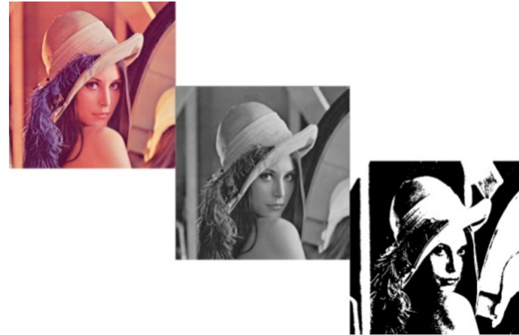
OpenCV



http://opencv.willowgarage.com/wiki/

Introdução a OpenCV

Geralmente, quando iniciamos estudos numa determinada linguagem de programação é comum que o iniciante implemente um pequeno programa de fácil entendimento chamado *HelloWorld*. Com a biblioteca OpenCV não é diferente, no entanto, ao invés de imprimir na tela a frase “Hello World!”, imprimiremos uma imagem de nosso interesse.

A OpenCV possui cinco ficheiros include descritos a seguir:

- *cv.h* - contém o básico de processamento de imagem e algoritmos de visão computacional;
- *ml.h* - é a biblioteca de aprendizagem de máquina, que inclui muitos classificadores estatísticos;
- *highgui.h* – contém rotinas I/O e funções para armazenar e carregar imagens e vídeos;
- *cxcore.h* – contém as estruturas de dados básicos;
- *cvaux.h* – contém as áreas de reconhecimento facial e algoritmos experimentais.

Para nós será necessário apenas as headers *cv.h* e *highgui.h,* por tanto, não esqueça de incluir as seguintes linhas de comando no inicio de cada programa:

```
#include "highgui.h"
#include "cv.h"
```

Exercício 1.1: HelloWorld!

Neste primeiro exercício, iremos implementar o código HelloWorld, como foi dito anteriormente, que permite a visualização da imagem na tela.

O primeiro passo consiste em carregar a imagem de interesse para a memória, retornando o ponteiro do tipo IplImage, (struct pré definida pela OpenCV), que aponta para a imagem carregada, conforme se segue:

```
IplImage * img = cvLoadImage(const char* filename, int iscolor =
CV_LOAD_IMAGE_COLOR);
```

A função *cvLoadImagem* recebe como argumento o nome ou path da imagem e um valor inteiro, correspondente ao número de canais que a imagem irá ter:

- Se valor inteiro > 0, a imagem carregada é forçada a ser uma imagem colorida de 3 canais;
- Se valor inteiro = 0, a imagem carregada é forçada a ser em tons de cinza;
- Se valor inteiro < 0, a imagem será carregada com o seu próprio número de canais.

Vale ressaltar que os formatos de imagem suportados atualmente são: BMP, DIB, JPEG, JPE, PNG, PBM, PGM, PPM, SR, RAS, TIFF, TIF.

Exemplo 1:

```
IplImage * img = cvLoadImage("lena.jpg", 1);
```

Em seguida, iremos criar uma janela, na qual será exibida a imagem:

```
cvNamedWindow(const char* name, int flags);
```

O primeiro argumento recebe o nome da janela de nossa escolha, o segundo é a flag relativa à forma como a janela se comporta, neste caso "*autosize*", ou seja, adéqua-se ao tamanho da imagem. Falta agora associar a imagem à janela, que será feito através da seguinte função:

```
cvShowImage(const char* name, const CvArr* image);
```

Os parâmetros são, respectivamente, o nome da janela onde quero exibir a imagem, e o ponteiro que aponta para onde está a imagem.

Exemplo 2:

```
cvNamedWindow ( "Imagem", CV_WINDOW_AUTOSIZE);
cvShowImage("Imagem", img);
```

Finalizamos o programa com a função a seguir que permite que todas as janelas abertas pelo programa sejam fechadas ao pressionarmos uma tecla:

```
cvWaitKey();
```

A partir das funções acima, escreva o código que abra a imagem **lena.jpg** que se encontra na pasta "**03 - opencv**".

Experimente manipular a função *cvLoadImage()* de forma que a imagem seja carregada em tons de cinza.

Exercício 1.2: Convertendo escala de cores

Alguns algoritmos de PI exigem que a imagem antes de ser trabalhada esteja em uma determinada escala de cores. Para isto, é criada uma matriz que aloca os dados da nova

imagem com o tamanho da original e com o número de canais que se deseja ter, a partir da função a seguir:

```
IplImage* cinza = cvCreateImage( cvGetSize(img), int depth, int channels);
```

Os parâmetros são os seguintes:

- Tamanho da imagem: que é obtida através da função *cvGetSize()* que extrai o tamanho da imagem original;
- Depth (profundidade): que nada mais é que o número de bits por pixel;
- Número de canais por pixel.

Exemplo 3: `IplImage* cinza = cvCreateImage(cvGetSize(img),IPL_DEPTH_8U, 1);`

A variável "cinza" recebe o resultado da conversão de cor, que é realizada da seguinte forma:

```
cvCvtColor(const CvArr* src, CvArr* dst, int code);
```

Onde os parâmetros são o ponteiro para a imagem original, ponteiro para a imagem que conterá o resultado da conversão e o tipo de conversão, respectivamente. O ultimo argumento é um código pré determinado pela OpenCV o qual carrega em seu nome a escala de cor original e a resultante, como por exemplo, CV_RGB2GRAY que converte de RGB para níveis de cinza e a CV_GRAY2RGB que realiza o inverso.

Exemplo 4: `cvCvtColor(imagem, cinza, CV_RGB2GRAY);`

Na pasta "**03 - opencv**" existe um código de nome **"imagem_cinza.cpp"** que foi desenvolvido a partir do exercício anterior utilizando as novas funções. Compile e execute o programa e observe o que ocorre. Existe alguma diferença entre esta resultante e a imagem em tons de cinza obtida no exercício anterior?

Exercício 1.3: Binarização

Em geral, algoritmos de PI tem como uma de suas etapas a binarização da imagem. Para isto, a OpenCV oferece a seguinte função:

```
cvThreshold(const CvArr* src, CvArr* dst, double threshold, double maxValue,
int thresholdType);
```

Esta função converte uma imagem em níveis de cinza numa imagem binária e os parâmetros são:

- src: é o ponteiro que aponta para a matriz criada a partir da função *cvCreateImage()* que contem os dados da imagem em escala de cinza;
- dst: é o ponteiro que aponta para a matriz de destino onde serão alocadas as informações da imagem depois de ser binarizada. Caso a imagem em níveis de cinza não seja mais utilizada, o ponteiro dst pode apontar para a mesma matriz apontada por src;
- threshold: é a limiar ou valor de corte;
- maxValue: é o máximo valor que o corte pode assumir;

- thresholdType: neste parâmetro é onde o programador determina qual o tipo de binarização deve ser aplicada à imagem. Normalmente, utiliza-se o tipo mais básico que é o CV_THRESH_BINARY, mas a função também oferece CV_THRESH_BINARY_INV, CV_THRESH_TRUNC, CV_THRESH_TOZERO e CV_THRESH_TOZER_INV.

Exemplo 5: `cvThreshold(binaria,binaria,threshold,maxValue,thresholdType);`

Ainda na pasta "**03 - opencv**" encontra-se um código de nome "**binariza.cpp**". A partir do programa, observe a imagem resultante e, em seguida, altere na parte superior do código o valor da variável *threshold* para 50. O que ocorre com a saída?

Em seguida, edite e compile o seu programa utilizando os outros tipos de binarização alterando a variável *thresholdType* para: CV_THRESH_BINARY_INV, CV_THRESH_TRUNC, CV_THRESH_TOZERO e CV_THRESH_TOZER_INV . Você é capaz de descrever a diferença entre eles? Experimente variar o valor da variável threshold observando a mudança na saída.

Exercício 1.4: Detecção de Contornos

O algoritmo de detecção de contornos tem como objetivo contornar objetos de uma imagem. Esta implementação tem grande aplicação na área de PI, como por exemplo, cálculo da área dos objetos detectados. A seguir serão descritas as funções necessárias para a construção do código.

Primeiramente, é preciso reservar um espaço da memória vazio destinado a alocar os resultados retornados pelas funções que serão utilizadas:

```
CvMemStorage* storage = cvCreateMemStorage(int blockSize=0);
```

Exemplo 6: `g_storage = cvCreateMemStorage(0);`

Caso seja necessário, há uma forma de limpar o espaço de memória já alocado:

```
cvClearMemStorage(CvMemStorage* storage);
```

Exemplo 7: `cvClearMemStorage( g_storage );`

Antes de seguir, é preciso definir o que é contorno, que nada mais é que uma lista de pontos (pixels) que representam, de uma maneira ou de outra, uma curva de uma imagem. Esta representação varia de acordo com as circunstâncias, já que há muitas maneiras de representar uma curva. Na OpenCV os contornos são representados por seqüências em que cada entrada na seqüência codifica a informação sobre a localização do próximo ponto na curva. Para isso, existe a função a seguir:

```
cvFindContours(CvArr* image, CvMemStorage* storage, CvSeq** first_cont);
```

Os parâmetros são:

- image: o ponteiro que aponta para a imagem que deve estar binarizada, caso não esteja, a função irá usar os valores diferentes de zero como 1s e assim "fingir" que a imagem está binarizada ;

- storage: o ponteiro para o espaço de memória reservado onde será guardado as seqüências de pixels de contorno;
- first_cont: este parâmetro irá retornar um ponteiro que aponta para o primeiro pixel de contorno.

Como a função utiliza um ponteiro de retorno (first_cont), então precisamos criá-lo antes de chamar a função:

```
CvSeq* first_cont = 0;
```

Exemplo 8:
```
CvSeq* contours = 0;
cvFindContours( g_gray, g_storage, &contours );
```

Finalizado todo processo, então podemos exibir para o usuário o resultado utilizando a função que desenha os contornos:

```
cvDrawContours(CvArr *img, CvSeq* contour, CvScalar external_color, CvScalar
hole_color, int thickness=1);
```

Os parâmetros são:

- img: ponteiro para a imagem onde os contornos serão desenhados;
- contour: ponteiro para o primeiro pixel do contorno;
- external_color: cor dos contornos externos;
- hole_color: cor dos contornos internos;
- thickness: este parâmetro sinaliza para a função se ela deve desenhar contorno interno e externo se thickness ≥ 0 ou preencher a área delimitada pelos contornos se thickness < 0.

Exemplo 9:
```
cvDrawContours(g_gray,contours,cvScalarAll(255),cvScalarAll(255),1);
```

Agora que já sabemos algumas funções básicas da biblioteca OpenCV, já somos capazes de entender o código de nome “**detecontornos.cpp**”, para isto abra-o na pasta “**03 - opencv**” e execute o programa observando o resultado.

Para um maior aproveitamento do curso é essencial que o aluno desenvolva seus próprios códigos utilizando as funções aqui apresentadas.