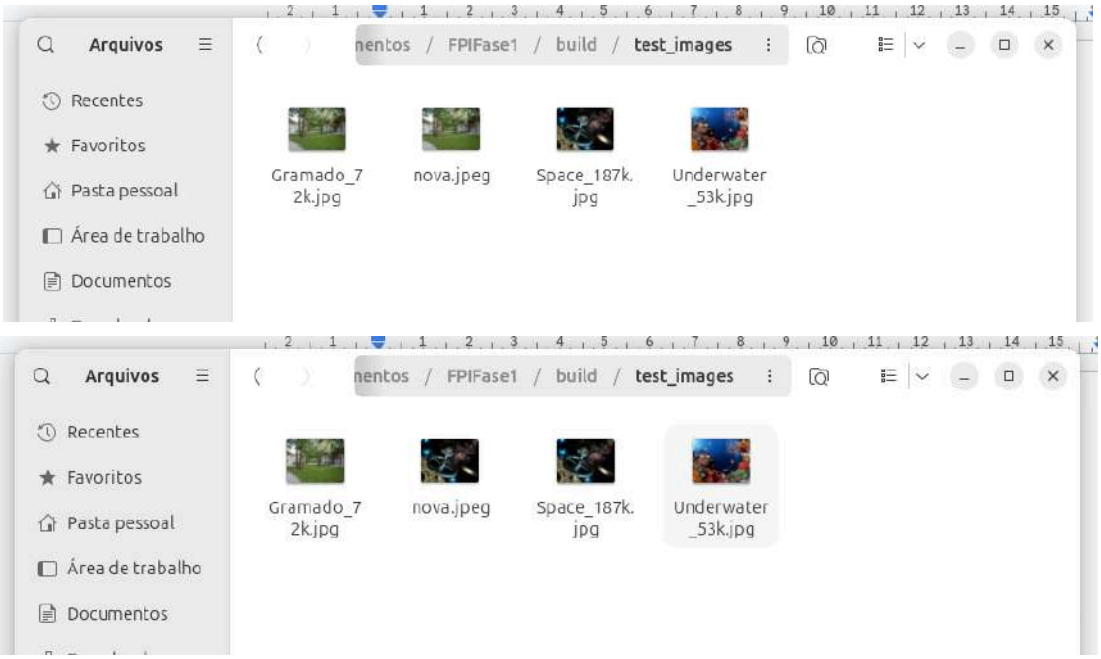
First Assignment Illustrated Report - Isadora Schwaab

Bibliotecas utilizadas
Leitura e gravação de imagens: OpenCV;
Gerenciamento de janelas (interface gráfica): Qt;

Parte I – Leitura e Gravação de Arquivos de Imagens

Na fase atual, os nomes dos arquivos de imagem de origem e destino foram definidos dentro do próprio código (mais tarde, serão tornados parâmetros). Não houveram maiores problemas para abrir ou salvar imagens. os testes foram feitos com ‘Gramado_72k.jpg’ e ‘Space_187k.jpg’, um após o outro, sem incluir ou deletar nenhum arquivo; ou seja, o programa não teve problemas para sobrescrever ‘nova.jpeg’:



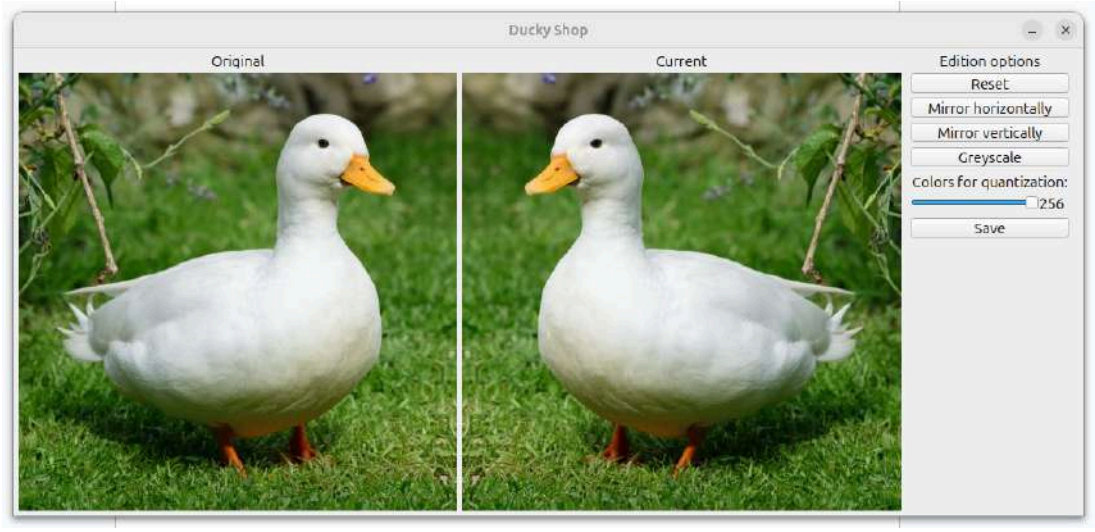
Novas imagens:

Não houveram diferenças visíveis entre as imagens original e destino, e as dimensões continuaram iguais. Para o teste com a imagem Gramado_72k.jpg, a imagem nova ficou com cerca de 51 kB. enquanto a imagem Space_187k.jpg ficou com aproximadamente 108 kB.
Após uma pesquisa breve, verifica-se que o formato JPEG tem um método de compressão propriamente moldado para reduzir o tamanho de um arquivo, ajustando informações para manter as partes mais significativas para a visão humana. Por exemplo, durante seu processo de quantização, algumas frequências são arredondadas para valores próximos, ocasionando a perda de informação sobre frequências mais altas e não tão significativas para o sistema visual humano. A função da biblioteca OpenCV utilizada possui parâmetros próprios que definem a compressão das imagens salvas; porém, como não foram passados no programa, podemos supor que as taxas de compressão 'default' da função são definidas para reduzir o tamanho da imagem sem reduzir drasticamente sua qualidade.

Parte II – Leitura, Exibição e Operações sobre Imagens

Para essa parte, as funções foram definidas antes da implementação da interface, sem grandes dificuldades além do processo de pesquisa. A combinação com a interface foi um pouco complexa em certos casos;

a) *Espelhamento horizontal e vertical da imagem original:*
Consegui usar a memcpy para inversão vertical, porém não horizontal. Isso se deve ao modo como a matriz é armazenada na estrutura cv::Mat da OpenCV. Nesse caso, recorri a algumas funções da própria classe da matriz para simplificar o processo de copiar uma coluna (provavelmente as funções não evitam a cópia pixel a pixel).
Abaixo, exemplos de uso na interface:



Após pressionar 'Mirror horizontally';

Após pressionar 'Mirror horizontally' e 'Mirror vertically';

b) *Conversão de imagem colorida para tons de cinza (luminância):*

Essa implementação também foi tranquila. O cálculo é feito pixel a pixel da matriz, acessando cada elemento da tupla BRG dela (cada célula de uma cv::Mat que guarda uma imagem é uma tupla BRG, em que BRG são do tipo *unsigned char*). Abaixo, exemplo de uso na interface:

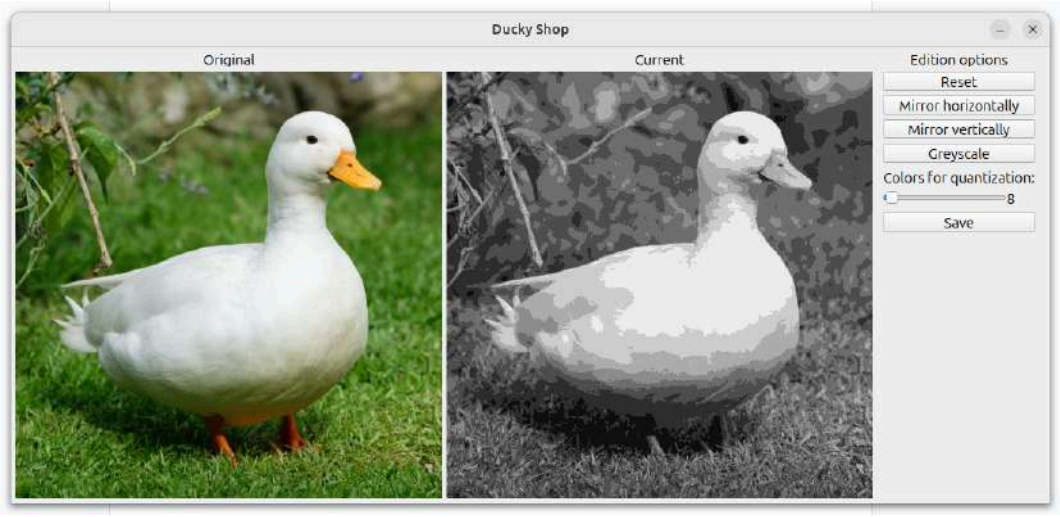
Após pressionar 'Greyscale';

Lembrando que aplicar o filtro Greyscale a uma imagem que já está em escala de cinza *não muda nada*, pois o cálculo para cada pixel é uma cálculo de porcentagem, e somar porcentagens que somam 1 (os pesos para R, G e B) de uma mesma quantidade (no caso, a luminância do pixel) resulta na própria quantidade original.

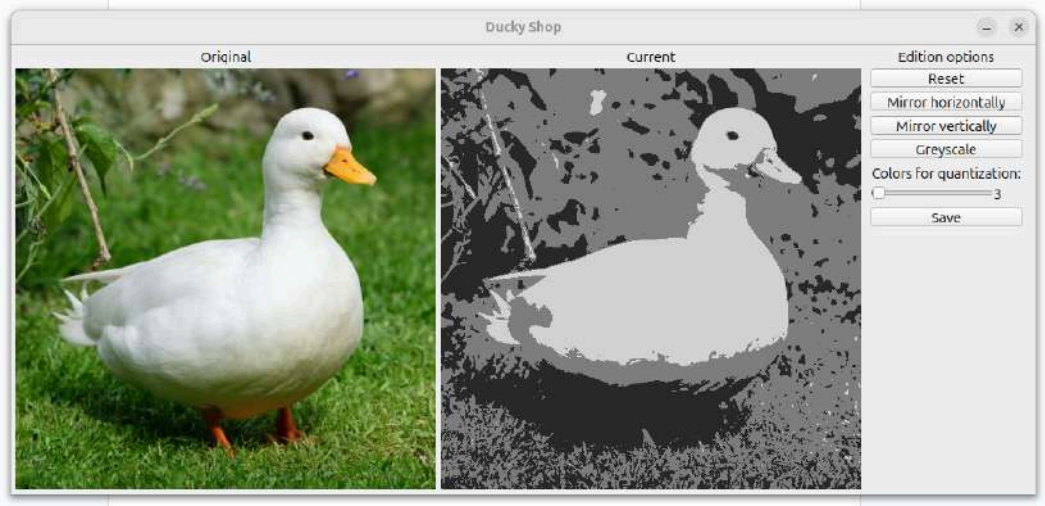
c) *Quantização (de tons) sobre as imagens em tons de cinza:*

A implementação do procedimento de quantização exigiu passar por todos os pixels 2 vezes: a primeira para definir os tons máximo e mínimo da imagem, e a segunda para modificar o número de tons, se necessário. Como a função só é útil de fato quando aplicada sobre imagens em tons de cinza, o ideal seria que a ativação do quantizador só estivesse disponível após pressionar o botão 'Greyscale'. Porém, preferi (em função do tempo que tinha para finalizar a aplicação) apenas definir, na

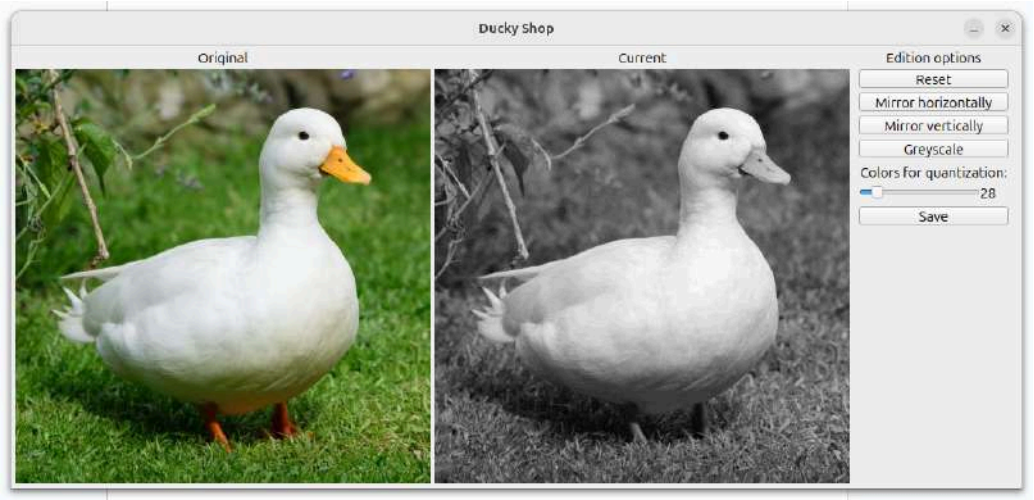
chamada da função de quantização através do slider, uma chamada extra à função Greyscale. Por mais que esse método garanta a robustez das chamadas, é redundante e não eficiente.
A aplicação na interface também exigiu a criação de um buffer para a imagem da edição atual (com todos os ‘mirrors’ aplicados) que nunca é quantizada. A main do programa já tinha variáveis para a imagem original e para a edição atual, mas se a edição atual receber uma imagem quantizada em X tons, será impossível utilizá-la para obter uma imagem quantizada com um número de tons maior que X. Por isso, as funções de ativação dos botões de espelhamento foram ajustadas para alterar tanto a edição atual quanto a imagem do buffer, enquanto a função de quantização ajusta a imagem do buffer e altera/exibe a edição corrente. Isso tambḿe dobra o custo dos processos de espelhamento.
Vale lembrar que, por mais que alguns custos sejam dobrados, eles não afetam o desempenho da aplicação: a aplicação das funções e exibição das imagens é rápida o suficiente para não ser perceptível ao usuário.
Abaixo, exemplo de uso na interface:



Após soltar o slider em 8;

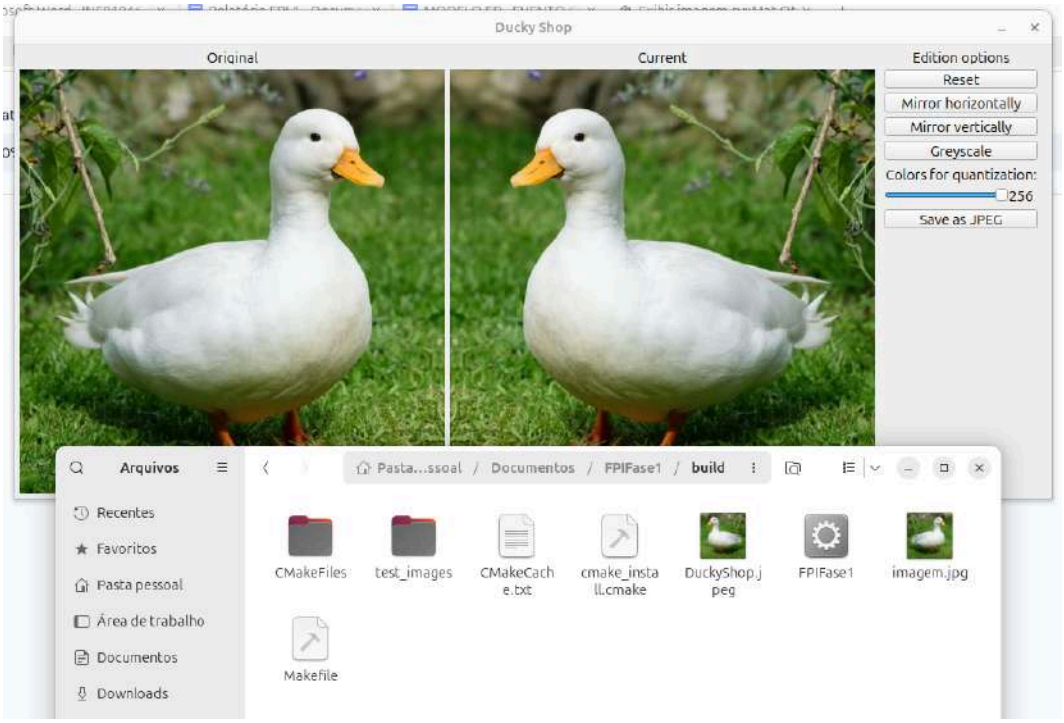


Após soltar o slider em 3;

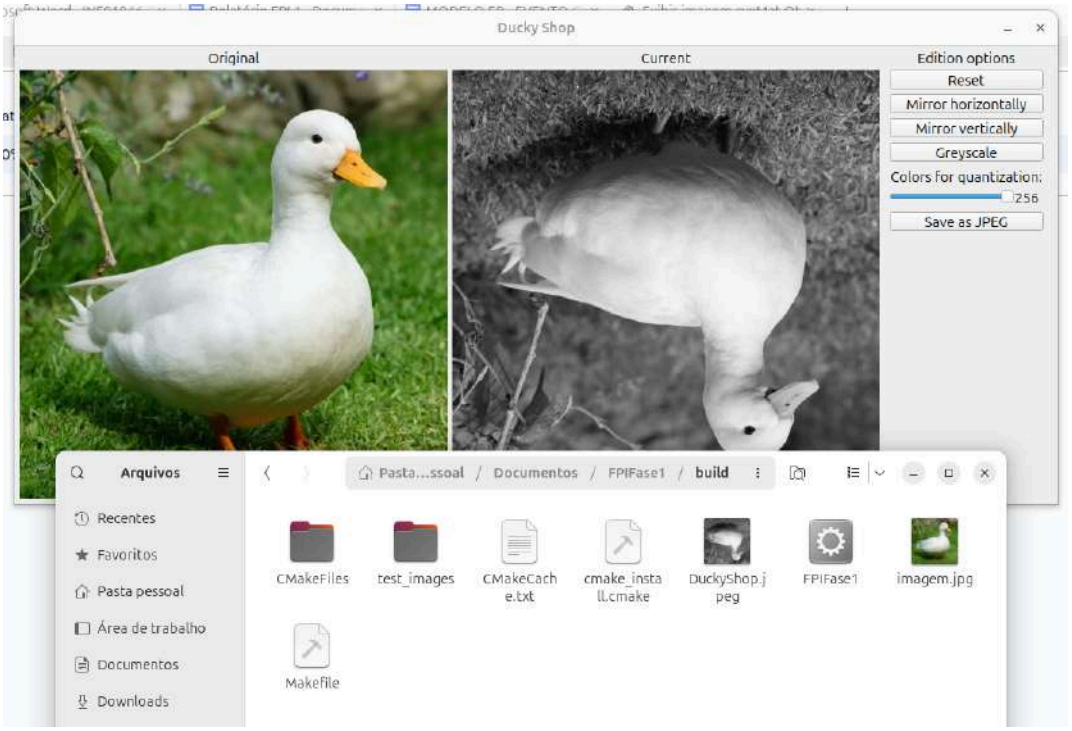


Após soltar o slider em 28 (mesmo depois das alterações anteriores);

d) *<u>Salvamento da imagem resultante das operações realizadas em um arquivo JPEG:</u>*
A função de salvamento baixa a edição atual (a que está sendo exibida na tela) com uma simples aplicação da cv::imwrite (função do OpenCV), salvando com o nome padrão “DuckyShop.jpeg”. Desse modo, cada ativação da função de salvamento **sobrescreve** o arquivo salvo atualmente.
Exemplos:



Salvamento da imagem espelhada;



O salvamento das novas alterações sobrescreve o antigo.

Outros detalhes importantes

- A ativação do programa se dá, pela linha de comando, chamando o caminho do executável e o caminho da imagem para edição.
- A aplicação não tem a função ‘Drag & Drop’;
- O salvamento da imagem é feito na pasta atual do terminal.