Tutorial: Introdução à Visão Computacional usando OpenCV

Maurício Marengoni[1]

Denise Stringhini[2]

Resumo: Este tutorial apresenta conceitos introdutórios de processamento de imagens (filtros) e de visão computacional (segmentação, classificação, reconhecimento de padrão e rastreamento). Estes conceitos serão introduzidos utilizando a biblioteca OpenCV, que é distribuída gratuitamente e possui documentação farta na internet, com exemplos e aplicações práticas. Será mostrado como obter e como instalar a ferramenta para diversos tipos de plataformas e linguagens de desenvolvimento. Os conceitos de processamento de imagens e visão computacional serão discutidos não apenas no aspecto teórico, mas também serão apresentados exemplos de implementação para que os leitores possam entender e utilizar os exemplos apresentados neste tutorial.

Abstract: This tutorial presents basic concepts in Image Processing (filtering) and Computer Vision (segmentation, classification, pattern recognition and tracking). These concepts are presented using the OpenCV library, which is free and has large documentation in the Internet including examples and practical applications. It will be shown how to get and install the library for different operating systems and development environments. This tutorial presents theoretical aspects and their implementations using OpenCV so readers can understand and use the library afterwords.

[1] Universidade Presbiteriana Mackenzie, Faculdade de Computação e Informática e Pós Graduação em Engenharia Elétrica, {mmarengoni@mackenzie.br }

[2] Universidade Presbiteriana Mackenzie, Faculdade de Computação e Informática, {dstring@mackenzie.br }

1 Introdução

Não é clara a fronteira entre o processamento de imagens e visão computacional. Podemos dizer que processamento de imagens é um processo onde a entrada do sistema é uma imagem e a saída é um conjunto de valores numéricos, que podem ou não compor uma outra imagem. A visão computacional procura emular a visão humana, portanto também possui como entrada uma imagem, porém, a saída é uma interpretação da imagem como um todo, ou parcialmente. Como será visto neste tutorial os processos de visão computacional geralmente iniciam com o processamento de imagens. Conforme Gonzalez [1] o espectro que vai do processamento de imagens até a visão computacional pode ser dividido em tres níveis: baixo-nível, nível-médio e alto-nível. Os processos de baixo-nível envolvem operações primitivas, tais como a redução de ruído ou melhoria no contraste de uma imagem. Os processos de nível-médio são operações do tipo segmentação (particionamento da imagem em regiões) ou classificação (reconhecimento dos objetos na imagem). Os processos de alto-nível estão relacionados com as tarefas de cognição associadas com a visão humana.

Fig. 1: Veículo em imagem escura (esquerda). Após uma equalização de histograma, em nível de cinza, onde a placa do veículo pode ser lida (direita). Informação da placa e do veículo no retângulo.

A Figura 1 mostra as diferenças entre processamento de imagens e visão computacional. A imagem da esquerda mostra o veículo, porém, não é possível ler a placa do veículo, pois a imagem está muito escura. Na imagem da direita foi feita uma operação de equalização de histograma da imagem (esta operação será discutida mais a frente), tipicamente uma operação de processamento de imagem. O resultado é uma imagem mais clara e que permite a leitura da placa do veículo. Uma operação de visão computacional é a aplicação de um operador que extrai a placa do veículo e identifica as letras e números da placa, possibilitando que os dados do veículo sejam encontrados em um banco de dados.

OpenCV (*Open Source Computer Vision*) é uma biblioteca de programação, de código aberto, desenvolvida inicialmente pela Intel Corporation. O OpenCV implementa

uma variedade de ferramentas de interpretação de imagens, indo desde operações simples como um filtro de ruído, até operações complexas, tais como a análise de movimentos, reconhecimento de padrões e reconstrução em 3D. O pacote OpenCV está disponível gratuitamente na Internet [2] bem como o manual de referência [3].

2 OpenCV

A biblioteca OpenCV foi desenvolvida pela Intel e possui mais de 500 funções [4]. Foi idealizada com o objetivo 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.

A biblioteca está dividida em cinco grupos de funções: Processamento de imagens; Análise estrutural; Análise de movimento e *rastreamento* de objetos; Reconhecimento de padrões e Calibração de câmera e reconstrução 3D.

As principais funções são apresentadas a seguir, juntamente com os conceitos de processamento de imagens e visão computacional que devem ser empregados em seu uso.

3 Processamento de Imagens

Os processos de visão computacional, muitas vezes, necessitam de uma etapa de pré-processamento envolvendo o processamento de imagens. As imagens de onde queremos extrair alguma informação em alguns casos precisam ser convertidas para um determinado formato ou tamanho e precisam ainda ser filtradas para remover ruídos provenientes do processo de aquisição da imagem.

Os ruídos podem aparecer de diversas fontes, como por exemplo, o tipo de sensor utilizado, a iluminação do ambiente, as condições climáticas no momento da aquisição da imagem, a posição relativa entre o objeto de interesse e a câmera. Note que ruído não é apenas interferência no sinal de captura da imagem, mas também interferências que possam atrapalhar a interpretação ou o reconhecimento de objetos na imagem. A Figura 2 mostra imagens de árvores em condições diferentes para exemplificar estes tipos de interferência.

Os filtros são as ferramentas básicas para remover ruídos de imagens, neste caso, o ruído é aquele que aparece no processo de aquisição da imagem. A Figura 3 apresenta um exemplo de uma imagem com ruído (à esquerda) e da imagem filtrada (à direita).

Os filtros podem ser espaciais (filtros que atuam diretamente na imagem) ou de frequência, onde a imagem é inicialmente transformada para o domínio de frequência usando da transformada de Fourier (geralmente através da transformada de Fourier discreta) e então é filtrada neste domínio e em seguida a imagem filtrada é transformada de volta para o domínio de espaço.

Fig. 2: Imagens de árvores obtidas em condições diferentes, topo à esquerda uma imagem "normal", topo à direita com interferência de iluminação, baixo à esquerda interferência do período do ano e baixo à direita mudança do tipo de sensor.

Fig. 3: Do lado esquerdo uma imagem com ruído, e na direita a mesma imagem após filtragem (Imagens retiradas de [5]).

3.1Domínio de Espaço

O termo dominio espacial se refere à imagem em si, e métodos que atuam no domínio espacial estão baseados na manipulação direta dos pixels da imagem. Os processos no domínio espacial são caracterizados pela equação 1:

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

onde: f(x,y) é a imagem original, T(.) é uma transformação na imagem e g(x,y) é a imagem transformada. T é uma operação definida sobre uma vizinhança de influência do pixel que está localizado na posição x, y. A idéia de vizinhança de influência considera os pixels ao redor da posição x, y. Esta vizinhança é definida por uma região quadrada (ou retangular) e de tamanho (lado) impar. A Figura 4 mostra alguns exemplos de vizinhança com tamanhos variados, estas regiões, que definem matrizes nas operações de transformação, também são chamadas de máscaras.

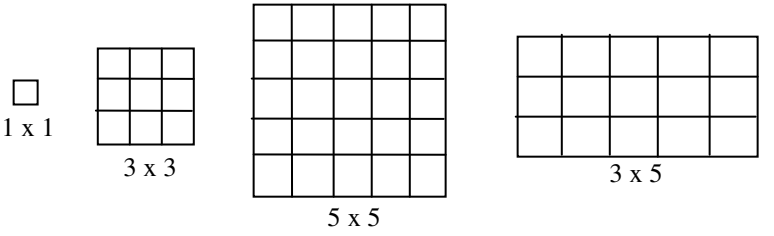


Fig.4: Regiões de vizinhança (máscaras) de tamanhos diferentes, porém, todos com lado impar.

3.1.1Transformação de Intensidade

No caso mais simples o operador T é computado em uma vizinhança de tamanho 1x1, isto é, apenas o valor do pixel no ponto é suficiente para determinar o valor na imagem processada. Este tipo de operação é chamada de função de transformação de intensidade, é utilizada para alterar a intensidade da imagem e pode ser aplicada a toda a imagem ou a uma parte dela. Uma operação bastante útil é a binarização de uma imagem, que utiliza um certo valor de corte (k). Este tipo de transformação é definida na equação 2:

$$g(x,y) = \begin{cases} 1 & , \text{se } f(x,y) \geq k \\ 0, & \text{caso contrario} \end{cases} \qquad (2)$$

A Figura 5 mostra a aplicação desta função para um valor de k=84. Esta técnica é utilizada para encontrar componentes conexas na imagem e isolar objetos de interesse.

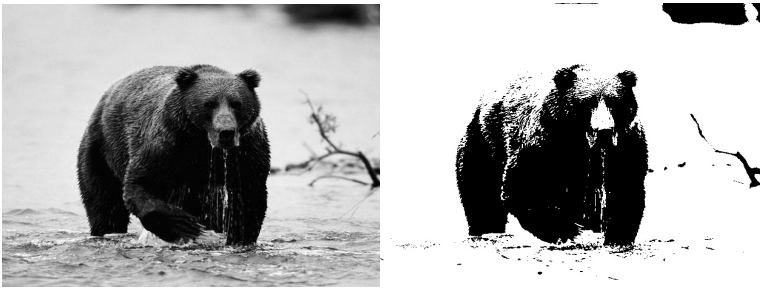


Fig.5: À esquerda uma imagem de um urso preto, à direita a imagem binarizada, com o urso em destaque (maior bloco de pixels com valor 0) . A binarizacao foi obtida com o valor de k=84.

Note que na função de binarização o corte é feito de forma abrupta, todos os valores acima de k (valores mais claros ou com a mesma intensidade de k) são mapeados para o valor 1 (branco) e os valores menores que k (mais escuros que k na imagem) são mapeados para 0 (preto). A função de binarização tem o formato apresentado na Figura 6.

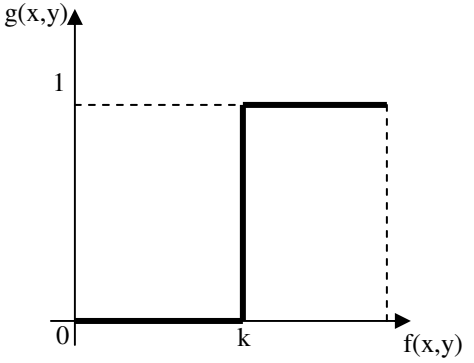


Fig.6: Função de binarização para um valor k.

Esta transição pode não ser tão abrupta como na função de binarização, por exemplo, a função de realce de contraste é dada pela equação 3:

$$g(x,y) = \frac{1}{1+(m/f(x,y))^{E}} \qquad (3)$$

onde, *m* é o valor médio da região que se deseja realçar o contraste, e *E* define a inclinação da curva, e consequentemente os valores de mapeamento para a imagem de saída. A função de transição é apresentada na Figura 7. Note que os valores de saída não são apenas 0 ou 1 mas, na região de interesse existe uma variação no contraste na imagem de saída.

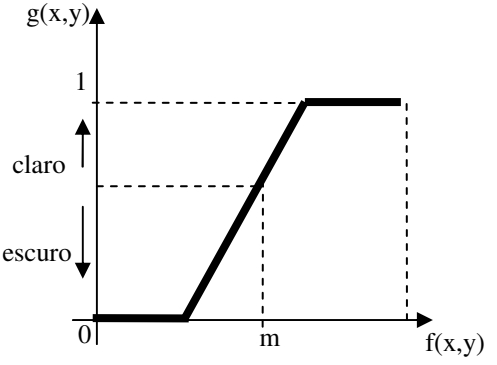


Fig. 7: Função de transição para realce de contraste.

Para alguns tipos de imagens, por exemplo, imagens representadas no domínio de frequência, é comum se obter valores de intensidade que vão de 0 a 10^7. Estas imagens, quando apresentadas no monitor não apresentam um contraste que permita a identificação do espectro de frequência. Para estes casos pode-se utilizar uma transformação logarítmica, que é definida pelo operador dado pela equação 4:

$$g(x, y) = C * \log(1 + f(x, y)) \qquad (4)$$

onde *C* é uma constante qualquer (valores para ajustar os valores dos pixels), geralmente maior que 1. O 1 na expressão do logaritmo é para se resolver o caso de f(x,y)=0.

3.1.2 Histogramas

Os histogramas são ferramentas de processamento de imagens que possuem grande aplicação prática. Os histogramas são determinados a partir de valores de intensidade dos pixels. Entre as principais aplicações dos histogramas estão a melhora da definição de uma imagem, a compressão de imagens, a segmentação de imagens ou ainda a descrição de uma imagem. Algumas destas aplicações serão descritas com maiores detalhes neste artigo.

O histograma de uma imagem I, cujos valores de intensidade estejam entre 0 e G, é definido pela equação 5:

$$h(I_k) = n_k \qquad (5)$$

onde I_k é um valor de intensidade k, ($0 \leq k \leq G$) da imagem I e n_k é o número de pixels na imagem I que possuem a intensidade k. É possível normalizar um histograma, representando os valores em termos de porcentagem, conforme mostrado na equação 6:

$$p(I_k) = \frac{h(I_k)}{n} = \frac{n_k}{n} \qquad (6)$$

onde *n* é o número de pixels da imagem. A Figura 8 mostra como um histograma é determinado.

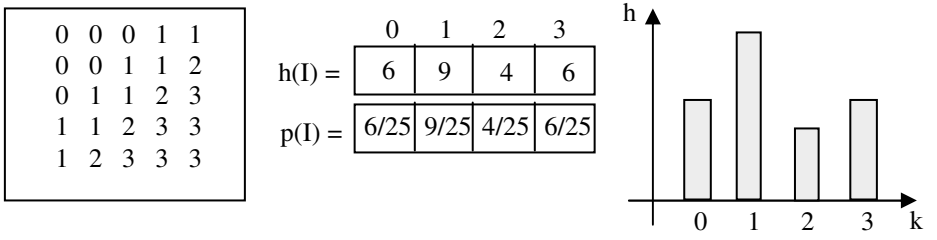


Fig. 8: À esquerda uma imagem I, ao centro o histograma da imagem em valores (h(I)) e em porcentagem (p(I)), à direita uma representação do histograma de forma gráfica.

Uma operação bastante comum utilizando histogramas é o ajuste dos valores de intensidade de forma a melhorar o contraste em uma imagem. Esta operação é chamada de equalização de histogramas. A idéia desta operação é mapear os valores de intensidade de uma imagem de um intervalo pequeno (pouco contraste) para um intervalo maior (muito contraste) e ainda distribuir os pixels ao longo da imagem de forma a obter uma distribuição uniforme de intensidades (embora na prática isso quase sempre não ocorra).

A expressão que fornece um histograma equalizado é apresentada na equação 7:

$$h\_eq(k) = \frac{(L-1)}{MN} \sum_{j=0}^{k} n_j \qquad (7)$$

onde k é a intensidade no histograma equalizado, L é o valor máximo de intensidade na imagem, M e N são as dimensões da imagem e n_j é o número de pixel na imagem com valor de intensidade igual a j. A Figura 9 mostra um exemplo de uma imagem que foi ajustada utilizando equalização de histograma.

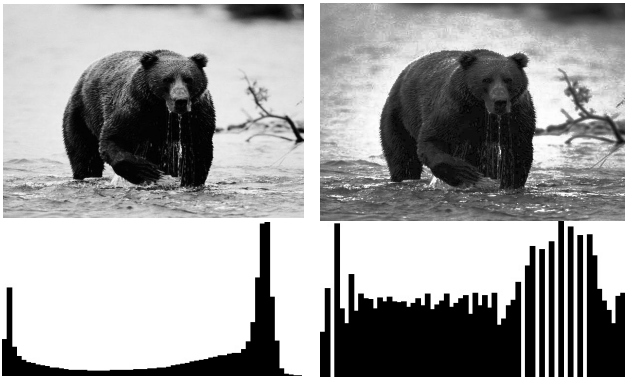


Fig. 9: Topo à esquerda, imagem em nível de cinza, e abaixo dela o histograma da imagem. Topo à direita, a mesma imagem após equalização, e o histograma equalizado da imagem.

3.1.3 Filtragem no Domínio Espacial

Existem dois conceitos, de certa forma similares, que estão relacionados com a filtragem no domínio de espaço, são eles os de correlação e convolução. Como já mencionado, as transformações no domínio de espaço dependem de uma vizinhança de influência (máscara) do pixel que está sendo considerado. A idéia destas duas operações é a seguinte: cria-se uma máscara com dimensão d, onde cada posição da máscara possui um determinado valor. Em seguida coloca-se a máscara com sua posição central sobre a imagem na posição (x,y) e, para cada posição da máscara executa-se o produto do valor da máscara pelo valor do pixel, faz-se a somatória destes valores obtidos na multiplicação e substitui-se o valor da posição (x,y) por este resultado. Esta operação é apresentada esquematicamente na Figura 10.

A diferença entre a correlação e a convolução está na forma como a máscara é utilizada: na correlação a operação é feita conforme indicado na Figura 10, já na operação de convolução a máscara é rotacionada de 180 graus. Note que, se a máscara for simétrica as operações de correlação e convolução são idênticas. As expressões que definem estas duas operações são apresentadas nas equações 8 e 9.

Correlação:

$$g(x,y) = \sum_{i=-m/2}^{m/2} \sum_{j=-n/2}^{n/2} f(x+i, y+j) * w(i,j) \quad (8)$$

Convolução:

$$g(x,y) = \sum_{i=-m/2}^{m/2} \sum_{j=-n/2}^{n/2} f(x-i, y-j) * w(i,j) \quad (9)$$

onde w é a máscara utilizada no processo de correlação/convolução, e m e n são as dimensões da máscara. O tipo de filtro depende dos valores da máscara, no geral, as máscaras são simétricas e, portanto, pode-se aplicar tanto a convolução como a correlação.

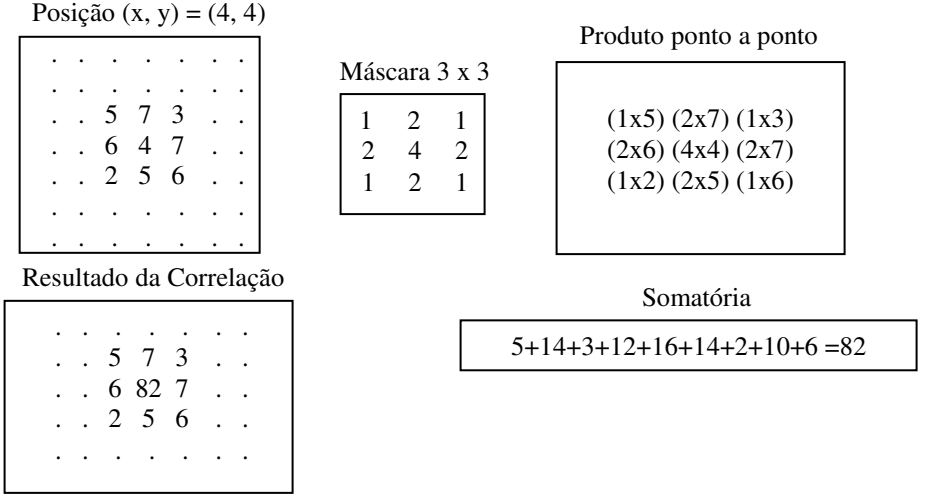


Fig. 10: Esquema da operação de Correlação, no topo à esquerda a imagem, ao centro a máscara utilizada. No topo à direita a indicação dos produtos dos valores da máscara pelos valores dos pixels da imagem, ponto a ponto. Embaixo à direita a somatória dos valores multiplicados e, finalmente, embaixo à esquerda o novo valor do pixel na imagem após a correlação.

Filtros Estatísticos

Os filtros estatísticos são os filtros espaciais mais comuns, entre eles podemos citar o filtro de média, mediana, moda, mínimo e máximo. O filtro de média, também chamado de

filtro-caixa, é um filtro do tipo passa-baixa. O efeito de um filtro passa-baixa é de suavizaçao da imagem e minimização dos ruídos, atenuando as transições abruptas que correspondem a frequências altas, porém, o efeito acaba sendo de embassamento ou borramento da imagem que acaba removendo os detalhes finos da imagem. A expressão de um filtro de média é dada pela equação 10:

$$h(i,j) = \begin{cases} \dfrac{1}{mn}, \text{ se } |i| < \dfrac{m}{2} \text{ e } |j| < \dfrac{n}{2} \\ 0, \text{caso contrario} \end{cases} \qquad (10)$$

onde m e n são as dimensões de uma máscara qualquer. A Figura 11 mostra dois exemplos de máscaras do tipo média.

$$h(i,j) = 1/9 * \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad h(i,j) = 1/16 * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Fig. 11: À esquerda tem-se um filtro média simples do tipo caixa. À direita tem-se um filtro de média ponderada com atenuação variando de acordo com a distância e orientação ao centro da máscara.

Os outros filtros estatísticos não são lineares e são utilizados de forma a evitar uma suavização homogênea da imagem. No filtro da mediana os valores dos pixels são ordenados e o valor que ocupa a posição mediana é selecionado para a posição (x, y) da imagem filtrada. Este filtro tende a reduzir o efeito de ruído de pulso, do tipo *salt and peper*, pois valores pontuais raramente aparecem juntos e portanto nunca ocupam a posição mediana. O filtro de máximo substitui o valor da posição (x, y) pelo valor máximo da máscara, este filtro tem a tendência de clarear a imagem. Analogamente o filtro de mínimo substitui o valor da posição (x, y) pelo valor mínimo da máscara, este filtro tem a tendência de escurecer a imagem. O filtro de moda seleciona para a posição (x, y) da imagem o valor que ocorre com maior frequência na máscara, este tipo de filtro tende a homogeneizar os valores na imagem.

Filtros Gaussianos

Um filtro Gaussiano tem os valores da máscara determinados a partir de uma função bidimensional Gaussiana discreta, com média igual a zero e desvio padrão σ, como mostrado na equação 11:

$$Gauss(x,y) = \frac{1}{2\pi\sigma^2} \exp(\frac{-(x^2+y^2)}{2\sigma^2} \qquad (11)$$

onde x e y são as posições na máscara e Gauss(x, y) dá o valor a ser colocado na posição (x, y) da máscara. Os filtros Gaussianos são filtros de média e são utilizados para suavizar a imagem de forma ponderada e simétrica. Um exemplo de máscara Gaussiana de tamanho 5x5 é apresentado na Figura 12.

$$\text{Gauss}(x,y) = 1/256 * \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Fig. 12: Máscara Gaussiana para um filtro do tipo passa-baixa.

Filtro Passa-Alta

O filtro do tipo passa-alta é utilizado para realçar bordas ou regiões de interesse com transições abruptas de intensidade. O problema deste tipo de filtro é que ele geralmente realça também ruídos do processo de obtenção da imagem. Alguns exemplos de máscaras para filtros passa-alta são apresentadas na Figura 13.

$$h(i,j) = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad h(i,j) = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Fig. 13: Máscara para um filtro do tipo passa-alta, note que a soma dos valores dentro da máscara somam zero.

3.2 Domínio de Frequência

É possível fazer uma troca de base em uma imagem e representá-la em termos de uma soma ponderada infinita de um conjunto de senóides. Esta representação mostra que mudanças rápidas na imagem são representadas por frequências altas e, por outro lado, mudanças suaves são representadas por frequências baixas. Esta mudança de base pode ser feita utilizando a transformada de Fourier. As expressões que computam a transformada de Fourier no modo contínuo e no modo discreto são apresentadas nas equações 12 e 13:

Contínuo:

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-j2\pi(ux+vy)} \qquad (12)$$

Discreto:

$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(ux/M+vy/N)} \qquad (13)$$

A expressão discreta é a utilizada na DFT (Discrete Fourier Transform) que geralmente é utilizada na implementação em computador da transformada de Fourier. Para mais detalhes sobre o uso da transformada de Fourier veja Gonzalez e Woods [1].

Matematicamente o processo de convolução no domínio espacial corresponde à multiplicação de duas expressões no domínio de frequência (veja a equação 14).

$$g(x,y) = f(x,y) ** w(i,j) \Leftrightarrow G(u,v) = F(u,v) * W(r,s) \qquad (14)$$

onde ** indica a convolução entre a imagem f(x,y) com a máscara w(i,j).
G(u,v) é a transformada de Fourier de g(x,y), que é o produto de F(u,v) por W(r,s) onde F(u,v) é a transformada de Fourier de f(x,y) e W(r,s) é a transformada de Fourier de w(i,j).

Existem, porém alguns detalhes importantes para se filtrar uma imagem no domínio de frequência que será apresentado abaixo, maiores detalhes sobre o procedimento pode ser encontrado em Gonzalez e Woods [1]:

a) Dada uma imagem f(x,y) de tamanho M x N, obter uma imagem com entorno fp(x,y) de tamanho P x Q onde P = 2M e Q = 2N preenchida com zeros no entorno.

b) Multiplicar fp(x,y) por $(-1)^{(x+y)}$ para centralizar a transformada.

c) Determinar a DFT da imagem obtida em b.

d) Criar um filtro simétrico H(u,v) de tamanho P x Q com centro em (P/2,Q/2)

e) Fazer o produto H(u,v)*F(u,v) obtido na etapa c.

f) Obter a imagem no domínio de espaço fazendo o inverso da transformada de Fourier gp(x,y).

g) Multiplicar gp(x,y) por $(-1)^{(x+y)}$

h) Remover o entorno da imagem obtendo então g(x,y), que é a imagem filtrada.

A expressão que determina o inverso da transformada de Fourier discreta é dada pela equação 15:

$$f(x,y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) e^{j2\pi(ux/M + vy/N)} \qquad (15)$$

Portanto, definir um filtro no domínio de frequência corresponde a encontrar uma máscara para ser utilizada em conjunto com a imagem transformada e assim obter a imagem filtrada desejada. A seguir veremos alguns casos de filtros no domínio de frequência.

Filtro Passa-baixa

Um filtro passa-baixa ideal é dado pela expressão 16:

$$H(u,v) = \begin{cases} 1, \text{ se } F(u,v) < F_0 \\ 0, \text{caso contrario} \end{cases} \qquad (16)$$

onde F_0 é uma frequência de corte. Este filtro faz um corte abrupto em uma certa frequência, por isso o nome de filtro ideal. O formato deste filtro é apresentado na Figura 14.

Outro filtro do tipo passa-baixa é o filtro de Butterworth cuja expressão é mostrada na equação 17.

$$H(u,v) = \sqrt{\frac{1}{1 + (F(u,v)/F_0)^{2n}}}$$

(17)

onde n define a ordem do filtro de Butterworth, na prática o valor de n define a forma como a função atenua a frequência a partir da origem do filtro.

Um outro filtro do tipo passa-baixa é o filtro Gaussiano que no domínio de frequência é dado pela expressão 18.

$$H(u,v) = e^{-F^2(u,v)/2\sigma^2} \quad (18)$$

O filtro Gaussiano obtém uma atenuação menos suave que o filtro de Butterworth para a mesma frequência de corte.

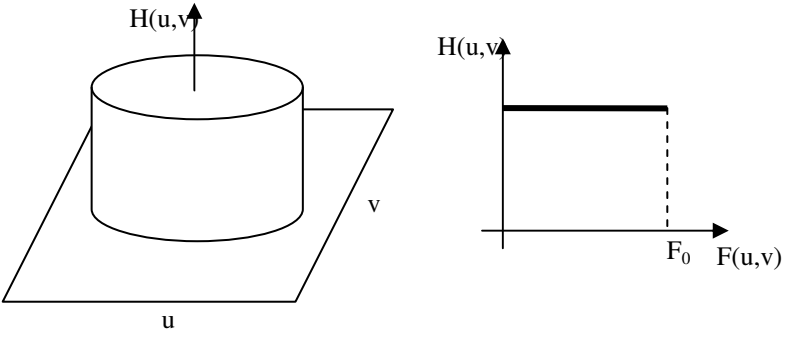


Fig. 14: Filtro ideal no domínio de frequência. À esquerda o formato do filtro em 3D mostrando o corte acentuado na frequência de corte. À direita uma vista esquemática do mesmo filtro.

Filtro Passa-alta

Os filtros passa-alta no domínio de frequência são definidos de forma similar aos filtros passa-baixa. O filtro passa-alta ideal é dado pela equação 19:

$$H(u,v) = \begin{cases} 1, \text{ se } F(u,v) > F_0 \\ 0, \text{caso contrario} \end{cases} \quad (19)$$

A expressão do filtro passa-alta de Butterworth é dada pela equação 20:

$$H(u,v) = \sqrt{\frac{1}{1 + (F_0 / F(u,v))^{2n}}} \quad (20)$$

A expressão do filtro passa-alta Gaussiano é dada pela equação 21:

$$H(u,v) = 1 - e^{-F^2(u,v)/2\sigma^2} \quad (21)$$

4 Visão Computacional

Na parte relacionada ao processamento de imagens ficou caracterizado o caráter de processo de baixo nível, mais precisamente a eliminação de ruídos e melhoria no contraste das imagens. Neste item começamos a migrar para os processos mais relacionados com a visão computacional. Veremos inicialmente um processo de nível médio (segmentação e reconhecimento) para em seguida analisarmos processos mais cognitivos, como o rastreamento de um objeto numa sequência de imagens.

4.1 Segmentação

O processo de segmentação consiste em particionar uma imagem em regiões, ou objetos distintos. Este processo é geralmente guiado por características do objeto ou região, como por exemplo, cor ou proximidade. O nível de detalhamento em um processo de segmentação depende da tarefa a ser executada e da resolução da imagem que se tem, por exemplo, se procuramos por uma casa em imagens que foram obtidas do nível da rua, estamos procurando regiões que ocupam uma boa porcentagem da imagem (regiões grandes), porém, se procuramos por casas a partir de imagens de satélite, estamos procurando regiões pequenas. Embora a tarefa seja a mesma, a resolução das imagens é diferente e o tratamento utilizado no processo de segmentação pode ser diferente também.

Existem diversas técnicas que podem ser utilizadas para a segmentação, nosso objetivo aqui não é esgotar estas técnicas, mas apresentar algumas delas e mostrar como utilizá-las através do OpenCV.

As técnicas de segmentação que serão apresentadas neste tutorial podem ser classificadas em três grupos, a saber:

- Segmentação por detecção de borda.
- Segmentação por corte.
- Segmentação por crescimento de região.

Segmentação por Detecção de Borda

Uma borda em uma imagem é caracterizada por uma mudança, normalmente abrupta, no nível de intensidade dos pixels. Os detectores de borda são definidos para encontrar este tipo de variação nos pixels e quando estes pixels estão próximos eles podem ser conectados formando uma borda ou um contorno e assim definindo uma região ou objeto.

Variações nos níveis de intensidade dos pixels podem ser determinadas pelas derivadas primeira e/ou derivada segunda. Alguns métodos de determinação de borda utilizam estas técnicas. O processo consiste em se definir máscaras que caracterizem estas variações e em seguida fazer a convolução da imagem pela máscara.

Os operadores mais comuns baseado na derivada primeira (operadores de gradiente) de uma imagem são os operadores de Prewitt e de Sobel, a Figura 15 mostra as máscaras relacionadas a cada um destes operadores para detecção de bordas verticais e horizontais. Para determinarmos bordas nas diagonais basta rotacionar a máscara do ângulo desejado. A

Figura 16 apresenta uma máscara do detector Sobel para um ângulo de 45º. As expressões matemáticas que definem estes operadores são apresentadas nas equações 22 e 23:

Prewitt:

$$g(x) = \frac{\partial f(x,y)}{\partial x} = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3)$$
$$g(y) = \frac{\partial f(x,y)}{\partial y} = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7) \quad (22)$$

Sobel:

$$g(x) = \frac{\partial f(x,y)}{\partial x} = (z_7 + 2*z_8 + z_9) - (z_1 + 2*z_2 + z_3)$$
$$g(y) = \frac{\partial f(x,y)}{\partial y} = (z_3 + 2*z_6 + z_9) - (z_1 + 2*z_4 + z_7) \quad (23)$$

Note que o operador de Sobel é semelhante ao de Prewitt sendo a única diferença o valor 2 que multiplica o termo central nas linhas da máscara. Este fator tende a suavizar o resultado do operador, atenuando ruídos.

Z_1	Z_2	Z_3
Z_4	Z_5	Z_6
Z_7	Z_8	Z_9

Prewitt

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

Sobel

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

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

Fig. 15: Em cima uma máscara genérica com a posição dos valores utilizados nas equações de Prewitt e Sobel. Abaixo, à esquerda, as máscaras dos operadores de Prewitt, horizontal e vertical e abaixo, à direita as máscaras dos operadores de Sobel.

Sobel

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

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

Fig. 16: Máscaras dos operadores de Sobel para um ângulo de 45º.

É possível ainda definir operadores que atuam utilizando a derivada segunda dos pixels (operadores Laplacianos), estes operadores são baseados no fato que quando existe uma borda a derivada segunda atravessa o eixo das absissas (este efeito é conhecido como *zero-crossing*). Um dos primeiros detectores de borda utilizando este conceito foi o de Marr-Hildreth [6], que foi baseado nos seguintes conceitos:

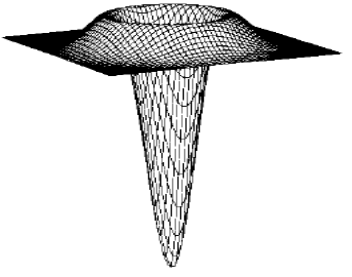
- A mudança de intensidade não é independente da escala da imagem, logo os operadores devem ter tamanhos diferentes.
- A mudança de intensidade de forma abrupta faz com que a derivada segunda atravesse o eixo das absissas.

Marr e Hildreth definiram um operador Laplaciano de uma função Gaussiana, conforme a equação 24, o formato da curva e a máscara 5x5 que representa a função são apresentados na Figura 17. O operador é simétrico, uma característica da curva Gaussiana, o que evita o uso de operadores múltiplos e tem um efeito de suavização da imagem. O tamanho da máscara deve ser definido de acordo com o valor de σ, normalmente este valor é um número impar maior que 6*σ, uma vez que 99,7% do volume de uma Gaussiana esta definido entre ±3σ. Note que os coeficientes da máscara somam zero, isto garante que em locais onde a imagem é homogênea não é acrescentado valor algum na imagem resultante.

$$\nabla^2 G(x,y) = \left(\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} \right) e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{24}$$

O detector de borda de Marr-Hildreth pode ser implementado facilmente através de duas operações :

- Filtrar a imagem com uma Gaussiana do tipo passa-baixa de tamanho n x n.
- Computar a Laplaciana usando uma máscara 3x3 do tipo mostrado na Figura 13, à direita, porém, com os sinais dos pesos trocados.



0	0	1	0	0
0	1	2	1	0
1	2	-16	2	1
0	1	2	1	0
0	0	1	0	0

Fig. 17: À esquerda a superfície da Laplaciana de uma Gaussiana e à direita uma máscara 5x5 representando a superficie.

A técnica de detecção de borda desenvolvida por Canny [7], embora seja a mais complexa, é também a de melhor desempenho entre as técnicas discutidas até aqui. O detector de bordas de Canny possui três objetivos básicos: baixa taxa de erro (todas as bordas devem ser encontradas), os pontos da borda devem ser bem localizados (as bordas localizadas devem ser próximas das bordas reais) e resposta única para os pontos de uma borda (o operador deve retornar apenas um ponto para cada ponto sobre a borda). Canny conseguiu representar estas idéias matematicamente e, utilizando otimização numérica, definiu um detector de borda basedo na primeira derivada de uma Gaussiana. Numa primeira etapa a imagem é convolvida por uma função Gaussiana, em seguida são determinados a magnitude e a direção, conforme indicado pelas expressões nas equações 25:

$$\begin{aligned} g(x,y) &= f(x,y) ** G(x,y) \\ M(x,y) &= \sqrt{g_x^2 + g_y^2} \\ \alpha(x,y) &= \tan^{-1}\left(\frac{g_y}{g_x}\right) \\ g_i &= \frac{\partial g(x,y)}{\partial i} \end{aligned} \qquad (25)$$

Os valores de g_i podem ser obtidos a partir dos operadores de Prewitt ou de Sobel. M(x, y) é uma imagem que contém a informação da magnitude em cada pixel e α(x, y) é uma imagem que contém a direção da normal à borda para cada pixel. A idéia é, para cada pixel *p*, verifica se pelo menos um dos vizinhos de *p* possui a mesma direção que *p*, se sim, marca o pixel com M(x, y) (as coordenadas de *p*), senão marca a imagem de saída com 0.

Finalmente utiliza-se um operador de corte para reduzir pontos de borda falsos. O operador de corte, neste caso, é baseado em histerese. O operador de corte baseado em histerese possui dois valores, um valor chamado de alto e outro chamado de baixo. São feitas as operacoes de corte usando estes valores e são obtidas duas imagens binárias, estas imagens são subtraídas uma da outra para se obter a imagem final ($Img_{baixa} - Img_{alta}$). O tamanho da Gaussiana utilizada é o mesmo definido para o operador de Marr-Hildreth. A Figura 18 mostra um exemplo de segmentação de imagem utilizando o operador de Canny.

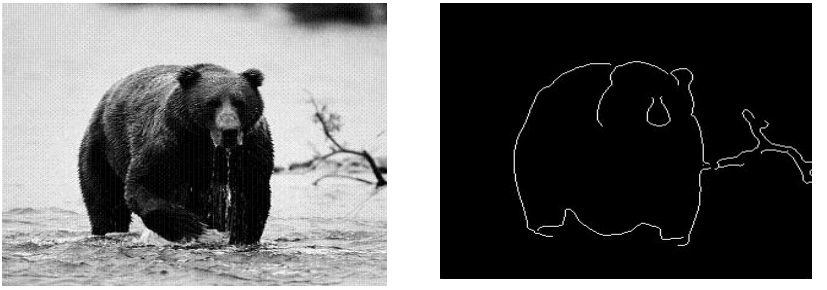


Fig. 18: Exemplo de segmentação usando o operador de Canny. À esquerda a imagem original e à direita a imagem segmentada. Foi utilizado um valor de corte alto de 0.5 e desvio padrão de 2.

Segmentação por Corte

A segmentação de imagens por corte é simples de ser implementada, é rápida em termos computacionais e utiliza de propriedades intuitivas para criar a imagem segmentada. A segmentação por corte particiona uma imagem diretamente em regiões baseado simplesmente nos valores de intensidade e/ou propriedades destes valores.

A idéia central deste tipo de operador é a de verificar no histograma da imagem quantas regiões existem (picos e vales) e segmentar a imagem baseado nesta informação. A Figura 19 apresenta um exemplo de imagem e mostra o histograma da imagem, apresentando um possível local onde a imagem poderia ser particionada e a imagem binarizada após o corte.

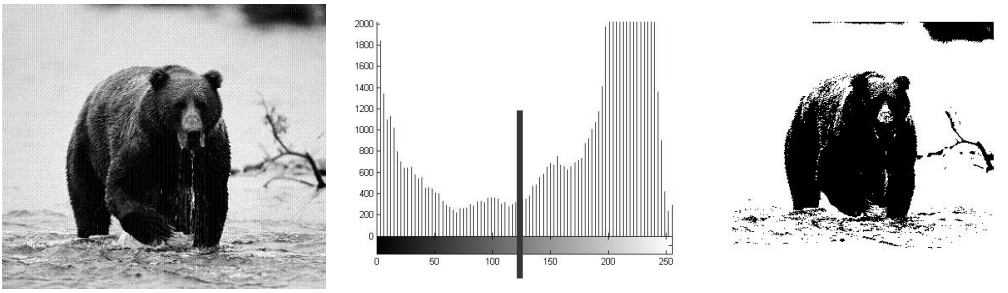


Fig. 19: Segmentação de imagem utilizando a técnica de corte. À esquerda a imagem em nível de cinza, ao centro o histograma com o ponto de corte e à direita a imagem binarizada.

Note que, utilizando esta técnica é possível definir mais de uma região e criar uma imagem segmentada que não seja simplesmente uma imagem binária. É possível ainda fazer tratamentos no histograma para minimizar efeitos de ruídos que apareçam na imagem.

Segmentação Baseada em Crescimento de Região

Esta técnica de segmentação encontra regiões diretamente na imagem agrupando pixels ou sub-regiões em regiões maiores baseado em critérios de crescimento pré-definidos. O procedimento parte de um conjunto de pontos, chamados de sementes, e, a partir destes pontos vai agrupando pontos utilizando uma vizinhança de influência, formando as regiões. Nesta vizinhança são analisadas propriedades e são medidas similaridades para determinar se o pixel faz parte ou não da região sendo considerada. As propriedades normalmente consideradas são: cor, intensidade de nível de cinza, textura, momentos, etc.

Detalhes importantes desta técnica são a definição das sementes e a definição de um critério de parada para o crescimento de regiões. Um algoritmo básico de crescimento de regiões que considera uma vizinhança de 8 é descrito abaixo, o algoritmo considera f(x,y) a imagem de entrada, S(x,y) uma imagem que define as sementes do processo de crescimento de região e Q a propriedade a ser considerada na definição da região:

a) Encontre as componentes conexas em S e ache, para cada uma delas, o ponto central da região.

b) Crie uma imagem t tal que, para cada posição (x,y) em f, t(x,y)=1 se f(x,y) possui a propriedade Q.

c) Crie uma imagem g fazendo a união entre S e t se os pontos em t estiverem conectados a uma semente, usando uma vizinhança de 8.

d) Rotule cada região de g com um valor diferente. Esta é a imagem final segmentada.

A Figura 20 apresenta um exemplo de segmentação pelo método de crescimento de região. Note que esta segmentação possui menos componentes conexas que a binarização.

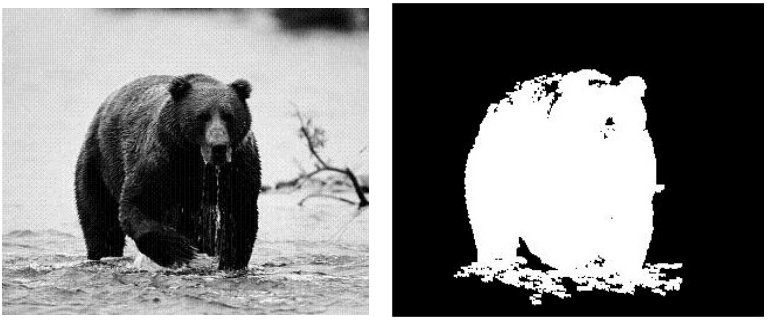


Fig.20: À esquerda a imagem original e à direita a imagem segmentada utilizando o método de crescimento de região. Neste caso a característica utilizada foi de valor de intensidade menor que 125.

Uma alternativa a este método é subdividir a imagem inicialmente em um conjunto de regiões disjuntas e unir ou não as regiões de forma a atender as características que estão sendo utilizadas no processo de segmentação. Este tipo de técnica pode utilizar uma estrutura de *quadtrees,* que são árvores onde um nó possui exatos quatro filhos. As imagens correspondentes a cada filho são chamadas de *quadregions* ou *quadimages*. Um exemplo do esquema utilizado no método de dividir e unir regiões é apresentado na Figura 21.

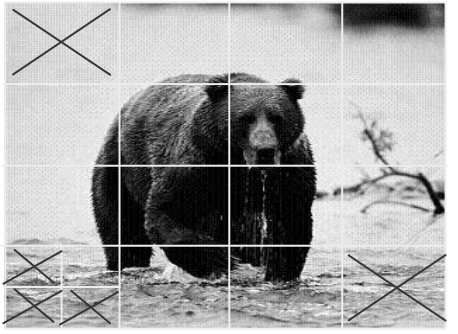


Fig. 21: Exemplo de funcionamento do método de divisão e união de regiões. As regiões marcadas com o X vermelho não são divididas novamente pois não apresentam pixels com valores abaixo de 125.

4.2 Reconhecimento de Padrões

Reconhecer significa conhecer de novo, e isto implica num processo onde existe algum conhecimento prévio e algum tipo de armazenamento do conhecimento sobre o objeto a ser reconhecido. Esta é a parte onde os sistemas de visão possuem uma intersecção com a área de inteligência artificial. Para fazer o reconhecimento um sistema de visão necessita uma base de conhecimento dos objetos a serem reconhecidos, esta base de conhecimento pode ser implementada diretamente no código, através, por exemplo, de um sistema baseado em regras, ou esta base de conhecimento pode ser aprendida a partir de um conjunto de amostras dos objetos a serem reconhecidos utilizando técnicas de aprendizado de máquina.

O reconhecimento de objetos é uma das principais funções da área de visão computacional e está relacionado diretamente com o reconhecimento de padrões. Um objeto pode ser definido por mais de um padrão (textura, forma, cor, dimensões, etc) e o reconhecimento individual de cada um destes padrões pode facilitar o reconhecimento do objeto como um todo. As técnicas de reconhecimento de padrões podem ser divididas em dois grandes grupos: estruturais, onde os padrões são descritos de forma simbólica e a estrutura é a forma como estes padrões se relacionam; o outro grupo é baseado em técnicas que utilizam teoria de decisão, neste grupo os padrões são descritos por propriedades quantitativas e deve-se decidir se o objeto possui ou não estas propriedades.

Os processos de reconhecimento de padrões podem ainda ser uma mistura das técnicas utilizadas nestes dois grupos, por exemplo, no processo de reconhecimento de faces apresentado em Cândido [8], é utilizado um modelo estrutural para determinar o local mais provável para se encontrar partes de uma face (boca, olhos e pele), conforme apresentado na Figura 22. Cada uma destas partes pode agora ser reconhecida utilizando outro tipo de técnica, por exemplo, os olhos podem ser reconhecidos utilizando uma rede neural, a pele pode ser reconhecida por uma análise estatística e a boca pode ser reconhecida por um critério de distância mínima, todas são técnicas de teoria de decisão.

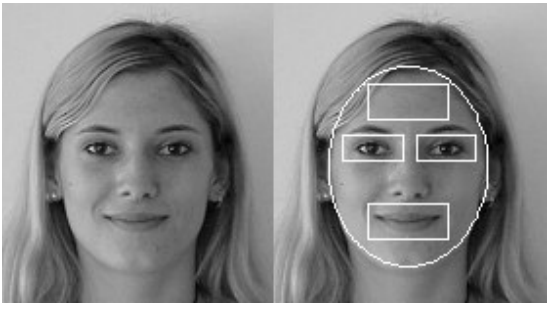


Fig. 22: Exemplo de estrutura para o reconhecimento de faces. O modelo indica locais onde se esperam encontrar olhos, boca e pele. Reconhecer as partes pode levar ao reconhecimento da face.

É difícil encontrar técnicas estruturais prontas em bibliotecas, uma vez que estas técnicas dependem da estrutura de cada objeto. Para alguns objetos específicos, porém, é

possível encontrar pacotes prontos. O OpenCV não possui uma ferramenta pronta que utilize este tipo de técnica. As técnicas baseadas em teoria de decisão são mais gerais e podem ser adaptadas a diferentes tipos de objetos. O OpenCV possui diversas técnicas nesta categoria [9], conforme apresentado na tabela 1.

Tabela 1: Algoritmos e comandos para reconhecimento de padrões no openCV.

Algoritmo	Descrição	Comando
Mahalanobis	Uma medida de distância que verifica a elasticidade do espaço dos dados.	`cvMahalanobis`
K-médias	Um algoritmo de agrupamento não supervisionado.	`cvKMeans2`
Classificador Bayesiano	Um classificador que assume que as características são Gaussianas e estatisticamente independentes	`cvNormalBayesClassifier`
Árvore de Decisão	Um classificador discriminativo	`cvDTree`
Boosting	Um classificador de grupo discriminativo. A classificação geral depende da combinação de pesos dada a cada classificador.	`cvBoost`
Árvores Randômicas	Um conjunto de árvores de decisão combinadas.	`cvRTrees`
Classificador Haar	Um classificador baseado em boosting.	`cvHaarDetectObjects`
Maximização esperada	Um agrupador não supervisionado baseado na técnica generativa.	`---`
K-vizinhos	O classificador discriminativo mais simples possível.	`---`
Redes Neurais	Baseado nos perceptrons de multiplas camadas.	`---`
Support Vector Machine	Um classificador discriminativo capaz de fazer regressões.	`---`

Para maiores detalhes sobre estes algoritmos de reconhecimento de padrões, como treiná-los em novos objetos e como utilizá-los, verifique em Bradski [9].

4.3 Rastreamento

O processo de rastreamento é um processo de reconhecer um padrão em uma sequência de imagens. O rastreamento poderia ser feito desta forma, porém, a busca em cada imagem de uma sequência sem o uso de qualquer conhecimento específico é relativamente lenta. Os processos de rastreamento atrelam um conhecimento sobre o movimento do objeto que está sendo rastreado para minimizar a busca entre as imagens em uma sequência.

Os processos de rastreamento podem ser aplicados em diversas áreas, indo de sistemas de segurança/vigilância até o uso em sistemas de interface humano-computador. Existem métodos para se prever a posição do objeto frame a frame, indo de filtros Kalman [11], até processos com filtros de partículas [10]. No openCV as técnicas de rastreamento incluem dois componentes principais: identificação de objetos e modelagem da trajetória. Existem algumas funções que são utilizadas para o rastreamento, baseadas nos algoritmos de "meanshift" e "camshift".

4.3.1 Corner Finding

Este tipo de técnica pressupõe a busca por um mesmo objeto de interesse em sequências de frames num *stream* de vídeo. A idéia básica é buscar pontos diferenciados em uma imagem, passíveis de serem novamente encontrados em frames subsequentes.

O OpenCV possui uma função de reconhecimento de objetos que implementa uma técnica (descrita em [**9**]) baseada no cálculo de derivadas de segunda ordem usando operadores de Sobel e que são usadas para o cálculo dos *eigenvalues* necessários.

A função `cvGoodFeaturesToTrack` retorna um array de localizações de pixels onde se espera encontrar outra imagem similar. Além do reconhecimento de objetos em sequência, esta técnica pode ser aplicada para relacionar imagens capturadas sob diferentes pontos de vista, entre outras aplicações.

4.3.2 Subpixel Corners

A técnica anterior retorna coordenadas inteiras, o que é suficiente para reconhecimento de objetos, mas pode não ser para a extração de medidas geométricas que requeiram maior precisão. As técnicas de localização de subpixels são utilizadas para que se obtenha com maior precisão a localização de detalhes de uma imagem. Entre as aplicações estão o rastreamento em reconstruções tri-dimensionais, calibragem de câmera, reconstrução de imagens repartidas, localização precisa de elementos em uma imagem de satélite, entre outras.

O OpenCV implementa a função `cvFindCornerSubpix` que tem como um dos argumentos de entrada as localizações inteiras dos pixels obtidas com a função `cvGoodFeaturesToTrack`. Além disso, a busca por subpixels é iterativa e o critério de parada é estabelecido pelo usuário através de uma das funções do OpenCV.

4.3.4 Optical Flow

Este tipo de técnica possibilita a identificação de movimento entre sequências de frames sem que se conheça a priori o conteúdo destes. Tipicamente, o movimento em si indica que algo de interesse está acontecendo.

O OpenCV possui funções que implementam técnicas de detecção de movimento esparsas e densas. Algoritmos de natureza esparsa consideram algum conhecimento prévio sobre os pontos que se deseja rastrear, como por exemplo os *corners* descritos nas seções anteriores. Os algoritmos densos, por sua vez, associam um vetor de velocidade ou de deslocamento a cada pixel na imagem, sendo, portanto desnecessário o conhecimento prévio de pontos específicos da imagem. Para a maioria das aplicações práticas, entretanto, as técnicas densas possuem um custo de processamento muito alto, sendo preferíveis, portanto, as técnicas esparsas. A tabela 2 resume as funções de detecção de movimento presentes no OpenCV.

Tabela 2: Algoritmos de Optical Flow do OpenCV

Algortimo	Tipo	Comando
Lucas-Kanade	Esparso	`cvCalcOpticalFlowLK`
Lucas Kanade Piramidal	Esparso	`cvCalcOpticalFlowPyrLK`
Horn-Shunk	Denso	`cvCalcOpticalFLowHS`
Block Matching	Denso	`cvCalcOpticalFlowBM`

4.3.5 Mean-Shift e Camshift

Camshift (*Continuously Adaptive Mean-SHIFT*) é um algoritmo desenvolvido para o rastreamento de cor, possibilitando também o rastreamento de faces. É baseado numa técnica estatística onde se busca o pico entre distribuições de probabilidade em gradientes de densidade. Esta técnica é chamada de "média por deslocamento" (*mean shift*) e foi adaptada no Camshift para tratar a mudança dinâmica das distribuições de probabilidade das cores numa seqüência de vídeo. Pode ser usada no rastreamento de objetos e no rastreamento de faces, como descrito a seguir.

Para cada frame, a imagem (*raw*) é convertida para outra de distribuição de probabilidade de cor através de um modelo de histograma da cor da pele. O centro e o tamanho da face que se quer rastrear são encontrados através do CamShift operando na imagem de probabilidade de cores. O tamanho e a localização corrente da face são informados e usados para definir o tamanho e a localização da janela de busca da próxima imagem de vídeo.

A função `cvCamShift` chama o algoritmo CamShift para buscar o centro, o tamanho e a orientação do objeto sendo rastreado. A figura 23 mostra o programa

camshiftdemo.c em funcionamento. Este programa acompanha o pacote OpenCV, assim como várias outras demonstrações.

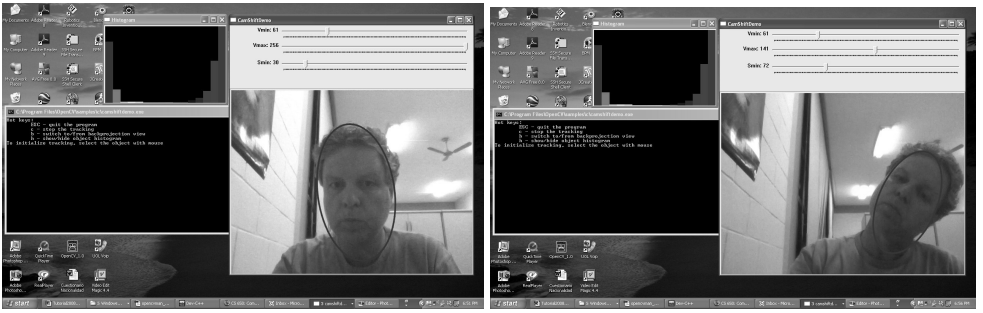


Figura 23: *camshiftdemo.c* em funcionamento (histograma, janela de opções e imagem capturada pela câmera e sendo rastreada, o que é demonstrado pela elipse ao redor da face).

4.3.6 Estimadores

Algoritmos deste tipo são capazes de estimar, por exemplo, a localização de uma pessoa ou objeto em movimento numa sequência de vídeo. Esta tarefa é dividida, basicamente, em duas fases: **predição**, baseada num conhecimento prévio de dados da imagem e **correção**, que usa novas medidas para apurar a predição realizada anteriormente.

A técnica mais conhecida para isto é o filtro de Kalman, implementado no OpenCV através de quatro funções, a saber:

- `cvCreateKalman` e `cvReleaseKalman`: servem para criar e destruir uma estrutura (struct) com os dados a serem processados pelo algoritmo.
- `cvKalmanPredict`: calcula as estimativas para o próximo passo
- `cvKalmanCorrect`: corrige as medidas estimadas

Além desta técnica, o OpenCV traz a implementação de uma alternativa, o **algoritmo de condensação**. Esta é um técnica mais sofisticada que se baseia no cálculo de múltiplas hipóteses para as estimativas a serem realizadas, ao contrário do filtro de Kalman. Maiores detalhes podem ser encontrados em [**9**].

5 Discussão Geral e Conclusões

Neste tutorial foram mostrados conceitos básicos de processamento de imagens e de visão computacional. Os exemplos descritos foram implementados utilizando a biblioteca openCV desenvolvida pela Intel. O openCV facilita a implementação de operadores simples até o

desenvolvimento de sistemas mais complexos na área de processamento de imagens e visão computacional.

6 Referências

[1] Gonzalez, R.C. e Woods, R.E. e Eddins, S.L., *Digital Image Processing using MATLAB,* Pearson, 2006.

[2] _____**,** *OpenCV website***,** http://sourceforge.net/projects/opencvlibrary**.**

[3] Open source computer vision library - reference manual, INTEL Corporation,1999-2001.

[4] Wilson, Gregory. Programmer´s Tool Chest – The OpenCV Library. Disponível em http://www.ddj.com/architect/184404319?pgno=1. Acessado em 28/08/2008.

[5] Velho, Luiz, Material de aula do curso de Image Processing and Computer Graphics, IMPA, http://www.visgraf.impa.br/Courses/ipcg.html. Acessado em 02/09/2008.

[6] Marr, D. E Hildreth, E. Theory of Edge Detection, Proc. Of The Royal Society of London, vol B207, pp. 187-217.

[7] Canny, J., A Computational Approach for Edge Detection, IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 8, no 6, pp. 679-698, 1986.

[8] Cândido, J., e Marengoni, M, Combining Information in a Bayesian Network for Face Detection, Brazilian Journal of Probability and Statistics, 2009 (to appear).

[9] Bradski, G. e Kaehler, A., Learning OpenCV, O'Reilly, 2008.

[10] Isard M. e Blake A., Condensation-conditional density propagation for visual tracking. International Journal in Computer Vision, IJCV 29(1):5-28, 1998.

[11] Kalman R. E., A new approach to linear filtering and prediction problems. Transactions of the ASME – Journal of Basic Engineering, 82:35-45, 1960.

7 Bibliografia

[a] Gonzalez, R.C. and Woods, R.E., Digital Image Processing, 3rd Edition, Pearson-Prentice-Hall, 2008.

[b] Pedrini, H. e Schwartz, W.R., Análise de Imagens Digitais – Princípios, Algoritmos e Aplicações, Thomson, 2006.

[c] Forsyth, D.A. and Ponce, J., Computer Vision – A Modern Approach, Prentice-Hall, 2003.

[d] Efford, N., Digital Image Processing – A Practical Introduction using Java, Addison-Wesley, 2000.

Anexo 1: Ajuste do ambiente do OpenCV

Windows

Ajustar a variável de sistema PATH para a biblioteca:

1) Em "Meu Computador" use o botão direito do mouse e selecione Propriedades.
2) Clique em "Avançado" e após no botão "Variáveis do Ambiente" na parte inferior da janela (no Vista siga o link "Advanced System Settings" antes deste passo).
3) Nas "Variáveis do usuário" está a PATH, acrescente o caminho para a biblioteca OpenCV (o padrão é: `C:\Arquivos de Programa\OpenCV\bin`).
4) Adicione o caminho para a biblioteca de interface (o padrão é `C:\Arquivos de Programa\OpenCV\otherlibs\highgui`). Lembre de separá-las com ";".

OBS: a PATH pode ser atualizada automaticamente na instalação, mas é preciso verificar.

Visual Studio 2005

1) Abrir o ambiente do Visual Studio 2005.
2) Abrir o arquivo *opencv.sln* na pasta *_make* da instalação (`C:\Arquivos de Programa\OpenCV\_make`). Isto carregará toda a solução do OpenCV no ambiente do Visual Studio 2005.
3) Selecionar a opção "Build" no menu e clicar em "*build-solution*" para compilar a biblioteca. Esta compilação pode levar algum tempo.

DevCPP

1) Abrir o Bloodshed Dev-C++ (*versão testada: 4.9.9.2*).
2) Para "linkar" os programas do Dev C++ com as bibliotecas do OpenCV:
    a. Clicar em "*Tools*" e selecionar "*Compiler Options*". O sistema deve abrir uma janela com a aba "*Compiler*" selecionada.
    b. Marcar o *box* "*Add these commands to the linker command line*" e copiar os seguintes comandos:

```
-lhighgui -lcv -lcxcore -lcvaux -lcvcam
```

3) Na aba **"***Directories***"**, em "*Binaries*" adicionar o caminho: `C:\Arquivos de Programa\OpenCV\bin`.
4) Em "*C Includes*" adicionar os caminhos:

```
C:\Arquivos de Programa\OpenCV\cxcore\include;C:\Arquivos
de Programa\OpenCV\cv\include;C:\Arquivos de
```

```
Programa\OpenCV\otherlibs\highgui;C:\Arquivos de
Programa\OpenCV\cvaux\include; C:\Arquivos de
Programa\OpenCV\otherlibs\cvcam\include
```

5) Em "*C++ Includes*" adicionar os mesmos caminhos acima.

6) Em "*Libraries*" adicionar o caminho:

```
C:\Arquivos de Programa\OpenCV\lib
```

Testar a instalação no Windows

1) Carregar no ambiente o arquivo: `C:\Arquivos de Programa\OpenCV\samples\c\contours.c`.

2) Criar um novo arquivo de programa em branco: `teste.cpp`.

3) Copiar e colar todo o arquivo `contours.c` para o arquivo em branco `teste.cpp`, salve o arquivo `teste.cpp`.

4) Compilar e executar o programa `teste.cpp`. O programa analisa uma imagem sintética com faces e extrai regiões diferentes da imagem. Pode-se alterar parâmetros ou linhas do código e ver os resultados. (Recompilar o programa após as alterações).

Linux e g++

1) Entrar com *login* de super-usuário.

2) Ir ao diretório onde foi feito o *download* do arquivo *tar.gz* do OpenCV e, na janela aberta, digitar os seguintes comandos:

```
tar –xzf opencv–0.9.7.tar.gz

cd opencv–0.9.7

./configure && make && make install
```

3) Fazer os ajustes para o G++:

   a. Ir ao diretório "*home*" e abrir o arquivo *.bashrc* (para o ambiente *bash*).

   b. Acrescentar a seguinte linha ao arquivo:

```
alias gcv="g++ –I/usr/local/include/opencv –lcv –
lcxcore –lcvaux –lhighgui"
```

Testar a instalação no Linux

1) Ir ao diretório onde está instalado o OpenCV.

2) Procurar o sub-diretório "*samples*" e entrar no sub-diretório "*c*".

3) Digitar a linha de comando abaixo que deve executar o programa *contours.c*:

```
gcv contours.c && ./a.out
```

Anexo 2: Códigos utilizando o OpenCV

Neste anexo serão apresentados alguns exemplos de implementação utilizando o openCV dos códigos utilizados ao longo deste artigo. Primeiramente será apresentado um código completo do openCV, nos demais exemplos serão apresentados apenas partes de código que ilustram a operação em questão.

Funções Básicas do openCV

```
1.  #include "cv.h" // inclui as funcoes definidas no OpenCV
2.  #include "highgui.h"   // inclui as funcoes definidas na biblioteca de interface grafica highGUI
3.  #include <stdio.h> // inclui as funcoes basicas da linguagem C para entrada/saida
4.  int main(){
5.      char name0[] = "imagens/exemplo.bmp";  //string com o caminho para o arquivo da imagem exemplo.bmp
6.      IplImage        *cvImagem,*cvImgCinza,*cvImgResultado, *cvImgSubtraida; // cria variaveis do tipo IplImage
7.      CvSize tamanhoImagem;     // cria uma estrutura do tipo CvSize que armazena o tamanho de uma imagem
8.      cvImagem = cvLoadImage( name0, -1 );      //carrega a imagem exemplo.bmp na estrutura cvImagem
9.      tamanhoImagem.width=cvImagem->width;  //copia para a estrutura CvSize o tamanho da imagem carregada
10.     tamanhoImagem.height=cvImagem->height;
11.     cvNamedWindow( "imagem Original", 1 );     // cria uma janela com o nome imagem Original
12.     cvShowImage( "imagem Original", cvImagem); // carrega na janela imagem Original a imagem lida do arquivo
13.     //cria tres imagens do tamanho da imagem carregada, tipo IPL_DEPTH_8U, (imagens de 8 bits), o ultimo valor indica apenas 1 banda.
14.     cvImgCinza=cvCreateImage(tamanhoImagem,   IPL_DEPTH_8U, 1);
15.     cvImgResultado=cvCreateImage(tamanhoImagem, IPL_DEPTH_8U, 1);
16.     cvImgSubtraida=cvCreateImage(tamanhoImagem, IPL_DEPTH_8U, 1);
17.     //converte a cvImagem para nivel de cinza em cvImagemCinza, CV_BGR2GRAY indica o tipo de conversao
18.     cvCvtColor(cvImagem, cvImgCinza, CV_BGR2GRAY);
19.     cvCopy(cvImgCinza,cvImgResultado,0);      //faz a copia de uma imagem sem modificacao alguma
20.     cvNamedWindow( "imagem Cinza", 1 );      //cria outra janela e apresenta a imagem em nivel de cinza
21.     cvShowImage( "imagem Cinza", cvImgResultado);
```

```
22.     cvNot( cvImgCinza, cvImgResultado ); //inverte a imagem em nivel de cinza e armazena em cvImagemResultado
23.     cvNamedWindow( "imagem Invertida", 1 );
24.     cvShowImage( "imagem Invertida", cvImgResultado);
25.     //"embassa" a cvImgCinza e salva em cvImagemResultado, o tipo de embassamento e o CV_BLUR, utilizando uma janela 3x3
26.     cvSmooth(cvImgCinza,cvImgResultado,CV_BLUR, 3, 3, 0, 0);
27.     cvNamedWindow( "imagem Embassada", 1 );
28.     cvShowImage( "imagem Embassada", cvImgResultado);
29.     // subtrai a imagem em nivel de cinza da imagem embassada e armazena o resultado na imagem subtraida, com mascara 0 (para imagem de 8 bits)
30.     cvSub(cvImgCinza, cvImgResultado, cvImgSubtraida, 0);
31.     cvNamedWindow( "imagem Subtraida", 1 );
32.     cvShowImage( "imagem Subtraida", cvImgSubtraida);
33.     //a diferenca pode ser pequena e negativa, o resultado e escalonado, 10 e o fator multiplicativo e 20 o valor somado. Diferenca(x,y)=diferenca(x,y)*10 + 20
34.     cvConvertScale( cvImgSubtraida, cvImgResultado, 10, 20 );
35.     cvNamedWindow( "imagem Subtraida Ajustada", 1 );
36.     cvShowImage("imagem Subtraida Ajustada",cvImgResultado);
37.     cvWaitKey( 0 ); //espera pela entrada do usuario para terminar o programa
38.     cvDestroyWindow( "imagem Cinza" );//fecha as janelas criadas
39.     cvDestroyWindow( "imagem Embassada" );
40.     cvDestroyWindow( "imagem Invertida" );
41.     cvDestroyWindow( "imagem Subtraida" );
42.     cvDestroyWindow( "imagem Original" );
43.     cvDestroyWindow( "imagem Subtraida Ajustada" );
44.     cvReleaseImage( &cvImagem );//faz a liberacao da memoria utilizada pela aplicacao
45.     cvReleaseImage( &cvImgCinza );
46.     cvReleaseImage( &cvImgResultado );
47.     cvReleaseImage( &cvImgSubtraida );
```

```
48.     return( 0 );    // encerra o programa}
```

Filtros Estatísticos em openCV

```
1.    // funcao para ler os valores da imagem que estao sob a mascara neste caso, especifico para uma mascara 3x3 esta funcao pode ser facilmente modificada para atender qualquer tamanho de mascara utilizando uma estrutura do tipo for
2.    void leMascara(int *mascara,IplImage *cvImagem,int i,int j){
3.      //define a estrutura scalar que armazena a informacao do pixel. A estrutura possui 4 valores double.Para imagens em nivel de cinza o valor esta na posicao 0 para imagens coloridas vai de 0, 1 e 2, para azul, verde e vermelho.
4.      CvScalar scalar;
5.      scalar=cvGetAt(cvImagem,i-1,j-1); //le a imagem na posicao i-1 e j-1 e armazena no scalar
6.      mascara[0]=int (scalar.val[0]);//faz  um cast para int e coloca na mascara, na posicao 0.
7.      scalar=cvGetAt(cvImagem,i-1,j);
8.      mascara[1]=int (scalar.val[0]);
9.      scalar=cvGetAt(cvImagem,i-1,j+1);
10.     mascara[2]=int (scalar.val[0]);
11.     scalar=cvGetAt(cvImagem,i,j-1);
12.     mascara[3]=int (scalar.val[0]);
13.     scalar=cvGetAt(cvImagem,i,j);
14.     mascara[4]=int (scalar.val[0]);
15.     scalar=cvGetAt(cvImagem,i,j+1);
16.     mascara[5]=int (scalar.val[0]);
17.     scalar=cvGetAt(cvImagem,i+1,j-1);
18.     mascara[6]=int (scalar.val[0]);
19.     scalar=cvGetAt(cvImagem,i+1,j);
20.     mascara[7]=int (scalar.val[0]);
21.     scalar=cvGetAt(cvImagem,i+1,j+1);
22.     mascara[8]=int (scalar.val[0]);
```

```
23.   }//end le mascara
24.   void ordenaMascara(int *mascara){ // um simples "insertion sort" para ordenar os valores da mascara
25.     int i,j, temp;
26.        for(i=1; i<9; i++){
27.           j=i;
28.           while((mascara[j]<mascara[j-1])&&(j>0)){
29.               temp=mascara[j-1];
30.               mascara[j-1]=mascara[j];
31.               mascara[j]=temp;
32.               j--;
33.           }  } }//end ordena mascara
34.   // determina o valor mais frequente na mascara se dois valores aparecerem com a mesma frequencia retorna o primeiro
35.   int encontraModa(int *mascara){
36.       int i, j, contador, maximo, valor;
37.       i=0;     contador=0;      maximo=1;
38.       valor=mascara[4];
39.       while(i<9){
40.           for(j=1; j<9; j++){
41.               if(mascara[i]==mascara[j]){    contador++;    } }//for
42.           if(contador>maximo){
43.              maximo=contador;
44.              contador=0;
45.              valor=mascara[i]; }
46.           else{  contador=0;  }
47.           i++; }//while
48.       return valor;  }//encontraModa
49.   //trecho do programa principal - cria a mascara 3x3
```

```
50.       mascara=(int *) malloc(sizeof(int)*9);       //cria um vetor do tipo int de tamanho 9 para uma mascara 3x3
51.   //trecho do programa principal - faz os filtros
52.       //faz o loop na imagem de entrada para determinar cada filtro. As bordas da imagem nao sao consideradas neste caso
53.   for(i=1; i<cvImagem->width-1; i++){
54.     for(j=1;j<cvImagem->height-1;j++){
55.                 leMascara(mascara, cvImgCinza, j, i);
56.                 ordenaMascara(mascara);
57.   scalar=cvScalar(double(mascara[4]),0.0,0.0,0.0);     //cria um
58.                 //filtro mediana: inseri o scalar na posicao i,j, tende a eliminar o ruido, nao inseri novos valores na
59.                 cvSetAt(cvImgMediana, scalar, j, i);
60.   scalar=cvScalar(double(mascara[0]),0.0,0.0,0.0);      //filtro
61.                 cvSetAt(cvImgMinima, scalar, j, i);
62.   scalar=cvScalar(double(mascara[8]),0.0,0.0,0.0);      //filtro
63.                 cvSetAt(cvImgMaxima, scalar, j, i);
64.                 pixel=encontraModa(mascara);
    //filtro da moda: tende a homogeneizar a imagem
65.                 scalar=cvScalar(double(pixel), 0.0, 0.0, 0.0);
    //insere a moda na imagem filtrada
66.                 cvSetAt(cvImgModa, scalar, j, i); } }//for
```

Função de Equalização de Histograma e de Binarização de Imagens

```
1.   //Colocar no programa principal
2.   //cria um histograma com o numero de "bins" definido em hist_size, para valores dentro de "ranges"
3.   imgHist=cvCreateHist(1,&hist_size,CV_HIST_ARRAY,ranges,1);
4.   cvCvtColor(cvImagem, cvImgCinza, CV_BGR2GRAY);   //converte a imagem original para nivel de cinza
5.   //os paramentros sao: imagem original, imagem binaria, valor de corte (84), valor maximo na imagem (255) e
6.   cvThreshold(cvImgCinza,cvImgBinaria,84,255,CV_THRESH_BINARY);
7.   //Para equalização de histogramas simplesmente acrescente
8.   cvEqualizeHist( cvImagemGray, cvImagemResultado );
```

Funções de Segmentação

1.	//Colocar no programa principal o código abaixo
2.	//void cvSobel(const CvArr* src, CvArr* dst, int xorder, int yorder, int aperture_size=3); aplica o operador de Sobel na imagem de entrada (cvImgCinza) e o resultado na imagem de saida de 16 bits (cvImgTemp1), os dois parametros seguintes definem a direcao da mascara a ser utilizada, o 1 indica a direcao X, o 0 indica que nao e na direcao Y e o 3 indica o tamanho da mascara (3x3)
3.	cvSobel(cvImgCinza, cvImgTemp1, 1, 0, 3);
4.	// transforma a imagem de 16 bits (cvImgTemp1) em uma equivalente de 8 bits (cvImgSobel) o valor 1 indica que nao ha mudanca de escala e o 0 que nao existe translacao de intensidade esta funcao e util, por exemplo para visualizar diferencas entre imagens
5.	cvConvertScale(cvImgTemp1,cvImgSobel, 1, 0);
6.	cvSobel(cvImgCinza, cvImgTemp1, 0, 1, 3); //aplica o operador Sobel na direcao de Y
7.	cvConvertScale(cvImgTemp1, cvImgTemp2, 1, 0);
8.	// combina as imagens dos operadores Sobel em uma unica usando o operador logico OU por elemento. O ultimo parametro pode ser uma mascara com valores a serem alterados na imagem de saida, ou NULL para repetir a imagem de entrada.
9.	cvOr(cvImgSobel,cvImgTemp2,cvImgSobel,NULL);
10.	// aplica o operador de Canny na imagem de entrada, os parâmetros sao o valor de corte baixo (84), o valor de corte alto (128) e o tamanho da mascara (3x3). Note que a mascara deve ter um tamanho >= a 6*DesvioPadrao, nem sempre possivel
11.	cvCanny(cvImgCinza, cvImgCanny, 84, 128, 3);
12.	// cria uma imagem binaria com valor de corte = 128
13.	cvThreshold(cvImgCinza,cvImgCorte,128,255, CV_THRESH_BINARY);

Filtros Espaciais

1.	//Colocar o codigo abaixo no programa principal
2.	CvMat *filtro; // define um filtro usando a estrutura CvMat
3.	int lado = 3; //define o tamanho da mascara a ser utilizada
4.	int soma = 1; //define a soma dos pesos de cada mascara no caso do filtro passa alta, como a soma e zero, utilizamos 1.
5.	//define dados para um filtro. Kernel e basicamente um detector de bordas verticais, kernel1 e uma gaussiana do tipo passa baixa e kernel2 e um filtro passa alta, que realca as bordas. Kernel e kernel1 sao mascaras 5x5 e kernel2 e 3x3

```
6.  double kernel[] = { 0, -1, 0, 1, 0,
                       -1, -2, 0, 2, 1,
                       -1, -2, 1, 2, 1,
                       -1, -2, 0, 2, 1,
                        0, -1, 0, 1, 0};
7.  double kernel1[] = { 1,  4, 6, 4, 1,
                         4, 16,24,16, 4,
                         6, 24,36,24, 6,
                         4, 16,24,16, 4,
                         1,  4, 6, 4, 1,};
8.  double kernel2[] = { -1,  -1, -1,
                         -1,   8, -1,
                         -1,  -1, -1};
9.  for(i=0; i<lado*lado; i++){ //define a mascara com os valores finais
10.     kernel2[i]=(1./soma)*kernel2[i];  }//for
11. //Cria o filtro, neste caso define o header da estrutura com tamanho lado x lado e utilizando 64 bits
12. filtro = cvCreateMatHeader( lado, lado, CV_64FC1 );
13. //copia o kernel para o campo "data" da estrutura do filtro o ultimo parametro indica quantos bytes sao necessarios para uma linha da mascara
14. cvSetData( filtro, kernel2, lado*8 );
15. //executa a convolucao da imagem de entrada (cvImgCinza) pela mascara definida em filtro e salva o resultado na imagem de saida (cvImgFiltrada), que pode ser a mesma. O ultimo parametro define o ponto que deve ser filtrado na mascara, (-1,-1) indica que deve ser considerado o centro da mascara.
16. cvFilter2D(cvImgCinza,cvImgFiltrada,filtro,cvPoint(-1,-1) );
```

Função de Crescimento de Região

```
1.  //funcao que faz a operacao de crescimento de regiao
2.  void crescimentoDeRegiao(IplImage *binaria, IplImage *resultado, int i, int j){
3.      int flag, posicao, row, col, p, q;
4.      int lista[2][50000]; //usado como pilha, mais eficiente se implementado com ponteiros
5.      double pixel;
6.      CvScalar scalar;
7.      CvSize tamanhoImagem;
8.      tamanhoImagem.width=binaria->width;
9.      tamanhoImagem.height=binaria->height;
10.     flag=1;//define a variavel de controle do loop principal
```

```
11.         posicao=0;   //inicializa  a  pilha
12.         lista[0][posicao]=i;
13.         lista[1][posicao]=j;
14.         while(flag==1){       //inicia o loop principal
15.               row=lista[0][posicao];  //remove  o  elemento  do
topo
16.               col=lista[1][posicao];
17.               posicao--;
18.               //verifica vizinhanca 8 na imagem binaria
19.               pixel=0;
20.               scalar=cvGetAt(binaria,row-1,col-1);
21.               pixel+=scalar.val[0];
22.               scalar=cvGetAt(binaria,row-1,col);
23.               pixel+=scalar.val[0];
24.               scalar=cvGetAt(binaria,row-1,col+1);
25.               pixel+=scalar.val[0];
26.               scalar=cvGetAt(binaria,row,col-1);
27.               pixel+=scalar.val[0];
28.               scalar=cvGetAt(binaria,row,col);
29.               pixel+=scalar.val[0];
30.               scalar=cvGetAt(binaria,row,col+1);
31.               pixel+=scalar.val[0];
32.               scalar=cvGetAt(binaria,row+1,col-1);
33.               pixel+=scalar.val[0];
34.               scalar=cvGetAt(binaria,row+1,col);
35.               pixel+=scalar.val[0];
36.               scalar=cvGetAt(binaria,row+1,col+1);
37.               pixel+=scalar.val[0];
38.               // testa se os elementos sao zero na vizinhanca 8
se sim coloca na imagem de saida o valor 255 (branco)
```

```
39.             if(pixel==0){
40.                 ((uchar*)(resultado->imageData + resultado->widthStep*row))[col] = ( char ) 255;
41.                 // verifica com vizinhanca 8 as novas sementes para inserir na pilha. Testa primeiro as bordas
42.                 for(p=-1;p<2;p++){
43.                     for(q=-1;q<2;q++){
44.                         if(((row+p)>0)&&((row+p) < tamanhoImagem.height - 1)){
45.                             if(((col+q)>0)&&((col+q) <tamanhoImagem.width-1)){
46.                                 scalar=cvGetAt(resultado, row+p,col+q);  // e verifica se o ponto ja nao foi processado
47.                                 if(scalar.val[0]==0){  //  inseri uma nova semente na pilha
48.                                     posicao++;
49.                                     lista[0][posicao]=row+p;
50.                                     lista[1][posicao]=col+q;}}}}}
51.             if(posicao==0){  //  se  a  pilha  vazia  termina  o loop
52.                     flag=0; }    }     }
53.   //Adicionar as linhas abaixo no programa principal
54.   //cria a imagem binaria que vai ser utilizada no processo de crescimento de regiao
55.   cvThreshold(cvImgCinza,cvImgCorte,128,255,CV_THRESH_BINARY);
56.   // cria a imagem que vai armazenar o crescimento de regiao e inicializa a imagem com 0s
57.   cvImgRegiao=cvCreateImage(tamanhoImagem, IPL_DEPTH_8U, 1);
58.   for ( i = 0; i < tamanhoImagem.width; i++ ){
59.     for ( j = 0; j < tamanhoImagem.height; j++ ){
60.       ((uchar*)(cvImgRegiao->imageData + cvImgRegiao->widthStep*j))[i] = ( char ) 0; }  }
61.   // chama o crescimento de regiao com a posicao da semente
62.   crescimentoDeRegiao(cvImgCorte, cvImgRegiao, 55, 63);
```