

Ada Maria Cayres Fernandez
Eric Batista da Silva
Guilherme Oliveira Goularte
Matheus Morandino di Giovanni Carneiro
Renan Ernesto Silva Pinto
Vinicius Alves Matias

Removedor de marca d'água visível para imagens

EP de Processamento de Imagens
ACH2117 - Computação Gráfica
Docente: João Luiz Bernardes Júnior

Julho de 2021
São Paulo / SP

1. Proposta do EP

Este projeto tem como objetivo desenvolver um algoritmo capaz de remover marcas d'água visíveis de figuras estáticas. Para tal, utiliza-se de técnicas de processamento gráfico com ajuda da biblioteca OpenCV (2015) disponível na linguagem de programação Python. Os testes foram realizados utilizando a versão 4.5 da biblioteca e as versões 3.8 e 3.9 do Python no Ubuntu 20.04 e Windows 10.

Marcas d'água são maneiras de evitar que arquivos sejam transferidos na internet sem o reconhecimento autoral devido. Essa técnica pode gerar marcas d'água visíveis à olho nu (foco do EP) ou não. Gonzales e Woods (2018) definem uma imagem com marca d'água simples f_w como a aplicação de uma imagem w com transparência α em uma imagem definida pela função f pela fórmula:

$$f_w = (1 - \alpha)f + \alpha w$$

Como a transparência α em geral varia de 0 à 1, marcas d'água de transparência 1 serão opacas, influenciando que técnicas de remoção de marcas d'água w tenderão a receber influência das cores da figura sobreposta na imagem original conforme a transparência diminui. O algoritmo final reduz essa presença das cores, mas, dada a variedade de imagens possíveis de se aplicar uma marca d'água, haverão figuras em que a remoção não possa ser completamente bem sucedida, ainda que consiga reduzir bastante a influência de w na imagem original.

1.1. Métodos de Processamento Gráfico

Para realizar a transição de uma imagem com marca d'água para uma sem, utilizamos alguns processamentos da figura original que possam gerar um comportamento de diminuição da presença da marca d'água ao final de maneira determinista - não realizando uma diferenciação entre figuras para aplicação dos filtros, tendo para toda entrada as mesmas etapas à serem realizadas, com a exceção do ajuste de um único parâmetro a ser realizado pelo usuário. O algoritmo final utiliza os seguintes processamentos:

Conversão das cores para escala de cinza: Imagens tipicamente transitam na internet em uma composição de três canais de cores: R (vermelho), G (verde) e B (azul). Para evitar processar os três canais (24 bits), assim como permitir a entrada da imagem para outros algoritmos que dependem de um único canal, realizamos uma transformação para uma figura definida em 8 bits de cor (0 à 255), caso da escala de cinza. A maneira que a cor Y de um pixel é definida em escala de cinza pela documentação da biblioteca é como a soma das diferentes intensidades, mas sob uma ponderação da importância de cada cor:

$$RGB \text{ to Gray: } Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

Blur mediana: Operações de *Blur* servem para reduzir a importância de detalhes isolados em uma imagem. Entre as opções de *Blur*, optamos pela mediana como uma forma

de remover avarias em uma imagem, no caso, a intenção era de remover as regiões de marca d'água sem gerar grandes diferenças na imagem. Outros filtros também foram considerados como média e o de Fourier, porém, esses mesclam mais as cores presentes no kernel e geram um resultado onde a região borrada fica mais aparente quando comparada ao uso do blur mediana.

Uma operação de blur considera uma matriz quadrada com o número de linhas/colunas ímpares, pois o pixel central dessa matriz (linha e coluna central) terá sua cor definida como a mediana de toda essa “janela” (neste método de blur). Neste EP a “janela” (chamada formalmente de kernel) é uma matriz 23X23 pixels.

Máscara de seleção: A definição da região de interesse para realizar algum processamento gráfico pode considerar diferentes variáveis que variam de imagem para imagem. Maneiras de detecção automática de regiões de interesse podem ser feitas, por exemplo, pela diferença entre a intensidade das cores, comparação entre sequências de imagens (comum em vídeos) e por intermédio de redes neurais convolucionais (Jan, Zainal e Jamaludin, 2020). Para um conjunto diverso de marcas d'água em imagens (que é o caso deste EP) as abordagens mais generalistas seriam usar Inteligência Artificial (que geram resultados positivos como em Wang et al, 2021) ou por intermédio da definição manual do usuário. Como não é o foco deste projeto a aplicação de técnicas de IA, utilizamos a definição manual pelo usuário para definir a região em que está a marca d'água. Posteriormente essa região selecionada gerará uma máscara para isolar os processamentos relativos a marca d'água (coordenadas x e y que definem o retângulo que está a marca d'água - região de interesse).

Algoritmo de Canny: Esse algoritmo identifica bordas em uma imagem. Esse algoritmo é um pouco mais devagar que o de Sobel visto que identifica bordas que tenham mais destaque na imagem (ainda que possam ser muitas, tipicamente é menos se comparado com a quantidade retornada pelo algoritmo de Sobel). Essas bordas de “maior destaque” são identificadas pois, antes de começar a procurar os contornos, aplica-se um filtro gaussiano (*blur*) na imagem com um kernel 5X5. Após aplicar o blur utiliza-se o filtro de Sobel à horizontal e vertical do pixel para agilizar a coleta da primeira derivada G_x e G_y , respectivamente. A partir delas, define-se a direção do gradiente de cada pixel (que se torna perpendicular ao contorno que está sendo desenhado) como:

$$\text{Edge Gradient}(G) = \sqrt{G_x^2 + G_y^2}$$

Assim como o ângulo que define cada pixel é definido por:

$$\text{Angle}(\theta) = \tan^{-1}(Gy/Gx)$$

Para suprimir pixels que tenham um gradiente direcionado à uma direção diferente das de seus vizinhos devemos identificar se este é um máximo local e, se for, faz parte de um contorno, senão, é suprimido.

Por fim, utiliza-se de um valor mínimo e máximo para identificar se um contorno está com intensidade suficiente para ser considerado um contorno da máscara (no caso, a região de interesse). Definimos como mínimo $\frac{1}{3}$ do threshold definido pelo usuário (inicia-se por padrão como 150, logo, o mínimo é 50), e o máximo como o próprio threshold (150, caso não seja modificado pelo usuário). Importante mencionar que o bom funcionamento do algoritmo depende de uma imagem em escala de cinza (*color-depth* de 8 bits), que foi convertida anteriormente.

Dilatação: O algoritmo de Canny retorna uma imagem não mais em escala de cinza, mas em preto e branco (1 bit de profundidade de cor). Isso permite que utilizemos uma operação morfológica como a dilatação, que consiste em utilizar um kernel (neste EP, 3X3) que trocará preto (cor 0) por branco (cor 1) toda vez que encontrar um pixel do kernel como branco. Isso permite destacar os contornos, logo, remover pequenos contornos que poderiam ser unificados a um único. Um exemplo dessa operação (retirado da documentação do OpenCV) é exibido na figura à seguir:



A esquerda temos a imagem original e à direita a imagem após uma dilatação.

Essa operação foi realizada com a intenção de preencher a área visível da marca d'água visto que elas geralmente consistem de bordas mais relevantes e conteúdo transparente.

AND bitwise modificado: A operação AND / E é uma técnica de processamento que verifica em duas imagens / máscaras em preto e branco o que é branco em ambas, e para cada pixel de tonalidade brancas nas duas máscaras, a operação gera uma máscara nova por meio dessa semelhança (o resto fica preto). No EP é comparada a imagem com o Canny (contornos em branco) com um fundo branco por meio de um AND para coletar os contornos, e em cada pixel do contorno aplicamos a operação de blur mediana na imagem original.

1.2. Outros processamentos utilizados em tentativas

Os processamentos que fazem parte da entrega final foram mencionados no item anterior, contudo, outros processamentos gráficos foram realizados, ainda que não obtivessem um resultado tão natural quanto o final.

Transformada Rápida de Fourier: É uma opção para separar altas frequências(bordas) em uma imagem ou para realizar blur por meio de um filtro passa baixa

(onde apenas as regiões de baixa frequência, superfícies, participariam da imagem final). Entretanto as bordas das marcas d'água não possuem tão alta frequência quanto detalhes da imagem e também não podem ser consideradas um padrão de avaria (com excessão de múltiplas marcas d'água repetidas distribuídas ao longo da imagem), assim, os detalhes da imagem boravam muito enquanto resquícios da marca d'água permaneciam. Por conta disso não se mostrou uma abordagem tão viável quanto o filtro mediana.

Erosão: Na versão alternativa são utilizadas ambas as operações morfológicas dilatação e erosão com um kernel 9X9. A erosão, ao contrário da dilatação, pinta de preto os pixels de sua janela ao encontrar um pixel preto. A sequência dilatação - erosão, conhecida como operação de fechamento, serve para remover detalhes pequenos de diferença de cor entre pixels em seu entorno, considerando o que poderiam ser cores diferentes como uma única e contínua. A figura a seguir (presente na documentação do OpenCV) mostra uma figura original (esquerda) e sua versão após uma erosão (direita).



Threshold adaptativo: Como imagens e intensidade das marcas d'água variam de uma figura para outra, um threshold fixo para identificar o que é contorno ou não é inviável. Maneiras de se encontrar contornos mais robustas (como Canny) são ideais para esses tratamentos de marca d'água que realizamos, contudo, a divergência entre a intensidade de cores de uma região da imagem para outra (que é comum nesses problemas, conforme diminui a transparência da marca d'água) permite que seja identificado um padrão na diferença de intensidade das cores de uma região e, assim, pintar de branco essa região. O resultado utilizando essa abordagem foi satisfatório para um método de threshold adaptativo gaussiano, onde captura-se os 11 pixels vizinhos de um que desejamos calcular se será branco ou não, e calculamos mediante uma maior ponderação de acordo com os mais próximos desse pixel central de 11 vizinhos. Essa ponderação é subtraída por 3 ao fim do método e, caso o resultado da intensidade do pixel fosse maior que o threshold calculado, nós definimos ele como branco, caso contrário preto.

Fast Marching Method (FMM): Para colorir novamente a imagem através de uma máscara de contornos em preto e branco no método alternativo foi utilizada a função INPAINT_TELEA que é uma adaptação do método numérico FMM proposta por Alexandru Telea em 2004. Ela consiste em capturar as cores próximas da região à ser pintada (na comparação entre imagem original e máscara de contornos) e atualizar a cor dos contornos para uma média ponderada entre os pixels próximos, “marchando” até o fim de cada contorno

da imagem, atualizando as cores no processo. Isso implica na visualização final ser, muitas vezes, perceptível como alguns grandes polígonos de uma cor única, já que a cor final depende do conhecimento da vizinhança, e como a vizinhança tende à ser uma marca d'água grossa, esses polígonos (ou outros formatos) ficam mais perceptíveis.

2. Leitura do Código

O código se inicia pelo arquivo main.py que: lida com a manipulação do parâmetro de entrada (path da imagem), instancia a classe responsável por remover a marca d'água e chama o método principal watermark_remover (que pode ser encontrado na linha 21 do arquivo Static_Remover.py).

Esse método então é responsável por lidar com a UI e orquestrar todo o processamento da imagem, ele é dividido em blocos comentados onde métodos são chamados para realizar determinados processamentos na imagem.

Em linhas gerais, são mantidas a imagem original, sua versão em escala de cinza, sua versão com blur mediana aplicado, além de uma máscara contendo somente a porção da imagem selecionada pelo usuário, onde é esperado estar a marca d'água. Após isso, o código entra em um laço onde o algoritmo de Canny é chamado com o high threshold ajustado pelo usuário na UI e o low threshold definido como um terço do high threshold. Com as bordas detectadas pelo algoritmo de Canny é então feito uma dilatação na imagem resultante com a intenção de fazer com que as bordas cubram as bordas da marca d'água. A partir disso a máscara de seleção do usuário é aplicada a imagem resultante da dilatação e por fim na região demarcada é realizada uma troca de todos os pixels que representam uma borda de acordo com o algoritmo de Canny na imagem original pelos pixels correspondentes na imagem com o blur mediana aplicado.

Por conta disso o método trabalha muito bem com marcas d'água de bordas transparentes e conteúdo vazio e em contrapartida não processa tão bem marcas d'água que sejam inteiriças como logos.

No código, especificamente no arquivo Static_Remover.py, há também métodos para aplicação de blur por meio de um filtro passa baixa de Fourier. As transformações de Fourier estão comentadas nesses métodos. Pode-se optar por usar este método de Fourier ao invés do blur mediana descomentando a linha 27, entretanto os testes demonstraram que o filtro mediana é uma melhor escolha para a remoção da marca d'água.

2.1. Versão Alternativa

A primeira versão do EP seguiu algumas abordagens diferentes, mas que ao final não ficaram tão naturais quanto a versão final. Um ponto positivo dessa versão é que ela faz um tratamento melhor para remoção de marcas d'água mais fechadas (como um círculo totalmente da mesma cor) pois o FMM de Telea consegue lidar bem com esses problemas. Essa versão é focada em imagens, mas permitimos uma alteração dela para tratar também vídeos, aplicando o mesmo processamento que seria aplicado em uma figura, mas agora para uma sequência de frames.

O arquivo main.py faz o tratamento da entrada (argumentos da linha de comando) e, dependendo do que o usuário quer, chama a classe Remover de Static_Remover.py para remoção de marcas d'água em uma imagem única, ou a classe Remover de Dinamic_Remover.py para remoção em vídeos.

O removedor de Static_Remover.py realiza o mesmo procedimento para coleta das coordenadas da região de interesse que o EP final. Dadas essas coordenadas, ele gera uma

máscara que consiste de um fundo preto somado a um retângulo branco, retângulo esse que está posicionado na região de interesse. Pegamos essa máscara e aplicamos um bitwise OR que, ainda que não mencionado diretamente até agora, funciona adicionando apenas a imagem original na região em branco (logo, temos uma imagem que é um fundo preto, com exceção do retângulo da região de interesse, que agora tem a imagem da marca selecionada pelo usuário na região correta). Convertemos essa máscara para escala de cinza, aplicamos o método de threshold adaptativo e temos, assim, um fundo preto e certas regiões contínuas em branco. Uma dessas regiões é a borda do retângulo, então fazemos uma máscara menor (outro retângulo) e aplicamos um AND com a imagem original para remover essa borda. Aplicamos um threshold entre o que é menor que 127 e o que é maior apenas para coletar os contornos (pois só existe preto e branco nessa imagem), e desenhamos eles na máscara final (o que não era um contorno fica preto). Fazemos uma erosão, dilatação e um filtro média (média da intensidade dos pixels em um kernel 7X7 define a cor do pixel central) e aplicamos esses contornos identificados na imagem original, que pintará essas regiões em branco pelo método de Telea.

Dinamic_Remover.py faz os mesmos processamentos, com diferença que agora esse arquivo lê frames de um vídeo e não imagens. Exibimos o frame 200 do vídeo para o usuário selecionar a região de interesse e a partir daí são tratados todos os frames desde o início como uma imagem, fazendo o processamento descrito anteriormente.

3. Compilação e Execução do Código

O projeto foi escrito na linguagem de programação Python e utilizando o framework de manipulação de imagens OpenCV. Os testes foram realizados utilizando a versão 4.5 da biblioteca e as versões 3.8 e 3.9 do Python no Ubuntu 20.04 e Windows 10.

Para compilar o código é necessário os dois requisitos acima (linguagem e framework) além da biblioteca numpy que é requisito do OpenCV. Para instalar o Python basta baixar o instalador do site oficial e seguir as instruções da instalação: <https://www.python.org/downloads/>

Após a linguagem instalada, para instalar o framework basta rodar os seguintes comandos no terminal do Linux ou Windows:

```
pip install numpy && pip install opencv-contrib-python
```

Agora, para compilar e rodar o código basta rodar o seguinte comando:

```
python3 main.py image_path ou python main.py image_path
```

 onde `image_path` é o caminho da imagem a ser processada. Exemplo:

```
python main.py images/test.jpg  
python3 main.py '/home/JB/Área de Trabalho/CG/test.jpeg'
```

3.1. Versão Alternativa

Para executar a versão anterior do projeto (não a final) basta ir na pasta Alternative_version e, de maneira semelhante à mencionada anteriormente, chamar o comando `python3` (ou `python / py`, dependendo do sistema operacional) no arquivo `main.py`, como exemplo:

Para imagens:

```
python3 main.py test.jpeg image
```

Para vídeos:

```
python3 main.py test.mp4 video
```

Para vídeos, especificando o framerate:

```
python3 main.py test.mp4 video 30
```

Ou seja, o formato para remoção de marca d'água em imagens é: `python3 main.py image_path image` onde `image_path` é o caminho da imagem a ser processada e `image` é uma flag para identificarmos que deve ser processada uma imagem.

Para vídeos o formato é `python3 main.py video_path video framerate` onde `video_path` é o caminho do vídeo a ser processada, `video` é uma flag para identificarmos que deve ser processada um vídeo e `framerate` é um inteiro para especificar em que taxa de frames devemos salvar o vídeo modificado (caso não seja entregue como argumento, perguntaremos antes de iniciar a conversão).

4. Como usar

Ao rodar o programa uma janela com a imagem aparecerá, o usuário deve então selecionar a porção da imagem correspondente a marca d'água com o mouse.



Após isso o usuário deve apertar a tecla ‘Enter’. Depois, uma janela com a imagem já processada aparecerá, no topo desta janela existe uma barra para ajustar a quantidade de threshold do algoritmo de Canny, o usuário pode ajustar o parâmetro dessa barra arrastando para qualquer um dos lados de acordo com o que gerar um melhor resultado final na imagem processada.



Por fim, para encerrar o programa basta apertar qualquer tecla. A nova imagem será salva no mesmo diretório da imagem de entrada.

O processo é o mesmo para a versão alternativa.

5. Testes

Inicialmente tentamos aplicar diferentes filtros de blur na região selecionada da imagem, entretanto os resultados ficaram com uma divisão aparente entre a região borrada e o restante da imagem independente do filtro usado.

Após isso pensamos em utilizar Canny para detecção das bordas e diminuir a seleção para os contornos, entretanto não conseguimos usar nenhuma forma de reconhecimento de objetos pelos contornos para então preencher esses objetos para o futuro blur.

Foi então que surgiu a ideia de utilizar o fechamento na imagem resultante do Canny e aplicar o filtro de blur somente nas regiões demarcadas pelo Canny. No fim conseguimos fazer isso guardando uma imagem com o blur aplicado por completo e trocando os demarcados pelo Canny na imagem original pelos pixels da imagem com blur.

A partir daí testamos diversos filtros de blur, parametrizações desses filtros e do algoritmo de Canny, e a operação de dilatação ao invés de fechamento.

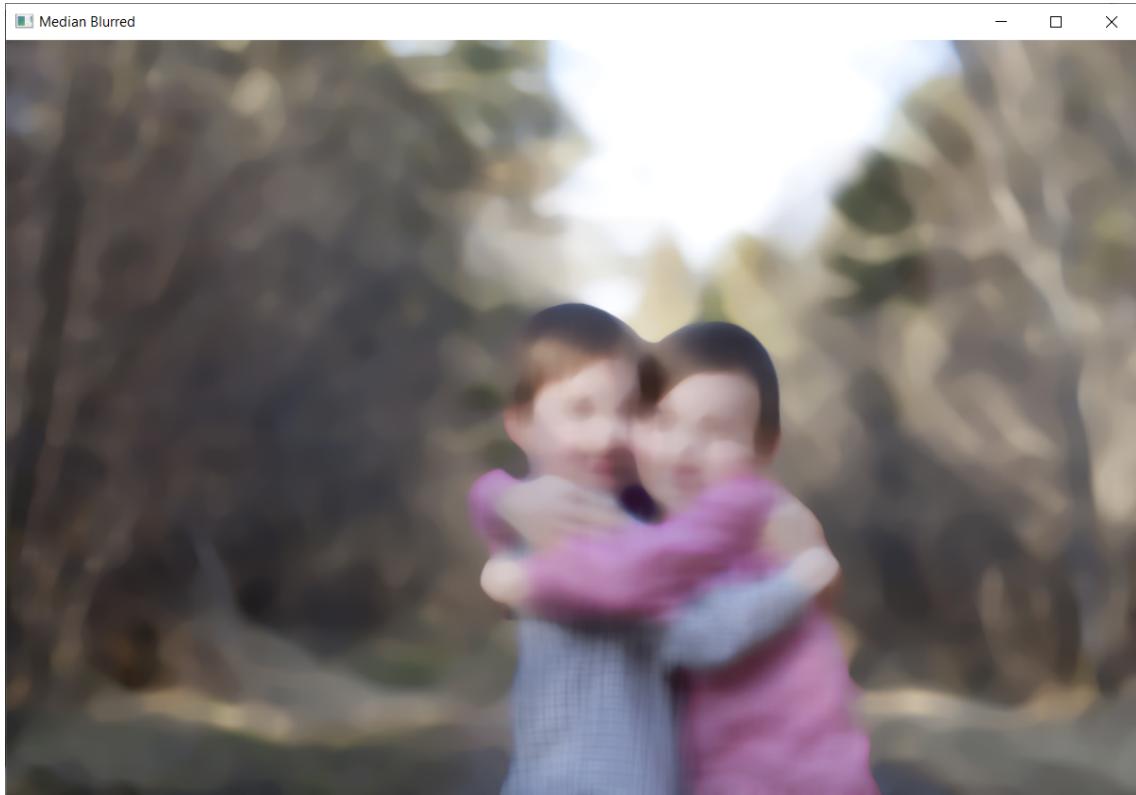
Percebemos então que a parametrização mais sensível às diferentes composições de fundo com a marca d'água era o threshold de Canny e decidimos criar um trackbar para o usuário ajustar esse parâmetro conforme cada imagem de entrada.

Passo a passo das transformações até o resultado final:

1. Transformar em escala de cinza



2. Guardar separadamente imagem com filtro mediana aplicado



3. Mascara de acordo com a seleção do usuário, esperado ser na região da marca d'água



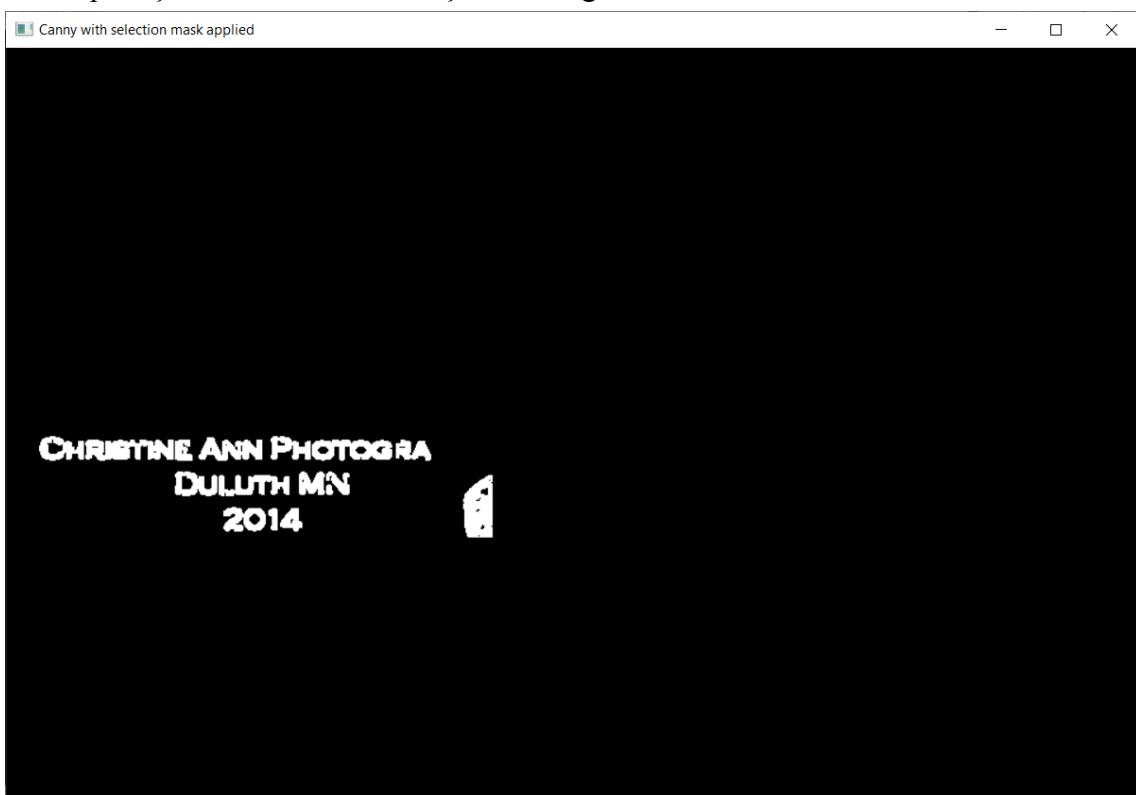
4. Aplicação do algoritmo de Canny na imagem grayscale



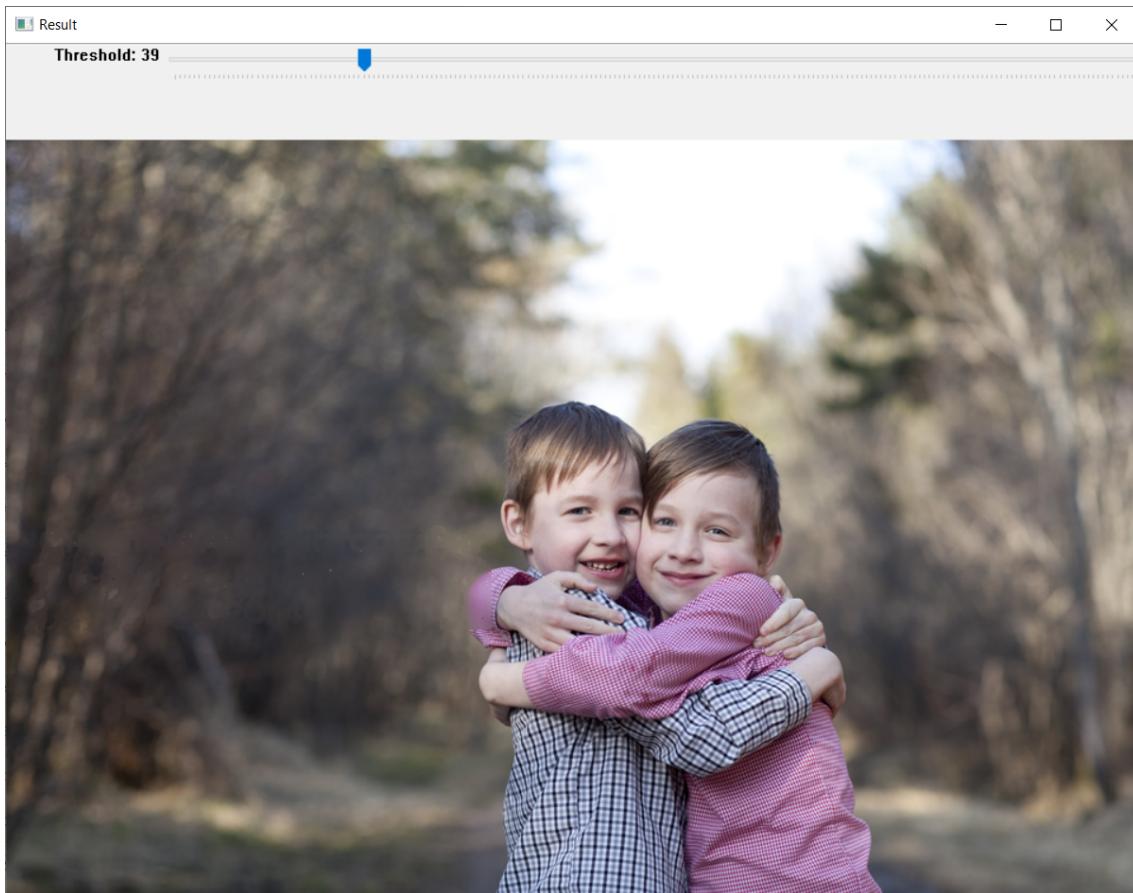
5. Dilatação da imagem resultante do algoritmo de Canny



6. Aplicação da máscara de seleção na imagem acima

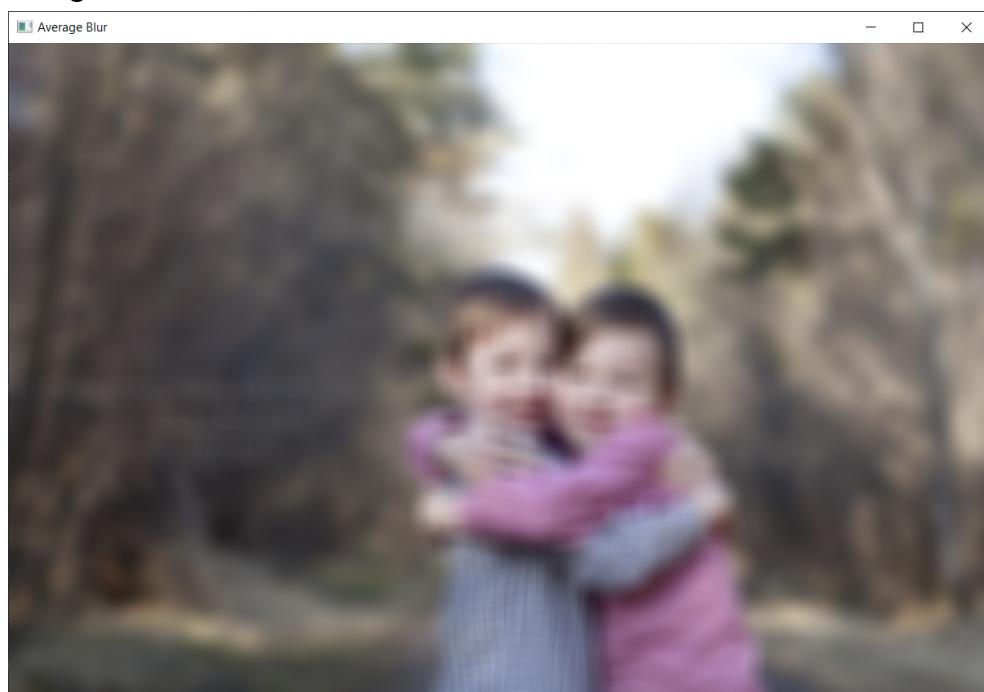


7. Imagem resultante após trocar na imagem original os pixels correspondentes às regiões em branco da imagem acima com os pixels da imagem com blur aplicado

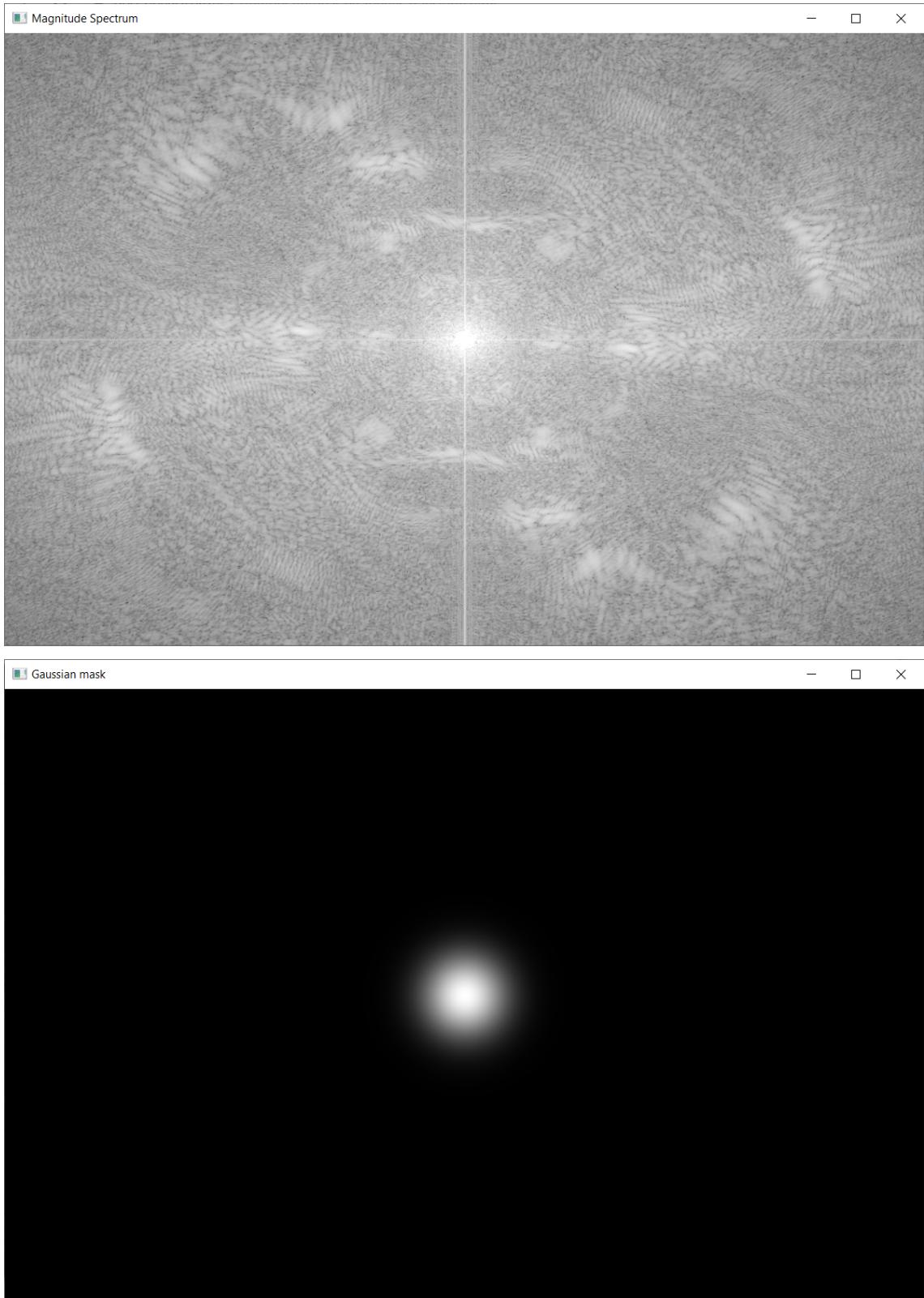


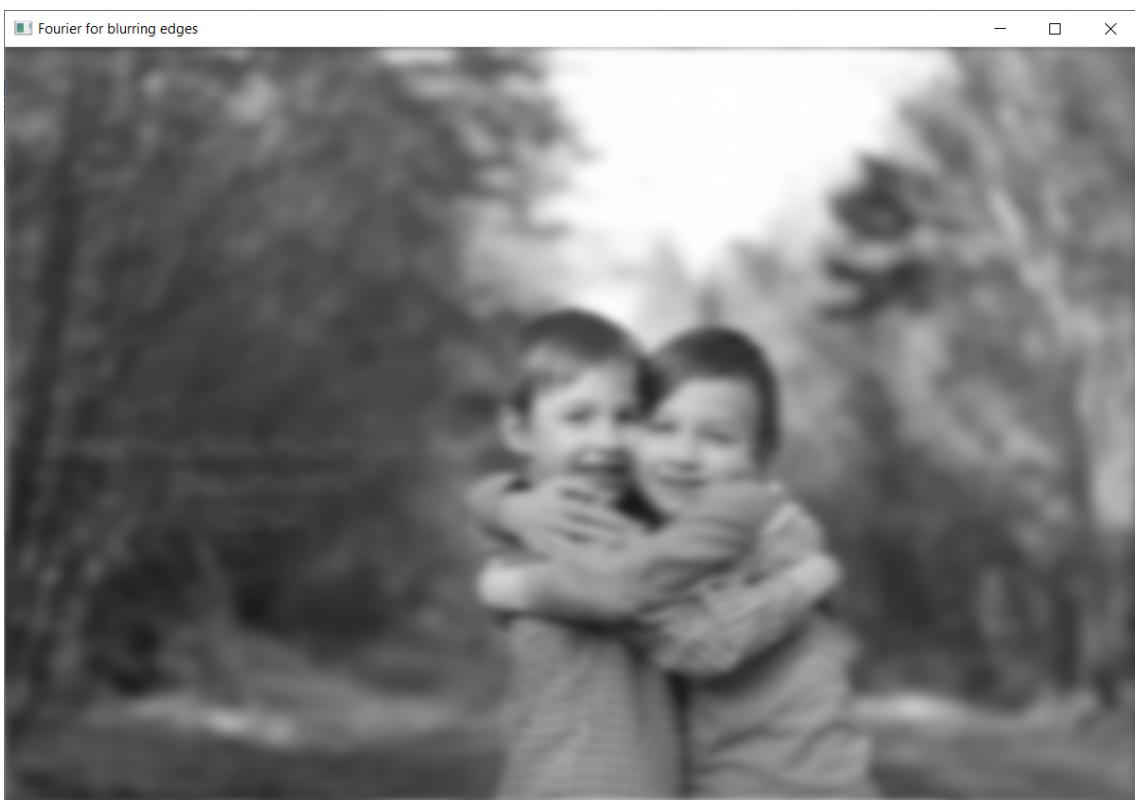
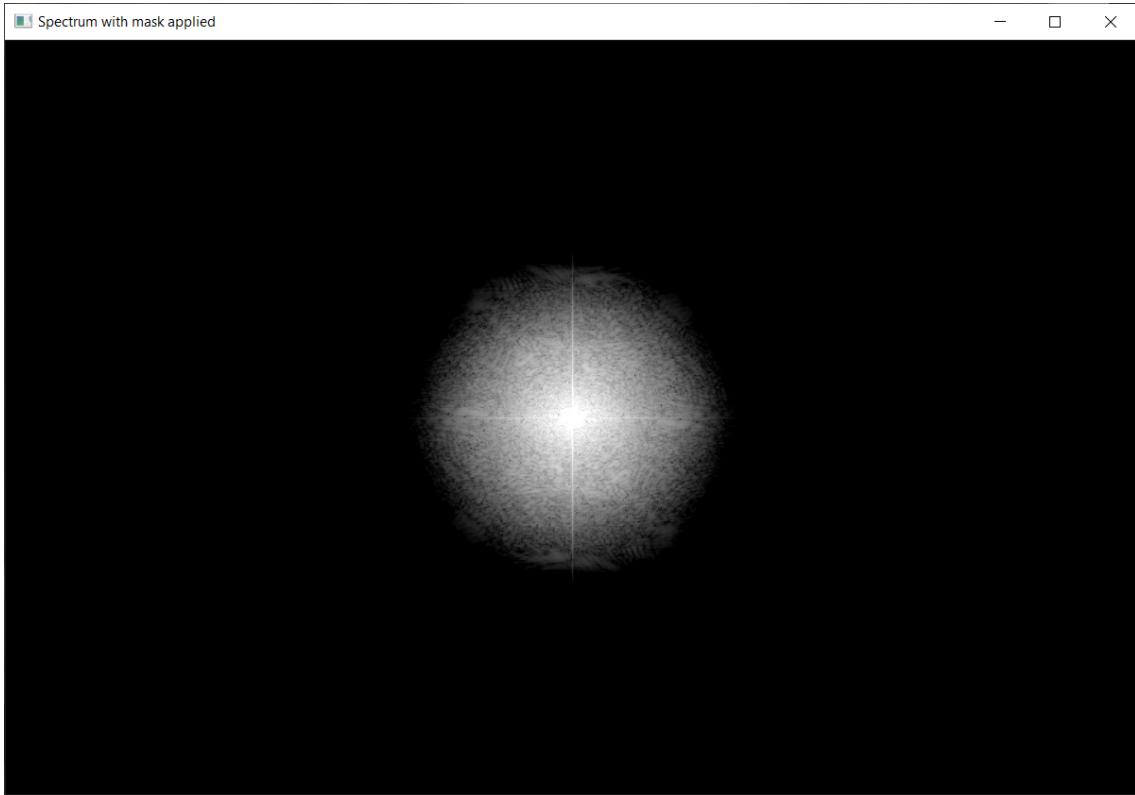
Testes com outros tipos de blur:

- Average blur:



- Fourier blur:



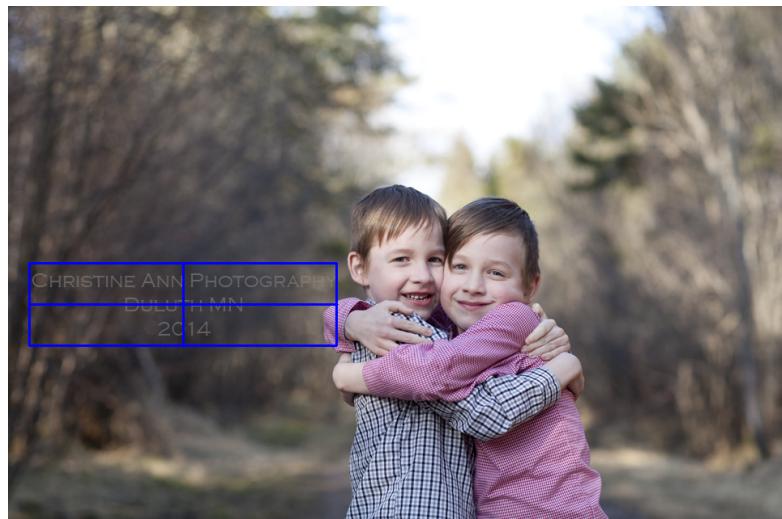


5.1. Versão Alternativa

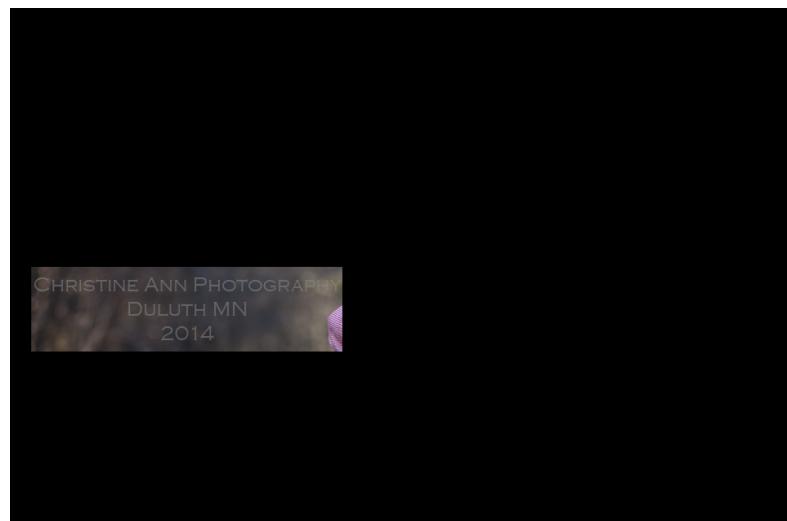
Além das considerações já feitas no início do tópico, destaca-se que a versão com threshold adaptativo considera outros detalhes além da marca d'água da imagem que acabam influenciando no resultado final propositalmente, pois o método de pintura pode preencher de

maneira mais completa os pixels de regiões “cheias” (replicando as cores de seus vizinhos) e perdendo em parte o identificador que poderia estar no meio da marca d’água.

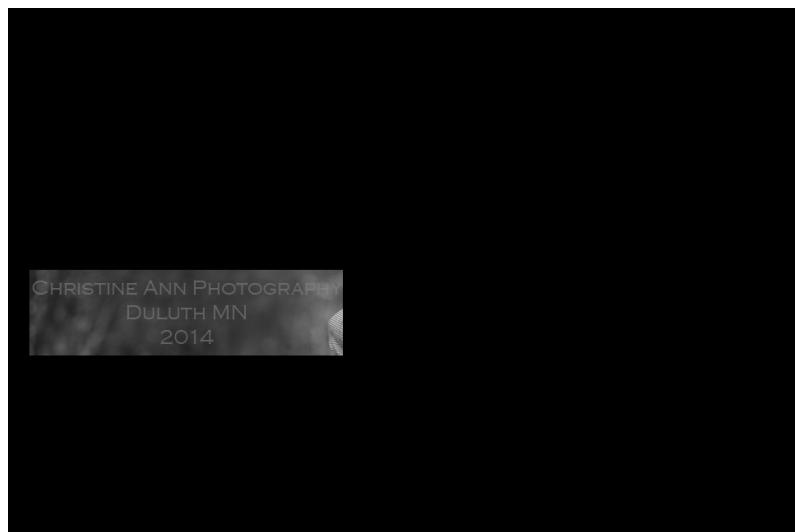
Região de Interesse:



Bitwise OR:



Escala de Cinza:



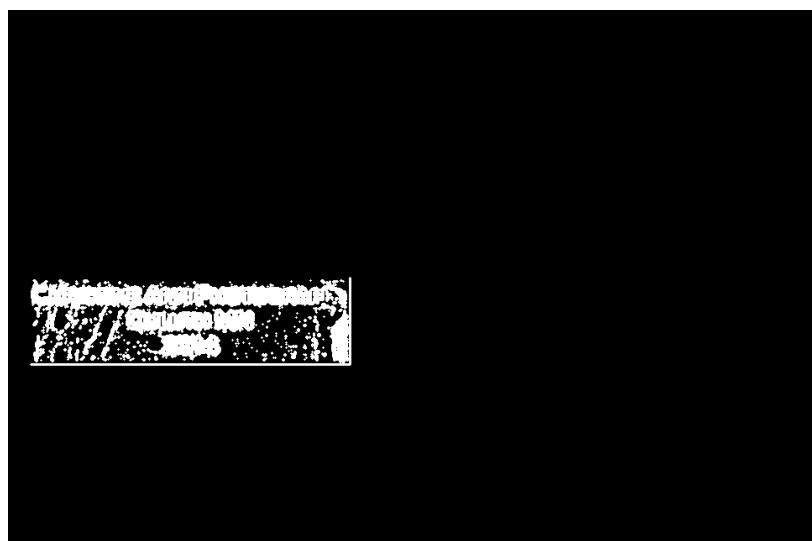
Threshold Adaptativo:



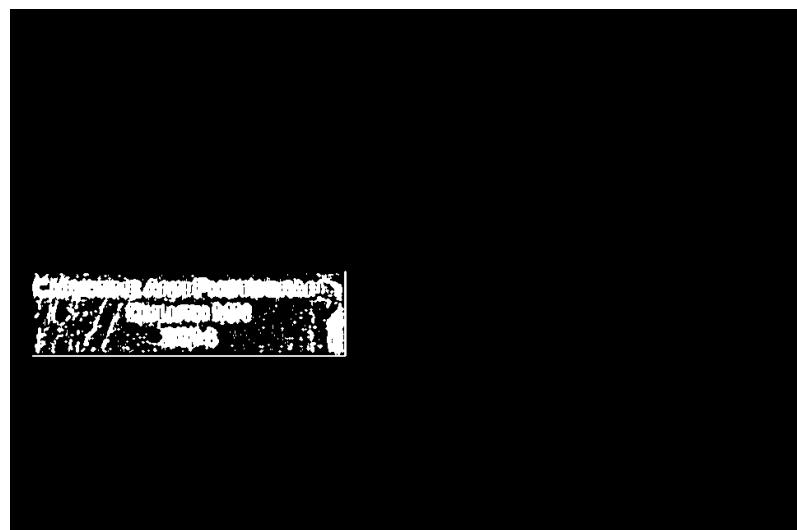
AND:



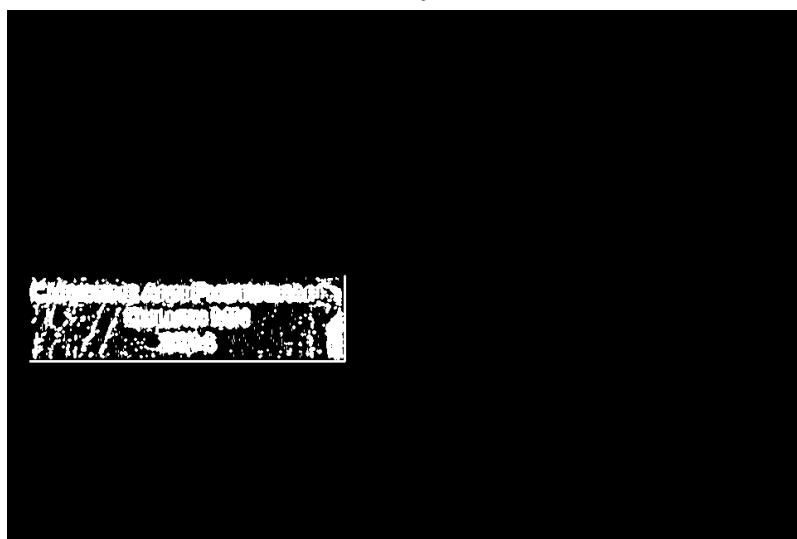
Desenhar contornos:



Erosão:



Dilatação:



Blur:



Final:



6. Observações

Existiram problemas no uso da função de callback do trackbar, a manipulação da quantidade de threshold na barra chama essa função, onde idealmente deveria estar o reprocessamento do Canny, dilatação e troca dos pixels como explicado na proposta e seção de testes. Entretanto usar a função de callback trocaria o paradigma orientado a objetos usado na implementação do EP, assim, quase todas as variáveis deveriam ter escopo global e o código se tornava menos legível, por conta disso preferimos manter o processamento em um loop que utiliza a variável threshold ajustada no trackbar, porém para isso o programa constantemente reprocessa as etapas elencadas acima a cada 60ms enquanto uma tecla não é pressionada para seu encerramento.

Também, não encontramos na documentação como manipular o tamanho do trackbar sem estabelecer um tamanho fixo para as imagens de entrada. Durante os testes foi visto que em imagens muito pequenas a barra é cortada e para manipular ela precisa-se alterar o tamanho da janela ou colocar em tela cheia.

A remoção da marca d'água em vídeos ou imagens sempre se dá mais completa quando se realiza um redimensionamento da imagem pela metade, contudo, isso gera uma perda de resolução significativa pela interpolação da área perder as informações da imagem original. Como o EP visa retornar uma imagem que possa ser reutilizável legalmente, não é o objetivo do usuário ter uma perda na qualidade da imagem mesmo que tenha um resultado melhor ou mais veloz (quando se trata de vídeos com 60 ou mais frames por segundo, principalmente). Por conta disso, na versão final todas as transformações significativas foram feitas com imagens no tamanho original. Além disso, o resultado final é sobreposto sobre a imagem original em sua resolução padrão.

7. Fontes

A principal fonte utilizada foi o vídeo “OpenCV Course - Full Tutorial with Python” do canal “freeCodeCamp.org” disponível no youtube. Utilizamos ele como base para uso do framework de processamento de imagens OpenCV, além do teste de técnicas mostradas no tutorial como aplicação de blur, criação de máscaras, e algoritmo de Canny. Além do vídeo, em todo o decorrer da implementação (assim como nas explicações deste relatório) foi usado a documentação do OpenCV para Python.

Houve uma breve tentativa de revisão de projetos de código aberto para remoção de marca d'água porém todos utilizavam inteligência artificial.

Também, foram feitas pesquisas de manipulação de regiões de interesse na imagem e, por fim, para os testes usando filtro de Fourier foram revisadas várias fontes como a documentação do OpenCV, Stackoverflow, além de blogs e sites explicando as manipulações da magnitude e as operações de transformação no domínio da frequência.

8. Vídeo

[Link](#) para o vídeo da apresentação:

https://drive.google.com/file/d/1S1NH_qIM5G07zpWS3Zxk1uV6bOzlu9rL/view?usp=sharing

Imagens e vídeos usados na apresentação podem ser baixados no [Google Drive, por meio do email USP](#).

Referências

Craig Chen. (2020). **Digital Image Processing using Fourier Transform in Python.** Disponível em:
<https://hicraigchen.medium.com/digital-image-processing-using-fourier-transform-in-python-bcb49424fd82>

freeCodeCamp.org. (2020). **OpenCV Course - Full Tutorial with Python.** Disponível em:
<https://youtu.be/oXlwWbU8l2o>.

Gonzalez, R. C. e Woods, R. E. (2018). **Digital image processing.** Edição Global. Pearson. Capítulo 8.

Jan, M. M., Zainal N. e Jamaludin, S (2020). **Region of interest-based image retrieval techniques: a review.** IAES International Journal of Artificial Intelligence (IJ-AI). Vol. 9, No. 3, Setembro de 2020, p. 520-528. DOI: 10.11591/ijai.v9.i3.pp520-528.

OpenCV. (2015). **Open Source Computer Vision Library.**

OpenCV Documentation.

Arithmetic Operations on Images. Disponível em:
https://docs.opencv.org/4.5.2/d0/d86/tutorial_py_image_arithmetics.html.

OpenCV Documentation. **Canny Edge Detection.** Disponível em:
https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html.

OpenCV Documentation. **Color conversions.** Disponível em:
https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html.

OpenCV Documentation. **Eroding and Dilating.** Disponível em:
https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html.

OpenCV Documentation. **Image Inpainting.** Disponível em:
https://docs.opencv.org/4.5.2/df/d3d/tutorial_py_inpainting.html

OpenCV Documentation. **Image Thresholding.** Disponível em:
https://docs.opencv.org/4.5.2/d7/d4d/tutorial_py_thresholding.html

OpenCV Documentation. **Smoothing Images.** Disponível em:
https://docs.opencv.org/4.5.2/d4/d13/tutorial_py_filtering.html.

OpenCV Documentation. **Image Inpainting.** Disponível em:
https://docs.opencv.org/4.5.2/df/d3d/tutorial_py_inpainting.html.

ProgrammerSought. **OpenCV-Python - Chapter 19: Fourier Transform.** Disponível em:
<https://www.programmersought.com/article/8437785923/>.

Stackoverflow. (2020). **How to inverse a DFT with magnitude with opencv python.** Disponível em:

<https://stackoverflow.com/questions/59975604/how-to-inverse-a-dft-with-magnitude-with-opencv-python>.

Wang, H., Xue M., Sun, S.; Zhang Y., Wang, J., Liu W. (2021). **Detect and remove watermark in deep neural networks via generative adversarial networks.** arXiv:2106.08104.