

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO UNIVERSITARIO NORTE DO ESPÍRITO SANTO

Emerson Patryck da Silva

Prof.: Oberlan Christo Romao

RELATÓRIO EXERCICIO DE PROGRAMAÇÃO I
FOTOXOP

SÃO MATEUS – ES

2021

DEFINIÇÃO DE IMAGEM

Na área de processamento de imagem usa-se o modelo matricial para trabalhar com manipulações de imagem. Uma imagem é definida como uma matriz de pixels, o tamanho dessa matriz indica as dimensões da imagem e cada posição desta matriz indica a intensidade da cor de pixel que possui valores de 0 – 255, onde 0 indica ausência de cor e 255 a presença total da cor. Uma forma básica de imagem é aquela formada pelas cores preto e branco, é a forma mais básica de imagem são chamadas de imagem monocromática. “Uma imagem monocromática é uma função bidimensional onde x e y são coordenadas espaciais ou posições na matriz de pixel, a função $f(x,y)$ indica a intensidade luminosa na posição (x,y) .”(JER de Queiroz, HM Gomes - Rita, 2006 – pág. 8)

Por outro lado, nas imagens cromáticas, ou seja, imagens que possuem três componentes de cores, a intensidade de um pixel é dada por um vetor que contém a intensidade de cada cor, no padrão RGB, esse vetor contém as informações sobre intensidade da cor vermelha, cor verde e da cor azul.

Imagen cromática pode ser composta como sendo um conjunto de imagens monocromáticas, isso quer dizer que uma imagem cromática é formada pela soma de imagens monocromáticas e como imagens monocromáticas é dada por uma função bidimensional, então:

$$f(x,y) = f_R(x,y) + f_G(x,y) + f_B(x,y)$$

Como pode ser visto a representação gráfica abaixo:

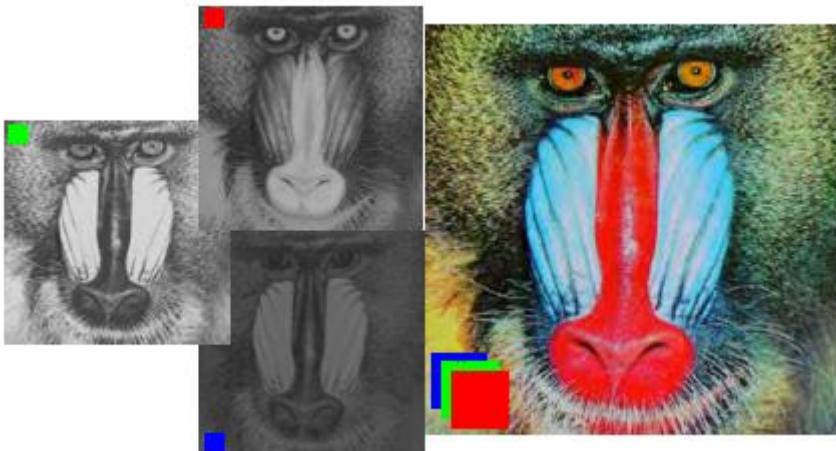


Figura 1 - Representação de uma imagem bidimensional.

FILTROS

Filtros de imagens são caracterizados pela alteração dos valores de pixel na matriz da imagem. Existem inúmeros filtros aplicáveis em imagens onde cada faz um tipo de modificação no pixel resultando numa modificação única na imagem. Alguns filtros podem ser utilizados somente para estética, outros filtros podem possuir base para utilização em outras áreas de estudos, como filtros de detecção de bordas utilizados na detecção de objetos. Mas para base do nosso estudo vamos somente falar sobre os filtros utilizados na implementação do projeto.

FILTRO DE ESCURECIMENTO

Filtro de escurecimento basicamente escurece uma imagem o quanto quisermos. Esse filtro altera proporcionalmente cada pixel de imagens, diminuindo o brilho da mesma.

O código abaixo está implementado em linguagem C e é a representação do filtro de escurecimento.

```
void escurecerImagem(Imagen *img)
{
    int v;
    printf("Digite o fator de escurecimento (0-255): ");
    scanf("%d", &v);

    for (int h = 0; h < obtemAltura(img); h++)
    {
        for (int w = 0; w < obtemLargura(img); w++)
        {
            Pixel pixel = obtemPixel(img, h, w);
            pixel.cor[RED] = (((int)pixel.cor[RED] - v) >= 0 ? (pixel.cor[RED] - v) : 0);
            pixel.cor[GREEN] = (((int)pixel.cor[GREEN] - v) >= 0 ? (pixel.cor[GREEN] - v) : 0);
            pixel.cor[BLUE] = (((int)pixel.cor[BLUE] - v) >= 0 ? (pixel.cor[BLUE] - v) : 0);
            recolorePixel(img, h, w, pixel);
        }
    }
}
```

O código acima requer que o usuário entre com um valor considerado fator de escurecimento. Esse valor varia de 0 a 255 que é a faixa de valor que um pixel pode assumir. O valor 0 não faz nenhum efeito na imagem, já o valor de 255 escurece totalmente a imagem, ou seja, a imagem original se torna uma imagem de cor preta.



Figura 2 - Imagem original, sem aplicação de algum fator.



Figura 3 - Imagem escurecida com o valor de fator igual à 0.

Note que não ouve nenhuma alteração quando aplicamos o valor de fator 0, isso é axiomático pois qualquer valor subtraído por zero é ele próprio, então temos uma “cópia da imagem original”.

Agora aplicando o valor de fator 255, temos:



Figura 4 - Imagem escurecida com o valor de fator igual à 255.

Quando o usuário entra com o valor do escurecimento, o algoritmo vai percorrer toda a matriz, subtraindo cada cor de pixel pelo valor do escurecimento, em seguida verifica se o valor está dentro da faixa de valor de um pixel que é de 0 a 255, lembrando que os valores de pixel não podem exceder o valor mínimo que é 0 e o valor máximo que é 255. Sempre haverá essa verificação.

Após a verificação, o canal RGB modificado substitui o canal RGB original da imagem com os valores de cada pixel alterado.

Aplicando um fator de valor 100, obtemos os seguintes resultados:



Figura 5 - A imagem acima temos a aplicação do filtro escurecer e na parte de baixo temos a imagem original.



Figura 6 - A imagem da esquerda é escurecida com fator 100 e a imagem da direita é imagem original.

Outro exemplo:



Figura 5 - A imagem da esquerda é escurecida com fator 100 e a imagem da direita é imagem original.

FILTRO DE CLAREAMENTO

Agora vamos falar sobre o filtro que pode ser considerado o inverso do filtro de escurecimento. Como foi dada a proposta acima, sendo o inverso do filtro de escurecimento, o filtro de clareamento tem por funcionalidade adicionar mais brilho a imagem, então isso quer dizer que ao invés de simplesmente subtrair o fator aplicado, simplesmente somamos com o valor do pixel.

O algoritmo para realizar esse tipo de operação é similar ao do filtro de escurecimento.

```
void clarearImagem(Imagen *img)
{
    int v;
    printf("Digite o fator de clareamento (0-255): ");
    scanf("%d", &v);
    for (int h = 0; h < obtemAltura(img); h++)
    {
        for (int w = 0; w < obtemLargura(img); w++)
        {
            Pixel pixel = obtemPixel(img, h, w);
            pixel.cor[RED] = (((int)pixel.cor[RED] + v) > 255 ? 255 : (pixel.cor[RED] + v));
            pixel.cor[GREEN] = (((int)pixel.cor[GREEN] + v) > 255 ? 255 : (pixel.cor[GREEN] + v));
            pixel.cor[BLUE] = (((int)pixel.cor[BLUE] + v) > 255 ? 255 : (pixel.cor[BLUE] + v));
            recolorePixel(img, h, w, pixel);
        }
    }
}
```

O usuário entra com o valor de clareamento do pixel onde varia de 0 a 255, em que 0 não aplica nenhum clareamento e 255 aplica o clareamento total. O valor do fator é somado pixel a pixel e depois verificado, como já sabemos o valor máximo do pixel é 255 e o valor mínimo é 0, se está abaixo de 255. O pixel verificado é atribuído para recoloração da imagem, assim aplicando o efeito de clareamento.

Para fator 0, temos uma imagem idêntica a original.



Figura 6 - Imagem original.



Figura 7 - Imagem clareada com fator 0.

Quando ajustamos o valor para 255, a imagem fica totalmente branca.



Figura 8 - Imagem clareada com fator 255.

Como podemos ajustar o valor entre 0 a 255, segui o mesmo passo do filtro de escurecimento aplicando um fator 100 para cada imagem.



Figura 9 - Acima temos a imagem com filtro clareado com fator 100 e abaixo a imagem original.



Figura 10 - A imagem da esquerda foi aplicada o filtro com fator 100. A imagem da direita é a imagem original sem filtro. (1)



Figura 11 - A imagem da esquerda foi aplicada o filtro com fator 100. A imagem da direita é a imagem original sem filtro. (2)

FILTRO ESCALA DE CINZA

O filtro de escala de cinza torna a imagem original colorida em uma imagem na cor cinza. A ideia do implemento é o mesmo dos outros filtros, relativamente é um filtro mais simples pois não precisa que o usuário entre com algum tipo de valor para fator e consequentemente não precisa realizar algumas verificações como são feitas nos filtros mencionados antes. Abaixo podemos ver a implementação do filtro:

```
void escalaDeCinzaImagem(Imagen *img)
{
    int media;
    for (int h = 0; h < obtemAltura(img); h++)
    {
        for (int w = 0; w < obtemLargura(img); w++)
        {
            Pixel pixel = obtemPixel(img, h, w);
            media = ((int)pixel.cor[RED] + (int)pixel.cor[GREEN] + (int)pixel.cor[BLUE]) / 3;
            pixel.cor[RED] = media;
            pixel.cor[GREEN] = media;
            pixel.cor[BLUE] = media;
            recolorePixel(img, h, w, pixel);
        }
    }
}
```

Perceba que o filtro de escala de cinza usa somente média aritmética para realizar os cálculos. Essa média é composta pelos valores dos pixels de cada cor, perceba também que esse valor da média nunca será maior que 255 e menor que 0, assim evitando verificações para correções de valores. A média calculada é aplicada em cada pixel com sua respectiva cor. Isso torna que os valores que estarão presente sejam o mesmo para todos as cores de pixel. Pegamos a ideia inicial de uma imagem cromática, mencionamos que a imagem cromática seja composta pela soma de três imagens bidimensionais, como foi apresentada acima. O filtro de escala de cinza torna os valores das cores iguais, ou seja, transformando uma imagem cromática em uma imagem monocromática isso dado que todas as cores de pixel tenham o mesmo valor.

Abaixamos mostramos os resultados obtidos quando aplicado o filtro.



Figura 12 - A imagem de cima tem o efeito escala de cinza aplicada. A imagem da debaixo é a imagem original sem efeito.



Figura 13 - A imagem da direita tem o efeito escala de cinza aplicada. A imagem da direita é a imagem original sem efeito. (1).



Figura 14 - A imagem da direita tem o efeito escala de cinza aplicada. A imagem da direita é a imagem original sem efeito. (2)

FILTRO DE SOBEL

Filtro de Sobel é usado em processamento de imagens para a detecção de bordas, basicamente calcula-se o gradiente de intensidade da imagem em cada ponto. A forma do cálculo é dada por duas matrizes, uma matriz do eixo X e outra matriz do eixo Y. Cada matriz de eixo tem seu cálculo separado e ainda cada uma delas percorre a mesma matriz imagem.

$$X = \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix} \quad Y = \begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix}$$

O produto da matriz X com a matriz imagem gera um único valor escalar, assim acontece o mesmo com a matriz Y. Isso significa que a operação gera dois valores escalares um no eixo X e outro no eixo Y para cada iteração que ocorre dentro do algoritmo. Como mostrado na implementação abaixo:

```
void filtroSobel(Imagen *img)
{
    Imagen *copySobel = copiaImagen(img);

    for (int i = 1; i < obtemAltura(copySobel) - 1; i++)
    {
        for (int j = 1; j < obtemLargura(copySobel) - 1; j++)
        {
            int eixoXRED = 0, eixoYRED = 0, distanciaRED = 0;
            int eixoXGREEN = 0, eixoYGREEN = 0, distanciaGREEN = 0;
            int eixoXBLUE = 0, eixoYBLUE = 0, distanciaBLUE = 0;

            // Matriz Y

            Pixel pixel = obtemPixel(copySobel, i - 1, j - 1);
            eixoYRED += (int)pixel.cor[RED] * (1);
            eixoYGREEN += (int)pixel.cor[GREEN] * (1);
            eixoYBLUE += (int)pixel.cor[BLUE] * (1);

            pixel = obtemPixel(copySobel, i - 1, j);
            eixoYRED += (int)pixel.cor[RED] * (2);
            eixoYGREEN += (int)pixel.cor[GREEN] * (2);
            eixoYBLUE += (int)pixel.cor[BLUE] * (2);

            pixel = obtemPixel(copySobel, i - 1, j + 1);
            eixoYRED += (int)pixel.cor[RED] * (1);
            eixoYGREEN += (int)pixel.cor[GREEN] * (1);
            eixoYBLUE += (int)pixel.cor[BLUE] * (1);

            pixel = obtemPixel(copySobel, i + 1, j - 1);
            eixoYRED += (int)pixel.cor[RED] * (-1);
            eixoYGREEN += (int)pixel.cor[GREEN] * (-1);
```

```

eixoYBLUE += (int)pixel.cor[BLUE] * (-1);

pixel = obtemPixel(copySobel, i + 1, j);
eixoYRED += (int)pixel.cor[RED] * (-2);
eixoYGREEN += (int)pixel.cor[GREEN] * (-2);
eixoYBLUE += (int)pixel.cor[BLUE] * (-2);

pixel = obtemPixel(copySobel, i + 1, j + 1);
eixoYRED += (int)pixel.cor[RED] * (-1);
eixoYGREEN += (int)pixel.cor[GREEN] * (-1);
eixoYBLUE += (int)pixel.cor[BLUE] * (-1);

pixel = obtemPixel(copySobel, i - 1, j);
eixoYRED += (int)pixel.cor[RED] * (0);
eixoYGREEN += (int)pixel.cor[GREEN] * (0);
eixoYBLUE += (int)pixel.cor[BLUE] * (0);

pixel = obtemPixel(copySobel, i, j);
eixoYRED += (int)pixel.cor[RED] * (0);
eixoYGREEN += (int)pixel.cor[GREEN] * (0);
eixoYBLUE += (int)pixel.cor[BLUE] * (0);

pixel = obtemPixel(copySobel, i + 1, j);
eixoYRED += (int)pixel.cor[RED] * (0);
eixoYGREEN += (int)pixel.cor[GREEN] * (0);
eixoYBLUE += (int)pixel.cor[BLUE] * (0);

// Matriz X

pixel = obtemPixel(copySobel, i - 1, j - 1);
eixoXRED += (int)pixel.cor[RED] * (1);
eixoXGREEN += (int)pixel.cor[GREEN] * (1);
eixoXBLUE += (int)pixel.cor[BLUE] * (1);

pixel = obtemPixel(copySobel, i - 1, j + 1);
eixoXRED += (int)pixel.cor[RED] * (-1);
eixoXGREEN += (int)pixel.cor[GREEN] * (-1);
eixoXBLUE += (int)pixel.cor[BLUE] * (-1);

pixel = obtemPixel(copySobel, i, j - 1);
eixoXRED += (int)pixel.cor[RED] * (2);
eixoXGREEN += (int)pixel.cor[GREEN] * (2);
eixoXBLUE += (int)pixel.cor[BLUE] * (2);

pixel = obtemPixel(copySobel, i, j + 1);
eixoXRED += (int)pixel.cor[RED] * (-2);
eixoXGREEN += (int)pixel.cor[GREEN] * (-2);
eixoXBLUE += (int)pixel.cor[BLUE] * (-2);

```

```

pixel = obtemPixel(copySobel, i + 1, j - 1);
eixoXRED += (int)pixel.cor[RED] * (1);
eixoXGREEN += (int)pixel.cor[GREEN] * (1);
eixoXBLUE += (int)pixel.cor[BLUE] * (1);

pixel = obtemPixel(copySobel, i + 1, j + 1);
eixoXRED += (int)pixel.cor[RED] * (-1);
eixoXGREEN += (int)pixel.cor[GREEN] * (-1);
eixoXBLUE += (int)pixel.cor[BLUE] * (-1);

pixel = obtemPixel(copySobel, i, j - 1);
eixoXRED += (int)pixel.cor[RED] * (0);
eixoXGREEN += (int)pixel.cor[GREEN] * (0);
eixoXBLUE += (int)pixel.cor[BLUE] * (0);

pixel = obtemPixel(copySobel, i, j);
eixoXRED += (int)pixel.cor[RED] * (0);
eixoXGREEN += (int)pixel.cor[GREEN] * (0);
eixoXBLUE += (int)pixel.cor[BLUE] * (0);

pixel = obtemPixel(copySobel, i, j + 1);
eixoXRED += (int)pixel.cor[RED] * (0);
eixoXGREEN += (int)pixel.cor[GREEN] * (0);
eixoXBLUE += (int)pixel.cor[BLUE] * (0);

distanciaRED = sqrt(pow(eixoYRED, 2) + pow(eixoXRED, 2));
distanciaGREEN = sqrt(pow(eixoYGREEN, 2) + pow(eixoXGREEN, 2));
distanciaBLUE = sqrt(pow(eixoYBLUE, 2) + pow(eixoXBLUE, 2));

// Canal vermelho
if (distanciaRED > 255)
{
    pixel.cor[RED] = 255;
}
else if (distanciaRED < 0)
{
    pixel.cor[RED] = 0;
}
else
{
    pixel.cor[RED] = distanciaRED;
}

// Canal verde
if (distanciaGREEN > 255)
{

```

```

        pixel.cor[GREEN] = 255;
    }
    else if (distanciaGREEN < 0)
    {

        pixel.cor[GREEN] = 0;
    }
    else
    {

        pixel.cor[GREEN] = distanciaGREEN;
    }

    // Canal azul
    if (distanciaBLUE > 255)
    {

        pixel.cor[BLUE] = 255;
    }
    else if (distanciaBLUE < 0)
    {

        pixel.cor[BLUE] = 0;
    }
    else
    {

        pixel.cor[BLUE] = distanciaBLUE;
    }
    recolorePixel(img, i, j, pixel);
}
}
liberaImagem(copySobel);
}

```

Após os cálculos para encontrar o valor do escalar, podemos aplicar duas operações, fica a critério do programador. A primeira operação com esses valores é fazer média aritmética, um cálculo relativamente simples, mas aplicado o filtro em uma imagem pode apresentar ruídos ou não ficar nítida a detecção da borda.

$$coeficienteSobel = \frac{eixo\ x + eixo\ y}{2}$$

A segunda operação, a que foi empregada no trabalho, é calcular a distância entre os valores, isso quer dizer que somamos o quadrado de cada um dos valores e obtivemos a raiz quadrada desse valor.

$$coeficienteSobel = \sqrt{(eixo\ x)^2 + (eixo\ y)^2}$$

Calculando o coeficiente utilizando o cálculo da distância a imagem se mostrou mais nítida e apresentou melhora nas bordas.



Figura 15 - Filtro de Sobel aplicando fórmula da média aritmética.



Figura 16 - Filtro de Sobel aplicando a fórmula da distância.

Note que a formula da distância se mostrou mais detalhada que aplicando a formula da média, realçou muitos detalhes que passariam despercebidos.

Após aplicar a formula para encontrar o valor do pixel, precisa ser feito uma verificação para saber se o coeficiente do filtro é maior que o valor máximo do pixel. Assim verificado, o valor será atribuído ao pixel da cor correspondente.

Abaixo temos alguns outros exemplos da aplicação do filtro.



Figura 21 - A imagem de cima tem o efeito do filtro de Sobel. A imagem da debaixo é a imagem original sem efeito.



Figura 17 - A imagem da esquerda tem o efeito filtro Sobel aplicada. A imagem da direita é a imagem original sem efeito. (1).



Figura 18 - A imagem da esquerda tem o efeito filtro Sobel aplicada. A imagem da direita é a imagem original sem efeito. (2).

FILTRO LAPLACE

Semelhante ao filtro de Sobel, o filtro de Laplace também é utilizado para detecção de bordas. A diferença entre os dois filtros é que o filtro de Laplace requer menos cálculos para ser executado pois só é utilizado somente uma única matriz de valores.

$$matriz \text{ } Laplace = \begin{matrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{matrix}$$

Os cálculos para esse filtro seguem os mesmos passos que o filtro de Sobel. A matriz percorrerá a matriz da imagem fazendo os cálculos dos valores e somando os valores da posição calculada.

```
void deteccaoBordasLaplace(Imagem *img)
{
    Imagem *copyImagen = copiaImagen(img);

    for (int i = 1; i < obtemAltura(copyImagen) - 1; i++)
    {
        for (int j = 1; j < obtemLargura(copyImagen) - 1; j++)
        {
            int modificadorLaplacianoRED = 0, modificadorLaplacianoGREEN = 0,
                modificadorLaplacianoBLUE = 0;

            Pixel pixel = obtemPixel(copyImagen, i, j);
            modificadorLaplacianoRED += (int)pixel.cor[RED] * (4);
            modificadorLaplacianoGREEN += (int)pixel.cor[GREEN] * (4);
            modificadorLaplacianoBLUE += (int)pixel.cor[BLUE] * (4);

            pixel = obtemPixel(copyImagen, i - 1, j);
            modificadorLaplacianoRED += (int)pixel.cor[RED] * (-1);
            modificadorLaplacianoGREEN += (int)pixel.cor[GREEN] * (-1);
            modificadorLaplacianoBLUE += (int)pixel.cor[BLUE] * (-1);

            pixel = obtemPixel(copyImagen, i, j - 1);
            modificadorLaplacianoRED += (int)pixel.cor[RED] * (-1);
            modificadorLaplacianoGREEN += (int)pixel.cor[GREEN] * (-1);
            modificadorLaplacianoBLUE += (int)pixel.cor[BLUE] * (-1);

            pixel = obtemPixel(copyImagen, i, j + 1);
            modificadorLaplacianoRED += (int)pixel.cor[RED] * (-1);
            modificadorLaplacianoGREEN += (int)pixel.cor[GREEN] * (-1);
            modificadorLaplacianoBLUE += (int)pixel.cor[BLUE] * (-1);

            pixel = obtemPixel(copyImagen, i + 1, j);
            modificadorLaplacianoRED += (int)pixel.cor[RED] * (-1);
            modificadorLaplacianoGREEN += (int)pixel.cor[GREEN] * (-1);
            modificadorLaplacianoBLUE += (int)pixel.cor[BLUE] * (-1);

            pixel = obtemPixel(copyImagen, i + 1, j + 1);
        }
    }
}
```

```

modificadorLaplacianoRED += (int)pixel.cor[RED] * (0);
modificadorLaplacianoGREEN += (int)pixel.cor[GREEN] * (0);
modificadorLaplacianoBLUE += (int)pixel.cor[BLUE] * (0);

pixel = obtemPixel(copyImagem, i - 1, j - 1);
modificadorLaplacianoRED += (int)pixel.cor[RED] * (0);
modificadorLaplacianoGREEN += (int)pixel.cor[GREEN] * (0);
modificadorLaplacianoBLUE += (int)pixel.cor[BLUE] * (0);

pixel = obtemPixel(copyImagem, i + 1, j - 1);
modificadorLaplacianoRED += (int)pixel.cor[RED] * (0);
modificadorLaplacianoGREEN += (int)pixel.cor[GREEN] * (0);
modificadorLaplacianoBLUE += (int)pixel.cor[BLUE] * (0);

pixel = obtemPixel(copyImagem, i - 1, j + 1);
modificadorLaplacianoRED += (int)pixel.cor[RED] * (0);
modificadorLaplacianoGREEN += (int)pixel.cor[GREEN] * (0);
modificadorLaplacianoBLUE += (int)pixel.cor[BLUE] * (0);

if (modificadorLaplacianoRED >= 255)
{
    pixel.cor[RED] = 255;
}
else if (modificadorLaplacianoRED <= 0)
{
    pixel.cor[RED] = 0;
}
else
{
    pixel.cor[RED] = modificadorLaplacianoRED;
}

if (modificadorLaplacianoGREEN >= 255)
{
    pixel.cor[GREEN] = 255;
}
else if (modificadorLaplacianoGREEN <= 0)
{
    pixel.cor[GREEN] = 0;
}
else
{
    pixel.cor[GREEN] = modificadorLaplacianoGREEN;
}

if (modificadorLaplacianoBLUE >= 255)
{
    pixel.cor[BLUE] = 255;
}

```

```

        else if (modificadorLaplacianoBLUE <= 0)
        {
            pixel.cor[BLUE] = 0;
        }
        else
        {
            pixel.cor[BLUE] = modificadorLaplacianoBLUE;
        }
        recolorePixel(img, i, j, pixel);
    }
}
liberaImagem(copyImagem);
}

```

Nesse filtro não precisamos calcular ou aplicar algum tipo de formula, basta verificar se está dentro dos valores aceitáveis do pixel, ou seja, verificar se o valor é menor que 0 ou maior que 255, e depois atribuir ao pixel de cor da imagem. Os resultados são semelhantes ao Sobel



Figura 19 - A imagem de cima tem o efeito do filtro de Laplace. A imagem da debaixo é a imagem original sem efeito



Figura 20 - A imagem da esquerda tem o efeito filtro Laplace aplicada. A imagem da direita é a imagem original sem efeito. (1).



Figura 21 A imagem da esquerda tem o efeito filtro Laplace aplicada. A imagem da direita é a imagem original sem efeito. (2).

FILTRO SEPIA (MEUFILTRO)

O filtro sépia segue a mesma lógica que o filtro de Laplace, temos uma matriz que é aplicada a matriz imagem, e em seguida alterando os valores dos pixels. A ideia é somar os valores dos pixels de cada cor obtidos pela multiplicação da matriz do filtro sépia, fazer a verificação em seguida e por final atribuir o valor para matriz imagem.

$$matrizSépia = \begin{matrix} 0,272 & 0,534 & 0,131 \\ 0,349 & 0,686 & 0,168 \\ 0,393 & 0,769 & 0,189 \end{matrix}$$

Cada pixel de cor diferente é multiplicar por uma linha da matrizSépia, por exemplo:

Temos o pixel de cor vermelha, então ela será multiplicada pela terceira linha sempre, o pixel azul é multiplicado pela segunda linha e assim por diante. Abaixo podemos observar a implementação do filtro sépia.

```
void meuFiltro(Imagem *img)
{
    Imagem *copySepia = copiaImagem(img);
    Pixel pixel;
    for (int i = 1; i < obtemAltura(copySepia) - 1; i++)
```

```

{
    for (int j = 1; j < obtemLargura(copySepia) - 1; j++)
    {
        int sepRED = 0;
        int sepGREEN = 0;
        int sepBLUE = 0;
        pixel = obtemPixel(copySepia, i, j);
        sepRED += (int)pixel.cor[RED] * 0.769;
        sepGREEN += (int)pixel.cor[GREEN] * 0.686;
        sepBLUE += (int)pixel.cor[BLUE] * 0.534;

        pixel = obtemPixel(copySepia, i, j - 1);
        sepRED += (int)pixel.cor[RED] * 0.393;
        sepGREEN += (int)pixel.cor[GREEN] * 0.349;
        sepBLUE += (int)pixel.cor[BLUE] * 0.272;

        pixel = obtemPixel(copySepia, i, j + 1);
        sepRED += (int)pixel.cor[RED] * 0.189;
        sepGREEN += (int)pixel.cor[GREEN] * 0.168;
        sepBLUE += (int)pixel.cor[BLUE] * 0.131;

        // sepRED /= 3;
        // sepGREEN /= 3;
        // sepBLUE /= 3;

        if (sepRED > 255)
        {
            pixel.cor[RED] = 255;
        }
        else if (sepRED < 0)
        {
            pixel.cor[RED] = 0;
        }
        else
        {
            pixel.cor[RED] = sepRED;
        }
        if (sepGREEN > 255)
        {
            pixel.cor[GREEN] = 255;
        }
        else if (sepGREEN < 0)
        {
            pixel.cor[GREEN] = 0;
        }
        else
        {
            pixel.cor[GREEN] = sepGREEN;
        }
    }
}

```

```

    if (sepBLUE > 255)
    {
        pixel.cor[BLUE] = 255;
    }
    else if (sepBLUE < 0)
    {
        pixel.cor[BLUE] = 0;
    }
    else
    {
        pixel.cor[BLUE] = sepBLUE;
    }
    recolorePixel(img, i, j, pixel);
}
}
liberaImagem(copySepia);
}

```

Após todas as operações obtivemos os seguintes resultados:



Figura 22 - A imagem de cima tem o efeito do filtro de sépia. A imagem da debaixo é a imagem original sem efeito



Figura 23- A imagem da esquerda tem o efeito filtro Sépia aplicada. A imagem da direita é a imagem original sem efeito. (1).



Figura 24 - A imagem da esquerda tem o efeito filtro Sépia aplicada. A imagem da direita é a imagem original sem efeito. (2).

GALERIA DE FOTOS

IMAGENS ORIGINAIS



Figura 25 - Imagem original A



Figura 26 - Imagem original B



Figura 27 - Imagem original C

FILTRO ESCURECER IMAGEM

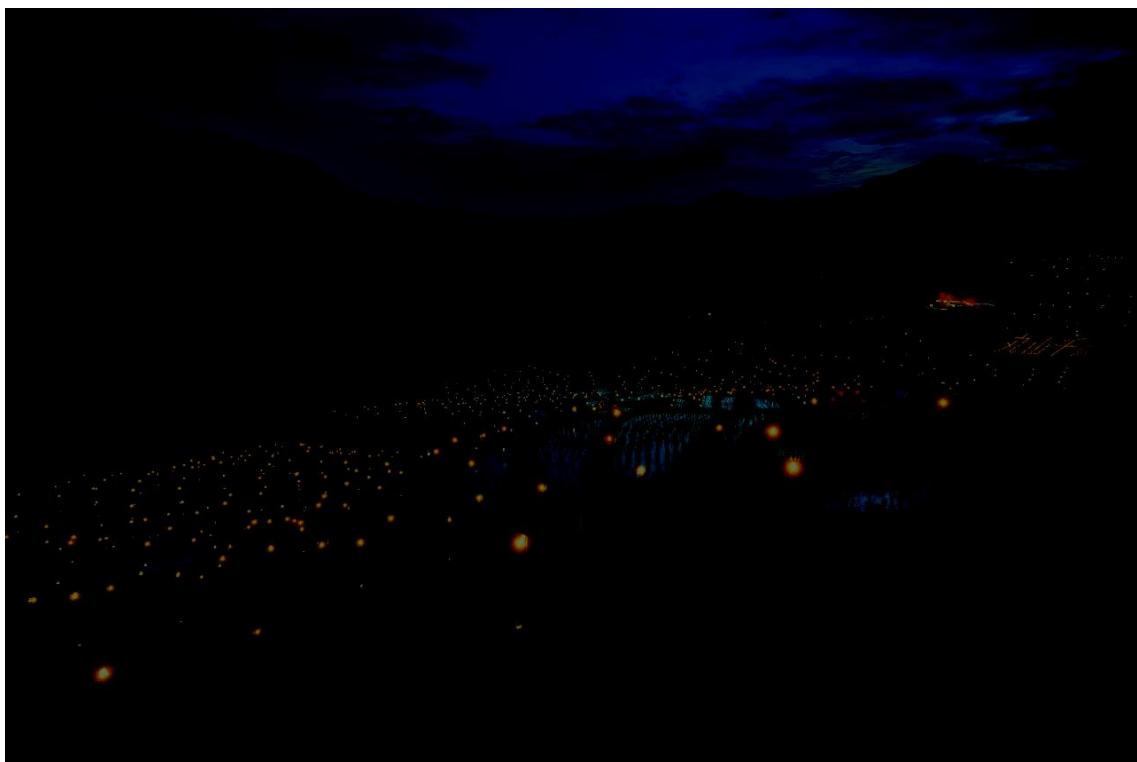


Figura 28 - Imagem A escurecida com fator 150

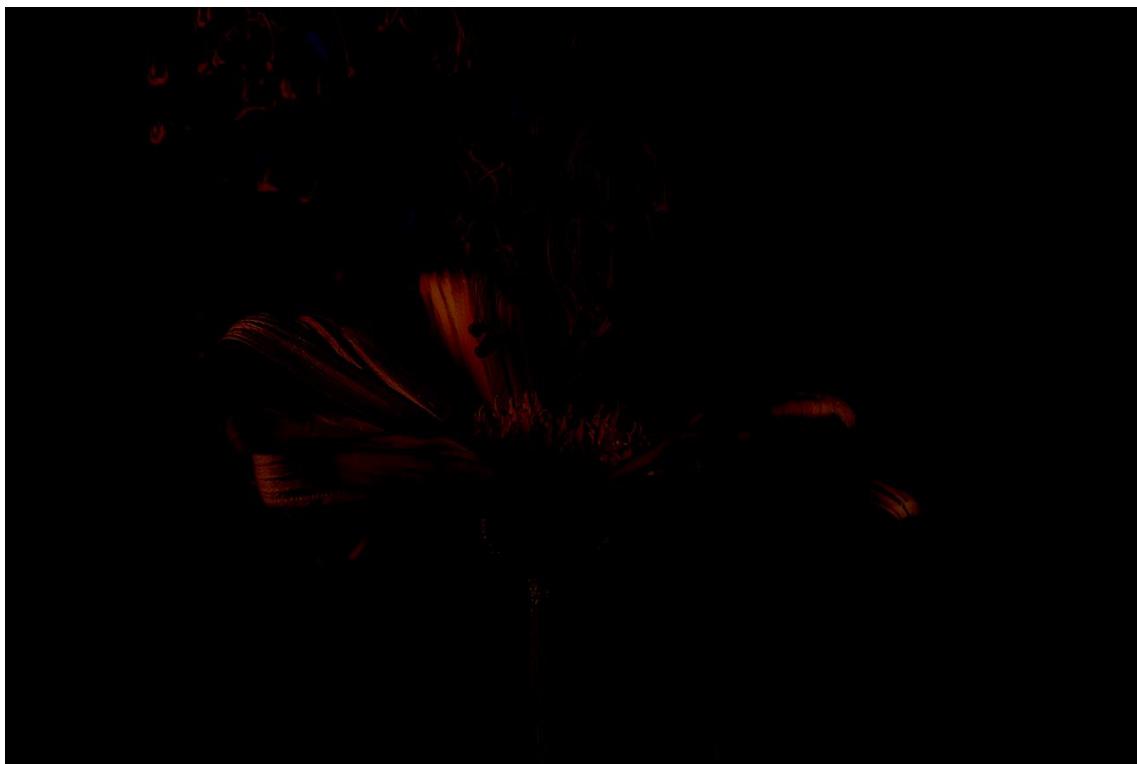


Figura 29 - Imagem B escurecida com fator 150.

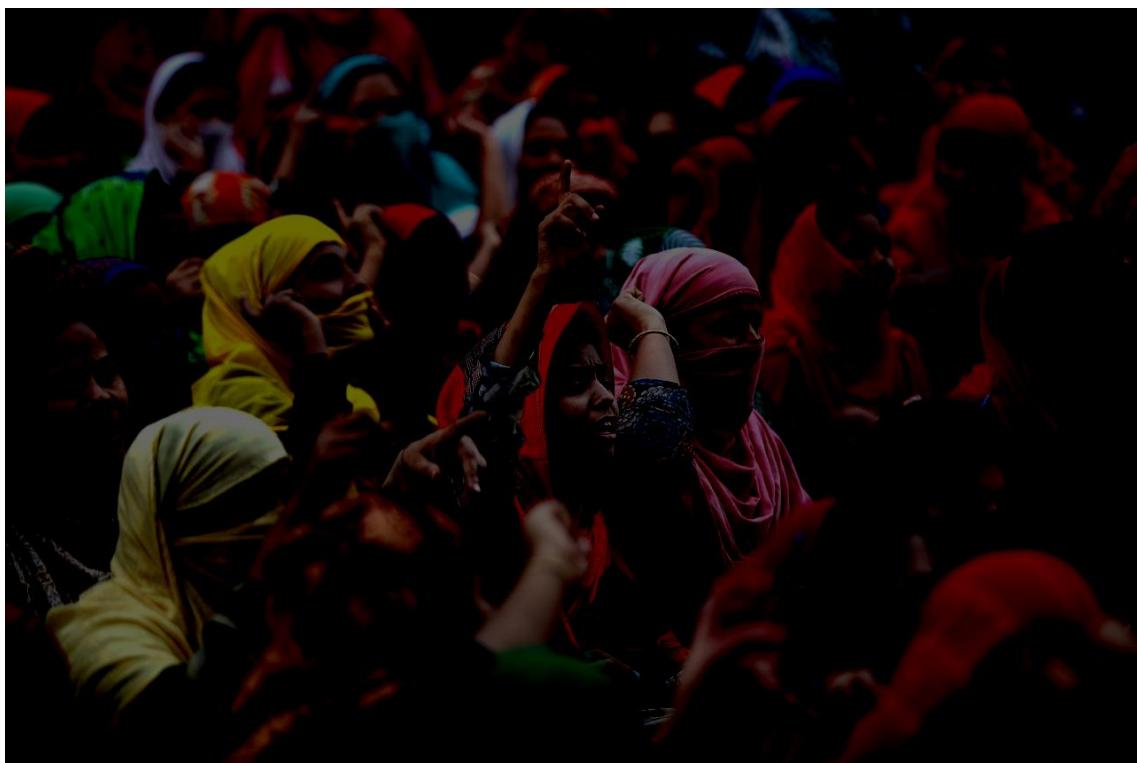


Figura 30 - Imagem C escurecida com fator 150.

FILTRO CLAREAR IMAGEM



Figura 31 - Imagem A clareada com fator 150.

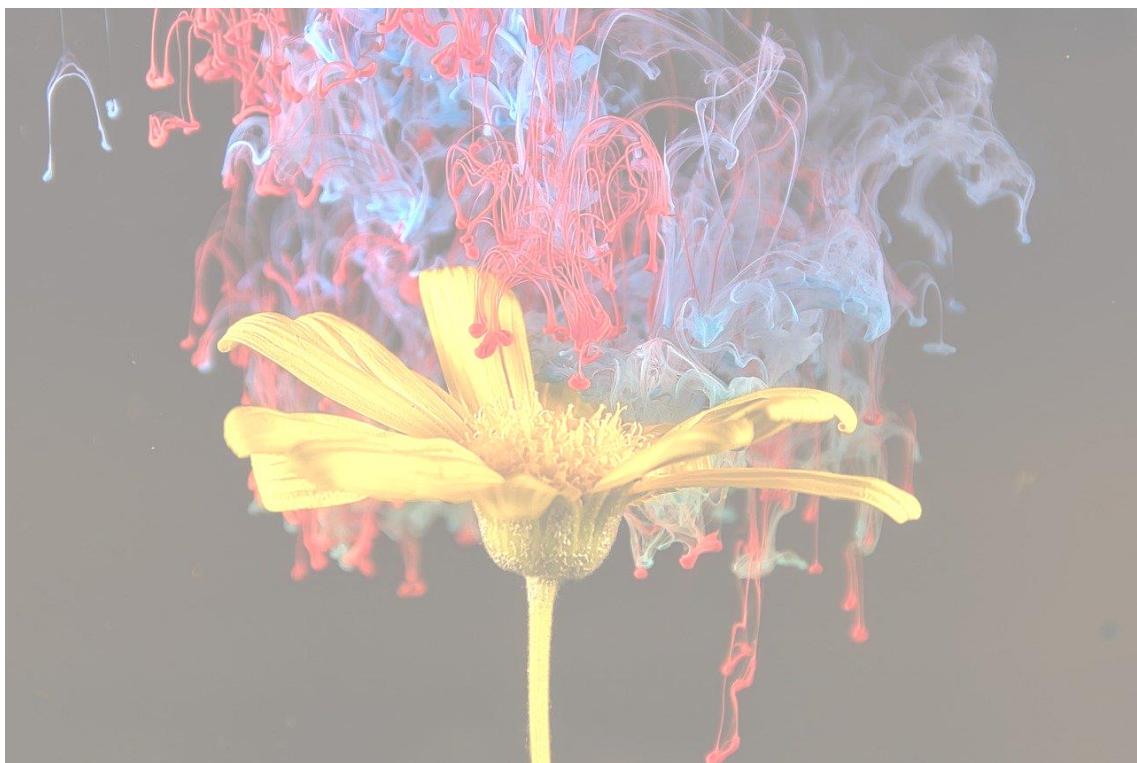


Figura 32 - Imagem B clareada com fator 150.



Figura 33 - Imagem C clareada com fator 150.

FILTRO ESCALA DE CINZA



Figura 34 - Imagem A com filtro de escala de cinza aplicado.



Figura 35 - Imagem B com filtro de escala de cinza aplicado.



Figura 36 - Imagem C com filtro de escala de cinza aplicado.

FILTRO DE SOBEL

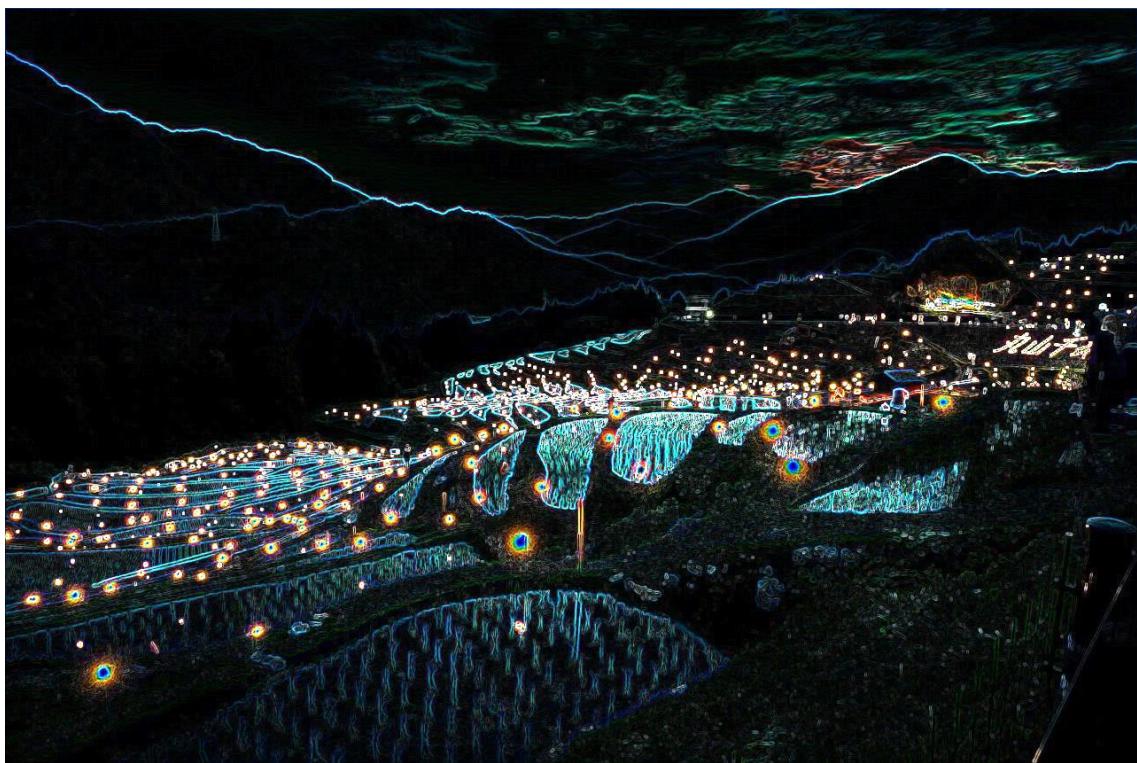


Figura 37 - Imagem A com filtro de Sobel aplicado.



Figura 38 - Imagem B com filtro de Sobel aplicado.



Figura 39 - Imagem C com filtro de Sobel aplicado.

FILTRO DE LAPLACE

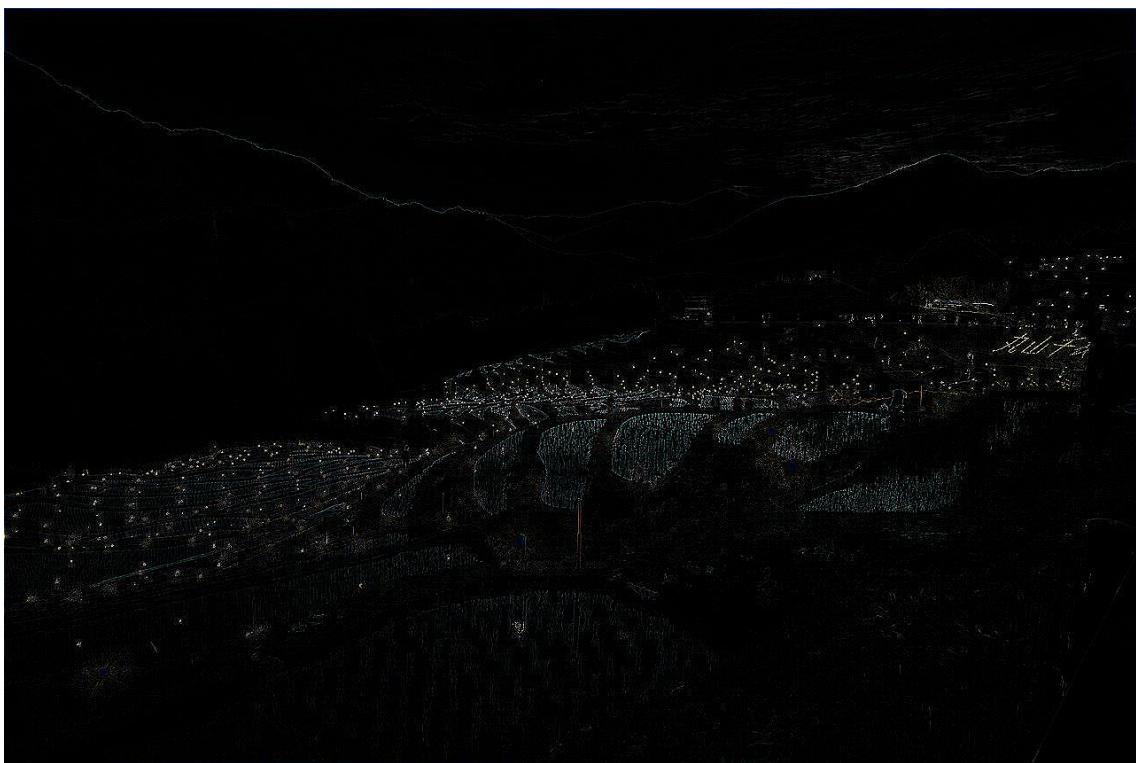


Figura 40 - Imagem A com filtro de Laplace aplicado.

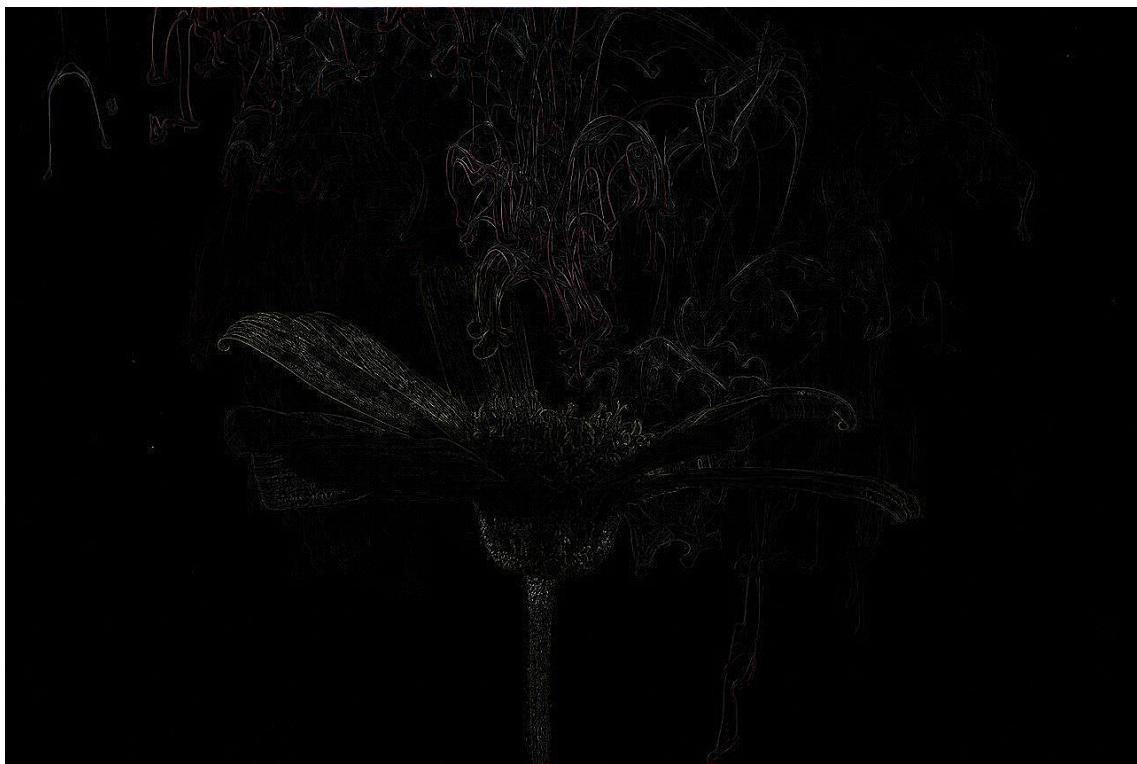


Figura 41 - Imagem B com filtro de Laplace aplicado.

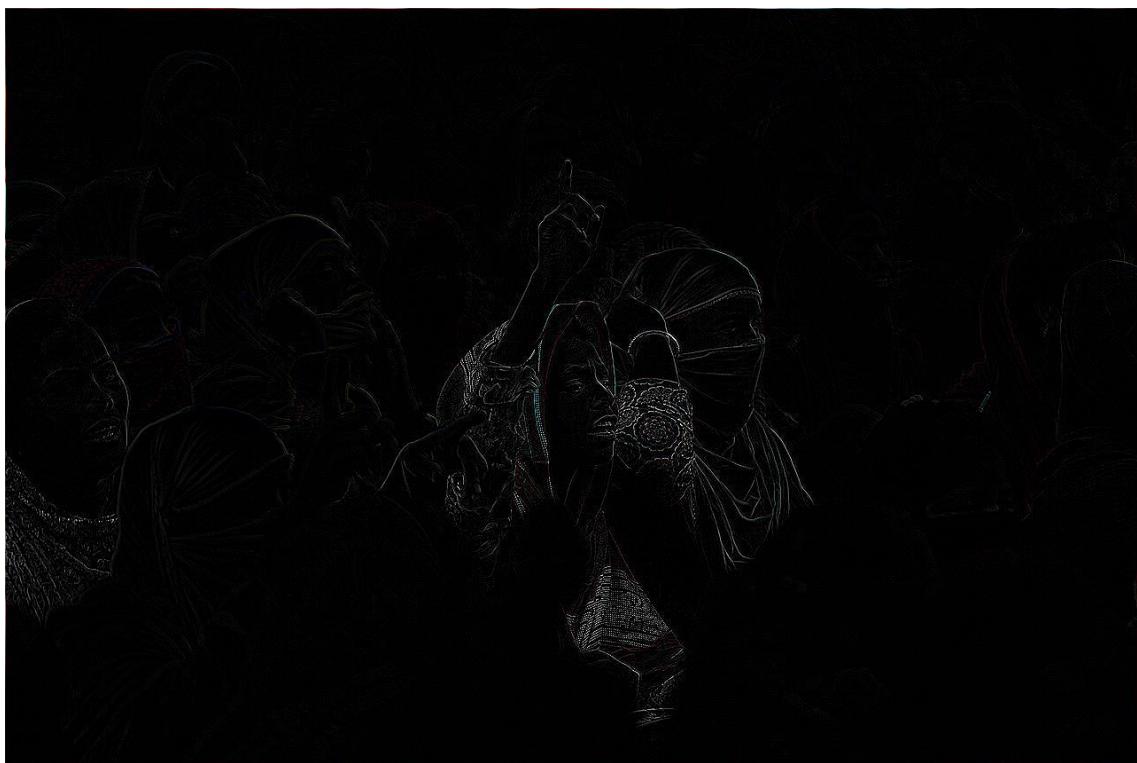


Figura 42 - Imagem C com filtro de Laplace aplicado.

FILTRO SÉPIA (MEU FILTRO)



Figura 43 - Imagem A com meu filtro aplicado.



Figura 44 - Imagem B com meu filtro aplicado.



Figura 45 - Imagem C com meu filtro aplicado.