COMPARANDO OPENCV 1.X E 2.X

Manoel Vieira Coelho Neto*

*SQS 203 Bloco J
Brasília, DF, Brasil

Email: vieiranetoc@gmail.com

Abstract— This article aims to show difference between OpenCV Version 1.x and 2.x, comparing differents algorithms to do same transformation on a black image with 8 bits pixel and only one channel. All the source files were writthen in C++ language and compiled with g++ compiler using OpenCV directives and libraries linker. The first algorithm uses OpenCV 1.x then funcions implemented there are pure C functions, which have 'cv' before function name, while in the second, OpenCV 2.x is used then functions have a namespace 'cv::'. Comparisons are made using Gaussian distributition, once time libraries uses CPU clock and because of this is highly innacurate so when modeling as a normal distributition we can disconsider random delays caused by system processes and background applications.

Keywords— OpenCV, Computational Vision, Image Transformation

Resumo— Esse artigo objetiva mostrar a diferença entre a Versão 1.x e 2.x do OpenCV comparando diferentes algoritmos para fazer a mesma transformação numa imagem preta de pixeis de 8 bits e apenas um canal de cor. Todos os arquivos fontes foram escritos em linguagem C++ e compilados com o compilador g++ usando as diretivas do OpenCV e seus linkadores de biblioteca. O primeiro algoritmo usa OpenCV 1.x, então as funções implementadas são funções C puras as funções C puras as quais têm 'cv' antes do nome da função, enquanto o segundo usa OpenCV 2.x assim as funções usadas são pertencentes ao namespace 'cv::'. Comparações são feitas usando distribuição Gaussiana, uma vez que as blibliotecas de tempo usam o clock da CPU e por causa disso são imprecisas. Ao modelar como uma distribuição normal, podemos desconsiderar atrasos aleatórios causados por processos do sistema e aplicações em segundo plano.

Palavras-chave— OpenCV, Visão Computacional, Transformação de Imagem

1 Introdução

Este artigo consiste em uma análise das versões do OpenCV e suas mudanças nas operações básicas no decorrer do tempo para que se possa entender as diferenças entre as elas. A versão 1.x usa funções de C puro usando como prefixo o termo 'cv' -como para criar uma imagem a partir de um ponteiro de imagem IPL usamos cvCreateImage()- e requer um tratamento maior sobre o código, uma vez que é necessário fazer todo o processo de criação e destruição da imagem chamando cada função no momento certo para tal diferentemente do que fazemos na versão 2.x em C++, onde os contrutores e destrutores de cada objeto abstraem todo esse processo.

Na sessão 3 - Códigos serão apresentados os códigos utilizados para os processos descritos.

Para o processo do OpenCV 1.x cria-se uma imagem, nomea-se uma janela para mostrá-la, mostra-se a imagem, destrói-se a janela e por último libera-se o ponteiro da imagem (processo semelhante a abertura e fechamento de arquivos em C, uma vez que uma IplImage é uma struct assim como FILE), caso o processo não seja feito por completo poderá haver problemas de acesso de memória ou de reutilização de ponteiros mais à frente do código. Por outro lado em C++ precisamos apenas declarar um objeto matriz e ao final seu destrutor se encarrega de fazer as operações necessárias

E ao final do programa o destrutor do objeto 'cv::Mat' se encarrega de liberar e deletar da memória o que for necessário. É importante ressaltar que mesmo que o processo em OpenCV 2.x seja abstraído não há perda em perfomance em relação ao código em OpenCV 1.x, pelo contrário, o desempenho da primeira aplicação em relação à segunda para uma imagem em 720p chega a ser 3ms melhor -demonstrado posteriormente neste artigo-. Assim, pode-se notar de início as vantagens de escrever o código em OpenCV 2.x, uma vez que consome menos linha de código e consequentemente menos tempo de desenvolvimento e tem um desempenho semelhante ou melhor que o código em C para obter o mesmo resultado esperado.

O computador usado para os testes foi um Inspiron 14R, 8gb de RAM e processador i5 de 2ª geração.

2 Análise dos códigos

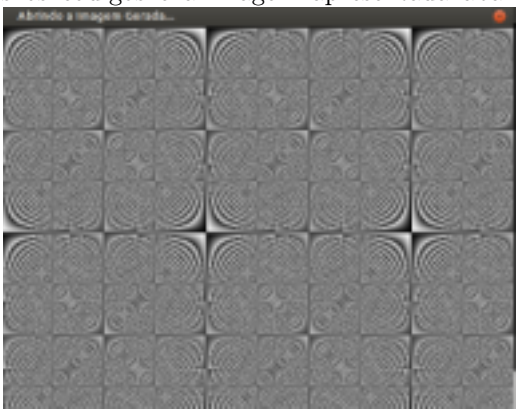
O código de OpenCV 1.x verifica-se ser um código demasiadamente extenso para criação de uma imagem e uma unica operação nela. Declara-se um ponteiro de imagem Ipl e uma struct de cvSize para associar o tamanho da imagem, a esse ponteiro, então criar uma imagem e fazer a operação que se deseja. Percebe-se como o acesso a um elemento dessa matriz é de difícil entendimento à primeira vez que se le a linha.

```
((uchar*)(cvImg->imageData
+ cvImg->widthStep*j1))[i1] =
```

```
( char ) ( ( i1 * j1 ) % 256 );
```

Há muito acesso referenciado '->' na struct em que se está trabalhando e não é tão intuitivo como deveria ser. Após a criação da janela que mostrará a imagem e da mostragem é estritamente necessário destruir tanto a janela como liberar o ponteiro de imagem para desalocar a memória usada durante a execução, como se oberva nas duas linhas que antecedem o return do código. [talvez explicar o funcionamento da ipl seja importante.

Para o código em OpenCV 2.x tem-se um código muito mais enxuto e legível em relação ao anterior, com menos declarações e com acessos mais fáceis de entender. As melhorias da versão 1.x para a 2.x são vísiveis e extremamente agrádaveis. Assim, basta declarar um objeto da classe Mat e inicializá-lo usando seu construtor[referenciar] de acordo com os parâmetros disponíveis na documentação[referenciar]. O acesso é feito como explicado nessse tutorial[referenciar] do próprio opencv - e citado como forma mais eficiente de acessar os elementos de uma cv::Mat -. O processo, em ambos os casos, consiste em multiplicar o índice i pelo índice j da matrix e extrair o módulo da divisão por 256 associando o resultado à esse pixel de índice (i,j). O resultado de ambos os códigos é a imagem apresentada abaixo.



2.1 Desempenho dos códigos

Os códigos foram repetidos para as resoluções 1280x720, 4320x1280, 8640x4320.

Todos os algoritmos foram rodados 10000 vezes e exportados para arquivo .txt para que se possa observar a convergência da curva normal para um valor de alta probabilidade, que neste artigo, será tomado como valor verdadeiro do teste. Isso foi feito porque há fatores aleatórios que afetam as medidas em pequena escala, como processos do sistema ou em segundo plano.

As distribuições e gráficos foram calculados e plotados usando as bibliotecas numpy e matplotlib e o programa de cálculo da estatística dos valores obtidos foi escrito em python.

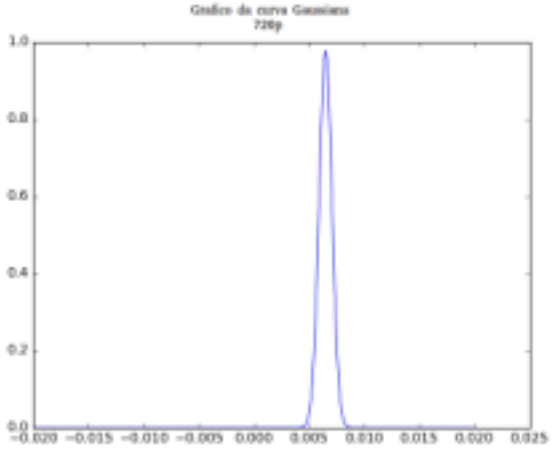
2.2 Comparação entre os algoritmos

Tabela 1: Valores de execução do Algoritmo

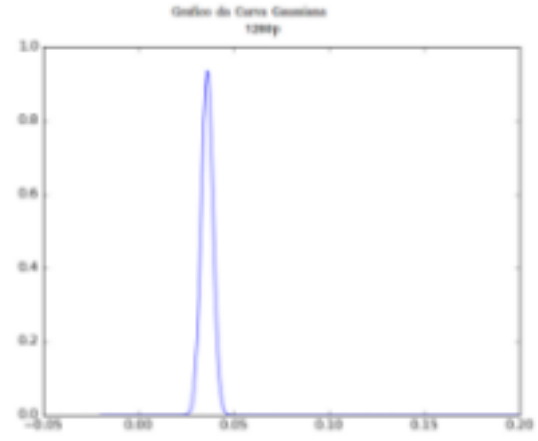
	1.x	2.x
720p	6.4ms	3.4ms
1280p	3.6x10ms	1.6x10ms
4320p	$3.1x10^2$ms	$1.2x10^2$ms

Pode-se perceber a diferença entre os dois algoritmos principalmente na maior resolução, onde há quase $1.9x10^2$ms de diferença de um para o outro. Tal discrepância é dada por causa da alocação e acesso linear que tem de ser feito a cada vez que for se criar uma imagem nova usando OpenCV 1.x, é necessário alocar mais structs, fazer mais referências e perder mais tempo liberando ponteiros, assim, fica provado a eficácia do algoritmo escrito em OpenCV 2.x em relação ao 1.x.

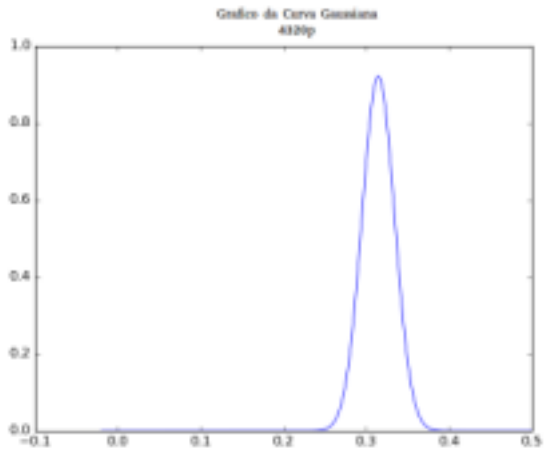
Os gráficos obtidos são apresentados logo abaixo:



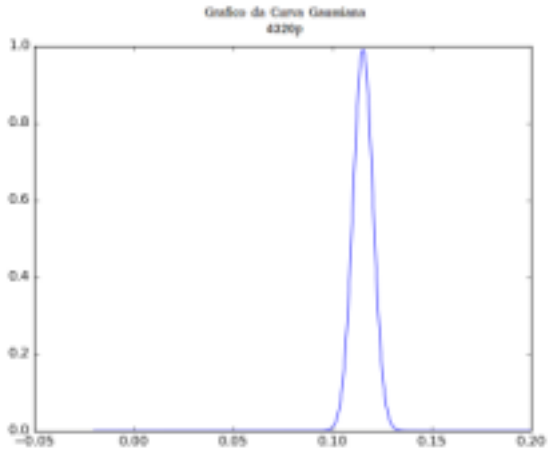
Opencv 1.x 720p: 6.4ms



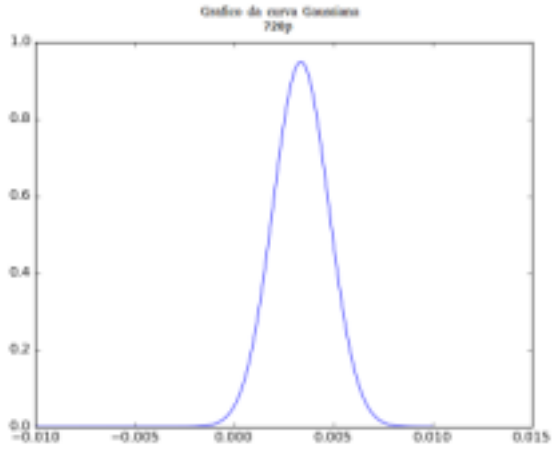
Opencv 1.x 1280p: 3.6x10ms



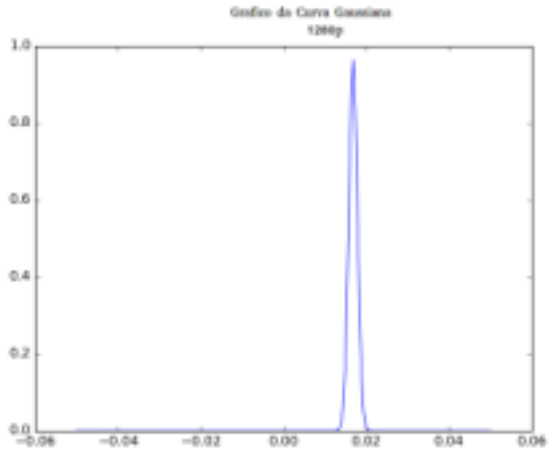
Opencv 1.x 4320p: $3.1x10^2$ms



Opencv 2.x 4320p: $1.2x10^2$ms



Opencv 2.x 720p: 3.4ms



Opencv 2.x 1280p: 1.6x10ms

3 Códigos utilizados

O código em Opencv 1.x utilizado foi:

```
int main()
{
  IplImage *cvImg;
   CvSize imgSize;
   int i1 = 0, j1 = 0;
   imgSize.width = COLS;
   imgSize.height = ROWS;

   cvImg = cvCreateImage( imgSize, 8, 1 );

   for ( i1 = 0; i1 < imgSize.width; i1++ )
       for ( j1 = 0; j1 < imgSize.height; j1++ )
           ((uchar*)(cvImg->imageData +
           cvImg->widthStep*j1))[i1]
           = ( char ) ( ( i1 * j1 ) % 256 );
   cvNamedWindow( "Abrindo a Imagem Gerada...", 1 );
   cvShowImage( "Abrindo a Imagem Gerada...", cvImg );
   cvWaitKey( 4000 );
   cvDestroyWindow( "image" );
   cvReleaseImage( &cvImg );
}
```

O código em Opencv 2.x foi:

```
int main(int argc, char const *argv[])
{
    cv::Mat Image(ROWS, COLS, CV_8U);
    uchar* p;

    for (int i = 0; i < Image.rows; ++i)
    {
        p = Image.ptr<uchar>(i);
        for (int j = 0; j < Image.cols; ++j)
            p[j] = ((i*j) % 256);
    }

    cv::imshow("Created Image", Image);
    cv::waitKey(0);

    return 0;
}
```

Onde "ROWS, COLS"são constantes definidas para a resolução desejada.

O código do algoritmo estatístico escrito em python segue:

```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.mlab as mlab
import math
import statistics

with open("timevaluesRESOLp.txt") as f:
    lines = f.read().splitlines()

numbers = list(map(float, lines))

mean = statistics.mean(numbers)
variance = statistics.variance(numbers)
sigma = math.sqrt(variance)
x = np.linspace(-0.02, 0.2, 300)
plt.plot(x, mlab.normpdf(x, mean, sigma)/80)
plt.show()
```

Onde 'RESOL' na função open() é a resolução desejada em p (720p, 1280p, 4320p).

O terceiro parametro da função linspace() deve ser ajustado a fim de deixar a curva o menos quadrada possivel assim como o segundo parametro da função plot deve ser ajustado para que a curva seja normalizada entre 0 e 1, configuração da Gaussiana.

4 Conclusão

Pode-se concluir com os valores mostrados que usar a versão 2.x é mais fácil, elegante e rápido - tanto em tempo de desenvolvimento como em tempo de execução - as melhorias trazidas são vísiveis e altamente aplicáveis, sendo assim, não há motivos para o uso da versão 1.x, seu conhecimento serve para que se possa acompanhar a evolução da biblioteca.

Os gráficos apresentados neste artigo comprovam as proposições apresentadas, uma vez que todos resultam em uma probabilidade acima de 92% para os valores citados em ms. Como dito anteriormente, os 8% restantes são distrubuidos entre atrasos aleatórios decorrentes de outros processos em execução na CPU.

Foi possível observar também o aumento do tempo de execução do algoritmo em decorrência do aumento da resolução da imagem o que está de acordo com o resultado esperado para o experimento já que, quanto maior a resolução da imagem, maior a quantidade de pixeis que o algoritmo tem de escanear. Sendo assim, uma resolução alta pode atrasar o tempo de processamento sem trazer muitas vantagens de detalhes se cada sensor da câmera não estiver associado a um único pixel como acontece na maioria das câmeras comuns e webcams.

Referências