

UNIVERSIDADE FEDERAL DA PARAÍBA
PROCESSAMENTO DIGITAL DE IMAGENS

RELATÓRIO DO TRABALHO PRÁTICO

LUCAS ISAAC PISSAIA DA SILVA
LUCIANO PEREIRA DE OLIVEIRA FILHO

Abril, 2023

1. Introdução

Este relatório apresenta o projeto de um sistema para abrir, exibir, manipular e salvar imagens RGB com 24 bits/pixel (8 bits/componente/pixel), sem o uso de bibliotecas ou funções especiais de processamento de imagens. A proposta envolve a implementação de diversas funcionalidades, como a conversão RGB-YIQ-RGB, negativo em RGB e na banda Y, correlação m x n com filtros definidos em um arquivo txt, bem como filtro de mediana.

Para fundamentar teoricamente o projeto, serão abordados conceitos relacionados a imagens digitais e seu processamento. Serão apresentados detalhes sobre as transformações de cores RGB-YIQ-RGB e sobre as operações de negativo em RGB e banda Y. Também será discutido o conceito de correlação, que é fundamental para a implementação dos filtros definidos em arquivo.

Os objetivos deste projeto são implementar as funcionalidades propostas e testá-las com diferentes filtros e parâmetros, além de comparar os resultados e tempo de processamento obtidos por diferentes filtros.

1.1. Imagem digital e Processamento Digital de Imagens

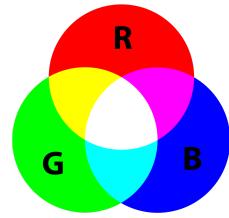
Uma imagem digital é uma representação bidimensional de uma imagem usando números binários que permitem que a imagem seja armazenada, transmitida, impressa, reproduzida e processada eletronicamente. Existem dois tipos principais de imagens digitais: as imagens rasterizadas e as imagens vetoriais. As imagens rasterizadas são formadas por um conjunto finito de pontos chamados pixels, que têm uma posição e um valor de cor definidos por coordenadas e quantificações numéricas. As imagens vetoriais são formadas por elementos geométricos como linhas, curvas e polígonos com atributos de cor e forma definidos por fórmulas matemáticas. As imagens rasterizadas são mais adequadas para representar imagens fotográficas, enquanto as imagens vetoriais são mais adequadas para representar desenhos técnicos e logomarcas.

Processamento digital de imagens é uma área da ciência da computação que estuda e aplica técnicas de manipulação e análise de imagens digitais. O processamento digital de imagens visa extrair informações relevantes das imagens, melhorar sua qualidade, restaurar defeitos, comprimir seu tamanho, entre outras finalidades. Para isso, são utilizados algoritmos e operações matemáticas que modificam os valores dos pixels de acordo com algum critério ou objetivo. O processamento digital de imagens pode ser aplicado em diversas áreas, como medicina, engenharia, astronomia, segurança, arte, entretenimento, reconhecimento de padrões, detecção de objetos, análise de satélite, monitoramento ambiental, entre outras.

Dentre as técnicas utilizadas no processamento digital de imagens, destacam-se as de filtragem, que remove ruídos e reforçam características importantes da imagem, as de segmentação, que dividem a imagem em regiões com características similares, e as de reconhecimento de padrões, que identificam objetos e formas específicas na imagem.

1.2 Espaço de cores RGB e YIQ

O espaço de cores RGB é um sistema de cores que constrói todas as cores a partir da combinação das cores vermelho (R), verde (G) e azul (B). Cada banda de cor pode ter um valor inteiro de 0 a 255, o que resulta em 16777216 cores possíveis. O espaço de cores RGB é usado em monitores de computador e outros dispositivos que emitem luz.



O espaço de cores YIQ é um sistema de cores que separa a luminosidade (Y) da cromaticidade (I e Q). O espaço de cores YIQ é usado no sistema NTSC de transmissão de televisão, pois permite a compatibilidade com as TVs em preto e branco, que usam a componente Y. A relação entre as cores dos espaços RGB e YIQ pode ser expressa por uma matriz de transformação apresentada abaixo:

Do RGB ao YIQ

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Do YIQ ao RGB

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0.956 & 0.621 \\ 1 & -0.272 & -0.647 \\ 1 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

1.3 Correlação

A correlação é uma operação matemática que é utilizada para fins de encontrar padrões em uma imagem ou em uma parte da imagem. É um conceito importante no processamento de imagens, pois permite extrair informações relevantes a partir de dados visuais. Ela envolve comparar uma imagem com um padrão ou máscara predefinida, normalmente uma matriz de números. A máscara percorre sobre a imagem, e para cada posição, os valores dos pixels da imagem e da máscara são multiplicados e somados. O resultado é então armazenado em uma nova matriz (uma nova imagem)

1.4 Filtros

1.4.1 Média

O filtro média é uma técnica para suavizar uma imagem, reduzindo as variações de intensidade entre os pixels próximos, o filtro funciona calculando a média dos valores de intensidade dos pixels em uma determinada região da imagem definida por uma máscara, sendo uma matriz de tamanho fixo onde cada ponto tem o valor de $1/\{\text{quantidade de elementos na máscara}\}$. A máscara é deslizada sobre a imagem, e para cada posição, a média dos valores dos pixels da região correspondente é calculada. O valor médio é atribuído ao pivô que geralmente é o pixel central da região.

Uma das principais vantagens do filtro média é que ele é simples e eficiente em termos de processamento, se for bem aplicado, a exemplo de usar um filtro 1x11 e 11x1 que envolvem 22 multiplicações ao invés de um único filtro 11x11 que envolvem 121 multiplicações. No entanto, ele é sensível a ruídos na imagem, o que pode levar a uma perda de detalhes importantes

1.4.2 Mediana

O filtro mediana é uma técnica utilizada para remover ruídos ou imperfeições em uma imagem, preservando os detalhes importantes. Ao contrário do filtro média, que usa a média dos valores dos pixels para suavizar a imagem, o filtro mediana usa a mediana dos valores dos pixels em uma determinada região da imagem. Isso significa que o valor do pixel central é substituído pelo valor da mediana dos valores de intensidade dos pixels na região correspondente da imagem. A mediana é eficiente em relação a valores extremos ou valores discrepantes, o que significa que ela é menos suscetível a ser influenciada por ruídos na imagem.

1.4.3 Sobel

O filtro Sobel é uma técnica utilizada para detectar bordas em uma imagem que usa duas máscaras, uma para detecção de bordas horizontais e outra para detecção de bordas verticais. As máscaras são matrizes de tamanho fixo (3x3) que são correlacionadas com a imagem original. O resultado da correlação é uma imagem de gradiente que mostra as mudanças de intensidade ao longo dos eixos horizontal e vertical da imagem.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

O próximo passo é combinar as duas imagens de gradiente para obter uma única imagem de bordas. Isso é feito calculando a magnitude do gradiente em cada pixel usando a fórmula:

$$magnitude = \sqrt{(gradienteX^2 + gradienteY^2)}$$

Uma das principais vantagens do filtro Sobel é sua capacidade de detectar bordas em uma imagem de forma eficiente e precisa. Ele é capaz de detectar bordas em diferentes direções e em diferentes níveis de intensidade

1.4.4 Emboss

O filtro emboss é uma técnica usada para criar um efeito de relevo em uma imagem. A matriz consiste em valores positivos e negativos, que são usados para calcular a diferença entre os valores dos pixels adjacentes na imagem, tal resultado é somado a um offset (127). Existem vários tipos de filtros emboss, o utilizado para este trabalho foi este abaixo:

-1	0
0	1

1.4.5 Gaussiano

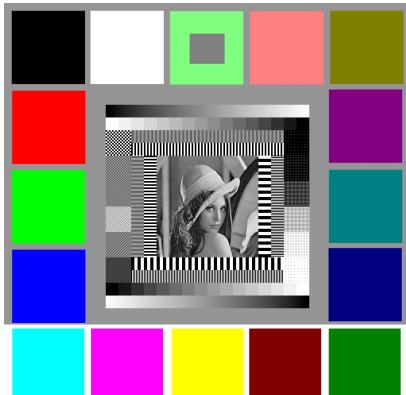
O filtro gaussiano é uma técnica de processamento de imagens usada para suavizar a imagem original, reduzindo o ruído e removendo detalhes finos. O filtro é chamado de gaussiano porque utiliza uma distribuição gaussiana para determinar o peso de cada pixel na imagem original. Em comparação com o filtro de média, o filtro gaussiano preserva as bordas existentes e é capaz de suavizar a imagem sem criar bordas artificiais, gerando um "borramento" mais realista.

2. Materiais e Métodos

O trabalho foi implementado utilizando o ambiente VSCode e a linguagem de programação Python, com a biblioteca numpy para manipulação de arrays. Inicialmente, o OpenCV foi utilizado para processamento de imagens, porém devido ao seu canal de cores invertido (BGR), foi substituído pela biblioteca Pillow. Para apresentação da imagem, o matplotlib foi empregado. Ao longo do desenvolvimento do projeto, além das ferramentas mencionadas, outras bibliotecas como argparse, pathlib e statistics foram incorporadas. Inicialmente o projeto seria executado sem argumentos, mas à medida que o projeto foi se desenvolvendo o programa passou a funcionar com entrada de argumentos para determinar a operação a ser feita

Os conhecimentos utilizados provém do conteúdo próprio da disciplina e consultas a internet

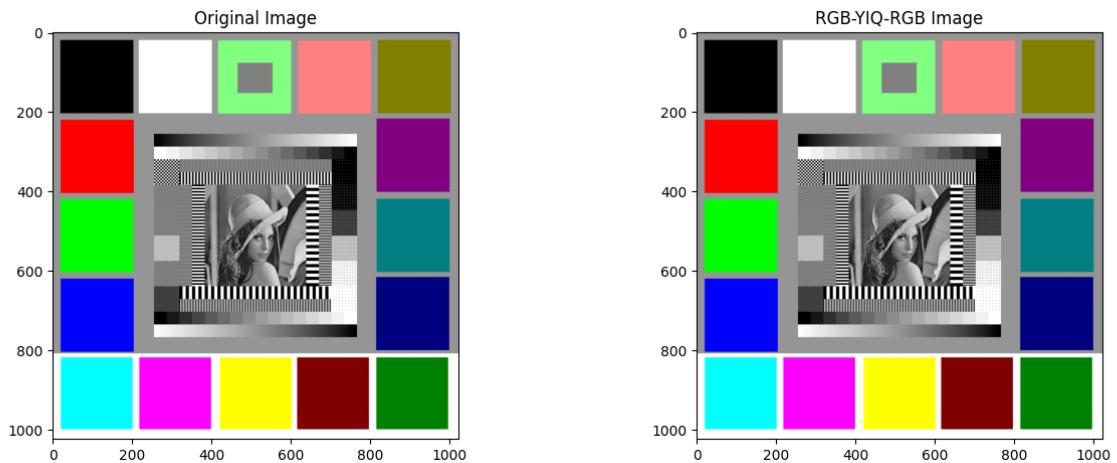
Sobre o uso das imagens, uma imagem de teste geral foi cedida pelo professor e uma imagem do centro de informática (a direita) foi obtida através de pesquisa



3. Resultados

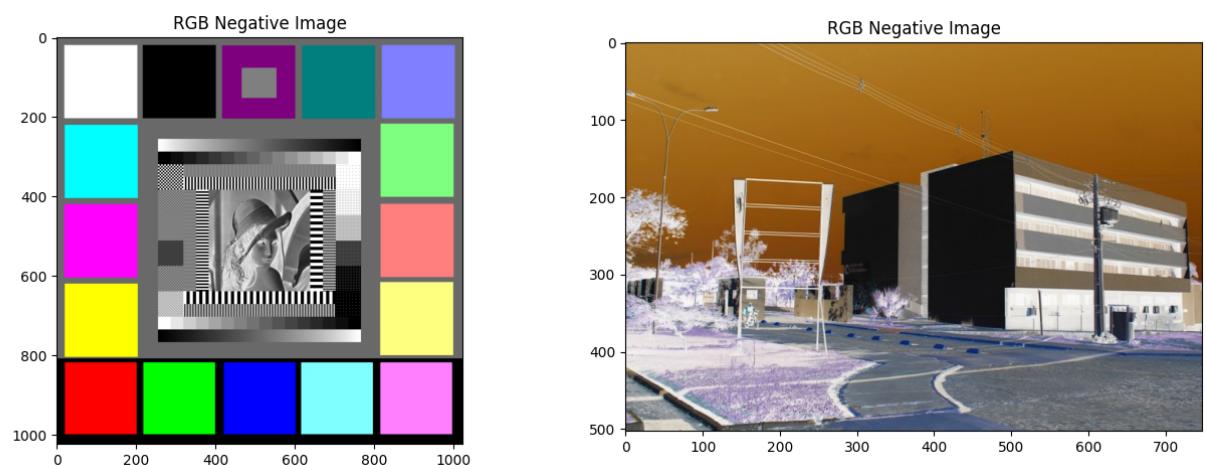
3.1 Conversão RGB-YIQ-RGB

Primeira questão do projeto, foi pedido uma conversão de uma imagem RGB para o sistema YIQ

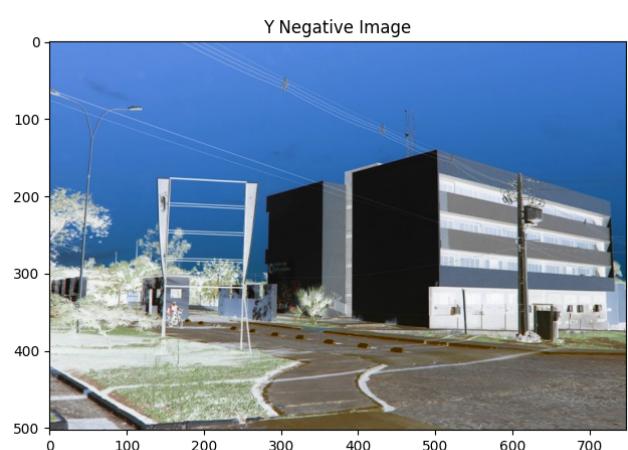
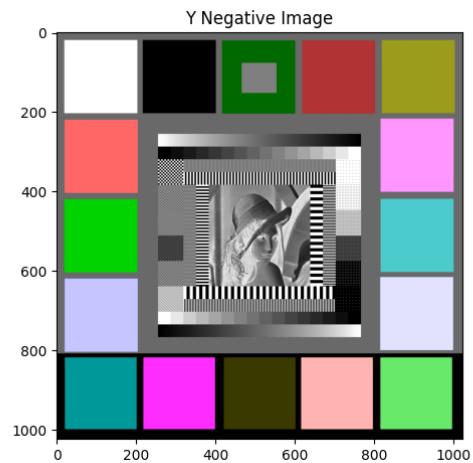


3.2 Negativos

3.2.1 Em RGB (banda a banda)

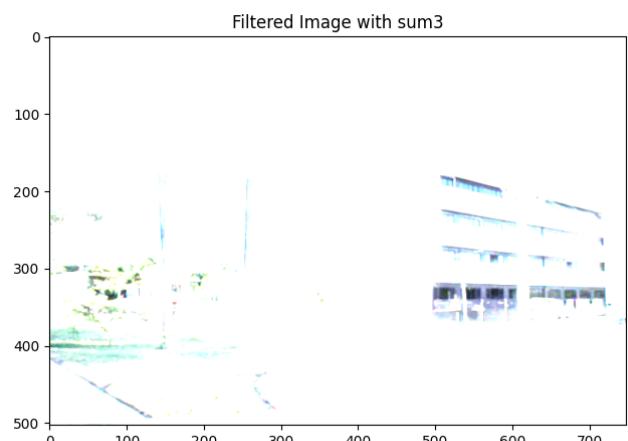
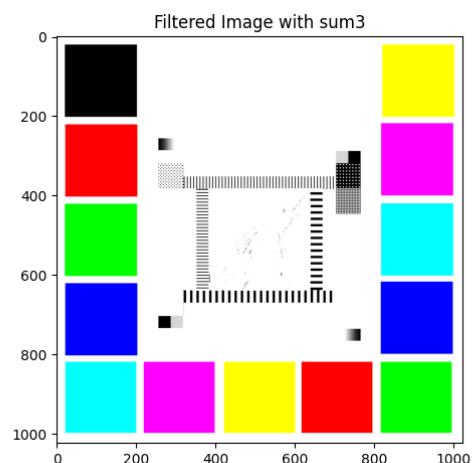


3.2.2 Na banda Y, com posterior conversão para RGB

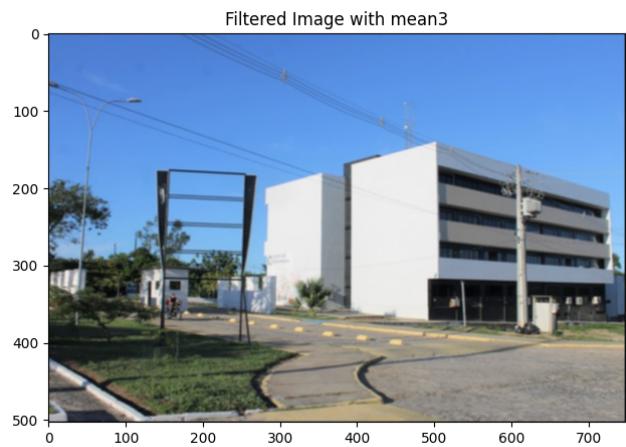
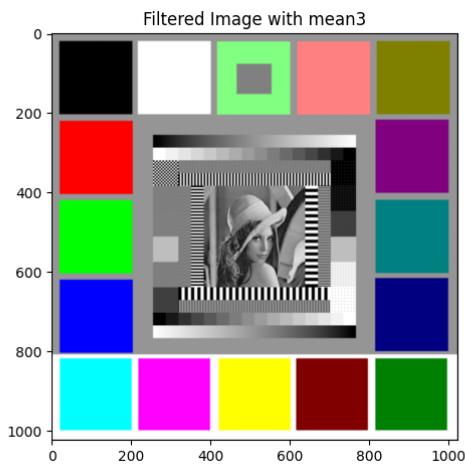


3.3 Correlações

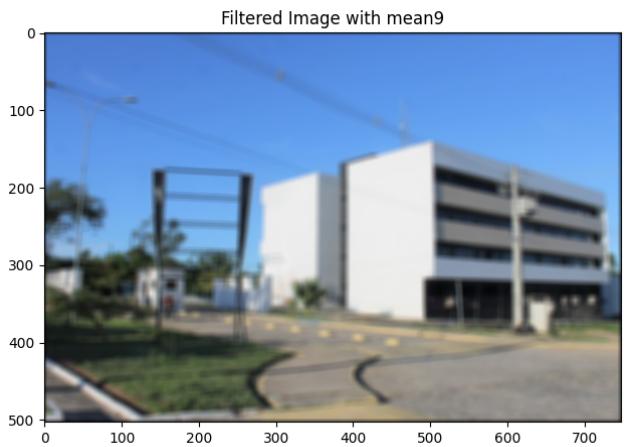
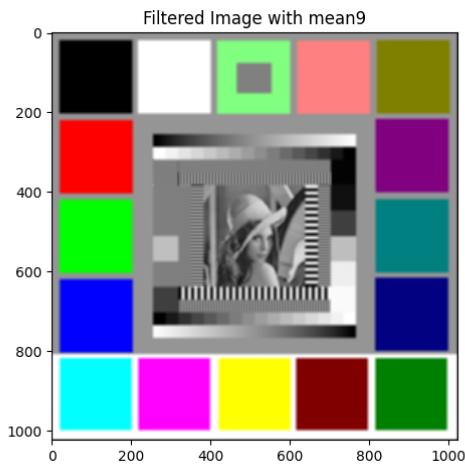
3.3.1 Soma



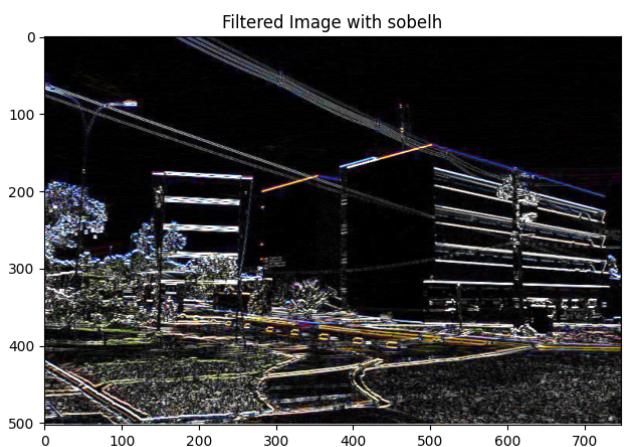
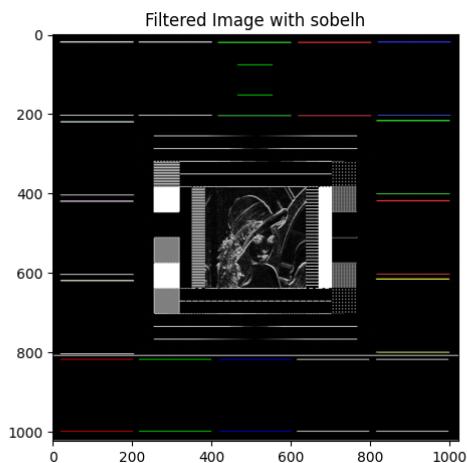
3.3.2 Média (3x3)



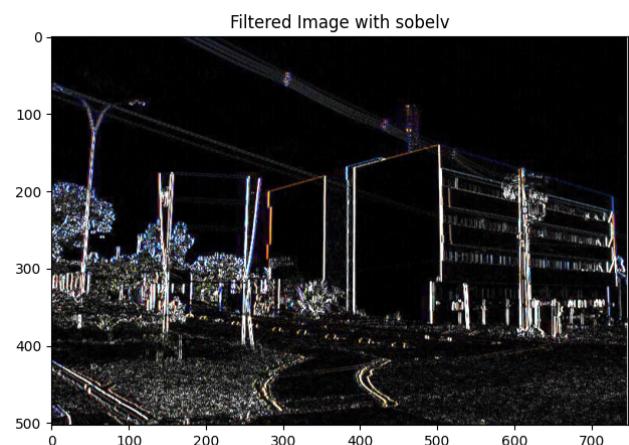
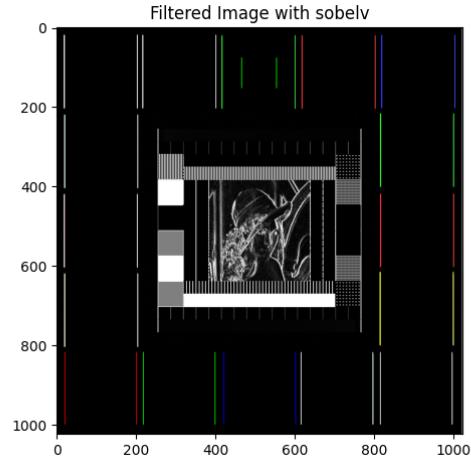
3.3.3 Média (9x9)



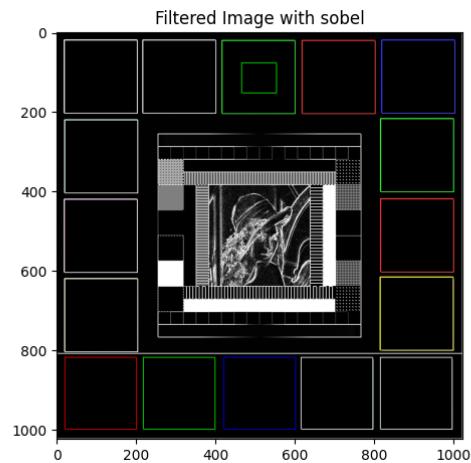
3.3.4 Sobel-Horizontal



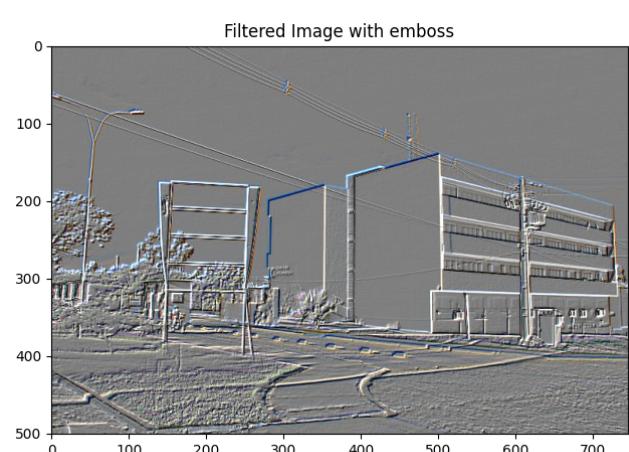
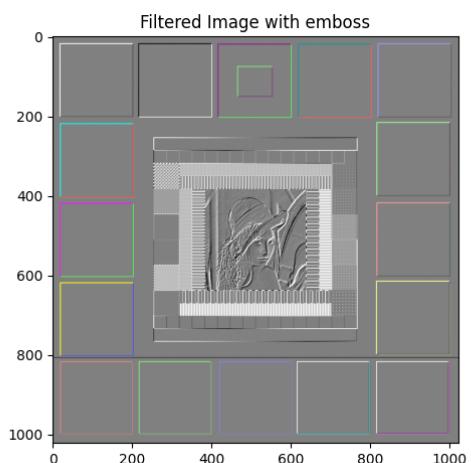
3.3.5 Sobel-Vertical



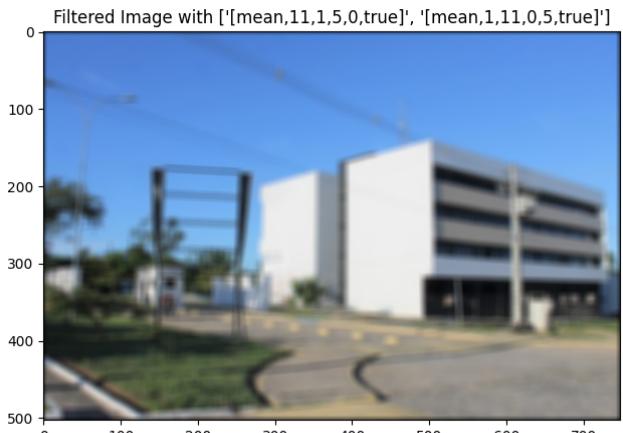
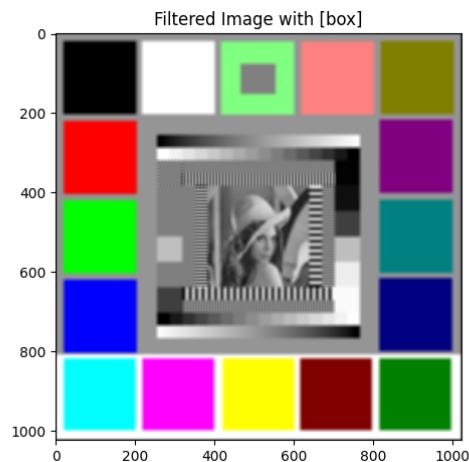
3.3.6 Sobel Completo



3.3.7 Emboss



3.3.6 Comparação Média 11x1 com Média 1x11 * Média 11x1



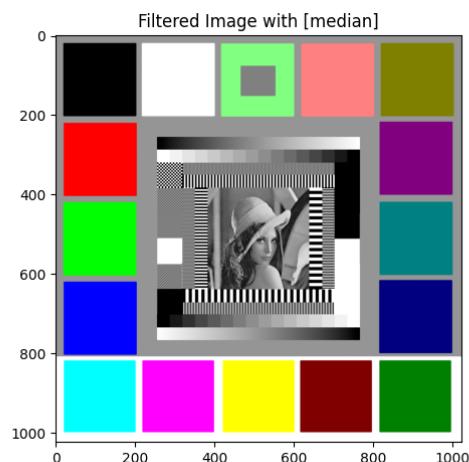
Tempo:

	Imagen de Teste	Imagen do CI
Média 1x11 & Media 11x1	103 segundos	37 segundos
Média 11x11	47 segundos	18 segundos

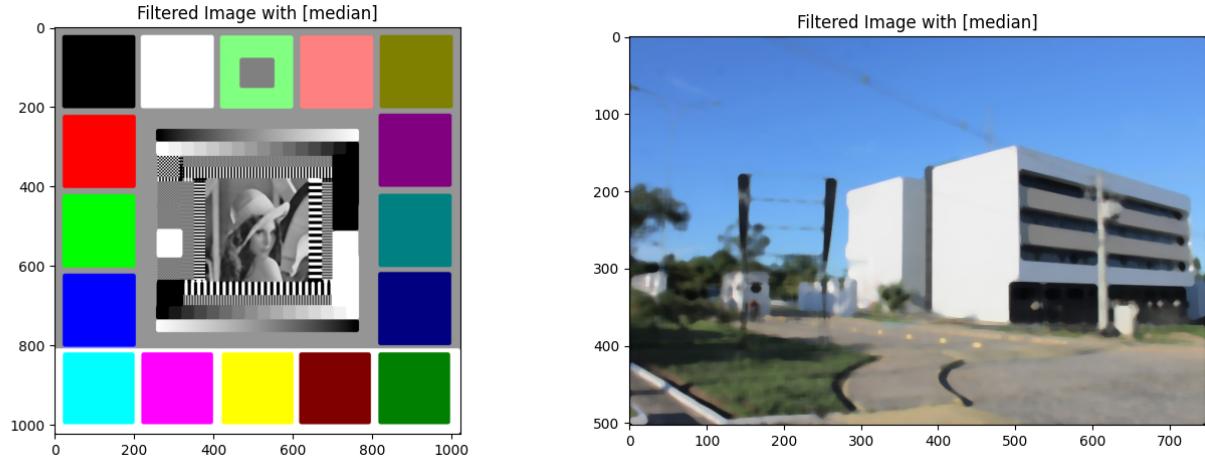
Obs: Usando o Measure-Command no terminal e desativando a apresentação do resultado

3.4 Mediana

3.4.1 Mediana 3x3



3.4.2 Mediana 9x9



4. Discussão e Conclusão

4.1 Conversão RGB-YIQ-RGB

Sobre a primeira questão, que se refere à conversão RGB-YIQ-RGB, verificamos que tudo foi realizado corretamente em sua primeira execução, utilizando a matriz de conversão. No entanto, notamos que, ao exibir o YIQ na tela, as cores pareciam avermelhadas, e entendemos que o componente vermelho, representado pelo Y na conversão, deveria ser entendido como escala de cinza. Além disso, percebemos que o componente I ficava mais forte em tons de vermelho e azul, enquanto o componente Q se destacava em tons de magenta e verde.

4.2 Negativo de RGB e Negativo de Y e conversão para RGB

Em relação à segunda questão, devido ao código desenvolvido na primeira questão, foi mais fácil construir a solução em partes. Para criar o negativo em RGB, utilizamos apenas a imagem original e subtraímos 255 de cada uma de suas bandas (R, G, B), o que gerou mudanças radicais, como o verde puro (0, 255, 0) se tornando magenta puro (255, 0, 255). Já para o negativo de Y, convertemos a imagem para YIQ, subtraímos 255 da banda Y e, em seguida, convertemos de volta para RGB para apresentação. O resultado mostrou que o brilho das cores estava invertido, o que nos levou a constatar que o Y é um índice de brilho de uma imagem.

4.3 Correlações

4.3.1 Introdução

Sobre a terceira questão, foi construído uma operação de correlação generalizada para comportar os diversos filtros que virão a ser usados, mas primeiro é preciso ser constatado que o programa aceita duas formas de obtenção de dados de uma máscara, sendo que a primeira permite construir todos os filtros menos o mediana e a segunda permite um certo set de variedades (que envolvem as funções média, mediana e entre outras no escopo do numpy) mas permite o uso de vários tamanhos de kernel e a escolha do pivô diretamente, resumidamente cada opção tem vantagens e desvantagens, a primeira é feita por um arquivo JSON que contém os valores do kernel, a posição do pivô, o booleano de extensão por zeros (ativada para todos os filtros, exceto sobel), a função limitadora de valores (se é cortado valores abaixo de 0 e acima de 255 ou se faz o módulo com expansão no histograma, como no sobel) e um offset, que é 0 para todos os filtros exceto o emboss que tem offset de 128, a segunda forma é usando colchetes contendo valores separados por vírgula, um exemplo de uso é [mean,3,3,1,1,true], o primeiro campo é a chave do filtro que pode ser box, mean, median, mode, erode, min, dilate, max, o segundo e terceiro campo é o tamanho do kernel em pixels, o quarto e quinto são a posição do pivô, e o ultimo é se tem extensão por zeros.

4.3.2 Filtro em JSON

Se o filtro for JSON, ele é encaminhado para uma classe chamada DataFilter, que extende a classe abstrata “AbstractFilter”, que contém um método chamado from_json que carrega as informações do filtro a partir de um arquivo JSON, neste método é obtido o caminho até o arquivo com o metodo path da pathlib e o conteúdo do arquivo JSON é carregado usando a função json.load() e a instância da classe tem como construtor o nome do arquivo JSON (sem a extensão) sendo usado como nome do filtro (para display), o kernel do filtro armazenado em um array NumPy, o pivô do filtro armazenado como uma tupla, um valor booleano que indica se a imagem deve ser estendida com zeros antes da aplicação do filtro, uma função de limite do filtro escolhida com base no valor de uma chave limit_function (podendo ser clip ou absolute ou normalize)

4.3.3 Filtro em conjunto de dados

Se o filtro for um conjunto de dados com colchetes e separados por vírgula a exemplo de [mean,3,3,1,1,true], ela entra na função get_function_filter, que analisa a string de entrada (que inclui o nome da função do filtro, o número de linhas e colunas do kernel, as coordenadas do pivô do kernel, e um valor booleano para indicar se a imagem deve ser estendida com zeros para aplicação do filtro) para determinar qual filtro deve ser aplicado a uma imagem

Na função de obter o filtro, se começa removendo os colchetes da string de entrada e dividindo-a em suas partes componentes usando o método split(). O nome da função é então verificado na tabela de filtros definida em FUNCTIONS_FILTER_TABLE (que será comentada no parágrafo a seguir). Em seguida, as variáveis rows, columns e pivot são convertidas em seus respectivos tipos de dados. A variável de extensão de zeros é convertida em um valor booleano de acordo com seu

valor em letras minúsculas (verdadeiro ou falso). Finalmente, a função retorna o filtro correspondente com base nos parâmetros fornecidos, que é obtido da tabela de filtros com base no nome da função.

Por fim ele é encaminhado para a classe FunctionFilter que extende a classe abstrata AbstractFilter e é construída pelos parâmetros: Uma string com o nome do filtro, o número de linhas e colunas do kernel, uma tupla com as coordenadas x e y do pivô, um valor booleano que indica se a imagem deve ser estendida com zeros antes da aplicação do filtro e a função que será usada para aplicar o filtro em cada janela deslizante

Além disso, nesta classe se define algumas funções parciais, como BOX_FILTER, MEDIAN_FILTER, o filtro média e mediana respectivamente, e a tabela de filtros, que obtém a primeira strings do conjunto de dados digitado pelo usuário, e de acordo com o que foi digitado, ele selecionará o filtro, a exemplo de: 'median': MEDIAN_FILTER, ou seja quando o usuário escrever [median,3,3,1,1,true] ele será encaminhado para a função de mediana

4.3.4 Tratamento de exceções e padding

Após ser obtido o filtro ou a sequência de filtros, é percorrido uma sequência de código de tratamento de exceções que checa se :

- Se todas as linhas da matriz do kernel têm o mesmo tamanho
- Se o kernel tem pelo menos uma linha e uma coluna
- Se cada "pixel" do kernel tem 1 ou 3 canais
- Se o pivô tem 2 dimensões e seus elementos são inteiros e se está dentro da máscara
- Se a variável de extensão por zeros é booleano
- Se a variável de offset é inteiro

Se a variável booleana de extensão de zeros for verdadeira, a imagem é expandida com zeros antes da aplicação do filtro. Essa expansão é feita usando a função np.pad() do pacote NumPy, que recebe como argumentos a imagem, o tamanho das bordas a serem adicionadas em cada dimensão e o valor de preenchimento das bordas (que no caso é 0). A área de aplicação do filtro é definida pela variável apply_area, que armazena os índices de linha e coluna a serem considerados na aplicação do filtro. Por outro lado, se a variável de extensão de zeros for falsa, a imagem é copiada e as bordas que o filtro não pode passar são removidas.

4.4 Correlações e Resultados e Comentários

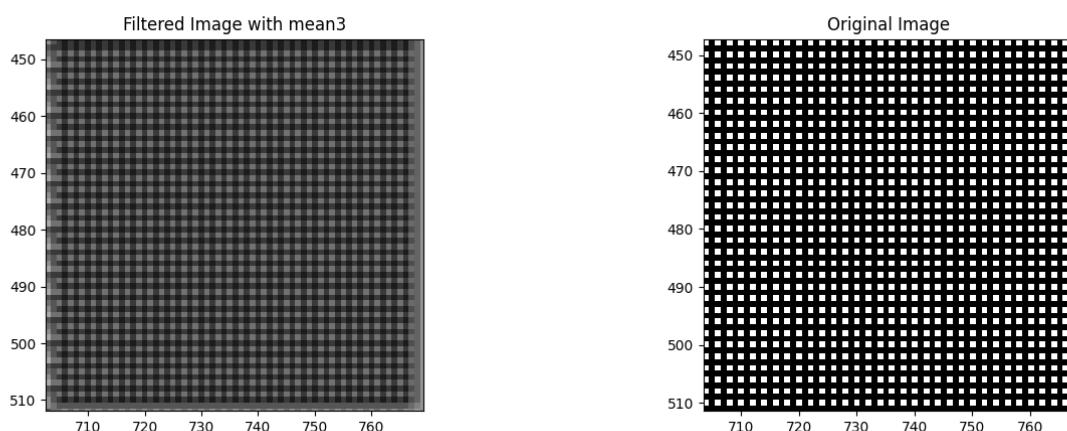
4.4.1 Filtro Soma

O filtro soma, resultou na imagem de teste uma imagem curiosamente apenas com valores máximos em alguma das bandas RGB, gerando cores sem saturação (exceto em poucos trechos), gerando cores ligadas às cores RGB e CMY

Na imagem do CI praticamente a imagem ficou branca com poucos detalhes mais escuros, em síntese, o filtro soma é um filtro de adição de brilho com a média, pois a soma do kernel é 9, e para manter o brilho a soma dos valores do kernel deve ser 1, isso também implica que se a soma estiver abaixo de 1 o filtro irá diminuir o brilho da imagem

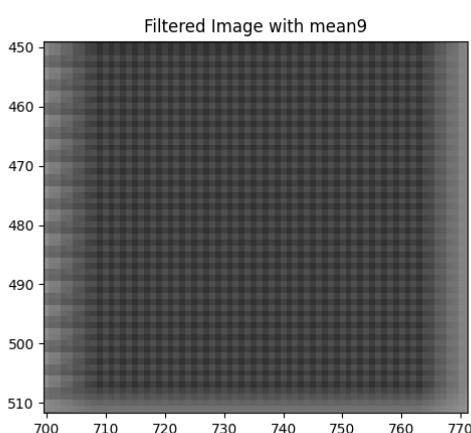
4.4.2 Filtro Média

O filtro média, resultou em um borramento com resultados interessantes em certas áreas da imagem de teste:



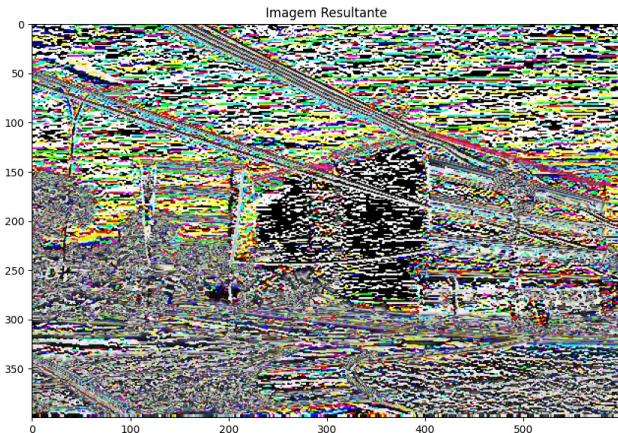
Constam-se uma nova imagem ruidosa com discrepâncias reduzidas e suavizadas, com tendência a se aproximar a um certo nível de cinza geral, tais ruídos permanecem mesmo aumentando o kernel da média para 9, ainda-se nota os antigos pontos brancos como pontos cinzas mais claros, e se

conclui que o filtro média ele promove alterações na vizinhança, podendo ser indesejáveis, além disso na imagem do ci nota-se ruídos “quadrados”, que divergem do borramento natural, ou seja próximo da realidade, uma opção para tal problema é o uso do filtro gaussiano.



4.4.3 Filtro Sobel

O filtro sobel resultou inicialmente em um resultado inesperado devido a falta de uma função limitadora de valores, pois quando o resultado da correlação ficava negativo o valor passava a começar de 255 para baixo, gerando ruídos absurdos como na imagem abaixo:



Isso foi solucionado primeiramente zerando valores negativos, gerando uma imagem mais correta e por fim usando a função de deixar o módulo dos valores, levando a imagem correta apresentada nas páginas anteriores

Observando a imagem de teste, constatou que o sobel tende a gerar resultados coloridos em imagens coloridas, devido às operações que alteram os valores RGB de cada pixel

4.4.4 Filtro Emboss

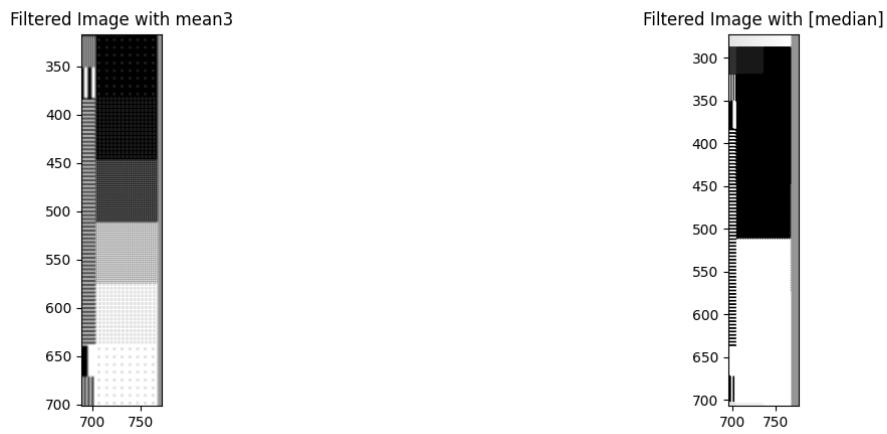
O filtro emboss resultou em uma imagem de relevo, com similaridade ao sobel, porém devido a seu offset as zonas sem bordas ficam cinzas, e as zonas de relevo podem ou ficar mais claras ou mais escuras

4.4.5 Filtro Média 1x11 * Média 11x1 versus Média 11x11

O tempo de execução saiu diferente do esperado e o uso de um único filtro média levou menos tempo para ser executado e concluído, possivelmente pelo fato ao usar dois filtros implica em percorrer pela imagem duas vezes, deixando em aberto se pode haver uma solução para o problema, possivelmente o uso de paralelismo possa permitir que o uso de dois filtros separados possa ser mais rápido que o uso de um filtro só

4.4.6 Mediana

O filtro mediana gerou resultados mais suaves que o filtro média, eliminando ruídos



Acima, a esquerda o resultado do filtro média e a esquerda o resultado do filtro mediana, concluindo que tal filtro elimina valores discrepantes de uma certa região, com o uso de uma mediana 3×3 todas as regiões quadradas acima perderam seus ruídos e assumiram completamente ou a cor branca ou a cor preta

Anexo - Manual de Instruções no Terminal

Questão 1 :

```
python src/main.py .\img\teste.png --yiq
```

Questão 2:

```
python src/main.py .\img\teste.png --neg-rgb
```

```
python src/main.py .\img\teste.png --neg-y
```

Questão 3:

```
python src/main.py .\img\ci.jpeg --filter .\filters\sum3.json
```

```
python src/main.py .\img\teste.png --filter .\filters\mean3.json
```

```
python src/main.py .\img\teste.png --filter .\filters\mean9.json
```

```
python src/main.py .\img\teste.png --filter .\filters\sobelv.json
```

```
python src/main.py .\img\teste.png --filter .\filters\sobelh.json
```

```
python src/main.py .\img\teste.png --filter .\filters\sobel.json
```

```
python src/main.py .\img\teste.png --filter .\filters\emboss.json
```

```
Measure-Command {python src/main.py .\img\teste.png --filter-sequence [mean,11,1,5,0,true] [mean,1,11,0,5,true]}
```

```
Measure-Command {python src/main.py .\img\teste.png --filter [mean,11,11,5,5,true]}
```

Questão 4:

```
python src/main.py .\img\ci.jpeg --filter [median,3,3,1,1,true]
```