Universidade Federal do Rio Grande do Sul
INF01046 - Fundamentos de Processamento de Imagens - Turma B - Trabalho 3

Gabriela Copetti Maccagnan

25/11/2024

1 Processamento de Vídeo em Tempo Real

Este trabalho tem como objetivo implementar funções de proceessamento de vídeo em tempo real, utilizando a linguagem C++, juntamente com as bibliotecas *OpenCV* para auxílio nas operações de processamento de imagens e a biblioteca *cvui* (https://fernandobevilacqua.com/cvui/) para a construção de uma interface para o programa.

O programa foi construído utilizando CMake, portanto possui um arquivo *CMakeLists.txt* e pode ser rodado a partir dos comandos na pasta root:
- cmake -B build (para fazer o build do programa)
- cmake –build build –config Release (para criar o executável na pasta \build\Release)
- .\build\Release\FPI _T3.exe (para rodar o executável)

O programa foi desenvolvido em cima do Trabalho 2, sendo assim sua estrutura é bastante similar, contando com um arquivo main.cpp, onde foi desenhada a interface e de onde são chamadas as funções de processamento a partir do clique nos botões da tela. Foram adicionadas algumas pastas para arquivos .cpp e .h para melhor separar as funções criadas de acordo com seu objetivo.

Para fechar o programa, basta clicar na tecla *esc*.

1.1 Questão 1

Configure o seu ambiente de programação (e.g., Visual Studio, etc.) compile e execute o programa básico disponibilizado no Apêndice A deste documento. Este programa simples lhe permitirá capturar e exibir vídeos em tempo real, provendo a estrutura sobre a qual você implementará as tarefas solicitadas.
Com cada quadro do vídeo capturado pela câmera, realize as operações

abaixo e exiba o frame resultante em uma janela ao lado da original. Para utilização dos comandos mencionados para completar a tarefa, pode ser necessário algum tipo de pre-processamento aplicado ao quadro em questão. Neste caso, é parte da tarefa a identificação e aplicação de tal pre-processamento.

Para exibição de duas janelas capturando vídeos em tempo real, foi utilizada a seguinte estrutura:

No arquivo main.cpp foi criado um botão "Start Video" que será exibido na interface, permitindo que o usuário comece a captura do vídeo. Na interface também foi disponibilizado um menu de comandos (teclas) que o usuário pode utilizar para aplicar as transformações desejadas ao vídeo. Ao clicar em "r", por exemplo, o vídeo será gravado, "0" aplica o filtro Gaussiano, etc.

```
if (shouldStartVideo) {
    return startVideo();
}
```

O clique no botão para começar o vídeo chama a função startVideo() que possui em parte o código disponibilizado na descrição do trabalho, com modificações adicionais para exibir duas janelas, gravar o vídeo editado e identificar qual tecla é clicada pelo usuário.

```
int startVideo() {
    int camera = 0;
    VideoCapture cap;
    if(!cap.open(camera)) {
      return 0;
    }

    Mat editedFrame;
    bool shouldShowBar = false;

    VideoWriter video("output.mp4", fourcc, fps, frame_size);

    for(;;) {
        Mat frame;
        cap >> frame;
        if(frame.empty()) {
            break;
        }

        editedFrame = getEditedFrame(frame, applyOperation, shouldShowBar);

        cv::imshow(ORIGINAL_VIDEO_TITLE, frame);
        cv::imshow(EDITED_VIDEO_TITLE, editedFrame);
```

```
        if (shouldRecord) {
            video.write(editedFrame);
        }

        char key = waitKey(1);
        if (key == 48) {
            applyOperation = 0;
            shouldShowBar = true;
        } else if (key == 49) {
            applyOperation = 1;
            shouldShowBar = true;
        } else if (key == 50) {
            applyOperation = 2;
        } else if (key == 51) {
            applyOperation = 3;
            shouldShowBar = true;
        } else if (key == 52) {
            applyOperation = 4;
        } else if (key == 53) {
            applyOperation = 5;
        } else if (key == 54) {
            applyOperation = 6;
        } else if (key == 55) {
            applyOperation = 7;
        } else if (key == 56) {
            applyOperation = 8;
        } else if (key == 57) {
            applyOperation = 9;
        } else if (key == 114) {
            shouldRecord = true;
        } else if (key == 27) {
            break;
        }
    }

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

A segunda janela exibida que corresponde ao vídeo que será editado tem seus frames adquiridos a partir da função getEditedFrame(), essa função identifica qual foi a última tecla clicada pelo usuário e aplica a transformação correspondente ao frame corrente. Caso o usuário clique em uma nova tecla, na próxima iteração o frame é modificado de acordo.

```
Mat getEditedFram(const Mat &frame, int &applyOperation, bool &shouldShowBar) {
    Mat editedFrame;

    if (applyOperation == 0) {
        ...
    } else if (applyOperation == 1) {
        ...
    } else if (applyOperation == 2) {
        ...
    } else if (applyOperation == 3) {
        ...
    } else if (applyOperation == 4) {
        ...
    } else if (applyOperation == 5) {
        ...
    } else if (applyOperation == 6) {
        ...
    } else if (applyOperation == 7) {
        ...
    } else if (applyOperation == 8) {
        ...
    } else if (applyOperation == 9) {
        ...
    } else {
        editedFrame = frame.clone();
    }

    return editedFrame;
}
```

Algumas funções como filtro Gaussiano e Canny permitem que o usuário determine certos parâmetros utilizados nas funções, portanto possuem uma trackbar que é exibida caso a tecla correspondente à transformação seja clicada. Os ajustes da trackbar também se dão em tempo real.

Abaixo está uma captura de tela de como são exibidos os vídeos original e editado e mais detalhes sobre a interface podem ser visualizados na última sessão do relatório.

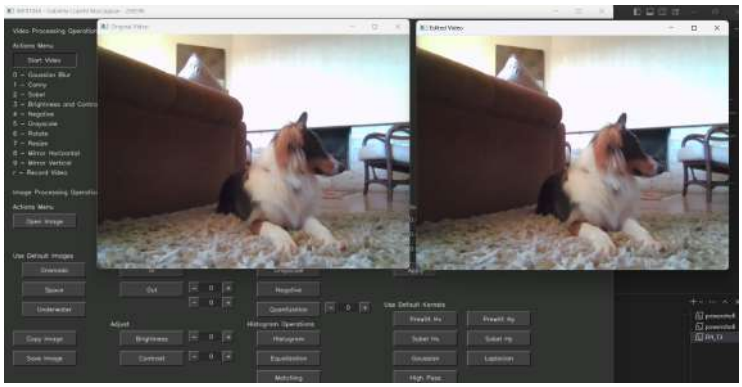

Figure 1: Captura de tela da abertura de duas janelas de vídeo ao começar o processamento de vídeo

1.2 Questão 2

Utilize o comando GaussianBlur para aplicar borramento ao video. Utilize um Trackbar para definir o tamanho do kernel Gaussiano.

Para aplicação do filtro Gaussiano, introduzindo o efeito de borramento no vídeo, foi utilizada a função "GaussianBlur" do Opencv. Essa função recebe como parâmetros a matriz original, a matriz de saída, o tamanho do kernel e o desvio padrão do kernel no eixo x. Também é possível modificar os parâmetros correspondentes ao desvio no eixo y e o tipo de borda, mas caso eles não sejam utilizados a função aplica a eles valores padrão.

Aqui o tamanho do kernel é estabelecido pelo usuário, sendo possível utilizar uma trackbar que ao ser modificada também altera em tempo real o filtro Gaussiano aplicado ao vídeo com o novo tamanho de kernel. Isso faz com que o vídeo possa ter diferentes níveis de borramento. O tamanho de kernel deve ser sempre maior que zero, caso contrário é retornado o próprio vídeo original.

```
Mat applyGaussianToVideo(const Mat &frame, bool &shouldShowBar) {
    Mat editedFrame;
    if (shouldShowBar && !alreadyCreatedKernel) {
        createTrackbar(...);
        ...
    }
    if (kernelSize != 0) {
        GaussianBlur(frame, editedFrame, Size(kernelSize, kernelSize), 0);
    } else {
        editedFrame = frame.clone();
    }
    shouldShowBar = false;
    return editedFrame;
```

```
}
```

Para não recriar a trackbar toda vez que a função do filtro Gaussiano seja chamada, foram estabelecidas as variáveis "shouldShowBar" e "alreadyCreatedKernel". Essas variáveis vão se repetir de forma similar sempre que for necessário criar uma trackbar para uma função e controlar sua visibilidade. Porém, uma limitação do Opencv não permite que sejam excluídas trackbars após criadas, fazendo com que, dependendo da ordem das operações escolhidas pelo usuário, múltiplas trackbars serão apresentadas na tela.

Figure 2:

1.3 Questão 3

Utilize o comando Canny para detectar as arestas no vídeo.

Para aplicação do filtro Canny e realizar a detecção de arestas do vídeo, foi utilizado o comando "Canny" do Opencv. Esse comando tem como parêmetros a matriz original, a matriz de saída e dois valores de threshold usados no procedimento de histerese. Ambos os valores de threshold devem ser especificados pelo usuário, também através de trackbars. Isso permite que o usuário visualize em tempo real a influência de cada parâmetro na detecção de arestas do vídeo.

```
Mat applyCannyToVideo(const Mat &frame, bool &shouldShowBar) {
    Mat editedFrame;
    if (shouldShowBar && !alreadyCreatedLowThresh && !alreadyCreatedHighThresh) {
        createTrackBar(LOW_THRESH_TRACKBAR, ...);
        createTrackBar(HIGH_THRESH_TRACKBAR, ...);
        ...
    }
    Canny(frame, editedFrame, lowThresh, highThresh);
    return editedFrame;
}
```

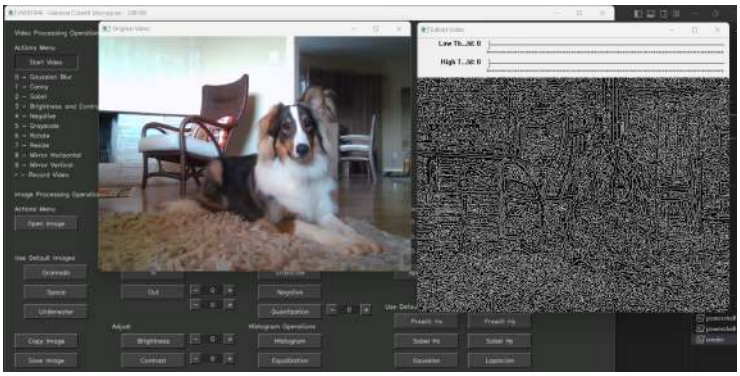

Figure 3: Captura da comparação entre o vídeo original e o filtro de Canny com thresholds em zero

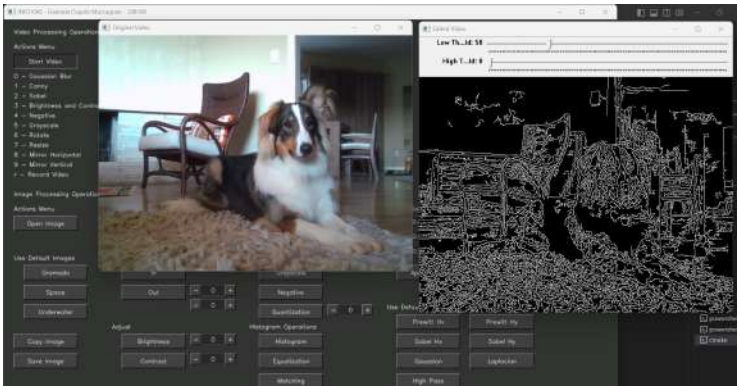

Figure 4: Captura da comparação entre o vídeo original e o filtro de Canny aumentando o primeiro threshold

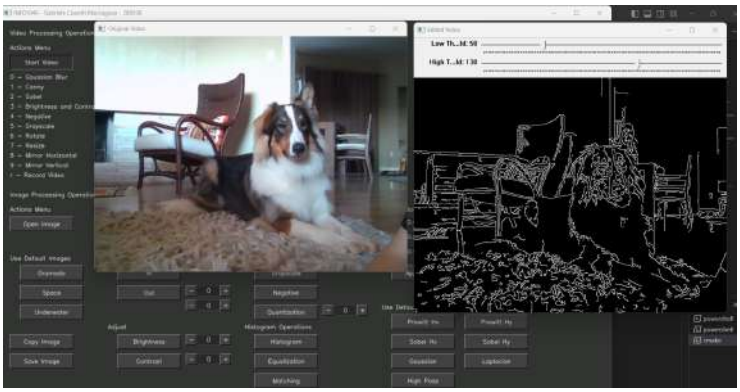

Figure 5: Captura da comparação entre o vídeo original e o filtro de Canny aumentando o segundo threshold

1.4 Questão 4

Utilize o comando Sobel para obter uma estimativa do gradiente do vídeo.

Para aplicação do filtro Sobel e obtenção de uma estimativa do gradiente do vídeo, foi utilizada a função "Sobel" do Opencv. Essa função calcula a primeira, segunda, terceira ou uma mistura de derivadas da imagem usando o operador extendido de Sobel. Ela usa como parâmetros a matriz de entrada, a matriz de saída, profundidade da imagem de saída, ordem da derivada em x, ordem da derivada em y, tamanho do kernel de Sobel (1, 3, 5 ou 7), fator de escala, valor delta e o tipo de borda.

```
Mat applySobelToVideo(const Mat &frame) {
    Mat editedFrame;
    ...
    Sobel(..., x_src, ..., 1, 0, ...);
    Sobel(..., y_src, ..., 0, 1, ...);
    convertScaleAbs(x_src, x_dst);
    convertScaleAbs(y_src, y_dst);
    addWeightned(x_dst, 0.5, y_dst, 0.5, 0, editedFrame);
    return editedFrame;
}
```

Figure 6: Imagem resultante da aplicação do filtro Sobel ao vídeo original

1.5 Questão 5

Utilize o comando convertTo para realizar ajuste de brilho, ajuste de contraste, e obter o negativo do video.

Para ajustes de brilho e contraste do vídeo, assim como obtenção do negativo do vídeo foi usada a função "addWeighted" do Opencv, uma vez que não foi possível usar a função indicada "convertTo".

A função "addWeighted" pode ser usada para combinar duas imagens em uma pois ela calcula a soma com pesos de dois vetores (ou matrizes) usando a equação abaixo:

$$dst = src1 * alpha + src2 * beta + gamma; \tag{1}$$

Onde "dst" é o vetor resultante, "src1" e "src2" os vetores a serem combinados, "alpha" o coeficiente angular de "src1", "beta" o coeficiente angular de "src2" e "gamma" o coeficiente linear da equação.

1.5.1 Brilho e Constraste

Para ajuste de luz e contraste, a função "addWeighted" teve que ser adaptada ao nosso caso. Como não estamos combinando duas imagens, apenas querendo utilizar os coeficientes da função, "beta" precisou ser estabelecido como 0.

Assim, "alpha" foi utilizado como o coeficiente de contraste da função e "gamma" como coeficiente de brilho. Ambos coeficientes podem ser ajusta-

dos pelo usuário através de trackbars e a modificação será aplicada ao vídeo, contanto que os valores dos coeficientes de brilho e contraste estejam dentro do esperado: de -255 a 255 para brilho, de 0 a 255 para constrate (invervalo aberto, valor deve ser > 0).

```
Mat applyBrightnessContrastToVideo(const Mat &frame, bool &shouldShowBar) {
    Mat editedFrame;
    if (shouldShowBar && !alreadyCreatedLight && !alreadyCreatedContrast) {
        createTrackbar(BRIGHTNESS_TRACKBAR, ...);
        createTrackbar(CONTRAST_TRACKBAR, ...);
        ...
    }
    if (brightness <= 255 && brightness >= -255 && constrast <= 255 && constrast > 0) {
        addWeighted(frame, constrast, frame, 0, brighness, editedFrame);
    } else {
        editedFrame = frame.clone();
    }
    return editedFrame;
}
```

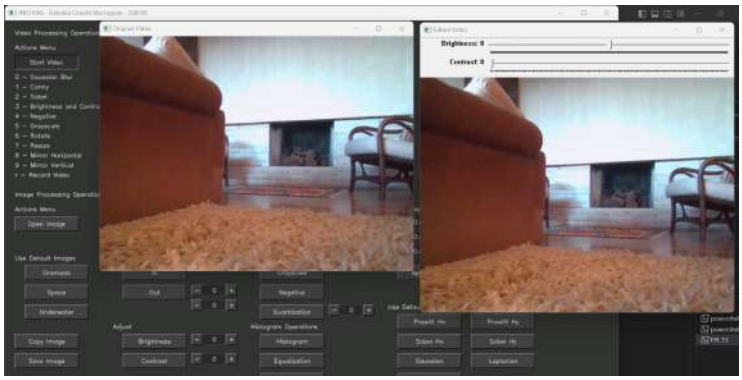

Figure 7: Captura da comparação entre o vídeo original e o editado com a opção de ajuste de brilho e contraste, aparição das novas trackbars

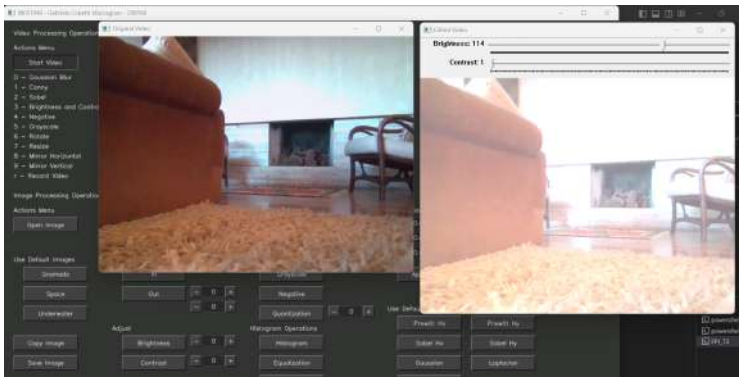

Figure 8: Captura da comparação entre o vídeo original e o editado com aumento de brilho

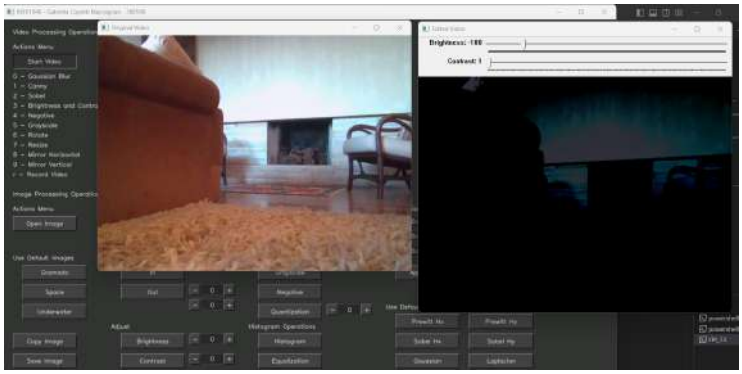

Figure 9: Captura da comparação entre o vídeo original e o editado com redução de brilho

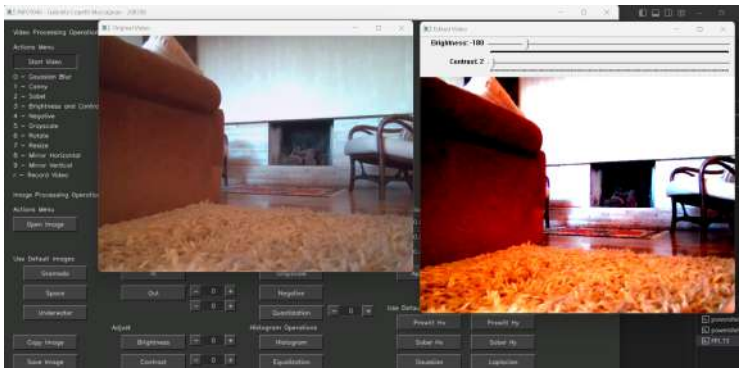

Figure 10: Captura da comparação entre o vídeo original e o editado com aumento de contraste

1.5.2 Negativo

Para obtenção do negativo do vídeo, a função "addWeighted" também recebeu como parâmetro "beta" o valor de 0, pois aqui também não queremos combinar duas imagens, apenas usar os coeficientes.

Já "alpha" recebeu o valor de -1 e "gamma" de 255, seguindo a equação para cálculo de negativo abaixo:

$$dst = 255 - src; \tag{2}$$

```
if (applyOperation == 4) {
    addWeighted(frame, -1, 0, 255, editedFrame);
}
```

Figure 11: Imagem resultante da aplicação da transformação para negativo no vídeo original

1.6 Questão 6

Conversão de cores (RGB) para tons de cinza (grayscale).

Para conversão de cores do vídeo foi utilizada a função "cvtColor" do Opencv. Essa é uma função que pode realizar diversos tipos de conversão de cor, não somente a conversão para tons de cinza. Ela recebe como parâmetros a matriz original, a matriz que será usada para gravar a saída e o código do espaço de cor desejado para a imagem de saída - nesse caso foi utilizado o código "6" que corresponde à escala de cinza. É possível ainda aplicar um quarto parâmetro à função, que seria correspondente ao número de canais de cor na imagem resultante, no entanto pode-se deixar esse valor vazio para que a própria função realize a inferência de quantos canais são necessários baseados na imagem fonte e na conversão de cores que será feita.

```
if (applyOperation == 5) {
    cvtColor(frame, editedFrame, 6);
}
```

Figure 12: Imagem resultante da aplicação da transformação para grayscale no vídeo original

1.7 Questão 7

Redimensionamento do vídeo para a metade do número de píxels em cada uma de suas dimensões.

Para a operação de redimensionamento do vídeo, foi utilizada a função "resize" do Opencv que recebe como parâmetros a matriz original, a que será usada para gravar a saída e o novo tamanho da matriz. Nesse caso, como o esperado era que o número de pixels das dimensões de saída fosse igual à metade do vídeo original em cada dimensão, obteu-se o tamanho do vídeo original através de "frame.cols" (para o número de colunas) e "frame.rows" (para o número de linhas) e dividiu-se esses valores por dois para formar o novo tamanho.

```
if (applyOperation == 7) {
    resize(frame, editedFrame, Size(frame.cols / 2, frame.rows / 2));
}
```

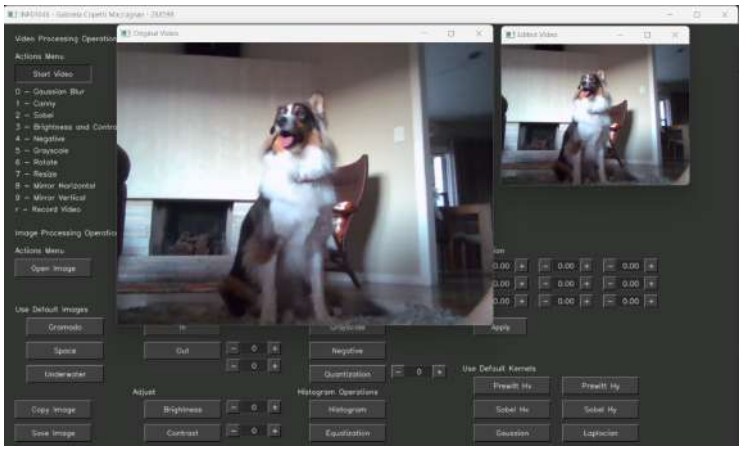

Figure 13: Captura da comparação entre o vídeo original e o redimensionado

1.8 Questão 8

Rotação do vídeo de 90 graus.

Para a rotação do vídeo em 90 graus, foi utilizada a função do Opencv "rotate". Essa função recebe três parâmetros: a matriz original; a matriz que será usada para gravar a saída; o código de rotação que será usado para determinar o sentido de rotação. Neste caso foi usado o código "ROTATE_90_CLOCKWISE" que indica uma rotação de 90 graus em sentido horário, caso se quisesse uma rotação em sentido anti-horário poderia ter sido utilizado o código "ROTATE_90 _COUNTERCLOCKWISE".

```
if (applyOperation == 6) {
    rotate(frame, editedFrame, ROTATE_90_CLOCKWISE);
}
```

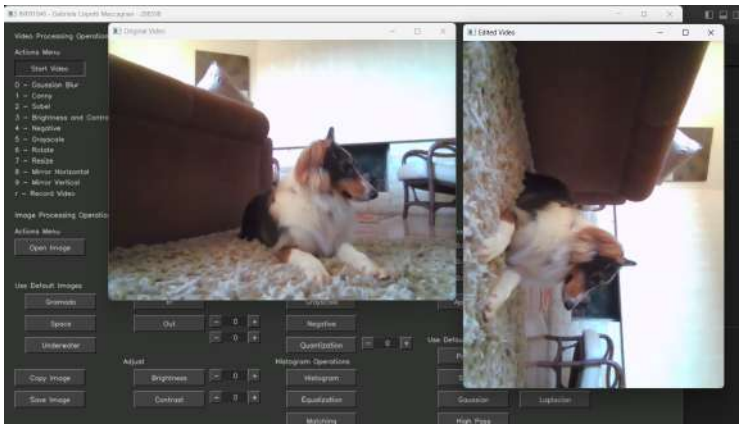

Figure 14: Captura da comparação entre o vídeo original e o rotado em 90 graus

1.9 Questão 9

Espelhamento do vídeo (horizontal e vertical).

Para ambas opções de espelhamento foi utilizada a função do Opencv "flip". Essa função recebe três parâmetros: a matriz original; a matriz que será usada para gravar a saída; a flag que especifica o tipo de flip - 0 significa inverter a matriz em torno do eixo x, 1 significa inverter a matriz em torno do eixo y.

Abaixo seguem os pedaços de código usados para realizar as transformações no vídeo e as capturas de tela com os resultados correspondentes:

```
if (applyOperation == 8) {
    flip(frame, editedFrame, 1);
}
```

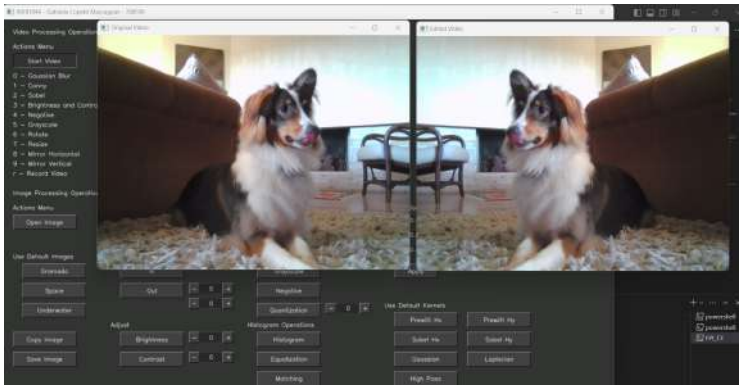

Figure 15: Captura da comparação entre o vídeo original e o espelhado horizontalmente

```
if (applyOperation == 9) {
    flip(frame, editedFrame, 0);
}
```

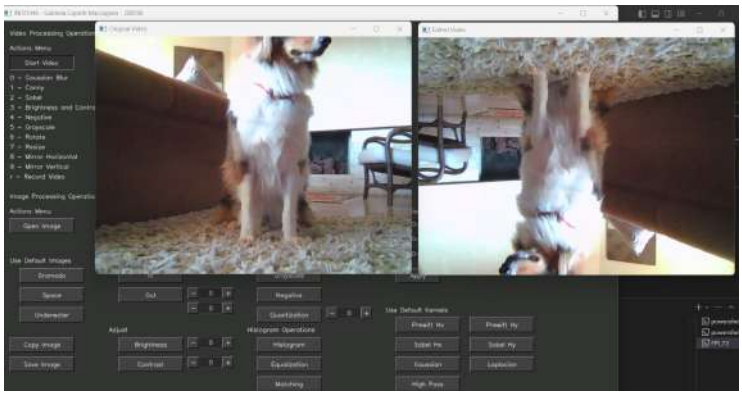

Figure 16: Captura da comparação entre o vídeo original e o espelhado verticalmente

1.10 Questão 10

Gravação de vídeo, levando em conta todos os efeitos acima, exceto Rotação e Redimentsionamento, visto que estas operações alteram as dimensões originais do frame, o que tenderia a ocasionar um erro durante a tentativa de gravação.

Link de acesso: `https://drive.google.com/file/d/1JOx9LbD2IjScbGPSz9xv73PTr8PSuwZz/view?usp=sharing`

1.11 Interface

A interface do terceiro trabalho, como já mencionado, foi construída usando como base o trabalho 2. Pode-se ver na imagem abaixo que a estrutura de processamento de imagens foi mantida, assim ainda é possível realizar qualquer uma das operações criadas para os trabalhos anteriores, mas também foi adicionado um menu na parte superior para lidar com as operações de vídeo específicas do trabalho 3.

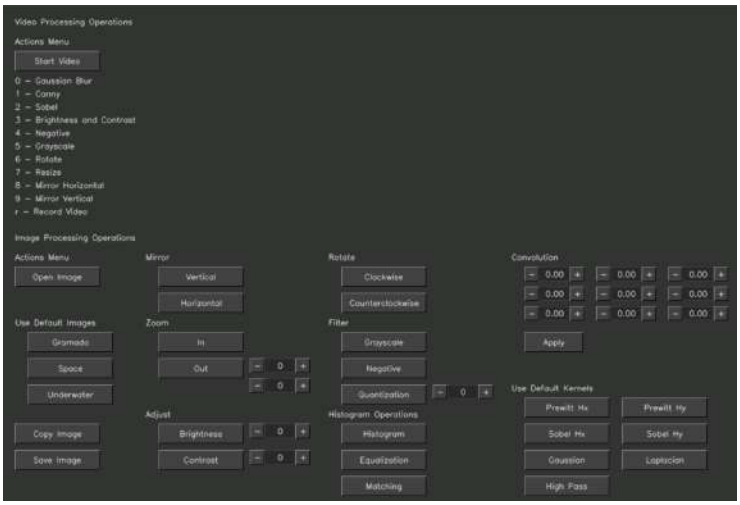

Figure 17: Interface do programa

Esse menu conta apenas com um botão para começar a captura de vídeo e abaixo estão as descrições das teclas que podem ser usadas para modificar o vídeo. Serão exibidas duas novas telas ao clicar no botão "Start Video", uma vai exibir o vídeo original/natural enquanto a segunda exibirá o resultado do vídeo modificado pelo usuário em tempo real. O usuário tem a opção de usar as teclas de 0 a 9 para modificar o vídeo, ou usar a tecla "r" (minúsculo) para gravar o vídeo editado.

Figure 18: Menu de ações para processamento de vídeo