

**NOMBRE**

Bryan Avila

**CARRERA**

Computación

**TEMA**

Informe Practica\_2

**MATERIA**

Visión por Computador

**DOCENTE**

Mgtr. Vladimir Robles

**CICLO**

Séptimo

**UNIDAD**

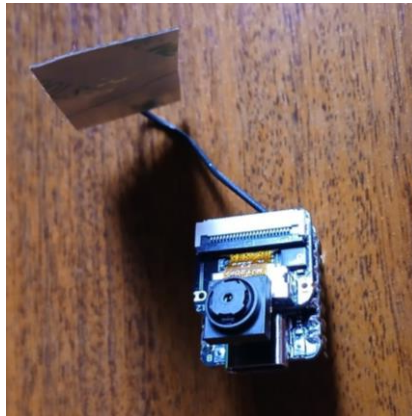
#2

# INFORME

## Configuraciones previas

### - Preparación del dispositivo (cámara ESP32)

Antes de comenzar, conectaremos la antena Wi-Fi al dispositivo para asegurar una correcta conexión inalámbrica y así poder cargar nuestra aplicación desde el entorno de desarrollo Arduino IDE.



*Imagen 1: Dispositivo ESP32*

### - Ejecución del programa

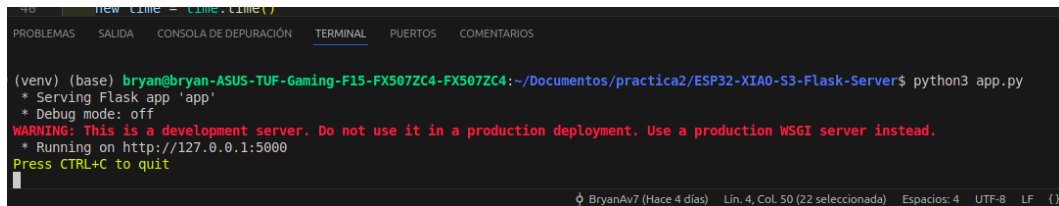
Después de instalar las librerías y seleccionar el puerto correspondiente en el Arduino IDE, cargaremos nuestra aplicación con el objetivo de realizar la transmisión de video en vivo. Utilizaremos los ejemplos que proporciona la propia herramienta y ejecutaremos el código para conectarnos a través de nuestra red local.

```
19  
20 void setup() {  
21   Serial.begin(115200);  
Output Serial Monitor x  
Message (Enter to send message to 'XIAO_ESP32S3' on '/dev/ttyACM0')  
  
Iniciando configuración de la cámara...  
  
Conectando a WiFi.....  
WiFi conectado.  
Dirección IP: 192.168.18.173  
  
¡Cámara lista! Usar http://192.168.18.173 para conectar
```

*Imagen 2: Ejecución del programa en Arduino IDE*

## Ejecución del servidor de Flask

Después de realizar el Fork del repositorio de la Guía Práctica 2 en GitHub, ejecutaremos nuestro servidor Flask para visualizar la transmisión a través de una interfaz web.



```
(venv) (base) bryan@bryan-ASUS-TUF-Gaming-F15-FX507ZC4-FX507ZC4:~/Documentos/practica2/ESP32-XIA0-S3-Flask-Server$ python3 app.py
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Imagen 3: Ejecución del servidor de Flask

## Funcionalidad del programa

### 1. Parte 1-A

#### Detección de movimiento y mostrar FPS

La detección de movimiento es una técnica que nos permite identificar cambios significativos entre cuadros consecutivos de un video, facilitando así la detección de actividad dentro de la escena. En este trabajo utilizamos la técnica Mixture of Gaussians (MoG) como método de sustracción de fondo para detectar dichas variaciones de forma eficiente.

Por otro lado, para evaluar el rendimiento del sistema se calcula el valor de FPS mediante una función llamada “calcular\_fps” que mide cuántos cuadros por segundo se están procesando. Este indicador es fundamental para asegurar una visualización fluida y un procesamiento en tiempo real del flujo de video.

#### Resultados



Imagen 4: Filtros de Detección de movimiento, escala de grises y cálculo de FPS

#### Detección de movimiento: [Mixture of Gaussians](#)

La técnica seleccionada fue Mixture of Gaussians (MoG), implementada en el código a través de una función que captura un flujo de video y detecta el movimiento en cada cuadro utilizando un sustractor de fondo. Previamente, la imagen original se convierte a escala de grises con el objetivo de reducir la complejidad del procesamiento y enfocarse únicamente en los cambios de intensidad entre cuadros, optimizando así la aplicación de la técnica.

Posteriormente, transmitimos los resultados en un frame a través de la interfaz web, donde podemos visualizar en tiempo real la imagen original con el cálculo de los FPS, la imagen convertida a escala de grises, y la salida generada por la técnica de Mezcla de Gaussianos (MoG) aplicada para la detección de movimiento.



*Imagen 5: Filtro Mixture of Gaussians*

### **Aplicación de Filtros**

En esta sección se aplica varios filtros a nuestra imagen previamente convertida en escala de grises con el objetivo de mejorar el contraste y la iluminación de la escena, las cuales son:

- **Histograma:** Mejora el contraste global de la imagen al distribuir los valores de píxeles de manera más uniforme.
- **Clahe:** Aplica una ecualización adaptativa que mejora el contraste local en diferentes regiones de la imagen, lo que ayuda a resaltar detalles en áreas que de otro modo podrían estar subexpuestas o sobreexpuestas.
- **Gamma(investigada):** Modifica el brillo de la imagen utilizando una transformación de gama, aumentando el brillo en las áreas más oscuras, permitiendo mejorar detalles en sombras sin afectar las zonas brillantes.

## Resultados



*Imagen 6: Aplicación de Filtros*

## Comparación

La aplicación del filtro de histograma mejora la visibilidad de los detalles en áreas oscuras o saturadas en la primera imagen. Al usar el filtro CLAHE en la segunda imagen, se ajusta el contraste en diferentes zonas, facilitando la visualización de los detalles. Por último, el filtro gamma aplicada en la tercera imagen, se aclara las áreas más oscuras, mejorando la imagen sin afectar las zonas que ya están bien iluminadas.

## Operaciones Bitwise(visualización de pixeles en movimiento)

Para la detección de movimiento en las imágenes utilizamos las operaciones Bitwise con el objetivo de poder resaltar las diferencias entre los cuadros consecutivos y de identificar áreas de cambio significativo. Estas operaciones nos permiten comparar imágenes y detectar el movimiento de manera eficiente. Las operaciones utilizadas son:

- **AND:** Identifica las áreas comunes entre las imágenes, ayudando a filtrar el ruido y centrarse en las zonas de cambio.
- **OR:** Combina las áreas con movimiento de diferentes cuadros, ampliando la región en movimiento y mejorando su visibilidad.
- **XOR:** Resalta las diferencias exactas entre las imágenes, mostrando solo las zonas que han cambiado.

## Resultados



*Imagen 6: Aplicación de Operaciones Bitwise*

## Comparación

En la primera imagen, la operación AND resalta las áreas comunes entre fotogramas y deja el fondo negro, enfocando la atención en las zonas constantes. En la segunda, la operación OR combina las regiones con movimiento, mostrando los cambios con más color y visibilidad. Finalmente, en la tercera imagen, la operación XOR destaca únicamente las diferencias entre fotogramas, marcando solo las zonas donde hubo cambio.

## 2. Parte 1-B

### Generación de Ruido

El ruido en una imagen es una alteración no deseada que afecta la calidad visual, generando puntos o variaciones que no forman parte de la escena original por la cual afecta su visualización. Para simular este efecto en el procesamiento de video, se definieron dos funciones llamadas "ruidoGaussiano", "ruidoSpeckle".

- **Gauss:** Esta consiste en aplicar variaciones suaves y aleatorias en los píxeles, simulando interferencias.
- **Speckle:** Esta aplica un patrón granular que distorsiona la imagen.
- **Combinado:** Aquí se combinan la Mezcla de Gaussiano y Speckle para generar una alteración más compleja.

## Resultados





Imagen 7: Generación de Ruido

### Parámetros de ruidos(generación de Ruido)

Desde la interfaz web desplegada con nuestro servidor Flask, el usuario cuenta con sliders que permiten ajustar los valores de cada imagen, como la media y desviación estándar para el ruido gaussiano aplicada en la imagen 1, y la varianza en la imagen 2 aplicada el ruido speckle.

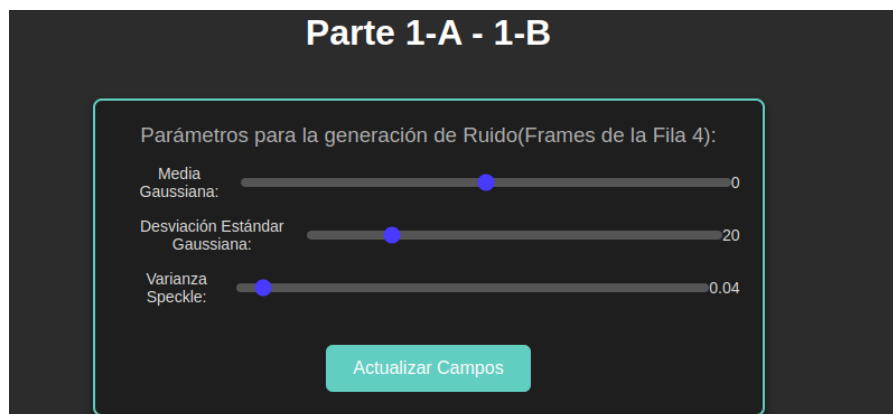


Imagen 8: Parámetros de Ruido

Al modificar estos valores, se puede observar en tiempo real cómo afectan a cada frame, especialmente en la tercera imagen, donde se combinan ambos tipos de ruido.

### Resultados

Valores:

- Media Gaussiana: 5
- Desviación Estandar Gaussiana: 10
- Varianza Speckle: 0.05



Imagen 9: Generación de Ruido

Valores:

- Media Gaussiana: 40
- Desviación Estandar Gaussiana: 80
- Varianza Speckle: 0.6



Imagen 10: Generación de Ruido con parámetros

## Generación de Filtros

Para mejorar la calidad de la imagen afectada por el ruido, se implementó una función llamada “aplicar\_filtros” que permite aplicar distintos filtros de suavizado con diferentes tamaños de mascara, cada uno con características específicas, las cuales son:

- **Mediana:** Este filtro elimina el ruido reemplazando cada píxel por el valor mediano de sus vecinos, es eficiente al eliminar ruido de tipo sal y pimienta.
- **Blur:** Este filtro suaviza la imagen calculando el promedio de los píxeles vecinos, eficiente para ruidos leves pero provoca pérdida de detalles en la imagen.
- **Gauss:** Este filtro consiste en aplicar una función de distribución gaussiana, suavizando la imagen de forma más natural y conservando mejor las transiciones y los contornos.



## Resultados



Imagen 11: Aplicación de Filtros

## Comparación

Al aplicar el filtro de mediana en la primera imagen, se eliminó eficazmente el ruido tipo sal y pimienta, lo que permitió una mejor visualización de los bordes y detalles. En la segunda imagen, el filtro Blur suavizó la imagen, aunque se notó una ligera pérdida de nitidez en los bordes. Finalmente, en la tercera imagen, el filtro Gaussiano logró un buen equilibrio entre la reducción de ruido y la conservación de los detalles, manteniendo una apariencia más natural.

## Algoritmos de detección de bordes

Los algoritmos de detección de bordes nos permiten identificar mejor las transiciones y contornos entre diferentes regiones, destacando las áreas de cambio más significativas, las cuales son:

### Sobel Total

Este algoritmo nos permite detectar los bordes en una imagen limpia(original), resaltando las transiciones de intensidad y contornos de forma precisa como se puede observar en la imagen 2.

- **Sobel con ruido(extra):** Se aplico un algoritmo extra en la cual nos permite observar los bordes distorsionados y falsos causados por el ruido como se puede observar en la imagen 1.

## Resultados



Imagen 12: Filtro detector de bordes Sobel

## Canny

Este algoritmo nos permite detectar bordes de una imagen limpia (original) mediante un proceso de suavizado, detección de gradientes, entre otros. Como podemos observar en la imagen 2, este algoritmo nos ofrece bordes nítidos y precisos.

- **Canny con ruido(extra):** Se aplico Canny con ruido en la imagen 1 con la finalidad de poder visualizar posibles bordes falsos o poco definidos debido a la interferencia del ruido.

## Resultados



Imagen 14: Filtro de detector de bordes Canny

### 3. Parte 2

#### Operaciones morfológicas

Son técnicas utilizadas en procesamiento de imágenes que nos permiten modificar la estructura de los objetos dentro de una imagen, destacando o eliminando ciertas características según su forma y tamaño. Son especialmente útiles para la eliminación de ruido, la mejora de contornos y la segmentación de imágenes.

Para realizar las operaciones morfológicas, se creó una función llamada "ope\_morfologicas", la cual aplica una serie de transformaciones a la imagen original, utilizando diferentes tipos de máscaras. Las cuales son:

- **Erosión:** Reduce los objetos en la imagen, eliminando pequeñas irregularidades.
- **Dilatación:** Expande los objetos en la imagen, rellenando huecos o ampliando detalles.
- **Top Hat:** Resalta las áreas brillantes que son más pequeñas que el kernel, útil para detectar detalles en un fondo oscuro.
- **Black Hat:** Resalta las áreas oscuras más pequeñas que el kernel, útil para detectar detalles oscuros sobre un fondo claro.
- **Imagen Original + (Top Hat – Black Hat):** Combina Top Hat y Black Hat para resaltar características prominentes de la imagen.

#### Pasos previos

- **Carga de imágenes**

Para la carga de imágenes médicas, estas deben guardarse previamente en la carpeta "static" antes de ejecutar el código.

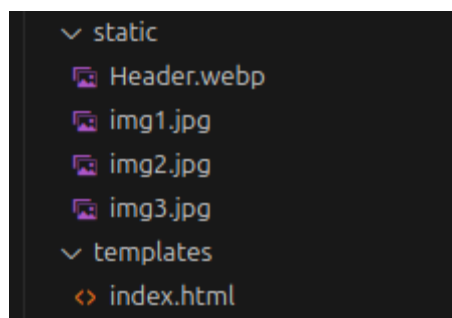


Imagen 13: Directorio de las imágenes medicas

- **Ejecución del programa**

En la interfaz web, debemos hacer clic en el botón "Ver Resultados" para visualizar los cambios aplicados a la imagen original, tras realizar las operaciones morfológicas en cada imagen con los diferentes tamaños de máscara.

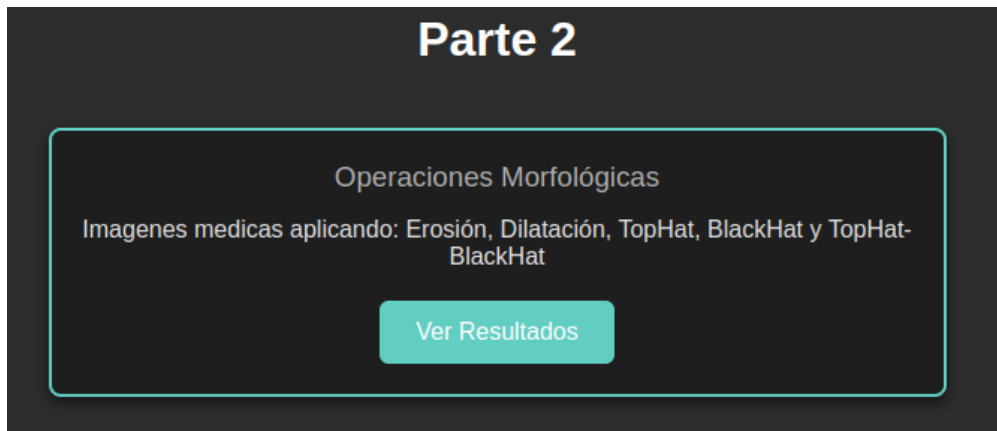


Imagen 14: Interfaz Web de los resultados

## - Resultados

En las siguientes imágenes podemos observar la mejora de la calidad de las imágenes médicas con diferentes tamaños de máscara al aplicar las operaciones morfológicas:

### - Imagen 1:

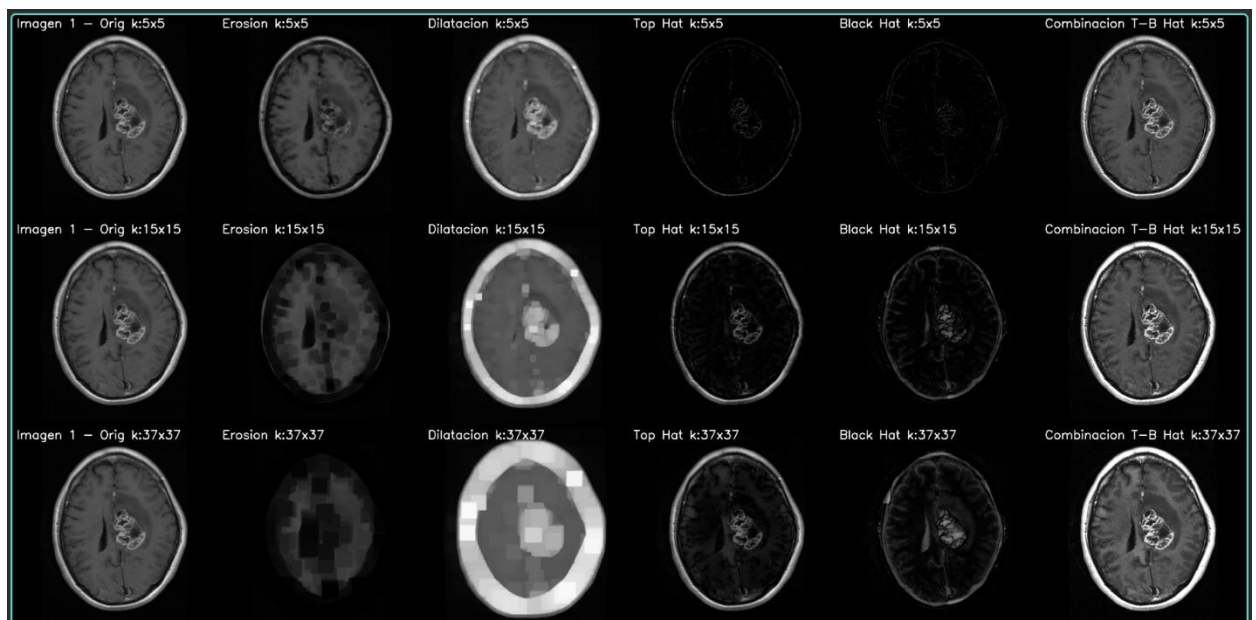
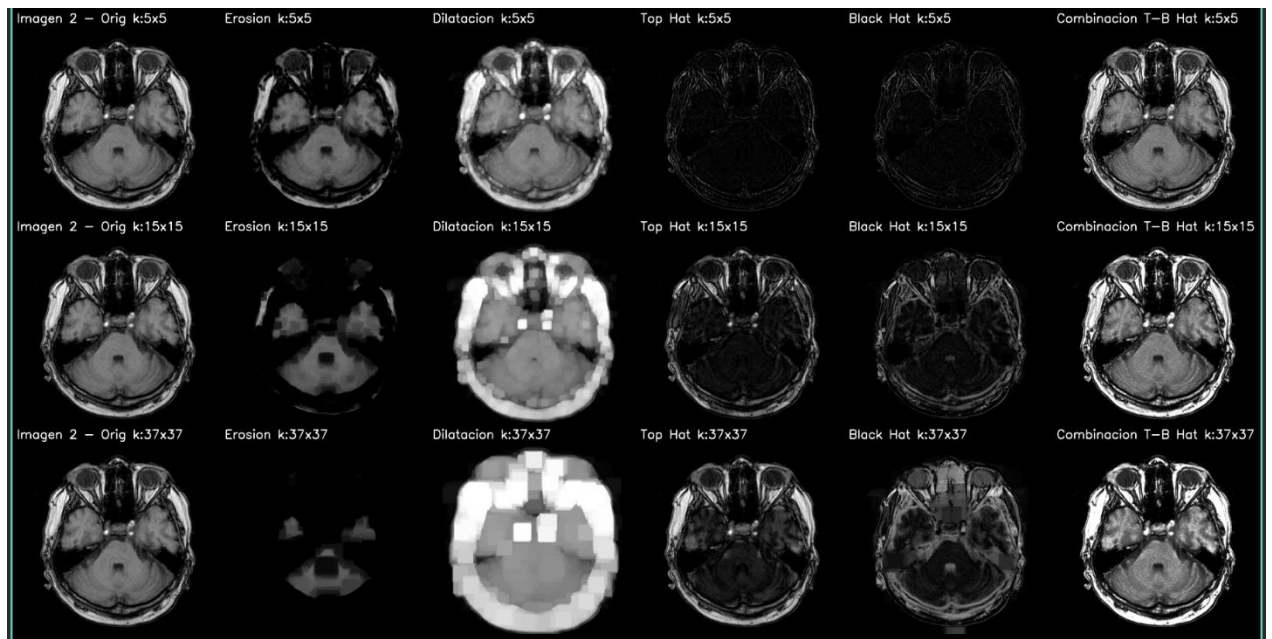


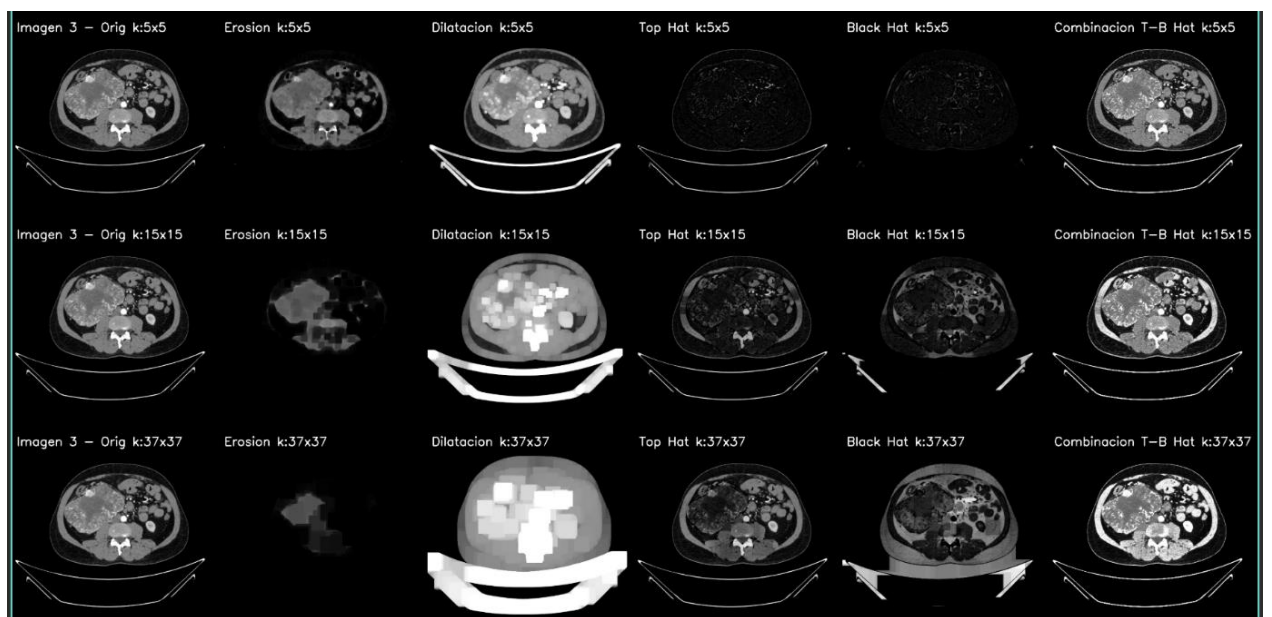
Imagen 15: Imagen de Tumor Cerebral

### - Imagen 2:



*Imagen 16: Imagen de Trumbosis Venosa Cerebral*

- Imagen 3:



*Imagen 17: Imagen de Tumor Renal*

## Comparación

Al comparar las imágenes originales con las que fueron cargadas previamente y tratadas mediante operaciones morfológicas, se puede observar una mejora notable en la nitidez y el contraste de estructuras clave, como los bordes de los tumores tanto cerebrales y renales.



Las técnicas como la dilatación y erosión permiten resaltar o limpiar regiones específicas, facilitando la diferenciación entre tejido sano y patológico. Además, las operaciones como Top Hat y Black Hat destacan detalles que podrían pasar desapercibidos en la imagen original.

En conjunto, estas transformaciones mejoran la capacidad de análisis visual y pueden apoyar de forma efectiva el diagnóstico médico al hacer más visibles los contornos y características relevantes de los tumores.

## **Anexos**

Código Github + archivo README.md(explicación del código):

<https://github.com/BryanAv7/ESP32-XIAO-S3-Flask-Server>

## **Conclusiones**

Para concluir, se evidenció la implementación práctica en relación a los conceptos fundamentales del procesamiento de imágenes mediante el uso de una ESP32-CAM y un servidor Flask, lo cual permitió la transmisión en vivo y la aplicación de diversas técnicas como detección de movimiento, filtros de mejora, generación de ruido, operaciones bitwise, detección de bordes y transformaciones morfológicas.

Además, se comprobó cómo cada una de estas técnicas contribuye a mejorar, analizar y visualizar imágenes, demostrando su utilidad tanto en tiempo real como en entornos médicos.

## Bibliografía

- Geeksforgeeks. (2025). *Geeksforgeeks*. Obtenido de <https://www.geeksforgeeks.org/python-opencv-canny-function/>
- Islam, N. (2021). *Kaggle*. Obtenido de <https://www.kaggle.com/datasets/nazmul0087/ct-kidney-dataset-normal-cyst-tumor-and-stone/data>
- Robles, V. (2025). *Medium*. Obtenido de <https://medium.com/@sparthakus/practical-guide-for-the-development-of-video-streaming-applications-using-the-04d76ebb886b>
- Safa, A. (2022). *Medium*. Obtenido de <https://medium.com/@abbessafa1998/motion-detection-techniques-with-code-on-opencv-18ed2c1acfaf>
- Sulani, I. (2024). *Kaggle*. Obtenido de <https://www.kaggle.com/code/sulaniishara/classifying-brain-tumors-from-mri-scans>