Tutorial de OpenCV para Tótós

Alexandra Ribeiro e Miguel Figueiredo
Undergraduate Students
Instituto Superior Técnico, Universidade Técnica de Lisboa, Portugal

Objectivo:

Este tutorial tem como objectivo "colocar em papel" as nossas descobertas mais recentes relativamente à biblioteca OpenCV, que iremos aplicar no nosso trabalho.

OpenCV significa *Intel Open Source Computer Vision Library* e consiste numa colecção de funções C e algumas classes C++ que implementam alguns algoritmos bastante conhecidos, relativos ao processamento de imagem.

Aqui encontrar-se-ão os passos necessários para conseguir aplicações básicas de processamento de imagem e o *source code* das mesmas .

Pode efectuar o download desta biblioteca em:

- http://sourceforge.net/project/showfiles.php?group_id=22870

Para os exemplos demonstrados abaixo, foi utilizada a versão Beta 5 desta biblioteca, em framework .NET, Visual Studio 2003 e utilizando DirectX 9.0.

Primeiros passos – Configuração do projecto:

Para realizar um primeiro programa simples, utilizando o OpenCV, necessita de seguir os seguintes passos:

1. Abra o Visual Studio e abra um novo projecto C++, do tipo "Console Application (.NET)". Bastará para isto seguir, no menu principal, o caminho "File → New → Project" e clicar no tipo de aplicação pretendida, existente na pasta "Visual C++ Projects".

2. Antes de começar a implementação do programa, terá de acrescentar nas propriedades do projecto, algumas bibliotecas necessárias para que sejam reconhecidas as funções de OpenCV que iremos utilizar. Para isso, aceda ao menu das propriedades, seguindo o caminho, no menu principal: "Project → project_name Properties".
   Aí, terá de seguir os seguintes passos:

- Adicionar ao projecto as directorias referentes ao OpenCV (ver figura 1), que estarão no directório onde instalou esta biblioteca:

  - "(path referente ao directório do OpenCV)\cvaux\include"
  - "(path referente ao directório do OpenCV)\cvaux\otherlibs\highgui"
  - "(path referente ao directório do OpenCV)\cxcore\include"
  - "(path referente ao directório do OpenCV)\cv\include"

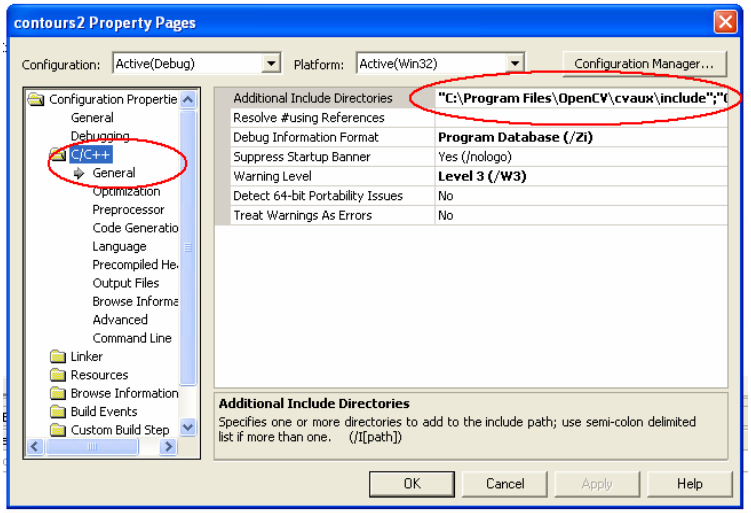


Figura 1 - "Includes" necessários

- Adicionar directorias adicionais, referentes ao OpenCV (ver figura 2):

  - "(path referente ao directório do OpenCV \bin"
  - "(path referente ao directório do OpenCV)\lib"

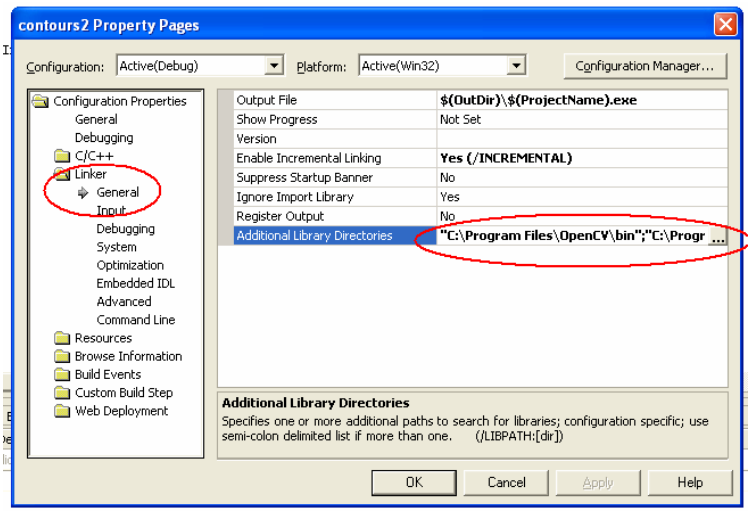
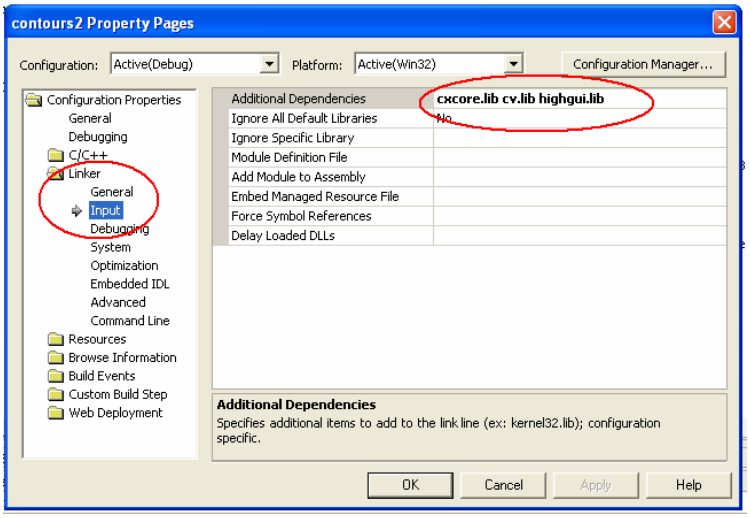


Figura 2 - Additional Libraries

- Adicionar bibliotecas:
  - cxcore.lib
  - cv.lib
  - highgui.lib



Neste momento o projecto está configurado para poder utilizar as funções que o OpenCV oferece. Passemos então aos exemplos.

Exemplo 1 – Load de uma imagem e respectiva visualização:

Neste primeiro exemplo, iremos apenas realizar uma aplicação que permite ao utilizador visualizar uma imagem através da aplicação.

Para isso, o primeiro passo consiste em fazer o load da imagem, guardando-a numa variável do tipo IplImage, conforme se segue:

```
image = cvLoadImage("./images/hand.jpg",-1);
```

Esta função efectua o load da imagem, recebendo como argumentos o path da imagem e um valor inteiro, correspondente ao número de canais que a imagem irá ter:

- >1 - a imagem resultante irá ter sempre 3 canais
- 1 - a imagem resultante irá ter sempre 1 canal
- <1 – a imagem resultante irá ter número de canais dependentes do ficheiro, ou seja, consoante o número de canais no ficheiro "loaded".

Retorna o apontador para a imagem.
Assim, a declaração da variável onde irá ser depositada esta informação será algo como:

```
IplImage *image = 0;
```

De realçar que os formatos de imagem suportados actualmente são:

Windows bitmaps - BMP, DIB; JPEG files - JPEG, JPG, JPE; Portable Network Graphics - PNG; Portable image format - PBM, PGM, PPM; Sun rasters - SR, RAS; TIFF files - TIFF, TIF.

De seguida, iremos criar uma janela, na qual será apresentada a imagem, para isso:

```
cvNamedWindow("contours",CV_WINDOW_AUTOSIZE);
```

O primeiro argumento é o nome da janela, o segundo é a flag relativa à forma como a janela se comporta,neste caso é "*autosize*", ou seja adequa-se ao tamanho da imagem. Falta agora visualizar a imagem o que é feito através da função:

```
cvShowImage("contours",image);
```

Isto é feito depois de testar se o load da imagem retornou uma imagem válida e dentro de um ciclo "*for*", que só será terminado caso a tecla "Esc" seja premida, o que permite ao utilizador sair da aplicação.

Assim, este pedaço de código terá o seguinte aspecto:

```
if(image){
      for(;;){
            cvShowImage("contours",image);

                  if(cvWaitKey(10) >= 0){
                        break;
                  }
      }
}
```

Por último é necessário libertar os recursos utilizados, neste caso, a imagem e a janela onde foi apresentada.

```
cvReleaseImage(&image);
cvDestroyWindow("contours");
```

Source Code – Exemplo 1:
http://web.tagus.ist.utl.pt/~alexandra.ribeiro/TFC/OpenCV/exemplo1-tutorialOpenCV.rar

Exemplo 2 – aplicação de um algoritmo a uma imagem (Canny):

Neste exemplo, iremos aplicar à imagem do exemplo anterior um algoritmo de processamento de imagem, conhecido como Algoritmo de Canny, que nos permite obter a visualização dos contornos da mesma. Por fim, iremos também salvar a imagem resultante da aplicação do algoritmo para um ficheiro.

Depois de ter feito o load da imagem à qual queremos aplicar o algoritmo, temos de transformá-la numa imagem do tipo grayscale (excepto se a imagem já for deste tipo), dado que este algoritmo é aplicado a imagens que possuam apenas um canal. Para isto, é criada uma nova imagem, com o tamanho da imagem original e apenas com um canal (não esquecer de inicializar a respectiva variável, tal como foi feito com a imagem original).

```
image_gray=
      cvCreateImage(cvSize(image->width,image->height),IPL_DEPTH_8U,1);
```

Os parâmetros são os seguintes:

- tamanho da imagem (com o tamanho da imagem original)
- depth (número de bits por pixel), neste caso unsigned integers, de 8 bits.
- número de canais por pixel

Será em image_gray que depositaremos o resultado da conversão de cor, do modelo RGB (com 3 canais) para grayscale, da seguinte forma:

```
      cvCvtColor(image, image_gray, CV_RGB2GRAY);
```

onde os parametros são a imagem original, a imagem que conterá o resultado da conversão e o tipo de conversão, respectivamente.

Depois de obtida a imagem, com apenas um canal, podemos aplicar então o algoritmo de Canny, para a detecção de contornos. Para esse efeito, é criada uma nova imagem, onde irá ser depositado o resultado da aplicação do algoritmo de Canny.

```
image_contours =
cvCreateImage(cvSize(image_gray->width,image_gray->height),
IPL_DEPTH_8U,image_gray->nChannels);
```

Aplicaremos agora o algoritmo, passando como parâmetros a imagem grayscale, a imagem que guardará os contornos encontrados, o valor do primeiro threshold, o valor do segundo threshold e o apertureSize, que tem, por omissão, o valor 3.

```
cvCanny(image_gray, image_contours, 0,255, 3);
```

É então mostrado o resultado, que se encontra armazenado na variável image_contours. Por fim, é feita a libertação de recursos, tal como no exemplo anterior.

Source Code – Exemplo 2:
http://web.tagus.ist.utl.pt/~alexandra.ribeiro/TFC/OpenCV/exemplo2-tutorialOpenCV.rar

Exemplo 3 – Capturando Video:

Para além de lidar com imagens, esta biblioteca permite aplicar o processamento às imagens de um video.

Para que isto aconteça,temos, numa primeira fase, de capturar o video.
Isto é possível através da seguinte função:

```
CvCapture* capture = 0;
capture = cvCaptureFromCAM(0);
```

O parâmetro desta função refere-se ao index da câmara que está a ser utilizada para a captura de imagem (valores de 0 a 10), ou seja, é possível capturar imagem de várias câmaras, chamando a função com os índices respectivos.

Neste caso, em particular, estamos a capturar video a partir de uma WebCam, mas seria possível também capturá-lo partindo de um ficheiro .AVI, através da função cvCaptureFromAVI(nome_do_ficheiro).

Depois deste passo e após ter inicializado uma janela, para visualizar o resultado, podemos aplicar uma função a cada frame do video, que será visualizada ao longo do mesmo.

Assim, é necessário efectuar o grab de cada frame. Para que isto aconteça é necessário recorrer a duas funções da biblioteca OpenCV.

```
if(!cvGrabFrame(capture)){
	break;
}
frame = cvRetrieveFrame(capture);
if(!frame){
	break;
}
```

A primeira função (cvGrabFrame) faz o grab da frame, armazenando-a internamente, tendo apenas como objectivo o armazenamento rápido da respectiva frame, facto importante na sincronização entre câmaras, caso existam mais do que uma.
Para aceder a esta frame é necessário recorrer à segunda função, cvRetrieveFrame, que retorna então a frame, sob a forma de uma IplImage.

Seguidamente é criada uma cópia da imagem, para não alterar directamente a imagem capturada.

```
	if(!frame_copy){
	frame_copy = cvCreateImage(cvSize(frame->width, frame->height),
	IPL_DEPTH_8U, frame->nChannels);
	}
```

```
Este teste verifica se a origem da imagem a copiar tem a sua origem
(como é normal) no canto superior esquerdo. Caso isso aconteça, a
imagem pode ser copiada directamente. Caso contrário é aplicado uma
rotação à imagem, em torno do eixo dos XX, de forma a ser copiada
corretamente.
```

```
if(frame->origin == IPL_ORIGIN_TL){
      cvCopy(frame, frame_copy, 0);
}
else{
      cvFlip(frame, frame_copy, 0);
}
```

Por último poderíamos chamar uma função de aplicação de filtros, por exemplo. Seria algo como:

```
applyFilterToFrame(frame_copy);
```

Não esquecer de declarar os cabeçalhos desta função, no início do ficheiro!

Source Code – Exemplo 3:
http://web.tagus.ist.utl.pt/~alexandra.ribeiro/TFC/OpenCV/exemplo3-tutorialOpenCV.rar

Bibliografia:

[1]– OpenCVDocumentation
http://www.cs.bham.ac.uk/resources/courses/robotics/doc/opencvdocs/

[2] – OpenCV Tutorial, by R. Laganiere
http://www.site.uottawa.ca/~laganier/tutorial/opencv+directshow/