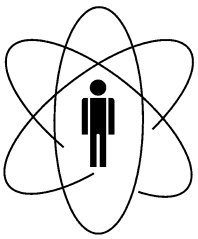
ISSN 0102-745X



CBPF - CENTRO BRASILEIRO DE PESQUISAS FÍSICAS

Rio de Janeiro

Notas Técnicas

CBPF-NT-005/18

maio 2018

Introdução à biblioteca de processamento de imagens OpenCV

Cleiton Silvano Goulart, André Persechino, Marcelo Portes de Albuquerque e Márcio Portes de Albuquerque

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES

dx.doi.org/10.7437/NT2236-7640/2018.02.005
Notas Técnicas, v. 8, n. 2, p. 1–29, 2018

Introdução à biblioteca de processamento de imagens OpenCV

Introduction to the image processing library OpenCV

Cleiton Silvano Goulart,* André Persechino,† Marcelo Portes de Albuquerque,‡ e Márcio Portes de Albuquerque§

Centro Brasileiro de Pesquisas Físicas - Coordenação de Desenvolvimento Tecnológico

Submetido: 05/09/2017 Aceito: 20/04/2018

Resumo: Apresentamos nestas Notas uma introdução aos elementos fundamentais da biblioteca de processamento de imagens OpenCV, versão 3.2. São cobertos tópicos diversos, como instalação e compilação do OpenCV em sistemas operacionais Windows e Linux, aritmética de imagens e aplicação de técnicas de filtragem linear. Todas as discussões são ilustradas com exemplos de códigos escritos em C++. O material aqui contido pode ser utilizado por iniciantes, desde que estes possuam um conhecimento mínimo de programação. Requer-se também um primeiro contato com os conceitos básicos de processamento de imagens; entretanto, leitores sem esse conhecimento podem fazer uso deste material para iniciarem estudos na área.

Palavras chave: OpenCV; processamento de imagens; C++.

Abstract: We introduce in this Report an introduction to the fundamental elements of digital image processing library OpenCV, version 3.2. It covers topics such as installation and compilation on Windows and Linux architectures, image arithmetics and linear filtering techniques. All subjects are developed with C++ examples support, requiring minimum knowledge on programming. It is expected from reader some background on image processing theory, but even those without prior knowledge can benefit from this work, using it to start investigations in the image processing field.

Keywords: OpenCV; image processing; C++.

1. INTRODUÇÃO

É difícil estimar o papel das imagens digitais em nosso mundo contemporâneo. Aplicações médicas, militares, industriais e de mídia dependem massivamente de imagens e, portanto, de seu entendimento e manipulações plenos. Embora relativamente jovem, o formalismo básico da análise de imagens evoluiu de maneira extremamente intensa com o advento dos computadores[1]. Hoje tratamos quase que exclusivamente de processamento *digital* de imagens. Nesse sentido, os profissionais envolvidos direta ou indiretamente com análise de imagens devem possuir um conhecimento mínimo das técnicas computacionais usadas, sem as quais correm o risco de lidar com dados e informações não confiáveis.

Com a intenção de contribuir para o desenvolvimento da área de processamento digital de imagens, os autores apresentam nestas Notas uma introdução elementar à biblioteca OpenCV (*Open Source Computer Vision*) [13], largamente utilizada na indústria e academia. Desde o ano de 1999, uma divisão de pesquisadores da Intel liderada por Gary Bradski, começou a desenvolver o que hoje é o OpenCV, contando atualmente com mais de 2500 funções e algoritmos promovem uma infraestrutura básica para análise de imagens e vídeos

*Electronic address: cleitonsg@cbpf.br
†Electronic address: aamerico@cbpf.br
‡Electronic address: marcelo@cbpf.br
§Electronic address: mpa@cbpf.br

[1] Nos primórdios do processamento de imagens, as manipulações eram feitas através de circuitos eletrônicos – lineares ou não. Decorre disso que boa parte da teoria dos sistemas lineares pode ser aplicada com sucesso à análise de sinais e imagens.

[11]. O OpenCV possui módulos específicos para [13]:

- processamento de vídeos;
- processamento de imagens;
- utilização de recursos de processador gráfico (GPU);
- processamento paralelo (*clusters*),

dentre vários outros. Alguns estão implementados com algoritmos clássicos e outros já estão implementados no estado da arte.

No ano de 2010 a NVIDIA passou a contribuir com o projeto de forma que processadores gráficos e de dispositivos móveis começaram a ser suportados. O OpenCV é desenvolvido em código-aberto em C/C++, e é multi-plataforma, sendo possível utilizá-lo em Microsoft Windows, macOS e sistemas baseados em Linux/Unix. Ademais, OpenCV possui suporte para programação em outras linguagens tais como: Phyton, Ruby, MATLAB®, dentre outras. Até a data de escrita deste artigo, o OpenCV se encontra na versão 3.2.

Buscamos – com o compromisso de manter o volume de texto razoável – introduzir os conceitos mais fundamentais da ferramenta, uma vez que esta guarda particularidades diversas, tornando-a não-trivial sob diversos aspectos[2]. Não se pretende que este trabalho seja uma referência completa sobre OpenCV, devido principalmente à sua magnitude. De fato, estas Notas podem ser tomadas como uma espécie de "alfabetização", pois fornecem os elementos básicos da biblioteca, bem como alguns de seus métodos mais usuais. A extensão para aplicações complexas fica a cargo da imaginação do leitor. O leitor interessado em se aprofundar em OpenCV pode recorrer à documentação oficial do projeto [13] e ao livro de Kaelher e Bradski [10]. Para uma introdução completa à linguagem C++, recomenda-se o livro de Savitch [16]. Por sua vez, uma introdução ao processamento digital de imagens é apresentada no trabalho de Persechino e Albuquerque [15]. Para aprofundamento, sugere-se o clássico de Gonzalez e Woods [7].

O trabalho está organizado da seguinte maneira: na Seção 1.1 são mostrados em detalhes os procedimentos para compilação do OpenCV em ambientes Windows e Linux, respectivamente. A Seção 2 compreende o principal conteúdo teórico deste trabalho, em que são apresentados os conceitos mais fundamentais do OpenCV, tais como a classe `cv::Mat` e alguns de seus métodos e construtores relacionados (Seção 2.1), além de procedimentos para entrada e exibição de imagens (Seção 2.1.1) e aritmética de imagens (Seção 2.1.4). A Seção 3 discute transformações sobre imagens, focando em transformações de intensidade (Seção 3.1) e análise de histograma (Seção 3.2). Por fim, transformações espaciais lineares são abordadas na Seção 3.3.

[2] Isso é ainda mais evidente quando pretende-se migrar certo processo desenvolvido em uma linguagem interpretada, tal como MATLAB®, para um código compilado em C++, por exemplo.

1.1. Infraestrutura básica para desenvolvimento

O ambiente mínimo de desenvolvimento em OpenCV para ilustrar os exemplos que serão abordados neste artigo requer um compilador e os códigos-fonte da biblioteca do OpenCV. Além destes requisitos, é recomendado – embora não essencial – o uso de uma *IDE*[3].

1.1.1. O compilador

O compilador é um programa responsável por converter todo o código-fonte para linguagem de máquina [1].Para este artigo, foi escolhida a linguagem C/C++ para a elaboração e demonstração dos exemplos. Para cada sistema operacional tem-se opções de compiladores de C/C++, sendo este assunto tratado à frente neste artigo, nas seções individuais de cada sistema operacional.

1.1.2. Os códigos-fonte do OpenCV

O OpenCV é constituído por um conjunto de arquivos com a implementação das rotinas e algoritmos. Estes arquivos podem ser obtidos a partir do site oficial do projeto [14]. Os códigos-fonte são independentes do sistema operacional em uso. Em função da extensão da biblioteca e dos numerosos arquivos que compõem o OpenCV, durante o processo de instalação será necessário o uso da ferramenta CMake[2], disponível para os sistemas operacionais já citados.

1.1.3. A Interface de Desenvolvimento Integrado - IDE

Embora não seja obrigatório o uso de uma interface de desenvolvimento integrado - IDE, aplicativos deste tipo auxiliam no gerenciamento dos arquivos do projeto. Diretivas e instruções essenciais para a correta compilação e vinculação com a biblioteca do OpenCV também podem ser gerenciados pelas IDEs. Uma IDE sugerida para o desenvolvimento é o Eclipse, disponível em seu site oficial [3]. Esta IDE possui código aberto, é multi-plataforma, e permite realizar a depuração em tempo real dos exemplos que serão tratados neste artigo. Sua instalação está amplamente descrita na documentação do projeto OpenCV [14].

1.2. Instalação no Windows

Para se ter a infra-estrutura de desenvolvimento de programas em C/C++ no sistema operacional Microsoft Windows, versão 10, deve ser instalado primeiramente um compilador. Qualquer computador que suporte o Windows 10 será capaz

[3] IDE - *Integrated Development Environment* ou Ambiente de desenvolvimento integrado.

de compilar e executar os exemplos contidos neste artigo. A seguir será descrito cada uma das etapas para instalação de compilador nesse.

1.2.1. Instalação do compilador

Como o Windows não possui um compilador de C/C++ nativo, devemos instalar um. Dentre os vários compiladores disponíveis, foi escolhido para estas Notas o MinGW, sendo este um compilador de código aberto. O MinGW está disponível no site oficial do projeto [12].

Nenhuma alteração será necessária no processo de instalação do MinGW. Caso o usuário deseje alterar o caminho padrão do MinGW este deve se atentar para **não** colocar nenhum espaço no caminho de instalação, pois alguns programas podem apresentar um mau funcionamento ou alguma incompatibilidade. Após instalado, será aberto o gerenciador de pacotes. Os seguintes pacotes devem ser instalados para que seja possível compilar e depurar os programas:

- mingw32-base;
- mingw32-gcc-g++;
- msys-base;
- mingw-developer-tools;

Após a marcação para instalação, aplica-se as alterações através do menu `Installation / Apply Changes`. Este processo irá efetuar o *download* dos arquivos necessários. Após a instalação, é necessário realizar algumas configurações:

- Acesse a tela contida em `Painel de Controle / Sistema / Configurações avançadas do sistema`, na aba `Avançado` clique no botão `Variáveis de Ambiente`;
- Edite ou crie a variável `PATH` na seção `Variáveis de usuário`, sendo que ela deverá conter o diretório `c:\MinGW\bin`. Caso necessário separe com `;` os diretórios existentes com o novo diretório;

Após estes passos, o MinGW deverá estar devidamente configurado e pronto para uso.

1.2.2. Preparação para compilação do OpenCV no Windows

Após obter os códigos-fonte do site oficial do OpenCV [14], é recomendado que os arquivos sejam descompactados para uma pasta específica dentro do diretório do MinGW: `c:\mingw\opencv\`.

Deve-se abrir o CMake para que se possa configurar corretamente o código-fonte. Caso o CMake não esteja instalado, ele pode ser obtido gratuitamente a partir do site oficial [2]. O processo de instalação do CMake é fácil e intuitivo, de forma que não é necessária nenhuma orientação adicional. Com o CMake aberto, devem ser tomadas as seguintes ações na ordem informada:

- No campo `Where is the source code` deverá ser informado o diretório onde estão os códigos-fonte. Observe que durante a extração dos arquivos foi criada uma nova pasta, cujo nome está vinculado à versão do OpenCV. Conforme os diretórios já citados temos: `c:\mingw\opencv\opencv-3.2.0`;
- No campo `Where to build the binaries` deverá ser informado o local onde os arquivos compilados da biblioteca serõ armazenados. Recomenda-se que este diretório não apresente nenhum espaço e esteja dentro da pasta do MinGW. Uma sugestão de entrada para este campo é: `c:\mingw\opencv\build-3.2.0`;
- As caixas de seleção `Grouped` e `Advanced` devem ser marcadas;
- Deve-se clicar no botão `Configure`. Ao clicar neste botão pela primeira vez, ele irá abrir uma tela solicitando algumas informações:
  - Na lista `Specify the generator for this project` certifique-se de escolher **Eclipse CDT4 - MinGW Makefiles**;
  - Certifique-se de deixar selecionado a opção `Use default native compilers`;
- O CMake irá realizar algumas verificações nos códigos-fonte do OpenCV. Esta etapa poderá demorar alguns instantes;
- Ao concluir a configuração inicial, irá aparecer uma lista com várias opções destacadas em vermelho. Estas são algumas opções de compilação necessárias para o OpenCV;
- Na lista de itens de configurações, abra a sub-lista `BUILD` e certifique-se de que o item `BUILD_opencv_ts` esteja **desmarcado**. Este módulo do OpenCV apresenta alguns conflitos de dependências no Windows e é necessário apenas para desenvolvedores do OpenCV;
- Deve-se clicar no botão `Configure` novamente, de forma que a lista que antes estava destacado em vermelho não esteja mais;
- Clique no botão `Generate` para gerar os arquivos necessários para a compilação do OpenCV;
- Após os arquivos serem gerados, o CMake poderá ser fechado;

1.2.3. A compilação do OpenCV

O processo de compilação do código-fonte do OpenCV no Windows deve ser feito a partir do terminal de comandos com privilégios de administrador. O terminal deverá ser aberto e alterado o diretório atual para o diretório que foi escolhido no CMake para armazenar os arquivos compilados - `c:\mingw\opencv\build-3.2.0`. O comando `mingw32-make` deverá ser executado; iniciará a compilação do código do OpenCV. Caso não seja gerado nenhum erro durante este processo, deverá ser executado o comando `mingw32-make install`. Este comando finaliza o processo de compilação do código fonte do OpenCV. Na ocorrência de erros durante este processo, todas as etapas já informadas deverão ser revisadas e a documentação disponível no site do projeto [14] deverá ser consultada.

1.3. Instalação no Linux

A instalação do OpenCV em ambiente Linux é facilitada pela utilização da ferramenta CMake [2]. Os requisitos mínimos para a instalação são [14]:

- Compilador GCC [5] (mín. versão 4.4);
- CMake [2] (mín. versão 2.8);
- GTK [8]; e
- FFmpeg [4].

O compilador GCC - *GNU Compiler Collection* - é uma iniciativa livre, que proporciona compiladores para C, C++, Fortran, Ada e Go. GTK é um *toolkit* para interfaceamento gráfico e é escrito em C, mas permite desenvolvimento em diversas linguagens [8]. Por fim, FFmpeg corresponde a uma ferramenta poderosíssima para edição e *streaming* de áudio e vídeo.

De posse dos requisitos mínimos, deve-se baixar o código-fonte do OpenCV no site oficial [14]. Os passos seguintes são realizados com o CMake e devem ser executados como super-usuário – *root* – no terminal.

No diretório em que o OpenCV for descompactado, deve-se criar um diretório para a compilação. Ao entrar neste diretório (que por simplicidade chamaremos apenas de `build/`), o usuário deve executar o CMake, indicando o arquivo `CMakeLists.txt` correto para configuração. Este arquivo se encontra no primeiro diretório acima de `build/`, se este foi criado no local correto. Em princípio, a sequência a seguir realiza as tarefas descritas.

```
mkdir build
cd build
cmake ..
```

Contudo, há uma série de parâmetros opcionais fornecidos ao CMake que determinam se módulos adicionais devem ser inseridos, se o OpenCV deve usar seu próprio FFmpeg ao invés de um nativo ao sistema, dentre muitas outras opções. Sugere-se ao usuário que verifique no arquivo `CMakeLists.txt` quais parâmetros lhe interessam. Deve-se notar que a alteração ou inserção de tais parâmetros pode ser realizada por edição do documento em questão ou passadas na linha de comando. Por fim, é interessante que a opção `BUILD_EXAMPLES` seja ativada, para que dezenas de aplicações didáticas sejam disponibilizadas em diretório próprio ao fim da instalação.

Após a configuração realizada, o usuário inicia o processo de construção das aplicações por meio do comando `make`. Finalizada esta etapa, o processo é seguido da instrução `make install`.

1.3.1. Compilação de programas no Linux via CMake.

Todos os códigos desenvolvidos em C++ precisam ser compilados para que seja gerado um executável e, aí sim, o usuário possa fazer uso da ferramenta. Como os códigos expostos neste trabalho fazem uso da biblioteca OpenCV, externa ao C++, vínculos[4] devem ser criados. Isso é feito facilmente por meio de um arquivo `CMakeLists.txt` com instruções ao CMake. Um exemplo simples é mostrado no Código 1.

```
1 # versao minima requerida
2 cmake_minimum_required(VERSION 2.8)
3 # nome do projeto
4 project(ProjetoOpenCV)
5 #localiza o OpenCV
6 find_package(OpenCV REQUIRED)
7 # declara o executavel construido com base no codigo
      fonte
8 add_executable(ProjetoOpenCV codigo_projeto_opencv.cpp)
9 # faz o link com o OpenCV
10 target_link_libraries(ProjetoOpenCV ${OpenCV_LIBS})
```

Código 1: Diretrizes para compilação de um programa fictício `ProjetoOpenCV` via CMake.

O arquivo `CMakeLists.txt` deve estar no mesmo diretório que o código (em nosso exemplo, `ProjetoOpenCV.cpp`). No terminal o usuário deve entrar com a seguinte sequência para que seja gerado o executável:

```
cmake .
make
```

Com isso, é gerado o executável e a aplicação pode ser testada.

2. CONCEITOS E OBJETOS BÁSICOS

2.1. Estrutura de uma imagem digital e a classe `cv::Mat`

Grosso modo, imagens digitais podem ser compreendidas como matrizes – eventualmente multidimensionais – que as-

[4] O termo mais usual é *link*.

sociam a cada entrada (i, j) um valor ou um *vetor* de intensidades proporcionais à intensidade luminosa[5] da estrutura retratada naquele ponto. Imagens monocromáticas – ou em tons de cinza – são ditas escalares, pois associam à entrada (i, j) um escalar I_{ij}. Por sua vez, imagens coloridas necessitam de uma maior quantidade de informação para serem descritas corretamente: a cada entrada (i, j) é atribuído um vetor $(I_1, I_2, \ldots, I_N)_{ij}$, cujas componentes correspondem à intensidade luminosa em seus respectivos canais. Exemplos corriqueiros são as imagens que seguem o padrão RGB: nesta configuração, cada entrada está associada a um vetor (I_B, I_G, I_R), que contém as intensidades nos canais azul, verde e vermelho[6], respectivamente. Discussões mais aprofundadas sobre espaços de cor são apresentadas no livro de Gomes e Velho [6].

Em OpenCV, dispomos da classe `cv::Mat`, seus métodos e atributos para manusear imagens. A classe `Mat` é composta fundamentalmente por duas partes [13]: cabeçalho e entradas. O cabeçalho guarda informações sobre as dimensões da imagem e sobre sua forma de armazenamento, enquanto as entradas são simplesmente apontadas por um determinado ponteiro. O detalhe fundamental quanto ao uso da classe `Mat` é que, embora cada imagem tenha seu próprio cabeçalho, as entradas podem ser compartilhadas [10, 13].

2.1.1. *Entrada e exibição de imagens*

Um primeiro exemplo instrutivo em OpenCV é apresentado no Código 2 . Nele, são mostrados os procedimentos básicos para abrir e exibir uma imagem simples. Vejamos agora o que cada instrução neste código significa:

Nas linhas 1 e 3 é incluído todo o conjunto de funções do OpenCV por meio das diretivas `#include<opencv2/opencv.hpp>` e `using namespace cv`. Embora este procedimento influa no tempo de compilação, será adotado em todos os exemplos destas Notas por tornar os desenvolvimentos mais simples. Uma alternativa a este procedimento seria inserir individualmente os arquivos de interesse da biblioteca [16].

```
1 #include <opencv2/opencv.hpp>
2
3 using namespace cv;
4
5 int main( int argc, char** argv ) {
6   Mat img = imread(argv[1], -1);
7
8   namedWindow("Imagem 1", WINDOW_NORMAL);
9
10  imshow("Imagem 1", img);
11
12  waitKey(0);
13
14  destroyWindow("Imagem 1");
15
16  return 0;
17 }
```

Código 2: Abrindo e exibindo uma imagem simples.

Na linha 6 a variável `img` do tipo `cv::Mat` é criada, recebendo a imagem indicada pelo parâmetro `argv[1]`. A imagem em questão é retornada por meio da função `cv::imread`, que requer dois argumentos de entrada: uma string relativa ao arquivo a ser carregado e uma *flag* que determina o espaço de cores sobre o qual a imagem será representada. As opções para espaços de cores são mostradas na Tabela I.

Flag	**Equivalente numérica**	**Espaço de cores**
`IMREAD_UNCHANGED`	-1	espaço original
`IMREAD_GRAYSCALE`	0	tons de cinza
`IMREAD_COLOR`	1	RGB

Tabela I: Representações possíveis ao carregar uma imagem.

Uma variação deste primeiro exemplo, em que o usuário é requisitado a informar qual o espaço de cores deve ser usado é mostrado no Código 3. Prosseguindo com a análise do Código 2, na linha 8 é criada uma janela identificada que receberá a imagem selecionada. A criação da janela é concretizada por meio da função `cv::namedWindow`, que demanda dois argumentos de entrada: uma string de identificação e uma flag, que definirá o controle de redimensionamento da janela. Algumas flags possíveis são mostradas na Tabela II.

Flag	**Comportamento da janela**
`WINDOW_NORMAL`	redimensionamento livre
`WINDOW_AUTOSIZE`	redimensionamento proibido
`WINDOW_OPENGL`	exibição com suporte OpenGL
`WINDOW_FULLSCREEN`	exibição em tela cheia

Tabela II: Possíveis modos de exibição e redimensionamento de janelas identificadas.

Na linha 12 é chamada a função `waitKey`, cujo argumento é um inteiro `t`. Se $\texttt{t} \leq 0$, um evento no teclado é aguardado indefinidamente. Caso positivo, um evento é aguardado por, pelo menos, `t` mili-segundos[7]. Por fim, na linha 14 é evocada a função `destroyWindow`, que tem por finalidade destruir a janela identificada pelo seu argumento.

[5] No caso de imagens capturadas no regime visível da luz. Contudo, deve-se ter em mente que outras grandezas completamente diferentes da luz podem ser representadas por meio de imagens, tais como resistividades, índices de refração, tempos de relaxação etc.

[6] Note que esta ordenação dos canais não é a usual (vermelho, verde e azul), mas sim uma inversão desta última.

[7] O tempo pode variar em função de procedimentos internos do sistema operacional.

```
1 #include <opencv2/opencv.hpp>
2 #include <iostream>
3
4 using namespace cv;
5 using namespace std;
6
7 void    SelecionaEspacoCor(int &m);
8
9 int main( int argc, char** argv ) {
10   int flagEspacoCor;
11
12   SelecionaEspacoCor(flagEspacoCor);
13
14   Mat img = imread(argv[1], flagEspacoCor);
15
16   namedWindow("Imagem 1", WINDOW_NORMAL);
17
18   imshow("Imagem 1", img);
19
20   waitKey(0);
21
22   destroyWindow("Imagem 1");
23
24   return 0;
25 }
26
27 void SelecionaEspacoCor(int &m){
28   int n;
29
30   cout << endl;
31   cout << " Informe espaco de cor desejado " << endl;
32   cout << "\t-1 - Espaço de cor original" << endl;
33   cout << "\t 0 - Espaço de cor monocromático" << endl;
34   cout << "\t 1 - Espaço de cor RGBA" << endl;
35
36   cin >> n;
37
38   m = n;
39 }
```

Código 3: Abrindo e exibindo uma imagem, solicitando ao usuário o espaço de cores desejado.

2.1.2. *Criação explícita de imagens*

Ao invés de carregar e exibir uma imagem, pode ser que desejemos criar uma. Nesse sentido, poderíamos declarar uma certa variável `img` sem, no entanto, atribuí-la uma imagem através da função `imread`. Como exemplo, suponha que se deseje criar três imagens de dimensões $512 \times 512 \times 3$, em que cada uma receberá um fundo vermelho, verde e azul, respectivamente. Nesse caso, as variáveis devem ser criadas através do construtor `Mat()`. A alocação das matrizes se dá pela função-membro `create()`. O Código 4 ilustra este procedimento.

Pode ser observado nas linhas 12, 15 e 18 do Código 4 a utilização das funções `setTo` e `Scalar`. A primeira delas serve para atribuir aos elementos da matriz em questão os valores especificados [10, 13], já a segunda corresponde na verdade a uma classe que implementa vetores quadridimensionais[8]. Uma aplicação usual desta classe consiste justamente em utilizá-la para a passagem de valores de pixels [13].

Para finalizar a discussão sobre criação explícita de imagens, devemos nos atentar ao fato de que ao criarmos a matriz, devemos informar o *tipo* desta. O tipo da matriz definirá se suas entradas são inteiros (com ou sem sinal) ou reais e quantos bits ocupam. A sintaxe básica para definição do tipo de dados é da forma

$$\texttt{CV\_}kl\texttt{\_C}m,$$

em que k é um inteiro, representando a profundidade em bits [15] das entradas da matriz (8, 16, 32 ou 64 bits); l é um caractere (U, S ou F) que define se o tipo será inteiro sem e com sinal ou real, respectivamente. Por fim, m é um inteiro e refere-se ao número de canais (no máximo 3, embora seja possível expandir [10]).

```
1 #include <opencv2/opencv.hpp>
2 #include <iostream>
3
4 using namespace cv;
5 using namespace std;
6
7 int main( int argc, char** argv ) {
8   int dimImagem[] = {512, 512};
9   Mat R, G, B;  // (R)ed, (G)reen e (B)lue
10
11  R.create(dimImagem[0],dimImagem[1],CV_8UC3);
12  R.setTo(Scalar(0,0,255));
13
14  G.create(dimImagem[0],dimImagem[1],CV_8UC3);
15  G.setTo(Scalar(0,255,0));
16
17  B.create(dimImagem[0],dimImagem[1],CV_8UC3);
18  B.setTo(Scalar(255,0,0));
19
20  namedWindow("Imagem em vermelho",WINDOW_AUTOSIZE);
21  namedWindow("Imagem em verde",WINDOW_AUTOSIZE);
22  namedWindow("Imagem em azul",WINDOW_AUTOSIZE);
23
24  imshow("Imagem em vermelho",R);
25  imshow("Imagem em verde",G);
26  imshow("Imagem em azul",B);
27
28  waitKey(0);
29
30  return 0;
```

[8] Deve-se notar, no entanto, que uma ou mais componentes podem ser negligenciadas, tal como mostrado no Código 4.

```
31 }
```

Código 4: Criando imagens explicitamente.

2.1.3. Saída de imagens

Frequentemente há a necessidade de salvar uma imagem processada, ao invés de simplesmente exibí-la. A instrução para salvar um arquivo de imagem em OpenCV é dada pelo comando `cv::imwrite`, cuja sintaxe

```
imwrite(stringArquivo, img, params)
```

salva a imagem `img` no arquivo nomeado em `stringArquivo`, segundo os parâmetros informados em `params`. Estes parâmetros dependem do formato escolhido (jpg, png etc.) e dizem respeito ao nível de compressão dos dados, definição de níveis usados em imagens binárias etc. Todas as opções são devidamente descritas na documentação oficial do projeto [13].

2.1.4. Operações aritméticas básicas

Operações aritméticas (soma, subtração, multiplicação e divisão) correspondem às transformações mais básicas em processamento de imagens. Tais operações são pontuais, isto é, operam entrada-a-entrada. Por exemplo, ao somarmos duas imagens monocromáticas quaisquer, x e y, teríamos então que a imagem resultante, z, seria da forma $z_{ij} = x_{ij} + y_{ij}$.

Naturalmente, as operações mais simples são as de soma e subtração entre matrizes. Para realizar uma operação desse tipo, faz-se uso dos operadores + e −. Portanto, a soma entre duas ou mais imagens segue a mesma sintaxe que a soma de dois escalares. Outra opção para somar imagens seria usar a função `cv::add(x, y, z)`, que realiza a mesma operação, atribuindo a `z` o resultado `x + y`. Uma soma mais geral seria dada por `z = ax + by`, em que `a` e `b` são pesos que ponderam a soma[9]. A realização de tal operação pode ser realizada por meio da função `cv::addWeighted(x, a, y, b, c, z)`, que realiza a soma ponderada de `x`, `y` e o nível constante, ou DC[10], `c`.

2.1.5. Saturação

Algumas observações sobre operações aritméticas devem ser feitas: a primeira é que deve-se sempre ter em mente que a aritmética de inteiros é diferente da aritmética de reais. Nesse sentido, se nos deparássemos com uma divisão inteira de 7 por 4, por exemplo, o resultado seria igual a 1, ao passo que em aritmética de reais o resultado seria 1,75. Portanto, deve-se sempre atentar para que os escalares e matrizes operados sejam do mesmo tipo de forma que sejam evitados erros decorrentes de diferentes aritméticas. A segunda observação diz respeito à possibilidade de os resultados de um certa operação aritmética ultrapassarem os limites inferior ou superior do intervalo típico da variável em questão (*under* e *overflow*). Por exemplo, ao operarmos duas matrizes inteiras de 8 bits (sem sinal) de profundidade em bits (U8), o resultado pode ser maior que 255 ou menor que 0. Para tratar esse tipo de ocorrência, OpenCV conta com uma aritmética de saturação, que garante a permanência dos valores dentro da faixa dinâmica adequada. A seguir é dado um exemplo deste tipo de ocorrência.

Suponha que, dadas as seguintes imagens quantizadas em 8 bits sem sinal,

$$x_1 = \begin{bmatrix} 53 & 84 & 216 & 17 \\ 192 & 169 & 174 & 143 \\ 0 & 160 & 223 & 168 \end{bmatrix}$$

e

$$x_2 = \begin{bmatrix} 185 & 59 & 225 & 237 \\ 50 & 58 & 166 & 54 \\ 138 & 55 & 78 & 79 \end{bmatrix},$$

pretenda-se calcular $y = x_1 + x_2$. Obviamente, o resultado exato é

$$y = \begin{bmatrix} 238 & 143 & 441 & 254 \\ 242 & 227 & 340 & 197 \\ 138 & 215 & 301 & 247 \end{bmatrix}.$$

Contudo, em um ambiente com aritmética de saturação, os valores da terceira coluna estariam fora do intervalo $[0, 255]$, caracterizando um caso de overflow. A aritmética de saturação do OpenCV proporcionaria o seguinte resultado:

$$y = \begin{bmatrix} 238 & 143 & 255 & 254 \\ 242 & 227 & 255 & 197 \\ 138 & 215 & 255 & 247 \end{bmatrix}.$$

O Código 6 implementa este exemplo. Há de se destacar nesta implementação a sobrecarga do operador de exibição `std::cout`: variáveis de diversos tipos em OpenCV podem ser exibidas por este comando, facilitando depuração de código. Destaque-se também a definição das entradas das matrizes como variáveis do tipo `char`. Isto foi feito pois este tipo de dado é do tipo inteiro 8 bits [16], exatamente o desejado para o exemplo.

Composição de imagens por combinação linear

O Código 7 implementa um programa que carrega um número N arbitrário de imagens com profundidade de 8

[9] Note que o OpenCV tolera operações entre escalares e matrizes.

[10] Terminologia esta inspirada na teorias das séries de Fourier e eletrônica analógica, em que o termo DC é, por definição, o único termo não-oscilante na composição do sinal. Outro termo bastante empregado é *offset*.

bits sem sinal, informadas na chamada da aplicação, e em seguida solicita ao usuário N coeficientes (reais) para a composição da imagem final, dada por

$$z_{ijk} = z_{\mathrm{DC}k} + \sum_{l=0}^{N-1} \alpha_l x_{ijk}^{(l)}, \qquad (1)$$

em que α_l corresponde ao l-ésimo coeficiente, $x_{ijk}^{(l)}$ à coordenada (i, j, k) da l-ésima imagem de entrada e $z_{\mathrm{DC}k}$ ao nível DC desejado no k-ésimo canal. Há algumas particularidades sobre este código que devem ser destacadas:

1. Tendo as imagens de entrada uma profundidade de 8 bits e sendo os coeficientes reais, é necessária uma conversão entre tipos para tornar consistente a operação dada pela Eq. (1);
2. Como o número de imagens não é conhecido a priori, o programa faz uso de alocação dinâmica de memória, como pode ser visto na linha 21;
3. A contribuição do nível DC é realizada por meio de uso de uma variável do tipo `Scalar`.

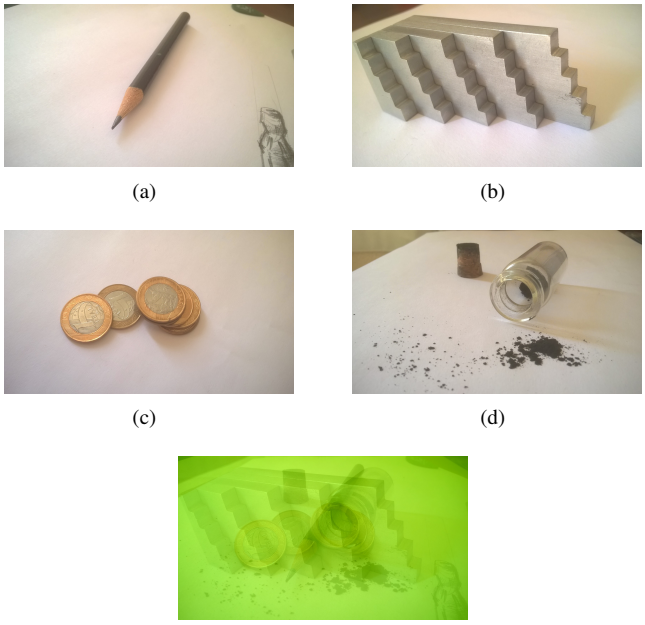
```
1 #include <opencv2/opencv.hpp>
2 #include <iostream>
3
4 int main(int argc, const char * argv[]) {
5
6     cv::Mat imgA, imgB, imgFinal01, imgFinal02,
       imgFinal03, imgfinal04;
7
8     imgA = cv::imread(argv[1], cv::IMREAD_COLOR);
9     imgB = cv::imread(argv[2], cv::IMREAD_COLOR);
10
11    std::string JanSomaDireta = "Soma com operador +";
12    imgFinal01 = imgA + imgB;
13    cv::namedWindow(JanSomaDireta);
14    cv::imshow(JanSomaDireta, imgFinal01);
15
16    std::string JanSomaSaturada = "Soma com comando add
       ";
17    cv::add(imgA, imgB, imgFinal02);
18    cv::namedWindow(JanSomaSaturada);
19    cv::imshow(JanSomaSaturada, imgFinal02);
20
21    std::string JanSomaPonderada = "Soma ponderada de
       imagens";
22    double alpha = 0.3;
23    double beta = 0.7;
24    double gamma = 0;
25    cv::addWeighted(imgA, alpha, imgB, beta, gamma,
       imgFinal03);
26    cv::namedWindow(JanSomaPonderada);
27    cv::imshow(JanSomaPonderada, imgFinal03);
28
29    std::string JanSomaEscalar = "Soma escalar de
       imagens";
30    double escala = 0.3;
31    cv::scaleAdd(imgA, escala, imgB, imgFinal04);
32    cv::namedWindow(JanSomaEscalar);
33    cv::imshow(JanSomaEscalar, imgFinal04);
34
35    cv::waitKey(0);
36    cv::destroyAllWindows();
37
38    return 0;
39 }
```

Código 5: Implementação de somas de imagens segundo quatro método distintos.

A conversão entre tipos a que a primeira observação se refere pode ser visualizada nas linhas 39 e 54. Nestas linhas é invocado o método `cv::x.convertTo(z, tipo, α, β)`, que atribui a `z` as entradas de `x` convertidas segundo a escolha `tipo` e submetem estas entradas à transformação de escala pelos fatores α e β, sendo este último um termo DC. Este procedimento explica a existência da variável `fatorEscala`, na linha 13: sua função é levar as imagens da escala típica de U8 para o intervalo real [0,1]. Por fim, deve-se notar que este método segue a aritmética de saturação discutida anteriormente.

Na linha 27 aparece pela primeira vez um método da classe `cv::Size`. Nesta linha, é feito o acesso das dimensões da imagem `x`, sendo estes membros – largura (`width`) e altura (`height`) – usados logo em seguida para definição explícita da imagem `z`. Note que a declaração explícita de `z` difere um pouco das formas usadas até agora. A Figura 1 ilustra a composição de imagens por meio da Eq. (1), viabilizada pelo Código 7.

```
1 # include <opencv2/opencv.hpp>
2 # include <iostream>
3
4 using namespace std;
5 using namespace cv;
6
7 int main(int argc, char** argv){
8
9   int M = 3;     // num. linhas
10  int N = 4;     // num. colunas
11
12  // as entradas das imagens sao declaradas abaixo.
      Note que estas
13  // poderiam ter sido declaradas como arrays 1-D
14
15  char entrImg1[M][N] = {
16    {53, 84, 216, 17},
17    {192, 169, 174, 143},
18    {0, 160, 223, 168}
19  };
20
21  char entrImg2[M][N] = {
22    {185, 59, 225, 237},
23    {50, 58, 166, 54},
24    {138, 55, 78, 79},
25  };
```



(a) (b) (c) (d) (e)

Figura 1: Operações aritméticas sobre imagens. De (a) a (d) são mostradas as imagens a serem operadas segundo a Equação (1). Em (e) é apresentado o resultado da média aritmética das imagens (isto é, $\alpha_i = 0.25\ \forall\ i$) com níveis DC nos canais R, G e B dados por -0.50, 0.10 e -0.20, respectivamente.

```
26
27   Mat img1 = Mat(M, N, CV_8UC1, entrImg1);
28   Mat img2 = Mat(M, N, CV_8UC1, entrImg2);
29   Mat img3 = Mat(M, N, CV_8UC1);
30
31   img3 = img1 + img2;
32
33   cout << "img1:\n" << img1 << "\n\n";
34
35   cout << "img2:\n" << img2 << "\n\n";
36
37   cout << "img3:\n" << img3 << endl;
38
39   namedWindow("Imagem 1", WINDOW_AUTOSIZE);
40   namedWindow("Imagem 2", WINDOW_AUTOSIZE);
41   namedWindow("Imagem resultante", WINDOW_AUTOSIZE);
42
43   imshow("Imagem 1", img1);
44   imshow("Imagem 2", img2);
45   imshow("Imagem resultante", img3);
46
47   waitKey(0);
48
49   destroyAllWindows();
50
51   return 0;
52 }
```

Código 6: Implementação de um código para visualização dos efeitos da aritmética de saturação em OpenCV.

```
1 # include <opencv2/opencv.hpp>
2 # include <iostream>
3
4 using namespace std;
5 using namespace cv;
6
7 int main(int argc, char** argv){
8
9   int N = argc - 1;                    // numero de
      imagens informadas
10
11  float* coefs;                        // matriz dos
      coeficientes
12  float  nivDC[3];                     // matriz dos
      niveis DC em cada canal
13  float  fatorEscala = 1.0 / 255.0;    // fator de
      normalizacao de escala
14
15  char charCanal[3] = {'R', 'G', 'B'};
16
17  Size dimsImg; // dimensoes das imagens de entrada
18
19  Mat x, z;     // matrizes a receberem as imagens
20
21  coefs = new float[N];
22
23  // o passo seguinte visa obter o tamanho das imagens
      para declaracao explicita de z
24
25  x = imread(argv[1], -1);
26
27  dimsImg = x.size();
28
29  z = Mat(dimsImg.height, dimsImg.width,CV_32FC3);
30
31  // loop para leitura e armazenamento dos coeficientes
      , exceto DC
32  for (int i = 1; i <= N; i++){
33    cout << "Informe o " << i <<"-esimo coeficiente:\t"
      ;
34    cin  >> coefs[i];
35
36    x = imread(argv[i],-1);
37
38    // conversao de U8 para F32
39    x.convertTo(x,CV_32FC3,fatorEscala,0.0);
40
41    z = z + (coefs[i] * x);
42  }
43
44  // loop para leitura dos niveis DC nos canais R, G e
      B
45  for (int i = 0; i <= 3; i++){
46    cout << "Informe o nivel DC no canal " << charCanal
      [i] <<":\t";
47    cin  >> nivDC[i];
48  }
49
```

```
50  // superposicao dos niveis DC informados
51  z = z + Scalar(nivDC[2], nivDC[1], nivDC[0]);
52
53  // conversao de F32 para U8 com readequacao de escala
54  z.convertTo(z, CV_8UC3,255,0);
55
56  namedWindow("Resultado Final", WINDOW_NORMAL);
57  imshow("Resultado Final", z);
58
59  // salva a imagem no arquivo especificado
60  imwrite("OperacoesAritmeticas.jpg",z);
61
62  waitKey(0);
63
64  destroyAllWindows();
65
66  delete coefs;
67
68  return 0;
69 }
```

Código 7: Implementação de um programa para composição de imagens segundo somas ponderadas.

2.1.6. *Multiplicação e divisão*

Para finalizar a discussão das operações aritméticas básicas de objetos da classe `cv::Mat`, apresentaremos brevemente as operações de multiplicação e divisão. Primeiramente, deve-se ter em mente que estas operações diferem das usuais entre escalares. De fato, os objetos da classe `Mat` obedecem a uma aritmética matricial, não sendo necessariamente igual à aritmética escalar. Talvez o caso mais emblemático da diferença entre estas aritméticas seja a operação de multiplicação. Se, por exemplo, uma linha de código contém

```
z = x * y;
```

em que `x` e `y` são variáveis do tipo `Mat`, o resultado seria dado pela expressão seguinte:

$$z_{ij} = \sum_{n=1}^{N} x_{in} y_{nj}, \tag{2}$$

em que N corresponde ao número de colunas de x e ao número de linhas de y. Como exemplo, o Código 8 implementa uma multiplicação matricial entre

$$x = \begin{bmatrix} 3 & 3 & 2 \\ 2 & 5 & 6 \\ 5 & 5 & 4 \\ 4 & 4 & 9 \end{bmatrix} \quad \text{e} \quad y = \begin{bmatrix} 0 & 4 & 7 & 6 \\ 4 & 2 & 2 & 1 \\ 2 & 1 & 1 & 6 \end{bmatrix}.$$

O resultado exibido na tela deve ser

```
Imagem z:
[16, 20, 29, 33;
32, 24, 30, 53;
28, 34, 49, 59;
34, 33, 45, 82],
```

que pode ser checado rapidamente abrindo-se o produto à mão, conforme a Equação (2).

```
1 # include <opencv2/opencv.hpp>
2 # include <iostream>
3
4 using namespace std;
5 using namespace cv;
6
7 int main(int argc, char** argv){
8   int M = 4;
9   int N = 3;
10
11  float entrImgX[M][N] = {
12    {3.0, 3.0, 2.0},
13    {2.0, 5.0, 6.0},
14    {5.0, 5.0, 4.0},
15    {4.0, 4.0, 9.0}
16  };
17
18  float entrImgY[N][M] = {
19    {0.0, 4.0, 7.0, 6.0},
20    {4.0, 2.0, 2.0, 1.0},
21    {2.0, 1.0, 1.0, 6.0}
22  };
23
24  Mat x = Mat(M, N, CV_32FC1, entrImgX);
25  Mat y = Mat(N, M, CV_32FC1, entrImgY);
26  Mat z;
27
28  z = x * y;
29
30  cout << "Imagem x:\n" << x << endl;
31  cout << "Imagem y:\n" << y << endl;
32  cout << "Imagem z:\n" << z << endl;
33
34  return 0;
35 }
```

Código 8: Implementação de um programa para realização de produto matricial entre duas matrizes pré-determinadas

Além do produto matricial, objetos da classe `Mat` também podem ser multiplicados (e divididos) entrada a entrada. Ou seja, dadas `x`, `y` e `z`, teríamos que o produto entrada a entrada entre as duas primeiras seria da forma

$$z_{ij} = x_{ij} \cdot y_{ij}, \tag{3}$$

desde que as matrizes tenham as mesmas dimensões. Este produto é mais largamente utilizado em processamento de imagens do que o produto matricial, pois permite a

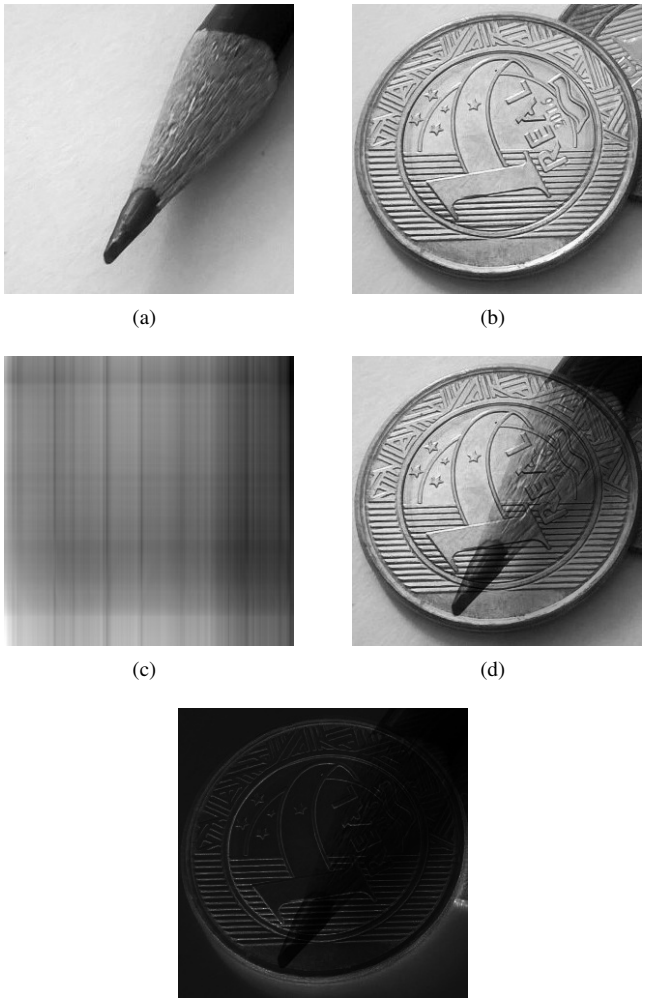
composição de imagens. A multiplicação entrada a entrada entre duas imagens é realizada por meio do método `cv::Mat::mul`, cuja chamada

```
z = x.mul(y)
```

atribui a `z` o produto entrada a entrada das matrizes `x` e `y`. Por sua vez, a divisão entrada e entrada entre `x` e `y` é implementada pelo operador `/`, sobrecarregado para este fim:

```
z = x / y.
```

A Figura 2 ilustra estas operações em imagens.



(a) (b) (c) (d) (e)

Figura 2: Multiplicação e divisão entre imagens. Em (a) e (b) são mostradas duas imagens a serem multiplicadas. Em (c) e (d) são mostrados os resultados dos produtos matricial e entrada a entrada, evidenciando a diferença significativa entre estas duas operações. Em (e) é mostrado o resultado da divisão entrada a entrada de (a) por (b).

Deve-se ter sempre em mente que as operações de multiplicação (matricial ou entrada a entrada) ou divisão podem gerar resultados fora dos intervalos de trabalho, sendo estes alterados segundo a aritmética de saturação do OpenCV. O Código 9 implementa um programa que realiza o produto matricial de duas imagens, informadas na chamada da aplicação. Naturalmente, as imagens devem ser tais que o número de colunas da primeira (`x`) seja igual ao número de colunas da segunda, `y`, caso contrário, um erro será gerado, encerrando a aplicação.

```
1  # include <opencv2/opencv.hpp>
2  # include <iostream>
3
4  using namespace std;
5  using namespace cv;
6
7  int main(int argc, char** argv){
8    double fatorEscala, nivDC, fatorEsc8U;
9    double minX, maxX, minY, maxY, minZ, maxZ;
10
11   Mat x, y, z;
12   Size dimsX, dimsY;
13
14   x = imread(argv[1],-1);
15   y = imread(argv[2],-1);
16
17   // fator de escala padrao para normalizacao de
       imagens 8U
18   fatorEsc8U = 1.0 / 255.0;
19
20   // conversao das imagens x e y de 8U para 64F
21   x.convertTo(x,CV_64FC1,fatorEsc8U,0.0);
22   y.convertTo(y,CV_64FC1,fatorEsc8U,0.0);
23
24   z = x * y;
25
26   dimsX = x.size();
27   dimsY = y.size();
28
29   // determinacao dos niveis minimo e maximo de
       intensidade
30   minMaxLoc(x, &minX, &maxX);
31   minMaxLoc(y, &minY, &maxY);
32   minMaxLoc(z, &minZ, &maxZ);
33
34   // ajuste linear de escala
35   fatorEscala    = 1.0 / (maxZ - minZ);
36   nivDC          =  - minZ * fatorEscala;
37
38   z = fatorEscala * z + nivDC;
39
40   minMaxLoc(z, &minZ, &maxZ);
41
42   cout << "\n\nNiveis maximos e minimos de x, y e z,
       respectivamente\n\n";
43
44   cout << minX << "\t" << maxX << endl;
45   cout << minY << "\t" << maxY << endl;
46   cout << minZ << "\t" << maxZ;
47
48   cout << "\n\nDimensoes de x, y e z, respectivamente \
       n\n";
49
50   cout << dimsX << endl;
51   cout << dimsY << endl;
```

```
52  cout << z.size() << endl;
53
54  namedWindow("Imagem x", WINDOW_AUTOSIZE);
55  namedWindow("Imagem y", WINDOW_AUTOSIZE);
56  namedWindow("Imagem z", WINDOW_AUTOSIZE);
57
58  imshow("Imagem x",x);
59  imshow("Imagem y",y);
60  imshow("Imagem z",z);
61
62  waitKey(0);
63
64  destroyAllWindows();
65
66  return 0;
67 }
```

Código 9: Implementação de um programa para realização de produto matricial entre duas imagens carregadas na chamada da aplicação

Há de interessante no Código 9 o fato de que, prevendo que o produto matricial entre as imagens possa gerar valores fora dos intervalos usuais, é realizado um **ajuste linear de contraste** que leva os valores de intensidade de `z` para o intervalo real $[0,1]$. Tal operação pode ser vista nas linhas 35 a 38. O ajuste linear de contraste é simplesmente uma transformação de escala dada por [15]

$$y = \frac{v_{\text{MAX}} - v_{\text{MIN}}}{u_{\text{MAX}} - u_{\text{MIN}}}(x - u_{\text{MIN}}) + v_{\text{MIN}}, \tag{4}$$

em que u_{MIN} e u_{MAX} correspondem aos extremos da escala original e v_{MIN} e v_{MAX} aos da nova escala. A determinação dos valores mínimo e máximo de uma matriz é realizada via utilização da função `cv::minMaxLoc`, cuja sintaxe é da forma

```
minMaxLoc(x,&xMin,&xMax).
```

Note-se que os valores `xMin` e `xMax` são passados por referência à referida função.

3. MANIPULAÇÕES E TRANSFORMAÇÕES BÁSICAS

Uma vez elucidados os aspectos básicos de entrada e aritmética básica de objetos da classe[11] `Mat`, podemos prosseguir a estudar um pouco das manipulações e transformações básicas disponibilizadas pelo OpenCV.

São várias as possibilidades de transformação em imagens, sendo as mais comuns:

- Transformações dos níveis de intensidade;
- Transformações geométricas;
- Transformações espaciais ou de coordenadas.

Nesta Seção discutiremos transformações de níveis de intensidade. As transformações espaciais serão tratadas em seção própria. Transformações geométricas não serão cobertas neste trabalho.

3.1. Transformações de níveis de intensidade

Como discutido na Seção 2.1, uma imagem digital pode ser compreendida como uma matriz multidimensional que associa a cada uma de suas entradas um certo nível de intensidade. Transformações de níveis de intensidade agem diretamente sobre estes valores, não se preocupando com um eventual relacionalmento inter-pixels [15]. Em outras palavras, dada uma transformação T qualquer – linear ou não –temos

$$y_{ij} = T[x_{ij}], \tag{5}$$

em que x e y são as imagens de entrada e saída, respectivamente. Um exemplo já abordado na Seção 2.1.6 é o ajuste linear de contraste, dado pela Equação (4).Resultados extremamente diversos podem ser obtidos por transformações de intensidade.

3.1.1. *Inversão de níveis*

Como primeiro exemplo, consideremos a transformação

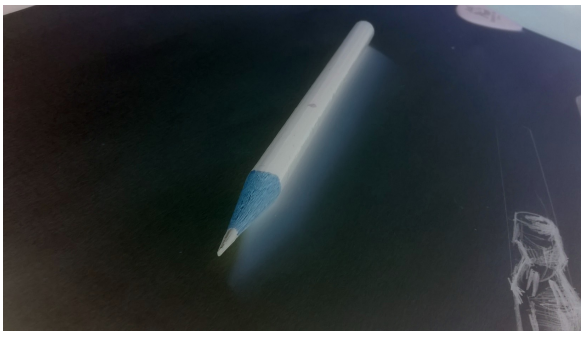
$$y_{ijk} = x_{\text{MAX}k} - x_{ijk}, \tag{6}$$

em que $x_{\text{MAX}k}$ corresponde ao nível máximo de intensidade do k-ésimo canal. A transformação dada pela Equação (6) implementa a inversão dos valores de intensidade, ou "tira o negativo" da imagem, como pode ser visto na Figura 3. O Código 10 implementa esta transformação.

```
1 # include <opencv2/opencv.hpp>
2 # include <iostream>
3
4 using namespace std;
5 using namespace cv;
6
7 int main(int argc, char** argv){
8   Mat x, y;
9   Mat canaisX[3];       // matriz de matrizes para usar
        na decomposicao de x
10
11  double fatorEscala;
12  double nivMax[3];     // matriz a receber os niveis
      maximos em cada canal
13
14  fatorEscala = 1.0 / 255.0;
15
```

[11] Evitamos usar termos como "imagens da classe `Mat`", uma vez que esta classe é muito mais abrangente, de forma que matrizes completamente arbitrárias podem ser descritas por seus objetos, e não apenas imagens.

(a)



(b)

Figura 3: Inversão de níveis de intensidade. São mostradas em (a) e (b) as imagens original e transformada, respectivamente. Nota-se claramente o efeito de "negativo" na imagem resultante.

```
16  x = imread(argv[1], -1);
17  x.convertTo(x,CV_64FC3,fatorEscala,0.0);
18
19  y = Mat(x.size(), CV_64FC3);
20
21  // decomposicao de x em seus canais
22  split(x,canaisX);
23
24  // determinacao dos maximos
25  minMaxLoc(canaisX[0],NULL, &nivMax[0]); // B
26  minMaxLoc(canaisX[1],NULL, &nivMax[1]); // G
27  minMaxLoc(canaisX[2],NULL, &nivMax[2]); // R
28
29  // inversao de niveis de intensidade
30  y = Scalar(nivMax[0], nivMax[1], nivMax[2]) - x;
31
32  // exibicao
33  namedWindow("Imagem original", WINDOW_AUTOSIZE);
34  namedWindow("Imagem transformada", WINDOW_AUTOSIZE);
35  namedWindow("Canal R",WINDOW_AUTOSIZE);
36  namedWindow("Canal G",WINDOW_AUTOSIZE);
37  namedWindow("Canal B",WINDOW_AUTOSIZE);
38
39  imshow("Imagem original", x);
40  imshow("Canal R", canaisX[2]);
41  imshow("Canal G", canaisX[1]);
42  imshow("Canal B", canaisX[0]);
43  imshow("Imagem transformada", y);
44
45  waitKey(0);
46
47  destroyAllWindows();
48
49  return 0;
50 }
```

Código 10: Implementação de uma transformação de intensidade para inversão de níveis de uma imagem RGB informada na chamada da aplicação.

Deve-se perceber que neste exemplo, a inversão de níveis ocorreu em cada canal da imagem de entrada. Isto significa que os valores máximos de cada canal individual teve de ser levantado, demandando a separação dos canais através do uso da função `cv::split`, cuja sintaxe

```
split(x, *canaisRGB)
```

faz com que seja atribuída a cada entrada do array `canRGB` os níveis de intensidade dos canais de `x`. Isto significa, na prática, que a variável `canRGB` é uma matriz de matrizes (no nosso exemplo, de 3 matrizes). Isso é mostrado na linha 9 do código, na declaração da variável `canaisX`.

3.1.2. Modulação senoidal de intensidade

Para ilustrar a flexibilidade proporcionada por transformações de níveis de intensidade, consideremos uma transformação menos usual. Suponha, por exemplo, que a imagem de entrada, x, seja submetida à seguinte operação:

$$T[x_{ij}] = A\,\mathrm{sen}\left(2\pi\frac{\nu}{N-1}j\right)x_{ij}, \tag{7}$$

em que A é um fator de ganho, ν é uma frequência dada em px^{-1} e N[12] é o número de colunas da imagem. A transformação mostrada na Equação (7) gera um sinal modulado senoidalmente na direção horizontal. O Código 11 implementa esta transformação, carregando uma imagem arbitrária informada na chamada da aplicação. A imagem de entrada é convertida em seu carregamento para uma versão monocromática e aí sim a modulação é feita. A Figura 4 ilustra o uso desta transformação.

```
1 # include <opencv2/opencv.hpp>
2 # include <iostream>
3 # include <cmath>
4
5 using namespace cv;
6 using namespace std;
7
8 # define pi 3.14159265358979312
9
```

[12] O fator $N-1$ na Equação (7) – ao invés de simplesmente N – se deve ao fato de que em C++ a indexação de arrays começa em zero.

(a)

(b)

(c)

Figura 4: Modulação senoidal em imagens. Em (a) é mostrada a imagem de entrada. Em (b) e (c) são mostradas modulações com fatores de ganho e frequências dadas por $A = 1.00$, $A = 0.25$, $\nu = 5.0$ e $\nu = 12.0$, respectivamente. Deve-se notar a diferença entre os espaços de cores: a imagem (a) é RGB, ao passo que (b) e (c) são, deliberadamente, monocromáticas.

```
10 int main(int argc, char** argv){
11   Mat x, y;
12
13   Size dimsX;
14
15   x = imread(argv[1], 0);
16
17   y = Mat(x.size(),CV_64FC1);
18
19   dimsX = x.size();
20
21   double minX, maxX;     // extremos de x
22   double minY, maxY;     // extremos de y
23   double fatorEscala;    // fator de normalizacao
24   double nu;             // frequencia de oscilacao
25   double A;              // amplitude de oscilacao
26   double dx;             // incremento na malha
27
28   // conversao e normalizacao de x para intervalo [0,1]
29   fatorEscala = 1.0 / 255.0;
30
31   x.convertTo(x,CV_64FC1,fatorEscala,0.0);
32
33   // calculo do incremento da malha dentro do intervalo
         [0,1]
34   dx = 1.0 / static_cast<double> (dimsX.width - 1);
35
36   // leitura dos parametros da oscilacao
37   cout << "Informe a frequencia de oscilacao:\t";
38   cin  >> nu;
39   cout << "Informe a amplitude de oscilacao:\t";
40   cin  >> A;
41
42   // modulacao da imagem original
43
44   for (int i = 0; i < dimsX.height; i++){
45     for (int j = 0; j < dimsX.width; j++){
46       y.at<double>(i,j) = sin(2.0 * pi * nu * dx *
        static_cast<double> (j));
47     }
48   }
49
50   x = A * x.mul(y);
51
52   // normalizacao de escala para visualizacao
53
54   minMaxLoc(x, &minX, &maxX);
55   minMaxLoc(y, &minY, &maxY);
56
57   x = (maxX / (maxX - minX)) * (x - minX);
58   y = (1.0 / (maxY - minY)) * (y - minY);
59
60   x.convertTo(x,CV_8UC1,255,0);
61   y.convertTo(y,CV_8UC1,255,0);
62
63   namedWindow("Imagem modulada",WINDOW_AUTOSIZE);
64   namedWindow("Modulacao",WINDOW_AUTOSIZE);
65
66   imshow("Imagem modulada", x);
67   imshow("Modulacao", y);
68
69   waitKey(0);
70
71   destroyAllWindows();
72
73   return 0;
74 }
```

Código 11: Implementação da transformação descrita pela Equação (7), cujos parâmetros A e ν são informados pelo usuário.

O acesso às entradas (i, j) da matriz `y` é realizado pelo método `cv::Mat::at`, cuja sintaxe

```
y.at<idTipo>(i,j)
```

dá acesso ao elemento desejado, sendo este do tipo `idTipo`. O identificador `idTipo` não pode ser escolhido arbitrariamente , mas sim de acordo com o tipo da matriz cujo elemento é extraído [13]. A Tabela III exibe quais identificadores de tipo devem ser usados em cada caso.

Tipo	Identificador
8U	uchar
8S	schar
16U	ushort
16S	short
32S	int
32F	float
64F	double

Tabela III: Identificadores de tipo a serem usados no acesso a elementos individuais de matrizes. Adaptada da documentação oficial do OpenCV [13].

3.2. Análise de histograma

Transformações de níveis de intensidade podem ser melhor caracterizadas por meio da análise de histograma. O histograma h de luminâncias de uma imagem monocromática de dimensões $M \times N$ quantizada em L níveis de cinza é um sinal unidimensional de L elementos, tal que sua i-ésima entrada,

$$h_i = \frac{f_i}{MN}, \tag{8}$$

é dada pela razão entre a frequência absoluta do i-ésimo nível de intensidade, f_i, pelo número total de elementos da imagem. Dessa forma, h_i pode ser tomado como uma aproximação à probabilidade de ocorrência do i-ésimo nível de intensidade, desde que o histograma seja normalizado, isto é,

$$\sum_{i=0}^{L-1} h_i = 1. \tag{9}$$

Embora a Equação (8) esteja associada a imagens monocromáticas, a extensão para imagens coloridas é imediata: cada canal tem seu próprio histograma. Entretanto, deve-se ter em mente que é possível formar histogramas multidimensionais, bastando realizar contagens de duas ou mais variáveis conjuntas.

A Figura 5 mostra os histogramas dos canais R, G e B da Figura 3, antes e depois da transformação de inversão de seus níveis de intensidadade. Comparando-se os histogramas antes de depois de uma transformação de intensidade – mesmo que desconhecida *a priori* – pode-se obter informações importantes sobre a natureza desta. Neste exemplo em particular, verifica-se claramente que os níveis de intensidade foram redistribuídos de forma que houvesse, na prática, uma inversão (claro torna-se escuro). Ainda analisando estes histogramas, verifica-se que suas formas não foram alteradas, indicando que a transformação não compreendia nenhum ajuste de contraste. Por fim, a observação dos histogramas permite que se extraiam informações qualitativas e quantitativas da imagem em questão, desde que pertinentes à estatística desta, tais como desvio-padrão, variância, média, entropia etc [7, 15].

Em OpenCV, histogramas são objetos da já apresentada classe `cv::Mat`. Em versões anteriores da biblioteca, havia um classe separada para histogramas, tornando os processos menos eficientes [10]. A função utilizada para obtenção do histograma de uma matriz é `cv:calcHist`, cuja sintaxe geral é um pouco mais extensa que as chamadas que temos visto até o momento [10, 13]:

```
calcHist(Mat*           imagens,
         int            numImagens,
         const int *    canais,
         Mat            mascara,
         Mat            hist,
         int            dimsHist,
         const int*     tamHist,
         const float**  intervalos,
         bool           eUniforme,
         bool           eAcumulado)
```

O primeiro argumento é o array `imagens`, que – tal como sugerido – corresponde a um conjunto de imagens sobre as quais pretende-se computar o histograma. O parâmetro `numImagens` corresponde simplesmente ao número de imagens contidas no array `imagens`. O array `canais` corresponde ao número de canais para cada imagem do array `imagens`. Isto é, para cada `imagens[i]`, há um número de canais a ser especificado[13].

A matriz `mascara`, quando não-nula, delimita regiões de interesse nas imagens-fonte, de forma que os pixels contidos fora destas regiões não são considerados no processo de obtenção do histograma. A matriz `hist` é a saída esperada, isto é, o histograma das imagens de entrada. O inteiro `dimsHist` corresponde à dimensão do histograma. Por sua vez, o array `tamHist` contém os tamanhos (número de entradas) em cada dimensão[14].

A forma com que o array `intervalos` é passado à função depende do valor do booleano `eUniforme`: se este é igual a `true`, então as divisões do histograma (chamadas de *bins*) são espaçadas igualmente entre si. Nesse caso, cada entrada de `intervalos` contém um vetor de dois elementos, relativos aos extremo inferior (inclusivo) e superior (exclusivo) dos níveis de intensidade da imagem em questão. Em outras palavras, `intervalos` – no caso de `eUniforme = true` – é um vetor `numImagens`$\times 2$, cuja i-ésima linha contém os ex-

[13] Deve-se observar que os canais começam em zero.

[14] Deve-se tomar cuidado para não confundir *tamanho* com *dimensão*: o primeiro refere-se ao número de entradas de um determinado histograma, ao passo que o segundo refere-se ao número de variáveis avaliadas simultaneamente. Por exemplo, se em uma imagem contarmos as ocorrências de seus níveis de intensidade *e* as áreas dos objetos representados, teríamos um histograma 2-D, sendo esta dimensionalidade independente do número de entradas em cada contagem.

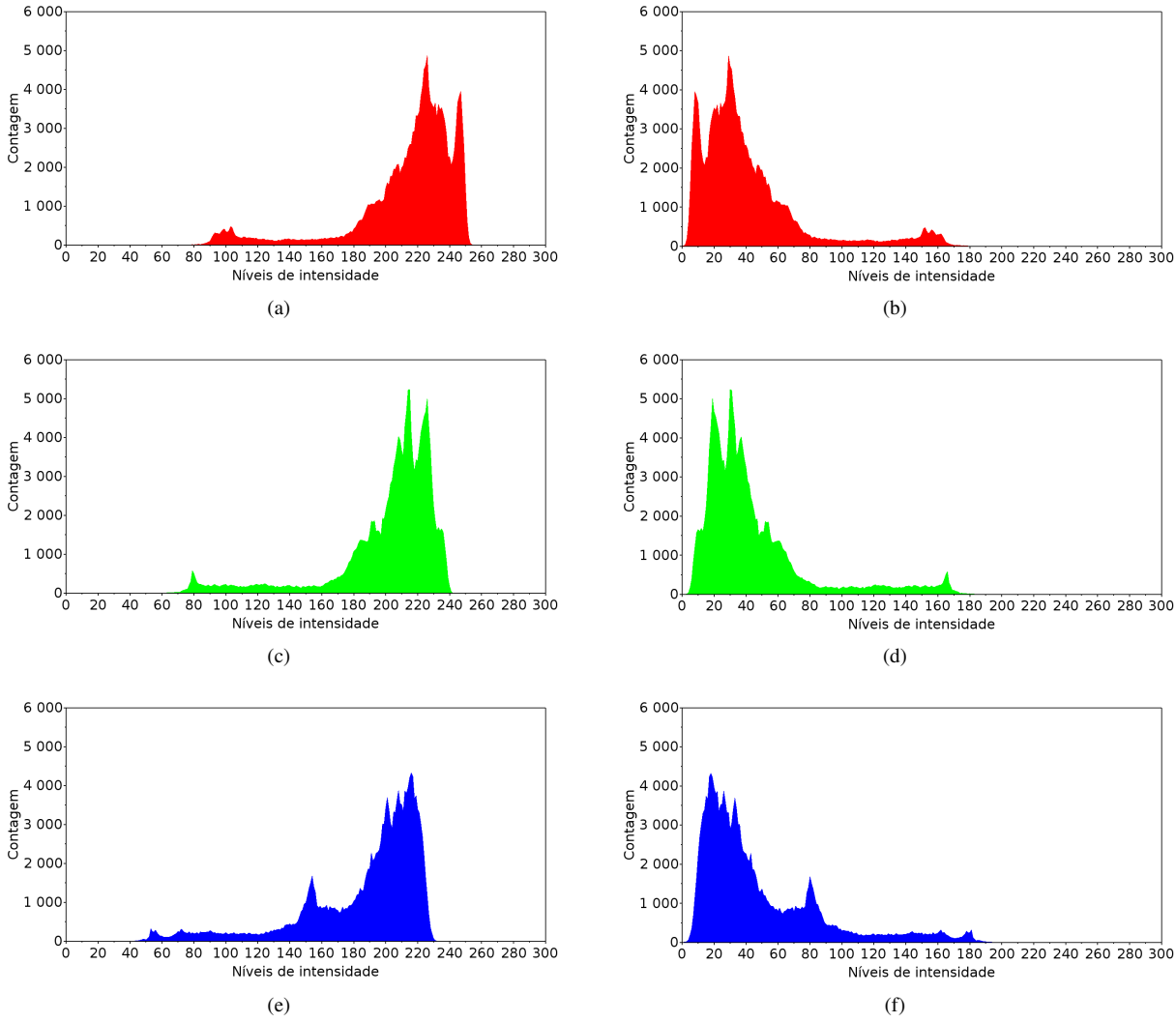


Figura 5: Análise de histograma da transformação de inversão dada pela Equação (6). Nas imagens (a), (c) e (e) são mostrados os histogramas dos canais R, G e B da imagem 3 (a), respectivamente. Em (b), (d) e (f) são mostrados os histogramas dos mesmos canais da imagem resultante, evidenciando as alterações. Nesse caso em particular, os histogramas sofreram espelhamentos, mas mantiveram suas formas, ou seja, não houve quaisquer ajustes de contraste.

tremos inferior e superior (sendo este último não incluído) dos bins do i-ésimo histograma. No caso não-uniforme (`eUniforme = false`), a iésima-entrada de `intervalos` contém um array de `tamHist[i] + 1` entradas.

3.2.1. *Determinação de parâmetros estatísticos*

Como primeiro exemplo, consideremos uma imagem U8 de 6×6 px, dada por

$$x = \begin{bmatrix} 4 & 3 & 4 & 5 & 3 & 5 \\ 3 & 5 & 4 & 1 & 7 & 3 \\ 6 & 7 & 7 & 4 & 2 & 2 \\ 3 & 4 & 8 & 6 & 5 & 6 \\ 3 & 4 & 5 & 5 & 2 & 6 \\ 5 & 6 & 6 & 4 & 5 & 3 \end{bmatrix}. \tag{10}$$

Desejamos calcular a ocorrência de cada um dos níveis de intensidade no intervalo inteiro $[0, 10]$. Além disso, queremos estimar direta e indiretamente o nível de intensidade médio e o desvio-padrão da imagem em questão. A estimativa direta da média e do desvio-padrão se dá por meio das seguintes expressões:

$$\langle x \rangle = \frac{1}{MN} \sum_{j=0}^{N-1} \sum_{i=0}^{M-1} x_{ij} \tag{11}$$

e

$$\sigma = \sqrt{\frac{1}{MN} \sum_{j=0}^{N-1} \sum_{i=0}^{M-1} (x_{ij} - \langle x \rangle)^2}. \tag{12}$$

A estimativa destas grandezas por meio do histograma normalizado parte do fato de que este é uma aproximação à função de densidade de probabilidade dos níveis de intensidade. Logo, média e desvio-padrão são dados indiretamente por

$$\langle x \rangle = \sum_{i=0}^{L-1} l_i h_i, \tag{13}$$

e

$$\sigma = \sqrt{\sum_{i=0}^{L-1} (l_i - \langle x \rangle)^2 h_i}, \tag{14}$$

em que l_i é o i-ésimo nível de intensidade, isto é, a coordenada do i-ésimo bin, e h_i é a i-ésima entrada do histograma normalizado. Desenvolvendo a soma da Equação (14), chegamos à expressão simplificada

$$\sigma = \sqrt{\langle x^2 \rangle - \langle x \rangle^2}, \tag{15}$$

que usaremos neste exemplo. Note que aparece o termo $\langle x^2 \rangle$, referente à média da matriz x quadrática, ou seja, com cada entrada original elevada ao quadrado. O Código 12 implementa o levantamento desejado.

```
# include <opencv2/opencv.hpp>
# include <iostream>
# include <cmath>

using namespace std;
using namespace cv;

int main(int argc, char** argv){
  Mat x, xQuad;
  Mat histX, histXQuad;

  int M = 6;     // dimensao da imagem x
  int tamHistX  = 11;   // tamanho do histograma de x
  int tamHistXQuad      = 101;  // tamanho do
     histograma de xQuad

  // extremos dos intervalos do histograma
  float rangeX[] = {0, 11};      // 0 a 10
  float rangeXQuad[] = {0, 101};         // 0 a 100

  const float* extrBinsHistX = { rangeX };
  const float* extrBinsHistXQuad = { rangeXQuad };

  // medias de x e de xQuad e desvio-padrao
  float medX[2] = {0.0, 0.0};
  float medXQuad[2] = {0.0, 0.0};

  // desvio-padrao
  float desvPadrao[2] = {0.0, 0.0};

  // variavel auxiliar: desvio quadratico
  float desvQuad = 0.0;

  // entradas da imagem x

  char entrX[M][M] = {
    {4, 3, 4, 5, 3, 5},
    {3, 5, 4, 1, 7, 3},
    {6, 7, 7, 4, 2, 2},
    {3, 4, 8, 6, 5, 6},
    {3, 4, 5, 5, 2, 6},
    {5, 6, 6, 4, 5, 3}
    };

  x = Mat(M, M, CV_8UC1, entrX);

  // atribuicao das entradas de xQuad
  xQuad = x.mul(x);

  cout << "Entradas da matriz x" << endl;
  cout << x << "\n\n";

  cout << "Entradas da matriz xQuad" << endl;
  cout << xQuad << "\n\n";

  // Calculo das medias diretamente sobre os dados
  for (int i = 0; i < M; i++){
    for (int j = 0; j < M; j++){
      medX[0] += static_cast<float>(x.at<char>(i,j));

      medXQuad[0]+=  + static_cast<float>(xQuad.at<char
     >(i,j));
    }
  }

  medX[0] = medX[0] / (M * M);
  medXQuad[0]   = medXQuad[0] / (M * M);

  // Computo dos histogramas
  calcHist(&x, 1, 0, Mat(), histX, 1, &tamHistX, &
     extrBinsHistX, true, false);

  calcHist(&xQuad, 1, 0, Mat(), histXQuad, 1, &
     tamHistXQuad, &extrBinsHistXQuad, true, false);

  // exibicao das entradas nao-nulas dos histogramas
  cout << "[ HISTOGRAMA DE x ]" << endl;
  cout << "bin\t | \tcontagem"  << endl;

  for (int i = 0; i < tamHistX; i++){
    if (histX.at<float>(i) != 0.0){
      cout << i << "\t | \t" << histX.at<float>(i) <<
     endl;
    }
  }

  cout << "\n\n[ HISTOGRAMA DE xQuad ]" << endl;
  cout << "bin\t | \tcontagem"  << endl;
```

```
85  for (int i = 0; i < tamHistXQuad; i++){
86    if (histXQuad.at<float>(i) != 0.0){
87      cout << i << "\t | \t" << histXQuad.at<float>(i)
      << endl;
88    }
89  }
90
91  // normalizacao dos histogramas
92  histX = histX / (M * M);
93  histXQuad = histXQuad / (M * M);
94
95  // calculo das medias indiretamente, por meio dos
      histogramas
96  for (int i = 0; i < tamHistX; i++){
97    medX[1] += static_cast<float>(i) * histX.at<float>(
      i);
98  }
99
100 for (int i = 0; i < tamHistXQuad; i++){
101   medXQuad[1] += static_cast<float>(i) * histXQuad.at
      <float>(i);
102 }
103
104 // calculo do desvio-padrao sobre os dados
105 for (int i = 0; i < M; i++){
106  for (int j = 0; j < M; j++){
107    desvQuad = static_cast<double>(x.at<char>(i,j)) -
      medX[0];
108    desvPadrao[0] += (desvQuad * desvQuad);
109   }
110 }
111
112 desvPadrao[0] = sqrt(desvPadrao[0] / (M * M));
113
114 // calculo do desvio-padrao baseado no histograma
115
116 desvPadrao[1] = sqrt(medXQuad[1] - (medX[1] * medX
      [1]));
117
118 cout << "\n\nMedias de x calculadas sobre dados e
      sobre histograma" << endl;
119 cout << medX[0] << "\t" << medX[1] << "\n\n";
120
121 cout << "Medias de xQuad calculadas sobre dados e
      sobre o histograma" << endl;
122 cout << medXQuad[0] << "\t" << medXQuad[1] << "\n\n";
123
124 cout << "Desvios-padrao calculados sobre dados e
      sobre o histograma" << endl;
125 cout << desvPadrao[0] << "\t" << desvPadrao[1] <<
      endl;
126
127 return 0;
128 }
```

Código 12: Implementação de programa para análise de histograma e cálculos direto e indireto de média e desvio-padrão da matriz dada pela Eq. (10).

No Código 12, as variáveis `histX` e `histXQuad` são quem receberão as contagens. As variáveis `tamHistX` e `tamHistXQuad` definem o tamanho (número de bins) de seus respectivos histogramas, ao passo que os arrays `rangeX` e `rangeXQuad` definem os intervalos, ou valores de cada bin. Deve-se notar nas linhas 17 e 18 que o limite superior contido nos vetores não é considerado, tal como explicitado nos parágrafos anteriores. Nas linhas 24, 25 e 28 são declarados os arrays `medX`, `medXQuad` e `DesvPadrao`, que receberão as médidas de `x`, `xQuad` e o desvio-padrão de `x` calculados direta e indiretamente, isto é, diretamente sobre os dados e indiretamente, sobre os histogramas. Os cálculos diretos das médias são iniciados no laço das linha 56. Nas linhas 68 e 70 são computados os histogramas propriamente ditos.

Os laços das linhas 76 e 85 exibem os bins e suas respectivas contagens, desde que estas sejam não-nulas. Deve-se esperar as seguintes saídas:

```
[ HISTOGRAMA DE x ]
bin       |       contagem
1         |       1
2         |       3
3         |       7
4         |       7
5         |       8
6         |       6
7         |       3
8         |       1


[ HISTOGRAMA DE xQuad ]
bin       |       contagem
1         |       1
4         |       3
9         |       7
16        |       7
25        |       8
36        |       6
49        |       3
64        |       1
```

A Figura 6 mostra um gráfico do histograma de x, evidenciando um perfil Gaussiano da distribuição.

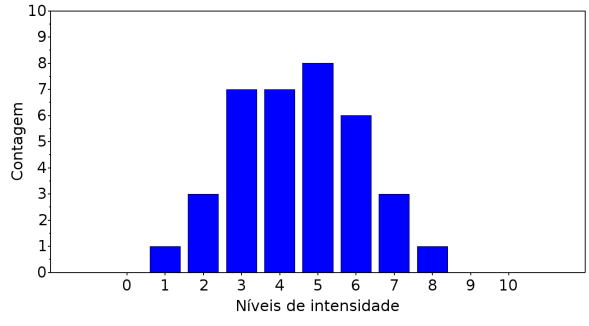


Figura 6: Distribuição dos níveis de intensidade da matriz dada pela Equação (10). Nota-se um perfil Gaussiano da distribuição.

Os laços das linhas 96 e 100 iniciam os cômputos indiretos das médias, ao passo que o laço da linha 105 inicia

o cômputo direto do desvio-padrão. O cálculo indireto do desvio-padrão é realizado na linha 116. Ao final do código, são mostrados os valores médios e desvio-padrão calculados direta e indiretamente. Os resultados obtidos são idênticos até a quinta casa decimal e são exibidos na Tabela IV.

Grandeza	**Resultado**
$\langle x \rangle$	4.47222
$\langle x^2 \rangle$	22.63888
σ	1.62422

Tabela IV: Resultados das estimativas sobre a matriz dada pela Equação (10). Resultados truncados na quarta casa decimal.

Os resultados deste exemplo ilustram que, de fato, o histograma – quando bem estimado – proporciona boa aproximação à função densidade de probabilidade das grandezas levantadas[15].

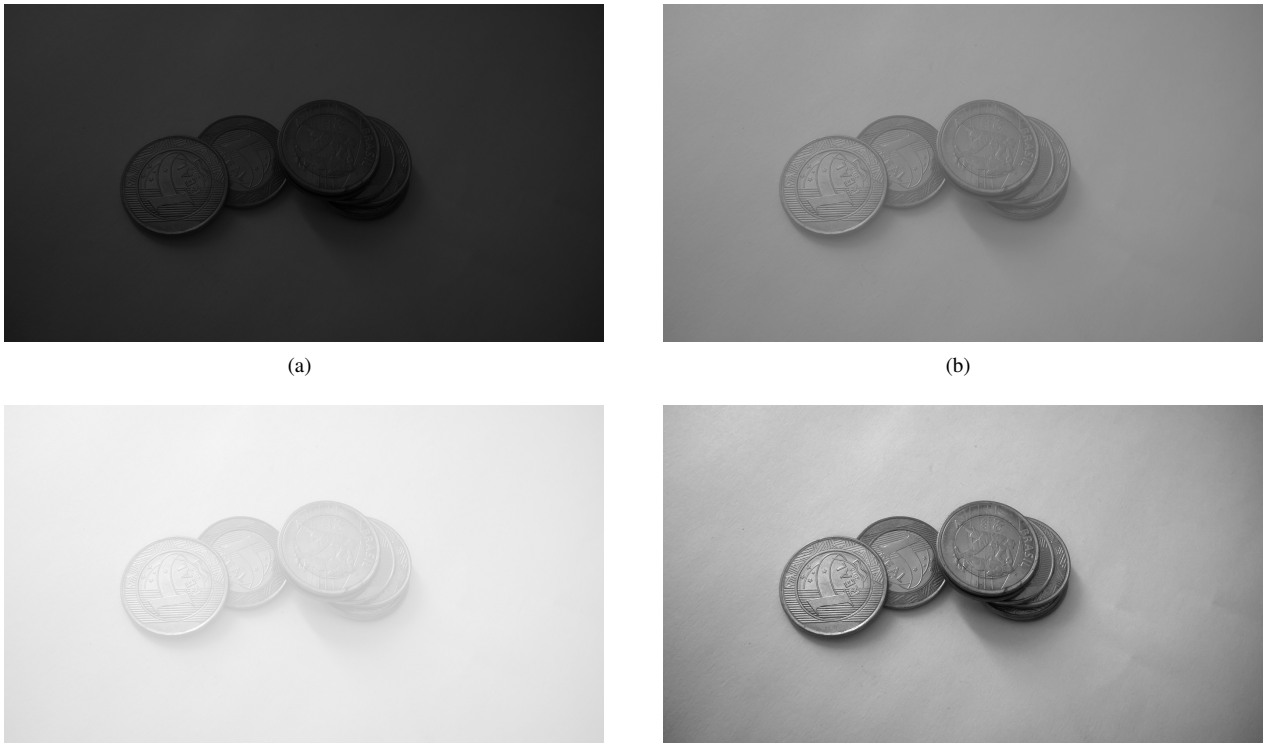
3.2.2. *Ajuste linear de contraste*

Como último exemplo, revisitemos a questão do ajuste linear de contraste, dado pela Equação (4). Tal como dito no início da discussão sobre histogramas, estes objetos permitem formar juízo sobre os ajustes sofridos pelas imagens sob análise. Consideremos então uma aplicação em que o usuário deva fornecer uma imagem de entrada e os extremos do intervalo de destino desejado. Por simplicidade, normalizemos a escala dinâmica da imagem de entrada para o intervalo $[0,1]$. Dependendo dos valores informados (que correspondem na verdade a percentuais de escala dinâmica), imagens de maior ou menor contraste podem ser obtidas. A Figura 7 ilustra algumas possibilidades usando como input a Figura 1-(c).

O Código 13 implementa este exemplo, recebendo como entrada uma imagem qualquer, convertendo-a para tons de cinza com intervalo normalizado, realizando por fim o ajuste. Há de se notar que além da imagem de entrada, esperam-se outros argumentos: `argv[2]` deverá conter o nome para o arquivo em que serão salvas as entradas do histograma, e `argv[3]` deverá conter o nome do arquivo a receber a imagem transformada.

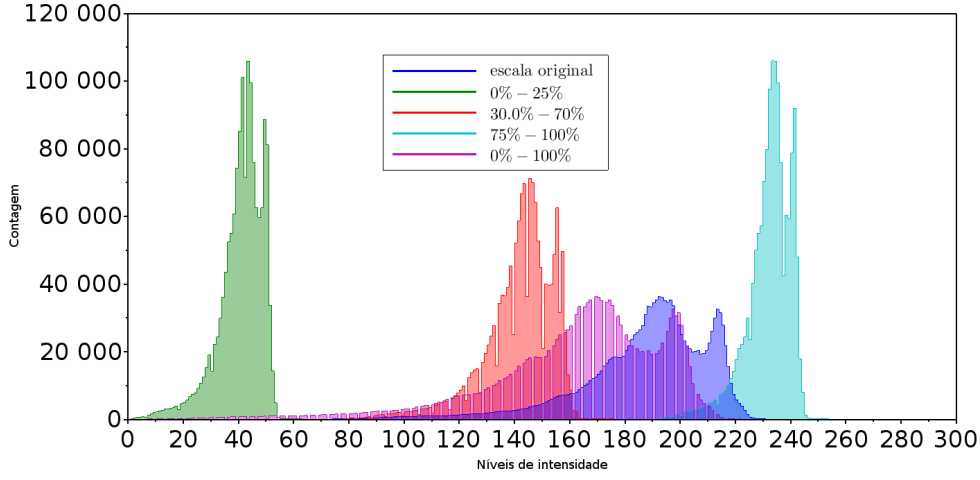
```
1 #include <opencv2/opencv.hpp>
2 #include <iostream>
3 #include <fstream>
4
5 using namespace std;
6 using namespace cv;
7
8 int main(int argc, char** argv){
9   ofstream escreveArquivo;
10
11   Mat x, y, histX, histY;
12
13   double extrEscalas[2][2];
14
15   const float fatorEscala = 1.0 / 255.0;
16   float range[] = {0, 256};
17   const float* extrBinsHist = { range };
18   float a, b;
19
20   int tamHist = 256;
21
22   x = imread(argv[1], 0);
23   x.convertTo(x, CV_32FC1, fatorEscala, 0.0);
24
25   minMaxLoc(x, &extrEscalas[0][0], &extrEscalas[0][1]);
26
27   cout << "\n\nInforme os extremos (normalizados) da
        nova escala:\n";
28   cin  >> extrEscalas[1][0];
29   cin  >> extrEscalas[1][1];
30
31   cout << "\nExtremos da escala original\n";
32   cout << extrEscalas[0][0] << "\t" << extrEscalas
        [0][1] << "\n";
33
34   cout << "\nExtremos da nova escala\n";
35   cout << extrEscalas[1][0] << "\t" << extrEscalas
        [1][1] << "\n\n";
36
37   a = (extrEscalas[1][1] - extrEscalas[1][0]) / (
        extrEscalas[0][1] - extrEscalas[0][0]);
38
39   b = extrEscalas[1][0];
40
41   y = a * (x - extrEscalas[0][0]) + b;
42
43   cout << "\nFator de escala: " << a << endl;
44   cout << "Nivel DC: " << b << "\n\n";
45
46   namedWindow("Imagem original", WINDOW_AUTOSIZE);
47   namedWindow("Imagem transformada", WINDOW_AUTOSIZE);
48
49   imshow("Imagem original", x);
50   imshow("Imagem transformada", y);
51
52   x.convertTo(x, CV_8UC1, 255, 0);
53   y.convertTo(y, CV_8UC1, 255, 0);
54
55   calcHist(&x, 1, 0, Mat(), histX, 1, &tamHist, &
        extrBinsHist, true, false);
56   calcHist(&y, 1, 0, Mat(), histY, 1, &tamHist, &
        extrBinsHist, true, false);
57
58   escreveArquivo.open(argv[2]);
59
60   for(int i = 0; i < tamHist; i++){
61     escreveArquivo << i << "\t" << histX.at<float>(i)
       << "\t" << histY.at<float>(i) << "\n";
62   }
```

[15] Um exercício interessante consiste em se alterar no Código 12 o número de bins ou os intervalos dos histogramas e verificar o efeito devastador sobre os resultados.



(a) (b)

(c) (d)



(e)

Figura 7: Análise de histogramas de transformações de intensidade da imagem 1-(c). São mostrados, de (a) a (c), os resultados de contrações para os intervalos $[0\%, 25\%], [30\%, 70\%], [75\%, 100\%]$, ao passo que em (d) é mostrado um alargamento linear total, levando a imagem original a toda extensão da escala dinâmica. Em (e) são mostrados os histogramas dos níveis de intensidade dos resultados, bem como da imagem de entrada. Neste gráfico fica evidente o caráter de deslocamento e deformação da operação de ajuste de contraste.

```
63
64  escreveArquivo.close();
65
66  imwrite(argv[3], y);
67
68  waitKey(0);
69
70  destroyAllWindows();
71
72  return 0;
73 }
```

Código 13: Implementação de programa para ajuste linear de contraste sobre uma imagem informada na chamada da aplicação.

3.3. Filtragem espacial

Finalizamos este trabalho com uma breve discussão sobre filtragem linear espacial sobre imagens e sua implementação em OpenCV. Naturalmente, este é um assunto bastante amplo de extremamente debatido na literatura especializada. Dito isto, o leitor interessado poderá recorrer às referências [15] e [7] para maior aprofundamento no conteúdo.

Diferentemente das transformações de intensidade discutidas na Seção 3.1, em que o resultado da operação depende única e exclusivamente do valor atual do pixel a ser processado, transformações espaciais consideram contribuições de pixels vizinhos. Nesse sentido, dizemos que o resultado de uma filtragem linear sobre um dado elemento I_{ij} depende da *vizinhança* deste. A Equação (16) ilustra as duas vizinhanças mais usuais em imagens bidimensionais, N_4 e N_8, respectivamente.

$$\begin{cases} N_4(i,j) & = \begin{matrix} & I_{i-1,j} & \\ I_{i,j-1} & I_{i,j} & I_{i,j+1} \\ & I_{i+1,j} & \end{matrix} \\ \\ N_8(i,j) & = \begin{matrix} I_{i-1,j-1} & I_{i-1,j} & I_{i-1,j+1} \\ I_{i,j-1} & I_{i,j} & I_{i,j+1} \\ I_{i+1,j-1} & I_{i+1,j} & I_{i+1,j+1} \end{matrix} \end{cases} \tag{16}$$

O que diferencia as vizinhanças N_4 e N_8 é que nesta última os pixels das diagonais são considerados. Deve-se notar que para dimensões superiores (imagens multidimensionais) estas vizinhanças devem ser estendidas. A importância do conceito de vizinhaça fica evidente ao se implementar processos de convolução, discutidos a seguir.

O principal resultado da teoria de sistemas lineares corresponde ao fato de que uma convolução no domínio espacial entre um filtro h e uma imagem I leva a uma multiplicação usual no domínio das frequências entre as transformadas de Fourier $\hat{h}$ e $\hat{I}$. Em uma dimensão contínua, tem-se então que

$$(h*I)(x) \longleftrightarrow \hat{h}(\nu)\cdot\hat{I}(\nu), \tag{17}$$

em que x e ν correspondem às variáveis espacial e frequencial, respectivamente. As expressões do produto de convolução e da transformada de Fourier são dadas – em uma dimensão – respectivamente por

$$(h*I)(x) = \int_{-\infty}^{\infty} I(u)h(x-u)du, \tag{18}$$

e

$$\hat{I}(\nu) = \int_{-\infty}^{\infty} I(x)\exp(-j2\pi\nu x)\,dx, \tag{19}$$

em que $j=\sqrt{-1}$. Deve-se notar que as Equações (18) e (19) foram definidas sobre uma variável contínua, x. Em imagens discretas, lida-se com, no mínimo, duas variáveis *discretas*. As versões discretas bidimensionais dos resultados anteriores são dadas por

$$(h*I)_{ij} = \sum_{k=0}^{M-1}\sum_{l=0}^{N-1} I_{kl}h_{i-k,j-l}, \tag{20}$$

e

$$\hat{I}_{mn} = \sum_{k=0}^{M-1}\sum_{l=0}^{N-1} I_{kl}\exp\left(-j2\pi\frac{m}{M}k - j2\pi\frac{n}{N}l\right). \tag{21}$$

Observando a Equação (20), fica claro o papel das vizinhanças: dependendo das dimensões do filtro h, vizinhanças de tamanhos variáveis podem ser usadas no cômputo da convolução, influenciando muito ou pouco o resultado da operação[16].

Em OpenCV contamos com a função `cv::filter2D`, que realiza uma operação de *correlação* discreta, dada por

$$\sum_{k=0}^{M-1}\sum_{l=0}^{N-1} I_{kl}h_{i+k,j+l}, \tag{22}$$

que difere da convolução mostrada na Equação (20) apenas por uma mudança de sinais nos índices das somas. Note-se que caso o filtro seja uma matriz simétrica, o efeitos da convolução e da correlação são idênticos. Mais à frente será feito uso de uma função específica para "rebater" as entradas de um filtro, fazendo com que a correlação corresponda, de fato, a uma convolução. A sintaxe básica para a chamada da função `filter2D` é da forma [13]

```
filter2D(Mat    x,
         Mat    y,
         int    profBits,
         Mat    h,
         Point  centro,
         double nivDC,
         int    extensaoBordas
         ).
```

As matrizes `x`, `y` e `h` correspondem às imagens de entrada e saída e o filtro, respectivamente. O inteiro `profBits` determina qual a profundidade em bits de `y`. Se `profBits = -1`, `y` será do mesmo formato que `x`. Outros enumeradores já usados nestas Notas podem ser usados, tais como `CV_8U`, `CV_32F` etc. Nesta chamada aparece pela primeira vez uma variável da classe `cv::Point`, que implementa coordenadas bi ou tridimensionais em OpenCV. Uma variável `P` da classe `Point` possui como funções membros o acesso às suas coordenadas, `P.x`, `P.y` e `P.z`, sendo esta última válida apenas

[16] Existe um compromisso entre tamanho do filtro e valor de seus coeficientes: o efeito prático de um filtro extenso mas com entradas distantes do pixel central pequenas comparativamente não difere muito de um filtro trucado de menor dimensão. Efetivamente, a magnitude dos coeficientes do filtro nas regiões distantes do centro da máscara influi na *correlação* espacial do pixel central com os demais.

quando o ponto for definido com três coordenadas. Há também flexibilidade no tipo de dado: coordenadas podem ser inteiras com e sem sinal e reais [10].

Na chamada da função `filter2D` a variável `centro` corresponde, obviamente, à posição em que o filtro (também chamado de máscara) será centralizado. Seu valor padrão é `(-1,-1)`, correspondente ao centro geométrico da máscara. A variável `nivDC` permite que seja adicionado um nível DC a `y`[17] O valor padrão para `nivDC` é `0.0`.

O enumerador `extensaoBordas` define o comportamento da convolução frente à operação de pixels que não constam na imagem original. Este é um requisito prático, uma vez que em um cenário de processamento digital, não é possível ter sinais ilimitados (vide limites da soma na Equação (20)). Os enumeradores básicos são mostrados na Tabela V. Nesta mesma pode ser visto que a opção padrão na chamada da função `filter2D`, correspondente ao enumerador `BORDER_DEFAULT`, trabalha com reflexão dos pixels, à exceção dos pixels de borda.

Enumerador	Descrição
`BORDER_CONSTANT`	Extrapola as bordas segundo um nível constante de intensidade informado
`BORDER_REPLICATE`	Extrapola as bordas por meio de replicação dos pixels de borda originais
`BORDER_REFLECT`	Extrapola as bordas refletindo os pixels da imagem naquela direção
`BORDER_WRAP`	Extrapola as bordas usando condições de contorno periódicas
`BORDER_REFLECT_101`	Análogo a `BORDER_REFLECT`, mas sem a replicação dos pixels das bordas
`BORDER_DEFAULT`	Idêntico à opção `BORDER_REFLECT_101`

Tabela V: Enumeradores básicos para controlar o comportamento das bordas no processo de filtragem espacial. Adaptado da documentação oficial do OpenCV [13].

3.3.1. *Filtro média*

Como primeira aplicação, consideremos um filtro de média aritmética de dimensões $M \times N$. Sendo uma média aritmética, as entradas do filtro são todas iguais, isto é:

$$h = \frac{1}{MN}\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & 1 \\ 1 & 1 & 1 & \cdots & 1 \end{bmatrix}_{M\times N}. \tag{23}$$

Na literatura filtros com os coeficientes idênticos são chamados do *box-filters* [7]. O filtro box dado na Equação (23) corresponde a uma passa-baixas [7, 15], de tal forma que sua principal aplicação é suavizar bordas e contornos. A suavização será tão intensa quão extenso for o filtro. O Código 14 implementa este processo de filtragem explicitamente.

```
1 #include <opencv2/opencv.hpp>
2 #include <iostream>
3
4 using namespace std;
5 using namespace cv;
6
7 int main(int argc, char** argv){
8   Mat x, y, h;
9
10  Size dimsH;
11
12  x = imread(argv[1], 0);
13
14  cout << "\n\nInforme as dimensoes do filtro h:\n";
15  cin  >> dimsH.width;
16  cin  >> dimsH.height;
17
18  h = Mat::ones(dimsH, CV_64F);
19
20  h = h / static_cast<double>(dimsH.width * dimsH.
      height);
21
22  filter2D(x, y, CV_8U, h, Point(-1,-1), 0.0,
      BORDER_DEFAULT);
23
24  namedWindow("Imagem original", WINDOW_AUTOSIZE);
25  namedWindow("Imagem filtrada", WINDOW_AUTOSIZE);
26
27  imshow("Imagem original", x);
28  imshow("Imagem filtrada", y);
29
30  waitKey(0);
31
32  destroyAllWindows();
33
34  imwrite(argv[2], y);
35
36  return 0;}
```

Código 14: Implementação de um processo de filtragem passa-baixas por meio de filtro box, cujas dimensões são informadas pelo usuário durante a execução.

A definição de `h` no Código 14 faz uso da função `cv::Mat::ones`, que retorna uma matriz cujas entradas são todas iguais a 1. Note que o filtro foi submetido a uma normalização na linha 20. Por fim, deve-se notar que o usuário deve informar na chamada da aplicação os arquivos de entrada e saída das imagens. A Figura 8 ilustra o efeito do uso de filtros boxes de diferentes tamanhos sobre a Figura 1-(d).

No exemplo mostrado, a definição do filtro `h` se deu de modo explícito. Entretanto, box filters são bastante usuais e por isso OpenCV dispõe de função própria para a realização desse tipo de filtragem. A função `cv::boxFilter` implementa este processo e sua sintaxe geral é da forma [13]

[17] É importante perceber que a atribuição do nível DC ocorre sobre a imagem de saída, e não a de entrada. Os resultados são distintos para cada caso.

(a)

(b)

(c)

Figura 8: Resultado de filtragens com filtros-box sobre a Figura 1-(d) com diferentes tamanhos de máscara. São mostrados os resultados do uso de máscaras de dimensões 20×20, 100×100 e 1×256, respectivamente.

```
boxFilter(Mat    x,
          Mat    y,
          int    profBits,
          Size   dimsH,
          Point   centro,
          bool   bNormaliza,
          int    extensaoBordas
         ).
```

Há duas diferenças básicas entre a chamada desta função e `cv::filter2D`: a exigência da variável booleana `bNormaliza`, cujo valor padrão é `true`. Nesse caso, o filtro box passa pela normalização a queh foi submetida no Código 14, linha 20. Naturalmente, caso `bNormaliza = false`, nenhuma normalização é feita, levando a uma amplificação no resultado final, juntamente com a suavização desejada. Ademais, é necessário que seja informado o tamanho de `h` por meio da variável do tipo `Size` dimsH.

3.3.2. *Filtro Gaussiano*

Filtros Gaussianos são filtros passa-baixas muito mais eficientes que box filters equivalentes[18]. A expressão geral para uma função Gaussiana em uma dimensão contínua é dada por

$$G(x;\sigma,\mu) = \frac{1}{\sqrt{\pi}\sigma} \exp\left[-\left(\frac{x-\mu}{\sqrt{2}\sigma}\right)^2\right], \qquad (24)$$

em que σ e μ correspondem ao desvio-padrão e à média, respectivamente. O desvio σ está ligado à largura da Gaussiana, tal como pode ser visto na Figura 9. Em um cenário

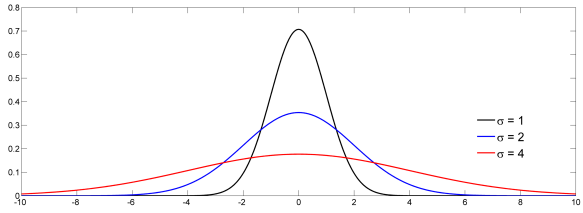


Figura 9: Influência do desvio-padrão sobre a largura da Gaussiana. Figura extraída da referência [15].

discreto, Gaussianas podem ser aproximados de algumas maneiras distintas [15].De modo mais geral, um filtro Gaussiano 1-D de dimensões $1 \times M$ e desvio-padrão σ é dado por

$$g_i = C \exp\left[-\left(\frac{i-\frac{M}{2}}{\sqrt{2}\sigma}\right)^2\right], \qquad (25)$$

em que c é uma constante de normalização tal que $\sum_{i=0}^{M-1} g_i = 1$. A Equação (25) é exatamente a expressão implementada em OpenCV [13].

Antes de apresentar os métodos para realização de filtragem Gaussiana, é importante notar que até agora tratamos apenas de Gaussianas 1-D, ao invés de explicitar as expressões 2-D. Isso se deve ao fato de que o filtro Gaussiano, diferentemente dos filtros-caixa, é um operador *separável* [7, 15]. Isto significa que a filtragem 2-D pode ser operacionalizada por uma sequência de duas filtragens 1-D: uma sobre as linhas e outra sobre as colunas (da imagem previamente filtrada na direção perpendicular). Nesse sentido, para se obter um núcleo 2-D de um filtro Gaussiano, basta realizar o produto matricial entre dois filtros unidimensionais. Isso nos dá a flexibilidade de usar diferentes desvios-padrão, um para cada direção da imagem.

A Figura 10 ilustra a utilização de três filtro Gaussianos sobre a imagem da Figura 1-(d), sendo um deles assimétrico.

[18] Isso se deve ao fato de que filtro caixa possuírem descontinuidades abruptas nas bordas, introduzindo artefatos na imagem conhecidos como fenômeno de Gibbs[7, 15].

O Código 15 implementa um processo de filtragem Gaussiana por meio da definição explícita de dois núcleos 1-D.

```
1 #include <opencv2/opencv.hpp>
2 #include <iostream>
3 #include <cmath>
4
5 using namespace std;
6 using namespace cv;
7
8 int main(int argc, char** argv){
9   Mat x, y, hx, hy, G;
10
11  int tamHx, tamHy;
12
13  double sigmaX, sigmaY, aux1, aux2, fatorEscala;
14
15  // quadrivetor a receber a constante de normalizacao
16  Scalar fatorNormalizacao;
17
18  x = imread(argv[1], 0);
19
20  cout << "\nInforme o tamanho do filtro hx:\n";
21  cin >> tamHx;
22  cout << "\nInforme o desvio-padrao do filtro hx:\n";
23  cin >> sigmaX;
24
25  cout << "\nInforme o tamanho do filtro hy:\n";
26  cin >> tamHy;
27  cout << "\nInforme o desvio-padrao do filtro hy:\n";
28  cin >> sigmaY;
29
30  hx = Mat(1, tamHx, CV_64FC1);
31  hy = Mat(1, tamHy, CV_64FC1);
32
33  // argumento auxiliar para Gaussiana em x
34  aux1  = 1.0 / (2.0 * sigmaX * sigmaX);
35
36  for(int i = 0; i < tamHx; i++){
37    aux2 = static_cast<double>(i - (tamHx) / 2);
38
39    hx.at<double>(i) = exp(-aux1 * (aux2 * aux2));
40  }
41
42  // argumento auxiliar para Gaussiana em y
43  aux1  = 1.0 / (2.0 * sigmaY * sigmaY);
44
45  for(int i = 0; i < tamHy; i++){
46    aux2 = static_cast<double>(i - (tamHy) / 2);
47
48    hy.at<double>(i) = exp(-aux1 * (aux2 * aux2));
49  }
50
51  // produto matricial entre hx transposto e hy
52  G = hx.t() * hy;
53
54  fatorNormalizacao = sum(G);
55
56  cout << "Scalar contendo fator de normalizacao:\n";
57  cout << fatorNormalizacao << endl;
58
59  // normalizacao do nucleo
60  G = G / fatorNormalizacao[0];
61
62  // rebatimento das entradas do nucle em ambas
      direcoes
63  flip(G, G, -1);
64
65  filter2D(x, y, CV_8U, G, Point(-1,-1), 0.0,
      BORDER_DEFAULT);
66
67  namedWindow("Filtro",WINDOW_AUTOSIZE);
68  namedWindow("Imagem original", WINDOW_AUTOSIZE);
69  namedWindow("Imagem filtrada", WINDOW_AUTOSIZE);
70
71  minMaxLoc(G, NULL, &fatorEscala);
72
73  fatorEscala = 1.0 / fatorEscala;
74
75  G.convertTo(G, CV_8UC1, 255 * fatorEscala, 0);
76
77  imshow("Filtro", G);
78  imshow("Imagem original", x);
79  imshow("Imagem filtrada", y);
80
81  waitKey(0);
82
83  destroyAllWindows();
84
85  imwrite(argv[2], G);
86  imwrite(argv[3], y);
87
88  return 0;}
```

Código 15: Implementação de um processo explícito de filtragem Gaussiana por meio da propriedade de separabilidade destes filtros. Ao usuário é solicitado informar o tamanho e desvio-padrão de cada filtro 1-D, além de fornecer os arquivos de entrada e saída para as imagens original e processada.

O Código 15 guarda diversos detalhes interessantes, além de conter métodos ainda não vistos. Na linha 16 é declarada a variável `fatorNormalizacao`, que como o próprio nome indica, tem a função de ser a constante de normalização a ser multiplicada pelo núcleo da Gaussiana 2-D. A razão para a definição desta constante como uma variável do tipo `Scalar` – que conforme já discutido implementa quadrivetores – ficará clara mais à frente. Entre as linhas 20 e 28 são implementadas as etapas de leitura do programa das dimensões e desvios-padrão dos dois filtros 1-D, `hx` e `hy`. Entre as linhas 36 e 49 as entradas das duas Gaussianas 1-D são calculadas. A linha 52 contém a instrução chave na definição da Gaussiana 2-D: conforme dito acima, um filtro Gaussiano 2-D pode ser expresso como o produto de dois filtros 1-D. Se os filtros, h_x e h_y têm dimensões $1 \times M$ e $1 \times N$, a Gaussiana

correspondente, G, é dada por

$$G = h_y^{\mathrm{T}} h_x, \tag{26}$$

em que T denota matriz transposta[19]. Entretanto, fizemos uso do padrão de coordenadas em processamento digital de imagens, em que o eixo y corresponde à direção vertical, crescente de cima para baixo (vide Equação (16)). Em OpenCV, a transposição de matrizes é implementada pelo método `cv::Mat::t()`, tal como pode ser visto na linha 52. A linha 54 contém uma chamada à função `cv::sum`, cuja sintaxe

```
S = sum(x)
```

atribui à variável `S` do tipo `Scalar` a soma dos elementos da matriz `x` em cada um de seus canais. Eis então a razão de termos declarado `fatorNormalizacao` como um `Scalar`. A normalização ocorre, de fato, na linha 60, em que o primeiro elemento deste quadrivetor é utilizado. Na linha 63 é chamada a função `cv::flip`, cuja função é realizar o rebatimento das entradas do núcleo em uma ou mais direções. Isto é necessário para que a soma de correlação, dada pela Equação (22) se transforme numa convolução discreta, dada pela Equação (20). A sintaxe básica desta função é da forma [10, 13]

```
flip(Mat x, Mat y, int codFlip),
```

em que o inteiro `codFlip` define sobre quais eixos as entradas serão rebatidas. Se `codFlip = 0`, `y` recebe uma versão de `x` rebatida em torno do eixo x. Caso `codFlip > 0` o rebatimento ocorre em torno do eixo y. Por fim, se `codFlip < 0`, o rebatimento ocorre em ambas as direções. O leitor mais atento deve ter notado que tanto no caso dos filtros-caixa quantos nos Gaussianos com $\sigma_x = \sigma_y$ lida-se com matrizes simétricas, de forma que `flip` não surte efeito prático. Entretanto, no caso de $\sigma_x \neq \sigma_y$ ou de qualquer outro filtro assimétrico, a não chamada da função de rebatimento levará o processo de filtragem a ocorrer como uma correlação, ao invés de uma convolução. Isto pode trazer problemas quando se deseja fazer análise no domínio das frequências [7, 15].

3.3.3. *Filtros separáveis*

Para finalizar a discussão sobre filtragem Gaussiana, cabe dizer que a definição dos filtros Gaussianos não precisa ocorrer de maneira explícita, tal como realizado no Código 15. Em OpenCV conta-se com a função `cv::getGaussianKernel`, cuja sintaxe geral é da forma [10, 13]

```
getGaussianKernel(int    tamNucleo,
                  double sigma,
                  int    tipoFiltro).
```

[19] Em princípio, a Equação (26) também estaria correta se usássemos $G = h_x^{\mathrm{T}} h_y$

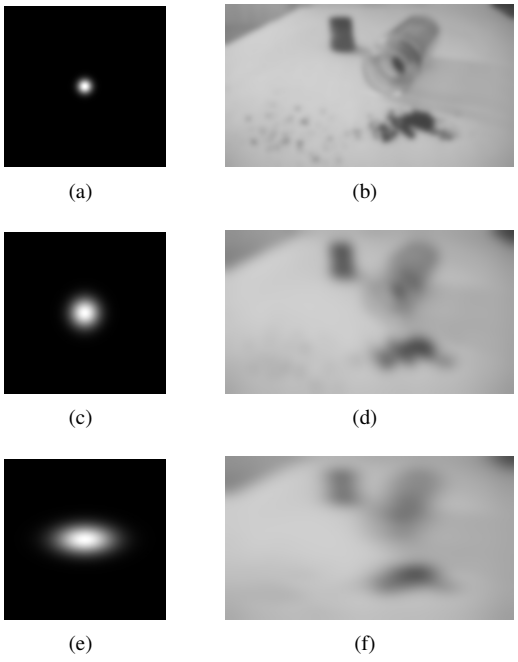


Figura 10: Exemplos de filtragem Gaussiana sobre a Figura 1-(d). Em (a), (c) e (e) são mostrados três núcleos com $(\sigma_x, \sigma_y) = (16, 16)$, $(32, 32)$ e $(64, 32)$, respectivamente. Os valores são dados em pixels e os respectivos resultados são mostrados em (b), (d) e (f). Em todos os casos, os núcleos têm dimensões de 512×512 pixels.

Esta função retorna um filtro unidimensional `tamNucleo` $\times 1$ cujos coeficientes aproximam uma Gaussiana com desvio-padrão dado por `sigma`. Há o requisito de `tamNucleo` ser ímpar. O inteiro `tipoFiltro` define qual o tipo das entradas do filtro, podendo ser F32 ou F64. Para realizar a filtragem bidimensional por meio da função `cv::getGaussianKernel` pode-se recorrer ao produto matricial usado no Código 15 ou à função `cv::sepFilter2D`, que implementa um processo de filtragem por meio de operadores separáveis. Sua sintaxe geral é da forma

```
sepFilter2D(Mat x,
            Mat y,
            int profBits,
            Mat hX,
            Mat hY,
            Point   centro,
            double  nivDC,
            int extensaoBorda),
```

em que `hX` e `hY` correspondem aos filtros 1-D a serem usados nas direções x e y, respectivamente. Os demais parâmetros de entrada são completamente análogos àqueles da chamada da função `filter2D`. Naturalmente, a função `sepFilter2D` atende qualquer processo de filtragem, desde que o operador seja separável, de forma que uma classe muito mais geral de filtros pode ser usada, e não apenas Gaussianos.

3.3.4. *Diferenciadores de primeira ordem e magnitude do gradiente*

Filtro diferenciadores buscam aproximar, ou estimar, o valor da derivada do sinal em determinada direção. Sua função principal é a detecção de bordas, o que os põe em posição antagônica aos filtros-caixa e Gaussiano, já que estes últimos levam a uma suavização generalizada nas imagens. Em geral, diferenciadores correspondem a filtros passa-altas, privilegiando altas frequências em detrimento das baixas frequências. Há uma profusão de filtros diferenciadores em processamento digital de imagens [7, 15], de forma que trataremos dos mais usuais: Prewitt, Sobel e Laplaciano.

De modo geral, os diferenciadores variam entre si devido aos coeficientes usados na expansão em diferenças finitas dos sinais digitais. Por exemplo, a diferenças finitas progressiva, regressiva e centrada na direção vertical de uma imagem são dadas respectivamente por [15]

$$\begin{aligned} \Delta_+[f_{ij}] &= f_{i+1,j} - f_{ij}, \\ \Delta_-[f_{ij}] &= f_{i,j} - f_{i-1,j}, \\ \Delta_c[f_{ij}] &= \frac{1}{2}(f_{i+1,j} - f_{i-1,j}). \end{aligned} \tag{27}$$

As aproximações para as derivadas na direção horizontal são semelhantes, bastando-se operar sobre o índice j. Observando os coeficientes das diferenças na Equação (27) e do arranjo de vizinhança N_4 da Equação (16) chega-se a uma forma matricial para os diferenciadores

$$\Delta_+ = \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}, \Delta_- = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix} \text{ e } \Delta_c = \begin{bmatrix} -0.5 \\ 0 \\ 0.5 \end{bmatrix}. \tag{28}$$

Para operar sobre colunas, basta tomar as transpostas das matrizes da Equação (28).

Conforme dito no início da seção, a variação entre os diferenciadores se dá principalmente no cômputo dos coeficientes do filtros e no papel dos elementos vizinhos no resultado final. Os diferenciadores apresentados na Equação (28), por exemplo, consideram apenas vizinhança N_4. Dois dos principais diferenciadores mais usados em processamento de imagens levam em conta o papel dos vizinhos das diagonais na aproximação das derivadas. São eles os filtros de Prewitt e Sobel [7, 15], dados na direção horizontal respectivamente por

$$P_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \text{ e } S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}. \tag{29}$$

Naturalmente, $P_y = P_x^{\mathrm{T}}$ e $S_y = S_x^{\mathrm{T}}$.

Uma aplicação não-linear envolvendo diferenciadores consiste em se gerar uma imagem das bordas de uma imagem, aproximando-as pela magnitude do gradiente. Isto é, dada uma imagem de entrada $f(x,y)$, busca-se a imagem

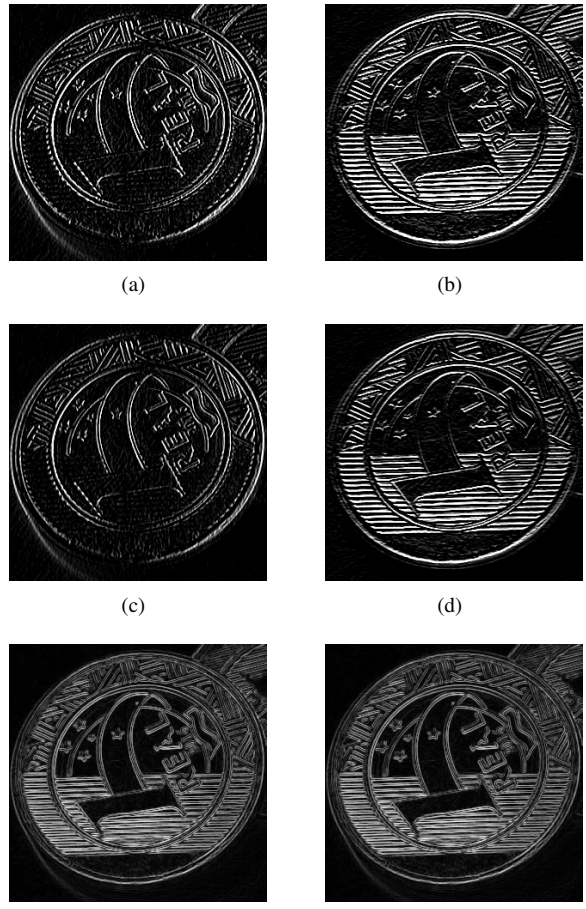


Figura 11: Ilustração de uso dos diferenciadores de Sobel e Prewitt sobre a imagem da Figura 2-(b). Em (a) e (b) são mostradas as bordas verticais e horizontais detectadas pelo filtro de Sobel, e em (c) e (d) são mostradas as respectivas bordas detectadas via filtro de Prewitt. Em (e) e (f) são mostradas as aproximações para a magnitude do gradiente via Sobel e Prewitt, respectivamente. Para melhor visualização, estas duas imagens foram submetidas a um alargmamento linear de contraste.

dada por

$$\|\vec{\nabla} f(x,y)\| = \sqrt{\left(\frac{\partial}{\partial x} f(x,y)\right)^2 + \left(\frac{\partial}{\partial y} f(x,y)\right)^2}. \tag{30}$$

A implementação discreta da Equação (30) é imediata, embora dependa do diferenciador usado. O resultado geral obtido é uma imagem com as bordas originais evidenciadas. A Figura 11 mostra a obtenção das bordas em ambas as direções sobre a Figura 2-(b) por meio dos filtros de Sobel e Prewitt, além de mostrar a aproximação de $\|\vec{\nabla} f\|$ para estes dois filtros. O Código 16 implementa estes processos.Não há muitas novidades em sua programação, exceto pelo fato de que foi feito uso pela primeira vez – na linhas 52 e 71 – o uso do operador `cv::sqrt`, cuja sintaxe

```
sqrt(Mat x, Mat y)
```

atribui a `y` uma matriz cujas entradas correspondem às raízes

quadradas das entradas da matriz x.

```
#include <opencv2/opencv.hpp>
#include <iostream>

using namespace std;
using namespace cv;

int main(int argc, char** argv){
  Mat x;
  Mat dxSobel, dySobel, hSobel;
  Mat dxPrewitt, dyPrewitt, hPrewitt;
  Mat magGradSobel, magGradPrewitt;

  double extrEscala[2] = {0.0, 0.0};
  double fatorEscala = 1.0 / 255.0;
  double a;

  double entrHSobel[3][3] = {
    {-1.0, 0.0, 1.0},
    {-2.0, 0.0, 2.0},
    {-1.0, 0.0, 1.0}};

  double entrHPrewitt[3][3] = {
    {-1.0, 0.0, 1.0},
    {-1.0, 0.0, 1.0},
    {-1.0, 0.0, 1.0}};

  x = imread(argv[1], 0);
  x.convertTo(x, CV_64FC1, fatorEscala, 0.0);

  // definicao dos filtros
  hSobel= Mat(3, 3, CV_64FC1, entrHSobel);

  hPrewitt = Mat(3, 3, CV_64FC1, entrHPrewitt);

  // declaracao das matrizes dos magnitudes de
     gradiente
  magGradSobel = Mat(x.size(), CV_64FC1);

  magGradPrewitt = Mat(x.size(), CV_64FC1);

  // inicio dos processos de diferenciacao por Sobel

  flip(hSobel, hSobel, -1);
  flip(hPrewitt, hPrewitt, -1);

  filter2D(x, dxSobel, CV_64F, hSobel, Point(-1,-1),
     0.0, BORDER_REPLICATE);
  filter2D(x, dySobel, CV_64F, hSobel.t(), Point(-1,-1)
     , 0.0, BORDER_REPLICATE);

  // estimativa da magnitude do gradiente por Sobel

  magGradSobel = dxSobel.mul(dxSobel) + dySobel.mul(
     dySobel);

  sqrt(magGradSobel, magGradSobel);

  // Ajuste linear de contraste para escala toda

  minMaxLoc(magGradSobel, &extrEscala[0], &extrEscala
     [1]);

  a = 1.0 / (extrEscala[1] - extrEscala[0]);

  magGradSobel = a * (magGradSobel - extrEscala[0]);

  // inicio dos processos de diferenciacao por Prewitt

  filter2D(x, dxPrewitt, CV_64F, hPrewitt, Point(-1,-1)
     , 0.0, BORDER_REPLICATE);
  filter2D(x, dyPrewitt, CV_64F, hPrewitt.t(), Point
     (-1,-1), 0.0, BORDER_REPLICATE);

  // estimativa da magnitude do gradiente por Prewitt

  magGradPrewitt = dxPrewitt.mul(dxPrewitt) + dyPrewitt
     .mul(dyPrewitt);

  sqrt(magGradPrewitt, magGradPrewitt);

  minMaxLoc(magGradPrewitt, &extrEscala[0], &extrEscala
     [1]);

  a = 1.0 / (extrEscala[1] - extrEscala[0]);

  magGradPrewitt = a * (magGradPrewitt - extrEscala[0])
     ;

  // Visualizacao

  namedWindow("Imagem original", WINDOW_AUTOSIZE);
  namedWindow("Derivada em x - Sobel", WINDOW_AUTOSIZE)
     ;
  namedWindow("Derivada em y - Sobel", WINDOW_AUTOSIZE)
     ;
  namedWindow("Derivada em x - Prewitt",
     WINDOW_AUTOSIZE);
  namedWindow("Derivada em y - Prewitt",
     WINDOW_AUTOSIZE);
  namedWindow("Magnitude do gradiente - Sobel",
     WINDOW_AUTOSIZE);
  namedWindow("Magnitude do gradiente - Prewitt",
     WINDOW_AUTOSIZE);

  imshow("Imagem original", x);
  imshow("Derivada em x - Sobel", dxSobel);
  imshow("Derivada em y - Sobel", dySobel);
  imshow("Derivada em x - Prewitt", dxPrewitt);
  imshow("Derivada em y - Prewitt", dyPrewitt);
  imshow("Magnitude do gradiente - Sobel", magGradSobel
     );
  imshow("Magnitude do gradiente - Prewitt",
     magGradPrewitt);

  waitKey(0);
```

```
98
99  destroyAllWindows();
100
101 // Saida das imagens
102
103 dxSobel.convertTo(dxSobel, CV_8UC1, 255, 0);
104 dySobel.convertTo(dySobel, CV_8UC1, 255, 0);
105 magGradSobel.convertTo(magGradSobel, CV_8UC1, 255, 0)
      ;
106
107 dxPrewitt.convertTo(dxPrewitt, CV_8UC1, 255, 0);
108 dyPrewitt.convertTo(dyPrewitt, CV_8UC1, 255, 0);
109 magGradPrewitt.convertTo(magGradPrewitt, CV_8UC1,
      255, 0);
110
111 imwrite("dxSobel.jpg", dxSobel);
112 imwrite("dySobel.jpg", dySobel);
113 imwrite("magGradSobel.jpg", magGradSobel);
114 imwrite("dxPrewitt.jpg", dxPrewitt);
115 imwrite("dyPrewitt.jpg", dyPrewitt);
116 imwrite("magGradPrewitt.jpg", magGradPrewitt);
117
118 return 0;}
```

Código 16: Implementação de processo de filtragem pelos filtros de Sobel e Prewitt com posterior obtenção da magnitude do gradiente da imagem de entrada.

Para finalizar a discussão sobre diferenciadores de primeira ordem, cabe ressaltar que OpenCV dispõe de função específica para diferenciação via filtro Sobel. O processo é realizado por meio dafunção `cv::Sobel`, cuja sintaxe geral é da forma [10, 13]

```
Sobel(Mat       x,
      Mat       y,
      int       profBits,
      int       ordemDx,
      int       ordemDy,
      int       tamNucleo,
      double    fatorEscala,
      double    nivDC,
      int       extensaoBorda).
```

Os parâmetros `x`, `y`, `profBits`, `tamNucleo`, `fatorEscala`, `nivDC` desempenham papel idêntico às chamadas de `filter2D`, `boxFilter` etc. A novidade está nos inteiros `ordemDx` e `ordemDy`, que determinam a ordem da derivada em cada direção. Tal como definido na Equação (29), o filtro de Sobel contém derivadas de primeira ordem em x e y. O que OpenCV disponibiliza é a generalização para ordens quaisquer [10, 13]. Note que o inteiro `tamNucleo` deve ser ímpar e seu valor padrão é 3 [10].

3.3.5. *Laplaciano*

Tal como dito no final da seção anterior, diferenciadores de ordem superior podem ser utilizados em processamento de imagens. O filtro Laplaciano visa aproximar as derivadas segundas da imagem,

$$\nabla^2[f(x,y)] = \frac{\partial^2}{\partial x^2} f(x,y) + \frac{\partial^2}{\partial y^2} f(x,y). \tag{31}$$

Em se tratando de um operador de segunda ordem, o Laplaciano permite que pontos extremos locais sejam detectados. Uma abordagem clássica consiste em se analisar os pontos de troca de sinal do Laplaciano [7]. Uma abordagem mais simples consiste simplesmente em se usar o Laplaciano como um detector de bordas.

Analogamente aos diferenciadores de primeira ordem, a definição do Laplaciano pode variar bastante, dependendo dos esquemas de vizinhanças e de diferenças finitas usados. A forma explícita mais simples do Laplaciano é dada por

$$L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}. \tag{32}$$

e é obtida por meio da operação iterada de diferenças centradas [15].

A filtragem por Laplaciano pode ser operacionalizada através da definição explícita do filtro e posterior convolução por meio `cv::filter2D`.Note que a matriz L dada na Equação (32) é simétrica e, portanto, não necessita rebatimento para implementação da convolução. Uma alternativa mais simples e poderosa consiste em se utilizar a função `cv::Laplacian`, cuja sintaxe geral é da forma [10, 13]

```
Laplacian(Mat       x,
          Mat       y,
          int       profBits,
          int       tamNucleo,
          double    fatorEscala,
          double    nivDC,
          int       extensaoBorda).
```

Se `tamNucleo > 1`, a aproximação do Laplaciano se dá por filtragens sucessivas com operadores de Sobel. Caso `tamNucleo = 1`, a matriz dada pela Equação (32) é utilizada. A Figura 12 mostra o efeito do filtro Laplaciano sobre a imagem da Figura 2-(b). O Código 17 implementa este processo.

Figura 12: Filtragem por Laplaciano 3×3 sobre a imagem da Figura 2-(b).

```
1 #include <opencv2/opencv.hpp>
```

```
2 #include <iostream>
3
4 using namespace std;
5 using namespace cv;
6
7 int main(int argc, char** argv){
8   Mat x, y;
9
10  x = imread(argv[1], 0);
11
12  Laplacian(x, y, CV_8U, 1, 1.0, 0.0, BORDER_REFLECT);
13
14  namedWindow("Imagem original", WINDOW_AUTOSIZE);
15  namedWindow("Imagem processada", WINDOW_AUTOSIZE);
16
17  imshow("Imagem original", x);
18  imshow("Imagem processada", y);
19
20  waitKey(0);
21
22  destroyAllWindows();
23
24  imwrite(argv[2], y);
25
26  return 0;}
```

Código 17: Implementação de filtragem por Laplaciano via chamada da função `Laplacian`.

4. CONCLUSÕES

O objetivo deste trabalho foi de apresentar as ferramentas essenciais da biblioteca OpenCV para processamento de imagens. Tal como dito no início das Notas, não seria possível cobrir todo o conteúdo da ferramenta, de forma que os autores escolheram a fração absolutamente necessária para um percurso auto-suficiente na programação com OpenCV. Importante ressaltar que existem inúmeros outros pontos sobre análise e processamento de imagens e vídeo que OpenCV disponibiliza, tais como [10] morfologia matemática, calibração de campo visual em vídeo, análise e reconhecimento de padrões etc. De qualquer forma, os autores esperam que este trabalho possa expandir os horizontes dos leitores, tanto no que diz respeito ao uso dos milhares de recursos não cobertos no texto quanto na elaboração de aplicações inventivas e inovadoras.

Referências Bibliográficas

[1] AHO, A. V.; LAM M. S.; SETHI R.; ULLMAN J. D. **Compiladores, Princípios, Técnicas e Ferramentas**. 2 ed. São Paulo: Pearson. 2008.

[2] CMake Disponível em `https://cmake.org/download/`. Acesso em Abril/2017.

[3] Eclipse CDT. Disponível em `https://eclipse.org/cdt/downloads.php`. Acesso em Maio/2017.

[4] FFMPEG. Disponível em `https://ffmpeg.org/`. Acesso em Julho / 2017.

[5] GCC, the GNU Compiler Collection. Disponível em `https://gcc.gnu.org/`. Acesso em Julho / 2017.

[6] GOMES, J.; VELHO, L. **Computação gráfica: imagem**. 2 ed. Rio de Janeiro: IMPA. 2002.

[7] GONZALEZ, R.; WOODS, R. E. *Digital Image Processing*. 3 ed. New Jersey: Pearson. 2007.

[8] GTK+ Project, The. Disponível em `https://www.gtk.org/`. Acesso em Julho / 2017.

[9] Java. Disponível em `https://java.com/pt_BR/`. Acesso em Maio/2017.

[10] KAELHER, A.; BRADSKI, G. **Learning OpenCV: Computer Vision with the OpenCV Library**. 1 ed. California: O'Reilly Media. 2008.

[11] LAGANIÈRE R. **OpenCV 2 Computer Vision Application Programming Cookbook**. 1 ed. Packt Publishing Ltd. 2011.

[12] MinGW 32 e 64 bits. Disponível em `https://sourceforge.net/projects/mingw-w64/`. Acesso em Abril/2017.

[13] OpenCV Documentation. Disponível em `http://docs.opencv.org/master/index.html`.

[14] OpenCV Releases. Disponível em `http://opencv.org/releases.html`. Acesso em Abril/2017.

[15] PERSECHINO, A.; de ALBUQUERQUE, M. P. *Processamento de imagens: conceitos fundamentais*. **Monografias do CBPF**. v. 1. n. 4. pp. 1–41. 2015. Disponível em `http://revistas.cbpf.br/index.php/MO/index`. Acesso em Abril/2017.

[16] SAVITCH, W.; MOCK, K. **Absolute C++**. 5 ed. New Jersey: Pearson. 2012.

Notas Técnicas é uma publicação de trabalhos técnicos relevantes, das diferentes áreas da física e afins, e áreas interdisciplinares tais como: Química, Computação, Matemática Aplicada, Biblioteconomia, Eletrônica e Mecânica entre outras.

Cópias desta publicação podem ser obtidas diretamente na página web http://revistas.cbpf.br/index.php/nt ou por correspondência ao:

Centro Brasileiro de Pesquisas Físicas
Área de Publicações
Rua Dr. Xavier Sigaud, 150 – 4º andar
22290-180 – Rio de Janeiro, RJ
Brasil
E-mail: alinecd@cbpf.br/valeria@cbpf.br
http://portal.cbpf.br/publicacoes-do-cbpf

Notas Técnicas is a publication of relevant technical papers, from different areas of physics and related fields, and interdisciplinary areas such as Chemistry, Computer Science, Applied Mathematics, Library Science, Electronics and Mechanical Engineering among others.

Copies of these reports can be downloaded directly from the website http://notastecnicas.cbpf.br or requested by regular mail to:

Centro Brasileiro de Pesquisas Físicas
Área de Publicações
Rua Dr. Xavier Sigaud, 150 – 4º andar
22290-180 – Rio de Janeiro, RJ
Brazil
E-mail: alinecd@cbpf.br/valeria@cbpf.br
http://portal.cbpf.br/publicacoes-do-cbpf