Universidade do Estado do Amazonas (UEA)
Escola Superior de Tecnologia (EST)
Curso: Engenharia de Computação
Data: 4 de dezembro de 2019
Disciplina: Processamento Digital de Imagens
Professor: Ricardo Barbosa
Alunos:
Carlos Diego Ferreira de Almeida
Jean Phelipe de Oliveira Lima
João Victor Melo de Oliveira
Luiz Carlos Silva de Araújo Filho

Atividade Prática
Filtragem Espacial em Imagens

O desenvolvimento dos métodos de processamento digital de imagens são de grande importância por dois motivos: melhoram a informação visual para interpretação humana e aumentam o desempenho para armazenamento, transmissão e representação de imagens em computadores. Todos esses métodos são aplicados, hoje, a partir de linguagens de programação como `Python3` através de APIs como `OpenCV`. Esse trabalho visa abordar conceitos de PDI e de `OpenCV` para apresentar uma solução à atividade prática nele apresentada.

1 Fundamentação Teórica

Nesta seção são descritos os principais conceitos e técnicas que serviram de base para o desenvolvimento deste trabalho.

1.1 Processamento Digital de Imagens

O campo do processamento digital de imagens (PDI) se refere ao processamento de imagens digitais por um computador digital. Observe que uma imagem digital é composta de um número finito de elementos, cada um com localização e valor específicos. Esses elementos são chamados de elementos pictóricos, elementos de imagem, pels e *pixels*. *Pixel* é o termo mais utilizado para representar os elementos de uma imagem digital.

Uma imagem pode ser definida como uma função bidimensional, $f(x, y)$, em que $\mathbf{x}$ e $\mathbf{y}$ são coordenadas espaciais (no plano), e a amplitude de $\mathbf{f}$ em qualquer par de coordenadas (x, y) é chamada de intensidade ou nível de cinza da imagem nesse ponto. Quando $\mathbf{x}$, $\mathbf{y}$ e os valores de intensidade de $\mathbf{f}$ são quantidades finitas e discretas, chamamos de imagem digital.

Há três tipos de processos computacionais em PDI: processos de níveis baixo, médio e alto. Os processos de nível baixo envolvem operações primitivas, como o pré-processamento de imagens para reduzir o ruído, o realce de contraste e o aguçamento de imagens. Um processo de nível baixo é caracterizado pelo fato de tanto a entrada

quanto a saída serem imagens. O processamento de imagens de nível médio envolve tarefas como a segmentação (separação de uma imagem em regiões ou objetos), a descrição desses objetos para reduzi-los a uma forma adequada para o processamento computacional e a classificação (reconhecimento) de objetos individuais. Um processo de nível médio é caracterizado pelo fato de suas entradas, em geral, serem imagens, mas as saídas são atributos extraídos dessas imagens (isto é, bordas, contornos e a identidade de objetos individuais). Por fim, o processamento de nível alto envolve "dar sentido" a um conjunto de objetos reconhecidos, como na análise de imagens e, no extremo dessa linha contínua, realizar as funções cognitivas normalmente associadas à visão.

1.1.1 Passos Fundamentais em Processamento Digital de Imagens

A Figura 1 resume todas as metodologias que podem ser aplicadas em imagens para diferentes propósitos e objetivos. Para o escopo da atividade, são necessários apenas os quadros que são descritos nos próximos parágrafos.

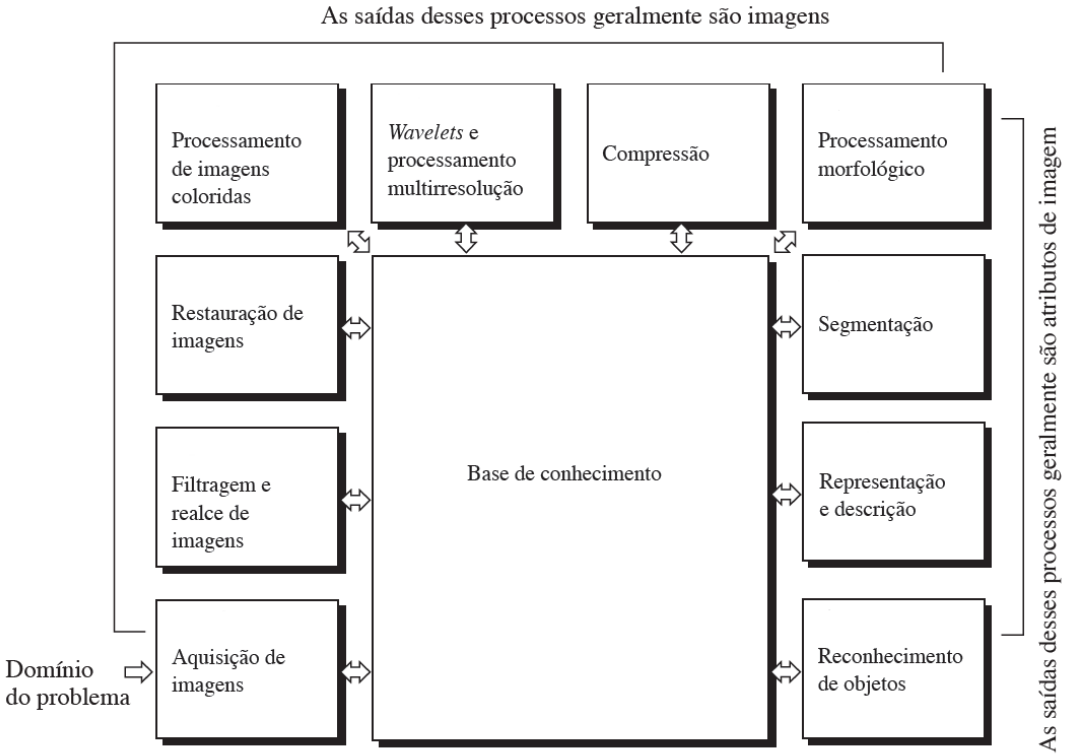

Figura 1: Passos fundamentais em PDI.

A **aquisição de imagens** é o primeiro processo da Figura 1. A aquisição pode ser tão simples quanto receber uma imagem que já esteja em formato digital ou através de um sensor de aquisição de imagem (ver seção 1.2.1). Em geral, o estágio de aquisição de imagens envolve um pré-processamento, por exemplo, o redimensionamento de imagens.

O **realce de imagens** é o processo de manipular uma imagem de forma que o resultado seja mais adequado do que o original para uma aplicação específica. A palavra *específica* é importante neste contexto, porque estabelece desde o início que as técnicas de realce são orientadas de acordo com o problema. Dessa forma, por exemplo, um método bastante útil para realçar imagens radiográfica pode não ser a melhor abordagem para realçar imagens de satélite capturadas na banda infravermelha do espectro eletromagnético.

1.1.2 Componentes de um Sistema de Processamento de Imagem

A Figura 2 mostra os componentes básicos que constituem um sistema de uso geral típico para o processamento digital de imagens. A função de cada componente é discutida nos próximos parágrafos.

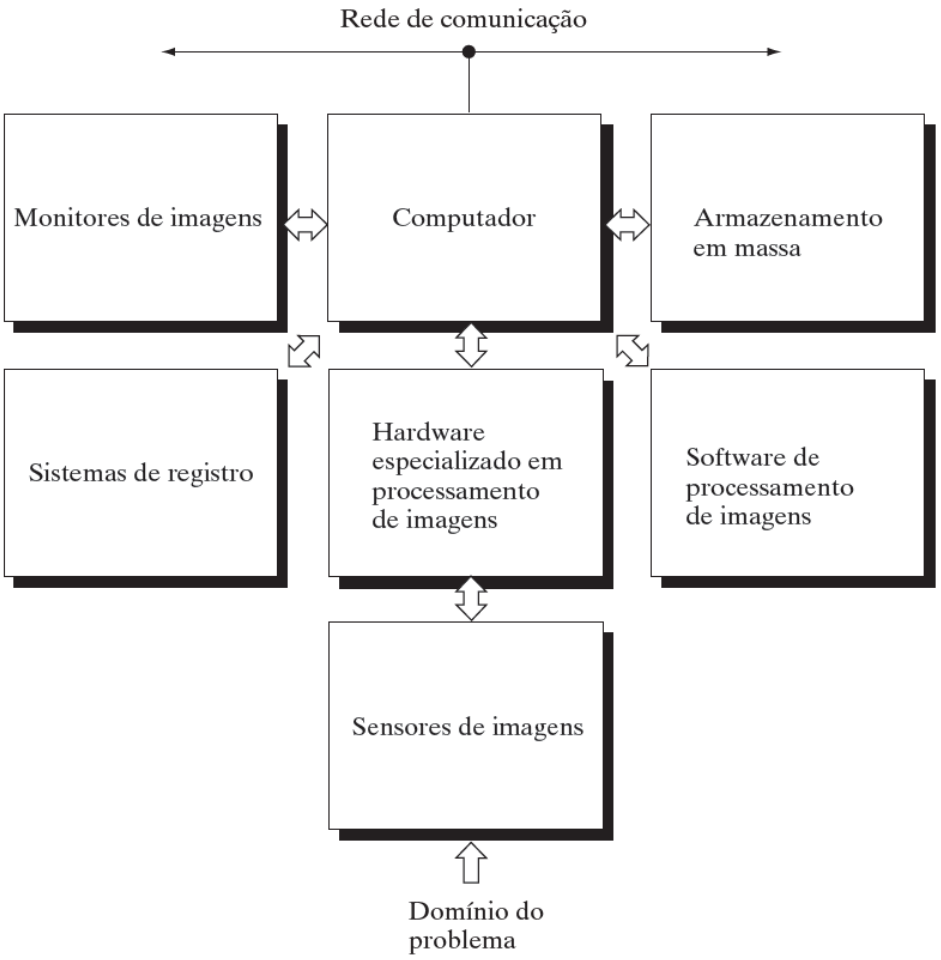

Figura 2: Componentes de um Sismate de PDI.

No que se refere ao **sensoriamento**, dois elementos são necessários para a aquisição de imagens digitais. O primeiro é um dispositivo físico sensível à energia irradiada pelo objeto cuja imagem desejamos capturar. O segundo, chamado de digitalizador, é um dispositivo utilizado para converter a saída do dispositivo físico de sensoriamento em formato digital. Por exemplo, em uma filmadora digital, os sensores produzem uma saída elétrica proporcional à intensidade da luz. O digitalizador converte essa saída em dados digitais.

O ***hardware* especializado** em processamento de imagens normalmente consiste no digitalizador mencionado acima, além de um hardware capaz de desempenhar outras operações primárias, como uma unidade lógica e aritmética (ALU, sigla em inglês), que realiza operações aritméticas e lógicas em paralelo em toda a imagem. Um exemplo de como uma ALU pode ser utilizada está no cálculo da média de uma imagem à medida que ela é digitalizada, com o propósito de redução de ruídos.

O **computador** em um sistema de processamento de imagens é um computador de uso geral, que pode variar e um computador pessoal a um supercomputador. Em aplicações especiais, algumas vezes computadores especializados são utilizados para atingir

o nível necessário de desempenho. Para realização da atividade prática, um sistemas de processamento de imagens de uso geral foi utilizado. Nesses sistemas, praticamente qualquer computador pessoal bem equipado é suficiente para as tarefas de processamento de imagens *offline*.

O ***software*** para o processamento de imagens consiste em módulos especializados que realizam tarefas específicas. Um bom pacote computacional também inclui a possibilidade de o usuário escrever códigos que, no mínimo, utilizem os módulos especializados. Os pacotes de aplicativos mais sofisticados permitem a integração desses módulos e dos comandos gerais de *software* a partir de pelo menos uma linguagem computacional. Para a realização da atividade prática, o pacote utilizado é o `OpenCV` e os códigos são escritos na linguagem de programação `Python3`.

A capacidade de **armazenamento em massa** é indispensável em aplicações de processamento de imagens. Uma imagem do tamanho de 1.024×1.024 *pixels*, na qual a intensidade de cada pixel requer 8 *bits*, necessita de um espaço de armazenamento de 1 *megabyte*, se a imagem não for comprimida. Ao lidar com milhares, ou até milhões, de imagens, o armazenamento adequado em um sistema de processamento de imagens pode ser um desafio.

Os *monitores de imagem* utilizados hoje em dia são, em sua maioria, monitores de TV em cores (preferencialmente de tela plana). Os monitores são controlados pelas placas de vídeo (gráficas ou de imagens), que são parte integral de um sistema computacional. Raramente os requisitos das aplicações de visualização de imagens não podem ser satisfeitos pelas placas de vídeo disponíveis comercialmente como parte do sistema computacional.

A *rede de comunicação* é quase um componente padrão de qualquer sistema computacional em uso hoje em dia. Em razão do grande volume de dados inerente às aplicações de processamento de imagens, a principal preocupação na transmissão de imagens é a largura de banda. Em redes dedicadas, isso normalmente não constitui um problema, mas as comunicações com sites remotos pela Internet nem sempre são eficientes. Felizmente, essa situação está melhorando rapidamente como resultado do advento da fibra óptica e de outras tecnologias de banda larga.

1.2 Fundamentos da Imagem Digital

Esta subseção descreve alguns conceitos relacionados a imagem digital que são aplicados na atividade prática.

1.2.1 Sensores de Aquisição de Imagens

A luz é uma onda eletromagnética onde a energia da iluminação é refletida pelos objetos ou transmitida através deles. Os sensores tem a funcionalidade básica de transformar a energia de iluminação em imagens digitais. A ideia é simples: a energia que entra é transformada em tensão pela combinação da energia elétrica de entrada e do material do sensor, sensível a um tipo específico de energia que está sendo detectado. A forma de onda da tensão de saída é a resposta do(s) sensor(es), e uma quantidade digital é obtida de cada sensor por meio da digitalização de sua resposta.

Há diversos tipos de sensores na literatura e na indústria. Um tipo bastante utilizado é o sensor de arranjo matricial. Esse é o arranjo predominante encontrado nas câmeras digitais. Um sensor típico para essas câmeras é uma matriz CCD, que pode ser fabricada com uma grande variedade de propriedades sensoras e dispostas em arranjos matriciais de 4.000×4.000 elementos ou mais. A resposta de cada sensor é proporcional à integral da

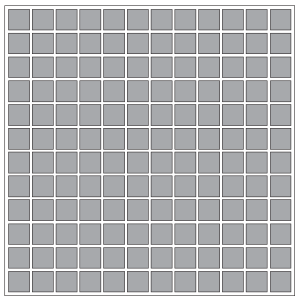

Figura 3: Sensor de área matricial.

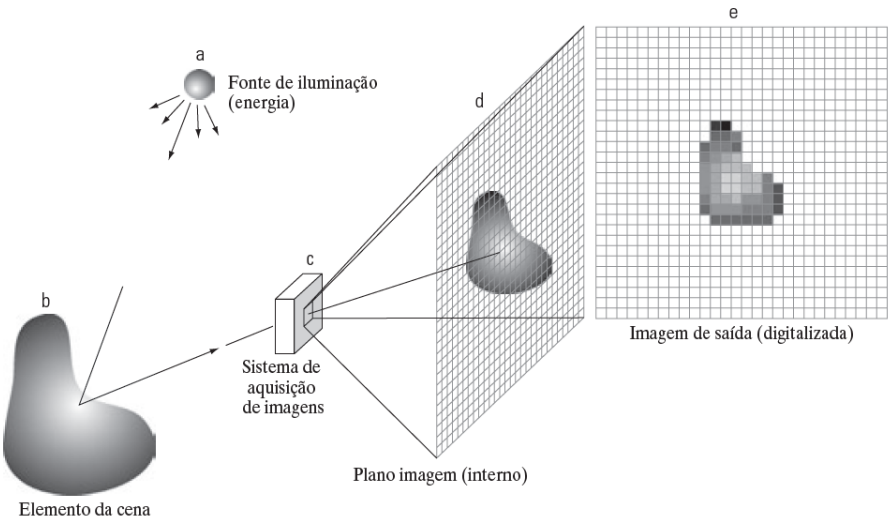

Figura 4: Exemplo do processo de aquisição de imagem.

energia luminosa projetada sobre a superfície do sensor. Pelo fato de a matriz de sensores da Figura 3 ser bidimensional, sua principal vantagem é que uma imagem completa pode ser obtida projetando o padrão de energia na superfície da matriz.

A principal forma na qual os sensores matriciais são utilizados é mostrada na Figura 4. Essa figura mostra a energia de uma fonte de iluminação sendo refletida de um elemento de uma cena. A primeira função realizada pelo sistema de aquisição de imagens da Figura 4(c) é coletar a energia de entrada e projetá-la em um plano imagem. Se a iluminação for luz, a entrada frontal do sistema de aquisição de imagens é uma lente ótica que projeta a cena vista sobre o plano focal da lente, como mostra a Figura 4(d). O arranjo de sensores, que coincide com o plano focal, produz saídas proporcionais à integral da luz recebida em cada sensor. Circuitos digitais e analógicos realizam uma varredura nessas saídas e as convertem em um sinal analógico, que é então digitalizado por um outro componente do sistema de aquisição de imagens. A saída é uma imagem digital, como mostra o esquema da 4(e).

1.2.2 Amostragem e Quantização de Imagens

A saída da maioria dos sensores consiste de uma forma de onda de tensão contínua cuja amplitude e o comportamento no espaço estão relacionados ao fenômeno físico que está sendo captado pelos sensores. Para criar uma imagem digital a partir de dados capturados por sensores precisamos converter os dados contínuos que foram captados para o formato digital. Isso envolve dois processos: amostragem e quantização.

A ideia básica por trás da *amostragem* e da *quantização* é ilustrada na Figura 2.16. A Figura 2.16(a) mostra uma imagem contínua f que queremos converter em formato digital. Uma imagem pode ser contínua em relação às coordenadas x e y e também em relação à amplitude. Para convertê-la ao formato digital, temos de fazer a amostragem da função em ambas a coordenadas e na amplitude. A digitalização dos valores de coordenada é chamada de amostragem. A digitalização dos valores de amplitude é chamada de quantização.

A função unidimensional da Figura 5(b) é um gráfico que representa os valores de amplitude (nível de intensidade) da imagem contínua ao longo do segmento de reta AB

na Figura 5(a). As variações aleatórias se devem ao ruído da imagem. Para realizar a amostragem dessa função, colhemos amostras igualmente espaçadas ao longo da linha AB, como mostra a Figura 5(c). A posição de cada amostra no espaço é indicada por uma pequena marca vertical na parte inferior da figura. As amostras são representadas por pequenos quadrados brancos superpostos na função. O conjunto dessas localizações discretas nos dá a função de amostragem. No entanto, os valores das amostras ainda cobrem (verticalmente) uma faixa contínua de valores de intensidade. Para formar uma função digital, os valores de intensidade também devem ser convertidos (quantizados) em quantidades discretas. O lado direito da Figura 5(c) mostra a escala de intensidade dividida em oito intervalos discretos, variando do preto ao branco. As marcas verticais indicam o valor específico atribuído a cada um dos oito níveis de intensidade. Os níveis de intensidade contínuos são quantizados atribuindo um dos oito valores para cada amostra. Essa atribuição é feita dependendo da proximidade vertical de uma amostra a uma marca indica dora.

As amostras digitais resultantes da amostragem e da quantização são mostradas na 5(d). Ao começar na parte superior da imagem e realizar esse procedimento linha por linha, produz-se uma imagem digital bidimensional. Está implícito na 5 que, além do número discreto de níveis utilizados, a precisão atingida na quantização depende muito do conteúdo de ruído do sinal da amostragem.

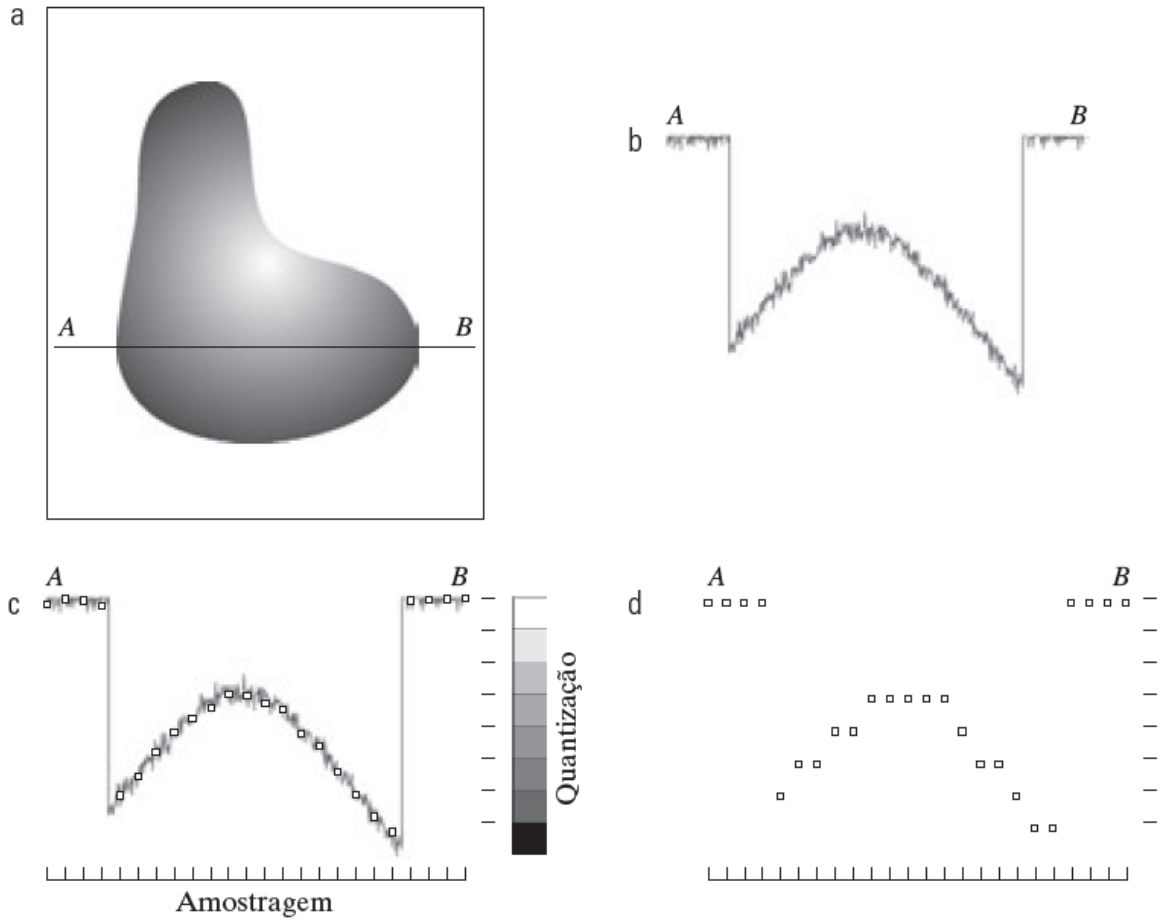

Figura 5: Processo de produção uma imagem digital.

1.2.3 Representação de Imagens Digitais

Seja $f(s,t)$ uma função de imagem contínua de duas variáveis contínuas, **s** e **t**. Convertemos essa função em uma imagem digital por meio de amostragem e quantização. Suponha que realizemos a amostragem da imagem contínua em uma matriz 2D, $f(x,y)$, contendo **M** linhas e **N** colunas, sendo que (x,y) são coordenadas discretas. Utilizamos números inteiros para essas coordenadas discretas: $x = 0, 1, 2, \ldots, M-1$ e $y = 0, 1, 2, \ldots, N-1$. Em geral, o valor da imagem em quaisquer coordenadas (x,y) é expresso por $f(x,y)$, onde **x** e **y** são números inteiros.

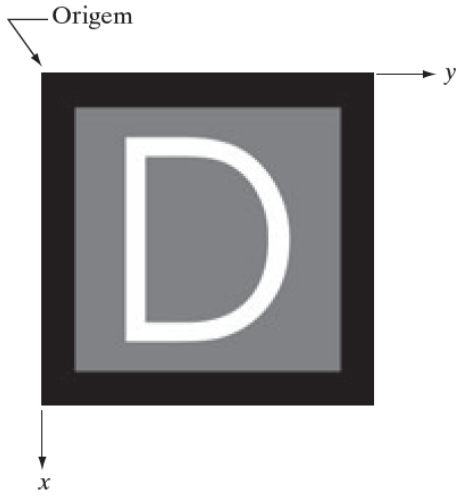

Figura 6: Imagem representada como uma matriz de intensidade visual.

A representação da Figura 6 é a mais comum representação de imagem digital. Ela mostra $f(x,y)$ como uma imagem que seria visualizada em um monitor ou uma fotografia. Aqui, o nível de cinza de cada ponto é proporcional ao valor da intensidade **f** desse ponto. Nessa figura, temos apenas três valores de intensidade igualmente espaçados. Se a intensidade for normalizada para o intervalo [0, 1], cada ponto da imagem tem o valor 0, 0,5 ou 1.

Matrizes numéricas são utilizadas para o processamento de imagens digitais e desenvolvimento de algoritmos que as manipulam. Na forma de equação, escrevemos a representação de uma matriz numérica $M \times N$ como

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & f(0,2) & \ldots & f(0,N-1) \\ f(1,0) & f(1,1) & f(1,2) & \ldots & f(1,N-1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ f(M-1,0) & f(M-1,1) & f(M-1,2) & \ldots & f(M-1,N-1) \end{bmatrix}$$

Os dois lados dessa equação são formas equivalentes de representar quantitativamente uma imagem digital. O lado direito é uma matriz de números reais. Cada ele mento dessa matriz é um *pixel*.

Em algumas discussões é vantajoso utilizar uma notação matricial mais tradicional para expressar uma imagem digital e seus elementos:

$$X = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,N-1} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{M-1,0} & x_{M-1,1} & x_{M-1,2} & \cdots & x_{M-1,N-1} \end{bmatrix}$$

As matrizes f e X são equivalentes, já que $a_{ij} = f(x = i, y = j) = f(i, j)$.

Podemos formatar as matrizes acima para representar uma imagem em um vetor coluna de tamanho $MN \times 1$. Esse processo é frequentemente necessário quando estamos trabalhando com imagens em linguagens de programação.

Observe na 6 que a origem de uma imagem digital se localiza na parte superior esquerda, com o eixo x positivo se estendendo para baixo e o eixo y positivo se estendendo para a direita. Isso se deve ao modo de como imagens são renderizadas em monitores, porém mais importante que isso é o fato de que o primeiro elemento de uma matriz é, por convenção, o elemento do canto superior esquerdo, de forma que a escolha da origem de $f(x, y)$ nesse ponto faz sentido matematicamente.

1.2.4 Resolução Espacial

A resolução espacial é uma medida do menor detalhe discernível em uma imagem. Quantitativamente, a resolução espacial pode ser expressa em várias formas, sendo que as mais comuns são pares de linha por unidade de distância e *pixels* por unidade de distância.

Para serem significativas, as medidas de resolução espacial de vem ser expressas com relação a unidades espaciais. O tamanho da imagem por si só não diz tudo. Dizer que uma imagem tem, digamos, uma resolução de 1.024×1.024 *pixels* não faz muito sentido se as dimensões espaciais da imagem não forem especificada.

1.3 Filtragem Espacial

A filtragem espacial é um dos conjunto de ferramentas mais utilizadas e importantes do ramo de processamento de digital de imagens. O termo *filtragem* se refere a aceitar ou rejeitar certos componentes de frequência (o termo é emprestado do processamento no domínio da frequência). Por exemplo, um filtro passa-baixas é aquele que aceita apenas baixas frequências. O efeito final de um filtro de imagem passa-baixas é borrar (suavizar) a imagem.

1.3.1 Funcionamento da Filtragem Espacial

Na Figura 7, é mostrado que um filtro espacial consiste em (1) uma vizinhança (normalmente um pequeno retângulo), (2) uma operação predefinida realizada sobre os *pixels* da imagem incluídos na vizinhança. A filtragem cria um novo pixel com coordenadas iguais às coordenadas do centro da vizinhança, e cujo valor é o resultado da operação de filtragem. Uma imagem processada (filtrada) é gerada à medida que o centro do filtro percorre cada *pixel* na imagem de entrada. Se a operação realizada sobre os *pixels* da imagem for linear, o filtro é chamado de filtro espacial linear (ou máscara, ou janela). Caso contrário, o filtro é não linear. Para realizar a atividade, foram utilizados apenas filtros lineares.

A Figura 8 ilustra o funcionamento da filtragem espacial linear utilizando uma vizinhança 3×3. Em qualquer ponto $(\mathbf{x}, \mathbf{y})$ da imagem, a resposta, $g(\mathbf{x}, \mathbf{y})$, do filtro é a soma

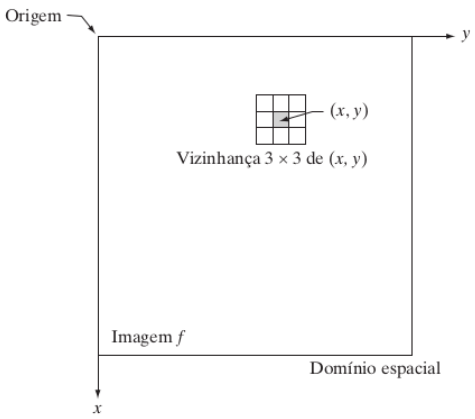

Figura 7: Uma vizinhança 3 × 3 ao redor de um ponto (x, y) em uma imagem no domínio espacial.

dos produtos dos coeficientes do filtro com os *pixels* da imagem englobados pelo filtro:

$$g(x,y) = w(-1,-1)f(x-1,y-1) + w(-1,0)f(x-1,y) + \cdots + \\ w(0,0)f(x,y) + \cdots + w(1,1)f(x+1,y+1)$$

Observe que o coeficiente central do filtro, $w(0,0)$, se alinha com o *pixel* da posição $(\mathbf{x},\mathbf{y})$. Para um tamanho de máscara $m \times n$, consideramos que $m = 2a+1$ e $n = 2b+1$, sendo a e b números inteiros positivos. Isso significa que nosso foco na discussão a seguir será em filtros de tamanho ímpar, e o menor é de tamanho 3×3. Em geral, a filtragem espacial linear de uma imagem de dimensões $M \times N$ com um filtro de dimensões $m \times n$ é dada pela expressão:

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

onde $\mathbf{x}$ e $\mathbf{y}$ variam de forma que cada *pixel* em w percorre todos os *pixels* em f.

1.3.2 Correlação e Convolução Espaciais

Correlação é o processo de mover uma máscara pela imagem e calcular a soma dos produtos em cada posição. O funcionamento da convolução é o mesmo, exceto o fato de o primeiro filtro ser rotacionado a 180°.

Para um filtro de dimensões $m \times n$, preenchemos com zeros a imagem com um mínimo de $m-1$ linhas acima e abaixo e $n-1$ colunas à esquerda e à direita. Neste caso, m e n são iguais a 3, de forma que preenchemos f com duas linhas de 0s acima e abaixo e duas colunas de 0s à esquerda e direita, como mostra a Figura 9(b). A Figura 9(c) mostra a posição inicial da máscara para realizar a correlação, e a Figura 9(d) mostra o resultado da correlação completa. A Figura 9(e) mostra o resultado correspondente após o recorte. Observe mais uma vez que o resultado é rotacionado a 180°. Para a convolução, pré-rotacionamos a máscara como antes e repetimos a operação de soma dos produtos de cada deslocamento como acabamos de explicar. As Figuras 9(f) a (h) mostram o resultado. Vemos novamente que a convolução de uma função com um impulso copia a função na posição do impulso. Deve ficar claro que, se a máscara for simétrica, a correlação e a convolução geram o mesmo resultado.

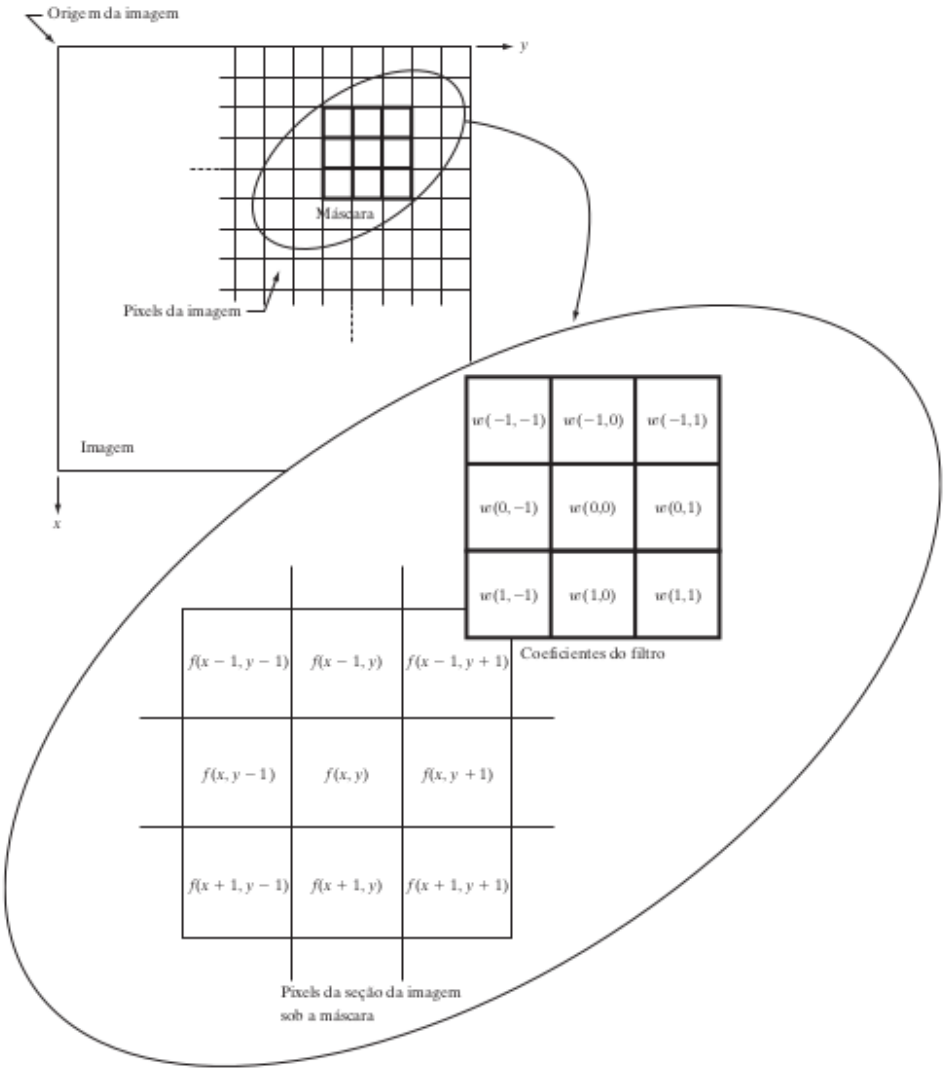

Figura 8: O funcionamento da filtragem espacial linear utilizando uma máscara 3×3.

Em termos matemáticos, temos que a correlação de um filtro $w(x, y)$ de tamanho $m \times n$ com uma imagem $f(x, y)$, é expressa como

$$w(x, y) \diamond f(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x + s, y + t)$$

Essa equação é calculada para todos os valores das variáveis de deslocamento x e y, de forma que todos os elementos de w percorram cada um dos *pixels* de f. Além disso, temos que $a = (m - 1)/2$, $b = (n - 1)/2$ e considera-se, para praticidade de notação, que m e n são números inteiros ímpares.

De forma similar, a convolução de $w(x, y)$ e $f(x, y)$, é expressa por

$$w(x, y) \star f(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x - s, y - t)$$

onde o sinal de menos à direita inverte f (isto é, a rotaciona a 180°). Inverter e deslocar f em vez de w são processos realizados para fins de simplicidade de notação e também

para seguir a forma convencional. O resultado é o mesmo. Como no caso da correlação, essa equação é calculada para todos os valores das variáveis de deslocamento x e y, de forma que todos os elementos de w percorram cada um dos os *pixels* de f apropriadamente preenchidos.

Utilizar a correlação ou a convolução para realizar a filtragem espacial é uma questão de preferência. Ambas as operação são capazes de realizar a função uma da outra se rotacionarmos os filtros. Para a atividade, decidimos utilizar convolução.

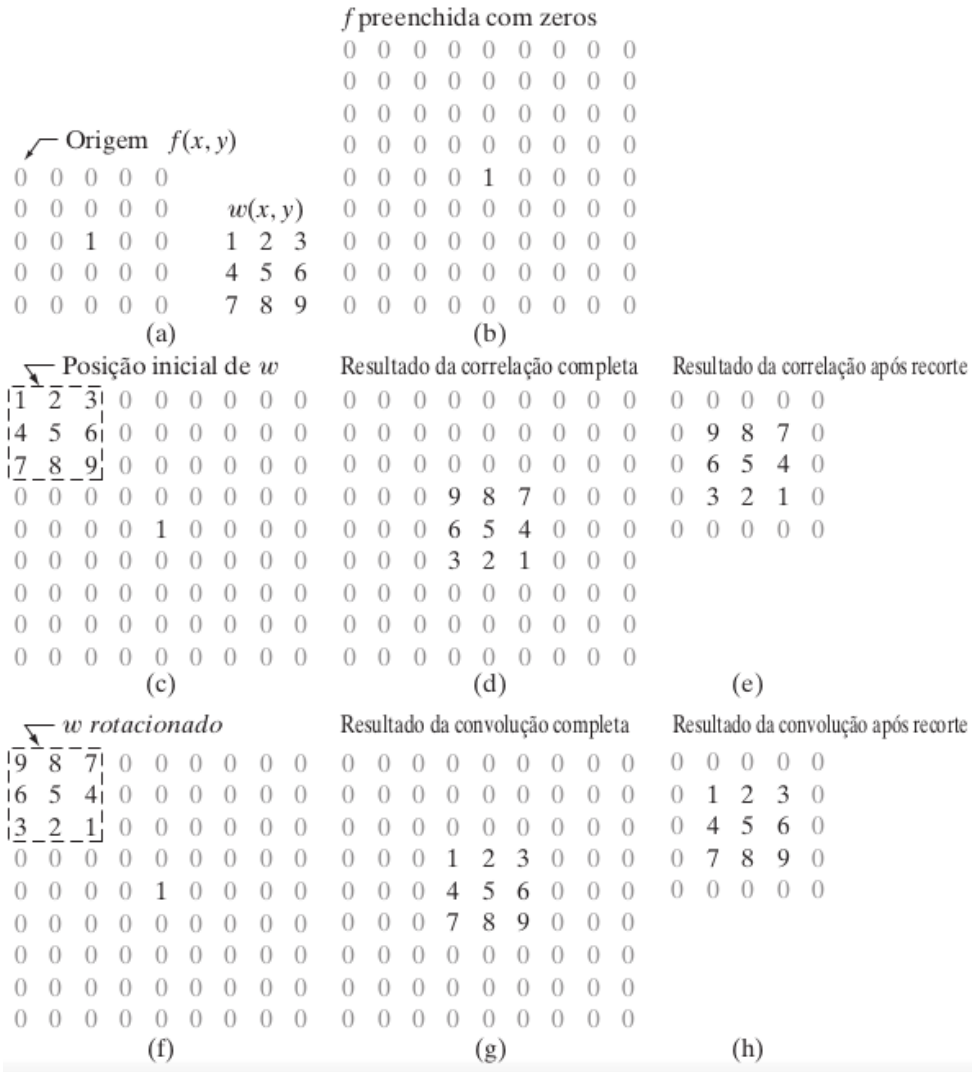

Figura 9: Correlação (linha do meio) e convolução (última linha) de um filtro 2D com um impulso unitário discreto 2D.

1.4 Filtros Lineares de Suavização

Os filtros de suavização são utilizados para borramento e redução de ruído. O borramento é aplicado em tarefas de pré-processamento, como remoção de pequenos detalhes da imagem antes da extração de objetos (grandes) e conexão de pequenas descontinuidades em linhas ou curvas.

A saída (resposta) de um filtro espacial linear de suavização é simplesmente a média dos *pixels* contidos na vizinhança da máscara de filtragem. Esses filtros por vezes são

chamados de filtros de média (ou filtros passa-baixa).

A ideia por trás dos filtros de suavização é direta. Ao substituir o valor de cada *pixel* de uma imagem pela média dos níveis de intensidade da vizinhança definida pela máscara, o processo resulta em uma imagem com perda da nitidez, ou seja, com redução das transições "abruptas"nas intensidades. Pelo fato de o ruído aleatório normalmente consistir em transições abruptas nos níveis de intensidade, a aplicação mais evidente da suavização é a redução de ruído. No entanto, as bordas (que quase sempre são características desejáveis de uma imagem) também são caracterizadas por transições abruptas de intensidade, de forma que os filtros de média apresentam o efeito colateral indesejável de borrar as bordas.

A Figura 10 mostra um filtro de suavização 3×3. A utilização do filtro gera a média aritmética simples dos *pixels* cobertos pela máscara. Ao final do processo de filtragem, toda a imagem é dividida por 9, ou seja, é feita a média dos níveis de intensidade dos pixels na vizinhança. Uma máscara $m \times n$ teria uma constante de normalização igual a $1/mn$.

$$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Figura 10: Uma máscara 3×3 (de média) para suavização.

1.5 Detector de Bordas de Canny

Carry Edge Detector ou detector de bordas Canny é um método que se utiliza de outras técnicas, como o Sobel, e realiza multiplos passos para chegar ao resultado final. Basicamente o Canny envolve: Aplicar um filtro para suavizar a imagem e remover o ruído; Encontrar os gradientes de intensidade da imagem; Aplicar Sobel duplo para determinar bordas potenciais; Aplicar o processo de "*hysteresis*" para verificar se o pixel faz parte de uma borda "forte" suprimindo todas as outras bordas que são fracas e não conectadas a bordas fortes.

Utilizando a biblioteca `OpenCV`, é preciso fornecer dois parâmetros para a função `cv2.Canny()`. Esses dois valores são o limiar 1 e limiar 2 e são utilizados no processo de "*hysteresis*" final. Qualquer gradiente com valor maior que o limiar 2 é considerado como borda. Qualquer valor inferior ao limiar 1 não é considerado borda. Valores entre o limiar 1 e limiar 2 são classificados como bordas ou não bordas com base em como eles estão conectados.

2 OpenCV

`OpenCV` (Open Source Computer Vision) é uma biblioteca de visão computacional e *machine learning* de programação de código aberto com o objetivo de tornar o processamento de imagens mais acessível a desenvolvedores e hobistas e pode ser utilizada em diversas linguagens de programação (C++, Python, Java e MATLAB).

A biblioteca possui mais de 2500 algoritmos otimizados, dos quais incluem um compreensivo conjunto de algoritmos clássicos e de estado-da-arte de visão computacional e *machine learning*. Esses algoritmos podem ser usados para detectar e reconhecer rostos, identificar objetos, classificar ações humanas em vídeos, rastrear movimentos de objetos, juntar imagens para produzir uma imagem de alta resolução de uma cena inteira, encontrar imagens similares em um conjunto de imagens, etc. `OpenCV` tem mais de 47 mil de usuários ativos na comunidade e mais de 18 milhões de *downloads*. A biblioteca é utilizada extensivamente por *hobistas*, companhias, grupos de pesquisadores e governos.

`OpenCV-Python` é a biblioteca em questão compatível para Python. Ela faz uso da biblioteca de operações numéricas otimizadas do Python, a `Numpy`. É possível escrever códigos em C++ dentro de um código Python utilizando *wrappers*. Dessa maneira, essa biblioteca é um *wrapper* da implementação original da `OpenCV-C++`.

A atividade prática deste trabalho foi realizada utilizando as abstrações e comodidades oferecidas pela `OpenCV-Python`.

3 Atividade e Metodologia

A atividade consiste em escolher uma imagem ao gosto do aluno e filtrá-la com intuito de suavizar características da mesma. A imagem original e as filtradas devem ser apresentadas como resultados.

Foi decidido realizar um processamento digital em uma imagem microscópica de uma amostra de sangue contendo células contaminadas por malária. Para a realização de um sistema automático de detecção de células contaminadas por malária é necessário que haja primeiro uma detecção de células, isto é, identificar a presença de células na imagem. Para isso, um processamento deve ser feito para explicitar as características necessárias para detecção de célula, assim, este trabalho buscou processar a imagem com um detector de bordas. Utilizou-se entao o uma suavização por média seguida de um detetor de bordas de Canny. A suavização foi realizada utilizando a biblioteca `OpenCV` e também utilizando uma função implementada pelos membros da equipe. Os resultados serão apresentados na Seção 4.

Primeiramente, a atividade foi dividia por integrante. O programa foi desenvolvido em máquinas relativamente modernas dos próprios alunos, em ambiente *Windows* ou *MacOS*, e após instalação de pacotes necessários e solução de problemas do processo de instalação. Em seguida, cada um estudou individualmente uma ou mais maneiras de resolver o problema. Majoritariamente, o procedimento de desenvolvimento consistiu em leitura de artigos pela *web* e visualização de vídeo-aulas em plataformas *online*. Após solucionar o problema, a equipe se reuniu e discutiu os códigos-fontes.

Os códigos da equipe foram desenvolvidos todos em Python por questões de comodidade. Apesar da existência de rotinas prontas para transformações de intensidades na `OpenCV`, os integrantes desenvolveram suas próprias rotinas baseadas no conteúdo do livro-texto, conforme solicitado pelo professor.

Por fim, a equipe decidiu juntar as melhores partes dos códigos desenvolvidos para apresentar como código-fonte final.

4 Resultados

Os programas escritos para realizar a atividade são apresentados nos blocos de código abaixo. No primeiro código-fonte, a equipe mostra como realizar o processo de filtragem utilizando suavização por média e detector de bordar *Canny* utilizando rotinas da biblioteca `OpenCV`.

```
import numpy as np
import cv2


img = cv2.imread('1 - malaria-original.jpg', 0) # leitura da
   imagem em escala de cinza
cv2.imwrite('2 - grayscale.jpg', img)

image = img.copy()


image = cv2.blur(image,(13,13)) # Aplicacao de filtro de
   suavizao por mdia (opencv)
cv2.imwrite('3.1 - mean-blur.jpg', image)
image = cv2.Canny(image, 10,16) # detector de bordas atravs do
   algoritmo de Canny
cv2.imwrite('4.1 - canny-edge-detector.jpg', image)
```

No segundo código, realizamos a mesma tarefa, porém com rotinas implementadas pela equipe.

```
import numpy as np
import cv2


#abre a imagem e converte para escala em cinza
def convolve(image, kernel):
    output = np.zeros(image.shape)
    image_row, image_col = image.shape
   #determina tamanho do filtro
    filter_row, filter_col = kernel.shape
   #variaveis para padding
    pad_height = int((filter_row - 1) / 2)
    pad_width = int((filter_col - 1) / 2)
   #coloca 0 nas extremidades da matriz
    padded_image = np.zeros((image_row + (2 * pad_height),
       image_col + (2 * pad_width)))
```

```
    padded_image[pad_height:padded_image.shape[0] - pad_height,
       pad_width:padded_image.shape[1] - pad_width] = image

   #convolucao
    for row in range(image_row):
       for col in range(image_col):
          output[row, col] = np.sum(kernel *
             padded_image[row:row + filter_row, col:col +
             filter_col])
    return output

def blur(image, filter_size = 9):
    #cria o filtro do blur
    kernel =
       np.ones((filter_size,filter_size),np.float32)/(filter_size*
       filter_size)

    return convolve(image, kernel)

#salva imagem apos aplicacao do filtro
def sobel_filters(img):
    Kx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]],
       np.float32)
    Ky = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]],
       np.float32)

    Ix = convolve(img, Kx)
    Iy = convolve(img, Ky)

    G = np.hypot(Ix, Iy)
    G = G / G.max() * 255
    theta = np.arctan2(Iy, Ix)

    return (G, theta)

def non_max_suppression(img, D):
    M, N = img.shape
    Z = np.zeros((M,N), dtype=np.int32)
    angle = D * 180. / np.pi
    angle[angle < 0] += 180

    for i in range(1,M-1):
       for j in range(1,N-1):
          try:
             q = 255
             r = 255

            #angle 0
```

```
            if (0 <= angle[i,j] < 22.5) or (157.5 <=
               angle[i,j] <= 180):
               q = img[i, j+1]
               r = img[i, j-1]
            #angle 45
            elif (22.5 <= angle[i,j] < 67.5):
               q = img[i+1, j-1]
               r = img[i-1, j+1]
            #angle 90
            elif (67.5 <= angle[i,j] < 112.5):
               q = img[i+1, j]
               r = img[i-1, j]
            #angle 135
            elif (112.5 <= angle[i,j] < 157.5):
               q = img[i-1, j-1]
               r = img[i+1, j+1]

            if (img[i,j] >= q) and (img[i,j] >= r):
               Z[i,j] = img[i,j]
            else:
               Z[i,j] = 0

         except IndexError as e:
            pass

   return Z

def threshold(img, lowThresholdRatio=0.05,
   highThresholdRatio=0.09):

   highThreshold = img.max() * highThresholdRatio;
   lowThreshold = highThreshold * lowThresholdRatio;

   M, N = img.shape
   res = np.zeros((M,N), dtype=np.int32)

   weak = np.int32(25)
   strong = np.int32(255)

   strong_i, strong_j = np.where(img >= highThreshold)
   zeros_i, zeros_j = np.where(img < lowThreshold)

   weak_i, weak_j = np.where((img <= highThreshold) & (img >=
      lowThreshold))

   res[strong_i, strong_j] = strong
   res[weak_i, weak_j] = weak
```

```
    return (res, weak, strong)

def hysteresis(img, weak, strong=255):
    M, N = img.shape
    for i in range(1, M-1):
        for j in range(1, N-1):
            if (img[i,j] == weak):
                try:
                    if ((img[i+1, j-1] == strong) or (img[i+1, j]
                        == strong) or (img[i+1, j+1] == strong)
                        or (img[i, j-1] == strong) or (img[i, j+1]
                            == strong)
                        or (img[i-1, j-1] == strong) or (img[i-1, j]
                            == strong) or (img[i-1, j+1] == strong)):
                        img[i, j] = strong
                    else:
                        img[i, j] = 0
                except IndexError as e:
                    pass
    return img



image = cv2.imread('1 - malaria-original.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
image = blur(image)
cv2.imwrite('output_blured_9x9.jpg', image)
gradientMat, thetaMat = sobel_filters(image)
cv2.imwrite('output_sobel.jpg', gradientMat)
nonMaxImg = non_max_suppression(gradientMat, thetaMat)
cv2.imwrite('output_nonMaxImg.jpg', nonMaxImg)
thresholdImg = threshold(nonMaxImg)
cv2.imwrite('output_thresholdImg.jpg', thresholdImg[0])
img_final = hysteresis(thresholdImg[0], 75)
cv2.imwrite('output9x9.jpg', img_final)
```

O processo realizado por ambos os códigos é ilustrado pela Figura 11, que apresenta, sequencialmente, os processamentos realizados nas imagens.

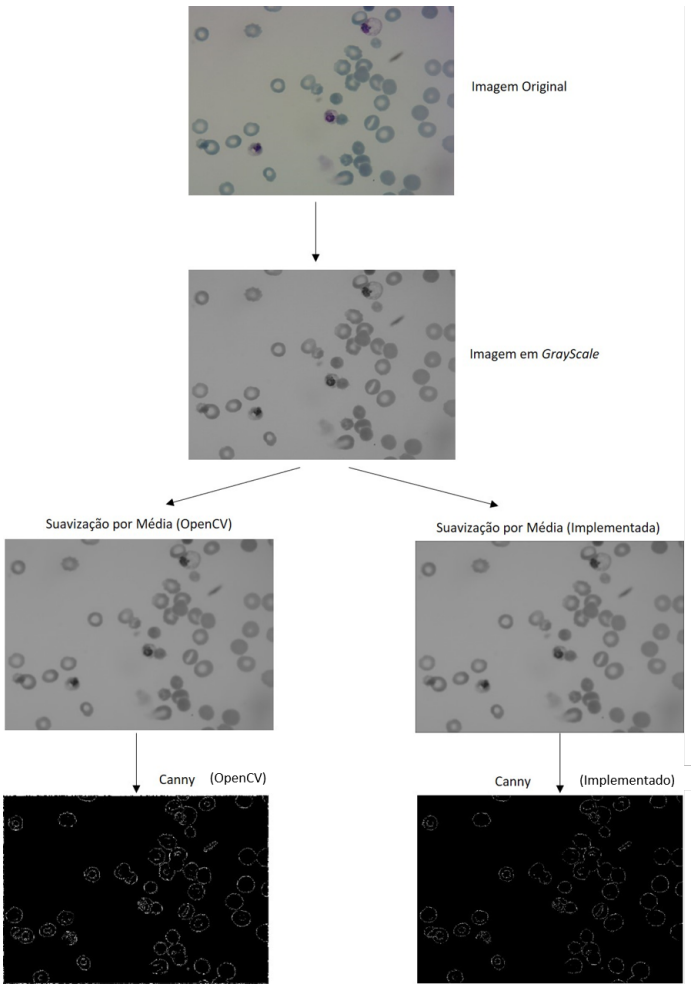

Figura 11: Passo a Passo do processamento da imagem.

Referências

[1] GONZALEZ, R. C.; WOODS, R. E. *Processamento Digital de Imagens*. 3. ed. São Paulo - SP, Brasil: Pearson Education do Brasil, 2010.

[2] Merlon de Alencar Rocha. *OpenCV - Uma breve introdução*. 2018. Disponível em `https://blog.cedrotech.com/opencv-uma-breve-introducao-visao-computacional-com-python/`. Acesso em 4 de dezembro de 2019.

[3] OpenCV Team. *About OpenCV*. 2019. Disponível em `https://opencv.org/about/`. Acesso em 4 de dezembro de 2019.

[4] Alexander Mordvintsev e Abid Rahman. *Introduction to OpenCV-Python Tutorials*. 2013. Disponível em `https://docs.opencv.org/master/d0/de3/tutorial_py_intro.html`. Acesso em 4 de dezembro de 2019.

[5] Ricardo Antonello. *Visão Computacional com OpenCV-Python*. 2017. Disponível em `http://professor.luzerna.ifc.edu.br/ricardo-antonello/wp-content/uploads/sites/8/2017/02/Livro-Introdu%C3%A7%C3%A3o-a-Vis%C3%A3o-Computacional-com-Python-e-OpenCV-3.pdf`. Acesso em 4 de dezembro de 2019.