando un valor de escala *alpha* de 1.9, lo que amplía las diferencias entre los valores de los píxeles, enfatizando las características más importantes.

Después se aplica el filtro gaussiano al canal verde lo que reduce el ruido y permite que los bordes sean más detectables. El propósito principal de esta acción es remover el ruido de la imagen generado por los canales rojo y azul, mejorar el contraste en el canal verde solamente y volver a unirlos para no perder información de dichos canales, sin que el canal principal que es el verde se vea afectado.

A la imagen que ha sido suavizada con el filtro gaussiano se le aplica el algoritmo de Canny para iniciar con la detección de bordes. Después, la imagen resultante se dilata y erosiona para acentuar los bordes, asegurar de que estén bien definidos y facilitar la búsqueda de círculos en la imagen usando la Transformada de Hough, el cual es un método para encontrar formas geométricas. Los círculos detectados se interpretan como las microgotas de interés, y cuyo objetivo es el análisis de sus tonalidades.

Cuando los círculos ya han sido detectados, la función sustrae imágenes individuales correspondientes a cada uno y calcula su tonalidad promedio. La tonalidad es la media de los píxeles dentro del círculo, representando así la luminosidad de la microgota.

Si hay tonalidades detectadas, la función obtiene el promedio total para calcular el umbral. Este valor representa la referencia para determinar si las tonalidades de futuras imágenes están dentro del rango esperado o si existe alguna irregularidad. También se evalúa la desviación estándar de las tonalidades, que mide la dispersión de los valores de tonalidad respecto al umbral. Una desviación estándar baja sugiere que las tonalidades son consistentes entre sí, por otro lado, una alta desviación podría indicar diferencias en las microgotas, probablemente causadas por errores de detección u otras condiciones experimentales. Si existen tonalidades presentes, la función muestra un cuadro de texto indicando cuántas tonalidades se obtuvieron. En caso de que no se detecten gotas (es decir, no se encuentran tonalidades), la función asigna None al umbral y a la desviación estándar, y muestra un mensaje de error.

3- Función para asignar valores definidos a los parámetros del umbral, sensibilidad de detección y radio mínimo y máximo.

```
def set_all_droplets():
    # Declara que las variables son globales para poder modificarlas dentro de la función.
    global param2_active, low_threshold_active, min_radius_active,
    max_radius_active
    # Asigna los valores globales específicos a las variables activas.
    param2_active = param2_all
    low_threshold_active = low_threshold_all
    min_radius_active = 19
    max_radius_active = 22
    # Actualiza los valores de los controles deslizantes (sliders) de la interfaz de usuario.
    # Esto sincroniza la interfaz con los nuevos valores de los parámetros.
    w.set(low_threshold_all) # Actualiza el control deslizante del umbral bajo.
    x.set(param2_all) # Actualiza el control deslizante del parámetro 2.
    y_slider.set(19) # Actualiza el control deslizante del radio mínimo.
    z_slider.set(22) # Actualiza el control deslizante del radio máximo.
    # Llama a la función `apply_canny` para aplicar la detección de bordes con los nuevos parámetros.
    apply_canny()
```

Esta función se encarga de configurar los parámetros necesarios para detectar todas las microgotas en la imagen. Utiliza variables globales para modificar los valores de los parámetros activos de detección: `param2_active`, `low_threshold_active`, `min_radius_active` y `max_radius_active`. La función asigna valores predefinidos a estos parámetros, específicamente el umbral bajo (`low_threshold_active`), el parámetro 2 (`param2_active`), y los radios mínimo y máximo de las microgotas (`min_radius_active`, `max_radius_active`). Luego, sincroniza los controles deslizantes de la interfaz de usuario con estos nuevos valores, actualizando los controles deslizantes correspondientes al umbral bajo (`w`), el parámetro 2 (`x`), el radio mínimo (`y_slider`) y el radio máximo (`z_slider`). Para finalizar, se llama a la función `apply_canny` para aplicar la detección de bordes con los nuevos parámetros, lo que actualiza la visualización de la imagen procesada en la interfaz.

4- Función para asignar valores definidos a los parámetros del umbral, sensibilidad de detección y radio mínimo y máximo de gotas positivas.

```
def set_positive_droplets():
    # Declara las variables globales para poder modificar sus valores
    global param2_active, low_threshold_active, min_radius_active,
    max_radius_active
    # Asigna los valores específicos para detectar microgotas positivas a las
    # variables globales
    param2_active = param2_positive
    low_threshold_active = low_threshold_positive
    min_radius_active = 19
    max_radius_active = 22
    # Actualiza el control deslizante del umbral bajo (low_threshold) en la
    # interfaz de usuario
    w.set(low_threshold_positive)
    # Actualiza el control deslizante del parámetro 2 (param2) en la interfaz de
    # usuario
    x.set(param2_positive)
    # Actualiza el control deslizante del radio mínimo (min_radius) en la interfaz
    # de usuario
    y_slider.set(19)
    # Actualiza el control deslizante del radio máximo (max_radius) en la interfaz
    # de usuario
    z_slider.set(22)
    # Aplica la detección de bordes usando los parámetros actualizados
    apply_canny()
```

La función `set_positive_droplets()` configura los parámetros específicos para la detección de microgotas positivas. Primero, declara las variables globales `param2_active`, `low_threshold_active`, `min_radius_active` y `max_radius_active` para poder modificarlas dentro de la función. Luego, asigna valores específicos para detectar microgotas positivas a estas variables: `param2_active` se establece en `param2_positive`, `low_threshold_active` en `low_threshold_positive`, `min_radius_active` se fija en 19 y `max_radius_active` en 22. Después, actualiza los controles deslizantes de la interfaz de usuario con estos valores: `w` se establece en `low_threshold_positive`, `x` en `param2_positive`, `y_slider` en 19 y `z_slider` en 22. Se llama a `apply_canny()` para aplicar la detección de bordes con los parámetros actualizados.

5- Función para la detección y visualización de las tonalidades

```
def update_image():
    # Copia la imagen original para trabajar sobre ella
    result_image = image.copy()
    all_circles = []

    # Si hay círculos detectados, agrégalos a la lista de todos los círculos
    if detected_circles is not None:
        all_circles = [(pt[0], pt[1], pt[2]) for pt in detected_circles[0, :]]

    # Inicializa las variables globales de tonalidades
    global tonalidades, tonalidades_bajas, tonalidades_altas
    tonalidades = []

    # Calcula las tonalidades de todas las gotas (detectadas y manuales)
    for (a, b, r) in all_circles + manual_circles_high + manual_circles_low:
        y1, y2 = max(int(b) - int(r), 0), min(int(b) + int(r), image.shape[0])
        x1, x2 = max(int(a) - int(r), 0), min(int(a) + int(r), image.shape[1])
        if y2 > y1 and x2 > x1:
            sub_image = image[y1:y2, x1:x2]
            if sub_image.size > 0:
                tonalidad = np.mean(sub_image)
                if not np.isnan(tonalidad):
                    tonalidades.append(tonalidad)

    # Define el umbral de tonalidad basado en el control negativo si está disponible
    if umbral_control is not None:
        umbral_tonalidad = umbral_control + 3 * desviacion_estandar_control
    # Si no hay control negativo, calcula el umbral basado en las tonalidades
    # detectadas
    elif len(tonalidades) > 0:
        if len(tonalidades) >= 2:
            tonalidades = np.array(tonalidades).reshape(-1, 1).astype(np.float32)
            criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100,
0.2)
            k = 2
            _, labels, centers = cv2.kmeans(tonalidades, k, None, criteria, 10,
cv2.KMEANS_RANDOM_CENTERS)
            centers = sorted(centers.flatten())
            umbral_tonalidad = sum(centers) / 2
        else:
            umbral_tonalidad = np.mean(tonalidades) + 10
    else:
        umbral_tonalidad = 0
    # Filtra las tonalidades en bajas y altas según el umbral calculado
    tonalidades = np.array(tonalidades).flatten()
    tonalidades_bajas = [tono for tono in tonalidades if tono <= umbral_tonalidad]
    tonalidades_altas = [tono for tono in tonalidades if tono > umbral_tonalidad]
```

```

# Dibuja los círculos en la imagen resultante, diferenciando por tonalidad
for (a, b, r) in all_circles:
    tonalidad = np.mean(image[max(int(b) - int(r), 0):min(int(b) + int(r),
image.shape[0]), max(int(a) - int(r), 0):min(int(a) + int(r), image.shape[1])])
    if tonalidad > umbral_tonalidad:
        cv2.circle(result_image, (a, b), r, (255, 0, 0), 2)
    else:
        cv2.circle(result_image, (a, b), r, (0, 0, 255), 2)
# Dibuja los círculos manuales en la imagen resultante
for (x, y, r) in manual_circles_high:
    cv2.circle(result_image, (x, y), r, (255, 0, 0), 2)

for (x, y, r) in manual_circles_low:
    cv2.circle(result_image, (x, y), r, (0, 0, 255), 2)
# Actualiza las etiquetas de la interfaz con el conteo de gotas negativas y
positivas
lblTonalidadesBajas.config(text=f"Negative" droplets:
{len(tonalidades_bajas)}")
lblTonalidadesAltas.config(text=f"Positive" droplets:
{len(tonalidades_altas)})"
# Añade el texto con el total de gotas en la imagen
total_circles = len(tonalidades_bajas) + len(tonalidades_altas)
text = f'Total de microgotas: {total_circles}'
font_scale = 0.5
thickness = 1
color = (255, 255, 255)
font = cv2.FONT_HERSHEY_SIMPLEX
text_size, _ = cv2.getTextSize(text, font, font_scale, thickness)
text_w, text_h = text_size
pos_x = result_image.shape[1] - text_w - 10
pos_y = result_image.shape[0] - 10
cv2.putText(result_image, text, (pos_x, pos_y), font, font_scale, color,
thickness)
# Convierte la imagen resultante a un formato compatible con Tkinter
im = Image.fromarray(result_image)
img = ImageTk.PhotoImage(image=im)
# Actualiza la etiqueta de la interfaz con la nueva imagen
lblOutputImage.configure(image=img)
lblOutputImage.image = img
# Muestra el umbral actualizado en la etiqueta correspondiente
lblUmbraletonalidad.config(text=f"Umbral de tonalidad:
{umbral_tonalidad:.2f}")

```

La función `update_image` comienza copiando la imagen original para trabajar sobre ella y asegurar que la imagen original no se modifique. Luego, verifica si hay círculos detectados y, de ser así, los agrega a la lista `all_circles`. Inicializa las variables globales `tonalidades`, `tonalidades_bajas`, y `tonalidades_altas`. Para cada círculo en `all_circles` y los círculos manuales, calcula la tonalidad promedio dentro de cada círculo y agrega estas tonalidades a la lista `tonalidades`. Si el umbral de control está disponible, se define el umbral de tonalidad basado en el control negativo; si no está disponible y hay tonalidades detectadas, calcula el umbral usando k-means clustering o simplemente el promedio de las tonalidades más un valor fijo. Si no hay tonalidades, el umbral se establece en 0. Luego, se clasifican las tonalidades en bajas y altas comparándolas con el umbral calculado. Se dibujan los círculos en la imagen resultante,

diferenciando los círculos con tonalidades altas en azul y los de tonalidades bajas en rojo. También se dibujan los círculos manuales en la imagen resultante. Actualiza las etiquetas de la interfaz con el conteo de gotas negativas y positivas y añade un texto con el total de gotas en la imagen. Convierte la imagen resultante a un formato compatible con Tkinter y actualiza la etiqueta de la interfaz con la nueva imagen. Por último, muestra el umbral actualizado en la etiqueta correspondiente.

6. Función para añadir o remover círculos manuales

```
def add_or_remove_circle(event):
    # Se declaran variables globales para que puedan ser modificadas dentro de
    # la función
    global manual_circles_high, manual_circles_low, detected_circles,
    add_high_tone_circle, add_low_tone_circle, remove_detected_circle

    # Obtención de las coordenadas del evento de clic del usuario
    x, y = event.x, event.y

    # Obtención de los valores de los controles deslizantes para el radio
    # mínimo y máximo de los círculos
    min_radius = y_slider.get()
    max_radius = z_slider.get()

    # Calcula un radio promedio para el nuevo círculo
    radius = (min_radius + max_radius) // 2

    # Si el usuario hace clic derecho (event.num == 3)
    if event.num == 3: # Clic derecho para agregar
        # Si se ha seleccionado agregar un círculo de tonalidad alta
        if add_high_tone_circle:
            # Añade un círculo con las coordenadas y el radio calculado a la lista de
            # círculos de tonalidad alta
            manual_circles_high.append((x, y, radius)) # Agregar círculo
            con tonalidad alta
        # Si se ha seleccionado agregar un círculo de tonalidad baja
        elif add_low_tone_circle:
            # Añade un círculo con las coordenadas y el radio calculado a la lista de
            # círculos de tonalidad baja
            manual_circles_low.append((x, y, radius)) # Agregar círculo
            con tonalidad baja
        # Actualiza la imagen para reflejar los cambios
        update_image()

    # Si el usuario hace clic izquierdo (event.num == 1)
    elif event.num == 1: # Clic izquierdo para eliminar
        # Si se ha seleccionado eliminar un círculo detectado automáticamente
        if remove_detected_circle:
            if detected_circles is not None:
```

Continúa...

```

# Recorre la lista de círculos detectados
    for i, pt in enumerate(detected_circles[0, :]):
        a, b, r = pt[0], pt[1], pt[2]
# Comprueba si el clic del usuario está dentro del área de algún círculo
# detectado
        if (x - a) ** 2 + (y - b) ** 2 <= r ** 2:
# Elimina el círculo de la lista
            detected_circles = np.delete(detected_circles, i,
axis=1)
            print(f"Círculo eliminado: ({a}, {b}, {r})")
# Actualiza la imagen para reflejar los cambios
            update_image()
            return
else:
# Recorre la lista de círculos añadidos manualmente de tonalidad alta
    for circle in manual_circles_high:
        cx, cy, r = circle
# Comprueba si el clic del usuario está dentro del área de algún círculo
        if (x - cx) ** 2 + (y - cy) ** 2 <= r ** 2:
# Elimina el círculo de la lista
            manual_circles_high.remove(circle)
# Actualiza la imagen para reflejar los cambios
            update_image()
            return
# Recorre la lista de círculos añadidos manualmente de tonalidad baja
    for circle in manual_circles_low:
        cx, cy, r = circle
# Comprueba si el clic del usuario está dentro del área de algún círculo
        if (x - cx) ** 2 + (y - cy) ** 2 <= r ** 2:
# Elimina el círculo de la lista
            manual_circles_low.remove(circle)
# Actualiza la imagen para reflejar los cambios
            update_image()
            return

```

La función `add_or_remove_circle(event)` gestiona la adición o eliminación de círculos en respuesta a eventos de clic del usuario. Primero, se declaran las variables globales `manual_circles_high`, `manual_circles_low`, `detected_circles`, `add_high_tone_circle`, `add_low_tone_circle` y `remove_detected_circle` para que puedan ser modificadas dentro de la función. Luego, se obtienen las coordenadas del clic del usuario (`x`, `y`) y los valores actuales de los controles deslizantes para el radio mínimo y máximo de los círculos, calculando un radio promedio para el nuevo círculo. Si el usuario hace clic derecho (`event.num == 3`), se añade un nuevo círculo en las coordenadas especificadas. Si `add_high_tone_circle` está activado, el círculo se añade a la lista `manual_circles_high`; si `add_low_tone_circle` está activado, se añade a `manual_circles_low`. Después, se actualiza la imagen para reflejar los cambios. Si el usuario hace clic izquierdo (`event.num == 1`), se elimina un círculo existente. Si `remove_detected_circle` está activado, se verifica si el clic del usuario coincide con algún círculo detectado automáticamente. Si coincide, el círculo se elimina de la lista.

detected_circles y se actualiza la imagen. Si *remove_detected_circle* no está activado, se verifica si el clic del usuario coincide con algún círculo añadido manualmente en las listas *manual_circles_high* o *manual_circles_low*. Si coincide, el círculo se elimina de la lista correspondiente y se actualiza la imagen.

5 Resultados

En este capítulo se presentan los resultados del análisis de imágenes vinculadas a casos de estudio. Estos casos corresponden a muestras orofaríngeas de pacientes del IMSS confirmados positivos y con cargas virales diferentes. Cada una de las imágenes se analizaron con la metodología y el software de aplicación propuestos en este trabajo. Los resultados de este análisis se compararon con los resultados y metodología presentados por la Dra. Kenia Chávez [9], quien empleó un software de procesamiento de imágenes comercial (ImageJ) para semi automatizar el conteo y hacerlo de forma más rápida.

5.1 Casos de estudio

Para el análisis de las imágenes de microgotas se usaron 5 imágenes representativas de cada paciente, de un conjunto total de aproximadamente 20 imágenes por paciente, para 3 casos de carga viral de SARS-CoV-2: baja, media y alta.

Para cada conjunto de imágenes a analizar, primero fue necesario obtener un conjunto de imágenes en estado basal (es decir, sin muestras de pacientes). Estas imágenes de control iniciales nos sirvieron posteriormente para ajustar los parámetros del software de aplicación, utilizando como referencia el umbral de valores obtenidos de la selección de alguna de estas imágenes en estado basal. En caso de no utilizar imágenes de control, el software de aplicación está programado para, al introducir una imagen, emplear el algoritmo de *k-means*, cuya explicación detallada se encuentra en la Sección 3.3.

5.2 Análisis de resultados

En esta sección, se presentan los resultados del análisis de las imágenes correspondientes a tres diferentes cantidades de carga viral de 3 diferentes pacientes: baja, media y alta. Asimismo, se presenta la comparación de los resultados con los resultados proporcionados por ImageJ previamente reportadas en [9]. Para tener un método estandarizado de análisis de imagen se determinó lo siguiente: una vez seleccionadas tanto la imagen de control como la imagen a analizar, los parámetros de ajuste predeterminados, como el radio de las gotas o la sensibilidad, permanecen sin modificar. Además, si la selección de microgotas no es correcta, es decir, que el registro sea menor al apreciable, se añaden manualmente, mediante un clic, solo aquellas microgotas que se visualizan completas excluyendo a aquellas que no se muestran completamente, como, por ejemplo, aquellas que se encuentran en los bordes.

Las tablas y figuras que se presentan a continuación muestran los resultados del conteo realizado con el software propuesto para las tres diferentes cargas virales, así como una comparación entre el total de microgotas contadas con el software propuesto y el total proporcionado por el software comercial ImageJ.

Para registrar los resultados de la aplicación, se utilizaron dos procedimientos: uno para los casos en los que se estableció una imagen de control negativo y otro en las que se utiliza el algoritmo de *k-means*. En el primer caso, se registró el conteo inicial sin modificaciones en los parámetros predeterminados y, posteriormente, se registró el conteo seleccionando manualmente las microgotas faltantes. En el segundo caso, que utiliza el algoritmo *k-means*, se registró el conteo inicial generado automáticamente al establecer la imagen, sin seleccionar microgotas manualmente. Para el registro del conteo con ImageJ, únicamente se anotó el conteo final, después de agregar manualmente las gotas que no fueron reconocidas automáticamente.

Para cada conjunto de imágenes se utilizó la misma imagen de control, excepto en los casos donde se compararon las imágenes utilizando el algoritmo *k-means*. En estos casos, el software arrojó un valor de umbral de 53.53. Además, se adjuntaron a cada imagen su histograma y gráfica de dispersión, en la cual se representa el valor de cada gota dentro de un rango de 0 a 255, junto con su número correspondiente.

5.2.1 Muestras de pacientes con baja carga viral

Los resultados del análisis del primer conjunto de imágenes, correspondientes a muestras de pacientes con baja carga viral y utilizando una imagen de control, se presentan en la tabla de la Figura 5.1. En ella se observa que el número de microgotas contabilizadas manualmente con ImageJ fue similar al obtenido con el software propuesto para automatizar el análisis, después de realizar la selección manual de las gotas no detectadas inicialmente con los parámetros predeterminados. La diferencia entre ambos métodos fue de solo 2 a 3 gotas.

Gotas con baja carga viral					
No. Imagen	Total (con imageJ)	Total (pre-selección)	Total (pos-selección)	positivas	negativas
1	289	251	289	1	288
2	257	248	258	7	251
3	229	219	233	3	230
4	230	220	231	2	229
5	222	219	222	3	219

Figura 5.1 Tabla con resultados del análisis de imágenes con baja carga viral con imagen de control.

La Figura 5.2 muestra las imágenes 1, 2, 3, 4 y 5 que fueron analizadas, y cuyos resultados se presentan en la tabla de la Figura 5.1. Nótese que las microgotas detectadas como positivas están marcadas con círculos rojos, mientras que las negativas están marcadas con círculos azules. Junto a cada imagen se incluye una gráfica del valor de tonalidad de cada gota, correspondiente al valor del canal verde en el espacio RGB, que varía de 0 a 255. Las microgotas determinadas como positivas presentan una tonalidad baja, por debajo de un umbral, mientras que las negativas tienen una tonalidad más alta, por arriba del mismo umbral. También se incluye un histograma que muestra la distribución de frecuencia de aparición de las microgotas en la imagen en función de su tonalidad.

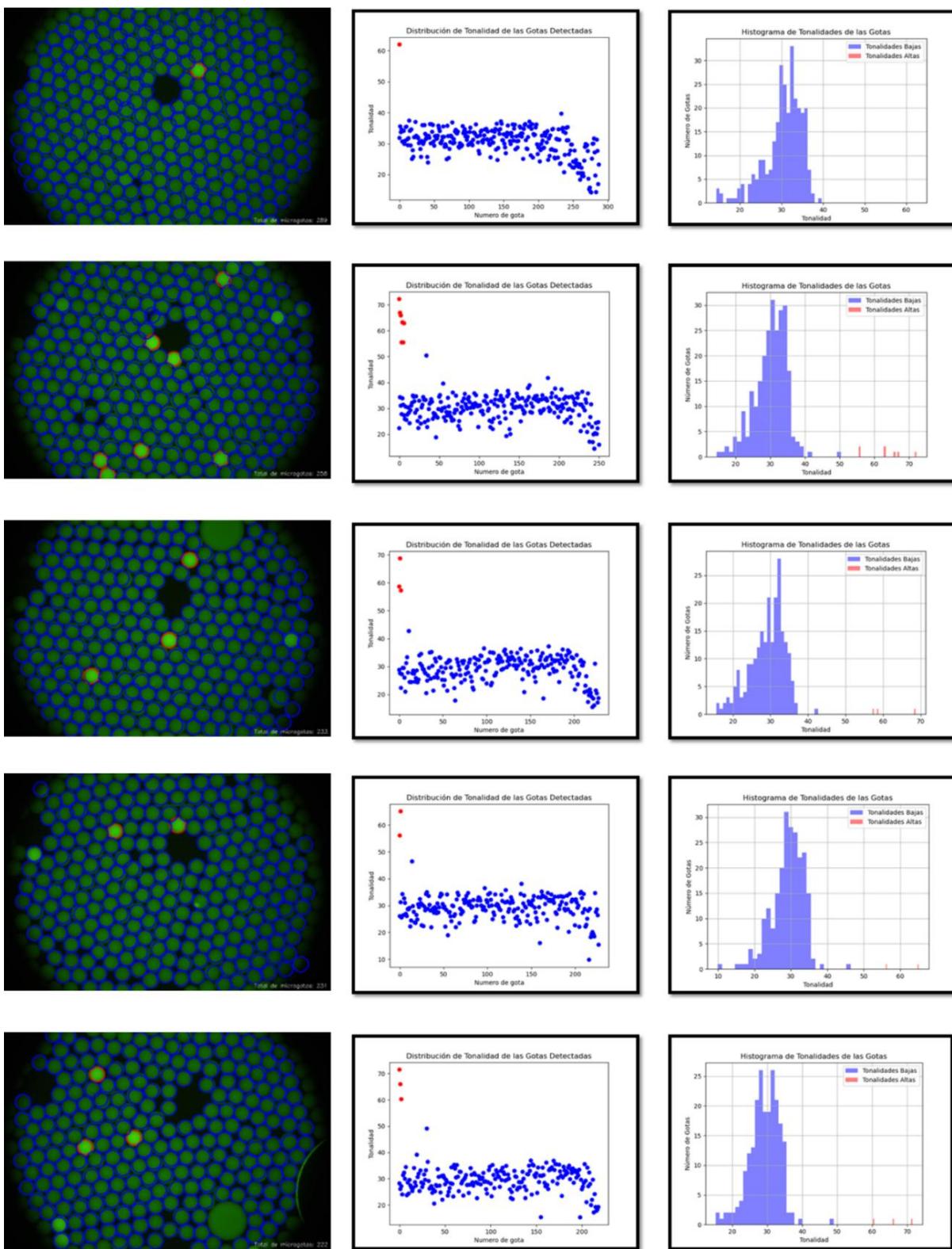


Figura 5.2 Análisis de muestras con carga viral baja. Izquierda: Imágenes de las microgotas, donde las gotas positivas detectadas están marcadas con un círculo rojo. Centro: Gráfica de las distribuciones de intensidad de fluorescencia para cada gota detectada; las gotas positivas se representan en rojo y las negativas en azul. Derecha: Histograma que muestra la frecuencia de aparición para cada valor de intensidad presente en las imágenes

La Figura 5.3 muestra los resultados numéricos del análisis de la imagen No. 1 sin usar una imagen de control negativo. En su lugar, utiliza el algoritmo de *k-means* para separar entre microgotas positivas y negativas. De esta forma, se proporcionan resultados notablemente diferentes a los conseguidos con una imagen de control negativo, como se puede ver en la Figura 5.2, donde existe un mayor número de microgotas positivas que negativas. Esta variación está presente en todas las imágenes cuya carga viral es baja, lo que indica la importancia de usar una imagen de control negativo específicamente para este grupo de imágenes. En caso contrario, se tendría que utilizar la selección manual de las microgotas, que va en contra del objetivo principal del programa: optimizar la selección del mayor número posible de gotas. En la Figura 5.3 se muestra la imagen No. 1 examinada sin imagen de control negativo, junto con un gráfico que presenta la distribución de las gotas dependiendo de su tonalidad y un histograma de la frecuencia en función de la tonalidad. Es importante notar que estas gráficas son altamente diferentes del análisis de la imagen No. 1 realizado con imagen de control negativo y cuyos resultados se encuentran en la primera fila de la Figura 5.2.

Gotas con baja carga viral sin imagen de control (kmeans)				
No. Imagen	Total (pre-selección)	positivas	negativas	Umbral
1	251	133	118	33.18

Figura 5.3 Resultado de imagen No. 1 con baja carga viral usando K-means.

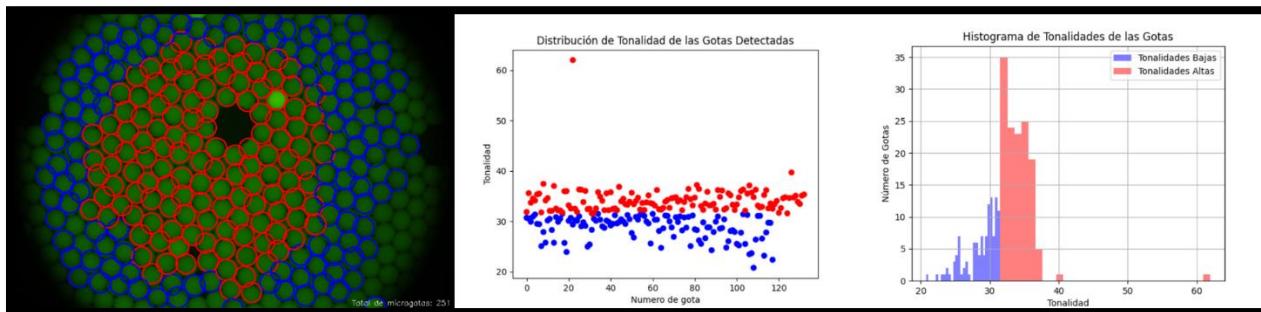


Figura 5.4 Imagen, histograma y gráfica de dispersión de muestra con carga viral baja usando k-means

5.2.2 Muestras de pacientes con carga viral media

Los resultados del análisis de un conjunto de imágenes con carga viral media muestran que el número total de microgotas contabilizado por nuestro software y por ImageJ varió aproximadamente en 3 gotas, una vez seleccionadas manualmente las gotas no detectadas por nuestro software con los parámetros de ajuste predeterminados. Sin embargo, antes de esta selección manual, la diferencia fue de alrededor de 10 microgotas, excepto en el caso de la imagen No. 1, donde los resultados mostraron una discrepancia considerable. Los resultados proporcionados por nuestro software son coherentes, como se evidencia en el tipo de diagrama de dispersión esperado para una muestra con carga viral media, en el cual el número de gotas positivas es significativamente menor que el de gotas negativas.

La Figura 5.5 resume los resultados del conteo de microgotas para cinco imágenes, obtenidos con el software propuesto. En la Figura 5.8 se presenta una tabla de imágenes en la que cada fila corresponde a los resultados gráficos de una imagen analizada. En la primera columna se muestra la imagen con las microgotas marcadas en azul o amarillo, según sean negativas o positivas. La segunda columna presenta una gráfica en la que el eje x representa el número de gota y el eje y su tonalidad. La tercera columna muestra los histogramas de la distribución de frecuencias de las microgotas en función de su tonalidad.

Carga Viral media					
No. Imagen	Total (con imageJ)	Total (pre-selección)	Total (pos-selección)	Positivas	Negativas
1	304	228	309	76	233
2	274	274	276	49	227
3	261	251	264	51	213
4	266	260	266	56	210
5	272	259	278	66	212

Figura 5.5 Tabla con resultados del análisis de imágenes con carga viral media.

Gotas con carga viral media sin imagen de control (kmeans)				
No. Imagen	Total (pre-selección)	positivas	negativas	Umbral
1	288	56	232	60.01

Figura 5.6 Resultado de imagen 1 con carga viral media usando k-means

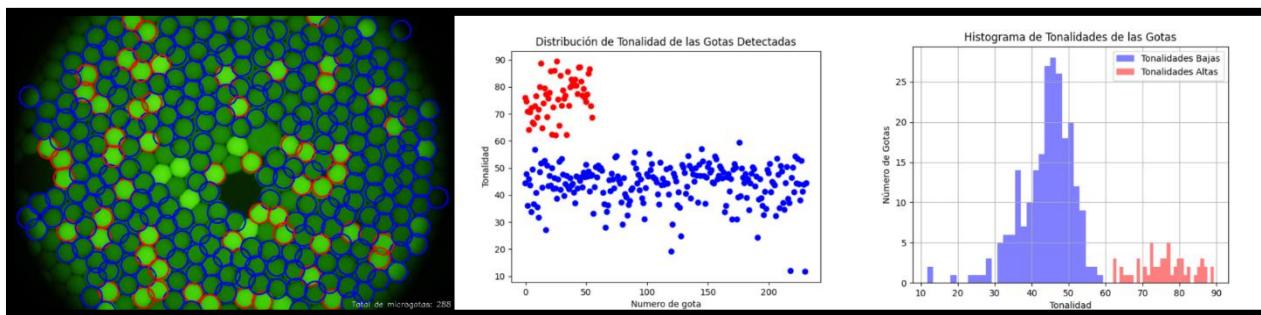


Figura 5.7 Imagen, histograma y gráfica de dispersión de muestra con carga viral media usando k-means

Por otro lado, al analizar la imagen No. 1 utilizando el algoritmo *k-means*, es decir, sin una imagen de control, se obtuvieron resultados bastante similares a los obtenidos con imágenes de control. De hecho, los resultados fueron incluso mejores que al emplear la imagen de control. La diferencia en el conteo entre el software ImageJ y nuestro software sin imagen de control fue de 21 gotas, una discrepancia considerablemente menor en comparación con la diferencia de 81 microgotas obtenida cuando se utilizó la imagen de control preselección de gotas manualmente.

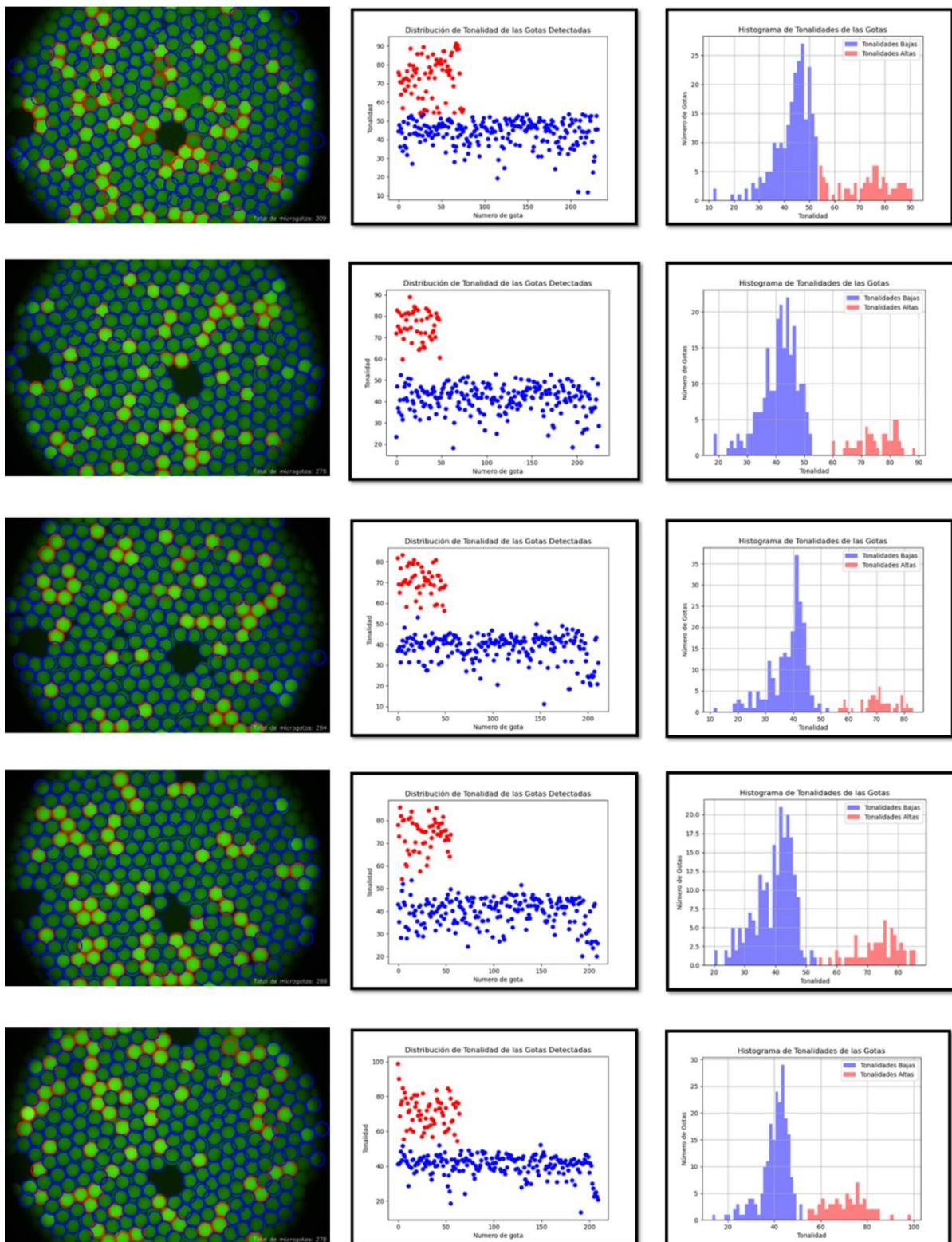


Figura 5.8 Análisis de muestras con carga viral media. Izquierda: Imágenes de las microgotas, donde las gotas positivas detectadas están marcadas con un círculo rojo. Centro: Gráfica de las distribuciones de intensidad de fluorescencia para cada gota detectada; las gotas positivas se representan en rojo y las negativas en azul. Derecha: Histograma que muestra la frecuencia de aparición para cada valor de intensidad presente en las imágenes.

5.2.3 Muestras de pacientes con carga viral alta

Los resultados del análisis del conjunto de imágenes de muestras de pacientes con alta carga viral se resumen en la tabla de la Figura 5.9. En estos resultados se observa que el conteo total preselección de microgotas difiere en aproximadamente 30 microgotas del conteo proporcionado por ImageJ, las cuales tuvieron que ser posteriormente seleccionadas de manera manual. Por otro lado, el número de microgotas detectadas con ImageJ fue similar al número contabilizado por el programa desarrollado, una vez realizada la selección manual de las gotas no incluidas en el análisis, con excepción de la cuarta imagen.

Carga viral alta					
No. Imagen	Total (con imageJ)	Total (pre-selección)	Total (pos-selección)	Positivas	Negativas
1	250	209	249	133	116
2	262	207	244	124	120
3	239	238	240	99	141
4	235	211	238	96	142
5	223	225	230	93	137

Figura 5.9 Tabla con resultados del análisis de imágenes con carga viral alta

Gotas con carga viral alta sin imagen de control (kmeans)				
No. Imagen	Total (pre-selección)	positivas	negativas	Umbral
1	209	99	110	56.67

Figura 5.1100 Resultado de imagen 1 con carga viral alta usando k-means.

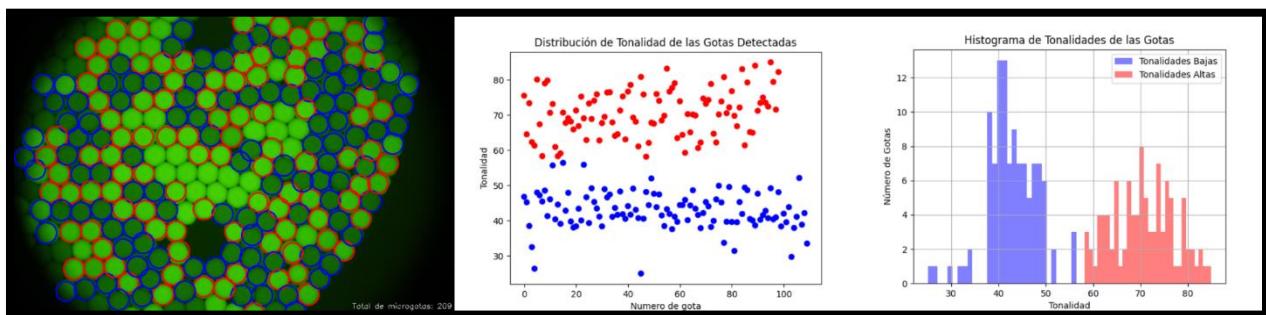


Figura 5.1111 Imagen, histograma y gráfica de dispersión de muestra con carga viral alta usando k-means.

Los resultados del análisis de la imagen No. 1 sin utilizar una imagen de control negativo se muestran en la tabla de la Figura 5.10 y en la Figura 5.11. Aquí, se observa que el conteo inicial de microgotas es el mismo que el conteo sin utilizar el control negativo. Sin embargo, se aprecia una pequeña variación en el número de microgotas clasificadas como positivas y negativas entre

ambos análisis. Esta variación fue más notable en las muestras con altas cargas virales, donde algunas gotas que debían clasificarse como positivas fueron marcadas como negativas.

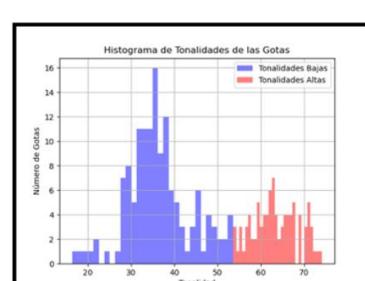
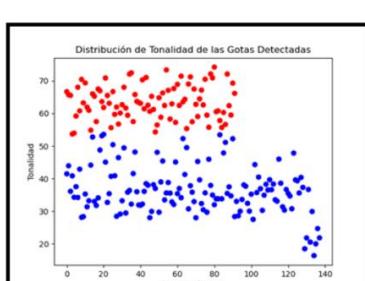
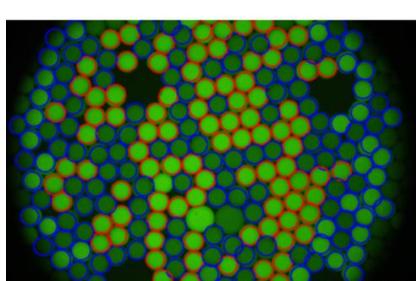
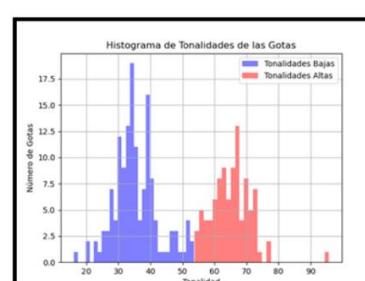
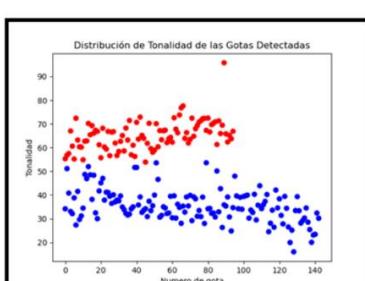
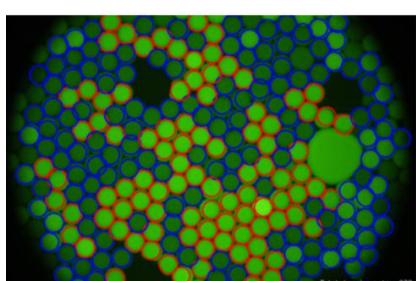
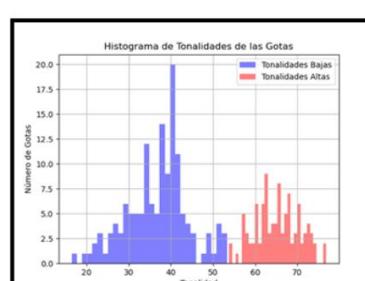
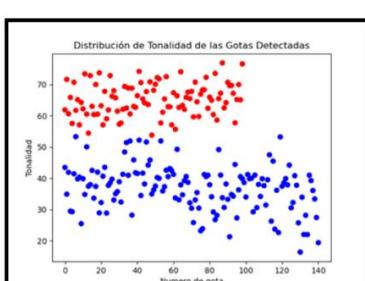
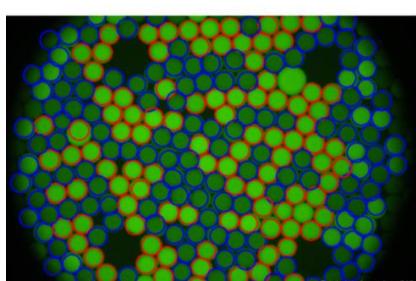
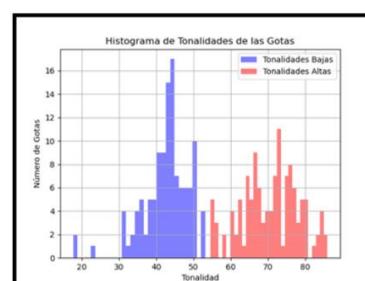
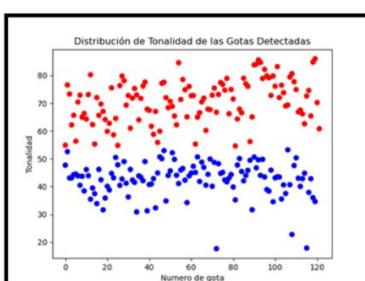
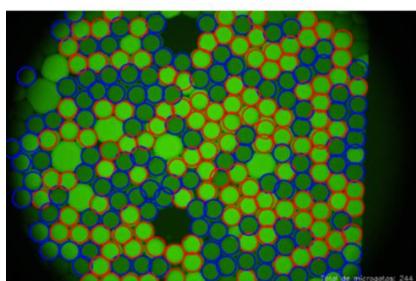
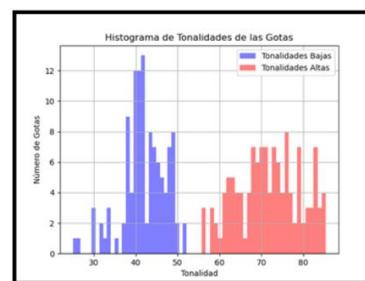
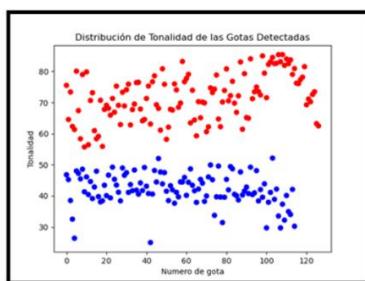
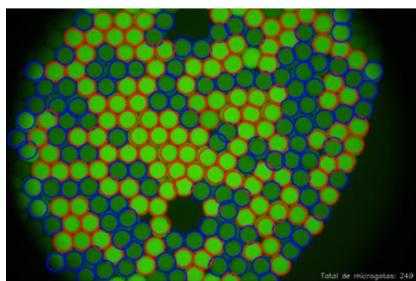


Figura 5.1122 Análisis de muestras con carga viral alta. Izquierda: Imágenes de las microgotas, donde las gotas positivas detectadas están marcadas con un círculo rojo. Centro: Gráfica de las distribuciones de intensidad de fluorescencia para cada gota detectada; las gotas positivas se representan en rojo y las negativas en azul. Derecha: Histograma que muestra la frecuencia de aparición para cada valor de intensidad presente en las imágenes

Al observar los resultados de los análisis, se puede notar que el uso de una imagen de control mejora significativamente los resultados en comparación con el análisis sin ella. Esto se debe a que, en el primer caso, cuando no se utiliza una imagen de control, el algoritmo de agrupación *k-means* genera un error considerable, lo que impide obtener resultados precisos. La imagen de control, por lo tanto, es fundamental para garantizar un análisis más confiable.

Para los análisis que se presentan, se eligió utilizar una imagen con los valores predeterminados y otra utilizando la selección manual de las microgotas no reconocidas automáticamente. Esto se decidió para simplificar los resultados, al enfocarse en los datos más importantes. No obstante, el ajuste de algunos parámetros, como el radio de las gotas o la sensibilidad de detección, mejora en gran medida el análisis. Así, es posible reducir el uso de la selección manual de las microgotas no detectadas, ya que lo que se pretende con el uso adecuado de las herramientas es reducir el proceso manual.

6 Conclusiones

En esta tesis se propuso un método para contabilizar las microgotas que reaccionaron con los marcadores fluorescentes, que después se categorizaron como microgotas positivas, para diferenciarlas de las que no reaccionaron. La metodología se basa en el análisis y procesamiento de imágenes tomadas con un microscopio óptico y procesadas con un algoritmo desarrollado en Python. También se propuso el diseño de una interfaz gráfica para sintonizar parámetros de ajuste, como el radio de las gotas y los umbrales de intensidad de fluorescencia, y se presentó una evaluación de resultados del análisis de imágenes de muestras tomadas a pacientes reales.

En el procedimiento del análisis de imágenes, fue muy importante desarrollar un método que permitiera adquirir los rasgos e información más importantes de cada imagen. Además, tener referencias de análisis anteriores, realizados ya sea de forma manual o mediante otros programas, como ImageJ, otorgó una base para comparar el conteo y el número aproximado de microgotas que se esperaban en las imágenes. Así, cada parámetro y etapa del procesamiento de las imágenes se fundamentó en esta información de referencia.

Uno de los mayores retos fue encontrar el ajuste adecuado de los parámetros al iniciar el programa, es decir, los parámetros predeterminados, debido a la cantidad de datos que era importante modificar. Por ejemplo, si se trabajaba con la imagen sin haberla separado en sus 3 canales principales, se generaba una pérdida de información debido al ruido. También se tuvo la dificultad de comprender con exactitud el funcionamiento de los distintos métodos de Python utilizados en el desarrollo del programa, debido a la gran variedad de parámetros ajustables, como la sensibilidad o el umbral de binarización, se podían alterar los resultados considerablemente de no ajustarse de manera correcta. Aún con las dificultades, se integraron las funciones que dan oportunidad al usuario de cambiar los parámetros según sus necesidades y visualizar los cambios en tiempo real, lo cual, aunque no estaba contemplado en un principio, se agregó para mejorar la interacción del usuario.

A partir del análisis de varias imágenes, se observaron particularidades interesantes al momento de usar el programa usando los dos algoritmos diferentes explicados los cuales son *k-means* y uso de una imagen de control. Los resultados obtenidos, en mayor o menor medida y con distintas cantidades de carga viral, fueron variables, como se mostró en la sección de resultados. Esto ofrece una visión general de las especificaciones en las que el programa presenta un sesgo considerable, especialmente si las condiciones no son las adecuadas, como el uso de imágenes con baja carga viral sin una imagen de control. A pesar de ello, en la mayoría de los casos, las imágenes se aproximan al resultado esperado, tanto en el conteo como en los diagramas de dispersión y en el histograma, que reflejan la correlación entre las imágenes con diferentes cargas virales.

Un aspecto importante por mencionar es que el número de gotas que fueron detectadas por la aplicación tienen una gran similitud con las detectadas con ImageJ, un software que no está totalmente diseñado para el análisis específico de las imágenes presentadas. La mayoría de las gotas se clasificaron correctamente como positivas o negativas. Asimismo, aunque agregar o eliminar microgotas manualmente es parte de las funcionalidades del programa, esto se puede reducir en gran medida al modificar algunos parámetros que el mismo programa tiene disponibles.

Por último, el presente trabajo se enfocó en el procesamiento de imágenes usando Python y algoritmos que conllevan procesos matemáticos, como la convolución, para detectar los bordes de las microgotas, por ejemplo. Si bien, los resultados a los que se llegaron fueron adecuados y se ajustaron con los objetivos de la tesis, el reconocimiento de cada microgota está sujeto al ajuste de los parámetros (radios, umbral, sensibilidad, etc.), especialmente si las microgotas son muy grandes o pequeñas. En cuanto a la agrupación utilizando el algoritmo de *k-means*, el no utilizar una imagen de control negativo acotó su eficiencia en ciertos casos (como en cargas virales bajas). Sin embargo, el programa tiene un gran potencial de implementarse técnicas avanzadas de aprendizaje profundo, como redes neuronales convolucionales. Estas redes, que necesitan imágenes exactas para su entrenamiento, podrían sacar ventaja de este programa, ya que, después de ajustar los parámetros, se permite extraer información precisa con un conteo adecuado de microgotas positivas y negativas. Esto ayudaría proporcionando las bases para que una posible red neuronal convolucional pueda clasificar y reconocer cada microgota eficientemente.

Anexos

A.1 Visual Studio Code

Microsoft Visual Studio Code es un editor de texto que fue liberado al público en el año 2016 y que cuenta con soporte para los 3 sistemas operativos principales (Microsoft Windows, Linux y MacOS). Actualmente es uno de los editores de texto más usados, ya que tiene características como resaltado en colores de sintaxis, depuración, así como soporte para varios lenguajes de programación entre ellos Python.

De manera breve, se muestra a continuación aspectos básicos del entorno de trabajo Visual Studio Code.

En la Figura A.1 (una vez descargado de su sitio oficial code.visualstudio.com y ejecutado para empezar a utilizarlo) se muestra la pantalla principal que contiene básicamente 3 aspectos principales.

El primero de ellos es la parte central, donde se tienen las opciones de creación de archivos nuevos, abrir archivos creados anteriormente o abrir carpetas que contenga más de un archivo de codificación.

Después se encuentra la barra lateral izquierda que contiene algunas pestañas y paneles, como el explorador de archivos, la búsqueda, control de código fuente (git), extensiones instaladas y otros paneles de ayuda.

También se encuentra la barra de herramientas superior que incluye los botones para acciones comunes, como abrir un archivo, guardar y rehacer, así como acceso a menús desplegables para configuraciones y extensiones.

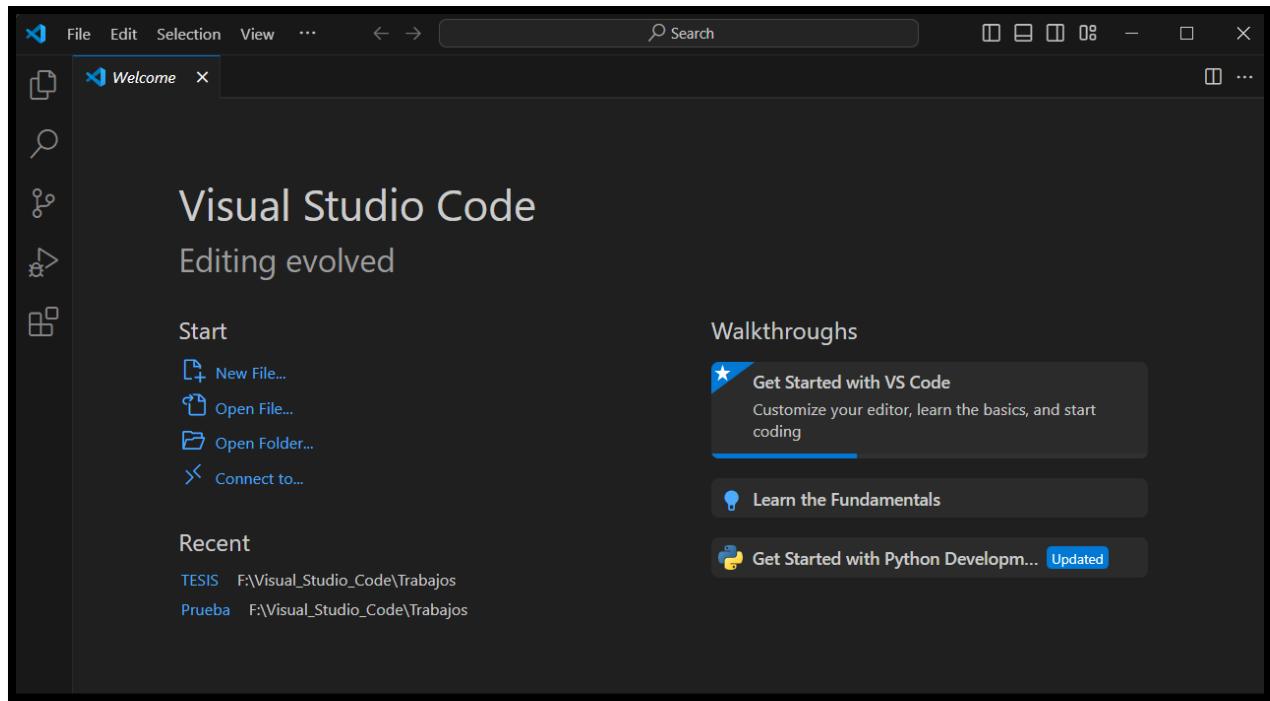


Figura A.0.1 Pantalla de inicio de Visual Studio Code

Al crear un nuevo archivo, como se muestra en la Figura A.2, es importante señalar el lenguaje con el que se empezará a codificar en el entorno de trabajo. Se recomienda guardar los archivos dentro de una carpeta para tener una mejor organización, pues en algunos casos, existen programas que “llaman” a otros códigos que se encuentran en diferentes direcciones y puede generar errores si no están en la dirección o ruta que se necesita para ser ejecutados.

Otra facilidad que proporciona Visual Studio Code es que pueden utilizarse plantillas o *templates*. Estas plantillas son códigos auxiliares reutilizables que proporcionan elementos y estructuras básicas que pueden ser adaptadas y personalizadas según sus necesidades específicas [27].

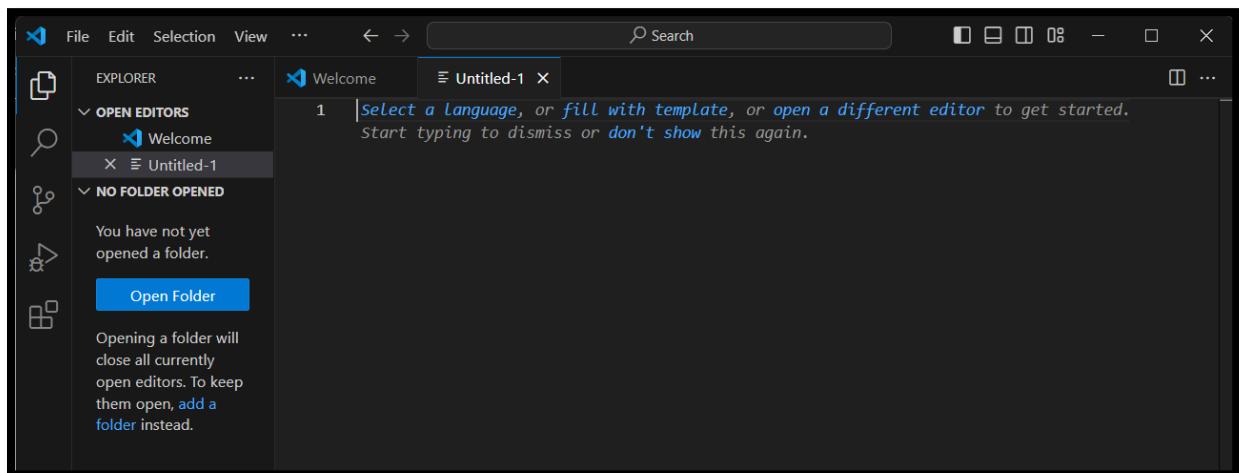


Figura A.0.2 Entorno de Visual Studio Code

A.1.1 Preparación del entorno de trabajo de VSC para la creación de la interfaz.

Para este trabajo es importante adecuar el entorno de trabajo con todas las herramientas necesarias para el procesamiento de imágenes que ayudará a la detección y conteo de gotas. En algunos casos se deben descargar bibliotecas en específico utilizando “pip” en la terminal de Python o en la terminal del sistema operativo que se utiliza.

De forma muy básica se muestra a continuación la manera en que pueden instalarse estas herramientas, teniendo en cuenta que pueden existir variaciones en la sintaxis dependiendo de la versión de python o del sistema operativo, por mencionar algunos ejemplos.

La biblioteca principal para todo lo relacionado con el ajuste de la imagen de las microgotas es cv2 de OpenCV, cuya forma de integrar a VSC es con el siguiente comando:

```
pip install opencv-python
```

Otra biblioteca que está estrechamente relacionada con OpenCV es Numpy y la cual puede descargarse con el comando a continuación:

```
pip install numpy
```

Dado que se van a trabajar con gráficas, es importante destacar el uso de la biblioteca matplotlib. Esta herramienta permite la creación de gráficas de dispersión e histogramas (entre otros) para el análisis de resultados. Uno de los comandos de instalación es:

```
python -m pip install -U matplotlib
```

Una biblioteca extra para manejo de operaciones matemáticas utilizada en este trabajo será SCIPY, la cual puede obtenerse con:

```
pip install scipy
```

También se incluirá la biblioteca Pillow, que amplía las capacidades de procesamiento de imágenes. Esta adición proporcionará herramientas adicionales para trabajar de manera más efectiva con imágenes en el proyecto.

```
pip install pillow
```

En este trabajo, donde se desarrollará una interfaz gráfica, una herramienta poderosa y útil dentro de Python es Tkinter, la cual permite crear interfaces gráficas de usuario de manera sencilla y eficiente. Con Tkinter es posible diseñar ventanas, botones, menús y otros elementos interactivos de forma intuitiva, lo que facilita la creación de aplicaciones con una interfaz visual según requiera el usuario.

Se instala en la terminal con:

```
pip install tk
```

A.2 Guía de funcionamiento de la interfaz de usuario

En los siguientes links se podrá encontrar un manual de referencia sobre el uso del programa realizado, así como instrucciones por si se desea correr el programa en un ordenador propio y el código fuente para descargar.

https://github.com/isaiisj/MicrodropletsDetection/blob/main/Manual_de_usuario.pdf

<https://github.com/isaiisj/MicrodropletsDetection/tree/main>

Referencias

- [1] D. K. Becerra-Paniagua, C. Chávez-Granados, y L. Oropeza-Ramos, «El gran mundo de la tecnología miniatura», *Rev. Cienc. UANL*, vol. 27, n.º 124, Art. n.º 124, mar. 2024, doi: 10.29105/cienciauanl27.124-2.
- [2] G. M. Whitesides, «The origins and the future of microfluidics», *Nature*, vol. 442, n.º 7101, pp. 368-373, jul. 2006, doi: 10.1038/nature05058.
- [3] A. Huebner, S. Sharma, M. Srisa-Art, F. Hollfelder, J. B. Edel, y A. J. deMello, «Microdroplets: A sea of applications?», *Lab. Chip*, vol. 8, n.º 8, p. 1244, 2008, doi: 10.1039/b806405a.
- [4] B. Kintses, L. D. van Vliet, S. R. Devenish, y F. Hollfelder, «Microfluidic droplets: new integrated workflows for biological experiments», *Nanotechnol. MiniaturizationMechanisms*, vol. 14, n.º 5, pp. 548-555, oct. 2010, doi: 10.1016/j.cbpa.2010.08.013.
- [5] Y. Zhu y Q. Fang, «Analytical detection techniques for droplet microfluidics—A review», *Anal. Chim. Acta*, vol. 787, pp. 24-35, jul. 2013, doi: 10.1016/j.aca.2013.04.064.
- [6] M. He, J. S. Edgar, G. D. M. Jeffries, R. M. Lorenz, J. P. Shelby, y D. T. Chiu, «Selective Encapsulation of Single Cells and Subcellular Organelles into Picoliter- and Femtoliter-Volume Droplets», *Anal. Chem.*, vol. 77, n.º 6, pp. 1539-1544, mar. 2005, doi: 10.1021/ac0480850.
- [7] M. Vaithiyathan, N. Safa, y A. T. Melvin, «FluoroCellTrack: An algorithm for automated analysis of high-throughput droplet microfluidic data», *PLOS ONE*, vol. 14, n.º 5, p. e0215337, may 2019, doi: 10.1371/journal.pone.0215337.
- [8] *Biodock.* (n.d.). Deep AI for biological images. [En línea]. Disponible en: <https://www.biodock.ai>
- [9] Kenia Chávez-Ramos y et al., «Advancements in Droplet Digital Isothermal Amplification: Overcoming Microfluidic Challenges», presentado en CD microfluidics MX Conference, México, CDMX., 3 de abril de 2024.
- [10] Y. Ding, P. D. Howes, y A. J. deMello, «Recent Advances in Droplet Microfluidics», *Anal. Chem.*, vol. 92, n.º 1, pp. 132-149, ene. 2020, doi: 10.1021/acs.analchem.9b05047.
- [11] F. Schuler et al., «Digital droplet LAMP as a microfluidic app on standard laboratory devices», *Anal. Methods*, vol. 8, n.º 13, pp. 2750-2755, 2016, doi: 10.1039/C6AY00600K.
- [12] G. van Rossum y F. L. Drake, *An introduction to Python: release 2.5*, 2. print. Bristol: Network Theory Limited, 2006.
- [13] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, y M. Cifrek, «A brief introduction to OpenCV», *2012 Proc. 35th Int. Conv. MIPRO MIPRO 2012 Proc. 35th Int. Conv.*, pp. 1725-1730, may 2012.
- [14] Alexander Mordvintsev y K Abid, «OpenCV-Python Tutorials Documentation, Release 1». 2014. [En línea]. Disponible en: <https://media.readthedocs.org/pdf/opencv-python-tutorials/latest/opencv-python-tutorials.pdf>.

- [15]D. Heras, «Clasificador de imágenes de frutas basado en inteligencia artificial», *Kill. Téc.*, vol. 1, p. 21, nov. 2017, doi: 10.26871/killkana_tecnica.v1i2.79.
- [16]«OpenCV: Operations on arrays». Accedido: 27 de febrero de 2024. [En línea]. Disponible en: https://docs.opencv.org/3.4/d2/de8/group__core__array.html#ga4ad01c0978b0ce64baa246811deeac24
- [17]J. Howse, *OpenCV computer vision with python*, vol. 27. Birmingham: Packt Publishing, 2013.
- [18]N. Zin Oo, «The Improvement of 1D Gaussian Blur Filter using AVX and OpenMP», en *2022 22nd International Conference on Control, Automation and Systems (ICCAS)*, nov. 2022, pp. 1493-1496. doi: 10.23919/ICCAS55662.2022.10003739.
- [19]A. Kaehler y G. Bradski, *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, Inc., 2016.
- [20]«OpenCV: detección de bordes astuta». Accedido: 7 de marzo de 2024. [En línea]. Disponible en: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html
- [21]G. Ji, X. Zhang, y X. Cheng, «Pedestrians Detection Based on the Integration of Human Features and Kernel Density Estimation», *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 490, p. 042027, abr. 2019, doi: 10.1088/1757-899X/490/4/042027.
- [22]«OpenCV: Feature Detection». Accedido: 12 de agosto de 2024. [En línea]. Disponible en: https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#ga47849c3be0d0406ad3ca45db65a25d2d
- [23]H. J. D., «Matplotlib: A 2D graphics environment», *IEEE Ann. Hist. Comput.*, vol. 9, n.º 03, 2007.
- [24]A. Rosebrock, *Practical Python and OpenCV: An introductory, example-driven guide to image processing and computer vision*. PyImageSearch, 2016.
- [25]F. Lundh y J. A. Clark, «Pillow: The friendly PIL fork». [En línea]. Disponible en: <https://python-pillow.org/>
- [26]H. R., «Openpyxl: A Python library to read/write Excel 2010 xlsx/xlsm files». Python Package Index, 2013.
- [27]ghogen, «Plantillas para proyectos y archivos - Visual Studio (Windows)». Accedido: 11 de abril de 2024. [En línea]. Disponible en: <https://learn.microsoft.com/es-es/visualstudio/ide/creating-project-and-item-templates?view=vs-2022>