



VISÃO COMPUTACIONAL **com Python e OpenCV:** **uma abordagem introdutória**



Lúcio Flávio de Albuquerque Campos

Lúcio Flavio de Albuquerque Campos

Visão Computacional com Python e OpenCV: uma abordagem introdutória



© copyright 2026 by UNIVERSIDADE ESTADUAL DO MARANHÃO
Qualquer parte desta publicação pode ser reproduzida, desde que citada a fonte.
Todos os direitos desta edição reservados à EDITORA UEMA.

Visão Computacional com Python e OpenCV: uma abordagem introdutória

EDITOR RESPONSÁVEL

Jeanne Ferreira Sousa da Silva

CONSELHO EDITORIAL

Alan Kardec Gomes Pachêco Filho • Ana Lucia Abreu Silva
Ana Lúcia Cunha Duarte • Cynthia Carvalho Martins
Eduardo Aurélio Barros Aguiar • Emanuel Cesar Pires de Assis
Denise Maia Pereira • Fabíola Hesketh de Oliveira
Helciane de Fátima Abreu Araújo • Helidacy Maria Muniz Corrêa
Jackson Ronie Sá da Silva • José Roberto Pereira de Sousa
José Sampaio de Mattos Jr • Luiz Carlos Araújo dos Santos
Marcos Aurélio Saquet • Maria Medianeira de Souza
Maria Claudene Barros • Rosa Elizabeth Acevedo Marin
Wilma Peres Costa

Diagramação: Paul Philippe

C198 Campos, Lúcio Flavio de Albuquerque

Visão computacional com Python e OpenCV: uma abordagem introdutória.
[Ebook] / Lúcio Flavio de Albuquerque Campos. – São Luis, MA: Eduema 2026.

201 f.: il. color.

ISBN: 978-85-8227-764-5

1. Visão Computacional. 2. Python. 3. OpenCv. I. Título.

CDU: 004.4'41

Elaborado por Luciana de Araújo- CRB 13/445

EDITORA UEMA

Cidade Universitária Paulo VI - CP 09 Tirirical - CEP - 65055-970 São Luís - MA

www.editorauema.uema.br – editora@uema.br

APRESENTAÇÃO

A Visão Computacional ocupa atualmente uma posição de destaque entre as áreas mais promissoras da Computação, da Engenharia e da Inteligência Artificial. Presente em sistemas de reconhecimento facial, veículos autônomos, diagnósticos médicos assistidos por computador, monitoramento industrial, agricultura de precisão e inúmeras outras aplicações, essa área do conhecimento tem transformado a forma como máquinas interagem com o mundo visual.

Apesar de sua ampla disseminação, muitos estudantes e profissionais encontram dificuldades ao iniciar seus estudos em Visão Computacional, seja pela forte interdisciplinaridade envolvida, seja pela necessidade de conciliar conceitos matemáticos, computacionais e práticos. Foi justamente pensando nesse público que este livro foi concebido.

A proposta desta obra é apresentar os fundamentos da Visão Computacional de forma acessível, gradual e aplicada, utilizando a linguagem Python e a biblioteca OpenCV como ferramentas centrais. Ao longo dos capítulos, o leitor será conduzido desde os conceitos básicos relacionados à formação das imagens digitais e ao funcionamento do sistema visual humano, até técnicas clássicas de processamento de imagens, filtragem, segmentação, extração de características e reconhecimento de padrões.

Mais do que apresentar algoritmos e códigos, este livro busca desenvolver uma compreensão sólida dos princípios que fundamentam cada técnica. Entender o porquê de um método funcionar é tão importante quanto saber implementá-lo. Por esse motivo, foram incluídas explicações teóricas, ilustrações e exemplos práticos que permitem ao leitor experimentar os conceitos discutidos.

Espera-se que esta obra desperte a curiosidade científica, incentive a experimentação e contribua para a formação de novos pesquisadores, desenvolvedores e entusiastas da área. A Visão Computacional é um campo em constante evolução, e este livro representa um convite para que o leitor dê seus primeiros passos nessa fascinante jornada de ensinar máquinas a interpretar o mundo por meio das imagens.

Boa leitura e excelentes descobertas.

Lúcio Flávio de Albuquerque Campos

LISTA DE FIGURAS

Figura 1 - O olho humano	22
Figura 2 - Passagem de luz pelo olho humano, e uma visão expandida da cascata de fototransdução da retina.	23
Figura 3 – Distribuição dos cones e bastonetes ao longo da retina	25
Figura 4 - Formação da imagem no olho.....	26
Figura 5 - Convenção dos eixos para representação de imagens digitais....	27
Figura 6 - Processo de aquisição de uma imagem com uma câmera digital que usa sensor CMOS.....	28
Figura 7 - Amostragem de uma imagem digital. Em (a) a imagem original. (b) mostra uma grade de amostragem, e em (c) a divisão da imagem em pequenos quadrados, chamados de amostra.	29
Figura 8 - Quantização de uma imagem digital. 10 (c) é a quantização numérica dos pixels do recorte 10 (b), segundo a escala monocromática de 8 bits, apontada em 8 (d).....	30
Figura 9 - (a) imagem com boa resolução. (b) imagem com baixa resolução.....	31
Figura 10 - Fluxograma de um Sistema Básico de Visão Computacional..	32
Figura 11 - Robô educacional wifi com câmera, e uma câmera de rodovias.....	32
Figura 12 – Etapas do pré-processamento para remoção de etiquetas e falhas na digitalização, utilizando thresholding e operadores morfológicos ...	33
Figura 13 – Imagem Original (15-a). Pré-Processamento (15-b). Equalização adaptativa (15-c). Melhoramento de contraste, apresentando a segmentação final (15-d).....	34
Figura 14 - Esquema de reconhecimento de Íris. Observe que o reconhecimento só ocorre depois que todas as etapas descritas acima são executadas.....	36

Figura 15 - Imagem carregada por meio do comando <code>cv2.imread</code> , e mostrado em <code>plt.imshow</code>	38
Figura 16 - Posição de pixel e zoom. Note que o cursor do mouse está em algum lugar da imagem, apontando para a posição descrita na direita. Observe também o valor do pixel: 103	39
Figura 17 - Redimensionamento de uma imagem 2 x 2 para imagem 4 x 4.....	40
Figura 18 - Imagem original cortada, redimensionada e rotacionada ...	42
Figura 19 - Imagem original negativa, obtida por meio do código descrito em 4.1	44
Figura 20 - Melhoramento de espectro de Fourier por transformação logarítmica.....	46
Figura 21 - Aplicação da Correção Gamma.....	47
Figura 22 - Equalização de Histograma de uma imagem de baixo contraste.....	48
Figura 23 - Histograma original da imagem.....	49
Figura 24 - Valores de intensidade de pixel - r_k 's, suas ocorrências e as suas distribuições cumulativas s_k 's.....	50
Figura 25 - Multiplicação da distribuição cumulativa (a) e novo mapeamento (b). Histograma equalizado (c)	50
Figura 26 - Equalização de Histograma em Python.....	52
Figura 27 – Ilustração de redistribuição de pixels após aplicação de técnicas de limitação de contraste. O gráfico à esquerda ilustra um histograma equalizado, apresentando um limiar (aqui chamado de clipping). O gráfico à direita ilustra a redistribuição dos pixels ao longo do eixo de intensidade ($L-1$), e apresenta o clipping atual, modificado devido a nova distribuição.	54
Figura 28 – Função de transformação não linear para alargamento de contraste.....	55

Figura 29 – Mamografia com alta densidade, evidenciando baixo contraste entre tecidos	56
Figura 30 - Equalização adaptativa de histograma e limitação de contraste	58
Figura 31 - Exemplos de aplicação de Filtragem Espacial.....	60
Figura 32 - Convolução de uma matriz 9x9 com um kernel 3x3.....	60
Figura 33 - Borda de (a) alta frequência e (b) baixa frequência.....	61
Figura 34 - Exemplo de imagem ruidosa tratada com filtro da mediana	62
Figura 35 - Filtro da mediana em operação	63
Figura 36 - Kernels com tamanho 3x3, 5x5 e 7x7, respectivamente.	64
Figura 37 - Representação da função Gaussiana em 2D com média em (0,0) e desvio padrão $\sigma = 1$	65
Figura 38 - Kernels com tamanho 3x3, 5x5, respectivamente.	65
Figura 39 - Exemplo de Filtro da mediana em Python	67
Figura 40 - Exemplo de Filtro da média em Python.....	68
Figura 41 - Exemplo de Filtro Gaussiano em Python.....	69
Figura 42 - Operador de Sobel de tamanho 3x3	70
Figura 43 - Exemplo de aplicação de diferenciação em uma linha de transição de uma imagem.....	71
Figura 44 - Kernels da filtragem por Laplacianos	72
Figura 45 - Detecção de bordas de Canny	73
Figura 46 - Imagem Original. Suavização (b). Diferenciação (c). Omissão de mínima intensidade (d). Limiarização por histerese e resultado final (e)	75
Figura 47 - Operador de Sobel com kernel 3x3.	76
Figura 48 - Aplicação do filtro Laplaciano.	77

Figura 49 - Filtragem com o algoritmo de Canny.....	79
Figura 50 - Magnitude (b) e fase (c) de uma imagem (a) no Domínio da Frequência	81
Figura 51 - Espectro de magnitude de imagem. Espectro original (a). Espectro Centralizado (b)	81
Figura 52 - Espectro centralizado (a). Máscara de filtro passa baixa (b). Espectro filtrado (c)	82
Figura 53 - Espectro centralizado (a). Espectro com as componentes de frequências localizadas nos cantos, conforme espectro e magnitude original (b).....	83
Figura 54 - Imagem suavizada (b), a partir de (a).....	83
Figura 55 - Fluxograma da filtragem de imagens no domínio da frequência...	84
Figura 56 - Filtragem passa baixas. 56 (a) mostra a imagem original, e seu espectro de frequência em 56 (b). 56 (c) mostra a representação 2-D de um filtro da Gaussiana, passa baixas, e seu espectro em 56 (d). A multiplicação do espectro da imagem com o espectro do filtro gera um espectro resultante, e que após a aplicação da transformada inversa do espectro, resulta na imagem 56 (e).....	85
Figura 57 - Filtragem passa altas: 57 (a) mostra a imagem original, e seu espectro de frequência em 57 (b). 57 (c) mostra a representação 2-D de um filtro da Gaussiana, passa altas, e seu espectro em 57 (d). A multiplicação do espectro da imagem com o espectro do filtro gera um espectro resultante, e que após a aplicação da transformada inversa do espectro, resulta na imagem 57 (e).....	86
Figura 58 - Projeto de filtro passa baixas ideal.....	88
Figura 59 - Projeto de Filtro Butterworth Passa Baixa.....	88
Figura 60 - Filtro Passa Baixa Gaussiano	89
Figura 61 (a) - Espectros de magnitude da image teste para filtros passa baixa....	92
Figura 61 (b) - Resultado de filtrações Passa-Baixa no domínio da frequência, através dos filtros: Ideal, Butterworth e Gaussiano.....	93

Figura 62 - Projeto de filtro passa altas ideal.....	94
Figura 63 - Projeto de Filtro Passa Alta de Butterworth	95
Figura 64 - Projeto de Filtro Passa Alta Gaussiano.....	96
Figura 65 (a) - Espectros de magnitude da imagem teste para filtros passa alta ..	99
Figura 65 (b) - Resultado de filtragens Passa-Alta no domínio da frequência, através dos filtros: Ideal, Butterworth e Gaussiano. Observe o realce das bordas, característico de filtros de aguçamento, que são similares à filtros de passa alta, mas em tempo discreto.....	100
Figura 66 - Filtragem no domínio da frequência a partir de filtros de domínio espacial.....	102
Figura 67 - Modelo do processo de restauração de imagens	103
Figura 68 - Ruídos mais encontrado e suas PDF's.....	105
Figura 69 - Imagem padrão para estimar a PDF de um ruído aleatório em imagem.	106
Figura 70 - Imagens teste com adição de ruído: Gaussiano (a), Rayleigh (b), Gamma (c), Exponencial (d), Uniforme (e), Salt and Pepper (f) e seus respectivos histogramas.	106
Figura 71 - Restauração no Domínio Espacial em Python por meio dos Filtros: Mediana, Média e Gaussiano	108
Figura 72 - Comparativos entre espectros de imagens sem ruído periódico e com ruído periódico	109
Figura 73 - Gráficos em perspectiva dos Filtros: Ideal, Passa Banda, Gaussiano.	110
Figura 74 - (a) Imagem com ruído periódico. (b) Espectro de Fourier. (c) Região demarcada do espectro onde se encontra os pontos de ruído. (d) Filtro Rejeita banda, que atenua a região demarcada em(c). e) Espectro Resultante. (f) Imagem filtrada.	110
Figura 75 - Filtros Notch: Ideal (a). Butterworth (b). Gaussiano (c)	111

Figura 76: (a) Imagem com ruído periódico. (b) Espectro de Fourier. (c) Região demarcada do espectro onde se encontra os pontos de ruído. (d) Filtro Notch, que atenua a região demarcada em(c). e) Espectro Resultante. (f) Imagem filtrada.	112
Figura 77 (a) - Imagem original e seu espectro de Fourier.....	113
Figura 77 (b) - Filtro Notch	114
Figura 77 (c) - Filtro Notch.....	115
Figura 78 - Espectro de luz visível	116
Figura 79 - Comprimento de onda das principais cores	117
Figura 80 - Mistura de cores.....	119
Figura 81 - Cubo de cores do modelo RGB.....	119
Figura 82 - Mistura de cores, modelo CMYk.....	120
Figura 83 - Modelo de Cor HSV	122
Figura 84 - Canais RGB de uma imagem colorida	122
Figura 85: Imagem RGB e BRG	124
Figura 86 - Visualização de canais de cor em uma imagem.....	125
Figura 87 - Equalização de uma imagem colorida.....	126
Figura 88 - Segmentação de uma Digital.....	127
Figura 89 - Formas mais comuns de elementos estruturantes	128
Figura 90 - Modelo de código de elemento estruturante em OpenCV.....	128
Figura 91 - Processo de Erosão	130
Figura 92 - Processo de dilatação	131
Figura 93 - Abertura.....	132
Figura 94: Fechamento	133

Figura 95 - Aplicação do gradiente morfológico, para detecção de bordas...	134
Figura 96 - Aplicação das técnicas de abertura e fechamento para eliminação de ruídos em uma imagem de impressão digital	135
Figura 97 - Aplicação do gradiente morfológico, com kernels de 3x3 e 5x5...	136
Figura 98 - Tomografia de pulmão, e a visualização de possíveis nódulos pulmonares (a). Em (b), observa-se apenas os prováveis nódulos segmentados, para melhor visualização.....	137
Figura 99 - Histograma de uma imagem de 8 bits, em níveis de cinza.....	138
Figura 100 - Histograma de uma imagem em condições ideais de contraste..	139
Figura 101 - Histograma de uma imagem em condições ruins de contraste ..	139
Figura 102 - Retinografia colorida.....	140
Figura 103 - Segmentação por limiarização de uma retinografia	142
Figura 104 - Segmentação por meio do Método de OTSU	143
Figura 105 - Transformação entre espaços geométricos.....	143
Figura 106 - Espaço de parâmetros	144
Figura 107 - Transformada de Hough na detecção de Retas. No gráfico de domínio espacial, temos dois pontos que fazem parte de uma reta, e no gráfico de domínio transformado, temos duas senóides (cada uma corresponde a um ponto) que possuem um ponto de intersecção, o que sugere que os dois pontos pertencem à mesma reta.	145
Figura 108 - Transformada de Hough na Detecção de Retas. Em (a), temos um conjunto finitos de pontos, que representam uma reta. Em (b), observa – se finitos senóides que possuem o mesmo ponto de intersecção. O ponto de interseção representa a reta.	145
Figura 109 - Transformada de Hough na Detecção de círculos No espaço-imagem, temos um conjunto finitos de pontos, que representam um círculo. Já no espaço transformado, observa – se finitos senóides que possuem o mesmo ponto de intersecção. O ponto de interseção representa a origem do círculo.	146

Figura 110 - Modelo de extração de características para detecção facial	147
Figura 111 - Exemplos de textura em imagens.....	148
Figura 112 - Imagem teste para detecção de formas.....	152
Figura 113 - Imagem binarizada.....	153
Figura 114 - Imagem original com contornos e formas geométricas detectadas.....	155
Figura 115 - Imagem teste do algoritmo de detecção de placas.....	156
Figura 116 - Imagem redimensionada, e em escala de cinza	158
Figura 117 - Imagem teste, após aplicação do filtro gaussiano	159
Figura 118 - Imagem teste, após aplicação do filtro de detecção de bordas	160
Figura 119 - Detecção de contornos aplicada na imagem segmentada por canny.....	161
Figura 120 - Redução de contornos detectados.....	162
Figura 121 - Delimitação da placa	163
Figura 122 - Placa segmentada da imagem original	163
Figura 123 - Imagem contendo cinco moedas, de tamanhos distintos	165
Figura 124 - Imagem em escala de cinza	166
Figura 125 - Detecção de bordas das moedas	167
Figura 126 - Dilatação das bordas das moedas.....	168
Figura 127 - Imagem original, com as moedas detectadas	169
Figura 128 - Mamografia com falhas na aquisição, e placa de identificação do paciente.....	170
Figura 129 - Máscara obtida por meio de threshold binário.....	172
Figura 130 - Correção de falhas da máscara por meio do operador de abertura.....	173

Figura 131 - Correção de falhas da máscara por meio do operador de abertura.....	174
Figura 132 - Comparativo entre a mamografia original, e o resultado final do algoritmo proposto.....	175
Figura 133 - Imagem de teste, para execução do pytesseract.....	176
Figura 134 - Resultado obtido por meio do algoritmo proposto.....	177
Figura 135 - Wavelets de Haar mais utilizadas.....	178
Figura 136 - Wavelets de Haar utilizadas para detecção de faces	180
Figura 137 - Cálculo da imagem integral.....	181
Figura 138 - Modelo de face, utilizando Haar features.....	182
Figura 139 - Coordenadas do bounding-box	183
Figura 140 - Fotografia com 18 faces frontais.....	184
Figura 141 (a) - Faces detectadas por meio do algoritmo Haar Cascade....	186
Figura 141 (b) - Cálculo dos Histogramas de gradientes orientados (HOG) ..	189
Figura 142 - Vetores de Direção do Gradiente	190
Figura 143 - Desenho da Bouding box.....	191
Figura 144 - Faces detectadas por meio do algoritmo Haar Cascade	193

Sumário

1. INTRODUÇÃO.....	19
1.1 Aplicações.....	20
1.2 Python, OpenCV, Bibliotecas e IDE's.....	21
 2. FUNDAMENTOS DE IMAGEM DIGITAL	22
2.1 O Olho Humano.....	22
2.1.1 Cones e Bastonetes.....	24
2.1.2 Formação da imagem no olho.....	25
2.2 Imagem Digital	26
2.2.1 Processo de aquisição de uma imagem.....	27
2.2.2 Resolução	31
2.3 Sistema Básico de Visão Computacional.....	31
 3. OPERAÇÕES BÁSICAS	36
3.1 Carregar e salvar uma imagem.....	36
3.2 Visualização de Pixels e Zoom	38
3.3 Cortando, redimensionando e rotacionando uma imagem.....	39
 4. TRANSFORMAÇÕES DE INTENSIDADE.....	43
4.1 Negativo da imagem.....	43
4.2 Transformação Logarítmica	45

4.3 Transformação Gamma	46
4.4 Equalização de Histograma	48
4.4.1 Passos para a equalização de um histograma	49
4.5. Equalização adaptativa	52
4.5.1. Alargamento de Contraste.....	54
 5. FILTRAGEM ESPACIAL	58
5.1. Filtros de Suavização.....	62
5.1.1. Filtro da Mediana.....	62
5.1.2. Filtro da Média.....	63
5.1.3. Filtro Gaussiano.....	64
5.2. Filtros de Aguçamento.....	69
5.2.1. Filtro de Sobel.....	70
5.2.2. Filtro Laplaciano	70
5.2.3. Filtros de Canny	72
 6. PROCESSAMENTO DE IMAGENS NO DOMÍNIO DA FREQUÊNCIA.....	79
6.1. Filtro Passa Baixa.....	86
6.1.1. Filtro Passa Baixa Ideal	87
6.1.2. Filtro Passa Baixa Butterworth.....	88
6.1.3. Filtro Passa Baixa Gaussiano	89
6.2. Filtro Passa Alta.....	93

6.2.1. Filtro Passa Altas Ideal	93
6.2.2. Filtro Passa Alta de Butterworth	94
6.2.3. Filtro Passa Alta Gaussiano.....	95
6.3. Projeto de filtros no domínio da frequência a partir de filtros de domínio espacial	100
7. RESTAURAÇÃO DE IMAGENS.....	103
7.1. Ruído de natureza espacial	104
7.1.1. Filtros de Estatística de Ordem.....	107
7.2. Filtros de Natureza Periódica.....	108
7.2.1. Filtro Rejeita Banda	110
7.2.2. Filtro Notch	111
8. IMAGENS COLORIDAS	116
8.1. Fundamento de Cores	116
8.1.1 Modelo de Cores	118
8.1.2. Modelo RGB	118
8.1.3. Modelo CMYk.....	120
8.1.4. Modelo HSV	121
8.2. Imagens coloridas em Python/OpenCV.....	122
9. MORFOLOGIA MATEMÁTICA.....	127
9.1. Elementos Estruturantes	128
9.2. Operadores Morfológicos	129

9.2.1. Erosão.....	129
9.2.2. Dilatação.....	130
9.3. Combinação de técnicas de morfologia.....	131
9.3.1. Abertura.....	131
9.3.2. Fechamento.....	132
9.3.3. Gradiente morfológico	133
10. SEGMENTAÇÃO DE OBJETOS	137
10.1. Segmentação por Limiarização.....	138
10.2. Método de Otsu.....	142
10.3. Transformada de Hough.....	143
10.3.1. Transformada de Hough na detecção de retas.....	144
10.3.2. Transformada de Hough na detecção de Círculos	146
11. EXTRAÇÃO DE CARACTERÍSTICAS.....	147
11.1. Descritor de aspectos	148
11.2. Descritores Dimensionais	149
11.3. Descritores Inerciais	151
12. APLICAÇÕES.....	156
12.1. Detecção de placas	156
12.2. Identificação de moedas.....	164
12.3. Melhoramento de imagens médicas	170

12.4. Reconhecedor Óptico de Caracteres (OCR)	175
12.5. Detecção e Reconhecimento de Faces	177
12.5.1. Algoritmo Haar – Cascade	177
12.5.2. Histograma de Gradientes Orientados (HOG)	186
 13. CONSIDERAÇÕES FINAIS.....	194
 REFERÊNCIAS.....	196

1. INTRODUÇÃO

Visão Computacional é um ramo da inteligência artificial que se dedica a desenvolver algoritmos e sistemas capazes de interpretar imagens ou sequências de imagens (vídeos) de forma automatizada, com o objetivo de extrair, processar e compreender informações visuais em nível simbólico ou semântico

A visão computacional habilita que máquinas “vejam” e interpretem imagens e vídeos, extraindo informações significativas para tomar decisões ou executar ações automaticamente, oferecendo a capacidade de “enxergar” e “entender” o mundo visual, similar à visão humana, mas com maior precisão e capacidade de processamento.

Entenda “máquina” de qualquer forma, seja uma máquina – computador de mesa, ou uma máquina – robô que desarma bomba ou robôs de exploração espacial. Tais exemplos não precisam ser tão robustos, já que temos um poderoso computador em mãos, que colocamos no bolso, e usa continuamente técnicas de visão computacional: o Smartphone.

Em visão computacional, estuda-se métodos para aquisição, processamento, análise e compreensão de imagens, com objetivo de adquirir dados multidimensionais extraídos a partir do mundo real, para produzir, por exemplo, informação numérica ou simbólica na forma de decisão.

A visão computacional é uma evolução gradual do Processamento Digital de Imagens (PDI), e inclusive, iniciaremos o curso com todas as noções básicas de PDI, para, somente depois, aplicar estes conceitos para ensinar uma máquina a “enxergar sozinha”, podemos assim dizer.

Neste sentido, vamos usar PDI para procurar emular a visão humana, em que será dado como entrada uma imagem, que será pré-processada, filtrada, extraído características, e se for o caso, rotulada, e a saída do sistema serão números, textos, avisos de alerta, etc...

Não vamos aqui discutir a fronteira entre Processamento de Imagens e Visão Computacional, e sim usar as duas ciências ao nosso favor, gerando assim conhecimento, curiosidades, descobertas e claro, muita, muita, mas muita pesquisa.

1.1 Aplicações

A visão computacional evoluiu em várias áreas, desde ciência, e até no dia a dia do cidadão comum. Abaixo listamos uma gama de aplicações em várias áreas específicas.

Na Indústria

- Manufatura: detecção de falhas, controle de qualidade automatizado.
- Logística: rastreamento de produtos, inspeção de empacotamento.
- Agricultura: monitoramento por drones, classificação de culturas.
- Mineração: mapeamento 3D, análise de risco.
- Petróleo e Gás: inspeção de dutos, detecção de vazamentos.

No Cotidiano do Cidadão Comum

- Smartphones: desbloqueio facial, realidade aumentada.
- Redes sociais: filtros visuais inteligentes.
- E-commerce: provas virtuais de roupas e acessórios.
- Transporte: veículos com visão assistida.
- Varejo automatizado: lojas sem caixa com visão computacional.

Na Ciência

- Medicina: diagnóstico por imagem, cirurgia assistida.
- Biologia: análise celular, microscopia computacional.
- Astronomia: classificação de galáxias, eventos cósmicos.
- Ciências ambientais: monitoramento via satélite.
- Física de partículas: rastreamento em detectores visuais.

A visão computacional está no centro da transformação digital, permitindo a automatização cognitiva de processos que antes exigiam percepção e julgamento humano, mesmo com subjetividades. Com a integração com IA generativa e modelos multimodais, espera-se um grande salto no crescimento de aplicações criativas, diagnósticas e adaptativas. A tendência é que modelos generalistas e auto-explicáveis dominem os próximos anos, tornando a visão computacional ainda mais acessível, ética e robusta.

1.2 Python, OpenCV, Bibliotecas e IDE's

OpenCV é uma biblioteca multiplataforma, desenvolvida pela Intel em 2000, totalmente livre, para o desenvolvimento de aplicações de Visão Computacional. A biblioteca possui mais de 500 funções distribuídos entre seus módulos de Processamento de Imagens e Vídeo I/O, Estrutura de dados, Álgebra Linear, GUI (Interface Gráfica do Usuário) básica com sistema de janelas independentes, Controle de mouse e teclado, além de mais de 350 algoritmos de Visão computacional, tais como: Filtros de imagem, calibração de câmera, reconhecimento de objetos, análise estrutural e outros. Além disso, a biblioteca OpenCV possui processamento em tempo real.

A ideia principal da biblioteca foi a de tornar a visão computacional acessível a usuários e programadores em áreas tais como a interação humano-computador em tempo real e a robótica. A biblioteca está disponível com o código fonte e os executáveis (binários) otimizados para os processadores Intel. Um programa OpenCV, ao ser executado, invoca automaticamente uma DLL (Dynamic Linked Library) que detecta o tipo de processador e carrega, por sua vez, a DLL otimizada para este. Juntamente com o pacote OpenCV é oferecida a biblioteca IPL (Image Processing Library), da qual a OpenCV depende parcialmente, além de documentação e um conjunto de códigos exemplos.

Juntamente com a OpenCV, algumas bibliotecas Python também são necessárias, tais como Numpy (Numeric Python), SciPy (Cientific Python) e Matplotlib, que é uma biblioteca para plotagem de gráficos com sintaxe similar ao Matlab.

O Jupyter Notebook é uma aplicação web de código aberto amplamente utilizada na programação, especialmente em ciência de dados, aprendizado de máquina, análise estatística e ensino de computação. Ele permite a criação e o compartilhamento de documentos que contêm código executável, texto formatado (em Markdown), visualizações e equações matemáticas. A principal linguagem utilizada com o Jupyter é o Python, embora ele também ofereça suporte a outras linguagens por meio de diferentes kernels.

Um dos grandes atrativos do Jupyter Notebook é a sua interatividade: os usuários podem executar blocos de código em células individuais, executá-las individualmente, visualizar os resultados e fazer alterações de forma dinâmica. Isso facilita testes, análises exploratórias e a documentação do processo de desenvolvimento em tempo real.

Uma ferramenta bastante similar ao Jupyter Notebook é o Google Colab (abreviação de Colaboratory), oferecido gratuitamente pelo Google. O Google Colab possui uma interface muito parecida com a do Jupyter, permitindo a execução de código Python em células, inclusão de textos explicativos e visualizações gráficas. A principal diferença é que o Colab roda totalmente na nuvem, ou seja, não requer instalação local e permite o uso de recursos computacionais gratuitos, como GPUs e TPUs, diretamente nos servidores do Google. Além disso, o Colab facilita a colaboração entre usuários, pois está integrado ao Google Drive, permitindo o compartilhamento e edição conjunta de notebooks, semelhante ao funcionamento de um Google Docs.

Todos os exemplos deste trabalho serão executados no Google Colab, e disponibilizados na webpage do livro.

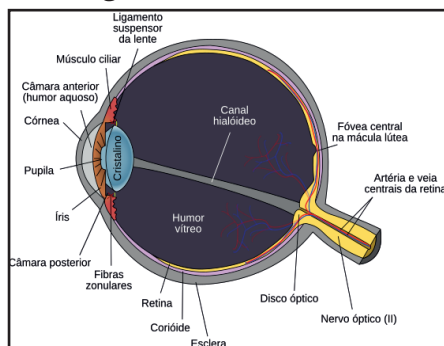
2. FUNDAMENTOS DE IMAGEM DIGITAL

2.1 O Olho Humano

O olho humano é um sistema óptico complexo, formado por vários meios transparentes além de um sistema fisiológico com inúmeros componentes. Todo o conjunto que compõe a visão humana é chamado globo ocular. A luz incide na córnea e converge até a retina, formando as imagens.

A Figura 1 abaixo ilustra a estrutura do olho humano

Figura 1 - O olho humano



Fonte: MediaCommons, 2025

Observando a Figura 1, vemos a extremidade anterior, a coróide e dividida em **corpo ciliar e íris**, que contrai e expande de forma a controlar a quantidade de luz que entra no olho.

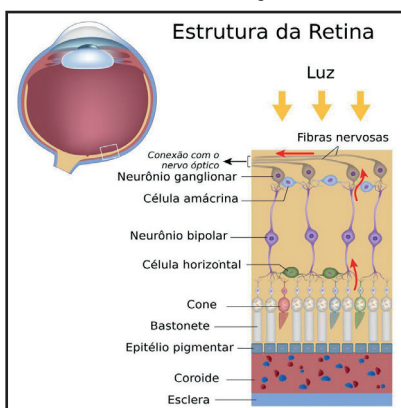
A abertura central da íris, chamada de pupila, varia de 2mm a 8mm em diâmetro. A variação do diâmetro da pupila regula a quantidade de luz que incide no interior do olho.

O cristalino é formado por camadas concêntricas de células. Ele absorve ~8% da luz visível com absorção relativamente maior nos comprimentos de onda mais curtos.

A membrana mais interna do olho é a retina, onde a luz de um objeto fora do olho se torna uma imagem. É na retina que se encontram as células responsáveis pela recepção da luz e pela emissão do estímulo elétrico responsável pela visão.

Dois tipos de células que contêm pigmentos em seu interior chamam a atenção: os **cones** e os **bastonetes**. Os bastonetes existem em maior quantidade na periferia da retina e são estimulados com luz de baixa intensidade. É frequente dizer que são usados para a visão no escuro e não registram cores. Os cones, por sua vez, ocorrem principalmente na região central da retina, principalmente na fóvea, e seu estímulo depende de altas intensidades luminosas, reconhecem cores e diz-se que são células utilizadas quando há claridade. A Figura a seguir ilustra a passagem de luz no olho humano, e em especial na retina, onde as células fotorreceptoras são ativas, e enviam impulsos neuronais para o córtex visual.

Figura 2 - Passagem de luz pelo olho humano, e uma visão expandida da cascata de fototransdução da retina.



Fonte: Retina, 2025

Quando o pigmento das células fotorreceptoras são estimulados, eles geram modificações energéticas, que são transmitidas às células sensitivas, cujos prolongamentos se reúnem, formando o nervo óptico. Este se conecta com o cérebro, conduzindo os impulsos para determinada área do lobo occipital, onde as informações são decodificadas e as imagens são reconhecidas.

2.1.1 Cones e Bastonetes

Os cones e os bastonetes são as células fotorreceptoras localizadas na retina, que são responsáveis por iniciar a cascata de transdução entre a luz e o nervo óptico, que está ligado ao córtex visual, onde as imagens são formadas e reconhecidas.

Basicamente, cada retina possui cerca de 150 milhões de bastonetes, que estão situados nas laterais do globo ocular. Os bastonetes fornecem visão **escotópica**, ou seja, **NÃO** tem resolução para os detalhes e contornos dos objetos nem para determinação de sua cor.

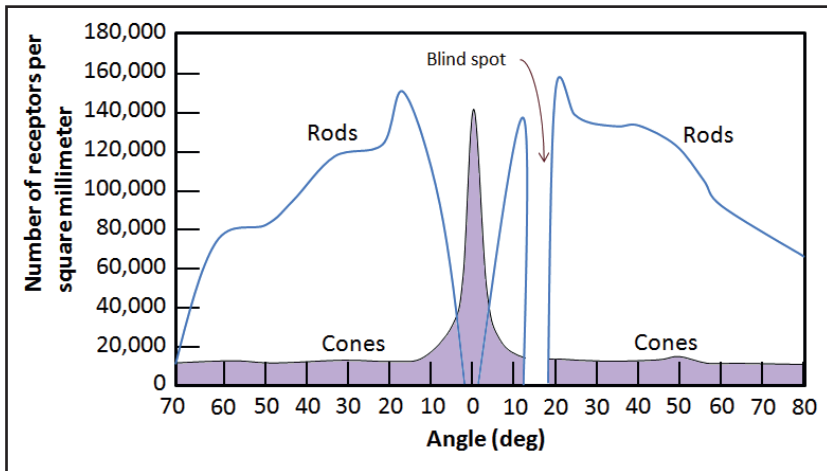
Dentre várias características dos bastonetes, podemos citar: Baixo limiar para a detecção de luz; alta sensibilidade; especializado para a visão noturna; alta amplificação, detecção de um único fóton; na luz do dia sofre saturação, ou seja, deixa de responder; operam melhor sob reduzida condição de luminosidade; mais sensível à luz espalhada; contém mais pigmento visual fotossensível do que os cones, capacitando-os a uma maior captura de luz.

Já os cones são em ordem de 6 a 7 milhões por retina, estão situados mais próximos à fóvea, região mais central da retina. Os cones fornecem uma visão fotópica, ou seja, alta acuidade, com grande resolução para os detalhes e contornos dos objetos, importante para visão das cores.

Pode-se citar como demais características dos cones: Baixa sensibilidade; especializado para a visão diurna; menos fotopigmentos; menor amplificação; saturação somente com luz muito intensa; alta resolução temporal: resposta rápida e tempo de integração curto; mais sensíveis a raios axiais diretos.

A Figura 3 abaixo ilustra a distribuição dos cones e bastonetes ao longo da retina. Pode-se observar que os bastonetes estão presentes nas regiões laterais da retina, totalmente ausentes no ponto cego, e praticamente ausentes na fóvea. Já os cones, estão pouco presentes nas laterais, totalmente ausentes no ponto cego, e em enorme quantidade na fóvea.

Figura 3 – Distribuição dos cones e bastonetes ao longo da retina

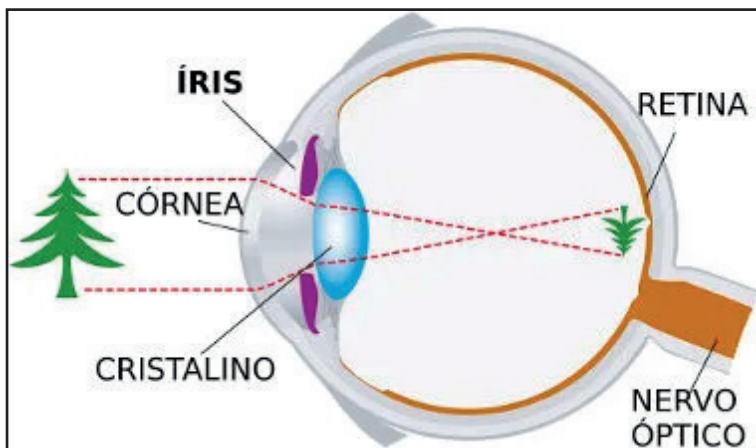


Fonte: Adaptado de LIN *et al.* (2012).

2.1.2 Formação da imagem no olho

Como já foi visto na seção anterior, as células fotorreceptoras da retina transformam as ondas luminosas em impulsos elétricos. A partir daí, o nervo ótico os conduz até o cérebro, onde ocorre a interpretação da imagem. Cabe ressaltar que no cristalino a imagem sofre refração, logo, forma-se uma imagem invertida na retina, o correto posicionamento da imagem é feita no córtex visual. A Figura 4 ilustra o processo de formação da imagem no olho.

Figura 4 - Formação da imagem no olho



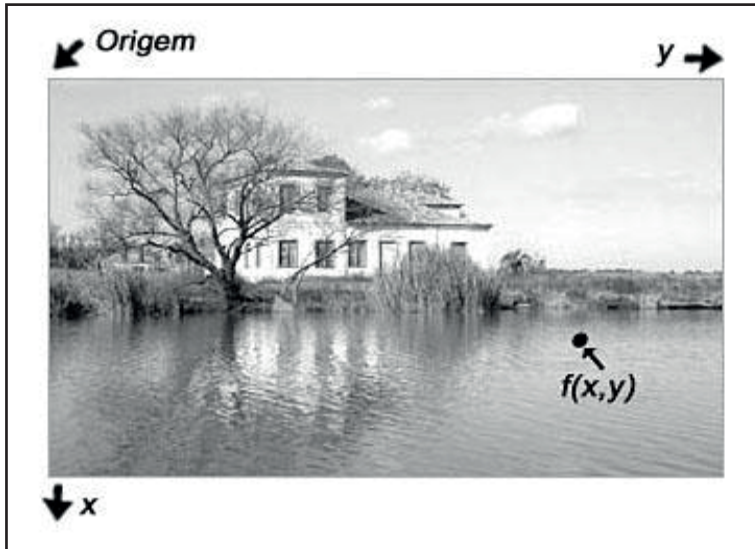
Fonte: Mundo Educação (2025).

2.2 Imagem Digital

Uma imagem digital monocromática pode ser definida como uma função bidimensional 2D, de intensidade de luz $f(x,y)$, sendo x e y coordenadas espaciais. Em uma imagem digital, cada ponto $f(x,y)$ representa um elemento infinitesimal de pintura, do qual chamamos de pixel.

A Figura 5 abaixo ilustra uma imagem digital e orientação do eixo na convenção de coordenadas cartesianas $f(x,y)$ adotada. Observe que a origem é na parte superior da imagem, e não no canto inferior esquerdo, como todo e qualquer gráfico de uma função $f(x,y)$.

Figura 5 - Convenção dos eixos para representação de imagens digitais

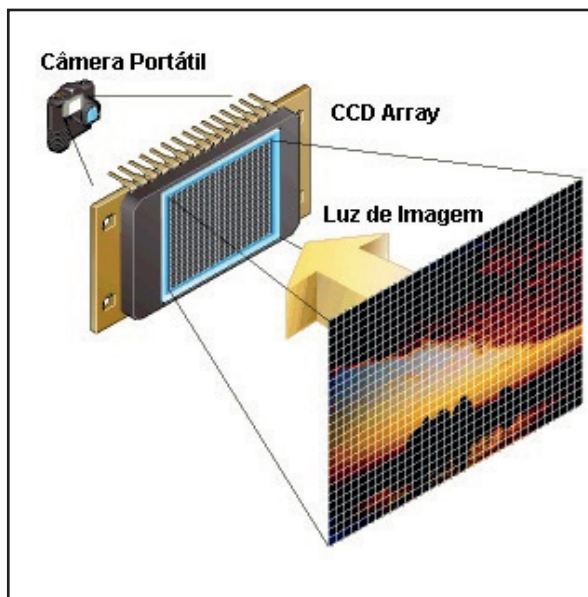


Fonte: Adaptado de Gonzales (2001).

2.2.1 Processo de aquisição de uma imagem

Uma imagem, de uma forma bem geral, é captada por um sensor, e a partir daí, passará por todas as etapas de digitalização. A Figura 6 abaixo ilustra o processo de captura de uma imagem por um sensor CMOS (complementary metal-oxide semiconductor), comumente encontrado em câmeras de fotografia.

Figura 6 - Processo de aquisição de uma imagem com uma câmera digital que usa sensor CMOS



Fonte: Adaptado de UFJF (2025)

Após a captura da imagem, é necessário seguir etapas necessárias para a digitalização. Amostragem e quantização. A seguir falaremos sobre tais passos.

A. Amostragem

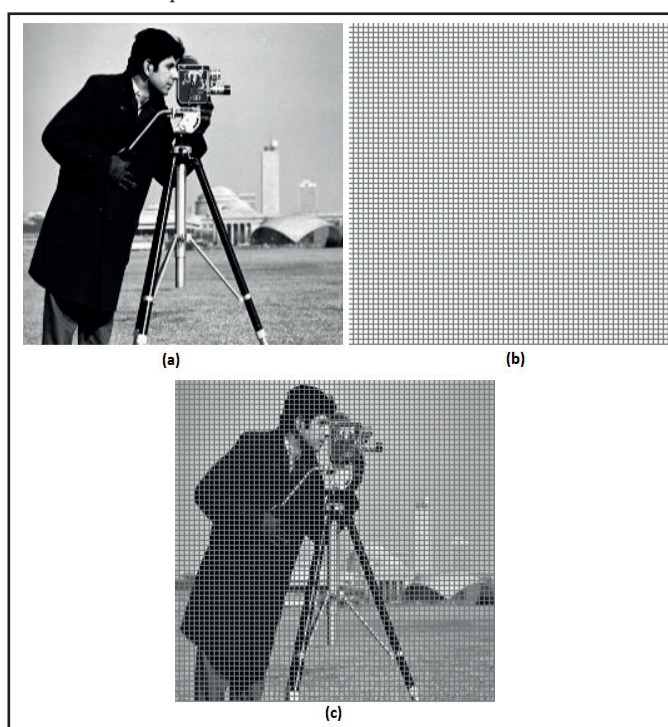
Amostragem é a escolha de um ponto dentro de um espaço, ou melhor, a escolha de um conjunto de pontos dentro de um espaço real que irão compor a imagem digital. Definir a amostragem é conseguir selecionar uma quantidade finita de pontos de uma imagem com infinitos pontos, em outras palavras, é conseguir selecionar pequenos conjuntos de espaços de forma que, devidamente arranjados, estes conjuntos se tornem a imagem discreta.

A amostragem está diretamente ligada com a quantidade de informação que se deseja guardar. Quanto maior a amostragem, mais detalhes teremos e consequentemente maior será o espaço necessário para o armazenamento. Para execução da amostragem sobre uma imagem

real, faz-se necessário definirmos uma grade de amostragem, essa grade é aplicada sobre a imagem de forma que cada célula da grade contém uma sub-imagem.

A Figura 7 abaixo ilustra a sequência de um processo de amostragem. Em 7 (a) temos a nossa imagem de exemplo. Em 7(b) uma grade de amostragem.

Figura 7 - Amostragem de uma imagem digital. Em (a) a imagem original. (b) mostra uma grade de amostragem, e em (c) a divisão da imagem em pequenos quadrados, chamados de amostra



Fonte: Adaptado de Alencar (2025)

Cada um dos pequenos quadrados / células desta imagem é uma sub-imagem, o conjunto destas sub-imagens é uma amostragem da imagem original. Na prática, o nome dado à célula em questão é pixel, que significa elemento de pintura.

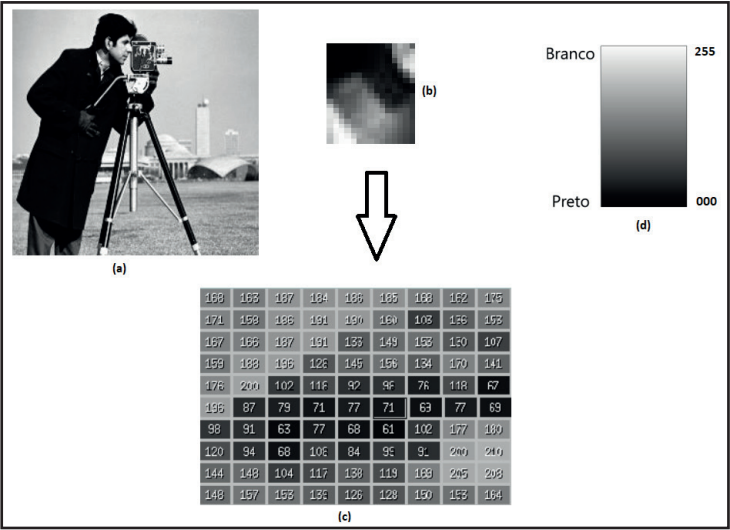
Dentro de cada um desses pixels iremos escolher uma cor (no caso da Figura acima, um tom de cinza), e este processo é conhecido como quantização.

B. Quantização

É a representação de um valor medido de um pixel aproximado por um inteiro. Pode se tomar como exemplo que uma imagem monocromática terá como quantização 28 bits, ou será terá 256 níveis de cinza, onde podemos convencionar que a cor preta pura terá o valor 0, e a cor branca pura 255. Sendo assim, um sistema de quantização vai ler a cor do pixel e atribuir um número entre 0 e 255, obedecendo a escala monocromática.

A Figura 8 (c) ilustra a quantização de um recorte da imagem 8 (a).

Figura 8 - Quantização de uma imagem digital. 10 (c) é a quantização numérica dos pixels do recorte 10 (b), segundo a escala monocromática de 8 bits, apontada em 8 (d)

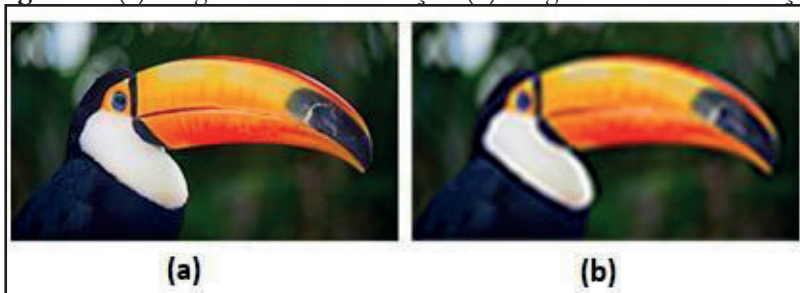


Fonte: Adaptado de Alencar (2025)

2.2.2 Resolução

Resolução é o elemento que mede a definição da imagem, que pode ser chamada de PPI (Pixels por polegada), ou DPI (pontos por polegada). Dessa forma, quanto maior o PPI ou DPI, maior a quantidade de pixels em um quadrado de 1 x 1 polegada (que equivale a 2,54cm x 2,54cm). O grande segredo dos dispositivos de alta resolução (impressoras e monitores), é o tamanho do pixel (ou dot) ser muito pequeno. Assim, no espaço de 1×1 polegadas cabem uma quantidade imensa de pontos. A Figura a seguir ilustra duas imagens, uma com alta resolução, ou seja, com grande quantidade de pixels por polegadas, e a outra com baixa resolução, pequena quantidade de pixels por polegada.

Figura 9 - (a) imagem com boa resolução. (b) imagem com baixa resolução

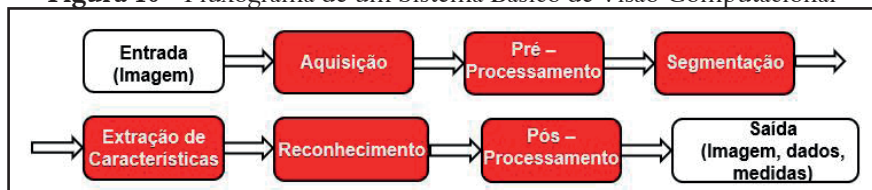


Fonte: Adaptado de Bonito e Pantanal (2025)

2.3 Sistema Básico de Visão Computacional

Um esquema básico que pode ilustrar um sistema básico de visão Computacional é mostrado na Figura abaixo. Basicamente existem 6 passos a serem seguidos: Aquisição da imagem, Pré – Processamento, Segmentação, Extração de Características, Reconhecimento de Padrões e Pós processamento. A seguir falaremos de cada um deles de forma bem breve.

Figura 10 - Fluxograma de um Sistema Básico de Visão Computacional



Fonte: O autor (2026)

A. Aquisição de dados

Este é o passo primordial para o sistema. A imagem deve ser bem capturada, de preferência com boa resolução. Atualmente, já contamos com câmeras robustas que conseguem captar cenas a metros de distância, e na grande maioria das vezes, em movimento. Existem nos mercados câmeras de aquisição de imagens específicas para visão computacional, tal como as câmeras de robôs autônomos, que são capazes de, desde reconhecer trajetos até detectar movimentos, e câmeras de alta resolução e alcance, como as câmeras de rodovias, que detectam velocidade e também servem para a detecção e reconhecimento das placas dos veículos.

A Figura a seguir ilustra um exemplo de cada câmera citada acima

Figura 11 - Robô educacional wifi com câmera, e uma câmera de rodovias



Fonte: Adaptado de Levi (2023) e Santana (2022)

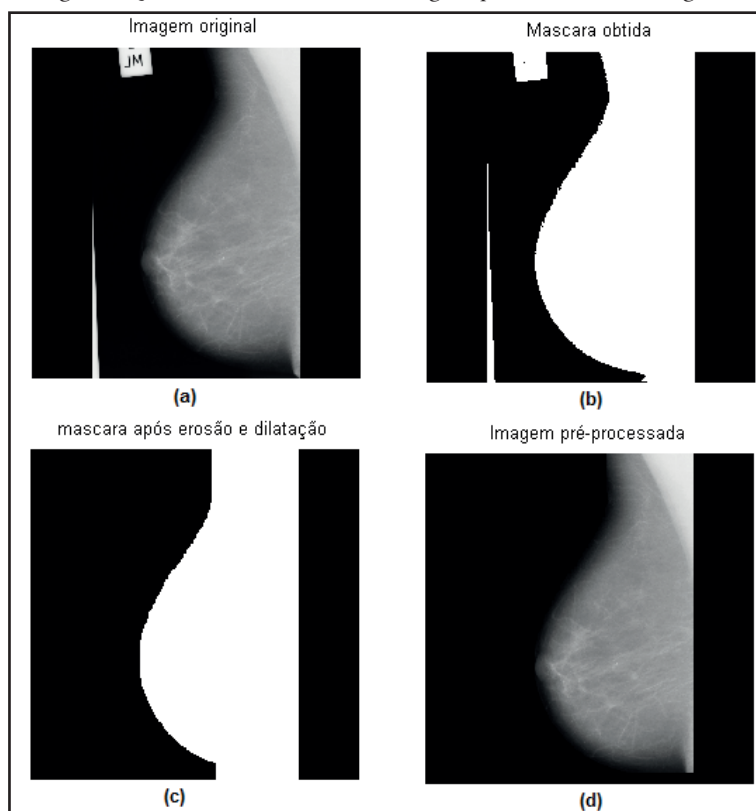
B. Pré-processamento

A etapa de pré-processamento é necessária para remover imperfeições da imagem capturada, ou prepará-la para a etapa de segmentação. Neste passo, pode utilizar filtros para remover ruídos ou

realçar bordas, técnicas em melhoramento de contraste, e até técnicas de morfologia matemática, para remover descontinuidades ou partes indesejadas da imagem.

A Figura a seguir ilustra as etapas de pré-processamento de uma mamografia digital. A imagem original contém imperfeições, tais como falhas na digitalização, e estruturas indesejadas, como a placa de identificação do paciente. Foram utilizadas técnicas de thresholding e operadores morfológicos de abertura e fechamento, para obter a imagem otimizada, pronta para ser utilizada no próximo passo, a segmentação.

Figura 12 – Etapas do pré-processamento para remoção de etiquetas e falhas na digitalização, utilizando thresholding e operadores morfológicos

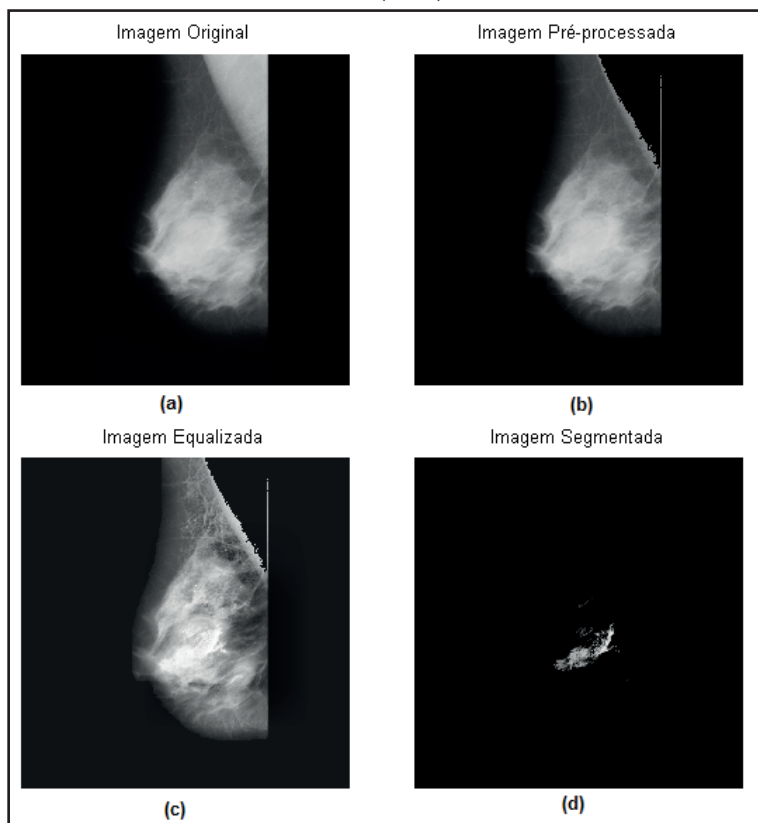


Fonte: Adaptado de Suckling (1994)

C. Segmentação

A etapa de segmentação é outro passo importante do fluxograma. Consiste em extrair a região de interesse a ser estudada, desprezando partes não importantes da imagem. A Figura a seguir ilustra a segmentação de um tumor de câncer, em uma mama de alta densidade, onde é difícil o diagnóstico preciso, já que a textura do tecido mamário é muito similar à do tumor. Neste exemplo, foi utilizado técnicas de equalização adaptativa e melhoramento de imagens, para conseguir segmentar somente a região de interesse.

Figura 13 – Imagem Original (15-a). Pré-Processamento (15-b). Equalização adaptativa (15-c). Melhoramento de contraste, apresentando a segmentação final (15-d).



Fonte: Adaptado de Suckling (1994)

As técnicas mais comuns de segmentação são a clusterização (agrupamento). Usa-se de forma mais comum os algoritmos K-means, K-nn, Mapas auto-organizáveis, dentre outros. Todas as técnicas descritas acima, e nos exemplos, serão estudadas neste curso.

D. Extração de Características

A etapa de extração de características consiste em extrair informações (da região segmentada) que possam diferenciar, por exemplo, um tecido normal de um tecido acometido por uma neoplasia. Nessa etapa, será utilizada como poderosa ferramenta a estatística, assim como técnicas robustas de processamento de sinais, e, dependendo da complexidade do caso, deep learning. É válido ressaltar que as melhores técnicas de extração de características possuem um custo computacional elevado, ou seja, são muito “pesadas”, e a grande maioria dos sistemas de visão computacional carecem de uma resposta rápida. Dessa forma, o critério de escolha de uma técnica de extração de características deve levar em consideração, além da acurácia, o tempo de resposta exigido. Se por exemplo o sistema necessitar de uma resposta em tempo real, é mais interessante uma técnica mais leve, mesmo que não tenha uma acurácia extremamente satisfatória.

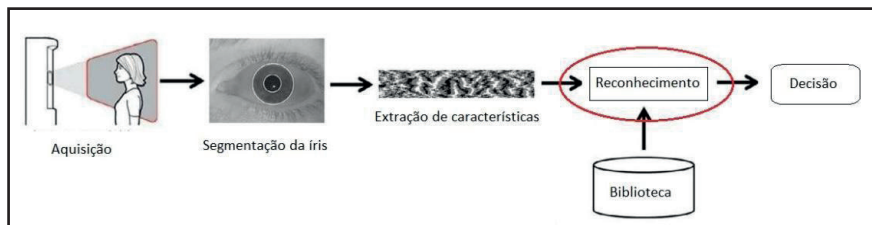
E. Reconhecimento de padrões

Reconhecimento de padrões é uma área da ciência que têm crescido continuamente. Hoje existem inúmeras técnicas que conseguem fazer reconhecimentos precisos, em com um pequeno intervalo de tempo, e com um hardware mínimo. Hoje em dia é muito comum a identificação biométrica, seja em smartphones, em catracas de academia, terminais de atendimento bancário, etc. O reconhecimento pode ser supervisionado ou não supervisionado, a depender da complexidade do caso e da velocidade da resposta.

As Redes Neurais Artificiais, sobretudo a Rede MLP (Multilayer Perceptron),

A Figura a seguir ilustra o processo de reconhecimento de uma íris.

Figura 14 - Esquema de reconhecimento de Íris. Observe que o reconhecimento só ocorre depois que todas as etapas descritas acima são executadas.



Adaptado de Bezerra (2018)

F. Pós-Processamento

Dependendo da forma que os dados sejam apresentados, pode ser necessário uma normalização ao final do processo. Esta etapa deve ser útil para que o usuário comum do sistema possa entender e interpretar com mais clareza a saída do sistema. Como um exemplo simples, o sistema pode indicar uma saída “0” e “1”, mas na verdade, para o usuário, seja mais útil que no resultado apareça **NEGATIVO** ou **POSITIVO**.

3. OPERAÇÕES BÁSICAS

Algumas operações em visão computacional são básicas, e destas não podemos esquecer seu passo a passo. Apresentaremos neste capítulo operações básicas de Visão Computacional utilizando as ferramentas propostas: Python, OpenCV e a IDE Jupyter notebook.

3.1 Carregar e salvar uma imagem

A função mais básica e simples de Visão Computacional consiste em carregar, exibir e salvar uma imagem. Abaixo um código simples de carregamento de uma imagem em escala de cinza.

Note que os primeiros comandos consistem em carregar as bibliotecas cv2, Numpy e matplotlib.

```
import numpy as np

import cv2

from matplotlib import pyplot as plt
```

Após o carregamento das bibliotecas, escrevemos o comando **cv2.imread** para carregar a imagem, e **plt.imshow**, para mostrar a imagem no notebook jupyter. Também utilizamos o comando `print`, para mostrar os parâmetros solicitados.

```
imagem=cv2.imread("dog.jpg")
plt.imshow(imagem)
plt.title('Doguinho'), plt.xticks([ ]), plt.
yticks([ ])
print('Altura em pixels:',end='')
print(imagem.shape[0])
print('Largura em pixels:',end='')
print(imagem.shape[1])
print('Qtde canais:',end='')
print(imagem.shape[2])
```

O comando `print` retorna uma resposta em texto e/ou números, e o comando **plt.imshow** retorna a imagem que foi carregada em **cv2.imread**.

```
Altura em pixels:3264
Largura em pixels:4896
Qtde canais:3
```

A saída do comando **plt.imshow** é mostrada na Figura 15.

É sempre válido lembrar que, para facilitar, tanto o script como a imagem devem estar na mesma pasta. Nos casos em que não é possível (ou então o usuário não queira) script e imagem em uma mesma pasta,

é necessário apontar o caminho da imagem. Mais na frente veremos exemplos de como proceder nesses casos.

Figura 15 - Imagem carregada por meio do comando `cv2.imread`, e mostrado em `plt.imshow`



Fonte: Ungaro (2016)

3.2 Visualização de Pixels e Zoom

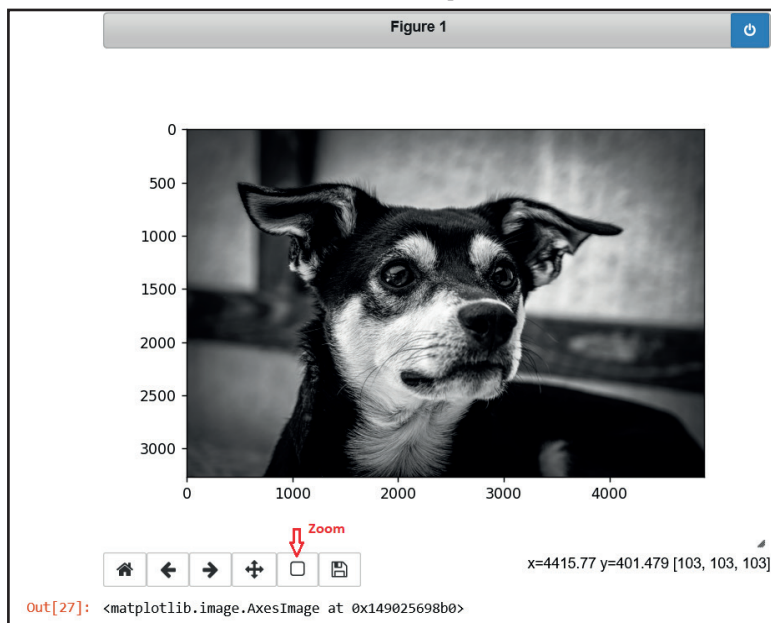
A biblioteca Matplotlib fornece a ferramenta de visualização de informação de pixel e zoom, de uma forma extremamente simples. Ao iniciar o script, logo após a importação das bibliotecas, use:

```
%matplotlib notebook
```

Dessa forma, na região da imagem onde o cursor do mouse apontar, será dada a informação acerca do pixel, tais como: intensidade e posição xy.

Utilizando o mesmo exemplo de 3.1, apenas inserindo o comando `%matplotlib notebook` após a importação das bibliotecas, obtemos o resultado descrito na Figura abaixo:

Figura 16 - Posição de pixel e zoom. Note que o cursor do mouse está em algum lugar da imagem, apontando para a posição descrita na direita. Observe também o valor do pixel: 103



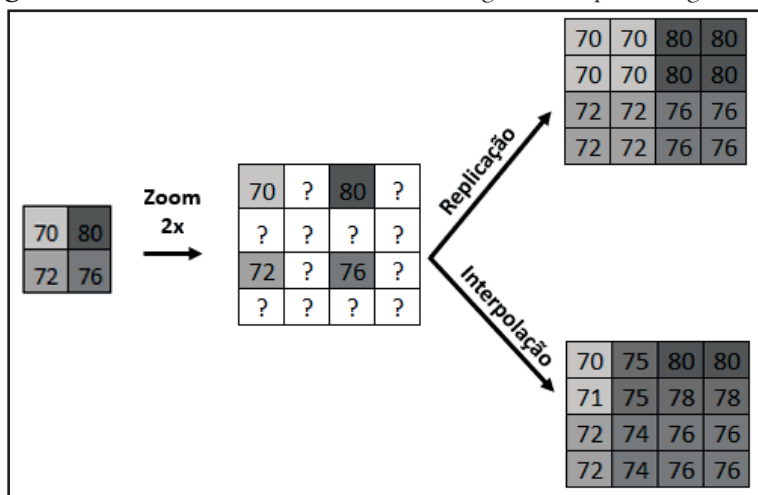
Fonte: Adaptado de Ungaro (2016).

3.3 Cortando, redimensionando e rotacionando uma imagem

Dependendo da aplicação, somente o zoom não será suficiente, sendo necessário fazer o corte da região de maior interesse da imagem. A forma mais simples de realizar o corte consiste em seguir as coordenadas (x,y) dos pixels. No exemplo que faremos aqui, recortaremos uma imagem, na posição `[100:200, 100:200]`, ou seja, o recorte abrangerá os pixels de 101 a 200 no eixo x, e 101 a 200 no eixo y.

Para redimensionar uma imagem, ou seja, aumentar ou diminuir uma imagem, já temos uma função específica para tal, que o comando `resize`. O usuário só deve especificar o novo tamanho da imagem. Convém lembrar que, ao diminuir a imagem, será feita uma média aritmética entre os pixels, de forma a não perder informação. Contudo, caso a imagem seja aumentada, será feita uma interpolação ou uma replicação, com o objetivo de preencher os novos pixels, conforme é observado na Figura abaixo. Ambos os métodos vão trazer perda na qualidade da imagem.

Figura 17 - Redimensionamento de uma imagem 2 x 2 para imagem 4 x 4



Fonte: O autor (2025)

Rotacionar a imagem, sobretudo em seu próprio eixo, pode ser útil, caso, por algum motivo, no momento da aquisição, não tenha sido capturada na orientação desejada. Em alguns casos, durante um teste de detecção de movimento, captura de objetos em uma cena, o programador dispõe de apenas uma imagem para executar o teste. Dessa forma, rotacionar em vários ângulos diferentes pode ser uma alternativa para se produzir mais amostras. No Python, utilizamos o comando `cv2.getRotationMatrix2D` para rotacionar uma imagem sob seu próprio eixo.

A Figura a seguir ilustra o código das três operações descritas

acima. Foi utilizada uma imagem colorida, que foi transformada para escala de cinza, apenas para facilitar o processo. No capítulo específico sobre imagens coloridas aprendemos mais sobre canais, códigos de cores, etc, mas é importante saber que em OpenCV, o canal zero ('0'), significa o canal em escala de cinza de uma imagem, conforme é descrito no código.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img=cv2.imread("coruja.jpg",0)
(a,b)=img.shape
crop=img[100:200,100:200]
novo_tamanho=(a//2,b//2)
img_redimensionada = cv2.resize(img, novo_
tamanho,      interpolation = cv2.INTER_AREA)
#Rotacionar a imagem em 30 graus
centro=(a//2,b//2) # Encontrar o centro da
imagem.
M = cv2.getRotationMatrix2D(centro, 30, 1.0)
#30 graus
img_rotacionada = cv2.warpAffine(img, M, (a,
b))
```

Utilizamos subplots, para mostrar as imagens emparelhadas. Não utilizamos um subplot 2 x 2, porque as imagens possuem tamanhos diferentes, e ainda não possuímos tanta habilidade com subplots, mas no decorrer do curso será bem fácil dominá-la.

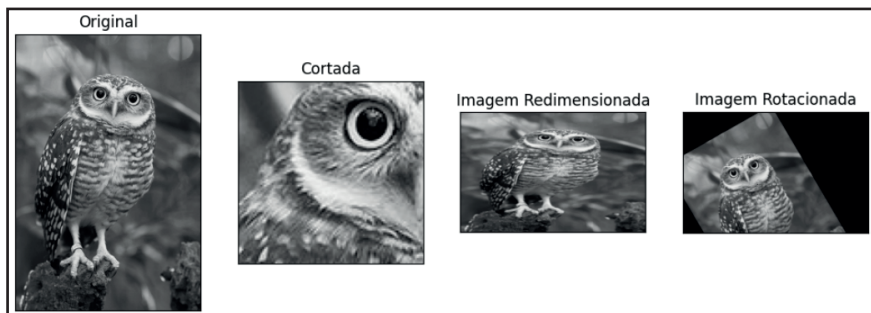
No exemplo da imagem cortada, observa-se que esta pode ser útil para observar algum detalhe específico da imagem de uma forma mais nítida. Já no exemplo do redimensionamento, foi realizado uma deformação proposital, para mostrar que se a imagem é redimensionada para mais, vai ocorrer perda de qualidade. A rotação pode ser útil em técnicas que exijam, por exemplo, treinamento de sistemas de detecção de movimento.

```
plt.figure(figsize=(12,18))
plt.subplot (1,4,1)
plt.imshow(img, cmap=plt.cm.gray)
plt.title('Original'),plt.xticks([ ]), plt.
```

```
yticks([ ])
plt.subplot (1,4,2)
plt.imshow(crop,cmap=plt.cm.gray),plt.
xticks([ ]), plt.yticks([ ])
plt.title('Cortada'),plt.xticks([ ]), plt.
yticks([ ])
plt.subplot (1,4,3)
plt.imshow(img_redimensionada,cmap=plt.
cm.gray)
plt.title('Imagem Redimensionada'),plt.
xticks([ ]), plt.yticks([ ])
plt.subplot (1,4,4)
plt.imshow(img_rotacionada,cmap=plt.cm.gray)
plt.title('Imagem Rotacionada'),plt.xticks([
]), plt.yticks([ ])
```

O resultado final do código é mostrado na Figura 18 a seguir:

Figura 18 - Imagem original cortada, redimensionada e rotacionada



Fonte: Adaptado de Parque das aves (2022)

4. TRANSFORMAÇÕES DE INTENSIDADE

Após o processo de captura de uma imagem, às vezes faz-se necessário algum tipo de melhoramento, para que as etapas seguintes do processo sejam mais satisfatórias. Geralmente, esse melhoramento de imagem, do inglês *image enhancement*, acontece na etapa de pré-processamento, conforme visto em 2.3 (a). Na prática, tal melhoramento pode ser feito pixel a pixel, ou na matriz inteira. Neste momento, vamos abordar técnicas de transformações de intensidade no domínio discreto, ou seja, a transformação ocorrerá em domínio espacial, pixel a pixel.

Na prática, a expressão geral para a **Função de Transformação** nos níveis de cinza pode ser dada por:

$$g(x,y)=T[f(x,y)] \quad (1)$$

Sendo $f(x,y)$ a imagem de entrada e $g(x,y)$ a imagem de saída ou a imagem processada. T é um operador de f .

Neste capítulo serão apresentadas diversas técnicas de transformação de intensidade dos pixels de uma imagem, de forma a obter algum tipo de melhoramento. As técnicas a serem estudadas são: negativo da imagem, transformação logarítmica, transformação gamma, limiarização de contraste, e equalização de histograma. A seguir serão apresentadas sobre cada uma delas.

4.1 Negativo da imagem

Reverter os níveis de intensidade de uma imagem produz o equivalente a um negativo fotográfico, que, dependendo do caso, pode ser útil para realçar detalhes brancos ou cinza, incorporados a regiões escuras da imagem, sobretudo quando regiões escuras possuem grande extensão.

O negativo de uma imagem com níveis de intensidade na faixa $[0,L-1]$ é obtido utilizando a expressão abaixo:

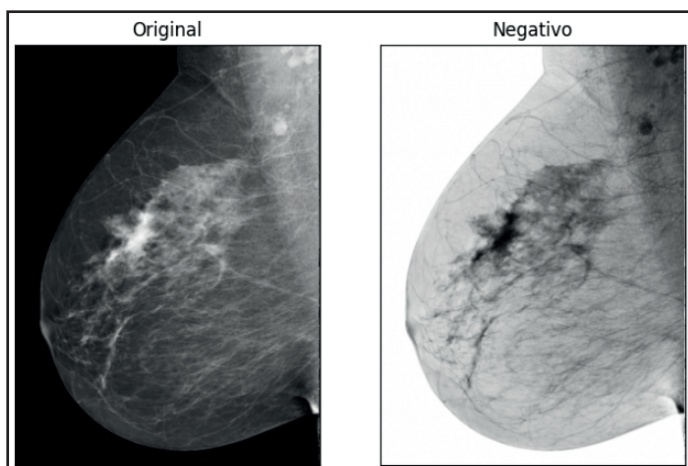
$$s=L-1-r \quad (2)$$

Sendo L o maior valor de intensidade de pixel da codificação da imagem, r o valor do pixel, e s o pixel transformado. A maioria das aplicações de processamento de imagens e visão computacional utiliza a codificação de 8 bits, ou seja, a imagem tem 256 níveis de cinza. O código para obter o negativo da imagem é descrito abaixo:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img=cv2.imread('mama.jpeg',0)
neg = 255 - img
plt.figure(figsize=(8,5))
plt.subplot(1,2,1)
plt.imshow(img,cmap=plt.cm.gray)
plt.title('Original'),plt.xticks([ ]), plt.yticks([ ])
plt.subplot(1,2,2)
plt.imshow(neg,cmap=plt.cm.gray)
plt.title('Negativo'),plt.xticks([ ]), plt.tricks([ ])
plt.show( )
```

A Figura 19 apresenta a comparação entre a imagem original e sua versão negativa, obtida por meio do código acima:

Figura 19 - Imagem original negativa, obtida por meio do código descrito em 4.1



Fonte: Adaptado de Suckling (1994)

4.2 Transformação Logarítmica

Em algumas aplicações, em que a faixa de detalhe das imagens é muito grande, e praticamente ficam comprimidas em 256 níveis de cinza, é necessário aplicar alguma técnica de transformação de intensidade, para que mais detalhes possam ser visualizados. Como exemplo a Figura 20 abaixo mostra a visualização 2 - D da imagem original de um espectro de Fourier, logicamente no domínio da frequência. Pode-se observar poucos detalhes, já que os valores de um espectro de Fourier variam de 0 a 10^6 . Para este caso, podemos aplicar uma transformação logarítmica nos pixels, que possui com objetivo principal de comprimir a faixa dinâmica das imagens com grandes variações de detalhes. Após a Transformação Logarítmica, é visível a riqueza de detalhes agora mostrados, fruto do melhoramento que a imagem foi submetida.

A Fórmula geral da Transformação Logarítmica pode ser dada por:

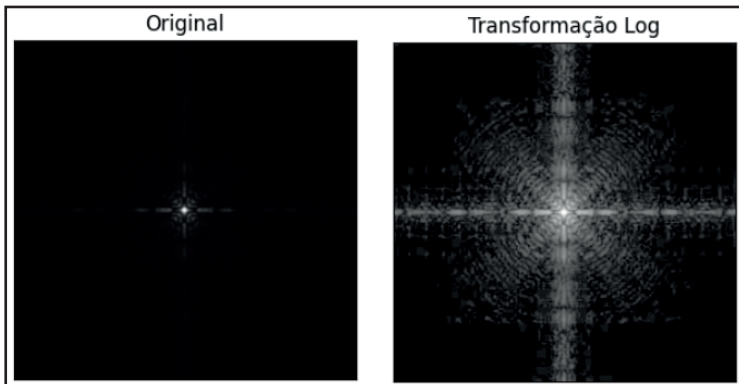
$$s=c.\log(1+r) \quad (3)$$

Sendo r o valor do pixel, c uma constante e s o pixel transformado.

O código abaixo realiza a transformação logarítmica em um espectro de fourier qualquer

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img=cv2.imread("fourier.jpg",0)
img2=np.array(img, dtype = np.float64)
log1 = 10 * np.log(1 + img2)
log2 = np.array(log1, dtype = np.uint8)
plt.figure(figsize=(15,8))
plt.subplot (2,2,1)
plt.imshow(img,cmap=plt.cm.gray)
plt.title('Original'), plt.xticks([ ]), plt.
yticks([ ])
plt.subplot (2,2,2)
plt.imshow(log2,cmap=plt.cm.gray)
plt.title('Transformação Log'), plt.xticks([
]), plt.yticks([ ])
plt.show( )
```

Figura 20 - Melhoramento de espectro de Fourier por transformação logarítmica



Fonte: O autor (2025)

4.3 Transformação Gamma

A transformação gamma é uma transformação de potência, da forma:

$$s = c \cdot r^\gamma \quad (4)$$

Sendo c e γ constantes positivas, r o valor do pixel, e s o pixel transformado.

A correção gamma é amplamente empregada para compensar não linearidades de dispositivos de exibição. Imagens que não são adequadamente corrigidas podem aparecer desbotadas ou com brilho excessivo. A correção gamma pode ser realizada através de níveis de controle na tela do computador, como uma forma de realce da imagem, subjetivo ao critério do observador.

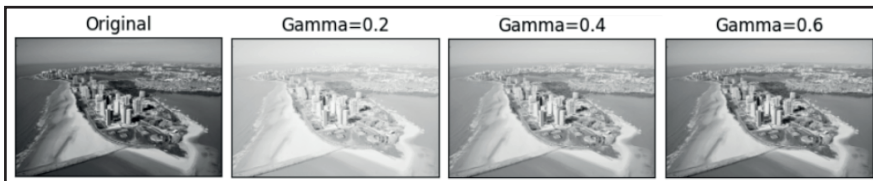
O código a seguir aplica a correção gamma com três valores distintos em uma imagem.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img=cv2.imread("gamma.jpg",0)
```

```
gamma = 0.2
img_1=np.power(img,gamma)
gamma = 0.4
img_2=np.power(img,gamma)
gamma = 0.6
img_3=np.power(img,gamma)
plt.figure(figsize=(12,8))
plt.subplot(4,2,1)
plt.imshow(img,cmap=plt.cm.gray)
plt.title('Original'),plt.xticks([]), plt.
yticks([])
plt.subplot(4,2,2)
plt.imshow(img_1,cmap=plt.cm.gray)
plt.title('Gamma=0.2'),plt.xticks([]), plt.
yticks([])
plt.subplot(4,2,3)
plt.imshow(img_2,cmap=plt.cm.gray)
plt.title('Gamma=0.4'),plt.xticks([]), plt.
yticks([])
plt.subplot(4,2,4)
plt.imshow(img_3,cmap=plt.cm.gray)
plt.title('Gamma=0.6'),plt.xticks([]), plt.
yticks([])
plt.show()
```

A imagem original e suas respectivas transformações gamma são mostradas a seguir:

Figura 21 - Aplicação da Correção Gamma



Fonte: Adaptado de Meireles Junior (2014).

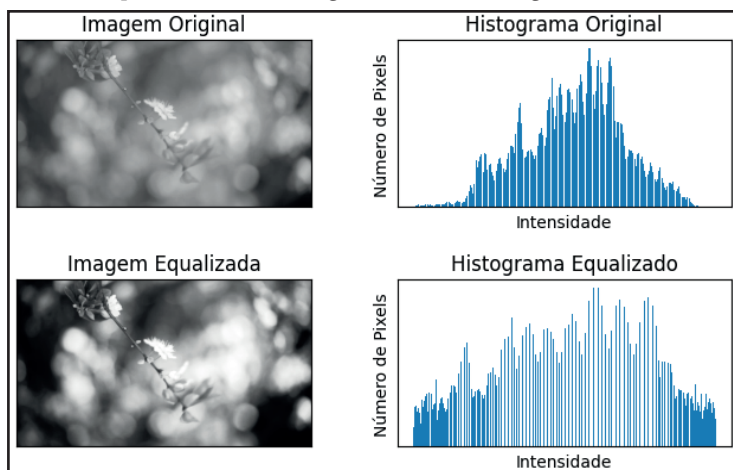
4.4 Equalização de Histograma

Um histograma é um gráfico de colunas ou de linhas que representa a distribuição dos valores dos pixels de uma imagem, ou seja, a quantidade de pixels mais claros (próximos de 255) e a quantidade de pixels mais escuros (próximos de 0).

O eixo X do gráfico normalmente possui uma distribuição de 0 a 255 que demonstra o valor (intensidade) do pixel e no eixo Y é plotada a quantidade de pixels daquela intensidade.

A Figura 22(a) ilustra uma imagem, e seu histograma em 22(b). Observe que existe uma maior intensidade de pixels na região entre 30 e 40 do gráfico, aproximadamente. Tais pixels possuem a intensidade muito escura, o que é observado inclusive nos tons da imagem 22 (a). O objetivo de equalizar o histograma de uma imagem consiste basicamente em distribuir uniformemente os pixels ao longo do eixo x, proporcionando uma imagem com alto contraste e melhor visualização de detalhes, o que pode ser observado na imagem 22 (c) e no seu histograma equalizado em 22 (d).

Figura 22 - Equalização de Histograma de uma imagem de baixo contraste.



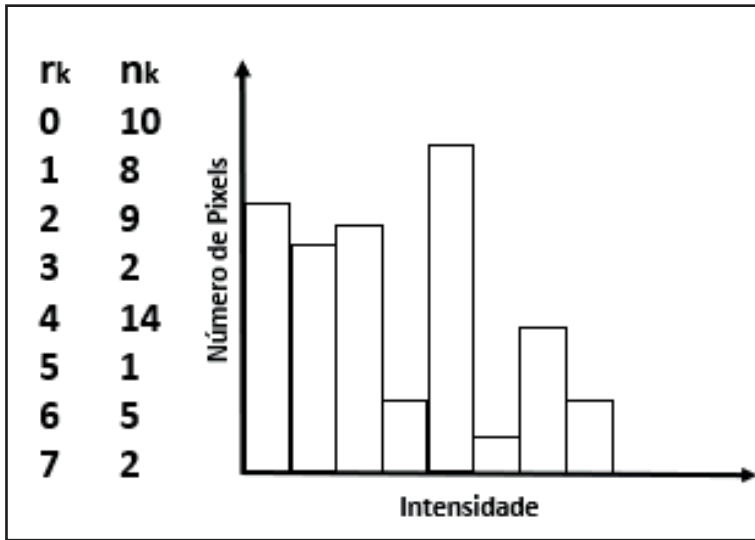
Fonte: Adaptado de Htoon (2020)

4.4.1 Passos para a equalização de um histograma

Precisamos recorrer a algumas equações matemáticas da probabilidade e estatística para realizar a equalização. A seguir os passos. Considere r o valor do pixel, n a quantidade pixels da imagem, e n_k a ocorrência de um pixel na distribuição total.

A partir de um exemplo gráfico, o procedimento pode ser compreendido por meio do exemplo apresentado. A Figura 23 ilustra o histograma de uma determinada imagem, juntamente com os seus rk 's e nk 's.

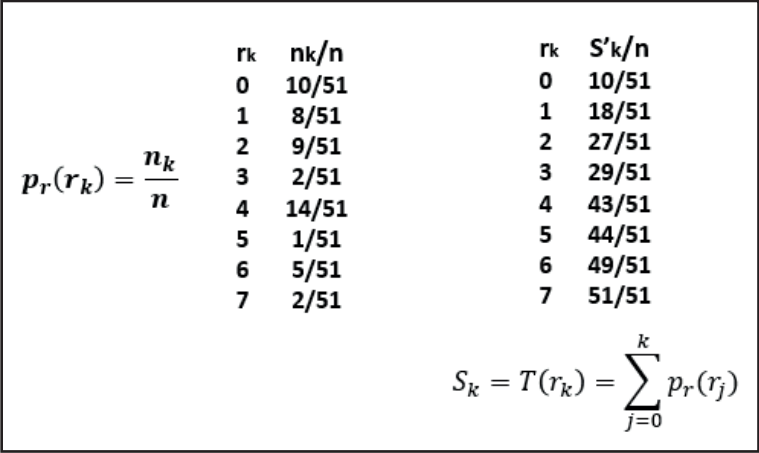
Figura 23 - Histograma original da imagem



Fonte: O autor (2025)

A Figura acima nos indica que o pixel de valor 0 ocorreu 10 vezes, o pixel 1 ocorreu 8 vezes, e assim por diante. Dessa forma, pode-se dizer que o número total de pixels, ou seja, o nosso $n=51$. Dessa forma, calcularemos a probabilidade da imagem sobre o valor de cada pixel, e a distribuição cumulativa da imagem conforme mostrado na Figura abaixo.

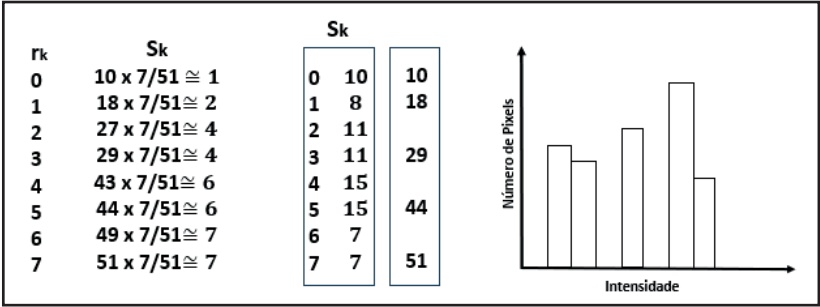
Figura 24 - Valores de intensidade de pixel - rk's, suas ocorrências e as suas distribuições cumulativas sk's



Fonte: O autor (2025)

Logo em seguida, faz-se a multiplicação da distribuição cumulativa pelo maior valor de pixel, no caso, 7, e logo após, o novo mapeamento dos níveis de cinza, conforme ilustrado na Figura 25

Figura 25 - Multiplicação da distribuição cumulativa (a) e novo mapeamento (b). Histograma equalizado (c)



Fonte: O autor (2025)

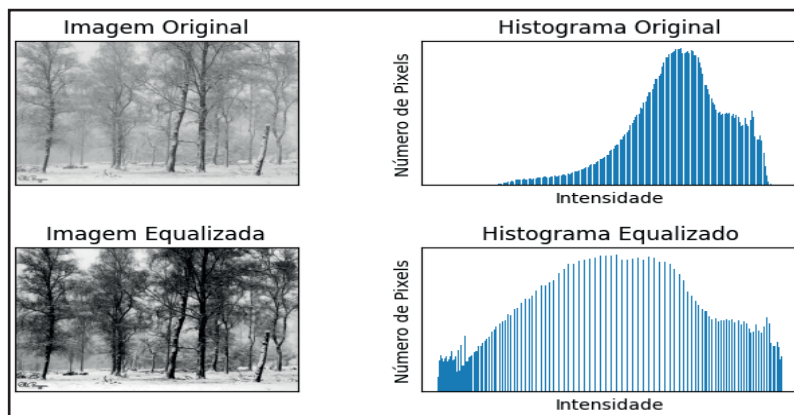
Em seguida, a distribuição cumulativa é multiplicada pelo maior

valor de intensidade de pixel, que, neste exemplo, corresponde a 7, resultando no novo mapeamento dos níveis de cinza.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('lowcontrast1.jpg', 0)
h_eq = cv2.equalizeHist(img)
imgEqualizada = cv2.equalizeHist(img)
plt.figure(figsize=(8, 4))
plt.subplot (2, 2, 1)
plt.imshow(img, cmap=plt.cm.gray)
plt.title('Imagem Original'), plt.xticks([]),
plt.yticks([])
plt.subplot (2, 2, 2)
plt.title("Histograma Original"), plt.
xticks([], plt.yticks([])
plt.xlabel("Intensidade")
plt.ylabel("Número de Pixels")
plt.hist(img.ravel(), 256)
plt.figure(figsize=(8, 4))
plt.subplot (2, 2, 1)
plt.imshow(imgEqualizada, cmap=plt.cm.gray)
plt.title('Imagem Equalizada'), plt.
xticks([], plt.yticks([])
plt.subplot (2, 2, 2)
plt.title("Histograma Equalizado"), plt.
xticks([], plt.yticks([])
plt.xlabel("Intensidade")
plt.ylabel("Número de Pixels")
plt.hist(imgEqualizada.ravel(), 256)
plt.show ()
```

A Figura 26 ilustra a imagem original, e a imagem equalizada. Pode-se observar mais uma vez a diferença entre os histogramas, evidenciando claramente que a equalização realizou melhoramento de intensidade na imagem

Figura 26 - Equalização de Histograma em Python



Fonte: Adaptado de Borggreve (2019)

4.5. Equalização adaptativa

Uma variação da técnica tradicional de Equalização de Histograma é a Equalização Adaptativa de Histograma, que divide a imagem em pequenas regiões, chamadas de “regiões contextuais”. Dessa forma, em vez de trabalhar com a imagem inteira, tal como a equalização de histograma tradicional, a equalização adaptativa modifica os pixels baseados em uma pequena região contextual, com poucos vizinhos. Os pixels localizados nas regiões limítrofes de cada região são combinados utilizando interpolação bilinear, a fim de evitar bordas, artificialmente induzidas (PIZER; AMBURN, 1987).

O resultado da Equalização Adaptativa é uma imagem com contraste local ampliado, apresentando mais detalhes. A grande desvantagem é que o ruído/saturação gerado em regiões de baixo contraste é ampliado junto com a imagem.

Uma alternativa para resolver a desvantagem da Equalização Adaptativa é limitar o contraste, modificando o aclave da função de transformação do histograma e redistribuir os pixels novamente, conforme ilustra a Figura 9 (PIZER, AMBURN, 1987; SUNDARAMI et al. 2011).

O algoritmo mais usual que realiza a limitação do contraste em histogramas pode ser descrito pela fórmula abaixo: (RAI et al., 2012)

$$g = g_{min} - \left(\frac{1}{\alpha}\right) * \ln[1 - \arg(G_j)] \quad (5)$$

Sendo:

g = Valor do pixel atualizado

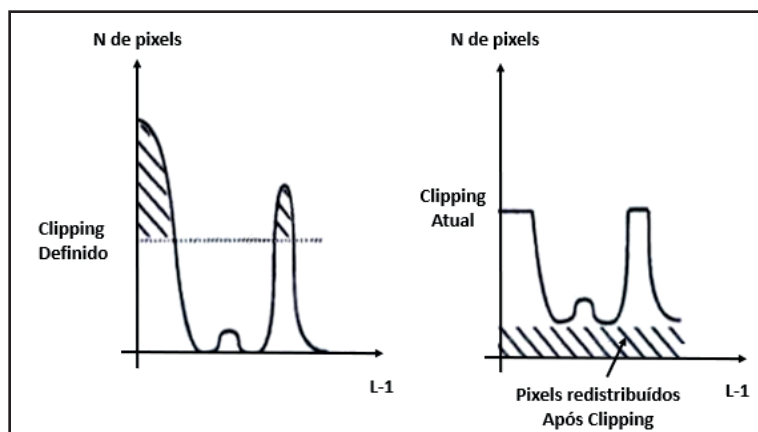
g_{min} = Valor do menor pixel

G_j = Função de Distribuição cumulativa

α = Clip limit

A Figura 27 ilustra a nova distribuição dos *pixels* de um histograma gerado através de equalização adaptativa após a utilização da técnica de limitação de contraste, também chamada de *Contrast Limited Adaptive Histogram Equalization* – CLAHE. Pode-se notar que um novo valor de *clip* é gerado, pois os *pixels* situados acima do *clip* são remapeados ao longo do eixo de intensidade L-1.

Figura 27 – Ilustração de redistribuição de *pixels* após aplicação de técnicas de limitação de contraste. O gráfico à esquerda ilustra um histograma equalizado, apresentando um limiar (aqui chamado de clipping). O gráfico à direita ilustra a redistribuição dos *pixels* ao longo do eixo de intensidade ($L-1$), e apresenta o *clipping* atual, modificado devido a nova distribuição



Fonte: O autor (2025)

4.5.1. Alargamento de Contraste

Alargamento de contraste é uma das mais simples transformações não-lineares definidas por partes. Imagens de baixo contraste resultam em baixa riqueza de detalhes, sendo de difícil visualização e interpretação. O Alargamento de Contraste é uma técnica que expande uma determinada faixa de níveis de intensidade de modo a incluir todo o intervalo, sendo que a expansão (alargamento) ocorre somente no intervalo determinado pelos parâmetros da função de transformação linear por partes.

A Figura 28 mostra uma transformação típica utilizada para o alargamento de contraste. As posições dos pontos ($r1$ e $s1$) e ($r2$ e $s2$) controlam o formato da função de transformação.

Figura 28 – Função de transformação não linear para alargamento de contraste



Fonte: O autor (2025)

Uma aplicação prática desta técnica, pode ser observada na tarefa de detecção de nódulos de mamas densas. O código em python apresentado abaixo é capaz de segmentar somente a região suspeita de câncer (de acordo com o laudo da imagem), que está totalmente envolvida por tecido mamário de baixo contraste.

A imagem de teste do nosso algoritmo é apresentada na Figura 29, composta de uma mama com grande quantidade de tecido fibroglandular, que torna difícil a detecção de qualquer região suspeita de qualquer anormalidade (Nissan, 2025).

Figura 29 – Mamografia com alta densidade, evidenciando baixo contraste entre tecidos



Fonte: Suckling (1994)

O Código abaixo executa a técnica de equalização adaptativa, juntamente com o alargamento de contraste, para identificar o tecido suspeito de anormalidade, na mamografia apresentada acima.

O primeiro trecho do código é responsável por importar as bibliotecas, e carregar a imagem, em escala de cinza.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img=cv2.imread('mom_clahe.jpg',0)
```

A seguir, é aplicado o comando `cv2.createCLAHE` para realizar a equalização, onde devemos inserir os valores de `clipLimit` e `tileGridSize` adequados.

```
clahe = cv2.createCLAHE(clipLimit=2.0,  
tileGridSize=(8,8))  
  
cl = clahe.apply(img)
```

Após a equalização do histograma, é necessário realizar o alargamento de contraste, conforme trecho a seguir:

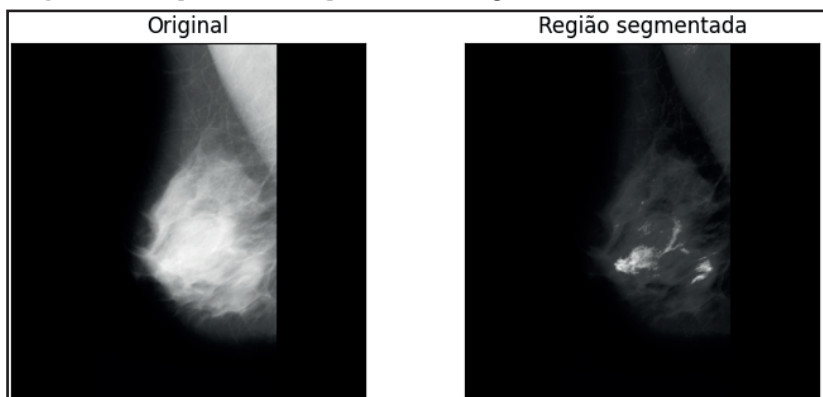
```
def pixelVal(pix, r1, s1, r2, s2):  
    if (0 <= pix and pix <= r1):  
        return (s1 / r1)*pix  
    elif (r1 < pix and pix <= r2):  
        return ((s2 - s1)/(r2 - r1)) * (pix - r1) +  
        s1  
    else:  
        return ((255 - s2)/(255 - r2)) * (pix - r2)  
        + s2  
# Definindo valores de r1, r2, s1, s2  
r1 = 215  
s1 = 100  
r2 = 300  
s2 = 800  
pixelVal_vec = np.vectorize(pixelVal)  
contrast_stretched = pixelVal_vec(cl, r1,  
s1, r2, s2)
```

Por fim, os resultados são apresentados no subplot abaixo:

```
plt.figure(figsize=(12,8))  
plt.subplot (2,2,1)  
plt.imshow(img,cmap=plt.cm.gray)  
plt.title('Original'),plt.xticks([]), plt.  
yticks([])  
plt.subplot (2,2,2)  
plt.imshow(contrast_stretched,cmap=plt.  
cm.gray)  
plt.title('Região segmentada'),plt.  
xticks([]), plt.yticks([])  
plt.show()
```

A Figura a seguir ilustra a mamografia original (a), e a mamografia segmentada após o alargamento de contraste (b). As áreas que ficaram em evidência, ou seja, mais claras, são as regiões onde é provável que tenha um achado, ou seja uma lesão. Cabe ao radiologista a interpretação, e cruzamento com exames complementares.

Figura 30 - Equalização adaptativa de histograma e limitação de contraste



Fonte: O autor (2026)

5. FILTRAGEM ESPACIAL

O capítulo 4 nos mostrou técnicas, onde é possível obter o melhoramento de imagens, alterando a intensidade dos pixels, de forma individual. Foi visto, por exemplo, na técnica de Equalização de Histograma, como o contraste de uma imagem pode ser melhorado, apenas com um novo mapeamento dos níveis de cinza. Contudo, quando se é exigido um melhoramento mais robusto, ou uma modificação de um conjunto de pixels por vez, se tornam necessárias outras metodologias de melhoramento de imagem.

A Filtragem espacial é uma espécie de transformação pixel a pixel também, mas não depende somente da intensidade de nível de cinza de um determinado pixel, mas também do valor dos níveis de cinza dos pixels vizinhos, e tal processo é realizado utilizando matrizes denominadas máscaras ou kernels, as quais são aplicadas sobre a imagem.

Quando se fala “filtragem de imagens”, a primeira ideia que vem em nossa mente é que a imagem possui algum tipo de imperfeição ou ruído, e as técnicas de filtrações são capazes de eliminá-los, tornando a imagem mais próxima da original. Essa na verdade é apenas uma das atribuições da filtragem espacial, que também tem como objetivo, além da filtragem de ruídos, a suavização de imagens, aguçamento e detecção de bordas.

A Figura 31 abaixo ilustra 4 exemplos de aplicação de Filtragem Espacial em imagens. Em 31 (a) temos uma imagem de ultrassonografia, que foi capturada com bastante ruído e ao lado a imagem já filtrada, com muito melhor contraste e riqueza de detalhes visíveis.

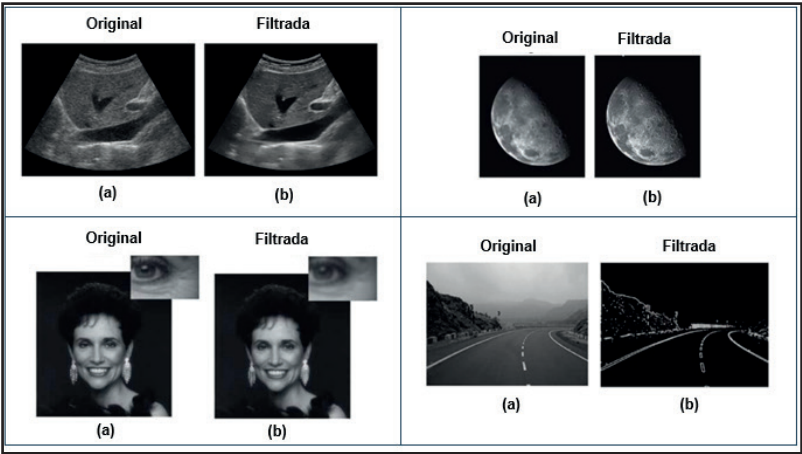
Na Figura 31 (b) observa-se uma aplicação de filtragem espacial em realce de imagens. Pode-se ver que a imagem original possui alguns detalhes indesejáveis, e que na imagem filtrada foram suavizados, ficando quase imperceptíveis.

Aguçamento de detalhes de uma imagem também pode ser realizado com filtragem espacial, conforme a Figura 31 (c). Pode-se observar mais detalhes nas crateras da imagem filtrada da lua, sobretudo nos seus contornos, em comparação a imagem original.

O último exemplo, é uma aplicação de detecção de bordas, visto na Figura 31 (d). A imagem filtrada possui apenas as bordas de visualização, em relação à imagem original. Esta aplicação é muito útil em robótica, em reconhecimento de objetos, faces, etc...

A Filtragem Espacial é realizada através de uma operação matemática chamada Convolução. Esta operação matemática consiste na aplicação de uma máscara (que aqui chamaremos de kernel) sobre a imagem, gerando uma nova imagem como resultado. O kernel também é uma matriz de dimensão bem menor que a imagem, e na prática, ele “desliza” sobre toda a imagem, modificando-a. Para um dado deslocamento, os elementos do kernel são multiplicados pelos pixels da imagem, que, por fim, são somados e resultam em um elemento da imagem convolucionada.

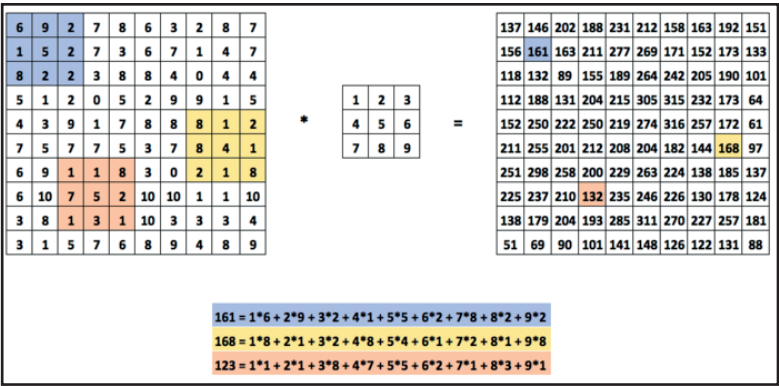
Figura 31 - Exemplos de aplicação de Filtragem Espacial



Fonte: Adaptado de Gil (2023), Yelmanov (2020), Fathima (2019)

A Figura 32 demonstra a convolução pixel a pixel, com alguns exemplos.

Figura 32 - Convolução de uma matriz 9x9 com um kernel 3x3



Fonte: Adaptado de Barbosa & Bonadio (2023)

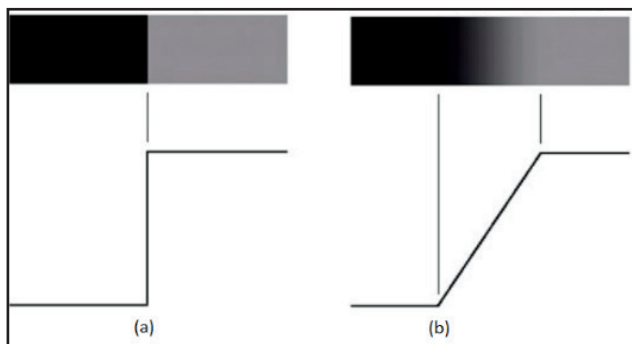
É válido ressaltar que o custo computacional da convolução é bastante elevado. Podemos calcular que, em uma imagem de 512x512, aplicando a convolução com um kernel de 16x16, por exemplo, são necessárias 67.108.864 multiplicações. Os processadores multinúcleos de hoje conseguem resolver essa grande quantidade de multiplicações em poucos segundos, mas na prática, a grande maioria dos processadores digitais de imagem realizam operações no domínio da frequência e não necessitam da convolução (esses conceitos serão aprofundados em capítulo posterior.).

Os filtros espaciais que exibem bordas com transição mais suave nas imagens resultantes são chamados de filtros de suavização. Já os filtros que aguçam, ou seja, realçam mais as bordas das imagens são chamados de filtros de aguçamento.

Na prática, as bordas de uma imagem, ou de um objeto contido em uma imagem, é uma região de transição abrupta de frequência. Os filtros que suavizam as bordas também podem ser chamados de filtros de passa-baixas, e os que aguçam ou realçam, passa-altas.

A Figura 33 abaixo ilustra dois tipos de bordas. Em 33 (a) observa-se uma borda de alta frequência, ou seja, a transição foi feita de forma abrupta do preto para o tom cinza rapidamente. 33(b) mostra uma transição mais lenta do preto para o tom cinza, passando por diversos tons intermediários. Os filtros capazes de transformar uma borda qualquer em uma borda similar a 33(a) são os filtros de aguçamento de bordas, e os capazes de transformar bordas similares a 33(b) são os filtros de suavização.

Figura 33 - Borda de (a) alta frequência e (b) baixa frequência



Fonte: MARABA (2020)

No capítulo presente, primeiramente trataremos de estudar os filtros de suavização. Serão apresentados os filtros da Mediana, da Média e Gaussiano. Logo em seguida estudaremos os filtros de aguçamento. Será a vez de estudar os filtros de Sobel, Laplaciano e Canny. Existe uma infinidade de filtros, para as mais diversas aplicações. Neste material, estudaremos os filtros mais simples, e à medida que nosso conhecimento for avançando, seremos apresentados com mais uma gama de filtros espaciais.

5.1. Filtros de Suavização

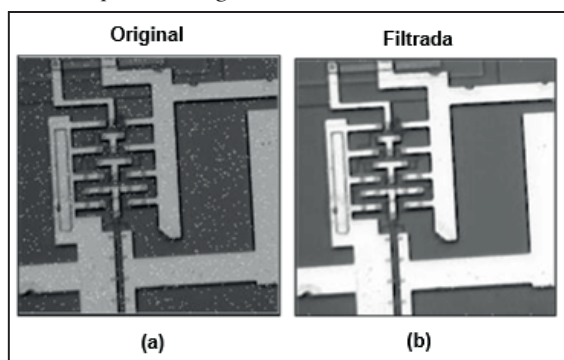
Apresentaremos neste tópico 3 filtros, sendo o primeiro um filtro de ordenação de pixels, sendo classificado com um filtro não-linear, e os 2 seguintes filtros lineares, que usam kernels e convolução.

5.1.1. Filtro da Mediana

O filtro da mediana ordena a intensidade dos pixels dentro da área da máscara em ordem crescente ou decrescente, alocando ao pixel da imagem correspondente à posição central da máscara, o valor da intensidade do pixel que corresponde à posição intermediária do respectivo intervalo ordenado.

A Figura 34 ilustra a imagem ruidosa de uma placa de circuito impresso, e a mesma filtrada, com o filtro da mediana, com máscara de 3x3.

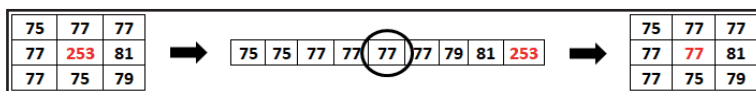
Figura 34 - Exemplo de imagem ruidosa tratada com filtro da mediana



Fonte: Adaptado de Xian et al. (2017)

Na Figura 35 um exemplo da aplicação do filtro da mediana na imagem 34. A imagem foi dividida em janelas de 3x3, onde em algumas dessas janelas foi possível observar um pixel com o valor totalmente diferente dos vizinhos, o que pode caracterizar um pixel de ruído. Dessa forma, as janelas 3x3 foram organizadas sob a forma de um vetor linha, os pixels foram organizados em ordem crescente, e logo em seguida foi alocado o pixel correspondente à posição central do vetor para a posição central da janela, justamente onde se encontrava o pixel ruidoso. Essa operação minimiza o ruído, já que elimina os pixels potencialmente ruidosos. No entanto, nas janelas onde não existem pixels ruidosos, a operação é feita da mesma forma, o que pode acarretar na diminuição do realce das bordas da imagem.

Figura 35 - Filtro da mediana em operação



Fonte: O autor (2026)

5.1.2. Filtro da Média

O filtro da média entrega uma imagem com suavização de bordas, sendo essa suavização discreta ou robusta, a depender do tamanho do kernel. No filtro da média, o kernel possui tamanho $n \times n$, com todos os seus coeficientes iguais a 1, e dividindo –se o valor obtido pelo número de pixels por n^2 .

A imagem resultante de um filtro espacial linear de suavização, tal como o filtro da média, é simplesmente a média dos pixels contidos na vizinhança do kernel.

A ideia por trás do filtro é simples. Substituir o valor de cada pixel de uma imagem pela média dos níveis de intensidade da vizinhança definido pelo kernel, que resultará em uma imagem com perda de nitidez, ou seja, com redução de transições abruptas, nas bordas da imagem.

Pelo fato do ruído aleatório normalmente consistir em transições abruptas nos níveis de intensidade, uma das aplicações do filtro é a redução do ruído, possuindo o efeito indesejado de “suavizar excessivamente.”, ou seja, suavizar as bordas.

É válido ressaltar que, quanto maior o kernel, mais borrada a imagem resultante.

A Figura abaixo possui um exemplo de 3 kernels de filtros da média, de tamanhos 3x3, 5x5 e 7x7.

Figura 36 - Kernels com tamanho 3x3, 5x5 e 7x7, respectivamente.

$$Z = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad Z = \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad Z = \frac{1}{49} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Fonte: O autor (2026).

5.1.3. Filtro Gaussiano

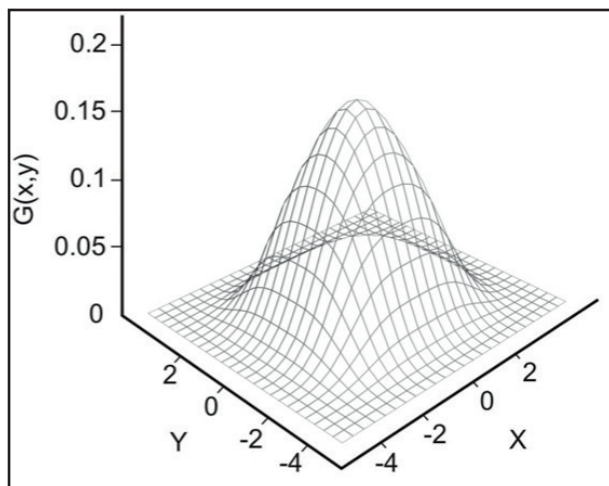
O Filtro Gaussiano é um filtro do tipo linear, que também é utilizado como um filtro de suavização, por meio de kernels produzidos a partir da aproximação digital 2-D da função gaussiana, dada por:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \left(e^{\frac{-(x^2 + y^2)}{2\sigma^2}} \right) \quad (6)$$

Sendo o desvio padrão sobre os eixos x e y.

A Figura 37 ilustra o gráfico da curva de Gauss 2-D

Figura 37 - Representação da função Gaussiana em 2D com média em (0,0) e desvio padrão $\sigma = 1$



Fonte: O autor (2026)

A Figura a seguir possui um exemplo de 3 kernels de filtros da média, de tamanhos 3x3, 5x5.

Figura 38 - Kernels com tamanho 3x3, 5x5, respectivamente.

$$Z = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad Z = \frac{1}{273} \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Fonte: O autor (2026)

O maior efeito colateral de um filtro de suavização é justamente suavizar as regiões de bordas da imagem, causando um efeito de

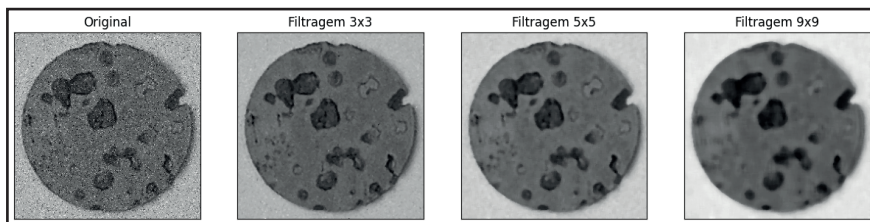
borramento que, dependendo do tamanho do kernel aplicado, pode ser mais evidente ou não. Neste subitem, não entramos muito no estudo de ruídos, embora estejamos tratando de filtros. Nossa ideia aqui é somente de apresentar os tipos mais usuais de filtros de suavização, e a parte relativa à aplicação destes filtros sobre uma imagem com ruído, será feita no capítulo sobre **Restauração de Imagens**, que veremos mais à frente.

Os códigos a seguir executam aplicações em python com os filtros: Mediana, média, e gaussiano, respectivamente.

```
import numpy as np
from matplotlib import pyplot as plt
img=cv2.imread("median_test.jpg",0)
imgr1 = cv2.medianBlur(img, 3) # Filtragem
com kernel de 3x3
imgr2 = cv2.medianBlur(img, 5) # Filtragem
com kernel de 5x5
imgr3 = cv2.medianBlur(img, 9) # Filtragem
com kernel de 9x9
plt.figure(figsize=(15,10))
plt.subplot (1,4,1)
plt.imshow(img,cmap=plt.cm.gray)
plt.title('Original'),plt.xticks([]), plt.
yticks([])
plt.subplot (1,4,2)
plt.imshow(imgr1,cmap=plt.cm.gray)
plt.title('Filtragem 3x3'),plt.xticks([]),
plt.yticks([])
plt.subplot (1,4,3)
plt.imshow(imgr2,cmap=plt.cm.gray)
plt.title('Filtragem 5x5'),plt.xticks([]),
plt.yticks([])
plt.subplot (1,4,4)
plt.imshow(imgr3,cmap=plt.cm.gray)
plt.title('Filtragem 9x9'),plt.xticks([]),
plt.yticks([])
```

A Figura 39 ilustra o resultado obtido com o código acima. Pode-se observar que, quanto maior o kernel, maior a ação do filtro, e consequentemente, menor o contraste final.

Figura 39 - Exemplo de Filtro da mediana em Python



Fonte: Adaptado de SSIM(2021)

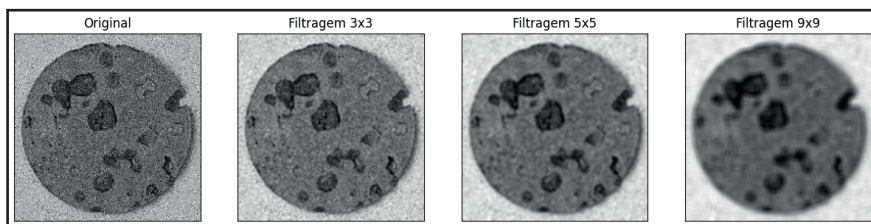
Baseado no código apresentado para filtragem da mediana, a seguir uma aplicação com filtro da média. Neste código, apenas foi alterado a função da filtragem, de `cv2.medianBlur(img, n)` para `cv2.blur(img, (n,n))`, onde `n` sinaliza o tamanho do kernel.

```
import numpy as np
from matplotlib import pyplot as plt
img=cv2.imread("median_test.jpg",0)
imgr1 = cv2.blur(img, (3,3)) # Filtragem com
kernel de 3x3
imgr2 = cv2.blur(img, (5,5)) # Filtragem com
kernel de 5x5
imgr3 = cv2.blur(img, (9,9)) # Filtragem com
kernel de 9x9
plt.figure(figsize=(15,10))
plt.subplot (1,4,1)
plt.imshow(img,cmap=plt.cm.gray)
plt.title('Original'),plt.xticks([]), plt.
yticks([])
plt.subplot (1,4,2)
plt.imshow(imgr1,cmap=plt.cm.gray)
plt.title('Filtragem 3x3'),plt.xticks([]),
plt.yticks([])
plt.subplot (1,4,3),plt.xticks([]), plt.
yticks([])
plt.imshow(imgr2,cmap=plt.cm.gray)
plt.title('Filtragem 5x5'),plt.xticks([]),
plt.yticks([])
plt.subplot (1,4,4)
```

```
plt.imshow(imgr3, cmap=plt.cm.gray)
plt.title('Filtragem 9x9'), plt.xticks([]),
plt.yticks([])
```

A Figura 40 ilustra o resultado obtido com o filtro da média. Podemos observar uma diferença sutil entre o filtro da média e o filtro da mediana, apresentado anteriormente, sobretudo no resultado com kernel de 9x9.

Figura 40 - Exemplo de Filtro da média em Python



Fonte: Adaptado de SSIM(2021)

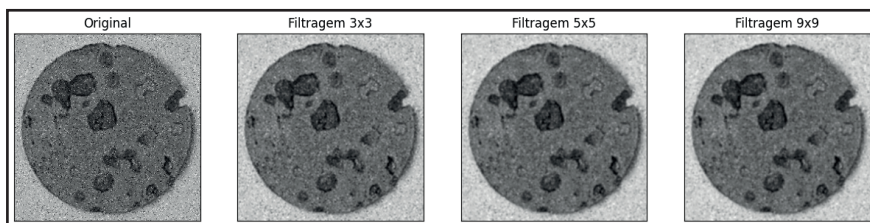
Finalizando o comparativo de filtros de suavização, vamos executar o código de um filtro gaussiano, usando a mesma ideia dos filtros anteriores, e apenas substituindo a função principal do código. No código presente, utilizaremos a função `cv2.GaussianBlur(img, (n,n), sigma)`, onde *n* representa o tamanho do kernel, e *sigma* = 1.

```
import numpy as np
from matplotlib import pyplot as plt
img=cv2.imread("median_test.jpg",0)
imgr1 = cv2.GaussianBlur(img, (3,3), 1) #
Filtragem com kernel de 3x3
imgr2 = cv2.GaussianBlur(img, (5,5), 1) #
Filtragem com kernel de 5x5
imgr3 = cv2.GaussianBlur(img, (9,9), 1) #
Filtragem com kernel de 9x9
plt.figure(figsize=(15,10))
plt.subplot (1,4,1)
plt.imshow(img, cmap=plt.cm.gray)
```

```
plt.title('Original'),plt.xticks([]),    plt.
yticks([])
plt.subplot (1,4,2)
plt.imshow(imgr1,cmap=plt.cm.gray)
plt.title('Filtragem 3x3'),plt.xticks([]),
plt.yticks([])
plt.subplot (1,4,3)
plt.imshow(imgr2,cmap=plt.cm.gray)
plt.title('Filtragem 5x5'),plt.xticks([]),
plt.yticks([])
plt.subplot (1,4,4)
plt.imshow(imgr3,cmap=plt.cm.gray)
plt.title('Filtragem 9x9'),plt.xticks([]),
plt.yticks([])
```

O resultado da filtragem gaussiana, apresentada na Figura 41, é similar aos outros filtros de suavização vistos até aqui. A escolha do filtro depende dos requisitos da aplicação, qual filtro se encaixa melhor em seu projeto.

Figura 41 - Exemplo de Filtro Gaussiano em Python



Fonte: Adaptado de SSIM(2021)

5.2. Filtros de Aumento

A ideia principal do filtro de aumento ou realce, é salientar transições de intensidade, para o aumento de nitidez das bordas de uma imagem. Enquanto o filtro de suavização é utilizado para borramento e suavização de ruído, o filtro de aumento pode ser útil na detecção de bordas, linhas e contornos. É válido lembrar que tais detecções são extremamente úteis em visão computacional.

Apresentaremos a seguir 3 filtros, sendo que o primeiro filtro utiliza a técnica de convolução, e os outros 2 filtros não lineares, com técnicas e características diferentes.

5.2.1. Filtro de Sobel

O Filtro de Sobel trabalha basicamente com dois kernels, chamados de operadores de Sobel. Estes kernels são capazes de detectar bordas e linhas na direção vertical e horizontal, sendo aplicados um de cada vez na imagem.

A Figura 42 ilustra um operador de Sobel, de tamanho 3x3. Observe que são dois kernels, um horizontal e outro vertical.

Figura 42 - Operador de Sobel de tamanho 3x3

$$Z_h = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad Z_v = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Fonte: O autor (2026)

5.2.2. Filtro Laplaciano

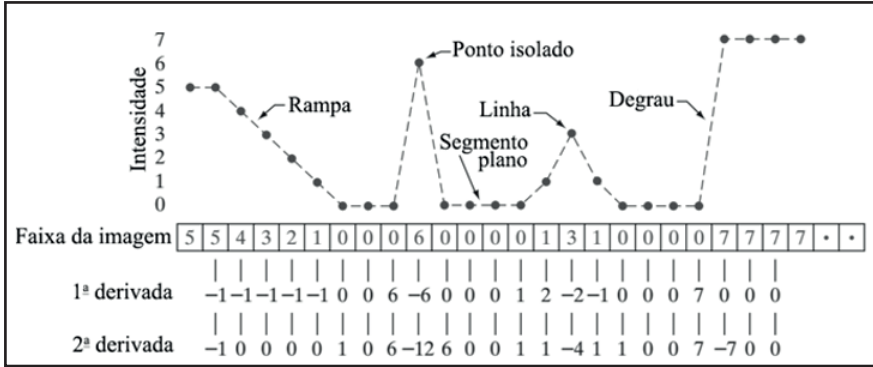
A diferenciação também pode ser utilizada para detecção de variações de intensidade em uma imagem, através da aplicação da derivada segunda.

A Figura 43 ilustra um exemplo de uma linha com intensidade variável em uma imagem. Nesta linha, é possível visualizar instantes de constância e de variação abrupta na intensidade dos pixels. Através do gráfico gerado, podemos aplicar a primeira derivada. Observe que em regiões constantes a primeira derivada é zero. Já em regiões de rampa agora possuem valor constante, e em regiões de transição de intensidade, observa-se mudança de direção de gradiente, o que é característico de funções de segundo grau.

Ao aplicar a segunda derivada, observa-se que as regiões constantes, continuam com valor zero, as regiões de rampa que eram

constantes, agora possuem valor zero, e somente existem valores, agora constantes, nas regiões onde existia mudança de gradiente, ou seja, nas regiões de bordas.

Figura 43 - Exemplo de aplicação de diferenciação em uma linha de transição de uma imagem



Fonte: Adaptado de González (2001)

Dessa forma descrita acima que funcionam os filtros de diferenciação, efetuando derivadas, até sobrar valores de intensidade somente nas regiões de variação abrupta de intensidade.

O Filtro Laplaciano pode ser definido com um operador linear derivativo isotrópico de segunda ordem, definido por:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad (6)$$

De forma discreta, usamos a equação com duas variáveis,

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y) \quad (7)$$

$$\frac{\partial^2 f}{\partial y^2} = f(x, y + 1) + f(x, y - 1) - 2f(x, y)$$

(8)

Dessa forma, o Laplaciano aplicado para a aplicação em questão é dada por:

$$\nabla^2 f(x, y) = f(x, y + 1) + f(x, y - 1) + f(x, y + 1) + f(x, y - 1) - 4f(x, y)$$

(9)

Sendo x e y as coordenadas da imagem.

A Figura 44 ilustra os kernels utilizados na filtragem por Laplaciano. 44(a) mostra o kernel padrão utilizado na aplicação, e em 44 (b) um kernel que inclui a detecção de linhas diagonais. Já 44 (c e d) mostram dois kernels utilizados na prática, em vários softwares de processamento de imagens/visão computacional.

Figura 44 - Kernels da filtragem por Laplacianos

a	<table><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>-4</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr></table>	0	1	0	1	-4	1	0	1	0	b	<table><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>-8</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	1	1	1	1	-8	1	1	1	1	c	<table><tr><td>0</td><td>-1</td><td>0</td></tr><tr><td>-1</td><td>4</td><td>-1</td></tr><tr><td>0</td><td>-1</td><td>0</td></tr></table>	0	-1	0	-1	4	-1	0	-1	0	d	<table><tr><td>-1</td><td>-1</td><td>-1</td></tr><tr><td>-1</td><td>8</td><td>-1</td></tr><tr><td>-1</td><td>-1</td><td>-1</td></tr></table>	-1	-1	-1	-1	8	-1	-1	-1	-1
0	1	0																																									
1	-4	1																																									
0	1	0																																									
1	1	1																																									
1	-8	1																																									
1	1	1																																									
0	-1	0																																									
-1	4	-1																																									
0	-1	0																																									
-1	-1	-1																																									
-1	8	-1																																									
-1	-1	-1																																									

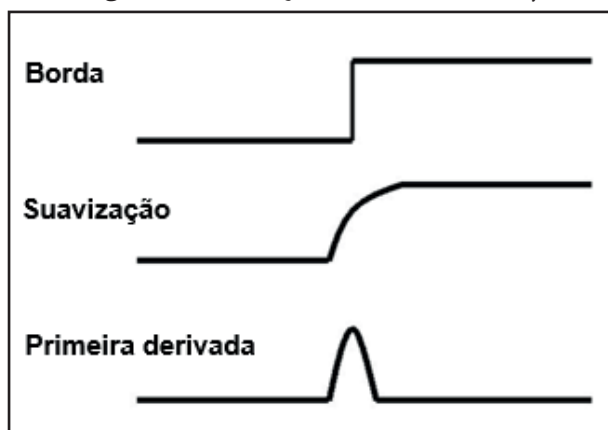
Fonte: O autor (2026)

5.2.3. Filtros de Canny

O operador de Canny foi desenvolvido por John F.Canny em 1986 (Canny, 1986) a partir de sua tese de doutorado. Nele se usa um algoritmo multi-passos para detectar a possibilidade de bordas. O filtro de Canny é um filtro de convolução que usa a primeira derivada. Ele suaviza o ruído e localiza bordas, combinando um operador diferencial

com um filtro Gaussiano. Considere uma borda conforme mostrada na primeira imagem na Figura 45. Se tal borda sofrer um processo de suavização gaussiana, terá como resultado uma variação contínua do valor inicial ao final, com uma inclinação máxima no ponto onde existe o “degrau”. Se esta função, agora contínua, for diferenciada em relação ao eixo x, a inclinação máxima indicará o máximo da nova função em relação à original, conforme ilustrado abaixo.

Figura 45 - Detecção de bordas de Canny



Fonte: Adaptado de Canny

O algoritmo Canny baseia-se nas seguintes regras:

- Boa detecção: Deve possuir habilidade para localizar e marcar todas as bordas realmente existentes.
- Boa localização: Minimizar a distância entre a borda detectada e a borda verdadeira (real).
- Boa resposta: Para cada borda deve haver somente uma resposta.
- Na Prática, o algoritmo de detecção de bordas de Canny possui 4 estágios, que apresentaremos em seguida:

1. Uniformização da imagem.

Segundo Canny, a ideia desta etapa é remover os ruídos localizados nas bordas, sendo extremamente necessário cuidado na escolha dos parâmetros, para não ocorrer um borramento importante nas bordas da imagem. Esta etapa é realizada por meio de um filtro de suavização gaussiano, na forma vista em 5.1.3.

2. Diferenciação

Após a suavização, deve-se fazer a primeira derivada da função gaussiana 2D aplicada no passo acima, sendo assim, é possível calcular o gradiente da superfície uniforme da imagem convolucionada.

Considerando que a diferenciação pode ser feita de forma unidimensional, ou seja, primeiramente no eixo x e depois no eixo y, conforme as equações,

$$G'(x) = -\frac{x}{\sigma^2} e^{\left(-\frac{x^2}{2\sigma^2}\right)} \quad G'(y) = -\frac{y}{\sigma^2} e^{\left(-\frac{y^2}{2\sigma^2}\right)} \quad (10)$$

e a suavização apresentada no estágio 1, os valores uniformizados na direção x são convolucionados utilizando a primeira derivada da função gaussiana unidimensional com mesmo alinhamento na direção y. Da mesma maneira, os valores uniformizados na direção y são convolucionados utilizando a primeira derivada da função gaussiana unidimensional com mesmo alinhamento, agora na direção x. Para somar os valores dos gradientes de x e y, a magnitude e o ângulo da inclinação podem ser calculados através da hipotenusa e arctg do ângulo, similarmente ao operador de Sobel.

3. Omissão de pontos de mínima intensidade

Encontrado a medida da intensidade de cada ponto da imagem, precisa-se localizar agora as bordas. Isso é possível localizando os pontos de máxima intensidade, ou de maneira inversa, pelos pontos de mínima intensidade, que precisam ser omitidos. Um valor de máxima intensidade ocorre no local mais alto da função gradiente ou onde a derivada da função gradiente possui valor zero. Entretanto, desejamos omitir os pontos de mínima intensidade ou pontos de mínimas direções perpendiculares com a borda.

Aproximações são freqüentemente usadas, além da diferenciação perpendicular para cada borda. Cada pixel em volta forma o centro de um novo pixel na vizinhança. Interpolando os valores dos pixels ao redor, as magnitudes dos gradientes são calculados pelos limites das fronteiras vizinhas em ambas as direções perpendiculares do pixel central, como mostrado na Figura abaixo. Se o pixel considerado na Figura abaixo possuir valor menor do que os valores dos pixels vizinhos, ele será omitido.

4. Limiarização de Borda (Thresholding)

A limiarização usada no algoritmo Canny usa o método chamado “histerese”.

Considerando um segmento de borda, para todo valor situado acima do limite superior de limiarização, ele é imediatamente aceito. Para todo valor situado abaixo do limite inferior de limiarização, ele é imediatamente rejeitado. Pontos situados entre os dois limites serão aceitos se eles estiverem relacionados com pixels que apresentem respostas fortes. As Figuras abaixo mostram aplicações utilizando o operador Canny considerando diferentes escalas e diferentes limiarizações. A Figura 46 ilustra uma imagem filtrada por Canny, e as suas resultantes durante todos os 4 estágios.

Figura 46 - Imagem Original. Suavização (b). Diferenciação (c). Omissão de mínima intensidade (d). Limiarização por histerese e resultado final (e)



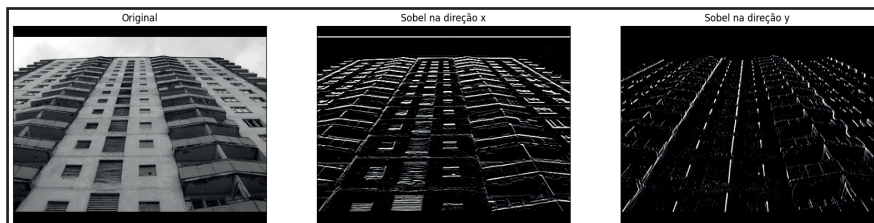
Fonte: Adaptado de Bess (2025)

Os códigos a seguir mostram aplicações dos filtros de aguçamento estudados até aqui. Começaremos com o código de um filtro Sobel, de kernel 3 x 3.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('pripyat2.jpg',0)
# SOBEL=(img, var_arm, h, v, ksize)
#img: imagem de entrada,
#var_arm: codificação da imagem
#h ou v: direção do filtro
#ksize: tamanho nxn do kernel.
sobelx = cv2.Sobel(img, cv2.CV_8U, 0, 1,
ksize=3) # kernel 3x3
sobely = cv2.Sobel(img, cv2.CV_8U, 1, 0,
ksize=3) # kernel 3x3
plt.figure(figsize=(20,15))
plt.subplot(1,3,1)
plt.imshow(img,cmap=plt.cm.gray)
plt.title('Original'),plt.xticks([]), plt.
yticks([])
plt.subplot(1,3,2)
plt.imshow(sobelx,cmap=plt.cm.gray)
plt.title('Sobel na direção x'),plt.
xticks([]), plt.yticks([])
plt.subplot(1,3,3)
plt.imshow(sobely,cmap=plt.cm.gray)
plt.title('Sobel na direção y'),plt.
xticks([]), plt.yticks([])
```

A Figura 47 abaixo mostra o resultado da execução do algoritmo acima, nas direções vertical e horizontal. Podemos observar claramente a diferença entre as saídas, onde a saída x evidencia as linhas horizontais, e a saída y, as linhas verticais.

Figura 47 - Operador de Sobel com kernel 3x3



Fonte: Adaptado de Davies (2025)

O filtro de laplaciano é aplicado na execução a seguir, onde foi utilizada a mesma imagem do filtro anterior, para uma melhor comparação.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('pripayat2.jpg', 0)
#LAP=Laplacian(img, var_arm)
#img: imagem de entrada
#var_arm: codificação da imagem
lap = cv2.Laplacian(img, cv2.CV_8U)
plt.figure(figsize=(25,20))
plt.subplot (1,2,1)
plt.imshow(img, cmap=plt.cm.gray)
plt.title('Original'), plt.xticks([]), plt.
yticks([])
plt.subplot (1,2,2)
plt.imshow(lap, cmap=plt.cm.gray)
plt.title('Laplaciano'), plt.xticks([]), plt.
yticks([])
```

Podemos observar na Figura 48, que a composição, ou seja, uma saída única, com filtros na horizontal e vertical, melhora a visualização, quando comparado com o filtro de Sobel.

Figura 48 - Aplicação do filtro Laplaciano



Fonte: Adaptado de Davies (2025)

Para finalizar este capítulo, abaixo a execução do filtro de canny, com valores limiares compreendidos entre 80 e 160. Cabe a comparação entre os dois algoritmos apresentados anteriormente com a mesma imagem.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('pripayat2.jpg', 0)
#edges = cv2.Canny(img, MinVal, MaxVal)
# MinVal e MaxVal: valores mínimos e máximos
da limiarização
#por histerese aplicada no último passo do
algoritmo
canny = cv2.Canny(img, 80, 160)
plt.figure(figsize=(25, 20))
plt.subplot (1, 2, 1)
plt.imshow(img, cmap=plt.cm.gray)
plt.title('Original'), plt.xticks([]), plt.
yticks([])
plt.subplot (1, 2, 2)
plt.imshow(canny, cmap=plt.cm.gray)
plt.title('Filtragem por Canny'), plt.
xticks([]), plt.yticks([])
```

A Figura 49 ilustra o resultado da filtragem de Canny. Podemos observar uma maior nitidez nas bordas, mas em contrapartida, observa-se também, algumas bordas que não foram bem definidas em Sobel e em Laplaciano, que podem até, prejudicar a aplicação do algoritmo. Para diminuir as bordas indesejadas, podemos ajudar melhor os valores máximos e mínimos de histerese.

Figura 49 - Filtragem com o algoritmo de Canny



Fonte: Adaptado de Davies (2025)

6. PROCESSAMENTO DE IMAGENS NO DOMÍNIO DA FREQUÊNCIA

Além do domínio espacial, podemos utilizar do domínio da frequência para análise e filtragem de imagens. A Transformada de Fourier é uma técnica útil para tal, sendo que, em imagens, é utilizada a Transformada de Fourier 2D, e a Transformada Rápida de Fourier (Fast Fourier Transform – FFT), para agilizar nossos cálculos.

A transformada de Fourier e 2D é dada por:

$$F(u, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) \exp[-j2\pi(ux + vy)] dx dy$$

E a sua transformada inversa:

$$f(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u, v) \exp[j2\pi(ux + vy)] du dv$$

Sendo $f(x, y)$ um pixel da imagem no domínio espacial, $F(u, v)$ o correspondente ao pixel $f(x, y)$ no domínio da frequência, e $j = -1$.

A visualização da imagem no domínio da frequência não é compreensível, já que a imagem será convertida em duas matrizes, sendo uma das matrizes correspondentes a magnitude e outra correspondente a fase da imagem. Dessa forma, sempre que é feita alguma manipulação

no domínio da frequência, é necessário voltar ao domínio do espaço para visualizar a alteração.

A seguir, veremos com detalhes, passos básicos para trabalhar no domínio da frequência, que, de forma resumida, são primeiramente apresentadas abaixo:

Passos para análise no Domínio da Frequência

1. Implementação da Transformada de Fourier na Imagem.
2. Visualizar e Centralizar as componentes de frequência.
3. Aplicar o processo de filtragem/ melhoramento na imagem
4. Descentralizar
5. Implementar a Transformada Inversa de Fourier, para visualização do resultado no domínio espacial

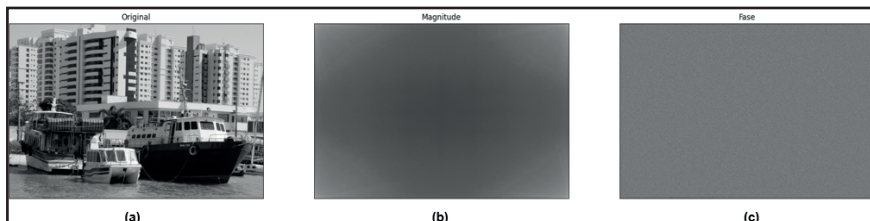
A seguir falaremos detalhadamente de cada passo descrito acima.

1. Implementação da Transformada de Fourier na Imagem.

A primeira etapa consiste em aplicar a Transformada de Fourier na imagem. O resultado do processo FFT, como já é sabido, é uma matriz numérica complexa que devido a sua difícil interpretação, devemos transformá-la em um espaço bidimensional, sendo que a primeira matriz possui os valores de magnitude, e a segunda, os valores de fase, da mesma forma como é vista em números complexos

A Figura 50 abaixo ilustra as matrizes de magnitude e fase de uma imagem.

Figura 50 - Magnitude (b) e fase (c) de uma imagem (a) no Domínio da Frequência



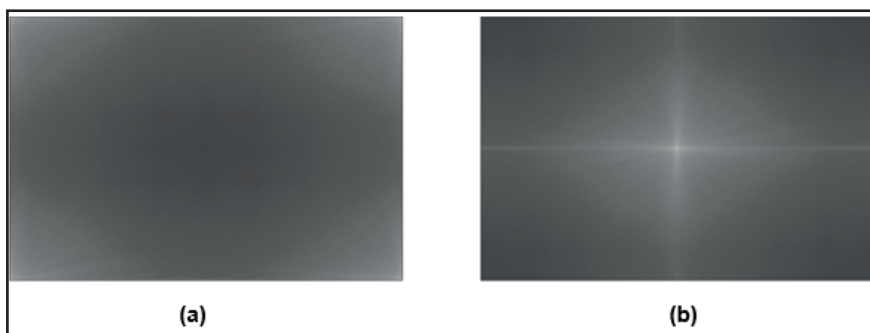
Fonte: O autor (2026)

A etapa a seguir ajusta a componente de magnitude para facilitar os processos de filtragem/melhoramento.

2. Visualizar e Centralizar as componentes de frequência.

No componente de magnitude da FFT, as componentes de frequência se localizam nos cantos da imagem. Observe a Figura 50 (b), onde os cantos são mais claros que o centro. Para facilitar os cálculos, as componentes de frequência são transladadas para o centro da imagem, sem a perda de informação, conforme ilustra a Figura 51.

Figura 51 - Espectro de magnitude de imagem. Espectro original (a). Espectro Centralizado (b)

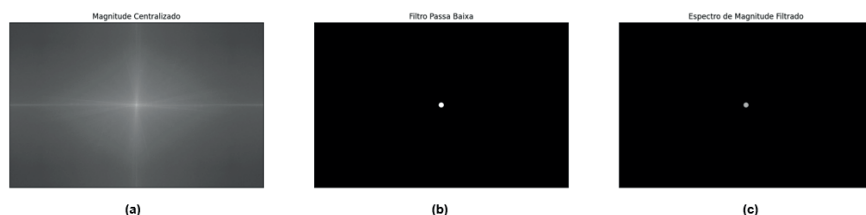


Fonte: O autor (2026)

3. Aplicar o processo de filtragem/ melhoramento na imagem

O Domínio da frequência, diferentemente do domínio espacial, realiza a filtragem através de uma simples multiplicação do espectro de magnitude da imagem, devidamente centralizado, com o kernel, que possui o mesmo tamanho da imagem. Não vamos entrar em detalhe sobre filtragem agora, somente em um subtópico específico neste capítulo, mas como exemplo, faremos um processo de filtragem passa baixa ideal, apenas de forma ilustrativa, conforme visto na Figura 52.

Figura 52 - Espectro centralizado (a). Máscara de filtro passa baixa (b). Espectro filtrado (c)

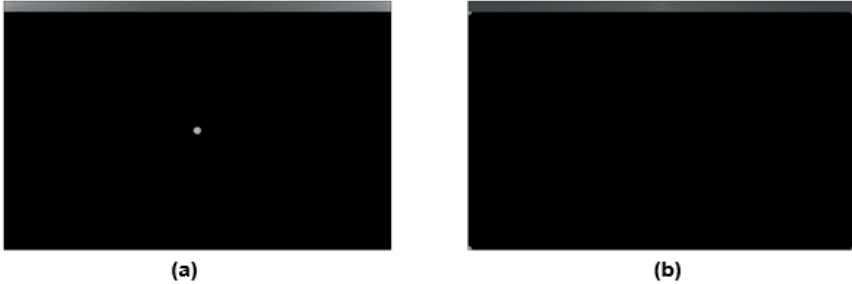


Fonte: O autor (2026)

4. Descentralização

Após o processo de filtragem, é necessário começar a fazer o processo inverso, para que a imagem volte ao domínio de compreensão da nossa visualização, ou seja, o domínio espacial. Dessa forma, o primeiro passo é volta do estado original do espectro de magnitude, ou seja, as componentes de frequência devem voltar para os cantos da imagem, conforme o espectro de magnitude original.

Figura 53 - Espectro centralizado (a). Espectro com as componentes de frequências localizadas nos cantos, conforme espectro e magnitude original (b).



Fonte: O autor (2026)

O último passo consiste em devolver a imagem para o domínio espacial, para que enfim, sejam visualizadas as modificações realizadas no domínio da frequência.

5. Implementação a Transformada Inversa de Fourier, para visualização

Ao fim do processo de filtragem/melhoramento, é necessário a volta para o domínio espacial, que é realizado por meio da Transformada Inversa de Fourier. A Figura 54 abaixo ilustra uma comparação entre a imagem original e a imagem filtrada através de todo o processo descrito acima.

Figura 54 - Imagem suavizada (b), a partir de (a)



Fonte: O autor (2026)

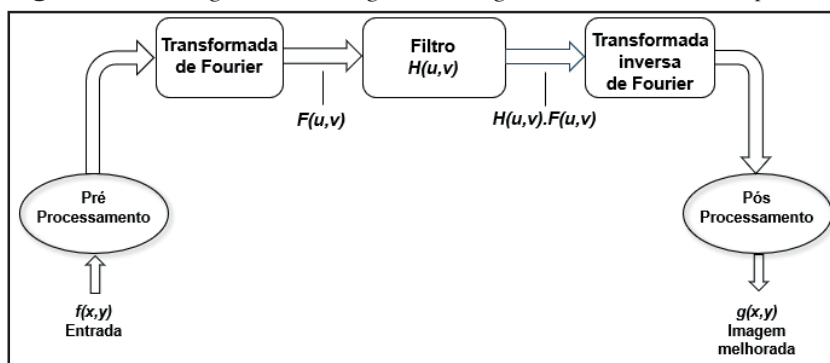
A filtragem no domínio espacial baseia-se na utilização de kernels, que são pequenas matrizes bidimensionais onde seus valores de seus coeficientes determinam o objetivo a ser alcançado durante o processamento, seja uma leve suavização ou até uma robusta detecção de bordas.

Como já foi falado linhas atrás, o custo computacional da convolução é bastante elevado. Foi dito como exemplo, que uma imagem de 512x512, aplicando a convolução com um kernel de 16x16, por exemplo, são necessárias 67.108.864 multiplicações.

Uma forma de sair dessa quantidade enorme de cálculos computacionais, e mudar para um domínio de transformação que exija menos custo computacional. O Domínio da Frequência, em especial a Transformada de Fourier, uma operação de convolução pode ser resolvida com uma simples multiplicação, reduzindo drasticamente a quantidade de operações.

A Figura 55 ilustra todo o processo de filtragem no domínio da frequência, por meio da Transformada de Fourier, conforme visto nos passos para análise do domínio da frequência.

Figura 55 - Fluxograma da filtragem de imagens no domínio da frequência.



Fonte: O autor (2026)

Observe na imagem que a função $f(x,y)$, que denota uma imagem no domínio espacial, é convertida para o domínio da frequência, e agora representada por $F(u,v)$. O filtro, que pode ser de suavização ou aguçamento, já está no domínio da frequência, sendo representado por $H(u,v)$. A operação de filtragem é realizada através de uma

simples multiplicação de $F(u,v)$ por $H(u,v)$, sendo representada por $G(u,v)=F(u,v).H(u,v)$. Após o processo descrito, é necessário transformar a imagem para o domínio espacial novamente, a fim de visualizar se os efeitos da filtragem foram satisfatórios ou não. Para esta etapa, é realizado o processo de transformada inversa de Fourier, denotado por:

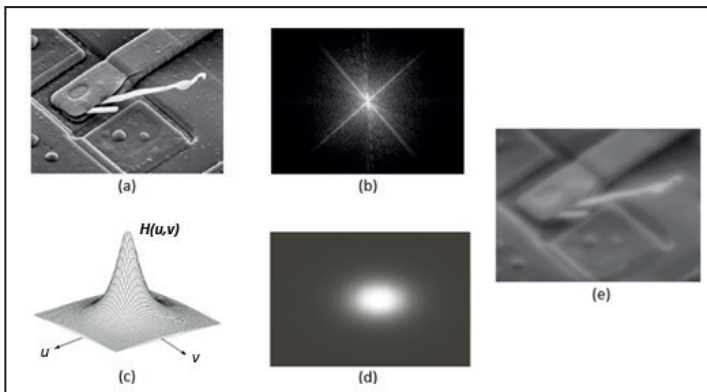
$$g(x,y)=T^{-1}\{G(u,v)\} \quad (11)$$

Observe que a imagem final $g(x,y)$, resultante de $f(x,y)$, retornou ao seu domínio original, o domínio espacial.

No domínio da frequência, a manipulação é realizada sobre o espectro da imagem e o espectro do filtro. Quando um filtro é capaz de permitir a passagem das frequências mais baixas e atenuar as frequências mais altas, é chamado de filtro passa baixas, e quando permite a passagem das frequências mais altas e atenua as frequências mais baixas, o filtro passa alta.

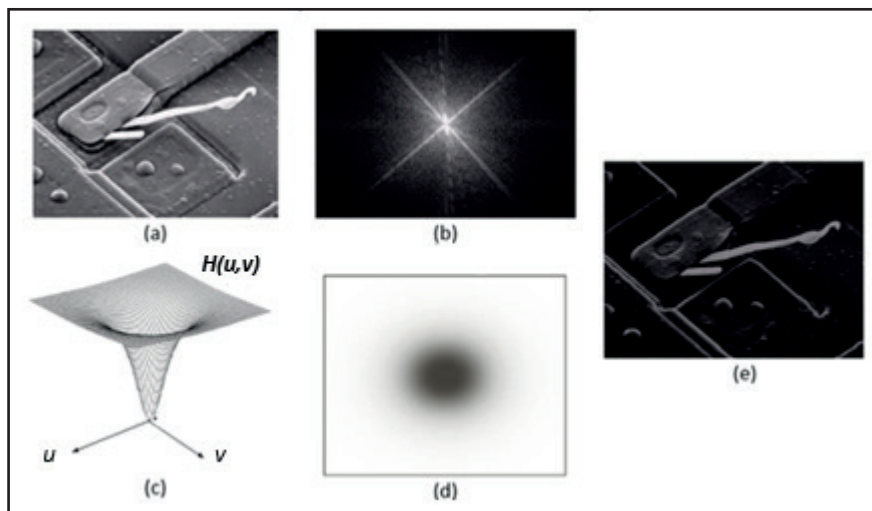
As Figuras abaixo ilustram a filtragem da mesma imagem no domínio da frequência, através de filtros passa baixa (Figura 56) e passa altas (Figura 57).

Figura 56 - Filtragem passa baixas. 56 (a) mostra a imagem original, e seu espectro de frequência em 56 (b). 56 (c) mostra a representação 2-D de um filtro da Gaussiana, passa baixas, e seu espectro em 56 (d). A multiplicação do espectro da imagem com o espectro do filtro gera um espectro resultante, e que após a aplicação da transformada inversa do espectro, resulta na imagem 56 (e)



Fonte: Adaptado de González (2008)

Figura 57 - Filtragem passa altas: 57 (a) mostra a imagem original, e seu espectro de frequência em 57 (b). 57 (c) mostra a representação 2-D de um filtro da Gaussiana, passa altas, e seu espectro em 57 (d). A multiplicação do espectro da imagem com o espectro do filtro gera um espectro resultante, e que após a aplicação da transformada inversa do espectro, resulta na imagem 57 (e).



Fonte: Adaptado de González (2008)

Pode-se observar que o comportamento do filtro de passa baixas é similar ao comportamento do filtro espacial de suavização, e que o comportamento do filtro de passa altas, é similar ao comportamento do filtro espacial de aguçamento/realce.

Neste capítulo, trataremos primeiro de falar sobre o filtro passa-baixa, e logo em seguida, do filtro passa altas. No capítulo referente à Restauração de imagens, falaremos também de outra categoria de filtros no domínio da frequência, que são os filtros passa-banda.

6.1. Filtro Passa Baixa

Apresentaremos neste tópico, três tipos de filtros passa baixa no domínio da frequência mais utilizados: Passa Baixa Ideal, Passa Baixa Butterworth e Passa Baixa Gaussiano.

6.1.1. Filtro Passa Baixa Ideal

O Filtro passa baixa 2D que permite a passagem, sem atenuação de todas as frequências em um círculo de raio D_0 , a partir da origem, e suprime todas as frequências fora desse círculo é chamado de filtro passa baixa ideal HLP, e é determinado pela função:

$$H(u, v) = \begin{cases} 1 & \text{se } D(u, v) \leq D_0 \\ 0 & \text{se } D(u, v) > D_0 \end{cases} \quad (12)$$

Sendo:

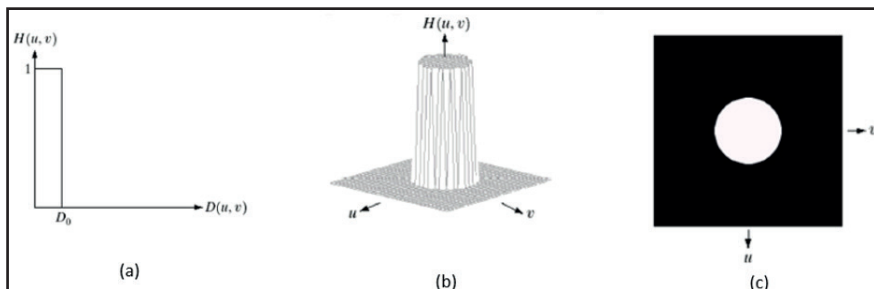
Do : Valor específico definido (não negativo)

$D(u, v)$: Distância do ponto (u, v) à origem do plano de frequência, dado por:

$$D(u, v) = (u^2 + v^2)^{1/2} \quad (13)$$

O Filtro é desenhado através do gráfico da Figura 54(a), e através de sua revolução, obtemos sua perspectiva em 2D, mostrada em 54(b), e em 54(c) é apresentado o espectro de Fourier do filtro passa baixa ideal. Observe que somente a região central possui a cor branca, e o restante do espectro cor preta. Na multiplicação dos espectros, o filtro possuirá intensidade binária, ou seja, o branco representado pelo valor “1”, e o preto pelo valor “0”. Dessa forma, todo o espectro de Fourier da imagem será multiplicado por “1”, se estiver na região do círculo, e “0” se estiver fora. Assim, somente as baixas frequências serão preservadas, e as altas frequências atenuadas.

Figura 58 - Projeto de filtro passa baixas ideal



Fonte: Adaptado de González (2008)

6.1.2. Filtro Passa Baixa Butterworth

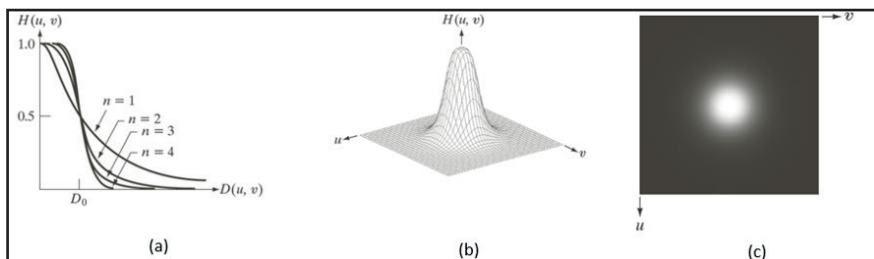
O Filtro Passa Baixa Butterworth é dado pela função de transformação:

$$H(u, v) = \frac{1}{1 + [D(u, v) / D_0]^{2n}} \quad (14)$$

Sendo $\mathbf{D}$ a frequência, $\mathbf{D}_0$ a frequência de corte, e $\mathbf{n}$ a ordem do filtro.

O Filtro de Butterworth, ao contrário do Filtro Passa Baixa Ideal, não possui transições abruptas na frequência de corte, ou seja, a atenuação é realizada de forma suave. O projeto de um filtro de Butterworth 2D Passa Baixa é mostrado na Figura 59 abaixo.

Figura 59 - Projeto de Filtro Butterworth Passa Baixa



Fonte: Adaptado de González (2008)

A imagem filtrada por meio de um Filtro de Butterworth não possui transições bruscas de intensidade de pixels, comparado ao Filtro Passa Baixa Ideal. A sua grande desvantagem é possuir uma ordem “n”, que na grande maioria das aplicações, interfere bem mais do que a frequência de corte estabelecida para o filtro.

6.1.3. Filtro Passa Baixa Gaussiano

O Filtro Passa Baixa Gaussiano é dado pela função de transformação:

$$H(u, v) = e^{-D^2(u, v) / 2\sigma^2}$$

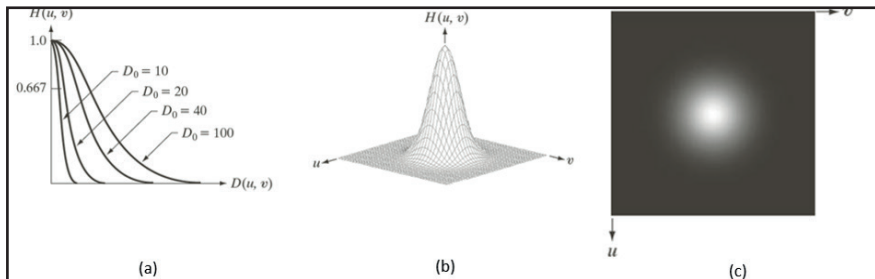
Sendo D a frequência e σ o desvio padrão

Se fizermos $s = D_0$ a frequência de corte, a notação fica compatível com os outros filtros, ou seja:

$$H(u, v) = e^{-D^2(u, v) / 2D_0^2}$$

O filtro gaussiano é o mais utilizado, por possuir uma transição mais suave, e não causar distorções na imagem filtrada. O projeto de um filtro Gaussiano 2D Passa Baixa é mostrado na Figura 60 abaixo.

Figura 60 - Filtro Passa Baixa Gaussiano



Fonte: Adaptado de González (2008)

O código abaixo executa um exemplo de Filtragem Passa Baixa Ideal, Butterworth e Gaussiano, em python.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from math import sqrt,exp
img = cv2.imread("lena.png",0)
fourier = np.fft.fft2(img) # gerando a FFT
fourier1 =np.log(1+np.abs(fourier)) #
Transformação de log para #visualização da
magnitudo
phase=np.angle(fourier) # Gerando a imagem
da fase
center = np.fft.fftshift(fourier) # Gerando a
magnitudo no centro
inv_center = np.fft.ifftshift(center)
processed_img = np.fft.ifft2(inv_center)
#Definindo parâmetros
def distance(point1,point2):
return sqrt((point1[0]-point2[0])**2 +
(point1[1]-point2[1])**2)
# Definindo Filtro Passa Baixa Ideal
def idealFilterLP(D0,imgShape):
base = np.zeros(imgShape[:2])
rows, cols = imgShape[:2]
center = (rows/2,cols/2)
for x in range(cols):
for y in range(rows):
if distance((y,x),center) < D0:
base[y,x] = 1
return base
# Definindo Filtro Passa Baixa de Butterworth
def butterworthLP(D0,imgShape,n):
base = np.zeros(imgShape[:2])
rows, cols = imgShape[:2]
center = (rows/2,cols/2)
for x in range(cols):
for y in range(rows):
base[y,x] = 1/(1+(distance((y,x),center)/
D0)**(2*n))
```



```
return base
# Definindo Filtro Passa Baixa Gaussiano
def gaussianLP(D0,imgShape):
base = np.zeros(imgShape[:2])
rows, cols = imgShape[:2]
center = (rows/2,cols/2)
for x in range(cols):
for y in range(rows):
base[y,x] = exp(((distance((y,x),center)**2)/(2*(D0**2))))
return base
# Filtragem Passa Baixa Ideal
# Frequência de corte de 50
LowPassIdealCenter = center *
idealFilterLP(50,img.shape)
# Volta do espectro filtrado para os cantos
LowPassIdealShift = np.fft.
ifftshift(LowPassIdealCenter)
# Inversa de Fourier final
#Filtragem Passa Baixa de Butterworth
inversa
inverse_LowPassIdeal = np.fft.
ifft2(LowPassIdealShift)
#Frequência de corte de 50 e n=10
LowPassButCenter=center*butterworthLP(50,img.
shape,10)
# Volta do espectro filtrado para os cantos
LowPassButShift = np.fft.
ifftshift(LowPassButCenter)
# Inversa de Fourier final
inverse_LowPassBut = np.fft.
ifft2(LowPassButShift)
#Filtragem Passa Baixa Gaussiano
#Frequência de corte de 50 e n=10
```

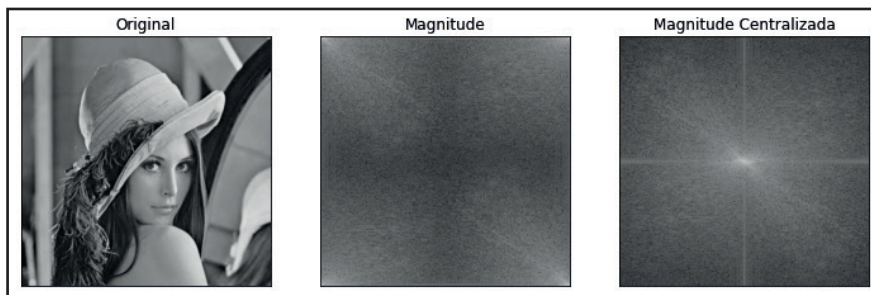
A partir do código acima, já podemos visualizar as componentes de magnitude da imagem, por meio do plot abaixo:

```
#Plotando matriz de magnitude da imagem
plt.figure(figsize=(12,8))
plt.subplot (1,3,1)
```

```
plt.imshow(img, cmap=plt.cm.gray)
plt.title('Original'), plt.xticks([]), plt.
yticks([])
plt.subplot (1, 3, 2)
plt.imshow(fourier1, cmap=plt.cm.gray)
plt.title('Magnitude'), plt.xticks([]), plt.
yticks([])
plt.subplot (1, 3, 3)
plt.imshow(np.log(1+np.abs(center)), cmap=plt.
cm.gray)
plt.title('Magnitude Centralizada'), plt.
xticks([]), plt.yticks([])
plt.show()
```

O resultado da plotagem pode ser visualizada na Figura 61 (a) abaixo:

Figura 61 (a) - Espectros de magnitude da image teste para filtros passa baixa



Fonte: Adaptado de Lenna Image (2025)

A partir da execução do próximo plot, podemos visualizar o resultado da filtragem final, ilustrado na Figura 61 (b).

```
plt.figure(figsize=(12, 8))
plt.subplot (1, 3, 1)
plt.imshow(np.abs(inverse_
LowPassIdeal), cmap=plt.cm.gray)
plt.title('Filtro Passa Baixa Ideal '), plt.
xticks([]), plt.yticks([])
```

```
plt.subplot (1,3,2)
plt.imshow(np.abs(inverse_
LowPassBut), cmap=plt.cm.gray)
plt.title('Filtro Passa Baixa de
Butterworth'), plt.xticks([]),
plt.yticks([])
plt.subplot (1,3,3)
plt.imshow(np.abs(inverse_
LowPassGauss), cmap=plt.cm.gray)
plt.title('Filtro Passa Baixa
Gaussiano'), plt.xticks([]), plt.yticks([])
plt.show()
```

Figura 61 (b) - Resultado de filtrações Passa-Baixa no domínio da frequência, através dos filtros: Ideal, Butterworth e Gaussiano



Fonte: Adaptado de Lenna Image (2025)

6.2. Filtro Passa Alta

Apresentaremos neste tópico, três tipos de Filtros Passa Baixa no domínio da frequência mais utilizados: Passa Baixa Ideal, Passa Baixa Butterworth e Passa Baixa Gaussiano.

6.2.1. Filtro Passa Altas Ideal

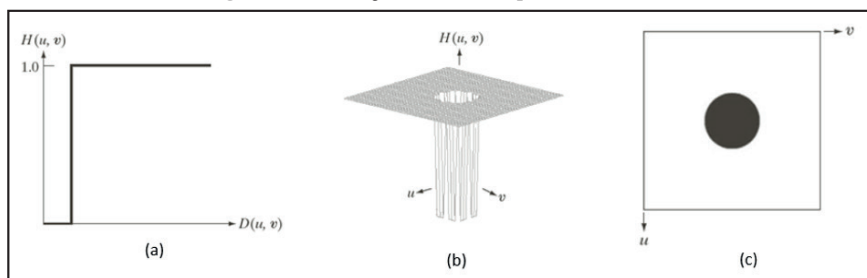
O Filtro passa altas ideal H_{HP} é obtido através do filtro passa baixa ideal H_{LP} através de:

$$H_{HP}(u, v) = 1 - H_{LP}(u, v)$$

Ou seja, o filtro passa altas ideal possui o efeito contrário do filtro passa baixa. Enquanto a filtragem passa baixa nos entrega uma imagem suavizada, a filtragem passa altas nos entrega uma imagem aguçada nas bordas, da mesma forma dos filtros de aguçamento/realce vistos no capítulo sobre filtragem espacial.

Similar a 6.1, O Filtro é desenhado através do gráfico da Figura 62(a), e através de sua revolução, obtemos sua perspectiva em 2D, mostrada em 62 (b), e em 62(c) é apresentado o espectro de Fourier do filtro passa altas ideal. De efeito contrário ao filtro passa baixa, observe que somente a região central possui a cor **preta**, e o restante do espectro cor **branca**. Na multiplicação dos espectros, o filtro possuirá intensidade binária, ou seja, o branco representado pelo valor “1”, e o preto pelo valor “0”. Dessa forma, todo o espectro de Fourier da imagem será multiplicado por “0”, se estiver na região do círculo, e “1” se estiver fora. Assim, somente as **altas frequências serão preservadas**, e as baixas frequências atenuadas.

Figura 62 - Projeto de filtro passa altas ideal



Fonte: Adaptado de González (2008)

6.2.2. Filtro Passa Alta de Butterworth

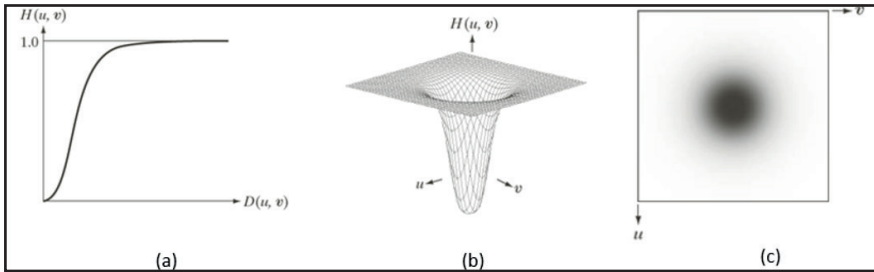
O Filtro Passa Alta de Butterworth é dado pela função de transformação:

$$H(u, v) = \frac{1}{1 + [D_0 / D(u, v)]^{2n}}$$

Sendo D a frequência, D_0 a frequência de corte, e n a ordem do filtro. Observe que existe uma diferença entre a função de transformação do Filtro de Butterworth Passa Baixas, para a função de transformação presente.

O Filtro de Butterworth, ao contrário do Filtro Passa Alta Ideal, não possui transições abruptas na frequência de corte, ou seja, a atenuação é realizada de forma suave, característica que também foi vista na versão passa-baixa. O projeto de um filtro de Butterworth 2D Passa Alta é mostrado na Figura 63.

Figura 63 - Projeto de Filtro Passa Alta de Butterworth



Fonte: Adaptado de González (2008)

6.2.3. Filtro Passa Alta Gaussiano

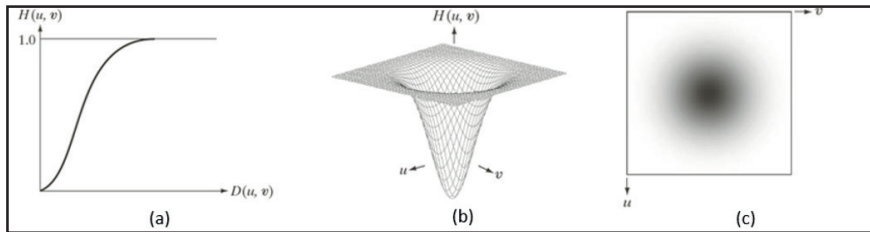
A função de transformação do Filtro Gaussiano Passa Alta é dado por:

$$H(u, v) = 1 - e^{-D^2(u, v) / 2D_0^2}$$

Sendo D_0 a frequência de corte.

A seguir o projeto do Filtro Gaussiano.

Figura 64 - Projeto de Filtro Passa Alta Gaussiano



Fonte: Adaptado de González (2008)

O código abaixo executa um exemplo de Filtragem Passa Alta Ideal, Butterworth e Gaussiano, em python.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from math import sqrt, exp
img = cv2.imread("lena.png", 0)
fourier = np.fft.fft2(img) # gerando a FFT
# Transformação de log para visualização da
magnitudo
fourier1 = np.log(1+np.abs(fourier))
# Gerando a imagem da fase
phase = np.angle(fourier)
# Gerando a magnitude no centro
center = np.fft.fftshift(fourier)
inv_center = np.fft.ifftshift(center)
processed_img = np.fft.ifft2(inv_center)
# Definindo parâmetros
def distance(point1, point2):
    return sqrt((point1[0]-point2[0])**2 +
                (point1[1]-point2[1])**2)
# Definindo Filtro Passa Alta Ideal
def idealFilterHP(D0, imgShape):
    base = np.ones(imgShape[:2])
    rows, cols = imgShape[:2]
    center = (rows/2, cols/2)
    for x in range(cols):
        for y in range(rows):
```

```
if distance((y,x),center) < D0:
    base[y,x] = 0
return base
# Definindo Filtro Passa Alta de Butterworth
def butterworthHP(D0,imgShape,n):
    base = np.zeros(imgShape[:2])
    rows, cols = imgShape[:2]
    center = (rows/2,cols/2)
    for x in range(cols):
        for y in range(rows):
            base[y,x] = 1-1/(1+(distance((y,x),center)/
            D0)**(2*n))
    return base
# Definindo Filtro Passa Alta Gaussiano
def gaussianHP(D0,imgShape):
    base = np.zeros(imgShape[:2])
    rows, cols = imgShape[:2]
    center = (rows/2,cols/2)
    for x in range(cols):
        for y in range(rows):
            base[y,x] = 1 - exp((( -
            distance((y,x),center)**2)/(2*(D0**2))))
    return base
# Filtragem Passa Alta Ideal
# Frequência de corte de 50
HighPassIdealCenter = center *
idealFilterHP(50,img.shape)
# Volta do espectro filtrado para os cantos
HighPassIdealShift = np.fft.
ifftshift(HighPassIdealCenter)
# Inversa de Fourier
inverse_HighPassIdeal = np.fft.
ifft2(HighPassIdealShift)
#Filtragem Passa Alta de Butterworth
#Frequência de corte de 50 e n=10
HighPassButCenter=center*butterworthHP(50,img.
shape,10)
# Volta do espectro filtrado para os cantos
HighPassButShift = np.fft.
ifftshift(HighPassButCenter)
# Inversa de Fourier
```

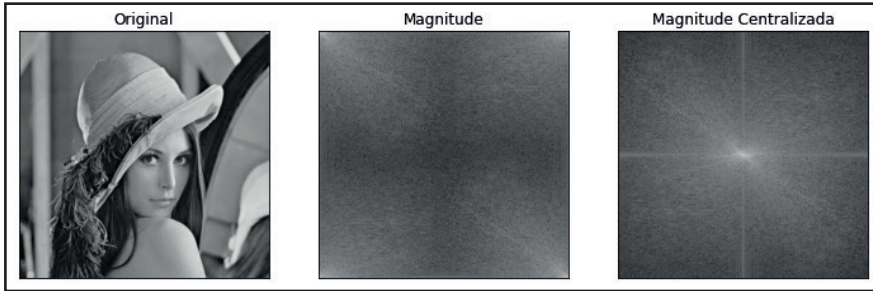
```
#Filtragem Passa Alta Gaussiano
inverse_HighPassBut = np.fft.
ifft2(HighPassButShift)
#Frequência de corte de 50 e n=10
HighPassGaussCenter=center*gaussianHP(50,img.
shape)
# Volta do espectro filtrado para os cantos
HighPassGaussShift = np.fft.
ifftshift(HighPassGaussCenter)
# Inversa de Fourier
inverse_HighPassGauss = np.fft.
ifft2(HighPassGaussShift)
```

A partir do código acima, de modo semelhante aos filtros passa baixa apresentados, podemos visualizar as componentes de magnitude da imagem, por meio do plot abaixo:

```
#Plotando matriz de magnitude da imagem
plt.figure(figsize=(12,8))
plt.subplot(1,3,1)
plt.imshow(img,cmap=plt.cm.gray)
plt.title('Original'),plt.xticks([]), plt.
yticks([])
plt.subplot(1,3,2)
plt.imshow(fourier1,cmap=plt.cm.gray)
plt.title('Magnitude'),plt.xticks([]), plt.
yticks([])
plt.subplot(1,3,3)
plt.imshow(np.log(1+np.abs(center)),cmap=plt.
cm.gray)
plt.title('Magnitude Centralizada'),plt.
xticks([]), plt.yticks([])
plt.show()
```

Observe a semelhança entre o resultado apresentado na Figura 61 (a) em comparação ao resultado da Figura 65 (a) a seguir:

Figura 65 (a) - Espectros de magnitude da imagem teste para filtros passa alta



Fonte: Adaptado de Lenna image (2025)

A semelhança é óbvia, pois, até o momento, não foi realizada qualquer filtragem na imagem. Somente foi executado, segundo os códigos, a conversão da imagem, do domínio discreto, para o domínio da frequência. A partir daí, podemos realizar a filtragem passa alta no domínio da frequência, obtendo o resultado visualizado na Figura 65 (b)

```
plt.figure(figsize=(12,8))
plt.subplot (1,3,1)
plt.imshow(np.abs(inverse_
HighPassIdeal),cmap=plt.cm.gray)
plt.title('Filtro Passa Baixa Ideal '),plt.
xticks([], plt.yticks([])
plt.subplot (1,3,2)
plt.imshow(np.abs(inverse_
HighPassButt),cmap=plt.cm.gray)
plt.title('Filtro Passa Baixa de
Butterworth'),plt.xticks([],
plt.yticks([])
plt.subplot (1,3,3)
plt.imshow(np.abs(inverse_
HighPassGauss),cmap=plt.cm.gray)
plt.title('Filtro Passa Baixa
Gaussiano'),plt.xticks([], plt.yticks([])
plt.show()
```

Figura 65 (b) - Resultado de filtrações Passa-Alta no domínio da frequência, através dos filtros: Ideal, Butterworth e Gaussiano. Observe o realce das bordas, característico de filtros de aguçamento, que são similares à filtros de passa alta, mas em tempo discreto.



Fonte: Adaptado de Lenna Image (2025)

6.3. Projeto de filtros no domínio da frequência a partir de filtros de domínio espacial

Existem alguns filtros clássicos, no domínio espacial. Podemos citar como exemplo, os filtros de Sobel e Canny, vistos em 5.2.1 e 5.2.3. Tais filtros, podem ser utilizados no domínio da frequência. Para tanto, devem passar por uma transformação, referente a tamanho e a espaço.

Em relação ao tamanho, é bom lembrar que as técnicas de filtragem linear usam *kernels*, que possuem tamanhos bem menores do que a imagem. No caso da filtragem em frequência, o filtro possui o mesmo tamanho da imagem, já que o processo consiste de uma simples multiplicação. Sendo assim, deve-se efetuar uma mudança de escala, para o *kernel* tenha o mesmo tamanho da imagem, sem perder as suas propriedades.

Além disso, o filtro deve passar para o Domínio de Fourier, da mesma forma que a imagem, e só assim, o processo de filtragem é realizado.

O código a seguir, apresenta um exemplo em python, da transformação de um filtro espacial de Sobel, no domínio da frequência.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from scipy import signal
import random

img = cv2.imread('portinho.jpg', 0)
fourier = np.fft.fft2(img)
fourier1 = np.log(1+np.abs(fourier))
phase=np.angle(fourier)
center = np.fft.fftshift(fourier)
inv_center = np.fft.ifftshift(center)
processed_img = np.fft.ifft2(inv_center)
# Gerando Filtro de Sobel no espaço e
aplicando no Domínio da #Frequência
# Sobel em x
sobelx=np.
array([[[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]]
(h,w)=img.shape
#função que normaliza o kernel para o
tamanho ideal
ftkernel = np.fft.fftshift(np.fft.fft2(sobelx,
(h, w)))
kernel_vis=np.log(1+np.abs(ftkernel))
# Sobel em y
sobely=sobelx.T
#função que normaliza o kernel para o
tamanho ideal
ftkernel_y = np.fft.fftshift(np.fft.fft2(sobely,
(h, w)))
kernel_visy=np.log(1+np.abs(ftkernel_y))
# Filtragem
# Na direção x
# filtragem com Sobel na direção x
Sobel=center*kernel_vis
# Volta do espectro filtrado para os cantos
SobelShift = np.fft.ifftshift(Sobel)
inverse_Sobel = np.fft.ifft2(SobelShift) #
Inversa de Fourier
# Na direção y
# filtragem com Sobel na direção x
Sobely=center*kernel_visy
```

```
SobelShifty = np.fft.ifftshift(Sobely)
# Volta do espectro filtrado para os cantos
inverse_Sobely = np.fft.ifft2(SobelShifty) #
Inversa de Fourier
#Resultados da Filtragem de Sobel na
Frequência
plt.figure(figsize=(10,12))
plt.subplot (1,3,1)
plt.imshow(img,cmap=plt.cm.gray)
plt.title('Imagem original'),plt.xticks([]),
plt.yticks([])
plt.subplot (1,3,2)
plt.imshow(np.abs(inverse_Sobel),cmap=plt.
cm.gray)
plt.title('Sobel_x na Frequência'),plt.
xticks([]), plt.yticks([])
plt.subplot (1,3,3)
plt.imshow(np.abs(inverse_Sobely),cmap=plt.
cm.gray)
plt.title('Sobel_y na Frequência'),plt.
xticks([]), plt.yticks([])
plt.show()
```

A Figura 66 a seguir ilustra o resultado do código. Convido o leitor para executar um código de um filtro de Sobel, com kernel 3x3, em modo discreto, utilizando a mesma imagem de entrada da Figura 66, e comparar os resultados.

Figura 66 - Filtragem no domínio da frequência a partir de filtros de domínio espacial.

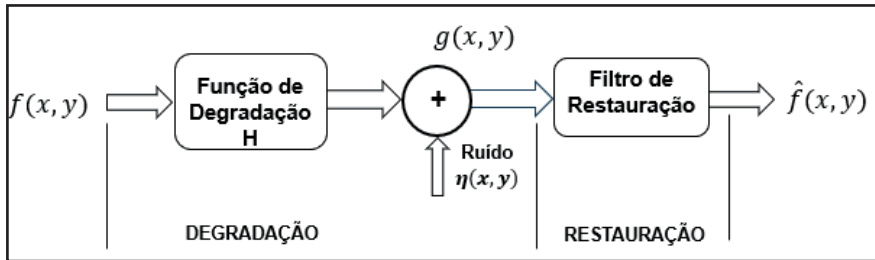


Fonte: O autor (2025)

7. RESTAURAÇÃO DE IMAGENS

O principal objetivo das técnicas de restauração é melhorar uma imagem sob algum aspecto. No nosso caso, as imagens podem ser melhoradas em relação a ruídos ou falhas no processo de aquisição. Uma técnica também muito usual para melhorar uma imagem sobre esses aspectos é o realce, entretanto, o realce é um processo heurístico, subjetivo, ou seja, seus parâmetros são escolhidos conforme a análise do usuário. A Restauração, entretanto, busca reconstruir ou recuperar uma imagem que foi degradada usando informações a respeito do processo de degradação. A Figura a seguir ilustra o processo de degradação e restauração de imagens. Na Figura, pode-se observar que a imagem $f(x,y)$ foi adicionada de um ruído $n(x,y)$, formando a função $g(x,y)$, que apresenta tanto as componentes da imagem, como a componente de ruído. O filtro de restauração aplicado deve conhecer o comportamento do ruído, para tentar anulá-lo, e assim gerar a imagem $\hat{f}(x,y)$, que é uma função mais aproximada possível de $f(x,y)$.

Figura 67 - Modelo do processo de restauração de imagens



Fonte: O autor (2026)

Dessa forma, podemos descrever no domínio espacial o modelo de degradação como:

$$g(x,y) = h(x,y) * f(x,y) + n(x,y) \cong \hat{f}(x,y)$$

Sendo $g(x,y)$ a imagem degradada, $h(x,y)$ a função de degradação, $n(x,y)$ o ruído (neste caso, ruído aditivo), e $f_{x,y}$ a estimativa da imagem, mais próximo da original.

Na prática, quanto mais conhecermos da natureza do ruído aditivo, e da função de degradação da imagem, podemos fazer o processo inverso, trazendo a imagem estimada mais próxima da original.

No domínio da frequência:

$$g(x,y) = h(x,y) * f(x,y) + n(x,y) \cong \hat{f}(x,y)$$

Sendo $G(u,v)$ o espectro da imagem degradada, $H(u,v)$ a função de degradação e $N(u,v)$ o ruído, todos no domínio da frequência. De forma análoga ao caso do domínio espacial, $F(u,v)$ é o espectro mais próximo de $F(u,v)$.

O grande causador de degradação em imagens digitais é o ruído. Conhecer-lo de forma a saber como será o seu comportamento é uma excelente alternativa para o desenvolvimento de filtros que possam ser eficazes, no sentido de que a imagem restaurada seja o mais próxima possível da original.

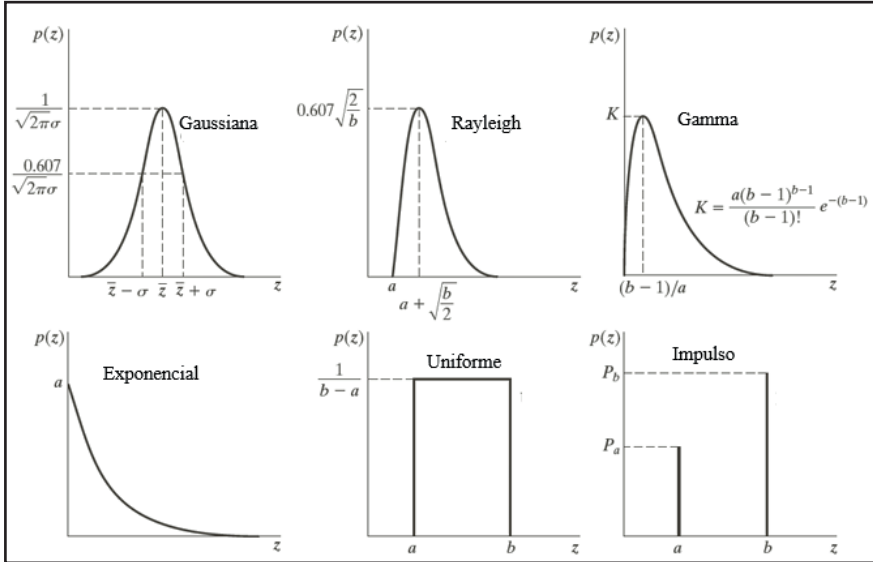
O ruído pode ser considerado uma variável aleatória, caracterizada por uma função de densidade de probabilidade, no caso de ruídos no domínio espacial, ou, no caso do domínio da frequência, é geralmente caracterizado por picos no espectro de Fourier, surgindo devido a interferência elétrica ou eletromagnética adquiridos no processo de aquisição ou no trajeto até o receptor. Abordaremos primeiro os ruídos de natureza espacial, e por último, os ruídos de natureza periódica.

7.1. Ruído de natureza espacial

A principal fonte de ruídos de natureza espacial é o processo de aquisição da imagem, que pode ocorrer devido a qualidade dos sensores, a sensibilidade e as condições ambientais. Pode-se dizer que o ruído espacial é independente de coordenadas espaciais e que não se correlaciona com a imagem em si, ou seja, não existe correlação entre os valores de pixel e os valores de componente de ruído.

Dessa forma, o ruído pode ser considerado de natureza aleatória, e o melhor descritor de um modelo de ruído neste caso é a sua função de densidade de probabilidade (PDF). Na Figura a seguir, são apresentados os ruídos mais comumente encontrados, e suas respectivas PDF's.

Figura 68 - Ruídos mais encontrado e suas PDF's



Fonte: O autor (2026)

Uma forma de encontrar a PDF de um ruído específico é através da subtração de uma imagem com ruído com a mesma imagem, sem ruído. A imagem padrão comumente usada para tal experimento é mostrada abaixo.

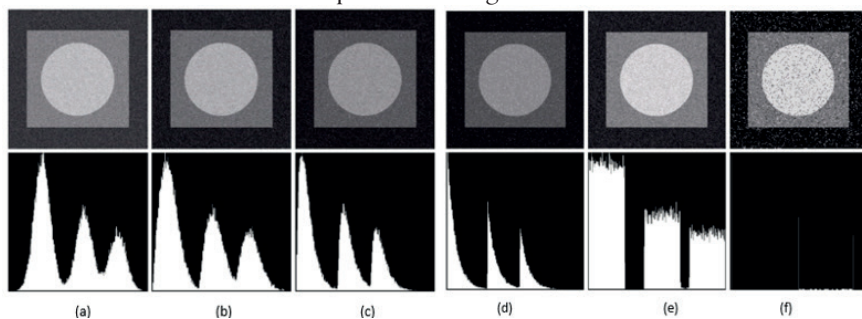
Figura 69 - Imagem padrão para estimar a PDF de um ruído aleatório em imagem.



Fonte: Akyol (2023)

A Figura a seguir ilustra a imagem de teste, adicionada de seis tipos de ruído, um de cada vez. Após a adição, a imagem ruidosa foi subtraída da imagem teste, e da resultante foi extraído o histograma, que por sua vez ilustra o tipo de ruído adicionado na imagem.

Figura 70 - Imagens teste com adição de ruído: Gaussiano (a), Rayleigh (b), Gamma (c), Exponencial (d), Uniforme (e), Salt and Pepper (f) e seus respectivos histogramas.



Fonte: Akyol (2023)

Esta técnica é útil quando possuímos uma imagem sem ruído, para depois comparar com a imagem ruidosa, e assim projetar o modelo de filtro esperado para quando o sistema apresentar imagens com ruído. Um exemplo que pode ser facilmente observado é o comportamento de câmeras DLSR em situação de baixa iluminação. O sistema eletrônico da câmera automaticamente regula a iluminação, entretanto, essa regulação traz ruído aditivo. Assim, a câmera já traz embarcado vários modelos de ruído, para diferentes padrões de luminosidade.

7.1.1. Filtros de Estatística de Ordem

Os Filtros de estatística de ordem são filtros espaciais cuja resposta se baseia na ordenação dos valores dos pixels contidos na área da imagem coberta pelo filtro.

Existem inúmeros filtros de estatística de ordem, que também são chamados de filtros de ordenação. Neste material, abordamos o filtro da mediana, em 5.1.1, e faremos exemplos deste filtro de estatística de ordem em python, para servir de comparação com os outros filtros espaciais vistos até aqui.

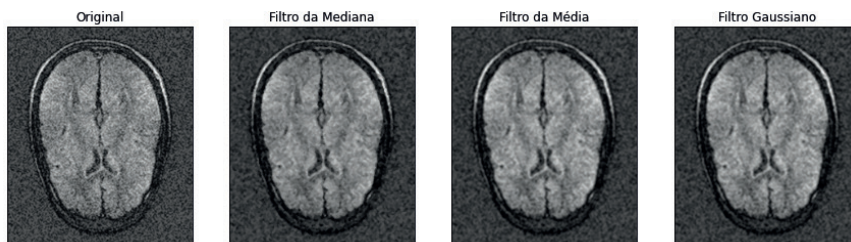
O código abaixo mostra um exemplo em python de 3 filtros espaciais: Mediana, Média Gaussiano.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img=cv2.imread("brain.jpg",0)
imgr1 = cv2.medianBlur(img, 3) #Filtragem da
mediana, kernel 3x3
imgr2 = cv2.blur(img, (3,3)) #Filtragem da
media, kernel 5x5
imgr3 = cv2.GaussianBlur(img, (3,3), 2)
#Filtragem gaussiana,kernel 5x5,sigma=2
#Mostrando os resultados
plt.figure(figsize=(15,10))
plt.subplot (1,4,1)
plt.imshow(img,cmap=plt.cm.gray)
plt.title('Original'),plt.xticks([], plt.
yticks([]))
```

```
plt.subplot (1,4,2)
plt.imshow(imgr1,cmap=plt.cm.gray)
plt.title('Filtro da Mediana'),plt.
xticks([], plt.yticks([])
plt.subplot (1,4,3)
plt.imshow(imgr2,cmap=plt.cm.gray)
plt.title('Filtro da Média'),plt.xticks([],
plt.yticks([])
plt.subplot (1,4,4)
plt.imshow(imgr3,cmap=plt.cm.gray)
plt.title('Filtro Gaussiano'),plt.
xticks([], plt.yticks([])
plt.show()
```

O resultado das filtragens acima podem ser vistas na Figura a seguir. Observe que a diferença entre a imagem original e a imagem filtrada é bem sutil, ou seja, o efeito da filtragem na remoção dos ruídos é mínima. Isso ocorre devido a natureza do ruído em tempo discreto ser de natureza aleatória, sem padrão nenhum, o que dificulta o processo de filtragem.

Figura 71 - Restauração no Domínio Espacial em Python por meio dos Filtros: Mediana, Média e Gaussiano



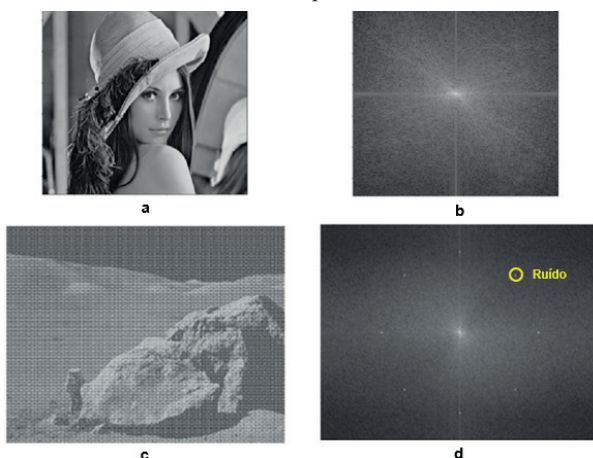
Fonte: Adaptado de Abdelmajid (2022)

7.2. Filtros de Natureza Periódica

O ruído periódico em uma imagem normalmente é proveniente de interferência eletromagnética durante o processo de aquisição da

imagem, ou no seu processo de transmissão – recepção. Este tipo de ruído é identificado através de anomalias no espectro de Fourier da imagem analisada. A Figura 72 ilustra uma imagem sem ruído (a), e seu respectivo espectro de Fourier (b), e outra imagem (c), dessa vez com ruído, e seu espectro com anomalias (d). Observe a presença de pontos isolados no espectro da imagem com ruído. Pode – se afirmar que tais pontos não fazem parte de um espectro padrão, e a forma de removê-los é a através da filtragem do espectro ruidoso.

Figura 72 - Comparativos entre espectros de imagens sem ruído periódico e com ruído periódico



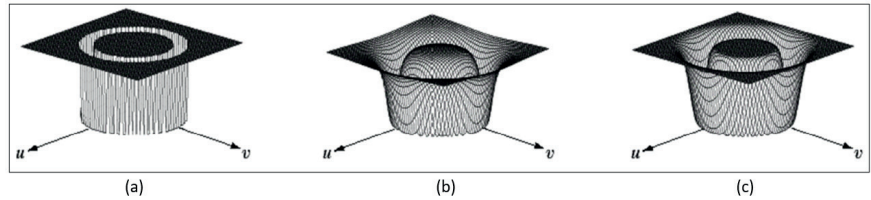
Fonte: Adaptado de Lenna Image (2025), Akyol (2023)

Dependendo da localização do ruído, o processo de remoção pode ser realizado utilizando os filtros passa baixa e passa alta já vistos aqui. Caso a localização não seja favorável à aplicação dos filtros descritos acima, faz-se necessário o uso de outras técnicas de filtragem. A seguir conheceremos os Filtros Rejeita Banda e Filtro de Notch, excelentes no processo de remoção de ruídos periódicos.

7.2.1. Filtro Rejeita Banda

Os Filtros Rejeita Banda são utilizados para extrair ruídos localizados em bandas - de preferência curtas - onde se localizam ruídos periódicos. A Figura a seguir ilustra a perspectiva dos Filtros Rejeita Banda mais comumente usados: Ideal, Butterworth e Gaussiano.

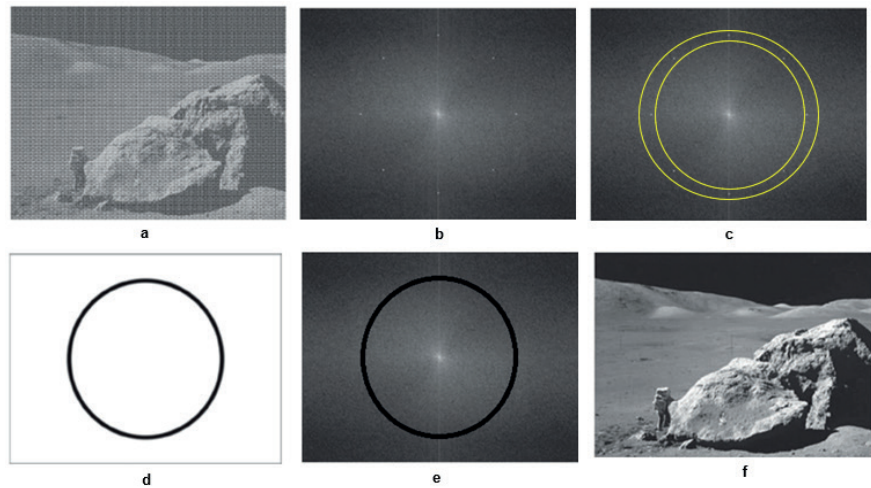
Figura 73 - Gráficos em perspectiva dos Filtros: Ideal, Passa Banda, Gaussiano



Fonte: O autor (2026)

Podemos tomar como um exemplo ideal, a Figura 72 (c) e seu espectro, em 72 (d), que são novamente ilustrados na Figura 74.

Figura 74 - (a) Imagem com ruído periódico. (b) Espectro de Fourier. (c) Região demarcada do espectro onde se encontra os pontos de ruído. (d) Filtro Rejeita banda, que atenua a região demarcada em(c). (e) Espectro Resultante. (f) Imagem filtrada.



Fonte: Adaptado de Akyol (2023)

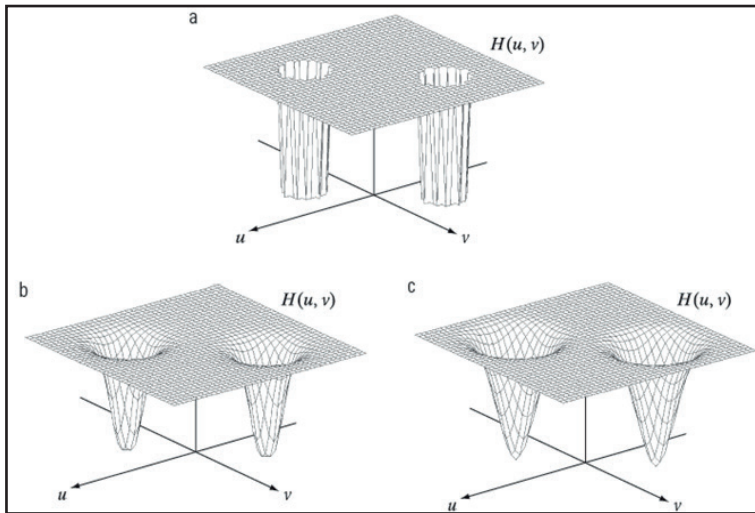
Na Figura, observa-se em 74(d) o espectro de Fourier do filtro rejeita banda ideal, que vai atenuar a região demarcada em 74(c). A multiplicação do espectro de 74(b) com 74(d) é visto em 74(e). Após as devidas transformações do domínio da frequência para o domínio espacial, a imagem resultante, já filtrada, é vista em 74(f).

No caso da Figura acima foi possível aplicar um filtro rejeita-banda, pois os pontos de ruído tinham a mesma distância do centro da imagem, ou seja, foi possível definir um raio, e a partir daí, uma banda de corte. Nem sempre isso é possível, e para tais casos, pode-se aplicar o Filtro Notch, que veremos em seguida.

7.2.2. Filtro Notch

Os Filtros Notch rejeitam frequências em vizinhanças pré-definidas em relação a uma frequência central. A Figura a seguir ilustra a perspectiva dos Filtros Notch mais comumente usados: Ideal, Butterworth e Gaussiano.

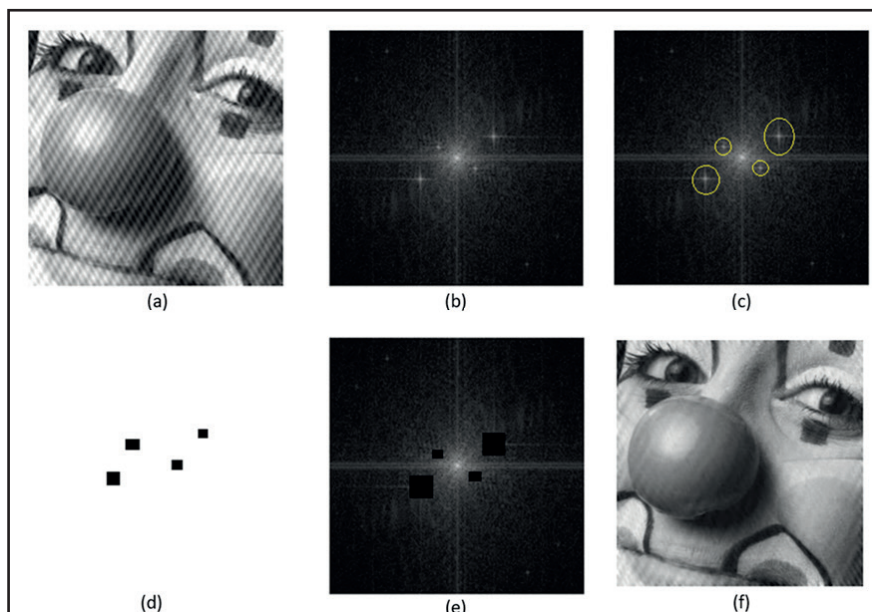
Figura 75 - Filtros Notch: Ideal (a). Butterworth (b). Gaussiano (c)



Fonte: Dungeon (1990)

De forma análoga aos Filtros Rejeita Banda, tomaremos um exemplo ideal para detalhamento do Filtro Notch.

Figura 76 - (a) Imagem com ruído periódico. (b) Espectro de Fourier. (c) Região demarcada do espectro onde se encontra os pontos de ruído. (d) Filtro Notch, que atenua a região demarcada em(c). (e) Espectro Resultante. (f) Imagem filtrada.



Fonte: Adaptado de Zhang (2013)

A aplicação do Filtro Notch é similar ao filtro-rejeita banda, com o detalhe de os pontos de ruído não estarem alinhados, sendo necessário outra abordagem. Pode-se observar que a imagem resultante não está totalmente filtrada. Neste caso, deve-se analisar o espectro com mais detalhes, à procura de mais pontos de ruído, e aplicar a técnica mais conveniente, e filtrar novamente.

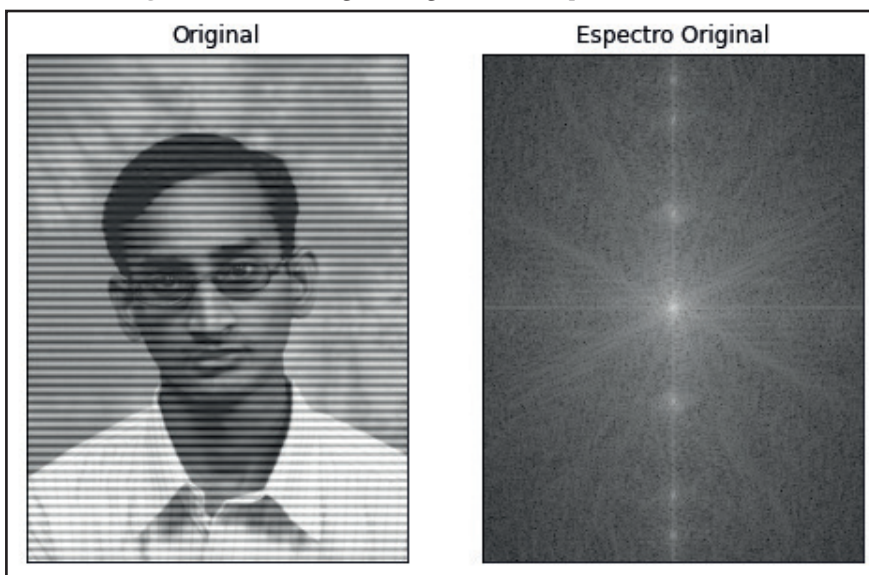
O código a seguir ilustra um exemplo em python de filtros notch.

Inicialmente, é necessário obter o espectro de frequência da imagem, que podemos executar conforme descrito abaixo.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from math import sqrt,exp
img = cv2.imread("indiano.png",0)
fourier = np.fft.fft2(img)
center = np.fft.fftshift(fourier)
centerlog =np.log(1+np.abs(center))
plt.figure(figsize=(8,5))
plt.subplot (1,2,1)
plt.imshow(img,cmap=plt.cm.gray),plt.
xticks([]), plt.yticks([])
plt.title('Original'),plt.xticks([]), plt.
yticks([])
plt.subplot (1,2,2)
plt.imshow(centerlog,cmap=plt.cm.gray)
plt.title('Espectro Original'),plt.
xticks([]), plt.yticks([])
plt.show()
```

O Resultado obtido pode ser visualizado na Figura 77(a).

Figura 77 (a) - Imagem original e seu espectro de Fourier



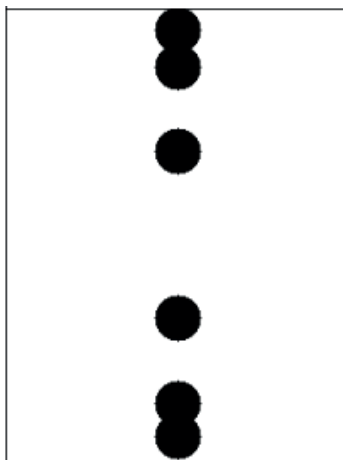
Adaptado de Stack Exchange (2025)

A seguir, devemos criar os filtros, com o objetivo de remover os ruídos do espectro de Fourier.

```
rows, cols = centerlog.shape
crow,ccol = rows//2 , cols//2
mask = np.ones((rows,cols),np.uint8)
#desenhando os círculos no espectro
preto=(0,0,0) #definindo cor
circ1=cv2.circle(mask, (120, 16), 16, preto,-1)
circ2=cv2.circle(mask, (120, 42), 16, preto,-1)
circ3=cv2.circle(mask, (120, 101), 16,
preto,-1)
circ4=cv2.circle(mask, (120, 218), 16,
preto,-1)
circ5=cv2.circle(mask, (120, 278), 16,
preto,-1)
circ6=cv2.circle(mask, (120, 301), 16,
preto,-1)
plt.figure(figsize=(8,5))
plt.imshow(mask,cmap=plt.cm.gray),plt.
xticks([]), plt.yticks([])
plt.show()
```

O filtro gerado é mostrado na Figura 77(b):

Figura 77 (b) - Filtro Notch



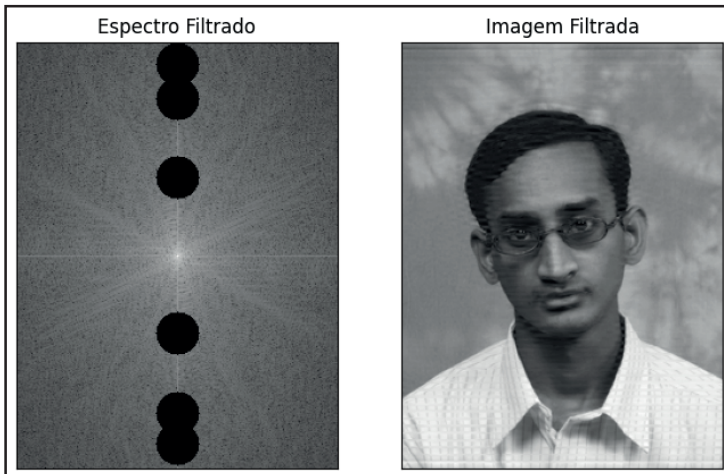
Fonte: O Autor (2026)

Por fim, agora devemos aplicar uma multiplicação de *arrays*, entre o espectro ruidoso, e o filtro projetado acima.

```
fshift=mask*center
f_ishift = np.fft.ifftshift(fshift)
inverse = np.fft.ifft2(f_ishift)
plt.figure(figsize=(8,5))
plt.subplot (1,2,1)
plt.imshow(np.log(1+np.abs(fshift)), cmap=plt.
cm.gray),plt.xticks([], plt.yticks([])
plt.title('\textcolor{red}{Espectro Filtrado}'),plt.
xticks([], plt.yticks([])
plt.subplot (1,2,2)
plt.imshow(np.abs(inverse), cmap=plt.
cm.gray),plt.xticks([],
plt.yticks([])
plt.title('\textcolor{red}{Imagem Filtrada}'),plt.xticks([],
plt.yticks([])
plt.show()
```

O resultado da filtragem pode ser visualizado a seguir. Observe que a filtragem eliminou grande parte do ruído, deixando algumas imperfeições na imagem, que podem ainda ser proveniente de ruídos fora do espectro filtrado, ou então, quando a filtragem remove componentes de baixa frequência.

Figura 77 (c) - Filtro Notch



Adaptado de Stack Exchange (2025)

8. IMAGENS COLORIDAS

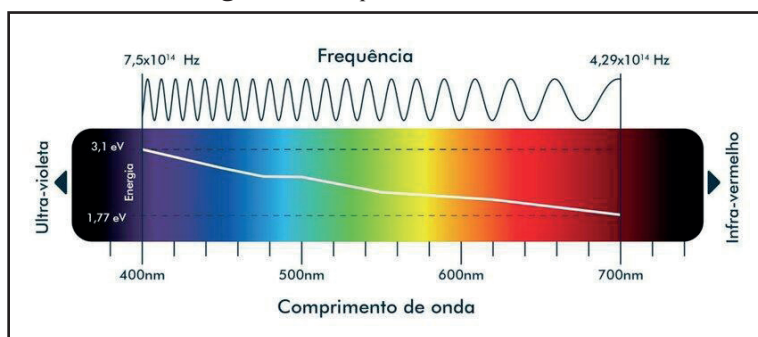
A cor de uma imagem é um poderoso descritor de identificação de objetos e sua extração em uma cena. O olho humano, conforme visto em 2.1, consegue discernir milhares de tons e intensidade de cores, em comparação com apenas duas dúzias de tons de cinza.

O estudo de cores em processamento de imagens/visão computacional é dividido em duas áreas principais: Cores Reais e Pseudocores. As cores reais são adquiridas através de um sensor de imagem real, tal como uma câmera digital, um scanner, etc. Já as pseudocores nada mais são do que uma atribuição de um tom de cor para uma intensidade monocromática particular ou a uma variação de intensidade.

8.1. Fundamento de Cores

A percepção de cor está relacionada com a composição do comprimento de onda da luz que atinge os olhos. A Figura 78 abaixo ilustra como as cores estão relacionadas com o comprimento de onda. A luz visível é composta por uma banda de frequências relativamente estreitas no espectro eletromagnético. No espectro visível nossos olhos são sensibilizados pelas cores respectivas.

Figura 78 - Espectro de luz visível



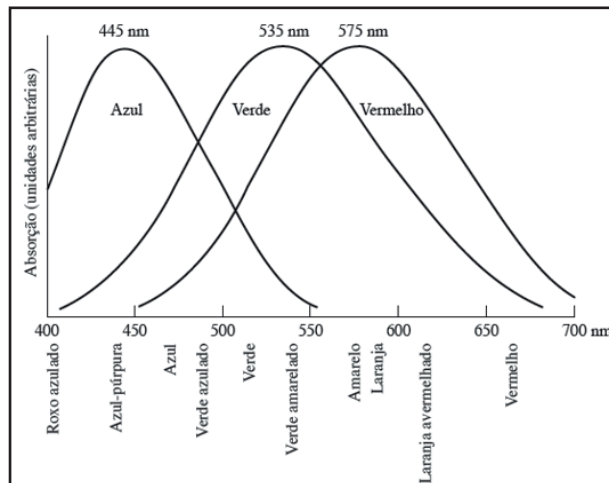
Fonte: Chaves (2021)

A luz acromática, como por exemplo a luz de uma imagem em tons de cinza, possui apenas um atributo: intensidade. A luz cromática, característica de imagens coloridas, possui 3 valores básicos: Radiância, luminância e brilho.

- Radiância: é a quantidade total de energia que flui de uma fonte de luz, dada em watts(W)
- Luminância: é uma medida de quantidade de energia que o observador percebe de fonte de luz, medida em lúmen (lm)
- Brilho: é um descritor subjetivo, que não é tão simples de ser medido. Incorpora função acromática de intensidade, que é um dos fatores chaves na descrição de cores

Devido à estrutura do olho humano, todas as cores são vistas como combinações variáveis das 3 chamadas cores primárias: vermelho, verde e azul (red, green, blue – RGB). Aproximadamente, 65% dos cones são sensíveis a cor vermelha, 33% a luz verde, e 2% a luz azul, entretanto, os cones responsáveis pela cor azul são os mais sensíveis. A Figura 79 ilustra o comprimento de onda dos componentes vermelho, verde e azul, e de outras cores adjacentes.

Figura 79 - Comprimento de onda das principais cores



Fonte: UFPR (2026)

8.1.1 Modelo de Cores

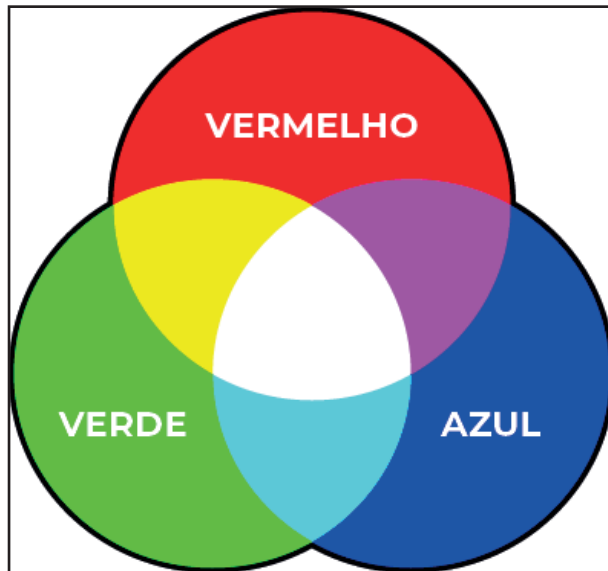
A grande ideia de um modelo de cores (ou sistema de cores) é facilitar a especificação de cores em alguma forma padronizada, amplamente aceita. A grande maioria dos modelos de cores utilizados atualmente é orientada em direção ao hardware, sem desconsiderar as características do sistema visual humano. Os modelos mais comumente estudados são os modelos RGB (red, green, blue – vermelho, verde e azul), CMYk (cyan, magenta, yellow, black – ciano, magenta, amarelo e preto) e HSI (hue, saturation, intensity – matiz, saturação e intensidade).

O modelo RGB é o mais utilizado, principalmente em processamento de imagens/visão computacional, CMYk é o modelo utilizado para impressão colorida, enquanto o modelo HSI corresponde estreitamente a forma como os seres humanos descrevem e interpretam as cores, e é muito utilizado em tarefas que envolvam detecção de cores em visão computacional.

8.1.2. Modelo RGB

O Modelo RGB é um sistema aditivo baseado na teoria dos três estímulos proposta por Young-Helmholtz. Segundo essa teoria, o olho humano percebe a cor através da estimulação dos três pigmentos visuais presentes nos cones da retina, que possuem picos de sensibilidade aproximadamente nos seguintes comprimentos de onda: $730\text{ m}\mu$ (vermelho), $546\text{ m}\mu$ (verde) e $435.8\text{ m}\mu$ (azul). No modelo RGB, a soma $R+G+B=1$, onde o 1 representa a cor branca, que é a soma de todas as cores. As cores primárias podem ser adicionadas entre si e produzir as cores secundárias: magenta, ciano e amarelo. Conforme dito acima, a soma das 3 cores primárias produzem a cor branca, mas é válido ressaltar que a soma de uma cor secundária com sua cor primária oposta também produz a cor branca, conforme ilustra a Figura a seguir.

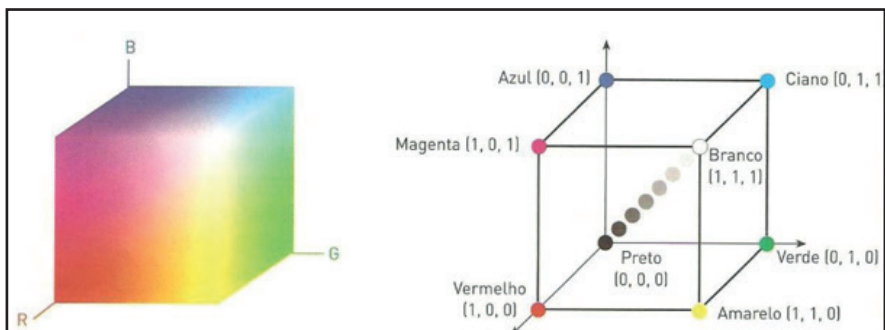
Figura 80 - Mistura de cores



Fonte: O autor (2026)

A Figura 81 apresenta uma representação das cores através do Cubo de Cores no sistema de referência 3-D com eixos representando as cores R, G e B. Os valores de intensidade no cubo variam de 0 a 1. O valor 0 significa ausência total da componente de cor e o valor 1 significa intensidade máxima relacionada à componente.

Figura 81 - Cubo de cores do modelo RGB



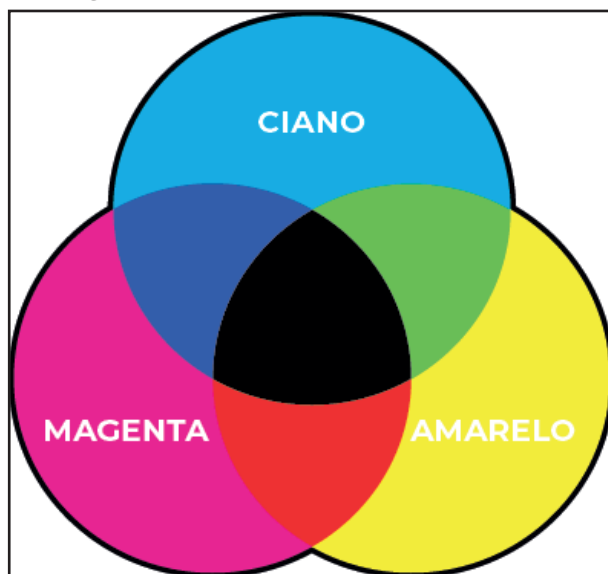
Fonte: Adaptado de KOLKUR et al (2017)

O próximo modelo a ser estudado, é considerado um modelo subtrativo, pois a diferença entre suas cores principais resulta na cor preta, ou seja, na ausência de cor.

8.1.3. Modelo CMYk

O modelo CMYK é um modelo subtrativo de cores, em que $C-M-Y=0$, onde zero representa o preto, representado pela letra k. Ciano, magenta e amarelo são cores secundárias, formadas a partir das cores primárias vermelho, verde e azul. A maior aplicação do modelo CMYK é a de impressão em papel. Esta assenta na sobreposição de camadas de tinta de ciano, magenta, amarelo e preto, representando as áreas brancas a inexistência de tinta e as áreas escuras uma concentração de tinta. A Figura 82 ilustra o modelo CMYK.

Figura 82 - Mistura de cores, modelo CMYk



Fonte: O Autor (2026)

8.1.4. Modelo HSV

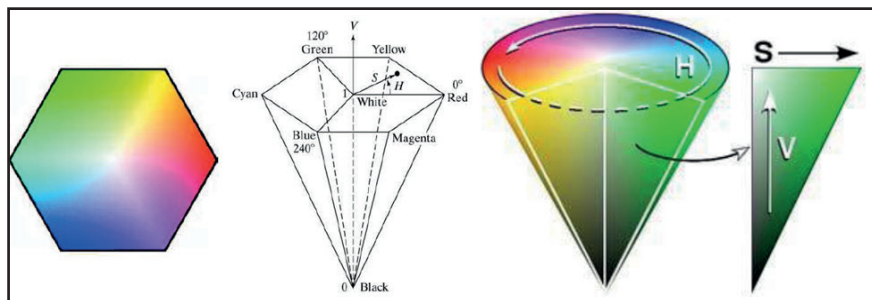
O modelo RGB visto em 8.2.1 possui uma desvantagem: ele não é adequado para definição de cores baseando-se no sistema de percepção visual humano, ou seja, as cores com representação no espaço RGB podem não corresponder às mesmas cores em percepção visual. Fica difícil determinar com exatidão, se uma cor é ou não a de interesse. Para contornar esse tipo de problema, houve a necessidade da criação de Sistemas de Interface de Cor que fossem mais apropriados para interface com o usuário, de forma que pudesse ser determinada uma cor de forma mais intuitiva.

O modelo HSV se baseia em três parâmetros: Hue (Matiz), Saturation, e Value.

- Hue: É a matriz da cor em si, e define a tonalidade dominante de uma área, diferenciando o azul do vermelho, por exemplo.
- Saturação: mede a pureza da cor, é a característica que diferencia a cor rosa da cor vermelha. Enquanto a cor vermelha é a cor pura, a cor rosa é um vermelho adicionado de cor branca.
- Value: O valor no modelo HSV está relacionado a luminância da cor. A luminância é a componente da imagem que contém as informações de brilho de uma imagem. Ou seja, a luminância varia de 0 (preto) a 1 (branco) ao longo do eixo V do cone (Figura abaixo), onde se encontra a escala de cinza. Os matizes puros são encontrados no plano onde a luminosidade é igual a 0.5 e a saturação é igual a 1.

O sistema HSV é representado por coordenadas em um espaço dentro do qual o modelo definido é um cone em forma hexagonal (hexacone), ou, uma pirâmide invertida de seis lados, conforme Figura abaixo. O topo do hexacone (na base) corresponde à $V = 1$, que contém as cores com mais brilho. Obviamente, as cores apresentadas no plano $V = 1$ não possuem todas as percepções possíveis de brilho, podendo ser identificadas por toda a área do hexacone.

Figura 83 - Modelo de Cor HSV



Fonte: Adaptado de KOLKUR et al (2017)

8.2. Imagens coloridas em Python/OpenCV

Conforme vimos em capítulos anteriores, uma imagem em escala de cinza é composta de apenas uma matriz de duas dimensões $[x,y]$, onde x representa a linhas, e y as colunas. Já para imagens coloridas, temos três dessas matrizes de duas dimensões, cada uma representando um canal, ou seja, uma das cores do sistema RGB. Dessa forma, cada pixel é formado de uma t-upla de 3 inteiros de 8 bits no sistema (R,G,B) sendo que (0,0,0) representa o preto, (255,255,255) o branco. A Figura 84 abaixo mostra uma imagem colorida, com seus 3 canais RGB.

Figura 84 - Canais RGB de uma imagem colorida



Fonte: Adaptado de González (2008)

Um detalhe muito importante que não devemos deixar de falar, é que em python, a ordem dos canais de cores não é RGB, e sim, BGR. Dessa forma, sempre que formos trabalhar com uma imagem RGB, deve-se imediatamente convertê-la para BGR, através do código em OpenCv abaixo:

```
imageBGR = cv2.cvtColor(imageRGB, cv2.COLOR_RGB2BGR)
```

O código abaixo ilustra um código em python, em que é realizada a conversão de RGB para BGR. Note que a imagem original RGB possui cores diferentes, justamente porque o python troca o canal R pelo G, e o G pelo R. Após a conversão, a imagem é exibida corretamente, conforme ilustrado na Figura 85.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
imgRGB = cv2.imread('navioescola.png')
imgBGR = cv2.cvtColor(imgRGB, cv2.COLOR_
RGB2BGR)
plt.figure(figsize=(10,6))
plt.subplot (1,2,1),plt.xticks([]), plt.
yticks([])
plt.imshow(imgRGB, cmap=plt.cm.gray)
plt.title('Imagem RGB')
plt.subplot (1,2,2)
plt.imshow(imgBGR, cmap=plt.cm.gray)
plt.title('Imagem BGR'),plt.xticks([]), plt.
yticks([])
plt.show()
```

Figura 85 - Imagem RGB e BRG



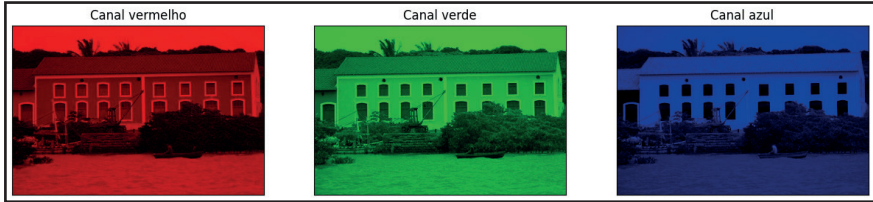
Fonte: O Autor (2025)

Também é possível visualizar cada canal de cor separadamente, criando uma matriz de zeros, e concatenando junto ao canal ao qual se deseja visualização, conforme exemplificado no código abaixo, e ilustrado na Figura 86.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
imagem = cv2.imread('navioescola.png') #
lendo imagem em RGB
x,y,z=imagem.shape #encontrando tamanho da
imagem
zeros=np.zeros((x,y), dtype = "uint8")
#matriz de zeros
(vermelho, verde, azul) = cv2.split(imagem)
#separando os canais RGB
# Gerando as imagens de apenas um canal por
vez
imvermelho=cv2.merge([vermelho, zeros,
zeros]) #canal vermelho
imverde=cv2.merge([zeros, verde, zeros])
#canal verde
imazul=cv2.merge([zeros, zeros, azul])
#canal azul
plt.figure(figsize=(15,10))
plt.subplot (1,3,1)
```

```
plt.imshow(imvermelho)
plt.title('Canal vermelho'), plt.xticks([]),
plt.yticks([])
plt.subplot(1, 3, 2)
plt.imshow(imverde)
plt.title('Canal verde'), plt.xticks([]),
plt.yticks([])
plt.subplot(1, 3, 3)
plt.imshow(imazul)
plt.title('Canal azul'), plt.xticks([]), plt.
yticks([])
plt.show()
```

Figura 86 - Visualização de canais de cor em uma imagem



Fonte: O Autor (2025)

As operações que foram realizadas até antes deste capítulo, tais como equalização, filtragem, restauração, etc, só que em imagens em escala de cinza, pode ser realizada perfeitamente em imagens coloridas, fazendo a operação em cada canal por vez.

A seguir, um código que executa o processo de equalização de histograma de uma imagem colorida, utilizando as técnicas vistas em 4.4.

```
import numpy as np
from matplotlib import pyplot as plt
imagem = cv2.imread('flores.jpg')
imgBGR = cv2.cvtColor(imagem, cv2.COLOR_
RGB2BGR)
x,y,z=imagem.shape #encontrando tamanho da
imagem
zeros=np.zeros((x,y), dtype = "uint8")
(r,g,b) = cv2.split(imagem) #separando os
canais RGB
```

```
red = cv2.equalizeHist(r)
green = cv2.equalizeHist(g)
blue = cv2.equalizeHist(b)
equalized=cv2.merge([blue, green, red])
plt.figure(figsize=(15,10))
plt.subplot (1,2,1)
plt.imshow(imgBGR)
plt.title('Imagem Original'),plt.xticks([]),
plt.yticks([])
plt.subplot (1,2,2)
plt.imshow(equalized)
plt.title('Imagem equalizada'),plt.
xticks([]), plt.yticks([])
plt.show()
```

Figura 87 - Equalização de uma imagem colorida



Fonte: O Autor (2025)

9. MORFOLOGIA MATEMÁTICA

A Morfologia Matemática foi desenvolvida na década de 60, baseada na teoria de conjuntos, a priori para resolver problemas relacionados à análise de imagens microscópicas. Uma vez as imagens segmentadas, era comum algumas imperfeições, principalmente a respeito da forma e estrutura da imagem resultante.

A partir daí, a morfologia matemática tornou-se útil na área de processamento de imagens como um todo, com o objetivo maior de **remover imperfeições** em um processo de segmentação.

A Figura 82 ilustra a segmentação de uma digital, como por exemplo, da cena de um crime. Observa-se que a digital foi extraída com vários componentes de ruído, que podem ser caracterizados como imperfeições do processo de segmentação. A morfologia matemática, através do processo de dilatação (que será visto aqui), conseguiu remover todo o ruído, entregando uma imagem mais nítida, sem imperfeições.

Figura 88 - Segmentação de uma Digital



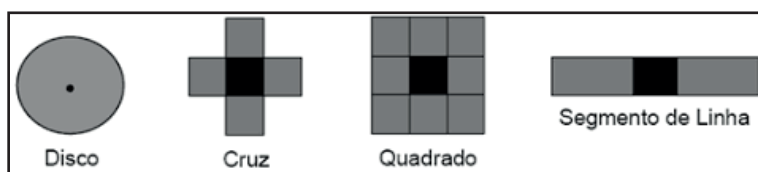
Fonte: Akyol (2023)

A base do processamento morfológico de imagens é a teoria de conjuntos. Definições de união, intersecção, reflexão e translação são úteis para compreender como a ideia foi desenvolvida. Entretanto, como o nosso curso tem o viés mais prático do que teórico, vamos pular essa parte, e focar nas aplicações.

9.1. Elementos Estruturantes

O elemento estruturante é um conjunto definido e conhecido (forma e tamanho), que é usado em uma operação com o conjunto da imagem para salientar determinada forma ou remover imperfeições. Ele pode assumir várias formas dependendo do efeito a ser obtido e sua origem pode ser definida em qualquer ponto. A Figura a seguir ilustra algumas formas de elemento estruturante.

Figura 89 - Formas mais comuns de elementos estruturantes



Fonte: Adaptado de Soille (1999)

Em Python, temos 3 tipos de elementos estruturantes bem utilizados: Retangular, Elipse e Cruz. Abaixo um modelo de código de cada elemento descrito acima, é um exemplo de tamanho 5x5.

Figura 90 - Modelo de código de elemento estruturante em OpenCV

```
# Rectangular Kernel
>>> cv.getStructuringElement(cv.MORPH_RECT, (5,5))
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]], dtype=uint8)

# Elliptical Kernel
>>> cv.getStructuringElement(cv.MORPH_ELLIPSE, (5,5))
array([[0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0]], dtype=uint8)

# Cross-shaped Kernel
>>> cv.getStructuringElement(cv.MORPH_CROSS, (5,5))
array([[0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0]], dtype=uint8)
```

Fonte: O autor (2026)

9.2. Operadores Morfológicos

Nesta seção serão abordados os operadores morfológicos mais comuns, um pouco de sua teoria e principais aplicações. A base da Morfologia Matemática se inicia com os operadores de erosão e dilatação. A partir desses dois, foram desenvolvidos mais quatro operadores, com resultados mais precisos: Abertura, fechamento e gradiente morfológico. Veremos cada um deles com detalhes a seguir, e ao fim do capítulo teremos exemplos em python de cada operador.

9.2.1. Erosão

Sendo A e B duas matrizes em Z^2 , a erosão de A por B, denotada por $A \ominus B$, é dada por:

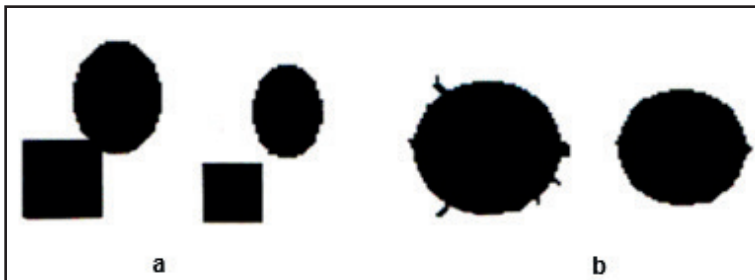
$$A \ominus B = \{x \mid (B)_x \subseteq A\}$$

Sendo A a imagem, e B o elemento estruturante.

A erosão de A por B é o conjunto de todos os pontos x tais que B, quando transladado por x, fique contido em A.

A erosão permite: diminuir as partículas, eliminar grãos de tamanho inferior ao tamanho do elemento estruturante, aumentar os buracos, permitir a separação de grãos próximos, dividir objetos e remover extrusões. A Figura 91 ilustra um processo de erosão. Em a, observa-se que as duas estruturas estavam soldadas, e com o processo de erosão foram separadas. Em b, a forma continha extrusões, e após a erosão, foi retificada. Um detalhe importante a ser salientado é que a erosão pode diminuir significativamente o tamanho do objeto, a depender do tamanho do elemento estruturante.

Figura 91 - Processo de Erosão



Fonte: Akyol (2023)

9.2.2. Dilatação

Sendo A e B duas matrizes em $\mathbb{Z}^2$ e $\emptyset$ o conjunto vazio, define-se a dilatação de A por B , denotada por $A \oplus B$, como:

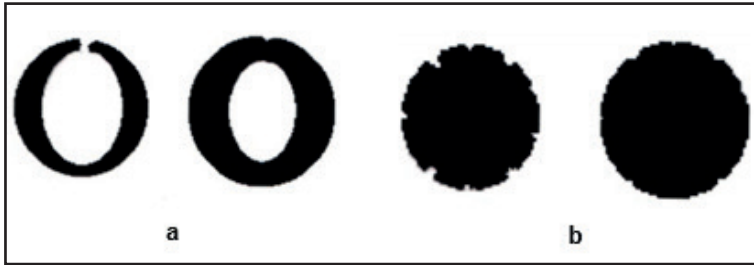
$$A \oplus B = \{x \mid (\hat{B})_x \cap A \neq \emptyset\}$$

Sendo A a imagem, e B o elemento estruturante.

O processo de dilatação começa na obtenção da reflexão de B em torno de sua origem, seguido da translação dessa reflexão por x .

A dilatação permite: aumentar as partículas, preencher buracos negros, conecta grãos próximos, reparar quebras e intrusões. A Figura a seguir ilustra um processo de dilatação. Em a, observa-se que a estrutura continha uma falha, e após o processo de dilatação, a falha foi em grande parte corrigida. Em b, a estrutura continha instruções importantes, que após a dilatação foram amenizadas. De forma contrária à erosão, a dilatação pode aumentar significativamente o tamanho dos objetos, a depender do tamanho do elemento estruturante.

Figura 92 - Processo de dilatação



Fonte: Akyol (2023)

9.3. Combinação de técnicas de morfologia

Técnicas de erosão e dilatação podem ser combinadas, gerando os mais diversos efeitos. As operações compostas mais utilizadas serão vistas a seguir.

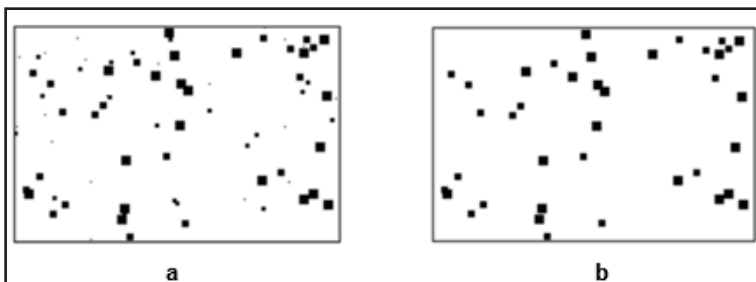
9.3.1. Abertura

A Abertura de uma imagem f por um elemento estruturante s , definida por $f \circ s$ é definida por uma erosão, seguida de uma dilatação:

$$f \circ s = (f \ominus s) \oplus s$$

A operação de abertura suaviza o contorno de uma imagem, quebra istmos e elimina protuberâncias finas, além de eliminar as partículas indesejáveis sem modificar o tamanho das outras entidades. A abertura também pode ser aplicada como filtro para suavizar bordas ou cantos em imagens. A Figura a seguir ilustra um exemplo de abertura em uma imagem. Observe em a uma grande quantidade de pontos pequenos, que podem ser interpretados como ruídos. Em b, após a aplicação da abertura, tais pontos desaparecem.

Figura 93 - Abertura



Fonte: O Autor (2025)

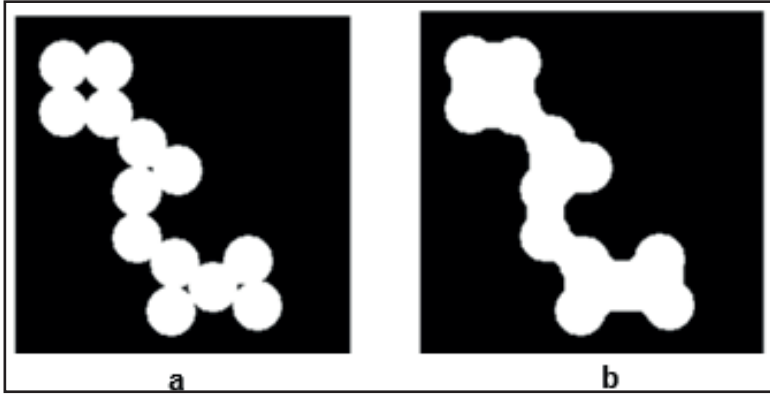
9.3.2. Fechamento

O fechamento de uma imagem f pelo elemento estruturante s definido como $f \bullet s$ é definida por uma operação de dilatação seguida de uma operação de erosão:

$$f \bullet s = (f \oplus s) \ominus s$$

A operação de fechamento conecta objetos próximos sem modificar o tamanho e a forma dos conjuntos, elimina pequenos buracos e preenche fendas em um contorno. A Figura a seguir ilustra o processo de fechamento. Observe que em a, que os círculos não estão bem conectados. Já em b, após o processo de fechamento, os círculos aparecem mais justos.

Figura 94 - Fechamento



Fonte: O Autor (2025)

9.3.3. Gradiente morfológico

O Gradiente Morfológico, denotado por g , pode ser calculado através da seguinte expressão:

$$g = (f \oplus b) - (f \ominus b)$$

Sendo f uma imagem, e b , um elemento estruturante.

O gradiente morfológico é composto de um fechamento seguindo de uma abertura, e possui como efeito enfatizar as transições marcadas nos níveis de cinza da imagem de entrada, conforme ilustra a Figura 95. O resultado da operação vai depender do elemento estruturante utilizado. Para mostrar contornos delgados e bem definidos, o ideal é utilizar um elemento estruturante pequeno.

Figura 95 - Aplicação do gradiente morfológico, para detecção de bordas



Fonte: O Autor (2025)

O código abaixo executa um processo de remoção de ruídos de uma impressão digital, por meio das técnicas de abertura e fechamento descritas acima.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
img=cv2.imread('finger.png',0)
kernel=cv2.getStructuringElement(cv2.MORPH_
ELLIPSE, (5,5))
opening = cv2.morphologyEx(img, cv2.MORPH_
OPEN, kernel)
closing = cv2.morphologyEx(opening, cv2.
MORPH_CLOSE, kernel)
plt.figure(figsize=(12,10))
plt.subplot (1,3,1)
plt.imshow(img,cmap=plt.cm.gray)
plt.title('Original'),plt.xticks([]), plt.
yticks([])
plt.subplot (1,3,2)
plt.imshow(opening,cmap=plt.cm.gray)
plt.title('Abertura'),plt.xticks([]), plt.
```

```
yticks([])
plt.subplot(1,3,3)
plt.imshow(closing,cmap=plt.cm.gray)
plt.title('Fechamento'),plt.xticks([], plt.
yticks([])
plt.show()
```

O resultado pode ser visualizado na Figura abaixo. Observe que a imagem original passa pelas duas técnicas (abertura e fechamento) antes do resultado final, no qual todos os ruídos são eliminados.

Figura 96 - Aplicação das técnicas de abertura e fechamento para eliminação de ruídos em uma imagem de impressão digital



Fonte: Adaptado de Akyol (2023)

Uma excelente aplicação da morfologia matemática é a detecção de bordas. Essa abordagem mostra-se particularmente útil, pois utiliza um código bem simples, sobretudo quando comparados a técnicas de detecção de bordas mais complexas e robustas.

O código abaixo executa uma aplicação de morfologia, com kernels 3x3, e 5x5.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
img = cv2.imread('lenna.jpg')
kernel1=cv2.getStructuringElement(cv2.MORPH_
ELLIPSE, (3,3))
kernel2=cv2.getStructuringElement(cv2.MORPH_
```

```
ELLIPSE, (5,5))
gradient1 = cv2.morphologyEx(img, cv2.MORPH_
GRADIENT, kernel1)
gradient2 = cv2.morphologyEx(img, cv2.MORPH_
GRADIENT, kernel2)
plt.figure(figsize=(10,7))
plt.subplot(1,3,1)
plt.imshow(img, cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.
yticks([])
plt.subplot(1,3,2)
plt.imshow(gradient1, cmap = 'gray')
plt.title('Gradiente Morfológico 3x3'), plt.
xticks([]), plt.yticks([])
plt.subplot(1,3,3)
plt.imshow(gradient2, cmap = 'gray')
plt.title('Gradiente Morfológico 5x5'), plt.
xticks([]), plt.yticks([])
plt.show()
```

Figura 97 - Aplicação do gradiente morfológico, com kernels de 3x3 e 5x5

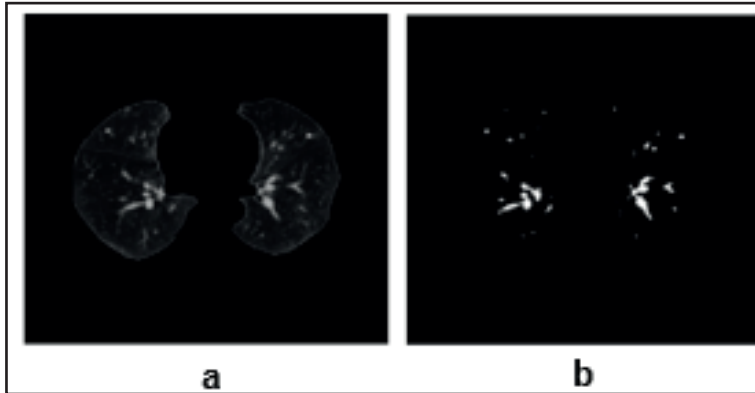


Fonte: Adaptado de Lenna Image (2025)

10. SEGMENTAÇÃO DE OBJETOS

O processo de segmentação em imagens, visa extrair a informação útil de uma imagem, colocando as demais estruturas em background. A informação útil da imagem, que pode ser chamado de ROI – Region on Interest – Região de interesse, pode ainda ser alvo de outros processamentos. A Figura a seguir ilustra o processo de segmentação de prováveis nódulos pulmonares, para posterior análise e confirmação de diagnóstico.

Figura 98 - Tomografia de pulmão, e a visualização de possíveis nódulos pulmonares (a). Em (b), observa-se apenas os prováveis nódulos segmentados, para melhor visualização.



Fonte: De Moura (2015)

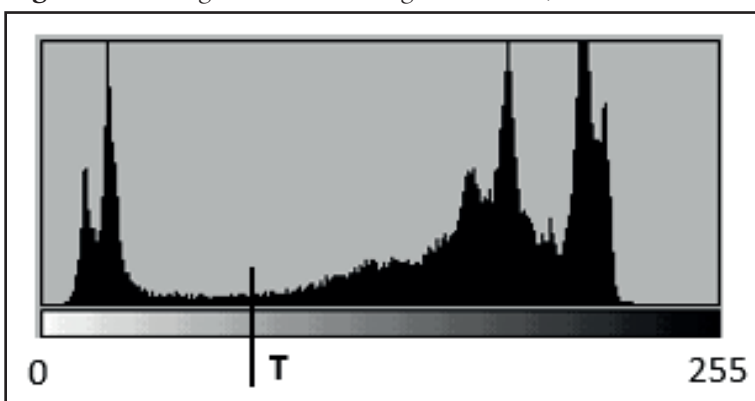
De um modo geral, a segmentação é um processo de binarização, onde a região de interesse é transformada na cor branca, e o background na cor preta.

Existem várias técnicas de segmentação atualmente, mas podemos destacar 3 tipos mais comuns: limiarização, binarização por Otsu, e transformada de Hough.

10.1. Segmentação por Limiarização

A segmentação por limiarização consiste em encontrar um limiar em um histograma de 256 níveis de cinza, que melhor separa as fases associadas às fases em questão. Neste caso mais comum, a definição do limiar é realizada por tentativa e erro. A Figura a seguir ilustra um histograma bimodal, ou seja, um histograma que possui somente dois picos, e um valor de limiarização T encontrado de forma experimental.

Figura 99 - histograma de uma imagem de 8 bits, em níveis de cinza



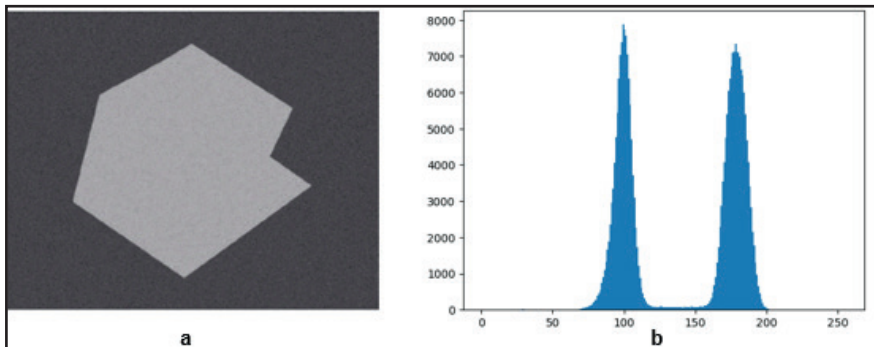
Fonte: O autor (2025)

Dessa forma, uma imagem limiarizada pode ser definida como:

$$g(x, y) = \begin{cases} 1, & \text{se } f(x, y) > T \\ 0, & \text{se } f(x, y) < T \end{cases}$$

Dadas condições de iluminação satisfatórias, e baixo ruído, a limiarização pode ser resolvida conforme definição acima. A Figura 100 ilustra uma imagem com boa iluminação, e seu histograma correspondente.

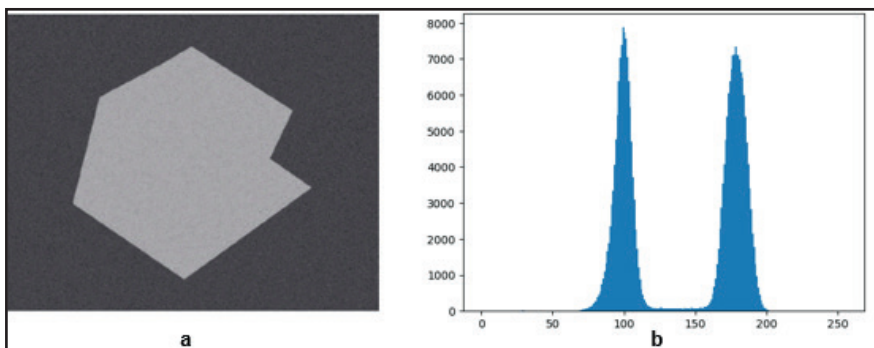
Figura 100 - histograma de uma imagem em condições ideais de contraste



Fonte: Adaptado de González (2001)

Na Figura, pode-se observar uma clara separação entre os dois picos do histograma, que facilita a escolha de um limiar, a fim de efetuar a limiarização. Entretanto, quando as condições de iluminação e ruído não são satisfatórias, o histograma se comporta conforme a Figura 101 abaixo:

Figura 101 - histograma de uma imagem em condições ruins de contraste



Fonte: Adaptado de González (2001)

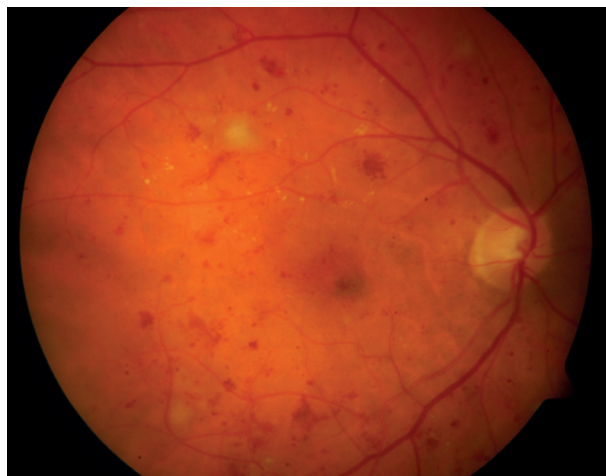
Para o caso da Figura acima, observa-se que o histograma não possui uma zona de separação entre os picos, ou seja, a decisão de se escolher um limiar torna-se mais difícil. Dessa forma, deve-se utilizar uma limiarização adaptativa, que vai dividir a imagem em blocos $N \times M$, aplicar

a limiarização (em cada bloco separado) e depois unir todos, formando a imagem binarizada.

Uma excelente aplicação de segmentação por binarização em imagens com condições ruins de contraste, pode ser visto em uma retinografia. A retinografia é um exame que mostra detalhadamente o fundo do olho (retina, nervo óptico e vasos sanguíneos), permitindo o diagnóstico precoce e acompanhamento de doenças oculares como retinopatia diabética, glaucoma, degeneração macular e hipertensão, muitas vezes antes que os sintomas apareçam, prevenindo problemas graves, como por exemplo, a perda de visão.

A Figura a seguir ilustra uma retinografia colorida.

Figura 102 - Retinografia colorida



Fonte: O autor (2025)

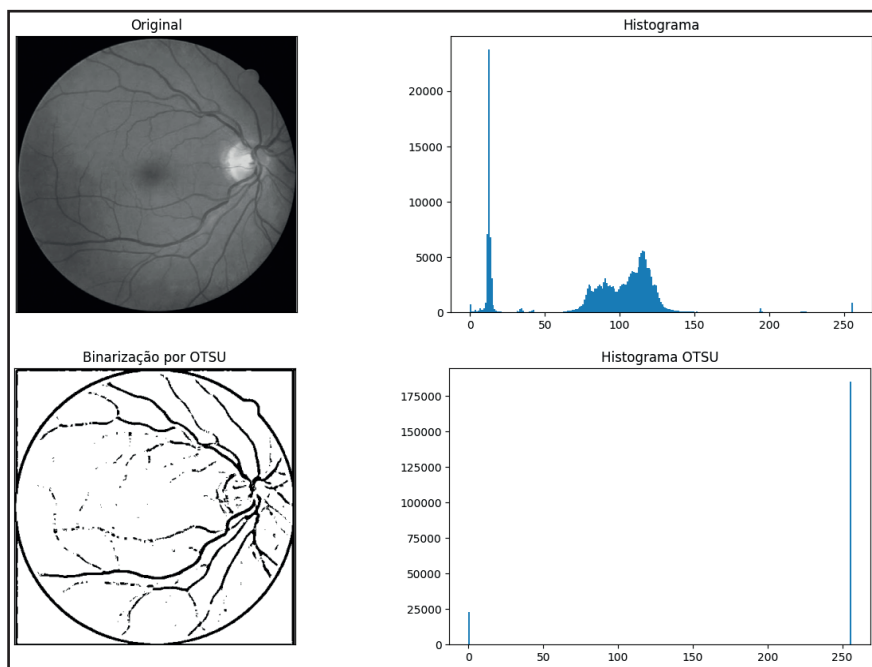
Pode-se observar claramente alguns poucos vasos, mas os vasos menores, e as manchas provenientes de derrames, não são visualizadas facilmente, devido ao baixo contraste da imagem.

O código a seguir realiza a segmentação de vasos sanguíneos em imagens de retinografia, com o objetivo de identificar vasos de menor calibre e microvasos, possibilitando a detecção de alterações associadas a patologias, como a retinopatia diabética.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('olho.PNG', 0)
imgGauss = cv2.medianBlur(img, 7)
#suavização para melhorar
#resultado
imgBin = cv2.adaptiveThreshold(imgGauss,
255,
cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_
BINARY, 9,
3)
plt.figure(figsize=(15,10))
plt.subplot (2,2,1)
plt.imshow(img,cmap=plt.cm.gray), plt.
xticks([]), plt.yticks([])
plt.title('Original')
plt.subplot (2,2,2)
plt.hist(img.ravel(), 256, [0,256])
plt.title('Histograma')
plt.figure(figsize=(15,10))
plt.subplot (2,2,1)
plt.imshow(imgBin,cmap=plt.cm.gray), plt.
xticks([]), plt.yticks([])
plt.title('Binarização por OTSU')
plt.subplot (2,2,2)
plt.hist(imgBin.ravel(), 256, [0,256])
plt.title('Histograma OTSU')
plt.show
```

A Figura 103 apresenta o resultado obtido a partir da execução do código descrito anteriormente. Observa-se que a imagem original, já convertida para escala de cinza, apresenta um histograma característico de baixo contraste, o que dificulta o processo de segmentação por limiarização. Dessa forma, por meio das técnicas aplicadas no código, foi possível redistribuir os níveis do histograma, resultando na geração de uma imagem binária com melhor definição e visualização das estruturas da retina.

Figura 103 - Segmentação por limiarização de uma retinografia



Fonte: O autor (2025)

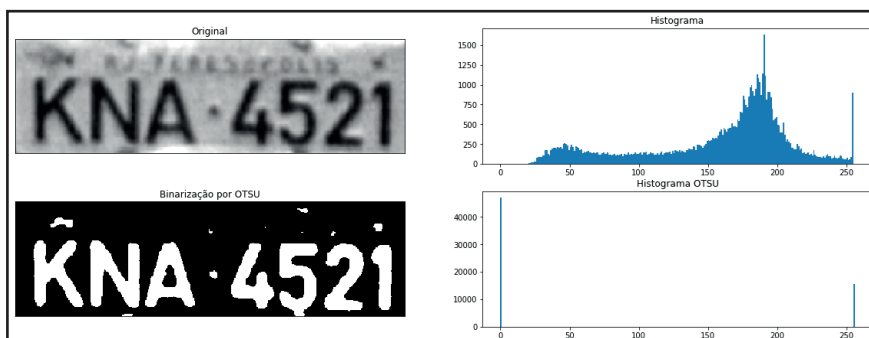
10.2. Método de Otsu

Técnica que determina automaticamente um limiar ótimo considerando uma imagem, que apresenta melhor funcionamento em imagens cujos histogramas são bimodais.

A ideia é aproximar o histograma de uma imagem por duas funções Gaussianas e escolher o limiar de forma a minimizar a variância intraclasses. Cada classe possui suas próprias características, ou seja, sua média e desvio-padrão.

A Figura a seguir ilustra a aplicação do método de OTSU em uma placa de carro.

Figura 104 - Segmentação por meio do Método de OTSU



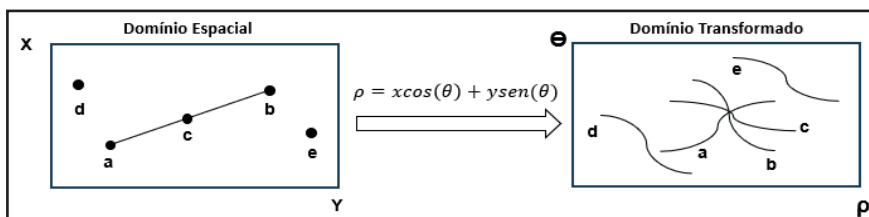
Fonte: Adaptado de Aconci (s.d.)

10.3. Transformada de Hough

A transformada de Hough foi desenvolvida por Paul Hough em 1962 e patenteada pela IBM, originalmente, para detectar características analiticamente representáveis em imagens binarizadas, assim como linhas, círculos e elipses, e nos últimos anos tornou-se uma ferramenta de uso comum na visão computacional.

A ideia principal da Transformada de Hough é aplicar na imagem uma transformação tal que todos os pontos pertencentes a uma mesma curva sejam mapeados num único ponto de um novo espaço de parametrização da curva procurada, conforme ilustrado na Figura 105.

Figura 105 - Transformação entre espaços geométricos



Fonte: O autor (2026)

Para realizar a transformação de espaços, usaremos a equação da reta, dada por:

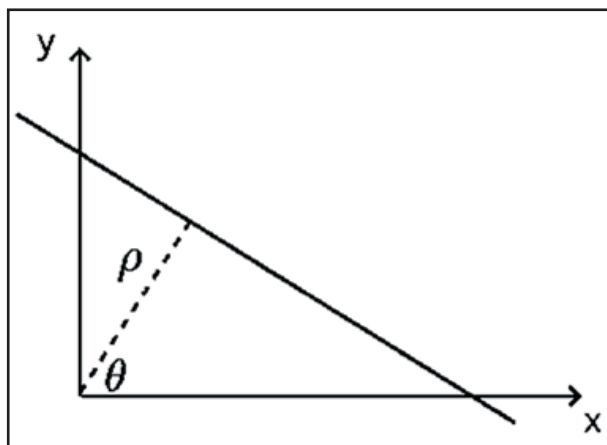
$$\rho = x \cos \theta + y \sin \theta$$

Sendo:

- ρ : distância da reta até a origem
- θ : ângulo entre o eixo x e a reta entre origem e ponto mais próximo da origem.

A Figura 106 abaixo ilustra melhor a equação da reta acima

Figura 106 - Espaço de parâmetros



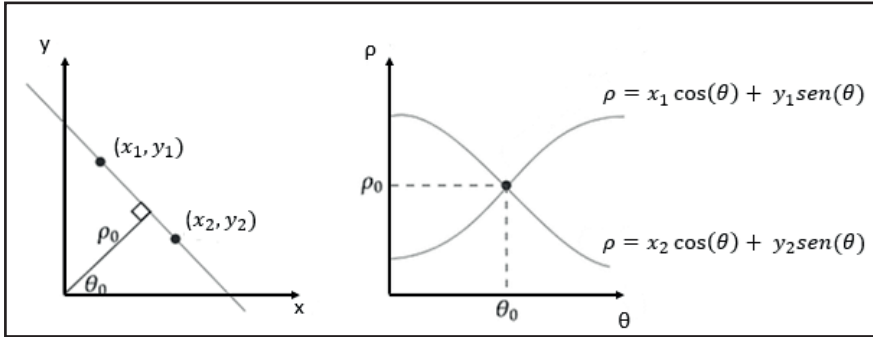
Fonte: O autor (2026)

10.3.1. Transformada de Hough na detecção de retas

Para todo ponto (x, y) não nulo em uma imagem, a matriz cumulativa (espaço transformado) será incrementada com a tupla (θ, ρ) , variando θ de 0 até o valor definido pelo usuário. Dessa forma, para cada ponto (x, y) , terá uma senóide no acumulador.

A Figura a seguir ilustra um exemplo com dois pontos, e o comportamento das senóides no espaço transformado.

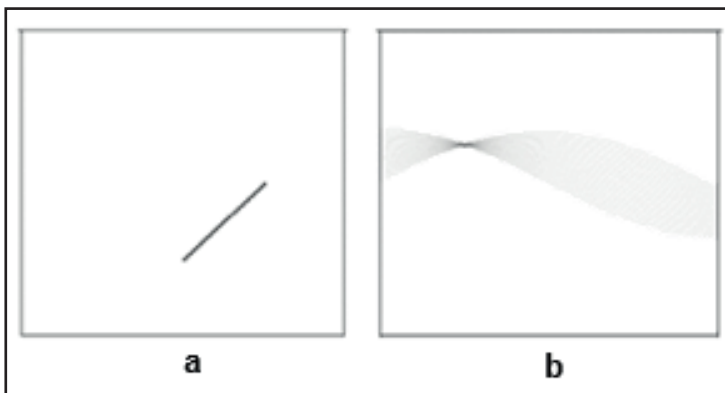
Figura 107 - Transformada de Hough na detecção de Retas. No gráfico de domínio espacial, temos dois pontos que fazem parte de uma reta, e no gráfico de domínio transformado, temos duas senoídes (cada uma corresponde a um ponto) que possuem um ponto de intersecção, o que sugere que os dois pontos pertencem à mesma reta.



Fonte: O autor (2026)

Em uma reta, temos uma quantidade finita de pontos, que gera uma quantidade finita de retas, que possuem um mesmo ponto de intersecção, conforme ilustra a Figura abaixo.

Figura 108 - Transformada de Hough na Detecção de Retas. Em (a), temos um conjunto finitos de pontos, que representam uma reta. Em (b), observa-se finitos senoídes que possuem o mesmo ponto de intersecção. O ponto de intersecção representa a reta.



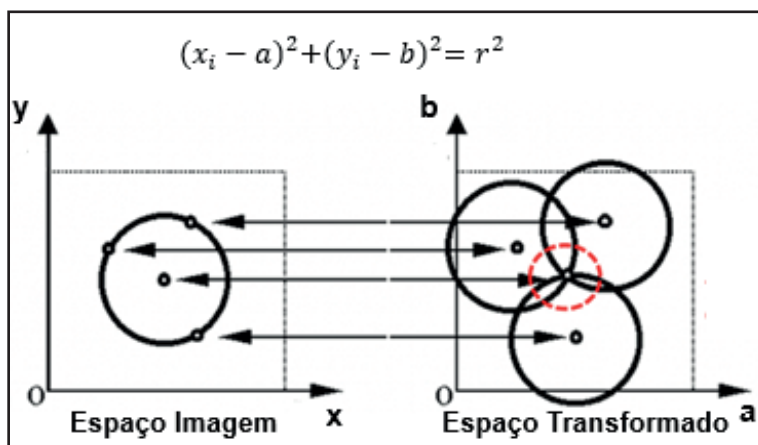
Fonte: O autor (2026)

10.3.2. Transformada de Hough na detecção de Círculos

Semelhante à detecção de linhas, desta vez definimos um espaço de Hough para o círculo, que é parametrizado por (a, b, r) , sendo (a,b) o centro, e r o raio.

Se r for conhecido, o espaço de Hough torna-se bidimensional, e funciona de forma similar à detecção de linhas. A transformada de Hough de um círculo desenha, no espaço de Hough, todas as localizações possíveis do centro do círculo para cada ponto de borda (x, y) no espaço - imagem. O conjunto de todos os centros de círculo possíveis, que é a saída da transformada de Hough, dado um ponto de borda (x, y) com raio fixo r , é novamente um círculo no espaço de Hough, como ilustrado abaixo. Então, o centro do círculo que procuramos, no espaço da imagem, é determinado pelo parâmetro (a,b) no qual o vetor acumulador $H[a,b]$ obtém o maior número de intersecções.

Figura 109 - Transformada de Hough na Detecção de círculos No espaço-imagem, temos um conjunto finitos de pontos, que representam um círculo. Já no espaço transformado, observa – se finitos senóides que possuem o mesmo ponto de intersecção. O ponto de intersecção representa a origem do círculo.

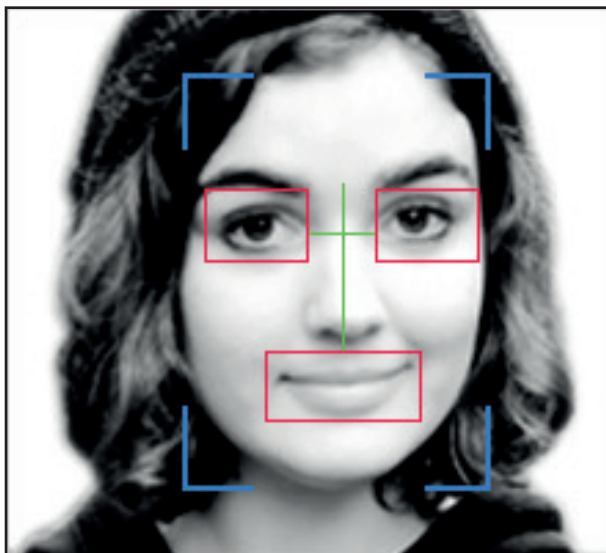


Fonte: O autor (2026)

11. EXTRAÇÃO DE CARACTERÍSTICAS

O grande objetivo de extração de características em visão computacional, é de obter informações que tornem possíveis a identificação e classificação de objetos. Em um grupo de imagens, pode-se identificar, e posteriormente extrair padrões, que podem ser utilizados para diferenciar e/ou classificar imagens ou classes. A Figura 110 abaixo ilustra o uso de extração de características de olhos, nariz e boca, para detecção facial, assuntos que exploraremos em breve no nosso curso.

Figura 110 - Modelo de extração de características para detecção facial



Fonte: Prasad (2014)

Tais características das imagens, que são utilizadas para a classificação/diferenciação, são aqui chamadas de descritores, que podem ser categorizados como descritores de:

- A. Aspectos (Cor e textura)
- B. Dimensionais
- C. Inerciais
- D. Topológicas

Nos itens a seguir, falaremos sucintamente sobre cada um deles.

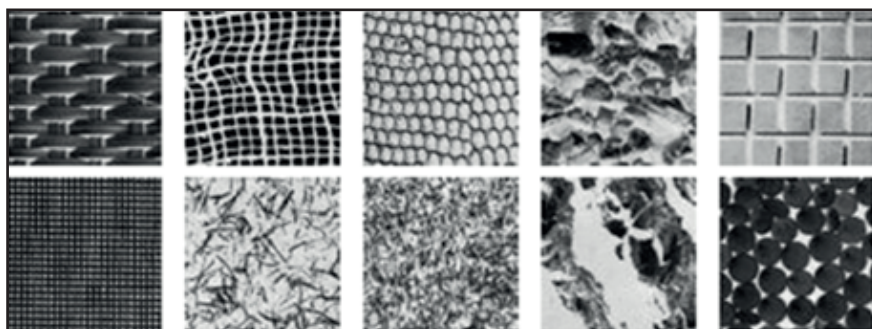
11.1. Descritor de aspectos

São relacionados à forma, dimensão, cores e textura. Para realizar a extração de características de aspectos, pode-se usar as mais variadas técnicas, como por exemplo, os descritores de textura de Haralick.

A textura é uma característica que pode ajudar a segmentar as imagens e classificar regiões de interesse.

Segundo Gonzalez, a textura pode ser definida como um conjunto de determinados padrões existentes nas superfícies físicas, perceptíveis ao olho humano, e que trazem grande quantidade de informações sobre a natureza da superfície, como suavidade e rugosidade. A Figura 111 ilustra um mosaico, com diferentes padrões de textura.

Figura 111 - Exemplos de textura em imagens



Fonte: Haralick (1973)

Os descritores de textura de Haralick descrevem um método para extração de características de texturas baseado na relação espacial existentes entre os níveis de cinza da imagem, calculando as chamadas matrizes de co-ocorrência e, a partir delas, determinaram 14 parâmetros estatísticos que caracterizam as texturas.

Um descritor de textura ou de Haralick é um valor calculado a partir da matriz co-ocorrência da imagem, que quantifica algumas características da variação dos níveis de cinza dessas imagens. As texturas definem características como uniformidade, densidade, aspereza, regularidade, intensidade, entre outras características de uma imagem. A classificação correta de qualquer textura depende dos descritores utilizados para caracterizá-la da melhor forma possível.

A matriz de co-ocorrência é o ponto de partida para a obtenção dos 14 descritores estatísticos que foram propostos, para representar características presentes em texturas:

- Segundo Momento Angular;
- Contraste;
- Correlação;
- Soma dos Quadrados – Variância;
- Momento de Diferença Inverso;
- Média da Soma;
- Variância da Soma;
- Entropia da Soma;
- Entropia;
- Variância da Diferença;
- Entropia da Diferença;
- Medidas de Informação da Correlação (2);
- Coeficiente de Correlação Máximo;

Essas características servem como medida para a diferenciação de texturas que não seguem um determinado padrão de repetitividade, fornecendo informações relevantes para a classificação das mesmas.

11.2. Descritores Dimensionais

Outra forma de se extrair características de uma imagem, é baseado nas suas dimensões, como por exemplo, área e perímetro, que podem auxiliar na discriminação de objetos diferentes.

Em python, temos 3 funções que podem auxiliar na extração da área e do perímetro de um objeto:

- *findContours*;
- *contourArea*;
- *arcLength*;

A função ***findContours*** é descrita da seguinte forma:

contorno, hierarquia = cv2.findContours(imgBin, modo, metodo)

Sendo:

- *img* = Imagem com a qual queremos trabalhar binária.
- *modo* = pontos extraídos da imagem serão armazenados(cv2.RETR_TREE)
- *metodo* = pontos da imagem binária(cv2.CHAIN_APPROX_NONE, cv2.CHAIN_APPROX_SIMPLE).
- *contorno* = Todos os contornos encontrados
- *hierarquia* = contornos em forma de lista

Após a aplicação da função acima, devemos utilizar a função ***contourArea***:

area = contourArea(Saida_findContours)

Sendo:

- *Saida_findContours* = Objeto segmentado obtido pela função *findContours*
- *area* = Área do objeto estudado.

E para encontrar o perímetro do objeto, utiliza-se a função ***arcLength***, descrita como:

perímetro=arcLength(Saida_findContours, contornos)

Sendo:

- *Saida_findContours* = Objeto segmentado obtido pela função *findContours*
- *contornos* = contorno que será obtido (TRUE = contorno fechado, FALSE = contorno não fechado)
- *perímetro* = perímetro do objeto estudado.

Uma excelente aplicação de descritores dimensionais é na detecção de formas de objetos. Entretanto, precisamos estudar primeiramente os descritores inerciais, que vêm logo a seguir:

11.3. Descritores Inerciais

São relacionadas as medidas dos momentos inerciais, centro geométrico, e forma dos objetos, que podem até, por exemplo, identificar objetos que tenham sofrido alterações em escala, rotação e translação.

Podemos calcular a área e o centróide de objetos utilizando momentos inerciais bidimensionais de uma imagem $f(x,y)$, de tamanho $M \times N$, definido como:

$$m_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} x^p y^q f(x, y)$$

Onde p e q correspondem a ordem dos momentos.
A área de um objeto (m_{00}) pode ser definido por:

$$m_{00} = \sum_m \sum_n B(m, n)$$

E os centróides (i_0, j_0):

$$i_0 = \sum_{k=1}^n B(i_k, j_k) i_k / \sum_{k=1}^n B(i_k, j_k) = m_{10} / m_{00}$$

$$j_0 = \sum_{k=1}^n B(i_k, j_k) j_k / \sum_{k=1}^n B(i_k, j_k) = m_{01} / m_{00}$$

Em python, podemos encontrar os momentos da imagem através da função **cv2.moments**.

momentos = cv2.moments(img)

Sendo:

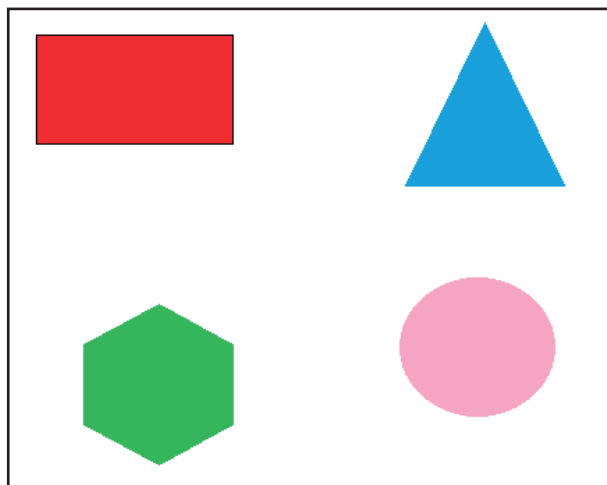
- Img: imagem
- Momentos: descreve os 24 momentos inerciais

O código a seguir realiza a detecção de formas geométricas, utilizando as funções estudadas neste capítulo:

Utilizaremos como exemplo, uma imagem colorida, que possui quatro Figuras geométricas, de cores distintas. O objetivo do algoritmo é detectar os contornos das Figuras geométricas, e escrever se tal Figura é um retângulo, triângulo, pentágono ou círculo.

A imagem do teste é ilustrada abaixo.

Figura 112 - Imagem teste para detecção de formas



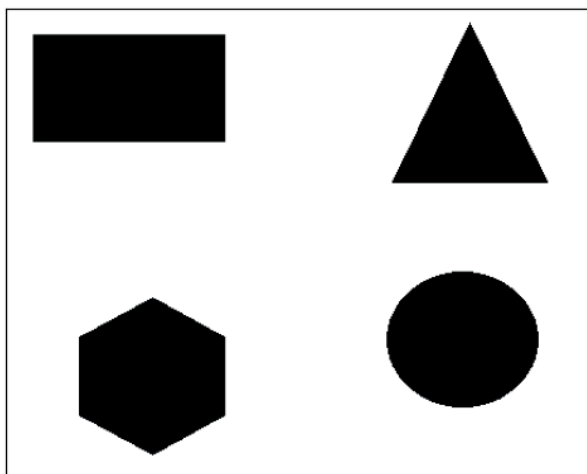
Fonte: O autor (2026)

O primeiro passo do algoritmo, é carregar as bibliotecas, converter a imagem para escala de cinza, e logo a seguir, para eliminar ruídos, executar uma pequena segmentação por limiarização, por meio de um threshold.

```
import cv2
import numpy as np
import imutils
from matplotlib import pyplot as plt
img = cv2.imread('ShapeColor.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
_, threshold = cv2.threshold(gray, 250, 255,
cv2.THRESH_BINARY)
```

Após a execução do trecho do código acima, a imagem original é transformada em uma imagem binária, conforme ilustrado na Figura abaixo.

Figura 113 - Imagem binarizada



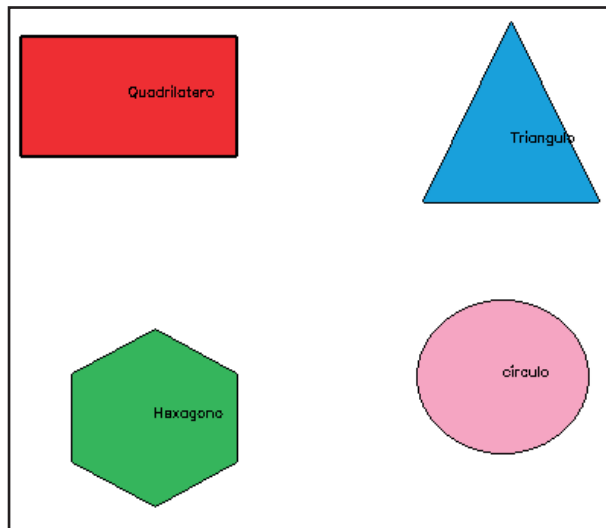
Fonte: O autor (2026)

Os próximos passos do algoritmo, resumem-se em encontrar o contorno de cada Figura geométrica, o centróide, e a partir daí, verificar a quantidade de cantos.

```
contours, _ = cv2.findContours(threshold, cv2.RETR_
TREE, cv2.CHAIN_APPROX_SIMPLE)
i = 0
for contour in contours:
    if i == 0:
        i = 1
        continue
    approx = cv2.approxPolyDP(contour, 0.01 * cv2.
arcLength(contour, True), True)
    cv2.drawContours(img, [contour], 0, (0, 0, 0), 1)
    M = cv2.moments(contour) # Cálculo dos momentos
    if M['m00'] != 0.0: #Verificando se a massa é
diferente de zero
        # Cálculo do centróide
        x = int(M['m10']/M['m00'])
        y = int(M['m01']/M['m00'])
        if len(approx) == 3:
            cv2.putText(img, 'Triangulo', (x, y),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.3, (0,
0, 0), 1)
        elif len(approx) == 4:
            cv2.putText(img, 'Quadrilatero', (x, y),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.3, (0,
0, 0), 1)
        elif len(approx) == 5:
            cv2.putText(img, 'Pentagono', (x, y),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.3, (0,
0, 0), 1)
        elif len(approx) == 6:
            cv2.putText(img, 'Hexagono', (x, y),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.3, (0,
0, 0), 1)
        else:
            cv2.putText(img, 'circulo', (x, y),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.3, (0,
0, 0), 1)
    #final=cv2.drawContours(img, [contour], 0, (0, 0, 255),
2)
plt.imshow(img), plt.xticks([]), plt.yticks([])
plt.show()
```


O resultado final do experimento pode ser visualizado na Figura 113. Note que o texto que indica qual Figura geométrica é inserido no centróide da Figura, calculado nos passos do algoritmo.

Figura 114 - Imagem original com contornos e formas geométricas detectadas



Fonte: O autor (2026)

Nos próximos capítulos do nosso curso, falaremos sobre as aplicações bem usuais de visão computacional, aplicando, na grande maioria delas, toda a teoria e códigos vistos até aqui. Seremos mais “mão na massa”, embora em alguns momentos seja necessário inserir um pouco mais de teoria.

12. APLICAÇÕES

O presente capítulo apresenta aplicações introdutórias de visão computacional, baseado na teoria e ferramentas vistas até aqui.

12.1. Detecção de placas

Nessa primeira aplicação, veremos o passo a passo de um algoritmo de detecção de placas de carro em uma cena e logo em seguida reconhecer os caracteres do objeto (no caso a placa de um carro). Sendo assim, vamos explicar passo a passo, quebrando o código em partes, para facilitar o entendimento.

A nossa tarefa será detectar e extrair os caracteres da placa do carro ilustrada na Figura a seguir:

Figura 115 - Imagem teste do algoritmo de detecção de placas



Fonte: Autor desconhecido

O primeiro passo consiste em importar as bibliotecas necessárias para a tarefa. Além das bibliotecas usuais, como **numpy**, **cv2**, e **matplotlib**, utilizaremos também as bibliotecas **imutils** e **pytesseract**.

A biblioteca **imutils** possui vários códigos que podem ser necessários para: zoom, cortar, redimensionar, fazer translação na imagem, e outros.

Já a **pytesseract** é uma biblioteca da google que trabalha como um OCR (Reconhecedor óptico de caracteres) adaptado para imagens. Seu uso é gratuito e de fácil instalação e operação.

Em seguida, devemos carregar a imagem, converter para BGR (lembrando que o python trabalha em BGR e não RGB), redimensionar a imagem com o comprimento de 500 pixels, converter para escala de cinza, e logo depois mostrar a imagem, conforme código abaixo:

```
import numpy as np
import cv2
import imutils
import pytesseract
import pytesseract as ocr
from matplotlib import pyplot as plt
original=cv2.imread("placa2.jpg")
original=cv2.cvtColor(original, cv2.COLOR_
RGB2BGR)
original=imutils.resize(original, width=500)
gray=cv2.cvtColor(original, cv2.COLOR_
BGR2GRAY)
plt.imshow(gray,cmap=plt.cm.gray), plt.
xticks([], plt.yticks([]))
plt.show()
```

A imagem, já redimensionada, e em escala de cinza, pode ser visualizada a seguir.

Figura 116 - imagem redimensionada, e em escala de cinza



Fonte: Adaptado de autor desconhecido (2026)

O próximo passo é remover qualquer ruído ou imperfeição da imagem, sem perder as bordas. Essa é uma tarefa muito delicada, e vários tipos de filtros podem ser aplicados aqui. Como nossa imagem já está bem definida, vamos apenas aplicar um filtro de suavização, para garantir que qualquer ruído simples seja eliminado. Dessa forma, iremos utilizar um filtro gaussiano, de kernel 3x3, conforme ilustra o trecho do código abaixo:

```
img_filter=cv2.GaussianBlur(gray, (3,3), 1)
plt.imshow(img_filter,cmap=plt.cm.gray), plt.
xticks([]), plt.yticks([])
plt.show()
```

A Figura 117 ilustra a imagem teste após a aplicação do filtro do código acima. Observe que a diferença entre as imagens é bem sutil, pois não temos imperfeições significativas na imagem de teste original.

Figura 117 - imagem teste, após aplicação do filtro gaussiano



Fonte: Adaptado de autor desconhecido (2026)

Logo em seguida passaremos para as etapas mais cruciais da tarefa. O nosso próximo objetivo é fazer a detecção das bordas da imagem como um todo, para depois detectar apenas as bordas da placa, e posteriormente dos caracteres. Para tanto, utilizaremos o detector de bordas de canny, visto em **5.5.3**.

```
canny=cv2.Canny(img_filter, 200, 230)
plt.imshow(canny,cmap=plt.cm.gray),      plt.
xticks([]), plt.yticks([])
plt.show()
```

A imagem, já com as bordas detectadas e bem definidas, é mostrada em seguida. Observe que as letras das placas já são nitidamente destacadas, e que futuramente facilitará no processo de reconhecimento dos caracteres.

Figura 118 - imagem teste, após aplicação do filtro de detecção de bordas



Fonte: Adaptado de autor desconhecido (2026)

Agora, deveremos segmentar apenas a placa do carro, ou seja, ignorar o restante da imagem, e preservar somente a região de interesse, que no nosso caso, é a placa do carro. Utilizaremos a função ***findcontours***, para encontrar os contornos, e a ***drawcontours***, para desenhá-los, conforme Figura a seguir:

```
contornos, _ = cv2.findContours(canny.copy(),
cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
draw = cv2.drawContours(original.copy(),
contornos, -1, (0,255,0), 4)
plt.imshow(draw, cmap=plt.cm.gray), plt.
xticks([]), plt.yticks([])
plt.show()
```

O resultado retornado do algoritmo, são contornos desenhados e cada possível contorno da imagem, como é visto em 118.

Figura 119 - Detecção de contornos aplicada na imagem segmentada por canny



Fonte: Adaptado de autor desconhecido (2026)

Pode-se observar da Figura acima, que foram encontrados vários contornos, e a grande maioria não serão necessários para a detecção somente da placa do carro. Então, utilizaremos a função ***sorted*** para reduzir os contornos encontrados, somente os contornos com área até 30 pixels.

```
contornos=sorted(contornos,key=cv2.  
contourArea, reverse=True)[:30]  
#Reduzindo para contornos de área até 30  
draw30 = cv2.drawContours(original.copy(),  
contornos, -1, (0,255,0), 3)  
plt.imshow(draw30,cmap=plt.cm.gray), plt.  
xticks([], plt.yticks([]))
```

Observe, no resultado apresentado na Figura abaixo, que, a placa e mais dois pequenos contornos foram preservados.

Figura 120 - Redução de contornos detectados



Fonte: Adaptado de autor desconhecido (2026)

Agora vamos isolar somente o retângulo que delimita a informação que precisamos da placa, por meio da função ***boundingrect***.

```
NumPlacaCnt = None
count = 0
idx = 1
for c in contornos:
    peri = cv2.arcLength(c, True)
    approx = cv2.approxPolyDP(c, 0.01*peri, True)
    if len(approx) == 4:
        NumPlacaCnt = approx
        x,y,w,h = cv2.boundingRect(c)
        new_img = gray[y:y+h,x:x+w]
        cv2.imwrite('Placa' + str(idx) + '.png' , new_img )
        break
detectada = cv2.drawContours(original,
[NumPlacaCnt],-1, (0,255,0), 3)
plt.imshow(detectada,cmap=plt.cm.gray), plt.
xticks([]), plt.yticks([])
plt.show()
```


Na Figura abaixo, podemos observar somente a placa com o contorno delimitado

Figura 121 - Delimitação da placa



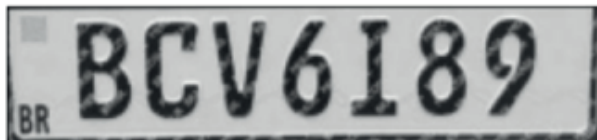
Fonte: Adaptado de autor desconhecido (2026)

Agora, de fato, vamos isolar somente a informação que precisamos, e salvar em seguida.

```
cortada = 'Placa' + str(idx) + '.png'  
plt.imshow(cv2.imread(cortada), cmap=plt.  
cm.gray), plt.xticks([]), plt.yticks([])  
plt.show()
```

e, finalmente, na Figura 122, temos somente a placa, segmentada da imagem original.

Figura 122 - Placa segmentada da imagem original



Fonte: O autor (2026)

Agora, nos resta somente ler os caracteres existentes na placa, por meio da biblioteca pytesseract.

Observe que a biblioteca também deve ser executada, na pasta onde foi instalada, conforme a Figura abaixo. O resultado final da tarefa é descrito logo em seguida, com êxito.

```
text = pytesseract.image_to_string(cortada)
print("Número e:", text)
Número e: BCV6189
```

É válido lembrar que este exemplo é bem simples, haja vista que a placa está bem posicionada, não foi verificada a presença de ruído, etc.

Existem várias técnicas mais robustas, para detecção e reconhecimento de placas, que abordam redes convolucionais e até Deep Learning, assuntos que serão abordados no volume 2 deste trabalho.

12.2. Identificação de moedas

A tarefa de segmentar objetos pode ser realizada de forma simples, sem a adoção de técnicas avançadas. Isso pode ser exemplificado, na tarefa de segmentação de moedas em um imagem, ou em até em um vídeo, utilizando códigos complementares.

A imagem teste do nosso exemplo é apresentada a seguir. Podemos observar 5 moedas, com um contraste relativamente adequado, e a nossa tarefa será de identificar as moedas, e identificá-las, com um círculo ao seu redor. Serão utilizadas um conjunto de técnicas simples vistas até aqui, de forma a deixar o código mais simples e leve.

Figura 123 - Imagem contendo cinco moedas, de tamanhos distintos



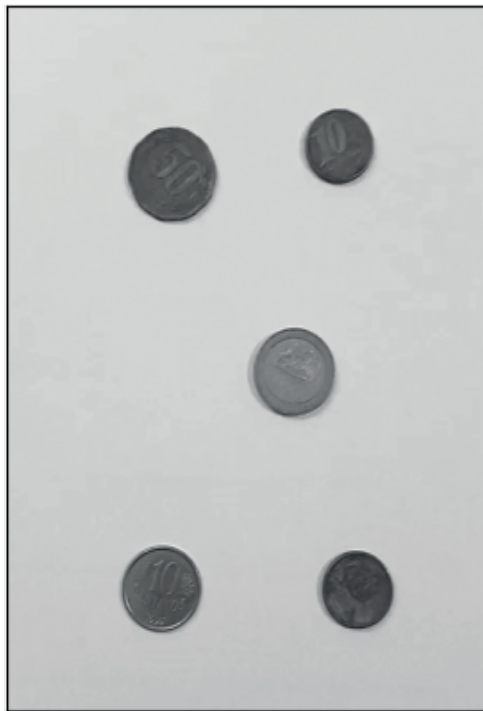
Fonte: O autor (2026)

O primeiro passo consiste em carregar as bibliotecas, a imagem, e logo em seguida, convertê-la para escala de cinza, conforme trecho do código a seguir.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
image = cv2.imread('moedas.png')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
plt.imshow(gray, cmap='gray'), plt.xticks([]),
plt.yticks([])
plt.show()
```

A imagem em escala de cinza é mostrada na Figura 124

Figura 124 - Imagem em escala de cinza



Fonte: O autor (2026)

Após a transformação em escala de cinza, devemos aplicar um filtro passa baixa gaussiano, que realize um efeito discreto, para eliminar algum tipo de ruído que pode ter se originado na aquisição da imagem.

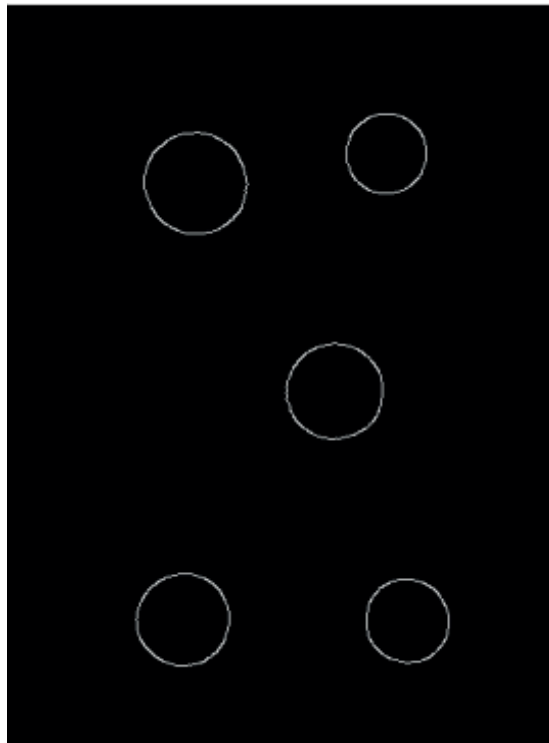
O efeito do filtro gaussiano é pouco perceptível na imagem. Sendo assim, vamos avançar para a etapa de detecção de bordas, que será executada por meio de filtros de canny.

A seguir, trechos do código da filtragem gaussiana, e da detecção de bordas de canny.

```
blur = cv2.GaussianBlur(gray, (15,15), 0)
canny = cv2.Canny(blur, 30, 120)
plt.imshow(canny, cmap='gray'),plt.
xticks([]), plt.yticks([])
plt.show()
```

Como resultado, podemos ver claramente na Figura 125, as bordas das moedas evidenciadas na imagem.

Figura 125 - Detecção de bordas das moedas



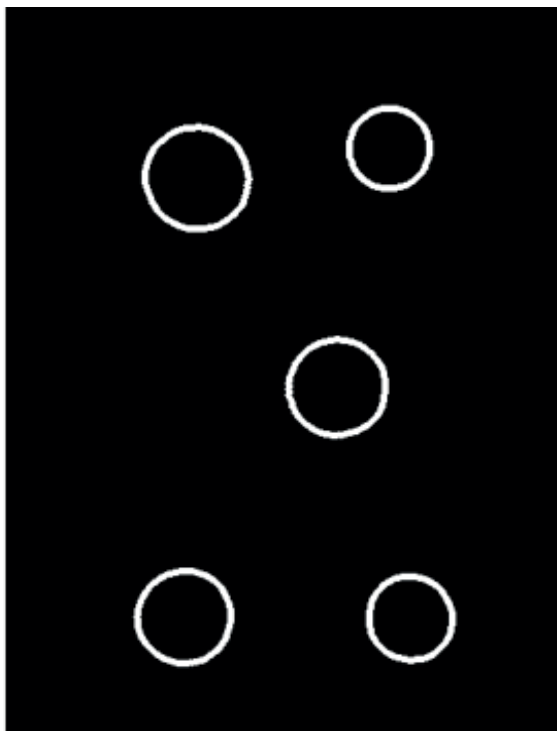
Fonte: O autor (2026)

As bordas, conforme ilustradas na Figura, são nítidas. Entretanto, devemos garantir que o algoritmo de detecção de contornos, encontre um contorno fechado, sem falhas. Para garantir que o contorno seja totalmente detectado, faremos uma dilatação nas bordas das moedas.

```
element_estr = cv2.getStructuringElement(cv2.  
MORPH_ELLIPSE, (3,3))  
dilated = cv2.dilate(canny, element_estr,  
iterations = 2)  
plt.imshow(dilated, cmap='gray'), plt.  
xticks([], plt.yticks([]))  
plt.show()
```

O efeito da dilatação é visualizado abaixo:

Figura 126 - Dilatação das bordas das moedas



Fonte: O autor (2026)

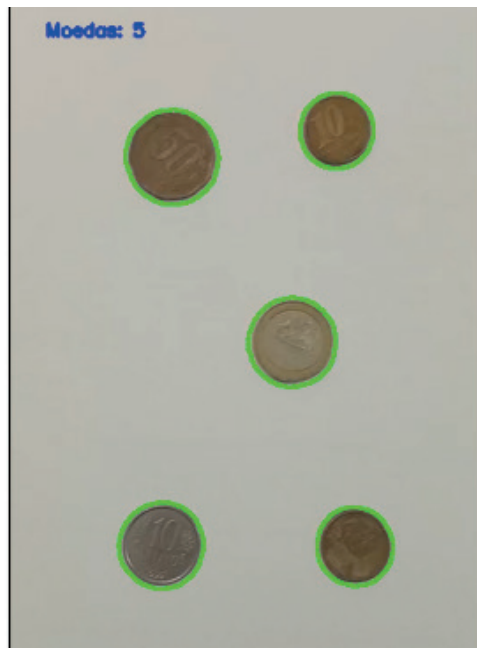
Os próximos passos se resumem a: Aplicação do algoritmo *findcountours*, para encontrar os contornos das bordas, e o desenho das bordas na imagem original por meio do algoritmo *drawcountours*. Aproveitando que o algoritmo ‘conta’ quantos contornos encontra, então vamos mostrar a quantidade de moedas na imagem, conforme código a seguir.

```
# Encontrar contornos nas bordas das moedas
contours, _ = cv2.findContours(canny.copy(),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
# Criando uma cópia da imagem para desenho
dos contornos
```

```
contour_image = image.copy()
#Desenho dos contornos
(cnt, hierarchy) = cv2.findContours(dilated.
copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_
NONE)
rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
cv2.drawContours(rgb, cnt, -1, (0,255,0), 2)
cv2.putText(rgb,f'Moedas: {str(len(cnt))}',
(30, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,
0, 255), 2)
plt.imshow(rgb),plt.xticks([], plt.
yticks([])
plt.show()
```

O resultado final do algoritmo pode ser visualizado na Figura 127.

Figura 127 - Imagem original, com as moedas detectadas



Fonte: O autor (2026)

12.3. Melhoramento de imagens médicas

A visão computacional aplicada a imagens médicas têm tido impacto direto e mensurável na melhora da aquisição, visualização, interpretação e, conseqüentemente, no diagnóstico e tratamento de pacientes.

Neste trabalho, apresentaremos duas aplicações simples, que fazem uma diferença fundamental na análise de mamografia digital, relacionadas ao pré-processamento, ou seja, em como a mamografia digital pode ser melhor visualizada antes da interpretação de um radiologista, ou até de um software de interpretação de diagnóstico.

A etapa de pré-processamento é fundamental para a remoção de estruturas indesejadas que possam dificultar o processo de análise e discriminação de uma imagem.

A Figura a seguir ilustra uma mamografia, com estruturas que possam dificultar os algoritmos de segmentação, melhoramento e classificação, por exemplo.

Figura 128 - Mamografia com falhas na aquisição, e placa de identificação do paciente



Fonte: Adaptado de Suckling et al, 2015

A falha na aquisição indicada na Figura, juntamente com a placa de identificação do paciente, deve ser removida, antes de processos automáticos de classificação, se for o caso.

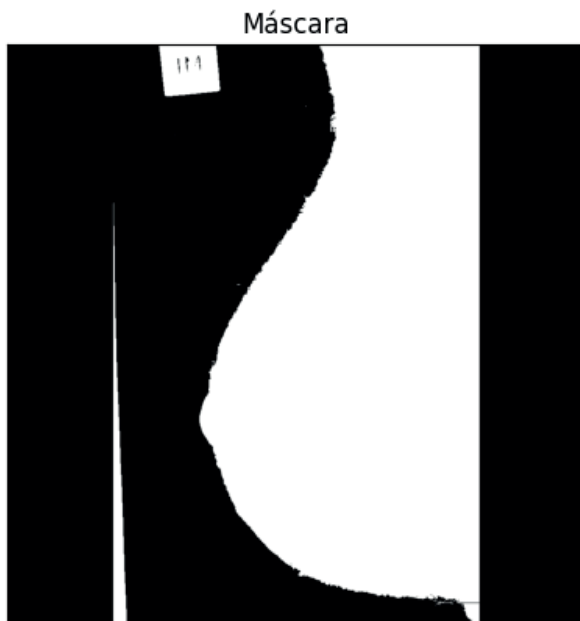
Para remover tais falhas, podemos recorrer às técnicas de thresholding e de morfologia matemática.

O primeiro passo, além da importação das bibliotecas, é fazer uma espécie de máscara, por meio de um threshold binário. Neste caso, escolhemos $T = 30$, que significa que todos os pixels menores ou iguais a 30, serão atribuídos valor 0 - correspondente ao preto - e caso contrário, valor 255, que corresponde ao branco, conforme código a seguir:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
#Gerando a máscara com Treshold=30
img=cv2.imread("mam_ex1.jpg",0)
method = cv2.THRESH_BINARY
ret,imgBin1 = cv2.threshold(img, 30,255,
method)
plt.imshow(imgBin1,cmap=plt.cm.gray)
plt.title('Máscara'),plt.xticks([]), plt.
yticks([])
plt.show()
```

A Figura 129 ilustra a máscara resultante dos passos acima. A partir dela, iremos utilizar métodos de morfologia matemática, para preencher os espaços referentes à falha e a placa de identificação do paciente.

Figura 129 - Máscara obtida por meio de threshold binário



Fonte: O autor (2026)

O próximo passo é preencher os espaços em branco da máscara gerada acima. Utilizaremos o operador morfológico de abertura.

A Abertura de uma imagem por um elemento estruturante é definida por uma erosão, seguida de uma dilatação, e tem como principal objetivo suavizar o contorno de uma imagem, quebrar istmos e elimina protuberâncias finas, além de eliminar as partículas indesejáveis sem modificar o tamanho da imagem original.

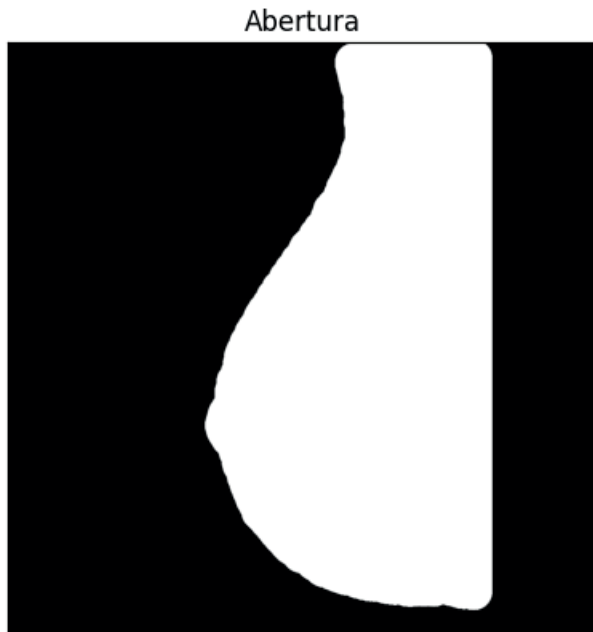
O elemento estruturante adotado para o exemplo, é de morfologia elíptica, e tem kernel na dimensão de 65x65.

```
kernel=cv2.getStructuringElement(cv2.MORPH_
ELLIPSE, (65,65))
# Elemento estruturante em elipse
opening = cv2.morphologyEx(imgBin1, cv2.
MORPH_OPEN, kernel)
```

```
plt.imshow(opening, cmap=plt.cm.gray)
plt.title('Abertura'), plt.xticks([], plt.
yticks([])
plt.show()
```

O resultado obtido é mostrado na Figura 130. Pode-se observar que tanto a falha, como o local da etiqueta, que na Figura 129 estavam na cor branca, agora aparecem na cor preta, pois foram preenchidos por meio do operador de abertura.

Figura 130 - Correção de falhas da máscara por meio do operador de abertura



Fonte: O autor (2026)

O próximo passo é a multiplicação da máscara gerada com a imagem. note que não é uma multiplicação de matrizes, e sim de arrays. Assim, devemos normalizar a intensidade dos pixels da máscara, dividindo toda a imagem por 255.

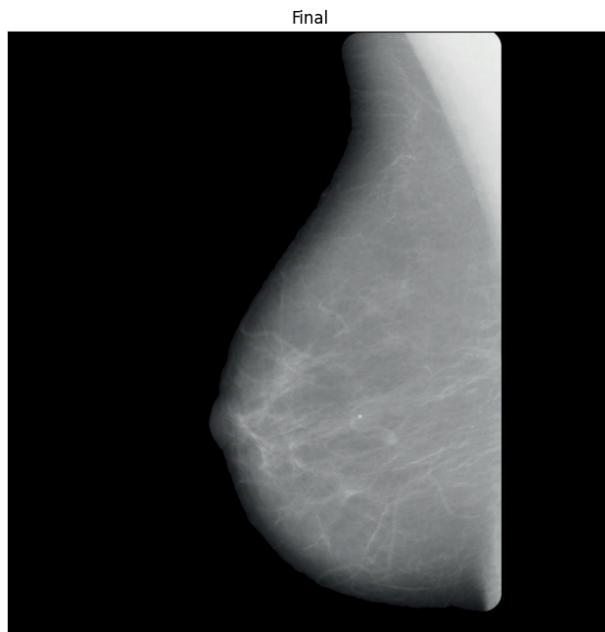
Os pixels escuros da máscara devem possuir valor zero, e os pixels brancos, valor hum.

O código abaixo ilustra a resolução deste passo:

```
opening_norm=opening//255
final=img*opening_norm
plt.figure(figsize=(12,8))
plt.imshow(final,cmap=plt.cm.gray)
plt.title('Final'),plt.xticks([]), plt.
yticks([])
plt.show()
```

A Figura 131 mostra o resultado obtido após a multiplicação da imagem original, pela máscara construída nos passos acima:

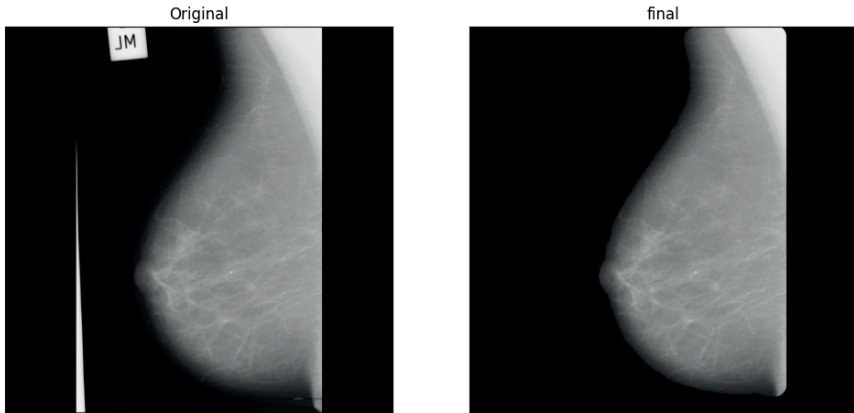
Figura 131 - Correção de falhas da máscara por meio do operador de abertura



Fonte: Adaptado de Suckling et al, 2015

O resultado da multiplicação da máscara pela imagem original nos entrega o resultado final, onde as falhas e a placa de identificação da imagem original são totalmente preenchidas pela cor preta. A Figura 130 mostra um comparativo entre a imagem original, e a imagem resultante do algoritmo proposto, evidenciado sucesso na consecução do objetivo.

Figura 132 - Comparativo entre a mamografia original, e o resultado final do algoritmo proposto



Fonte: Adaptado de Suckling et al, 2015

12.4. Reconhecedor Óptico de Caracteres (OCR)

A extração de texto de imagens e documentos pode ser muito demorada, se realizada de forma manual. Uma ferramenta chamada de OCR (*Optical Character Recognition* - reconhecimento óptico de caracteres) pode automatizar esse processo, permitindo a edição de imagens em arquivos de texto editáveis.

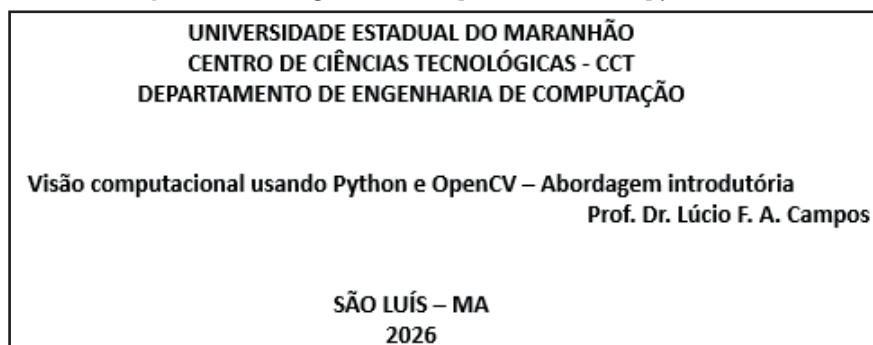
O reconhecimento óptico de caracteres é amplamente utilizado em:

- Transformação de documentos físicos em digitais;
- Automatização de entrada de dados;
- Carros autônomos.

O `pytesseract` é um *wrapper* Python para o motor Tesseract-OCR do Google, permitindo converter textos em imagens para strings editáveis. Ele exige a instalação prévia do executável Tesseract no sistema operacional e é amplamente utilizado para leitura de documentos e extração de dados.

A seguir, apresentaremos um código em python, que transforma o texto da Figura 133 abaixo em strings, na língua portuguesa.

Figura 133 - Imagem de teste, para execução do `pytesseract`



Fonte: O autor (2026)

A seguir, o código a ser executado

```
pip install pytesseract
pip install tesseract-ocr-por
!apt-get update
!apt-get install tesseract-ocr-por
import cv2
import pytesseract as pt
img=cv2.imread('teste_tesseract.png')
print(pt.pytesseract.image_to_string(img,
lang='por'))
```

Como resultado do comando `print`, a Figura 134 ilustra a saída do comando, com todas as letras da imagem 133 em forma de *string*.

Figura 134 - Resultado obtido por meio do algoritmo proposto

```
print(pt.pytesseract.image_to_string(img, lang='por'))
```

UNIVERSIDADE ESTADUAL DO MARANHÃO
CENTRO DE CIÊNCIAS TECNOLÓGICAS - CCT
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO

Visão computacional usando Python e OpenCV – Abordagem introdutória
Prof. Dr. Lúcio F. A. Campos

SÃO LUÍS - MA
2026

Fonte: O autor (2026)

12.5. Detecção e Reconhecimento de Faces

Algoritmos de detecção - e posterior reconhecimento – de faces, têm sido desenvolvidos ao longo das últimas décadas, mas nenhum consegue ser capaz de comparar o poder do cérebro humano em realizar tal tarefa.

O ser humano consegue detectar e reconhecer faces em qualquer situação, seja com baixa luminosidade, ruído, vários ângulos diferentes, etc., e somente agora, nas últimas décadas, devido a robustez das máquinas, e o avanço dos algoritmos, podemos dizer que existe uma nítida evolução nos algoritmos e técnicas de visão computacional, favorecendo a tarefa de detectar e reconhecer faces.

Neste curso, falaremos basicamente de duas técnicas de Detecção Facial. Haar – Cascade e Histograma de Gradientes Orientados – HOG.

Aplicações mais avançadas de Detecção e Reconhecimento Facial utilizam algoritmos de aprendizado de máquina, e redes neurais convolucionais, assuntos que abordaremos no volume II deste trabalho.

12.5.1. Algoritmo Haar – Cascade

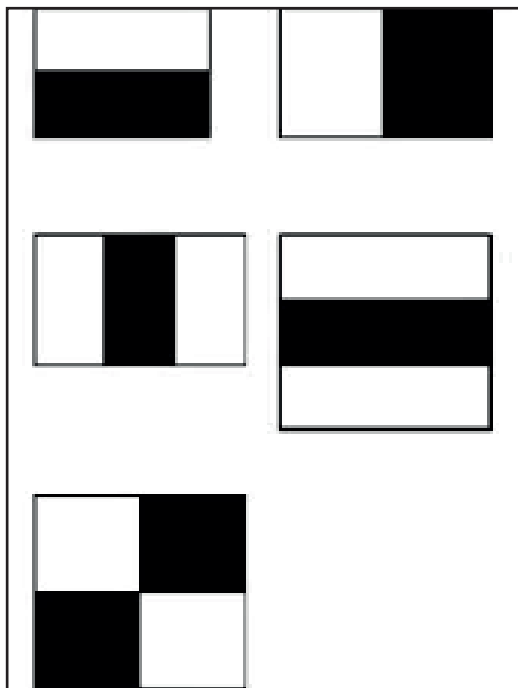
O algoritmo de Haar Cascade é baseado no algoritmo proposto por Paul Viola e Michael Jones, 2001, revisado em 2003, que obteve bastante sucesso, devido a sua simplicidade e rápida execução.

O detector de faces de Viola-Jones possui quatro fases:

- Seleção de características Haar-like retangulares;
- Criação da imagem integral;
- treino do classificador AdaBoost e, por fim;
- O uso de classificadores em cascata, que descarta as regiões de fundo para focar em áreas mais prováveis de conter uma face.

As características Haar-like descrevem características construídas a partir de um conjunto de ondaletas de Haar (Haar wavelets). A Figura a seguir ilustra as características mais utilizadas no algoritmo de viola Jones.

Figura 135 - Wavelets de Haar mais utilizadas



Fonte: Jagdale et al (2026)

As características de Haar consistem em um ou mais retângulos adjacentes e são utilizadas para analisar locais específicos em uma janela de detecção. Cada característica resulta em um único valor, calculado através

da soma das intensidades dos pixels sob os retângulos brancos subtraída da soma dos valores sob os retângulos pretos, conforme equação abaixo:

$$\Delta = \frac{1}{n} \sum_{\text{preto}}^n I(x) - \frac{1}{n} \sum_{\text{branco}}^n I(x)$$

Quanto mais próximo de 1, mais semelhante é a imagem real da característica aplicada. O algoritmo de Haar-Cascade é treinado com imagens positivas e negativas.

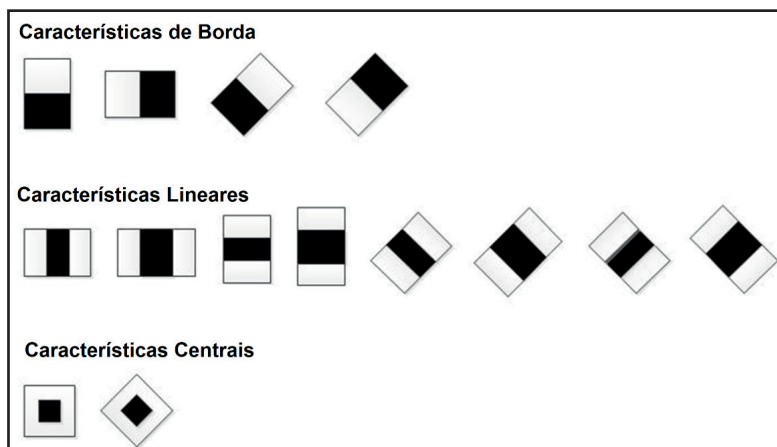
Imagens positivas contêm o objeto de interesse, ou seja, rostos humanos, no caso da detecção de faces. Já as imagens negativas, não contêm o objeto de interesse, ou seja, não possuem faces, e são formadas de objetos, paisagens, animais, texturas, ambientes internos, etc.

O classificador de Haar utiliza modelos pré-treinados, ou seja, o objeto que queremos detectar na imagem (face frontal, face lateral, olhos, óculos), possui um conjunto de parâmetros, salvos em um arquivo .xml. Os parâmetros salvos compreendem:

- As *features* Haar seleccionadas
- Os limiares (thresholds) de decisão
- Os pesos definidos pelo AdaBoost
- A estrutura da cascata de classificadores

Para detectar faces, são utilizadas todas as características da Figura 136 abaixo, e para propor novas características é realizada uma rotação de 45 graus nas características já existentes.

Figura 136 - Wavelets de Haar utilizadas para detecção de faces



Fonte: Adaptado de Yang (2019)

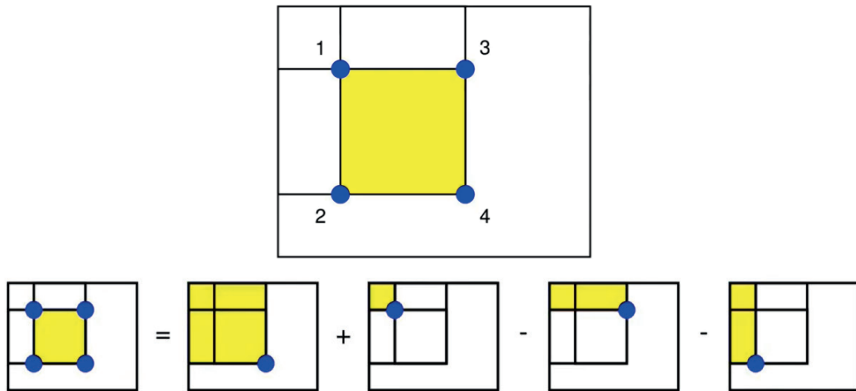
O processo de detecção de características pode causar um custo computacional gigante, pois em uma imagem de apenas 24x24 pixels, se formos aplicar todas características da Figura 135, e mais as rotações de 45 graus, teremos aproximadamente 160 mil testes e combinações. Para evitar o problema do custo computacional, é utilizado o conceito de imagem integral.

A imagem integral é capaz de calcular rapidamente a soma dos valores em um subconjunto retangular de uma matriz.

A imagem integral é uma tabela bidimensional do tamanho da imagem original, onde cada elemento equivale à soma de todos os níveis de cinza (intensidades) dos pixels à esquerda e acima do pixel atual, inclusive.

Utilizando a imagem integral, a soma dos níveis de cinza de qualquer área retangular pode ser calculada em quatro referências à memória. A soma dos valores na região 1234 podem ser calculadas conforme a Figura abaixo:

Figura 137 - Cálculo da imagem integral

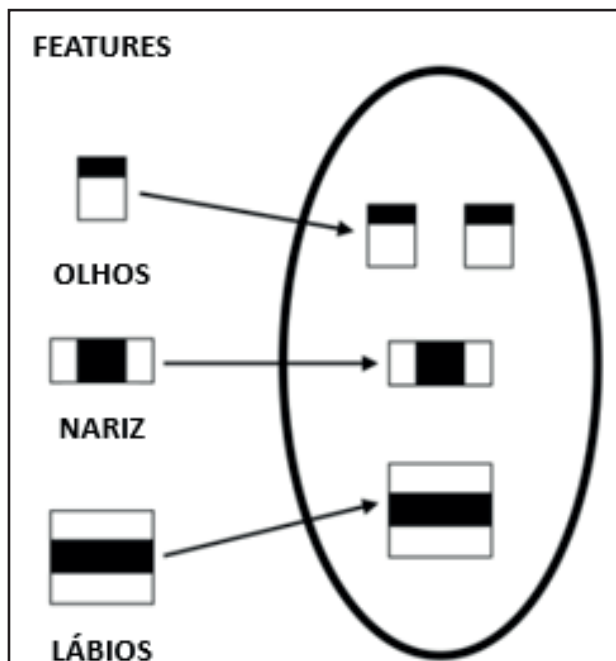


Fonte: O autor (2026)

A cascata de classificadores é uma estratégia usada no método Haar para tornar a detecção de faces rápida e eficiente, já que aplicar um teste complexo em toda a imagem demandaria um custo computacional elevado, então é realizado vários testes simples em sequência, utilizando janelas deslizantes (similar ao processo de filtragem em tempo discreto). Assim, a cada iteração da janela, é descartado rapidamente o que claramente não é uma face.

A Figura 138 a seguir ilustra um modelo teórico, de como um conjunto de *features* pode sugerir a detecção de uma face.

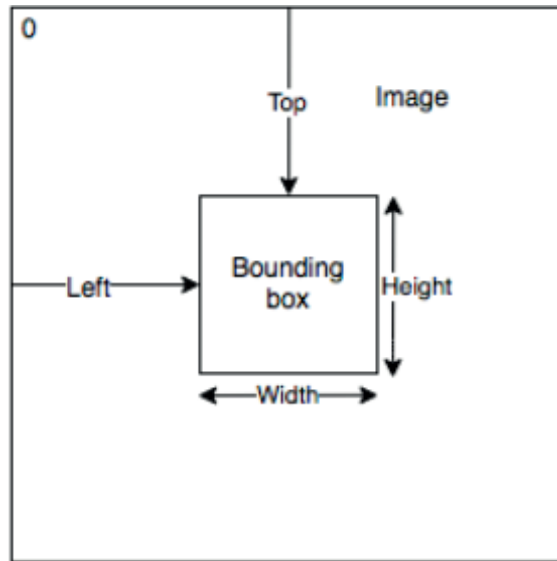
Figura 138 - Modelo de face, utilizando *Haar features*



Fonte: Adaptado de Mahboob, Mariyam & Mohsin, Nounman. (2024).

Assim que uma face é encontrada, o algoritmo deve fazer uma caixa delimitadora, chamada de “bounding box”, que é um retângulo traçado ao redor de um rosto detectado em uma imagem ou vídeo, do qual define as coordenadas espaciais, e o tamanho do rosto na imagem. A Figura 139 ilustra o desenho de um *bounding-box*.

Figura 139 - Coordenadas do bounding-box



Fonte: O autor (2026)

Uma vez que um rosto é detectado, o sistema retorna as coordenadas da caixa delimitadora. Conforme mostrado na Figura acima, a região da face é definida, usando:

- Top (x) e Left (y): Coordenadas de onde a caixa começa.
- Width (w) e Height (h): As dimensões da caixa.

Assim que todas as faces de uma imagem são detectadas, o algoritmo retorna a imagem original, com as demarcações das bounding-box.

A partir da Figura 140, faremos um exemplo de um detector de faces, por meio do algoritmo Haar - Cascade.

Figura 140 - Fotografia com 18 faces frontais



Fonte: Berdelli (2024)

O primeiro passo, se resume em importar as bibliotecas, carregar a imagem, e converter para escala de cinza, conforme trecho do código abaixo:

```
import cv2
from matplotlib import pyplot as plt
imagem = cv2.imread('varios.jpg') # carregando
a imagem teste
imagem = cv2.cvtColor(imagem, cv2.COLOR_
RGB2BGR)
```

Logo em seguida, devemos carregar o classificador treinado, objetivo da nossa detecção. No caso do exemplo em questão, nosso objetivo é detectar faces, que estão na visão frontal da imagem. Sendo assim, utilizaremos o classificador `haarcascade_frontalface_default`, apontando o seu caminho.

Além do carregamento do classificador correto, ainda devemos apontar dois parâmetros: A escala da imagem, e o tamanho das janelas.

A imagem pode ser ampliada ou reduzida, para aumentar a qualidade da detecção. Assim, devemos indicar a escala, na variável *scaleFactor*.

Também é necessário definir o tamanho das janelas em que a imagem será dividida, pois o Haar-Cascade faz a divisão da imagem em blocos menores, e executa a classificação bloco a bloco. O tamanho da janela é definido na variável *minSize*.

O trecho do código que executa os comandos citados acima, é apresentado a seguir:

```
classificador_face = cv2.CascadeClassifier(r"/  
content/haarcascade_frontalface_default.  
xml")  
gray = cv2.cvtColor(imagem, cv2.COLOR_  
BGR2GRAY)  
face_detector =  
classificador_face.detectMultiScale(gray,  
scaleFactor=1.8, minSize=(4, 4))
```

Por fim, devemos apontar quantas faces foram detectadas, e desenhar as *bounding-box* de cada face, conforme visto abaixo:

```
for (x, y, l, a) in face_detector:  
cv2.rectangle(imagem, (x, y), (x + l, y +  
a), (0, 255, 0), 4)  
cv2.putText(imagem, f'Faces Detectadas:  
{str(len(face_detector))}', (650, 50), cv2.  
FONT_HERSHEY_SIMPLEX, 1.2, (255, 0, 0), 4)  
plt.figure(figsize=(12, 8))  
plt.imshow(imagem), plt.xticks([]), plt.  
yticks([])  
plt.show()
```

A Figura 141 abaixo ilustra o resultado obtido do algoritmo acima.

Figura 141 (a) - Faces detectadas por meio do algoritmo Haar Cascade



Fonte: Adaptado de Berdelli (2024)

Pode-se observar uma quantidade significativa de falsos positivos, e também de faces não detectadas. Tal problema pode ser resolvido com um melhor ajuste das variáveis *scaleFactor* e *minSize*.

Contudo, a técnica de detecção por meio do classificador Haar-Cascade não produz bons resultados. A seguir, conheceremos uma técnica mais eficaz na detecção de faces.

12.5.2. Histograma de Gradientes Orientados (HOG)

O Histograma de Gradientes Orientados foi desenvolvido por Navneet Dalal e Bill Trigs, no qual é descrito como um método de extração de características aplicado a cada pixel de uma imagem, para extrair informações da vizinhança, tais como textura, e apresentar essas informações da imagem em por meio de vetores de características. Esses vetores descrevem características da imagem que são úteis para capturar

bordas e estruturas de gradiente.

Os passos para a detecção de objetos usando HOG são os seguintes:

1. Cálculo do gradiente
2. Classificação de orientação
3. Normalização e formação de Vetores de Características.
4. Delimitação das Bounding Box

A seguir, falaremos brevemente sobre cada passo:

1. Cálculo do Gradiente

O gradiente de uma imagem representa a mudança direcional na intensidade ou cor, fundamental para detectar bordas e variações em visão computacional. Para realizar o cálculo do gradiente, primeiro devemos extrair e capturar informações da borda da imagem. Para tanto, iremos aplicar o operador de Sobel para detecção de bordas, com kernel de 3x3, visto em 5.5.1.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} (\text{Magnitude do Gradiente})$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} (\text{Orientação do Gradiente})$$

Após aplicar o Kernel de Sobel à imagem, podemos calcular a magnitude e a orientação, por meio das fórmulas descritas abaixo:

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \tan^{-1}\left(\frac{G_x}{G_y}\right)$$

Os gradientes determinam a intensidade (magnitude) e a direção (orientação) das arestas em um ponto específico. A direção da aresta é perpendicular à direção do vetor gradiente no local onde o gradiente é calculado, ou seja, o comprimento e a direção do vetor.

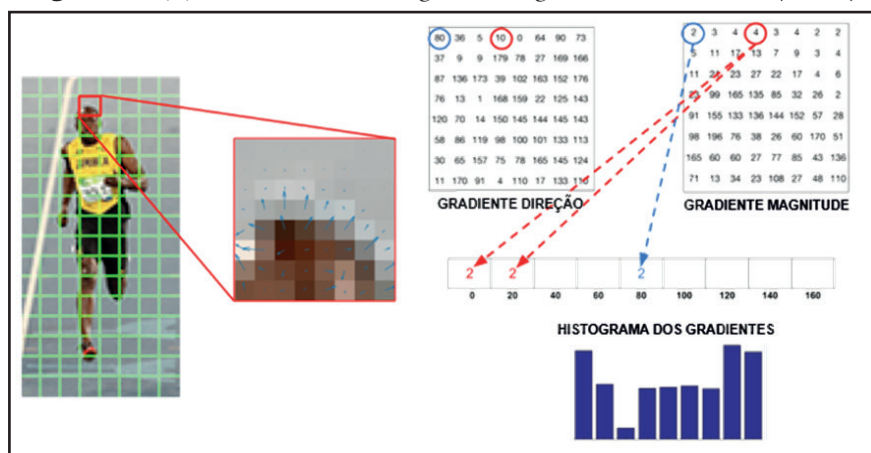
2. Cálculo dos Histogramas do Gradiente

Para cada pixel da imagem, calculamos dois valores: magnitude e orientação. Para combinar essas informações em algo significativo, usamos um histograma, organizando e interpretando esses valores de forma eficaz.

Para melhor interpretação deste passo, vamos utilizar a imagem descrita na Figura 141 (a), e dividi-la em blocos de 8x8 pixels, mostrado em 141 (b). A partir dos valores calculados de direção e magnitude para cada pixel, como ilustrado em 141 (c) e 141 (d), podemos construir o histograma de gradientes orientados na imagem, em 134 (e).

Observe que as setas desenhadas em 141 (b) representam a direção do gradiente, que tende a apontar para a região das bordas, ou seja, onde se observa uma mudança abrupta de intensidade dos pixels.

Figura 141 (b) - Cálculo dos Histogramas de gradientes orientados (HOG)



Fonte: Adaptado de Malick (2016)

3. Normalização e formação dos Vetores de Características.

Como a magnitude dos gradientes dependem das condições de iluminação, a normalização ajusta os histogramas para reduzir a influência das variações de iluminação e contraste.

Para calcular o vetor de características final para todo o patch da imagem, os vetores 36×1 (de cada bloco 8×8) são concatenados em um único vetor.

O descritor HOG de um fragmento de imagem geralmente é visualizado plotando os histogramas normalizados de cada bloco 8×8 . A Figura 135 ilustra os vetores de cada bloco, mostrando que a direção dominante do histograma captura o formato da pessoa, especialmente ao redor do tronco e das pernas.

Figura 142 - Vetores de Direção do Gradiente



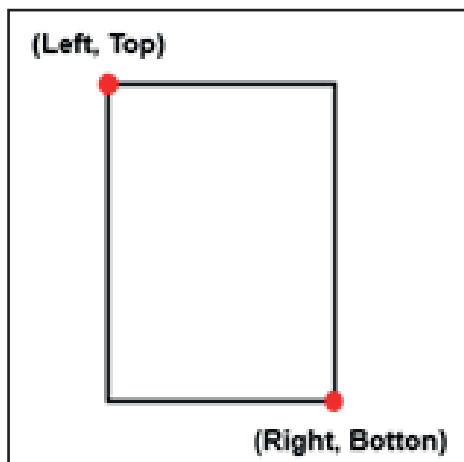
Fonte: Adaptado de Malick (2016)

4. Delimitação das Bounding Box

O descritor HOG é utilizado para detectar objetos, juntamente com a biblioteca Dlib, que é um toolkit e código aberto em C++ , bastante utilizado em visão computacional e análise de dados, fornecendo ferramentas robustas para detecção de faces e reconhecimento facial.

O Dlib desenha a bounding box utilizando dois pontos, que, juntos, possuem 4 parâmetros: Left, Top, Righth, Botton, conforme ilustra a Figura 143.

Figura 143 - Desenho da Bouding box



Fonte: O autor (2026)

A partir da Figura 140 - a mesma utilizada no exemplo do detector Haar-Cascade - executaremos o código de um detector de faces, utilizando HOG + a biblioteca Dlib.

O primeiro passo - assim como no algoritmo de Haar-Cascade - resume em importar as bibliotecas, carregar a imagem, e converter para escala de cinza. Vamos adicionar mais duas bibliotecas que são necessárias para o funcionamento do HOG: Dlib e Imutils.

```
import cv2
import imutils
import dlib
from imutils import face_utils
from matplotlib import pyplot as plt
imagem = cv2.imread('varios_.jpeg')
imagem = cv2.cvtColor(imagem, cv2.COLOR_
RGB2BGR)
```

Logo em seguida, vamos carregar o detector (já incluído na biblioteca Dlib), e aplicar um fator de escala unitário, conforme trecho do código abaixo:

```
detector = dlib.get_frontal_face_detector()
faces_detectadas = detector(imagem,1)
```

Por fim, de igual modo ao algoritmo Haar-Cascade, devemos apontar quantas faces foram detectadas, e desenhar as *bounding-box* de cada face, conforme visto abaixo:

```
for face in faces_detectadas:

    e, t, d, b = (int(face.left()), int(face.top()),
int(face.right()), int(face.bottom()))
    cv2.rectangle(imagem, (e, t), (d, b), (0, 255, 0), 2)

cv2.putText(imagem,f'Faces Detectadas:
{str(len(faces_detectadas))}', (650, 50), cv2.FONT_
HERSHEY_SIMPLEX, 1.2, (255, 0, 0), 4)
plt.figure(figsize=(12,8))
plt.imshow(imagem),plt.xticks([]), plt.yticks([])
plt.show()
```

A Figura 144 abaixo ilustra o resultado encontrado. Podemos claramente observar que o classificador HOG foi mais eficaz que o classificador Haar-Cascade, pois detectou 18 das 17 faces da imagem, e não apresentou nenhum falso positivo.

Figura 144 - Faces detectadas por meio do algoritmo Haar Cascade



Fonte: Adaptado de Berdelli (2024)

13. CONSIDERAÇÕES FINAIS

Ao longo desta obra, percorremos os principais conceitos introdutórios da Visão Computacional, partindo dos fundamentos da formação das imagens digitais e do funcionamento do sistema visual humano até a aplicação prática de técnicas clássicas de processamento e análise de imagens utilizando Python e OpenCV.

O estudo apresentado demonstrou que a Visão Computacional é uma área multidisciplinar, construída a partir da integração de conhecimentos provenientes da matemática, estatística, processamento digital de sinais, inteligência artificial e ciência da computação. Essa característica faz com que seu aprendizado seja simultaneamente desafiador e extremamente enriquecedor.

Os capítulos iniciais permitiram compreender como as imagens são adquiridas, representadas digitalmente e armazenadas em sistemas computacionais. Em seguida, foram exploradas técnicas fundamentais de transformação de intensidade, filtragem espacial, melhoramento de contraste e remoção de ruídos, etapas indispensáveis para a preparação adequada dos dados visuais. Observou-se que a qualidade do pré-processamento exerce influência direta no desempenho das etapas posteriores de segmentação, extração de características e reconhecimento de padrões. Outro aspecto importante evidenciado ao longo do texto é que não existe uma única solução universal para todos os problemas de Visão Computacional. Cada aplicação possui características próprias, exigindo a escolha criteriosa de algoritmos, parâmetros e estratégias de processamento. Em muitos casos, métodos simples podem produzir resultados extremamente satisfatórios quando corretamente aplicados, enquanto técnicas mais sofisticadas podem demandar elevado custo computacional sem necessariamente proporcionar ganhos proporcionais.

A utilização da linguagem Python e da biblioteca OpenCV mostrou-se uma combinação poderosa para fins didáticos e profissionais. Além de permitir a implementação relativamente simples de algoritmos complexos, essas ferramentas possibilitam que estudantes e pesquisadores concentrem seus esforços na compreensão dos fenômenos e na resolução dos problemas, sem a necessidade de lidar inicialmente com detalhes excessivamente complexos de programação.

Entretanto, este livro não pretende esgotar o vasto universo

da Visão Computacional. Pelo contrário, ele representa um ponto de partida. Diversos tópicos contemporâneos foram apenas mencionados ou introduzidos de forma preliminar, como aprendizado profundo (Deep Learning), redes neurais convolucionais (CNNs), modelos multimodais, detecção e rastreamento de objetos em tempo real, reconstrução tridimensional, visão robótica e sistemas inteligentes autônomos. Cada um desses temas constitui, por si só, um amplo campo de pesquisa e desenvolvimento.

O cenário atual aponta para uma convergência cada vez maior entre Visão Computacional e Inteligência Artificial. Sistemas capazes de interpretar imagens, compreender contextos e auxiliar processos decisórios já fazem parte do cotidiano de milhões de pessoas. Nos próximos anos, espera-se que tais tecnologias se tornem ainda mais presentes em áreas como saúde, educação, indústria, segurança pública, agricultura, mobilidade urbana e exploração científica.

Dessa forma, o leitor que conclui este material possui agora uma base sólida para avançar em estudos mais aprofundados, desenvolver projetos próprios e acompanhar a rápida evolução da área. O conhecimento adquirido aqui deve ser encarado não como um ponto de chegada, mas como o início de uma trajetória de aprendizado contínuo, experimentação e inovação.

Por fim, fica o incentivo para que cada leitor explore novos problemas, desenvolva soluções criativas e contribua para a construção do conhecimento científico e tecnológico. A Visão Computacional continua expandindo suas fronteiras, e as próximas descobertas poderão surgir justamente das mãos daqueles que hoje estão dando seus primeiros passos nesse fascinante campo da ciência.

Que este livro sirva como inspiração para novas pesquisas, aplicações e desafios, fortalecendo a formação acadêmica e profissional de todos aqueles que se interessam pela arte de fazer máquinas enxergarem, interpretar e compreenderem o mundo visual.

REFERÊNCIAS

ABDELMAJID EL HAKOUME; AFRAITES, Lckbir; LAGHRIB, Amine. An improved coupled PDE system applied to the inverse image denoising problem. *Electronic Research Archive*, 2022.

ACONCI, A. OTSU e outras técnicas de limiarização. Disponível em: <http://www2.ic.uff.br/~aconci/OTSUeOutras.pdf>. Acesso em: 10 mar. 2026.

AKYOL, Gökçenaz. How to restore images? Image Processing #7. *Medium*, 17 jan. 2023. Disponível em: <https://medium.com/@gokcenazakyol/how-to-restore-images-image-processing-7-cabf1186fbc5>. Acesso em: 16 dez. 2025.

ALENCAR, Maycol. *Detalhando amostragem e quantização*. Disponível em: <https://maalencar.wordpress.com/2012/03/15/detailing-sampling-and-quantization/>. Acesso em: 9 out. 2025.

BARBOSA, André; BONADIO, André. *Princípios de processamento de imagens: uma introdução à convolução*. Disponível em: <https://blog.elo7.dev/convolucao/>. Acesso em: 5 dez. 2025.

BERDELLI, Matheus. *Fotografia ao ar livre* [fotografia]. *Pexels*, 2024. Disponível em: <https://www.pexels.com/pt-br/foto/31393068/>. Acesso em: 28 maio 2026.

BEZERRA, Cides. *Uma abordagem de segmentação semântica de íris para fins biométricos usando aprendizagem profunda*. 2018. Tese (Doutorado em Engenharia Elétrica) – Universidade Federal do Paraná, Curitiba, 2018.

BONITO E PANTANAL. *Imagem de tucano-toco* [fotografia]. Disponível em: <https://bonitoepantanal.com.br/wp-content/uploads/2025/09/Tucano-Toco-1024x683.webp>. Acesso em: 12 out. 2025.

BORGGREVE, Ellen. Low contrast photography and how to correct it in post-processing. *PetaPixel*, 3 abr. 2019. Disponível em: <https://petapixel.com/2019/04/03/low-contrast-photography-and-how-to-correct-it-in->

[post-processing/](#). Acesso em: 7 dez. 2025.

CANNY, John F. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 8, n. 6, p. 679–698, 1986.

CHAVES, Bruno Edson. *Espectro da luz visível*. Disponível em: https://www.researchgate.net/figure/Figura-2-Espectro-da-luz-visivel-A-partir-de-aproximadamente-400-nm-roxo-ate_fig47_353273443. Acesso em: 4 dez. 2026.

CONTRAST-ENHANCED ultrasound safe and effective detector of liver cancer. *Imaging Technology News*, 19 out. 2017. Disponível em: <https://www.itnonline.com/content/contrast-enhanced-ultrasound-safe-and-effective-detector-liver-cancer>. Acesso em: 10 dez. 2025.

DAVIES, Nicola. Up the side of an apartment building. In: *Ukraine: A Visit to Chernobyl*. GoNomad. Disponível em: <https://www.gonomad.com/5946-ukraine-a-visit-to-chernobyl>. Acesso em: 14 dez. 2025.

DE MOURA, Maria; DE SOUSA, Alcilene; DE OLIVEIRA, Ivo; MESQUITA, Laércio; DRUMOND, Patrícia. Segmentação automática de candidatos a nódulos pulmonares em imagens de tomografia computadorizada. In: *SIMPÓSIO BRASILEIRO DE COMPUTAÇÃO APLICADA À SAÚDE*, 15., 2015, Recife. *Anais [...]*. Porto Alegre: Sociedade Brasileira de Computação, 2015. p. 131–140.

DUDGEON, D. E.; MERSEREAU, R. M. *Multidimensional digital signal processing*. Englewood Cliffs: Prentice Hall, 1990.

FATHIMA. Raw vision: a scratch-based exploration of image processing techniques. *Medium*, 2021. Disponível em: <https://medium.com/@fathima.offical.msg/raw-vision-a-scratch-based-exploration-of-image-processing-techniques-v-b6f856c7304a>. Acesso em: 5 dez. 2025.

GIL, Marisa Adán. Teste da NASA: você sobreviveria em uma missão na Lua? *Época Negócios*, 12 ago. 2023. Disponível em: <https://epocanegocios.globo.com/ciencia-e-saude/noticia/2023/08/teste-da-nasa-voce->

[sobreviveria-em-uma-missao-na-lua.shtml](#). Acesso em: 5 dez. 2025.

GONZALEZ, Rafael C.; WOODS, Richard E. *Processamento digital de imagens*. São Paulo: Edgard Blücher, 2001.

HAMITI, Bess. *Toddler wearing gray button collared shirt with curly hair* [fotografia]. *Pexels*. Disponível em: <https://www.pexels.com/photo/toddle-wearing-gray-button-collared-shirt-with-curly-hair-35537/>. Acesso em: 10 jan. 2026.

HARALICK, Robert M.; SHANMUGAM, K.; DINSTEN, I. Textural features for image classification. *IEEE Transactions on Systems, Man and Cybernetics*, 1973.

HTOON, Kyaw Saw. A tutorial to histogram equalization. *Medium*, 13 ago. 2020. Disponível em: <https://medium.com/@kyawsawhtoon/a-tutorial-to-histogram-equalization-497600f270e2>. Acesso em: 7 dez. 2025.

JAGDALE, Aditi *et al.* Attendance system using deep learning. *International Journal of Advanced Research in Science, Communication and Technology*, p. 326–331, 2023.

KOLKUR, S. *et al.* Human skin detection using RGB, HSV and YCbCr color models. Disponível em: <https://www.semanticscholar.org/paper/Human-Skin-Detection-Using-RGB%2C-HSV-and-YCbCr-Color-Kolkur-Ka%20bande/03e0ea040482660aa7f4ef73b893d339b2700bbe>. Acesso em: 12 jan. 2026.

LEVI, Rafael. *Guia para construir um robô 2WD com Arduino*. Disponível em: <https://medium.com/@rafaellevisse/guia-para-construir-um-rob%C3%B4-2wd-com-arduino-d50469480a2f>. Acesso em: 22 out. 2025.

LIN, Fang-Cheng; ZAO, John Kar-Kin; TU, Kuan-Chung; WANG, Yijun. SNR analysis of high-frequency steady-state visual evoked potentials from the foveal and extrafoveal regions of human retina. In: *ANNUAL INTERNATIONAL CONFERENCE OF THE IEEE*

ENGINEERING IN MEDICINE AND BIOLOGY SOCIETY (EMBC), 2012, San Diego. *Anais [...]*. San Diego: IEEE, 2012. p. 1810–1814. DOI: <https://doi.org/10.1109/EMBC.2012.6346302>.

MAHBOOB, Mariyam; MOHSIN, Nounman. *Attendance management system using face recognition*. 2024.

MALLICK, Satya. Histogram of oriented gradients explained using OpenCV. *LearnOpenCV*, 6 dez. 2016. Disponível em: <https://learnopencv.com/histogram-of-oriented-gradients/>. Acesso em: 31 maio 2026.

MARAABA, Luqman. *Raw vision: a scratch-based exploration of image processing techniques*. 2020. Dissertação (Mestrado) – Eastern Mediterranean University, Famagusta, 2020.

MEIRELES JÚNIOR. *Península da Ponta d'Areia* [fotografia]. 2014. Disponível em: <https://globoplay.globo.com/v/2484275/>. Acesso em: 10 jan. 2026.

MUNDO EDUCAÇÃO. *Olho humano: um instrumento óptico*. Disponível em: <https://mundoeducacao.uol.com.br/fisica/olho-humano-um-instrumento-optico.htm>. Acesso em: 5 jul. 2025.

NISSAN, Noam *et al.* Extremely dense breasts: a comprehensive review of increased cancer risk and supplementary screening methods. *European Journal of Radiology*, v. 182, art. 111837, jan. 2025.

PARQUE DAS AVES. *Resgatando corujas da Mata Atlântica*. Disponível em: <https://www.parquedasaves.com.br/corujas/>. Acesso em: 3 nov. 2025.

PRASAD, R.; MURTHY, C. R.; RAO, B. D. Joint approximately sparse channel estimation and data detection in OFDM systems using sparse Bayesian learning. *IEEE Transactions on Signal Processing*, v. 62, n. 14, p. 3591–3603, 2014. Disponível em: <https://ieeexplore.ieee.org/document/6826590/>. Acesso em: 23 abr. 2026.

SANTANA, Nicole. *Multa por câmera de monitoramento: o que*

muda? *Webmotors*. Disponível em: <https://www.webmotors.com.br/wm1/noticias/multa-por-camera-o-que-muda>. Acesso em: 19 set. 2025.

SILVA, Michele A. da. *Retina*. *InfoEscola*. Disponível em: <https://www.infoescola.com/visao/retina/>. Acesso em: 22 ago. 2025.

SOILLE, Pierre. *Morphological image analysis: principles and applications*. Berlin: Springer-Verlag, 1999.

SSIM, Claude. *계산사진학 Image Restoration DNNS*. *Vélog*, [s.d.]. Disponível em: https://velog.io/@claude_ssim/%EA%B3%84%EC%82%B0%EC%82%AC%EC%A7%84%ED%95%99-Image-Restoration-DNNS. Acesso em: 10 jan. 2026.

SUCKLING, J. *et al.* The Mammographic Image Analysis Society Digital Mammogram Database. *Excerpta Medica International Congress Series*, v. 1069, p. 375–378, 1994.

SUCKLING, J.; PARKER, J.; DANCE, D.; ASTLEY, S.; HUTT, I.; BOGGIS, C.; RICKETTS, I.; STAMATAKIS, E.; CERNEAZ, N.; KOK, S.; TAYLOR, P.; BETAL, D.; SAVAGE, J. *Mammographic Image Analysis Society (MIAS) database v1.21*. Apollo – University of Cambridge Repository, 2015. DOI: <https://doi.org/10.17863/CAM.105113>.

UFJF. *Projeto Física e Cidadania: câmera fotográfica*. Disponível em: <https://www2.ufjf.br/fisicaecidadania/conteudo/camera-fotografica/>. Acesso em: 13 ago. 2025.

UNGARO, Francesco. *Grayscale photography of short-coated dog* [fotografia]. *Pexels*. Disponível em: <https://www.pexels.com/photo/grayscale-photography-of-short-coated-dog-143657/>. Acesso em: 3 nov. 2025.

UNIVERSIDADE FEDERAL DO PARANÁ. *Espectro eletromagnético aplicado ao sensoriamento remoto*. Disponível em: https://docs.ufpr.br/~centeno/m_sr2/texto/aula003/sriiaula003-001.html. Acesso em: 10 jan. 2026.

WIKIMEDIA COMMONS. *Schematic diagram of the human eye horizontal*. Disponível em: https://commons.wikimedia.org/wiki/File:Schematic_diagram_of_the_human_eye_horizontal_pt.svg. Acesso em: 10 out. 2025.

XIAN-YONG, Meng; CHE, Lei; LIU, Zhi-Hui; CHE, Ning; JI, Xiao-Nan. Towards a partial differential equation remote sensing image method based on adaptive degradation diffusion parameter. *Multimedia Tools and Applications*, v. 76, p. 17651–17667, 2017.

YANG, Xiqi; ZHANG, Qingfeng; LI, Zhan. Contour detection in Cassini ISS images based on Hierarchical Extreme Learning Machine and Dense Conditional Random Field. *arXiv*, 2019. DOI: <https://doi.org/10.48550/arXiv.1908.08279>.

YELMANOV, S.; ROMANYSHYN, Y. A new approach to image intensity transformation based on equalizing the distribution density of contrast. In: SHAKHOVSKA, N.; MEDYKOVSKYY, M. O. (org.). *Advances in Intelligent Systems and Computing IV*. Cham: Springer, 2020. v. 1080. DOI: https://doi.org/10.1007/978-3-030-33695-0_28.

ZHANG, L. *Image restoration*. Shanghai: School of Software Engineering, Tongji University, 2013.