ESTUDIANTE:

KELLY PALTIN

ASIGNATURA:

VISIÓN POR COMPUTADOR

TÍTULO PRÁCTICA:

RECONOCIMIENTO DE FORMAS EN BASE A DETECCIÓN DE BORDES, BINARIZACIÓN POR UMBRAL, APLICACIÓN DE FILTROS Y OPERACIONES MORFOLÓGICAS

DOCENTE:

ING. VLADIMIR ROBLES

PERIODO ACADÉMICO:

64

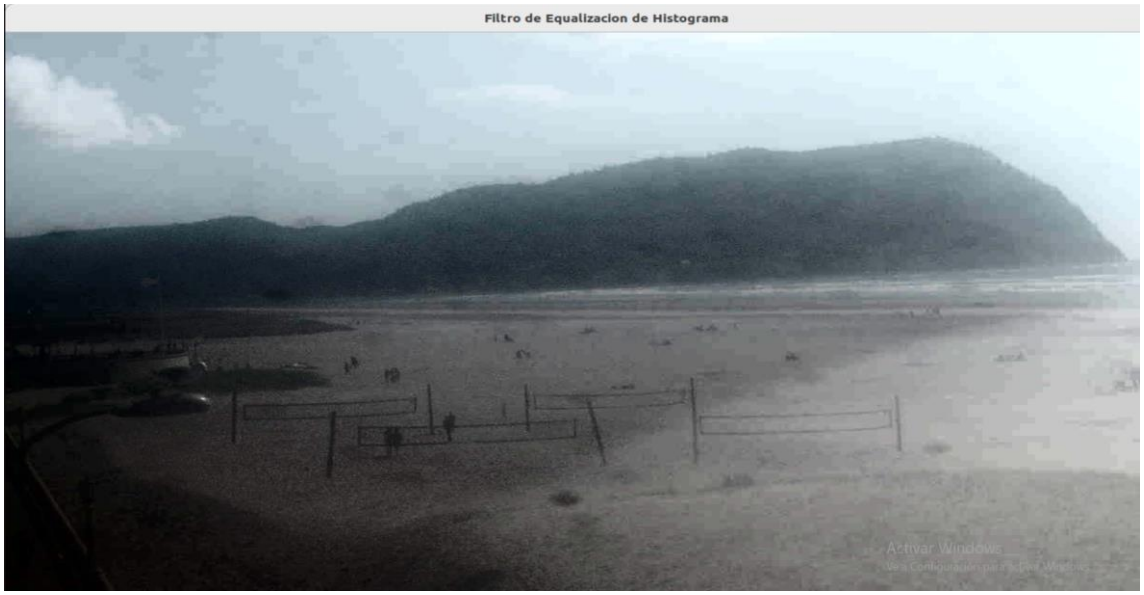
CUENCA – ECUADOR – 2024

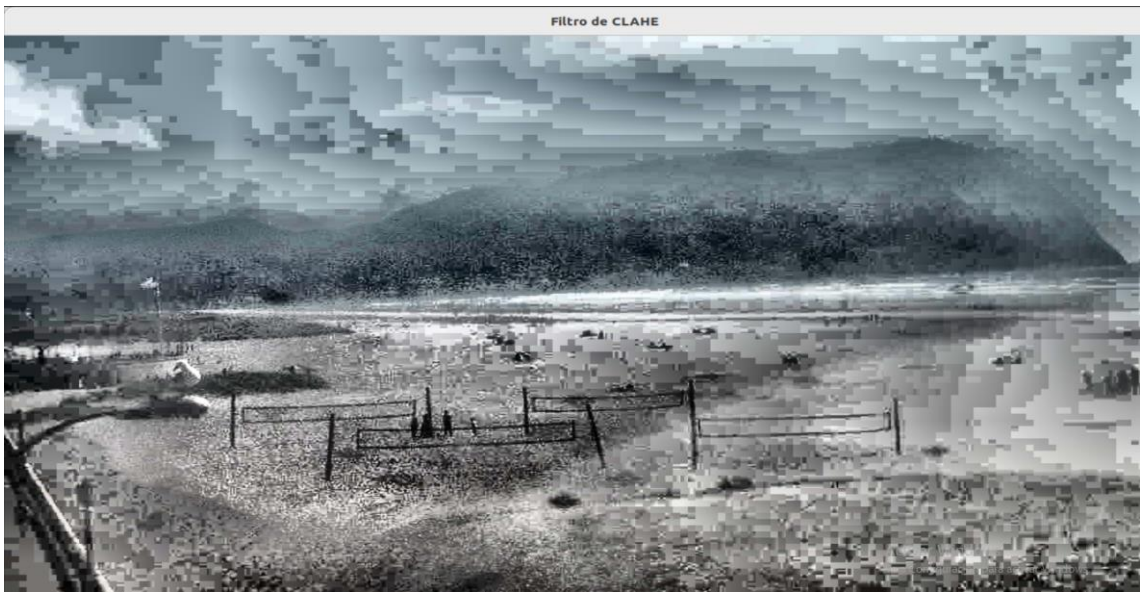
• Parte 1. Desarrollar una aplicación de escritorio que permita realizar la detección de movimiento usando la técnica de substracción adaptativa de fondo (Adaptive Background Substraction). Para ello, debe tomar en consideración lo siguiente:

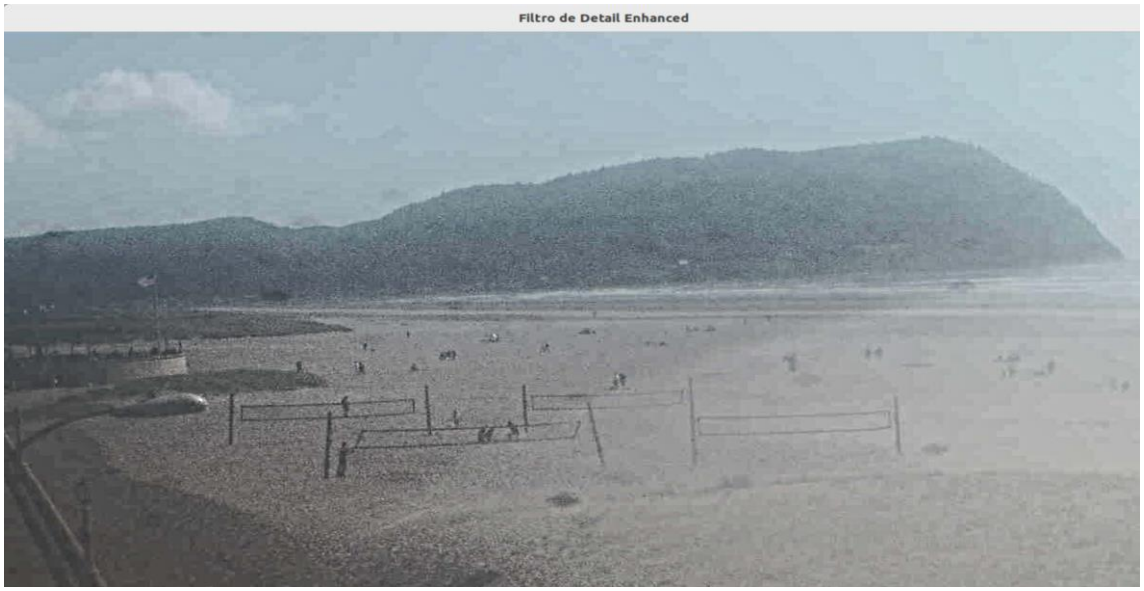
1. Programar un método que permita detectar el movimiento que se detecta en vídeos capturados de flujos de stream (cam live, cámaras remotas en vivo) a través de una aplicación de escritorio usando OpenCV/C++, de forma similar a cómo se hizo en clase.

2. Programar una función que permita detectar movimiento y mostrar los FPS en el vídeo original y los filtros u otras operaciones que aplique en los demás espacios disponibles. Para realizar esta tarea, deberá abrir un flujo de vídeo de Internet.

3. Aplique distintos filtros que permitan mejorar los problemas de iluminación vistos en clase. Para ello debe probar la ecualización de histograma, el método CLAHE y uno más que Usted investigue.







4. Explicar cómo funciona la técnica que ha investigado en comparación con la ecualización de histograma y el método CLAHE.

La técnica que investigue fue la función detailEnhance de OpenCV, dado que el live stream en sí no era de tan buena calidad y esta prometía mejorarlo mediante su metodo: *detailEnhance(Mat src, Mat dst, float sigma_s=10, float sigma_r=0.15f)* . Esta trabaja de manera en que toma una imagen de entrada y resalta sus detalles al aplicar un suavizado espacial controlado por el parámetro sigma_s, que determina la cantidad de difuminación de los píxeles, y una reducción de contraste controlada por sigma_r, que ajusta la intensidad del contraste local antes de realizar los detalles. La imagen de salida, representada por dst, muestra una mejora en los detalles de la imagen original, lo que puede mejorar la apariencia visual y hacer que los detalles sean más prominentes.

La técnica de detailEnhance y los métodos de ecualización de histograma y CLAHE son enfoques diferentes para mejorar la calidad visual de una imagen, cada uno con objetivos específicos. La ecualización de histograma mejora el contraste global de una imagen al redistribuir los valores de intensidad, haciéndola más uniforme y destacando detalles en áreas oscuras y claras. Sin embargo, puede sobreexponer algunas partes de la imagen. CLAHE, una versión mejorada de la ecualización de histograma, trabaja en pequeños bloques de la imagen y ajusta el contraste localmente, evitando la sobreexposición y manteniendo los detalles en áreas con diferentes niveles de brillo. Mientras que detailEnhance, está diseñado para resaltar los detalles finos y mejorar la nitidez de la imagen. A diferencia de los otros métodos, detailEnhance no mejora el contraste global, sino que se enfoca en hacer que la imagen se vea más nítida y definida.

SCRIPT:

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <chrono>
```

```
using namespace std;
using namespace cv;

inline void detailEnhance(const Mat& src, Mat& dst, float sigma_s = 10, float sigma_r = 0.15f) {
    Mat smoothed;
    bilateralFilter(src, smoothed, -1, sigma_s, sigma_r);

    subtract(src, smoothed, dst);
    dst = src + dst;
}

int main(int argc, char* args[]) {
    VideoCapture video("http://47.51.131.147/-wvhttp-01-
/GetOneShot?image_size=1280x720&frame_count=1000000000");

    if (!video.isOpened()) {
        cout << "No se puede abrir la cámara..." << endl;
        return -1;
    }

    Ptr<BackgroundSubtractorMOG2> pMOG2 = createBackgroundSubtractorMOG2();
    const int MAX_FRAMES = 1000;
    const double LEARNING_RATE = -1;

    Mat frame, motion_mask, equalized_frame, clahe_frame, detail_enhanced_frame;
    Mat background;

    auto start = chrono::steady_clock::now();
    int frameCount = 0;

    namedWindow("Video Original", WINDOW_FREERATIO);
    namedWindow("Deteccion de Movimiento", WINDOW_FREERATIO);
    namedWindow("Filtro de Equalizacion de Histograma", WINDOW_FREERATIO);
    namedWindow("Filtro de CLAHE", WINDOW_FREERATIO);
    namedWindow("Filtro de Detail Enhanced", WINDOW_FREERATIO);
```

```
    for (int t = 0; t < MAX_FRAMES; ++t) {
        video >> frame;

        pMOG2->apply(frame, motion_mask, LEARNING_RATE);

        cvtColor(frame, equalized_frame, COLOR_BGR2YUV);
        vector<Mat> channels;
        split(equalized_frame, channels);
        equalizeHist(channels[0], channels[0]);
        merge(channels, equalized_frame);
        cvtColor(equalized_frame, equalized_frame, COLOR_YUV2BGR);

        cvtColor(frame, clahe_frame, COLOR_BGR2Lab);
        vector<Mat> lab_channels;
        split(clahe_frame, lab_channels);
        Ptr<CLAHE> clahe = createCLAHE();
        clahe->apply(lab_channels[0], lab_channels[0]);
        merge(lab_channels, clahe_frame);
        cvtColor(clahe_frame, clahe_frame, COLOR_Lab2BGR);

        detailEnhance(frame, detail_enhanced_frame);

        auto end = chrono::steady_clock::now();
        double elapsed_seconds = chrono::duration_cast<chrono::duration<double>>(end -
start).count();
        if (elapsed_seconds >= 1.0) {
            double fps = frameCount / elapsed_seconds;
            frameCount = 0;
            start = chrono::steady_clock::now();
            putText(frame, "FPS: " + to_string(int(fps)), Point(10, 30), FONT_HERSHEY_SIMPLEX,
0.5, Scalar(0, 255, 0), 1);
        }
        frameCount++;

        imshow("Video Original", frame);
        imshow("Deteccion de Movimiento", motion_mask);
```

```
        imshow("Filtro de Equalizacion de Histograma", equalized_frame);
        imshow("Filtro de CLAHE", clahe_frame);
        imshow("Filtro de Detail Enhanced", detail_enhanced_frame);

        if (frame.empty() || waitKey(23) == 27) break;
    }

    destroyAllWindows();
    return 0;
}
```

5. Puede usar como referencia el tutorial "Motion Detection Techniques (With Code on OpenCV)" que explica 4 formas de detectar el movimiento:

• Tutorial:

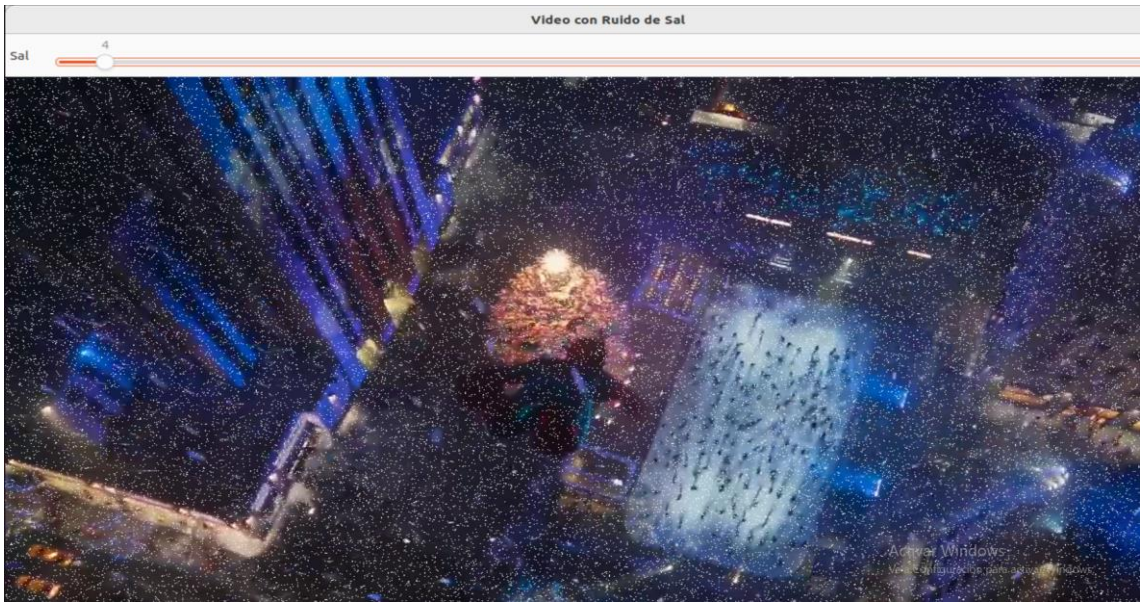
◦ https://medium.com/@abbessafa1998/motion-detection-techniques-with-code-on-opencv-18ed2c1acfaf

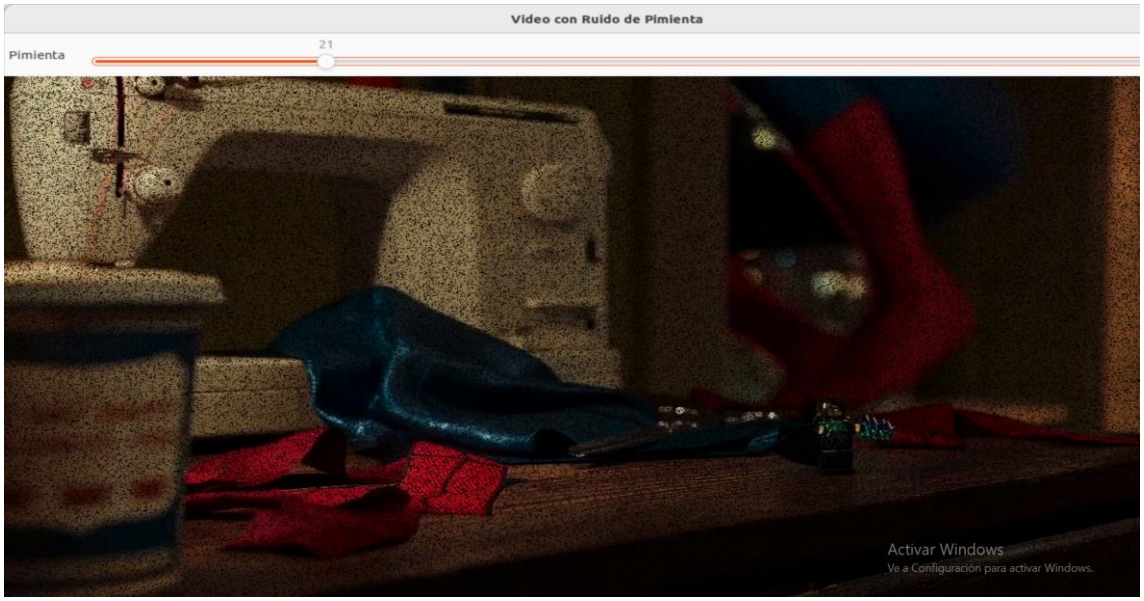
• GitHub:

◦ https://github.com/safaabbes/Motion-Detection-Techniques-using-OpenCV

• Parte 2. Desarrollar un programa de escritorio que permita generar ruido de sal y pimienta y aplicar filtros para reducir dicho ruido. Para ello, deberá llevar a cabo:

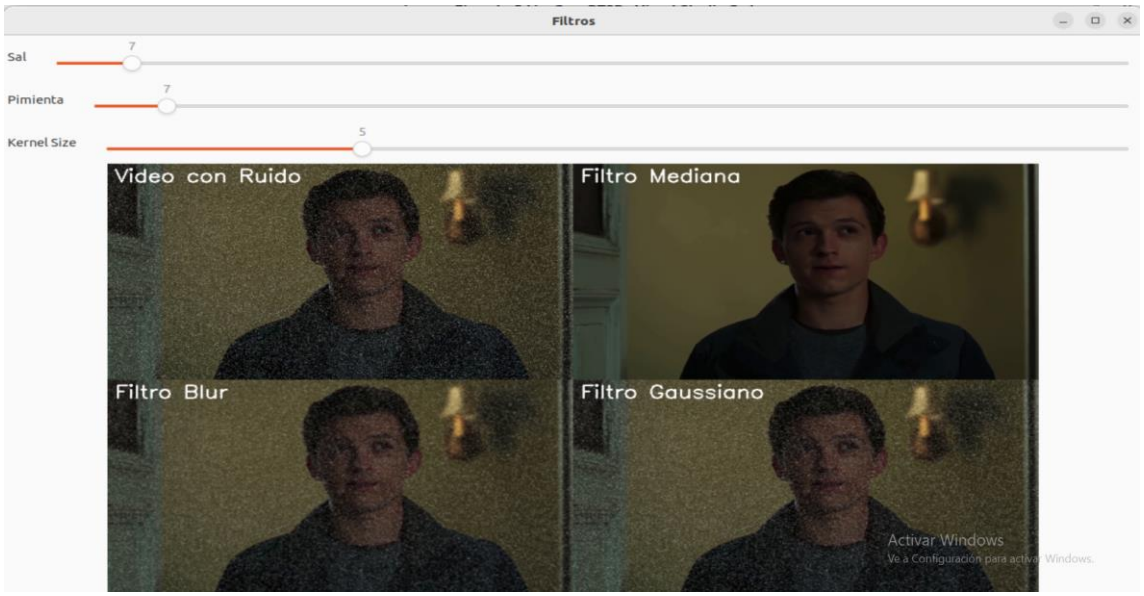
1. Programar un método que genere un porcentaje de ruido de sal o pimienta en un video a color, considerando las dimensiones del mismo. Se deberá poder ingresar un porcentaje de ruido a través de dos trackbars (uno para sal y otro para pimienta).





2. Programar una función para aplicar los siguientes filtros: mediana, blur, Gaussiano, probando con diferente tamaño de máscara.

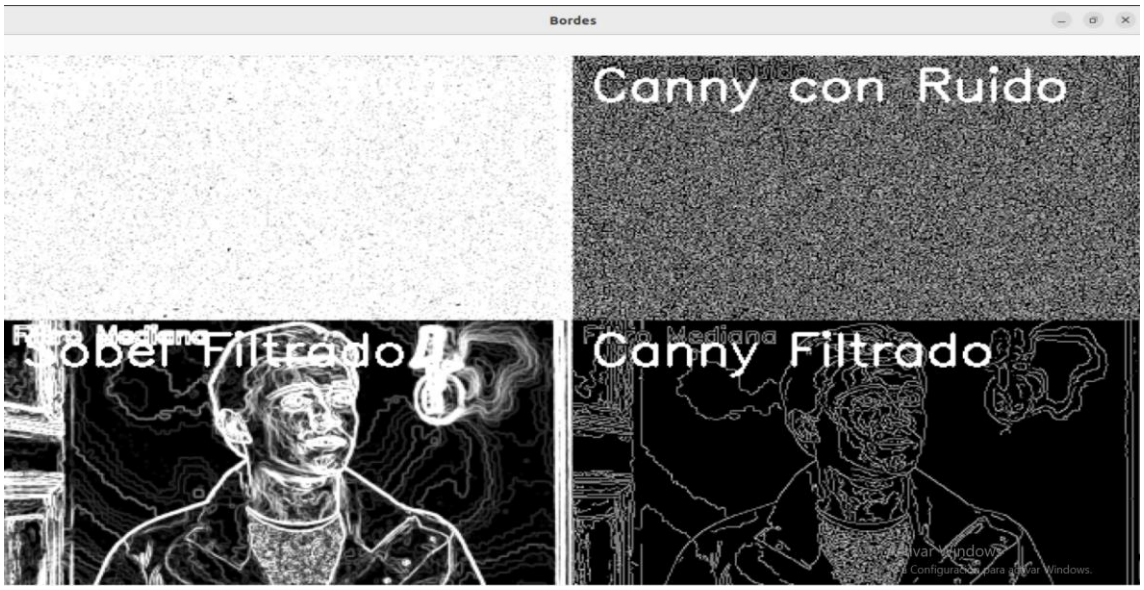
3. Compare los resultados obtenidos por cada filtro y muestre el resultado en una sola ventana (usando el método copyTo() de OpenCV), y reflexione cuál ha obtenido mejores resultados.



El filtro de la mediana se observa que es particularmente efectivo para suavizar el ruido de sal y pimienta, y esta se puede explicar debido a su forma de operar. Cuando se aplica ruido de sal y pimienta a una imagen, se agregan píxeles extremadamente brillantes (sal) o extremadamente oscuros (pimienta) de manera aleatoria. Estos píxeles atípicos pueden ser muy diferentes de los valores de los píxeles adyacentes, lo que provoca visuales indeseados. El filtro de mediana toma una ventana de píxeles vecinos y reemplaza el valor del píxel central con el valor mediano de todos los píxeles en la ventana. Esto significa que los valores extremadamente brillantes u oscuros introducidos por el ruido de sal y pimienta serán reemplazados por valores más cercanos a los de los píxeles circundantes. Como resultado, el ruido se suaviza sin distorsionar

significativamente la imagen, lo que hace que el filtro de mediana sea muy efectivo para eliminar el ruido de sal y pimienta mientras conserva los detalles de la imagen.

4. Aplicar al menos 2 algoritmos de detección de bordes y comparar los resultados de usar o no filtros de suavizado.



SCRIPT:

```
#include <iostream>
#include <opencv2/opencv.hpp>

using namespace std;
using namespace cv;

int main() {
    VideoCapture cap("/home/kelly/Descargas/spidey.mp4");
    if (!cap.isOpened()) {
        cout << "Error al abrir el video." << endl;
        return -1;
    }

    Mat frame;
    namedWindow("Filtros", WINDOW_FREERATIO);
```

```
namedWindow("Bordes", WINDOW_FREERATIO);

int porcentaje_sal = 0;
int porcentaje_pimienta = 0;
int kernel_size = 3;

createTrackbar("Sal", "Filtros", &porcentaje_sal, 100);
createTrackbar("Pimienta", "Filtros", &porcentaje_pimienta, 100);

createTrackbar("Kernel Size", "Filtros", &kernel_size, 20, [](int value, void*){
    if (value % 2 == 0) {
        value++;
        setTrackbarPos("Kernel Size", "Filtros", value);
    }
});

while (true) {
    cap >> frame;
    if (frame.empty()) break;

    Mat ruido_frame = frame.clone();

    int total_pixels = frame.rows * frame.cols;
    int sal_pixels = total_pixels * porcentaje_sal / 100;
    int pimienta_pixels = total_pixels * porcentaje_pimienta / 100;

    for (int i = 0; i < sal_pixels; ++i) {
        int row = rand() % frame.rows;
        int col = rand() % frame.cols;
        ruido_frame.at<Vec3b>(row, col) = Vec3b(255, 255, 255);
    }

    for (int i = 0; i < pimienta_pixels; ++i) {
```

```
            int row = rand() % frame.rows;
            int col = rand() % frame.cols;
            ruido_frame.at<Vec3b>(row, col) = Vec3b(0, 0, 0);
        }

        Mat filtro_mediana;
        medianBlur(ruido_frame, filtro_mediana, kernel_size);

        Mat filtro_blur;
        blur(ruido_frame, filtro_blur, Size(kernel_size, kernel_size));

        Mat filtro_gaussiano;
        GaussianBlur(ruido_frame, filtro_gaussiano, Size(kernel_size, kernel_size), 0);

        resize(ruido_frame, ruido_frame, Size(ruido_frame.cols / 2, ruido_frame.rows / 2));
        resize(filtro_mediana, filtro_mediana, Size(filtro_mediana.cols / 2, filtro_mediana.rows / 2));
        resize(filtro_blur, filtro_blur, Size(filtro_blur.cols / 2, filtro_blur.rows / 2));
        resize(filtro_gaussiano, filtro_gaussiano, Size(filtro_gaussiano.cols / 2,
filtro_gaussiano.rows / 2));

        putText(ruido_frame, "Video con Ruido", Point(10, 30), FONT_HERSHEY_SIMPLEX, 1,
Scalar(255, 255, 255), 2);
        putText(filtro_mediana, "Filtro Mediana", Point(10, 30), FONT_HERSHEY_SIMPLEX, 1,
Scalar(255, 255, 255), 2);
        putText(filtro_blur, "Filtro Blur", Point(10, 30), FONT_HERSHEY_SIMPLEX, 1, Scalar(255,
255, 255), 2);
        putText(filtro_gaussiano, "Filtro Gaussiano", Point(10, 30), FONT_HERSHEY_SIMPLEX,
1, Scalar(255, 255, 255), 2);

        Mat ventana_filtros(ruido_frame.rows * 2, ruido_frame.cols * 2, ruido_frame.type());
        ruido_frame.copyTo(ventana_filtros(Rect(0, 0, ruido_frame.cols, ruido_frame.rows)));
        filtro_mediana.copyTo(ventana_filtros(Rect(ruido_frame.cols, 0, filtro_mediana.cols,
filtro_mediana.rows)));
        filtro_blur.copyTo(ventana_filtros(Rect(0, ruido_frame.rows, filtro_blur.cols,
filtro_blur.rows)));
```

```
        filtro_gaussiano.copyTo(ventana_filtros(Rect(filtro_blur.cols, ruido_frame.rows,
filtro_gaussiano.cols, filtro_gaussiano.rows)));

        resizeWindow("Filtros", ventana_filtros.cols, ventana_filtros.rows);

        imshow("Filtros", ventana_filtros);

        Mat ruido_gray;
        cvtColor(ruido_frame, ruido_gray, COLOR_BGR2GRAY);

        Mat filtro_mediana_gray;
        cvtColor(filtro_mediana, filtro_mediana_gray, COLOR_BGR2GRAY);

        Mat sobel_x_ruido, sobel_y_ruido, sobel_ruido;
        Sobel(ruido_gray, sobel_x_ruido, CV_64F, 1, 0, kernel_size);
        Sobel(ruido_gray, sobel_y_ruido, CV_64F, 0, 1, kernel_size);
        magnitude(sobel_x_ruido, sobel_y_ruido, sobel_ruido);
        sobel_ruido.convertTo(sobel_ruido, CV_8U);

        Mat canny_ruido;
        Canny(ruido_gray, canny_ruido, 100, 200, kernel_size);

        Mat sobel_x_mediana, sobel_y_mediana, sobel_mediana;
        Sobel(filtro_mediana_gray, sobel_x_mediana, CV_64F, 1, 0, kernel_size);
        Sobel(filtro_mediana_gray, sobel_y_mediana, CV_64F, 0, 1, kernel_size);
        magnitude(sobel_x_mediana, sobel_y_mediana, sobel_mediana);
        sobel_mediana.convertTo(sobel_mediana, CV_8U);

        Mat canny_mediana;
        Canny(filtro_mediana_gray, canny_mediana, 100, 200, kernel_size);
        resize(sobel_ruido, sobel_ruido, Size(sobel_ruido.cols / 2, sobel_ruido.rows / 2));
        resize(canny_ruido, canny_ruido, Size(canny_ruido.cols / 2, canny_ruido.rows / 2));
        resize(sobel_mediana, sobel_mediana, Size(sobel_mediana.cols / 2, sobel_mediana.rows
/ 2));
```

```
        resize(canny_mediana, canny_mediana, Size(canny_mediana.cols / 2,
canny_mediana.rows / 2));


        Mat sobel_ruido_bgr, canny_ruido_bgr, sobel_mediana_bgr, canny_mediana_bgr;

        cvtColor(sobel_ruido, sobel_ruido_bgr, COLOR_GRAY2BGR);

        cvtColor(canny_ruido, canny_ruido_bgr, COLOR_GRAY2BGR);

        cvtColor(sobel_mediana, sobel_mediana_bgr, COLOR_GRAY2BGR);

        cvtColor(canny_mediana, canny_mediana_bgr, COLOR_GRAY2BGR);


        putText(sobel_ruido_bgr, "Sobel con Ruido", Point(10, 30), FONT_HERSHEY_SIMPLEX,
1, Scalar(255, 255, 255), 2);

        putText(canny_ruido_bgr, "Canny con Ruido", Point(10, 30), FONT_HERSHEY_SIMPLEX,
1, Scalar(255, 255, 255), 2);

        putText(sobel_mediana_bgr, "Sobel Filtrado", Point(10, 30), FONT_HERSHEY_SIMPLEX,
1, Scalar(255, 255, 255), 2);

        putText(canny_mediana_bgr, "Canny Filtrado", Point(10, 30),
FONT_HERSHEY_SIMPLEX, 1, Scalar(255, 255, 255), 2);


        Mat ventana_bordes(sobel_ruido_bgr.rows * 2, sobel_ruido_bgr.cols * 2,
sobel_ruido_bgr.type());

        sobel_ruido_bgr.copyTo(ventana_bordes(Rect(0, 0, sobel_ruido_bgr.cols,
sobel_ruido_bgr.rows)));

        canny_ruido_bgr.copyTo(ventana_bordes(Rect(sobel_ruido_bgr.cols, 0,
canny_ruido_bgr.cols, canny_ruido_bgr.rows)));

        sobel_mediana_bgr.copyTo(ventana_bordes(Rect(0, sobel_ruido_bgr.rows,
sobel_mediana_bgr.cols, sobel_mediana_bgr.rows)));

        canny_mediana_bgr.copyTo(ventana_bordes(Rect(sobel_mediana_bgr.cols,
sobel_ruido_bgr.rows, canny_mediana_bgr.cols, canny_mediana_bgr.rows)));


        resizeWindow("Bordes", ventana_bordes.cols, ventana_bordes.rows);


        imshow("Bordes", ventana_bordes);


        char c = waitKey(25);

        if (c == 27) break;

    }
```

```
    cap.release();
    destroyAllWindows();
    return 0;
}
```

• Parte 3. Desarrollar un programa de escritorio que permita aplicar operaciones morfológicas para mejorar la calidad de imágenes médicas, para ello deberá realizar las siguientes tareas:

1. Seleccionar 3 imágenes médicas a las que se les aplicarán las operaciones morfológicas. Las imágenes deben estar en escala de grises y deben corresponder a radiografías, angiografías, TACs, etc.

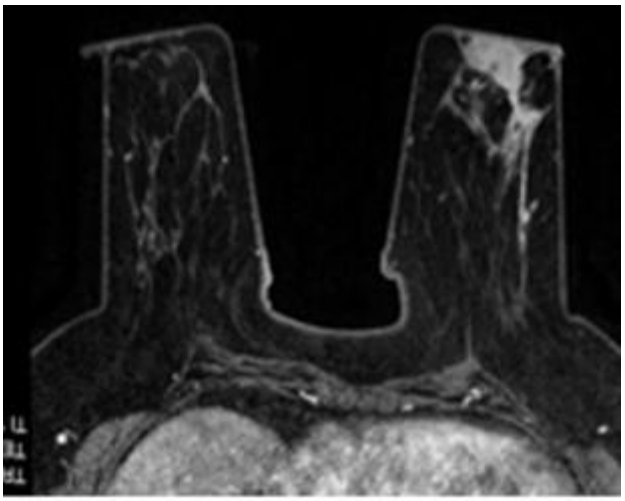


Fig. 1. La resonancia magnética DCE muestra una masa retroareolar con realce no homogéneo y márgenes espiculados.

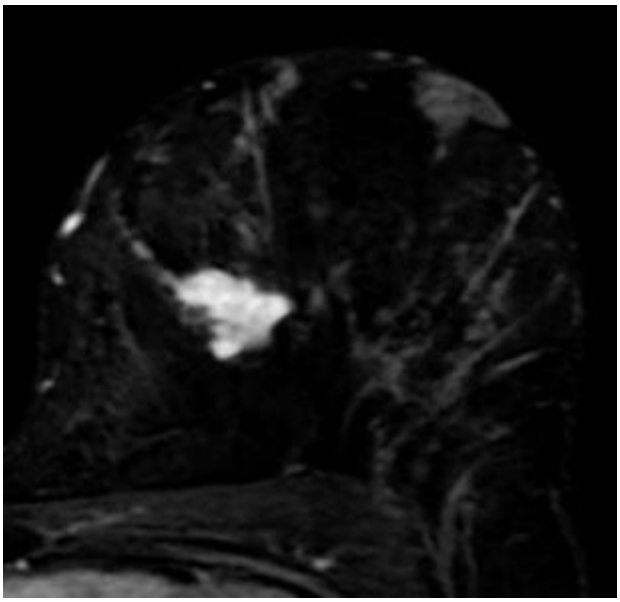


Fig. 2. DCE-MRI muestra una lesión de masa de contorno irregular que realza intensamente en el cuadrante superior interno de la mama izquierda con una cola

curvilínea de tejido que realza anormalmente y que se ve dirigida en dirección anteromedial.

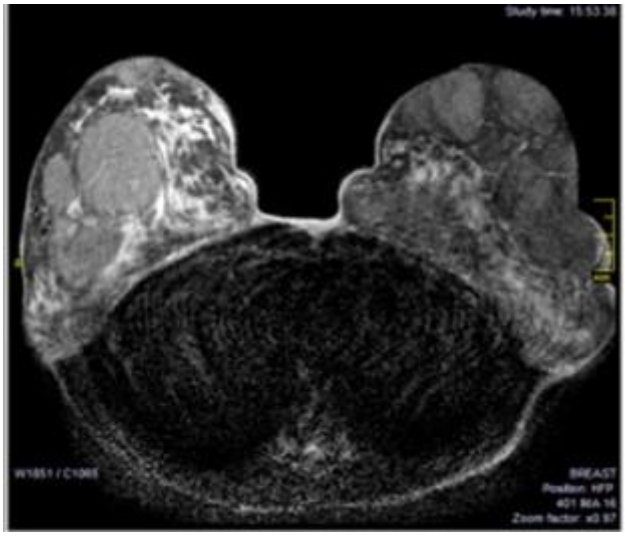


Fig. 3. Las imágenes axiales en T2W muestran múltiples lesiones bilaterales, ovaladas y globulares, bien definidas, de tamaño variable, que varían en tamaño desde unos pocos mm hasta un máximo de 7,5 cm de diámetro y exhiben una intensidad de señal intermedia.

SCRIPT:

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <vector>

using namespace std;
using namespace cv;

void applyMorphologicalOperations(const  Mat& img, const Size& kernel_size) {
   Mat kernel = getStructuringElement(MORPH_RECT,  kernel_size);

   Mat erosion;
   erode(img, erosion, kernel, Point(-1, -1), 1);

   Mat dilation;
   dilate(img, dilation, kernel, Point(-1, -1), 1);
```

```
    Mat tophat;
    morphologyEx(img, tophat, MORPH_TOPHAT, kernel);

    Mat blackhat;
    morphologyEx(img, blackhat, MORPH_BLACKHAT, kernel);

    Mat ecuacion;
    add(img, tophat - blackhat, ecuacion);

    imshow("Original (Kernel " + to_string(kernel_size.width) + "x" +
to_string(kernel_size.height) + ")", img);
    imshow("Erosión (Kernel " + to_string(kernel_size.width) + "x" +
to_string(kernel_size.height) + ")", erosion);
    imshow("Dilatación (Kernel " + to_string(kernel_size.width) + "x" +
to_string(kernel_size.height) + ")", dilation);
    imshow("Top Hat (Kernel " + to_string(kernel_size.width) + "x" +
to_string(kernel_size.height) + ")", tophat);
    imshow("Black Hat (Kernel " + to_string(kernel_size.width) + "x" +
to_string(kernel_size.height) + ")", blackhat);
    imshow("Ecuacion (Kernel " + to_string(kernel_size.width) + "x" +
to_string(kernel_size.height) + ")", ecuacion);

    waitKey(0);
    destroyAllWindows();
}

int main() {
    vector<string> image_files = {"/home/kelly/Descargas/imagen1.jpg",
"/home/kelly/Descargas/imagen2.jpg", "/home/kelly/Descargas/imagen3.jpg"};
    vector<Size> kernel_sizes = {Size(35, 35), Size(37, 37), Size(39, 39)};

    for (const string& file : image_files) {
        Mat img = imread(file, IMREAD_GRAYSCALE);
        if (img.empty()) {
            cout << "El archivo " << file << " no pudo ser leído, verifica la ruta" << endl;
            continue;
        }
```

```
        for (const Size& kernel_size : kernel_sizes) {
            applyMorphologicalOperations(img,  kernel_size);
        }
    }

    return 0;
}
```

2. Aplicar las siguientes operaciones sobre las imágenes, probando al menos 3 tamaños de máscaras (de tamaño aproximado de 37x37, como se sugiere en el artículo "Using morphological transforms to enhance the contrast of medical images1"):

a) Erosión

b) Dilatación

c) Top Hat

d) Black Hat

e) Imagen Original + (Top Hat – Black Hat)

Figura 1 (Kernel 35 x 35)

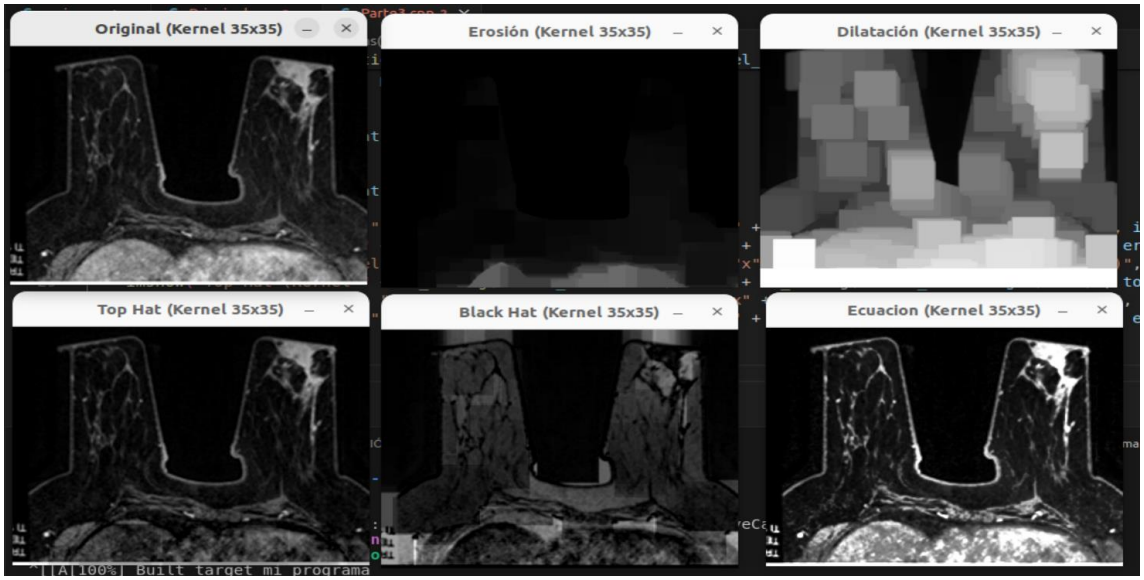


Figura 1 (Kernel 37 x 37)

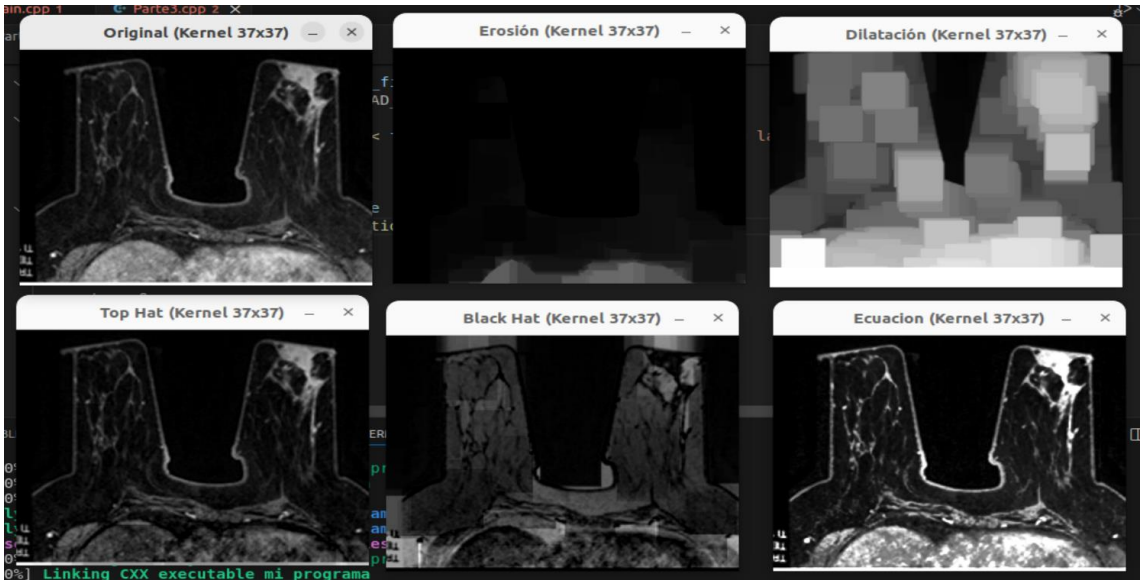


Figura 1 (Kernel 39 x 39)

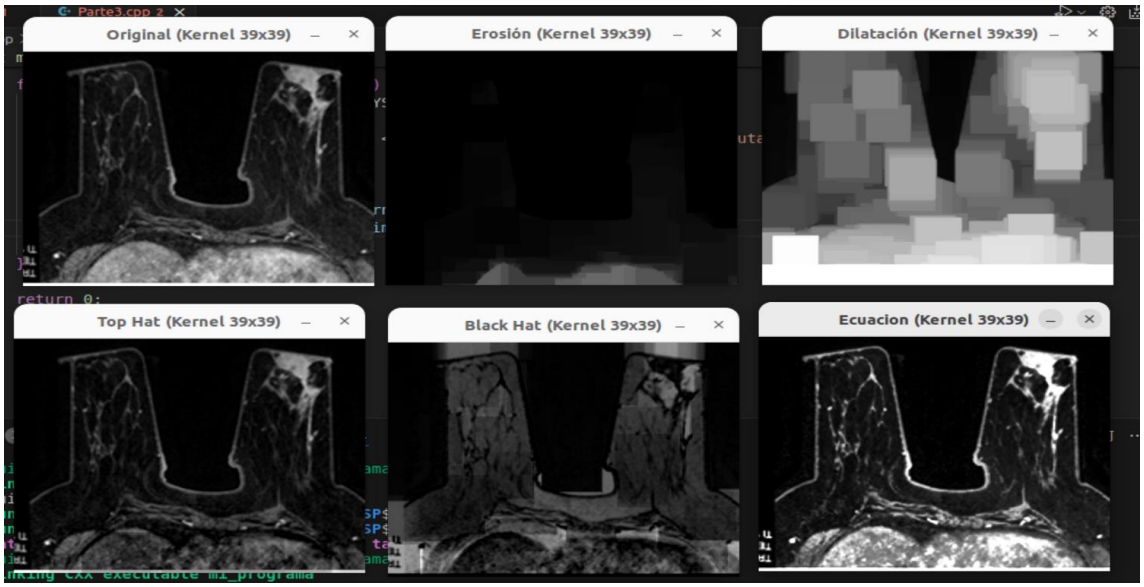


Figura 2 (Kernel 35 x 35)

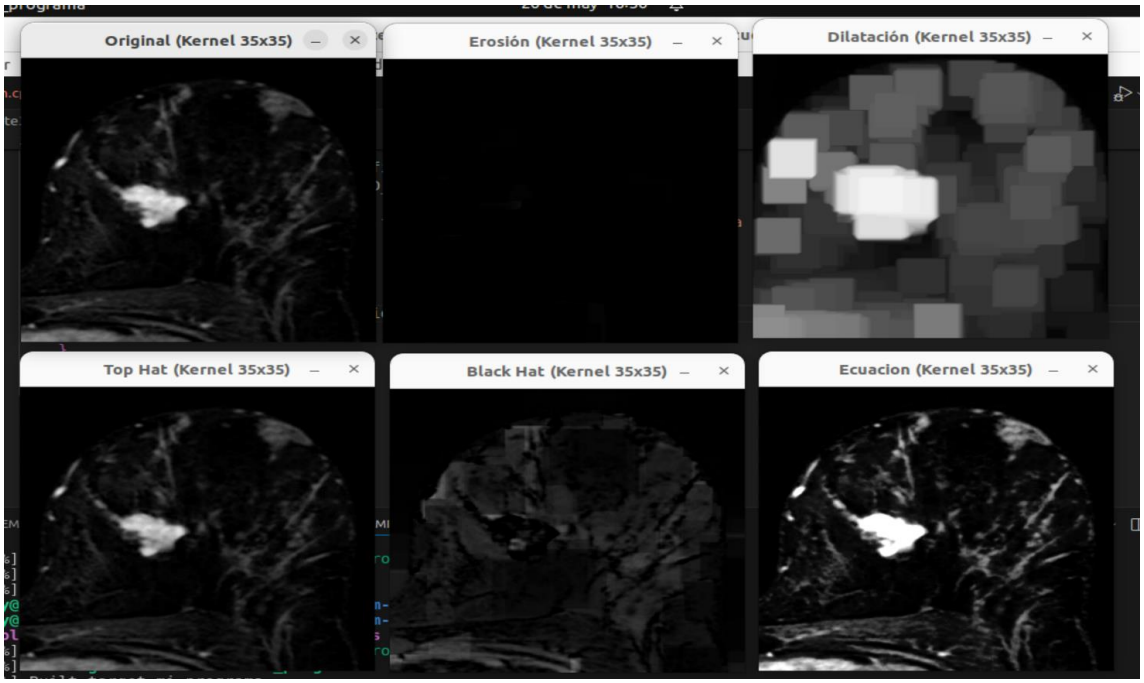


Figura 2 (Kernel 37 x 37)

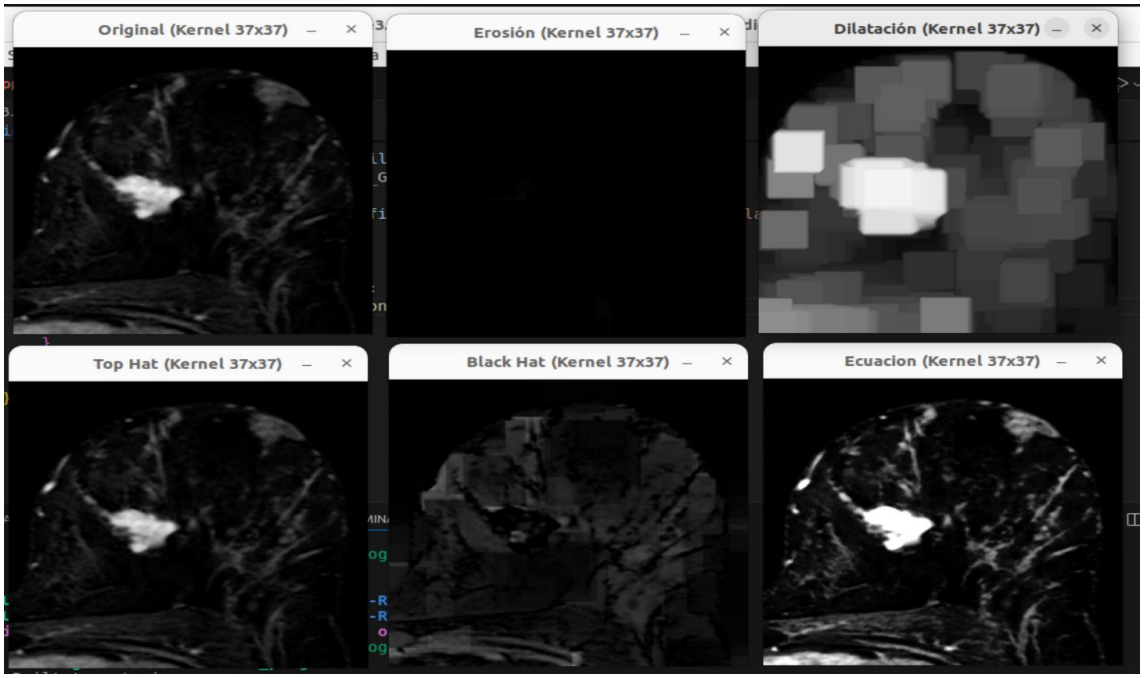


Figura 2 (Kernel 39 x 39)

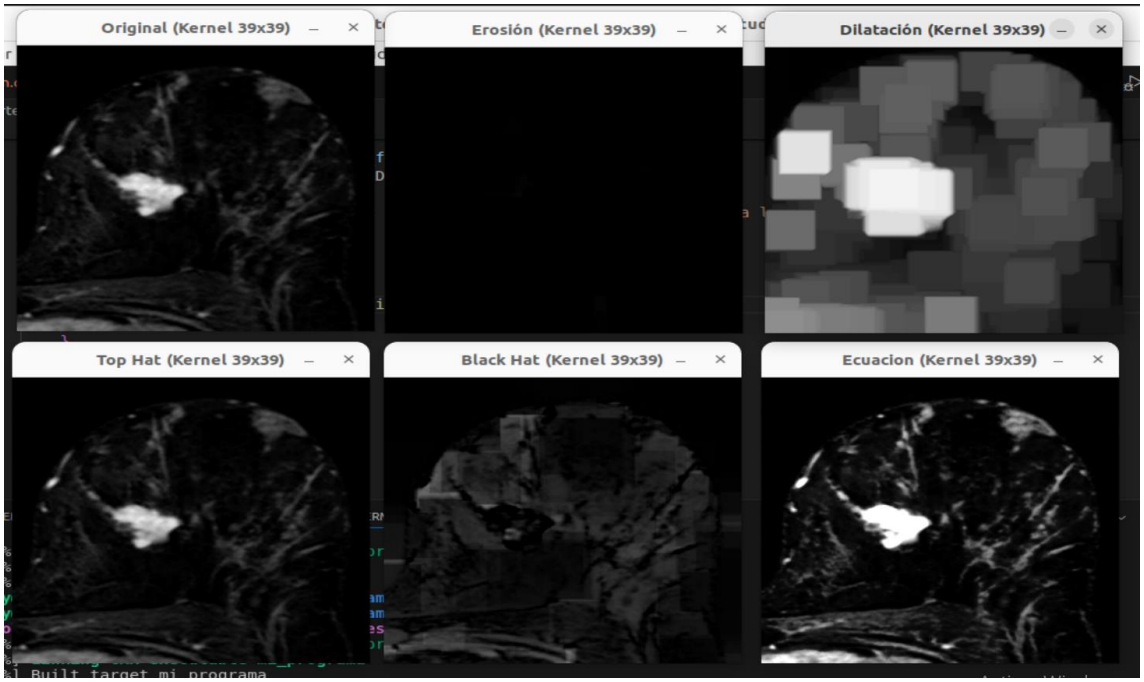


Figura 3 (Kernel 35 x 35)

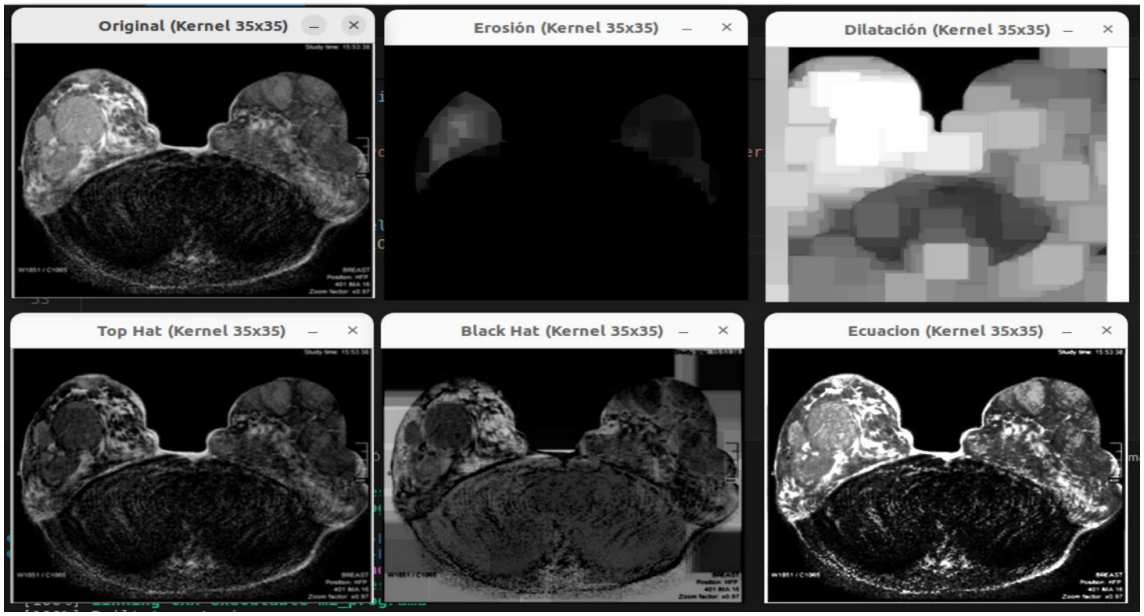


Figura 3 (Kernel 37 x 37)

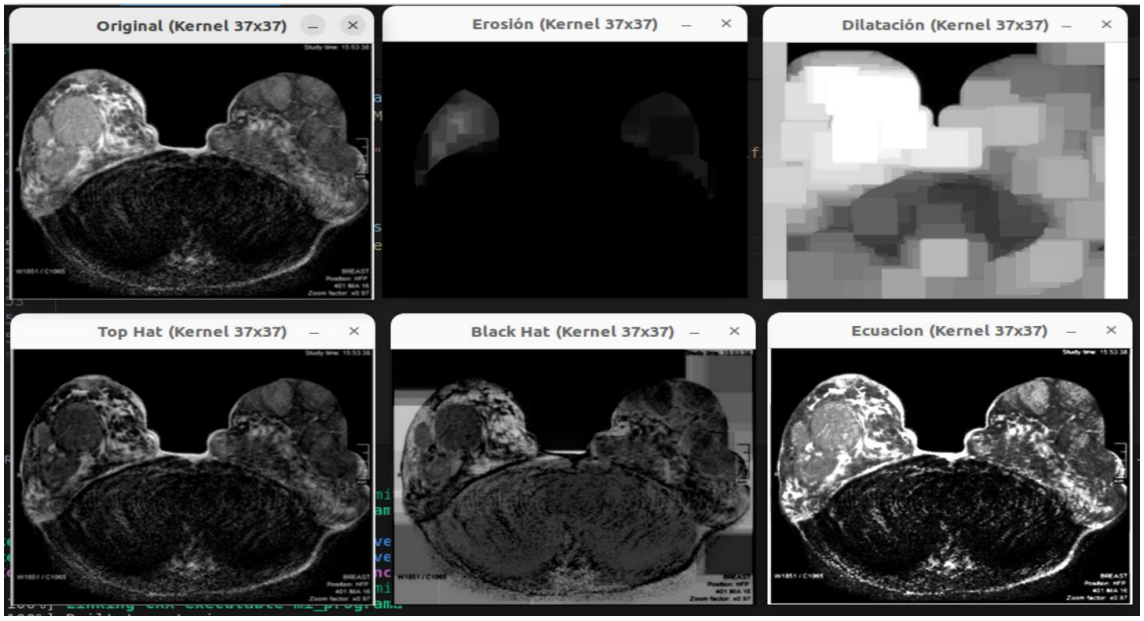
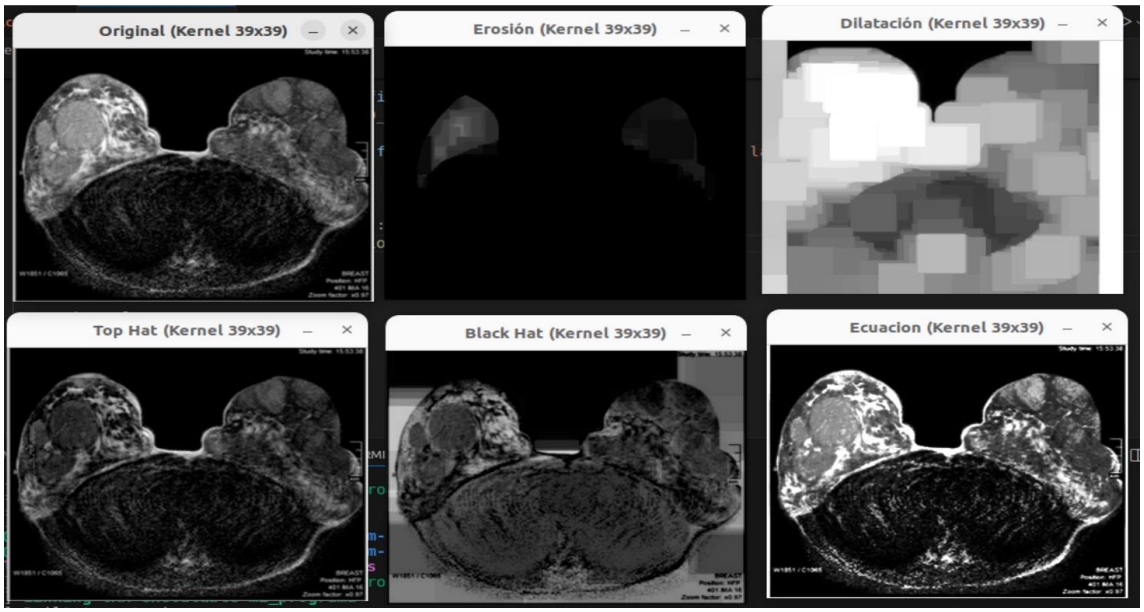


Figura 3 (Kernel 39 x 39)



3. Compare los resultados de la imagen original con respecto a las que tienen operaciones morfológicas y reflexione sobre su nitidez y si mejora la posibilidad de observar mejor los objetos.

Al aplicar las operaciones morfológicas y observar las imágenes resultantes, se puede reflexionar sobre cómo la nitidez cambia en:

- Erosión y Dilatación: Estas operaciones pueden ayudar a limpiar el ruido y definir mejor los bordes, pero también pueden eliminar detalles finos o hacer que los bordes se vean demasiado gruesos. En caso de estas imágenes medicas

se observa como en estas operaciones bien se elimina casi todo el detalle o se sobreexpande los detalles que en sí son finas, también este resultado es dado por el tamaño del kernel.

- Top Hat y Black Hat: Estas operaciones pueden resaltar detalles finos y aumentar el contraste local, mejorando la percepción de nitidez.
- Ecuación: La combinación de la imagen orignal + (Top Hat - Black Hat) aumenta significativamente la nitidez al mejorar los detalles y el contraste, permitiendo el equilibrio adecuado para mejorar la calidad de las imágenes médicas

Con las operaciones morfológicas trabajadas con, especialmente el Top Hat, Black Hat y Ecuación, presentan métodos que utiliza transformadas morfológicas para mejorar la calidad y el contraste de las imágenes médicas. Además de su simplicidad, este método tiene una alta potencialidad para procesar imágenes médicas de mala calidad. Este enfoque puede especificar órganos arbitrarios superpuestos en imágenes médicas para un mejor diagnóstico. (Hamid Hassanpour, 2015)

Bibliografía:

- Abbes, S. (2022, junio 23). Motion detection techniques (with code on OpenCV) - Safa abbes. Medium. https://medium.com/@abbessafa1998/motion-detection-techniques-with-code-on-opencv-18ed2c1acfaf
- Hamid Hassanpour, Najmeh Samadiani, S.M. Mahdi Salehi. (2015). Using morphological transforms to enhance the contrast of medical images. The Egyptian Journal of Radiology and Nuclear Medicine, 46(2), 481–489. https://doi.org/10.1016/j.ejrnm.2015.01.004
- Lobkov, G. (2018). Public RTSP sources as example #3. GitHub. https://github.com/grigory-lobkov/rtsp-camera-view/issues/3
- Mallick, S. (2015, marzo 21). Non-Photorealistic Rendering using OpenCV (Python, C++). LearnOpenCV – Learn OpenCV, PyTorch, Keras, Tensorflow with Code, & Tutorials; Satya Mallick. https://learnopencv.com/non-photorealistic-rendering-using-opencv-python-c/
- Yosra Abdelzaher Ibrahim, Lobna Habib, Ahmed Deif. (2015). Role of quantitative diffusion weighted imaging in characterization of breast masses. The Egyptian Journal of Radiology and Nuclear Medicine, 46(3), 805–810. https://doi.org/10.1016/j.ejrnm.2015.05.006