عنوان: **OpenCVSharp #1**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۱۱ ۱۵:۱۰
آدرس: www.dotnettips.info
گروه‌ها: OpenCV

معرفی OpenCV

پردازش تصاویر علمی است برای پیاده سازی الگوریتم‌های مختلفی بر روی تصاویر دیجیتال؛ برای مثال تشخیص خودکار شماره‌ی پلاک خودروهای وارد شده‌ی به محدوده‌ی طرح ترافیک، تا تشخیص چهره‌ی افراد، در گوشی‌های همراه. پردازش تصاویر، در صنایع مختلف، علوم پزشکی و همچنین نظامی، کاربردهای بسیاری دارند.
برای انجام این کار، کتابخانه‌های بسیار زیادی طراحی شده‌اند؛ اما در این بین OpenCV جایگاه خاصی دارد. این کتابخانه‌ی بسیار مشهور سورس باز، جهت پردازش تصاویر در سیستم عامل‌های مختلفی مانند Windows, Mac, Linux, Android و iOS بکار می‌رود.

محصور کننده‌های OpenCV مخصوص دات نت

تا امروز محصور کننده‌های زیادی جهت استفاده‌ی از کتابخانه‌ی OpenCV در دات نت طراحی شده‌اند که تعدادی از مهم‌ترین‌های آن‌ها به شرح زیر هستند:

الف) Emgu CV
این کتابخانه، یکی از مشهورترین محصور کننده‌های OpenCV است و دارای مجوزی دوگانه می‌باشد. برای کارهای سورس باز، مجوز GPL دارد (یعنی باید کارتان را سورس باز کنید) و برای کارهای تجاری باید مجوز آن‌را بخرید. البته باید توجه داشت که مجوز کتابخانه‌ی اصلی OpenCV از نوع BSD است و این محدودیت‌ها را ندارد.

ب) OpenCvSharp
کتابخانه‌ی OpenCvSharp دارای مجوز BSD است (همانند کتابخانه‌ی اصلی OpenCV) و محدودیتی برای استفاده ندارد. هر دو نوع مدل برنامه نویسی OpenCV را که شامل متدهای C و C++ آن‌است، پشتیبانی می‌کند و در طراحی آن سعی شده‌است که بیشترین نزدیکی به طراحی اصلی OpenCV وجود داشته باشد. همچنین این کتابخانه چندسکویی بوده و با Mono لینوکسی نیز سازگار است و از دات نت 2 به بعد را نیز پشتیبانی می‌کند. جامعه‌ی کاربری آن فعال است و مدام به روز می‌شود.

ج) SharperCV
دیگر نگهداری نمی‌شود.

د) OpenCVDotNet
آخرین تاریخ به روز رسانی آن سال 2007 است.

ه) DirectCV
آخرین تاریخ به روز رسانی آن سال 2011 است.

در این بین یکی از بهترین انتخاب‌ها، کتابخانه‌ی OpenCvSharp ژاپنی است. مجوز استفاده‌ی از آن محدود نیست. به روز رسانی مرتب و منظمی دارد و API آن طوری طراحی شده‌است که به سادگی بتوانید مثال‌های C و C++ کتابخانه‌ی OpenCV را تبدیل به معادل‌های C# کنید.

نصب OpenCvSharp

برای نصب کتابخانه‌ی OpenCvSharp می‌توان از بسته‌های نیوگت آن کمک گرفت. این کتابخانه به همراه دو بسته‌ی نیوگت ارائه می‌شود.

اگر فرمان ذیل را صادر کنید

```
PM> Install-Package OpenCvSharp-AnyCPU
```

علاوه بر اسمبلی‌های دات نتی OpenCVSharp، کتابخانه‌ی native مربوط به OpenCV سازگار با نگارش ارائه شده را نیز دریافت خواهید کرد.
و اگر دستور ذیل را اجرا کنید:

```
PM> Install-Package OpenCvSharp-WithoutDll
```

به این معنا است که تنها اسمبلی‌های دات نتی OpenCVSharp را دریافت می‌کنید. در این حالت نیاز است به سایت OpenCV مراجعه و بسته‌های کامپایل شده‌ی آن‌را دریافت کنید. سپس فایل‌های dll موجود در پوشه‌ی opencv\build\x64\vc12\bin را برای مثال به پوشه‌ی bin پروژه‌ی خود کپی نمائید.

روش توصیه شده‌ی در اینجا، همان نصب بسته‌ی نیوگت OpenCvSharp-AnyCPU است. به این ترتیب نگارش‌های X86 و X64 کتابخانه‌ی OpenCV سازگار با OpenCvSharp را نیز دریافت خواهید کرد.

نکته‌ای در مورد ارائه‌ی نهایی پروژه‌های مبتنی بر OpenCV

OpenCV یک کتابخانه‌ی native ویندوز است و دات نتی نیست . بنابراین DLLهای آن باید بسته به معماری CPU جاری، انتخاب شوند. یعنی اگر برنامه‌ی دات نتی خود را در حالت Any CPU کامپایل می‌کنید، این برنامه در یک سیستم 64 بیتی، 64 بیتی رفتار می‌کند و در یک سیستم 32 بیتی، 32 بیتی. بنابراین باید دقت داشت که اگر سیستم جاری 64 بیتی است و می‌خواهید از اسمبلی‌های X86 مربوط به OpenCV استفاده کنید، برنامه با پیام استثنای یافت نشدن OpenCV و BadImageFormatException کرش خواهد کرد. بسته‌ی نیوگت OpenCvSharp-AnyCPU شامل هر دو معماری X86 و X64 است و هر دو سری DLLهای OpenCV را به همراه دارد.
همچنین OpenCV تحت ویندوز، توسط کامپایلر ویژوال C++، کامپایل شده‌است. به همین جهت در این حالت، علاوه بر نصب دات نت، نیاز است VC++ redistributable packages را نیز بر روی کامپیوتر کلاینت نصب کرد.
پس از نصب بسته‌ی نیوگت OpenCvSharp-AnyCPU اگر به پوشه‌ی bin برنامه‌ی خود مراجعه کنید، پوشه‌ی جدید dll را نیز می‌توان مشاهده کرد. داخل این پوشه، دو پوشه‌ی X86 و X64 وجود دارند که حاوی DLLهای اصلی OpenCV می‌باشند. در این پوشه‌ها اگر برای مثال فایلی به نام msvcp120.dll را یافتید، یعنی این نگارش از OpenCV نیاز به بسته‌های مخصوص VC++ 12 دارد.

رعایت این دو نکته بسیار مهم است؛ در غیر اینصورت برنامه‌ی شما آغاز نخواهد شد.

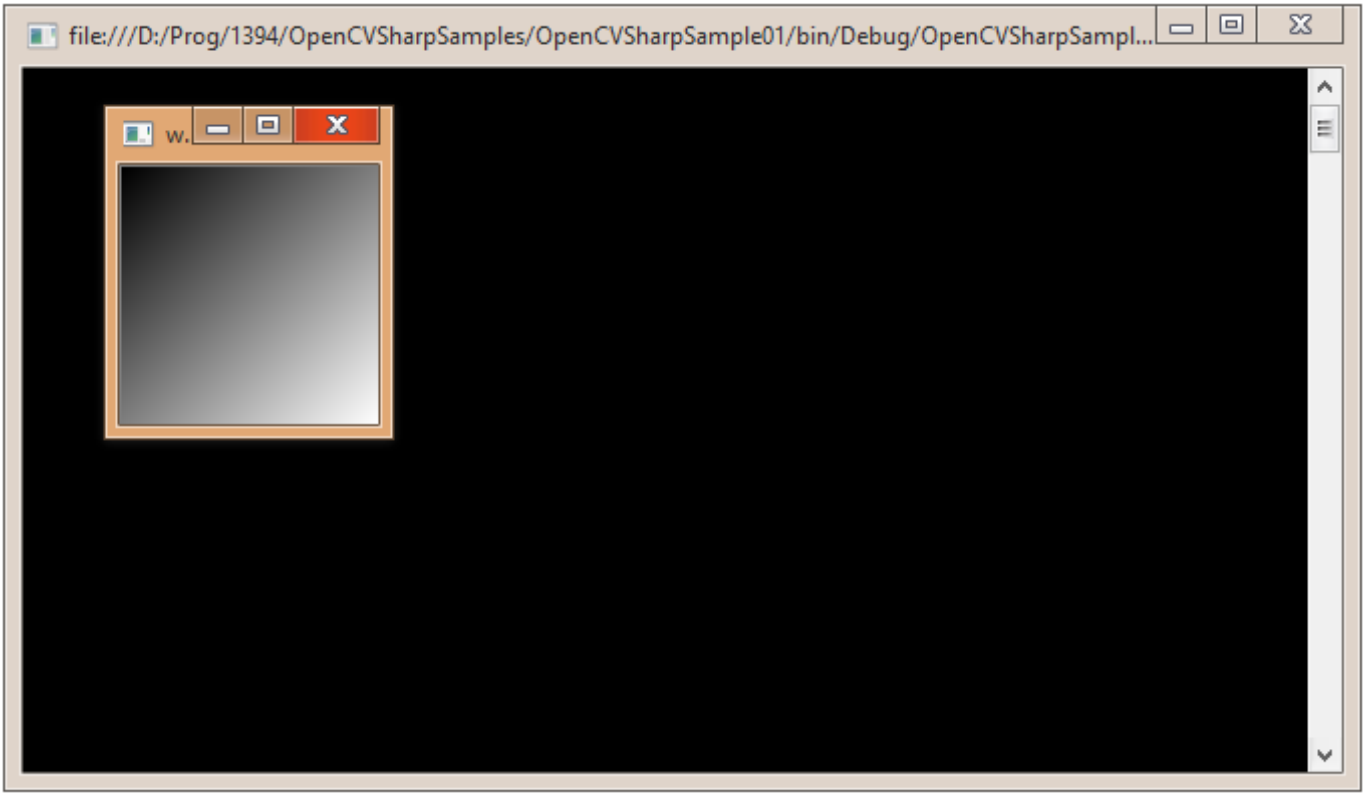
اولین برنامه‌ی OpenCVSharp

پس از نصب بسته‌ی نیوگت OpenCvSharp-AnyCPU، مقدمات نصب OpenCV به پایان می‌رسد. در ادامه یک برنامه‌ی کنسول جدید را ایجاد کرده و کدهای ذیل را به آن اضافه کنید:

```
using OpenCvSharp;

namespace OpenCVSharpSample01
{
    class Program
    {
        static void Main(string[] args)
        {
            var img = Cv.CreateImage(new CvSize(128, 128), BitDepth.U8, 1);

            for (var y = 0; y < img.Height; y++)
            {
                for (var x = 0; x < img.Width; x++)
                {
                    Cv.Set2D(img, y, x, x + y);
```

```
                }
            }

            Cv.NamedWindow("window");
            Cv.ShowImage("window", img);
            Cv.WaitKey();
            Cv.DestroyWindow("window");

            Cv.ReleaseImage(img);
        }
    }
}
```

این خروجی را دریافت خواهید کرد:



در این مثال یک تصویر 128*128 ایجاد شده و سپس با گرادیانی از رنگ خاکستری پر می‌شود. در ادامه یک پنجره‌ی native مخصوص OpenCV ایجاد شده و این تصویر در آن نمایش داده می‌شود.

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید.

نظرات خوانندگان

نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۴/۰۴/۰۹ ۲۱:۵۷

به نظر در مقدمه بحث جای خالی معرفی wrapper (محصور کننده) ها ی AForge.NET و Accord.NET (البته قسمت Imaging) احساس می‌شه.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۴/۰۹ ۲۲:۲

این‌ها محصور کننده‌های OpenCV نیستند . کدهای خالص دات نتی هستند که صفر نوشته شده‌اند.

عنوان: OpenCVSharp #2
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۱۲ ۱۱:۵
آدرس: www.dotnettips.info
گروه‌ها: OpenCV

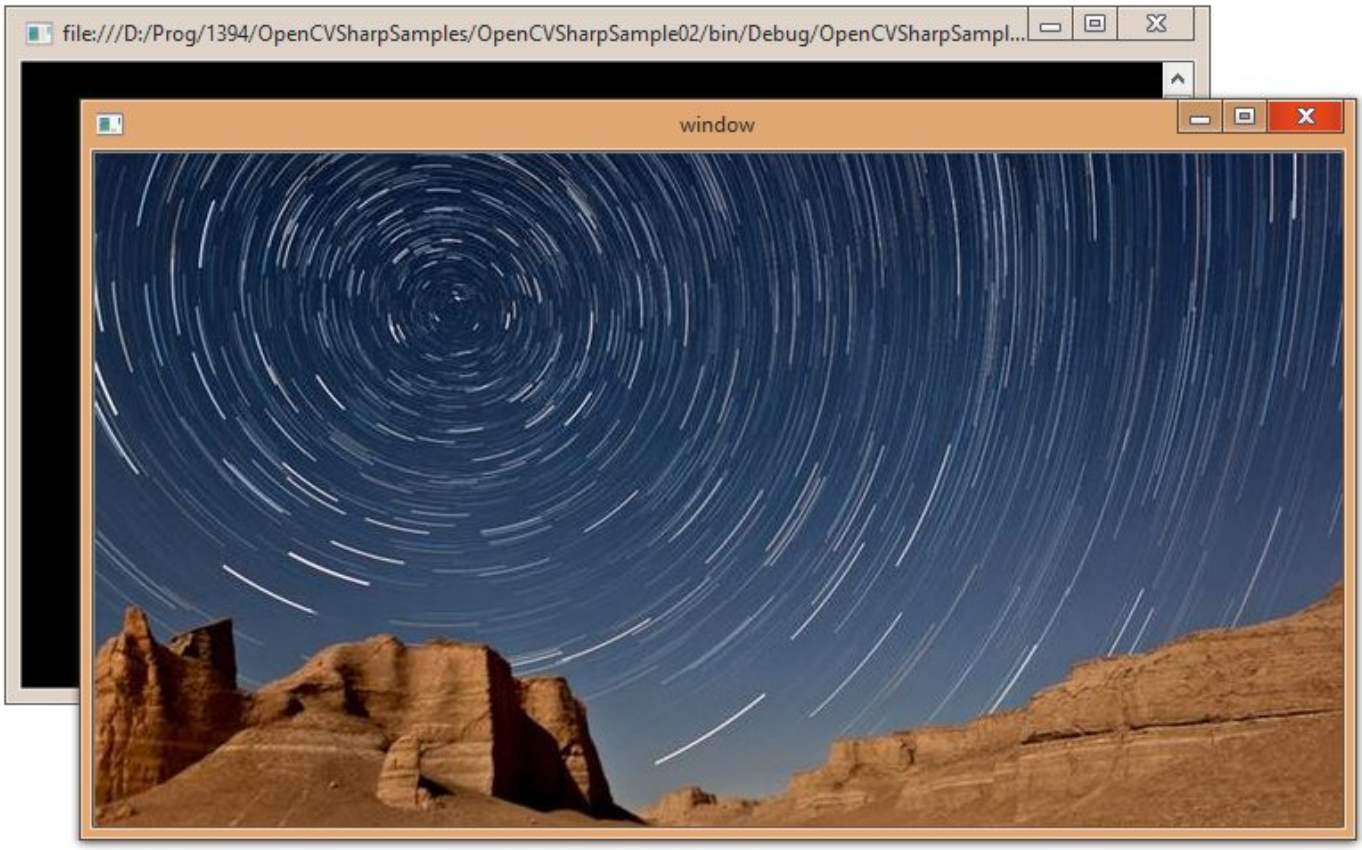
کتابخانه‌ی اصلی OpenCV، دارای دو نوع اینترفیس C و C++ است. اینترفیس C آن مرتبط است به نگارش‌های 1x آن و اینترفیس C++ آن به همراه نگارش‌های 2x آن ارائه شده‌اند. کتابخانه‌ی OpenCVSharp هر دو نوع اینترفیس یاد شده را پشتیبانی می‌کند. در این قسمت نگاهی خواهیم داشت به نحوه‌ی بارگذاری و نمایش تصاویر در OpenCV به کمک متدهای اینترفیس C آن، مانند cvLoadImage، cvShowImage، cvReleaseImage.

بارگذاری و نمایش تصاویر به کمک OpenCVSharp

متدهای اینترفیس C مربوط به OpenCV، در OpenCVSharp با ذکر کلاس Cv آن قابل دسترسی هستند. برای نمونه متدهای C یاد شده‌ی در ابتدای بحث، چنین معادلی را در OpenCVSharp دارند:

```
using OpenCvSharp;

namespace OpenCVSharpSample02
{
  class Program
  {
   static void Main(string[] args)
   {
    var img = Cv.LoadImage(@"..\..\images\ocv02.jpg");

    Cv.NamedWindow("window");
    Cv.ShowImage("window", img);

    Cv.WaitKey();

    Cv.DestroyWindow("window");

    Cv.ReleaseImage(img);
   }
  }
}
```

متد cvLoadImage اینترفیس C، به Cv.LoadImage تبدیل شده‌است و مابقی نیز به همین ترتیب.
در اینجا با استفاده از متد LoadImage، تصویری را از مسیر مشخصی، بارگذاری می‌کنیم. سپس یک پنجره‌ی OpenCV ایجاد و این تصویر در آن نمایش داده می‌شود. متد WaitKey منتظر فشرده شدن یک کلید بر روی پنجره‌ی OpenCV می‌شود. پس از آن این پنجره تخریب و همچنین منابع native این تصویر آزاد می‌شوند.



متد LoadImage، پارامتر دومی را نیز می‌پذیرد:

```
var img = Cv.LoadImage(@"..\..\images\ocv02.jpg", LoadMode.GrayScale);
```

برای مثال در اینجا می‌توان به کمک مقدار LoadMode.GrayScale، تصویر را به صورت سیاه و سفید بارگذاری کرد.
Enum تعریف شده‌ی در اینجا قابلیت or یا جمع منطقی را نیز دارد. برای مثال می‌توان مقدار LoadMode.AnyColor | LoadMode.AnyDepth را نیز مشخص کرد؛ جهت بارگذاری تصویر اصلی با مشخصات کامل آن که حالت پیش فرض است.

کلاس‌های پشت صحنه‌ی اینترفیس C در OpenCVSharp

علت وجود کلاس Cv در OpenCVSharp، سهولت برگرداندن مثال‌های C کتابخانه‌ی OpenCV به نمونه‌های دات نتی است. اما اگر قصد داشته باشید از کلاس‌های پشت صحنه‌ی این اینترفیس در OpenCVSharp استفاده کنید، می‌توان کدهای فوق را به نحو ذیل نیز بازنویسی کرد:

```
using (var img = new IplImage(@"..\..\images\ocv02.jpg", LoadMode.Unchanged))
{
  using (var window = new CvWindow("window"))
  {
   window.Image = img;
   Cv.WaitKey();
  }
}
```

خروجی متد LoadImage از نوع کلاس IplImage است. در اینجا می‌توان همین کلاس را وهله سازی کرد و مورد استفاده قرار داد. به علاوه اینبار این کلاس تهیه شده، اینترفیس IDisposable را نیز پیاده سازی می‌کند. بنابراین می‌توان با استفاده از عبارت using کار آزاد سازی منابع آن‌را خودکار کرد.

همچنین پنجره‌ی OpenCV نیز در اینجا با کلاس CvWindow پیاده سازی می‌شود که این کلاس نیز اینترفیس IDisposable را پیاده سازی می‌کند.

یک نکته‌ی تکمیلی

اگر متد LoadImage کتابخانه‌ی OpenCV قادر به بارگذاری تصویر شما نبود، متد دیگری به نام IplImage.FromFile نیز پیش بینی شده‌است. این متد از امکانات System.Drawing.Bitmap دات نت برای بارگذاری تصویر و تبدیل آن به فرمت OpenCV استفاده می‌کند.

کدهای کامل این مثال را از اینجا **می‌توانید دریافت کنید.**

عنوان: **OpenCVSharp #3**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۱۳ ۱۰:۳۰
آدرس: www.dotnettips.info
گروه‌ها: OpenCV

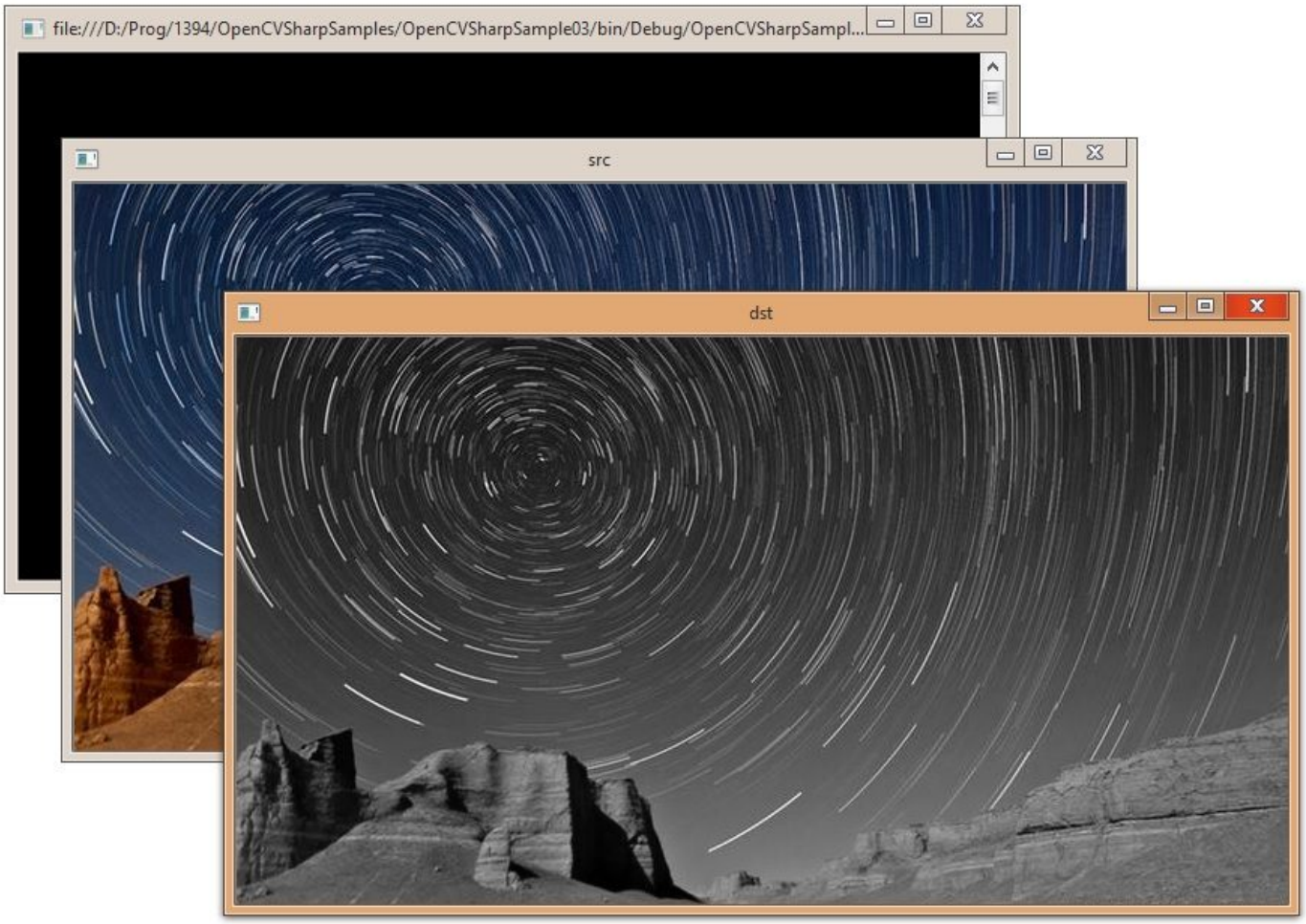
در قسمت دوم با نحوه‌ی بارگذاری تصاویر در OpenCVSharp آشنا شدیم. در این قسمت قصد داریم با نحوه‌ی ایجاد یک clone و نمونه‌ای مشابه از تصویر اصلی بارگذاری شده آشنا شویم. برای مثال هرچند متد LoadImage، دارای پارامتر بارگذاری تصویر، به صورت سیاه و سفید است، اما توصیه نمی‌شود که در بدو امر، تصویر را سیاه و سفید بارگذاری کنید. چون هرگونه تغییری در تصویر اصلی، امکان استفاده‌ی از آن‌را در سایر متدها و الگوریتم‌ها با مشکل مواجه می‌کند و استفاده‌ی از حالت LoadMode.GrayScale جهت بالا بردن سرعت عملیات، در کارهای پردازش تصویر بسیار معمول است.

تهیه‌ی یک نمونه‌ی سیاه و سفید از تصویر اصلی در OpenCVSharp

برای تهیه‌ی یک نمونه‌ی مشابه تصویر اصلی، از متد CreateImage استفاده می‌شود:

```
using (var src = Cv.LoadImage(@"..\..\images\ocv02.jpg", LoadMode.Color))
using (var dst = Cv.CreateImage(new CvSize(src.Width, src.Height), BitDepth.U8, 1))
{
    Cv.CvtColor(src, dst, ColorConversion.BgrToGray);

    using (new CvWindow("src", image: src))
    using (new CvWindow("dst", image: dst))
    {
        Cv.WaitKey();
    }
}
```

با این خروجی



معرفی متد CreateImage

پارامتر اول متد CreateImage ، اندازه‌ی تصویر تولیدی را مشخص می‌کند. پارامتر دوم آن تعداد بیت تصویر را تعیین خواهد کرد. این تعداد بیت عموما بر اساس نیاز متدهای مختلف پردازش تصویر، متغیر خواهند بود و برای تعیین آن نیاز است مستندات هر متد را مطالعه کرد. BitDepth.U8 به معنای 8bit unsigned است.
پارامتر سوم این متد، تعیین کننده‌ی تعداد کانال تصویر است. تصاویر رنگی دارای سه کانال سبز، قرمز و آبی، هستند. چون در اینجا قصد داریم تصویر را سیاه و سفید کنیم، تعداد کانال را به عدد یک تنظیم کرده‌ایم.

متد CreateImage جهت سازگاری با اینترفیس C مربوط به OpenCV در اینجا وجود دارد. معادل

```
using (var dst = Cv.CreateImage(new CvSize(src.Width, src.Height), BitDepth.U8, 1))
```

را می‌توان به نحو ذیل نیز نوشت:

```
var dst = new IplImage(new CvSize(src.Width, src.Height), BitDepth.U8, 1)
```

و یا حتی پارامتر تعیین اندازه‌ی تصویر را نیز می‌توان ساده‌تر کرد:

```
using (var dst = new IplImage(src.Size, BitDepth.U8, 1))
```

تبدیل تصویر به حالت سیاه و سفید

متد CvtColor جهت تغییر color space بکار می‌رود که در اینجا BGR (Blue/Green/Red) را به Gray تبدیل کرده‌است:

```
Cv.CvtColor(src, dst, ColorConversion.BgrToGray);
```

این متد را در OpenCVSharp به نحو ذیل نیز می‌توان بازنویسی کرد:

```
src.CvtColor(dst, ColorConversion.BgrToGray);
```

بنابراین به صورت خلاصه می‌توان کدهای ابتدای بحث را به صورت زیر نیز نوشت که با کلاس‌های OpenCVSharp بیشتر سازگاری دارد:

```
using (var src = new IplImage(@"..\..\images\ocv02.jpg", LoadMode.Color))
//using (var dst = new IplImage(new CvSize(src.Width, src.Height), BitDepth.U8, 1))
using (var dst = new IplImage(src.Size, BitDepth.U8, 1))
{
    src.CvtColor(dst, ColorConversion.BgrToGray);

    using (new CvWindow("src", image: src))
    using (new CvWindow("dst", image: dst))
    {
        Cv.WaitKey();
    }
}
```

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید.

عنوان: **OpenCVSharp #4**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۱۴ ۱۳:۰
آدرس: www.dotnettips.info
گروه‌ها: OpenCV

کار با فیلترها در OpenCVSharp

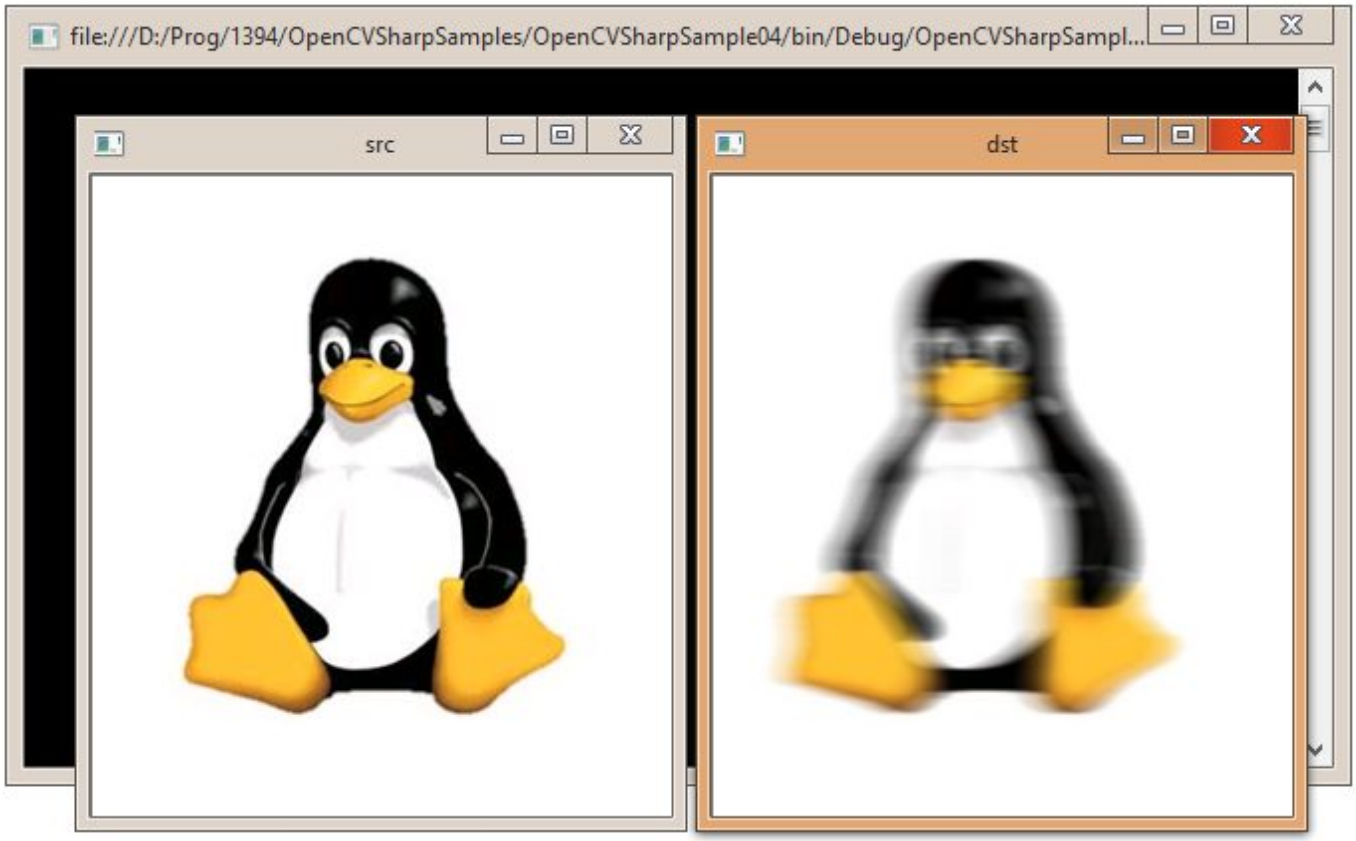
فرض کنید قصد داریم یک چنین مثال زبان C را که در مورد کار با فیلترها در OpenCV است، به نمونه‌ی دات نتی آن تبدیل کنیم:

```
#include <cv.h>
        #include <highgui.h>
        #include <stdio.h>

        int main (int argc, char **argv)
        {
          IplImage *src_img = 0, *dst_img;
          float data[] = { 2, 2, 2, 2, 2, 2, 2, 2, 2,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1
          };
          CvMat kernel = cvMat (1, 21, CV_32F, data);

          if (argc >= 2)
            src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR);
          if (src_img == 0)
            exit (-1);

          dst_img = cvCreateImage (cvGetSize (src_img), src_img->depth, src_img->nChannels);

          cvNormalize (&kernel, &kernel, 1.0, 0, CV_L1);
          cvFilter2D (src_img, dst_img, &kernel, cvPoint (0, 0));

          cvNamedWindow ("Filter2D", CV_WINDOW_AUTOSIZE);
          cvShowImage ("Filter2D", dst_img);
          cvWaitKey (0);

          cvDestroyWindow ("Filter2D");
          cvReleaseImage (&src_img);
          cvReleaseImage (&dst_img);

          return 0;
        }
```

معادل OpenCVSharp آن به صورت ذیل خواهد بود:

```
using (var src = new IplImage(@"..\..\Images\Penguin.Png", LoadMode.AnyDepth | LoadMode.AnyColor))
using (var dst = new IplImage(src.Size, src.Depth, src.NChannels))
{
    float[] data = { 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };
    var kernel = new CvMat(rows: 1, cols: 21, type: MatrixType.F32C1, elements: data);

    Cv.Normalize(src: kernel, dst: kernel, a: 1.0, b: 0, normType: NormType.L1);
    Cv.Filter2D(src, dst, kernel, anchor: new CvPoint(0, 0));

    using (new CvWindow("src", image: src))
    using (new CvWindow("dst", image: dst))
    {
        Cv.WaitKey(0);
    }
}
```

با این خروجی:



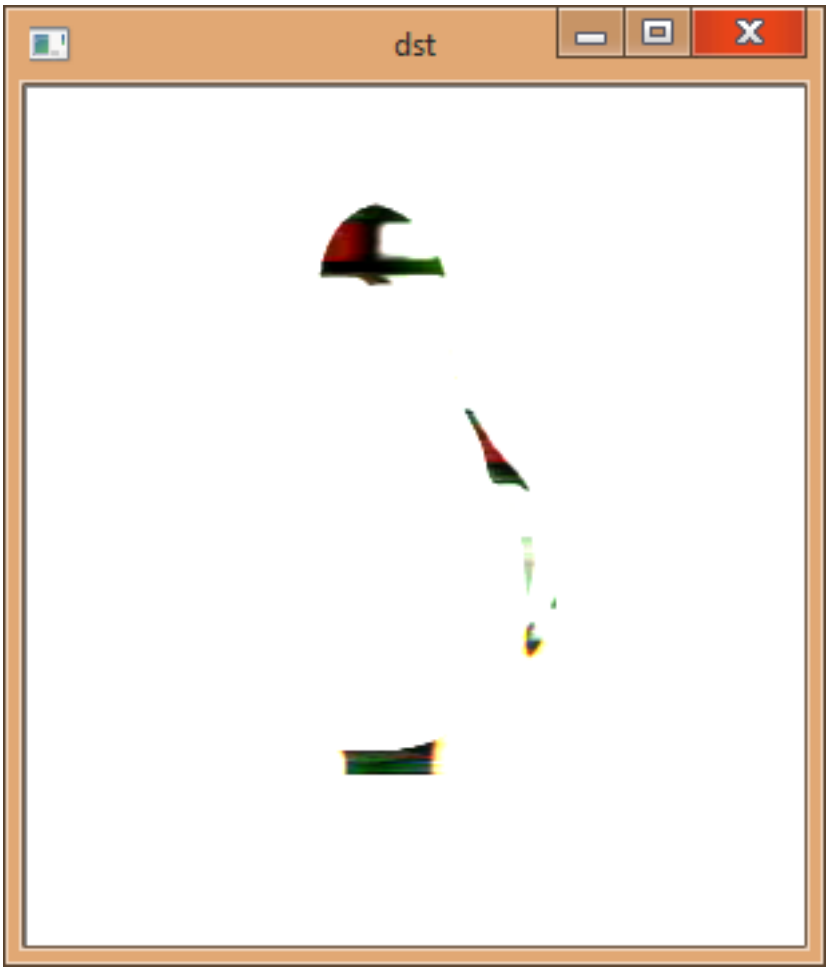
در قسمت‌های قبلی در مورد بارگذاری تصاویر، تهیه‌ی یک Clone از آن و همچنین ساخت یک پنجره به روش‌های مختلف و رها سازی خودکار منابع مرتبط، بیشتر بحث شد. در اینجا تصویر اصلی با همان عمق و وضوح تغییر نیافته‌ی آن بارگذاری می‌شود.
کار cvMat ، آغاز یک ماتریس OpenCV است. پارامترهای آن، تعداد ردیف‌ها، ستون‌ها، نوع داده‌ی المان‌ها و داده‌های مرتبط را مشخص می‌کنند.
در این مثال آرایه‌ی data، یک فیلتر را تعریف می‌کند که در اینجا، حالت یک بردار را دارد تا یک ماتریس. برای تبدیل آن به ماتریس، از شیء CvMat استفاده خواهد شد که آن‌را تبدیل به ماتریسی با یک ردیف و 21 ستون خواهد کرد.
در اینجا از نام کرنل استفاده شده‌است. کرنل در OpenCV به معنای ماتریسی از داده‌ها با یک نقطه‌ی anchor (لنگر) است. این لنگر به صورت پیش فرض در میانه‌ی ماتریس قرار دارد (نقطه‌ی 1- , 1-).

1	-2	1
2	-4	2
1	-2	1

مرحله‌ی بعد، نرمال سازی این فیلتر است. تاثیر نرمال سازی اطلاعات را به این نحو می‌توان نمایش داد:

```
double sum = 0;
foreach (var item in data)
{
     sum += Math.Abs(item);
}
Console.WriteLine(sum); // => .999999970197678
```

در اینجا پس از نرمال سازی، جمع عناصر بردار data، تقریبا مساوی 1 خواهد بود و تمام عناصر بردار data، به داده‌هایی بین یک و صفر، نگاشت خواهند شد.
اگر مرحله‌ی نرمال سازی اطلاعات را حذف کنیم، تصویر نهایی حاصل، چنین شکلی را پیدا می‌کند:



زیرا عملیات تغییر اندازه‌ی اطلاعات بردار صورت نگرفته‌است و داده‌های آن مطلوب متد cvFilter2D نیست.
و مرحله‌ی آخر، اجرای این بردار نرمال شده خطی، بر روی تصویر اصلی به کمک متد cvFilter2D است. این متد، تصویر مبدا را پس از تبدیلات ماتریسی، به تصویر مقصد تبدیل می‌کند. فرمول ریاضی اعمال شده‌ی در اینجا برای محاسبه‌ی نقاط تصویر خروجی به صورت زیر است:

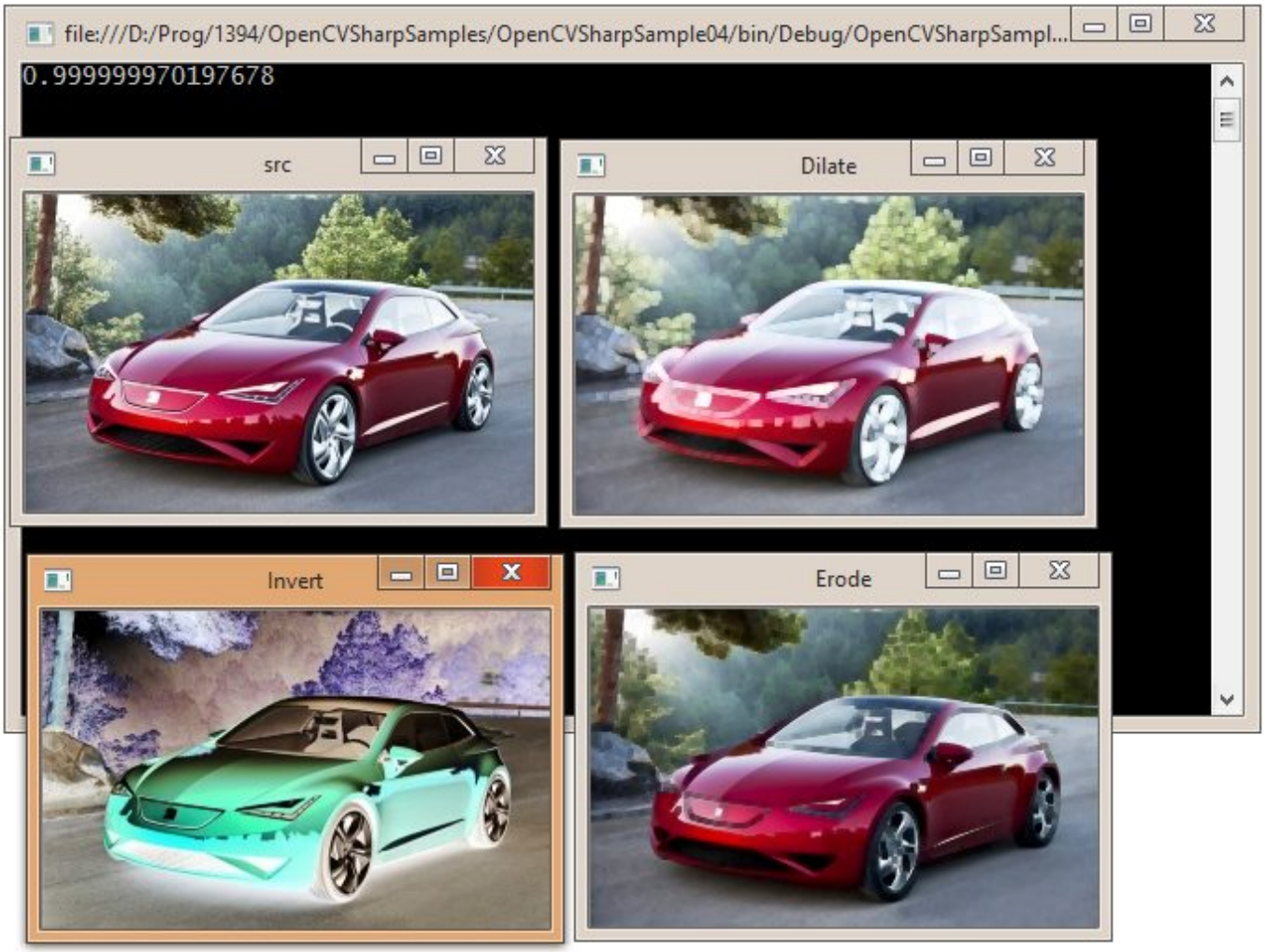
$$\mathrm{dst}(x,y) = \sum_{\substack{0 \le x' < \mathrm{kernel.cols}, \\ 0 \le y' < \mathrm{kernel.rows}}} \mathrm{kernel}(x',y') * \mathrm{src}(x + x' - \mathrm{anchor.x}, y + y' - \mathrm{anchor.y})$$

فیلترهای توکار OpenCV

علاوه بر امکان طراحی فیلترهای سفارشی خطی مانند مثال فوق، کتابخانه‌ی OpenCV دارای تعدادی فیلتر توکار نیز می‌باشد که نمونه‌ای از آن‌را در مثال ذیل می‌توانید مشاهده کنید:

```
using (var src = new IplImage(@"..\..\Images\Car.jpg", LoadMode.AnyDepth | LoadMode.AnyColor))
{
    using (var dst = new IplImage(src.Size, src.Depth, src.NChannels))
    {
        using (new CvWindow("src", image: src))
        {
            Cv.Erode(src, dst);
            using (new CvWindow("Erode", image: dst))
            {
                Cv.Dilate(src, dst);
                using (new CvWindow("Dilate", image: dst))
                {
                    Cv.Not(src, dst);
                    using (new CvWindow("Invert", image: dst))
                    {
                        Cv.WaitKey(0);
                    }
                }
            }
        }
    }
}
```

با این خروجی



کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید.

نظرات خوانندگان

نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۴/۰۴/۰۴ ۱۰:۳۶

سلام / چرا کرنل ی که برای بلور کردن تصویر مثال اول استفاده شده 1 در 21 است؟ (و آیا لزوم نباید یک کرنل مربعی استفاده شه)

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۴/۰۴ ۱۱:۵۷

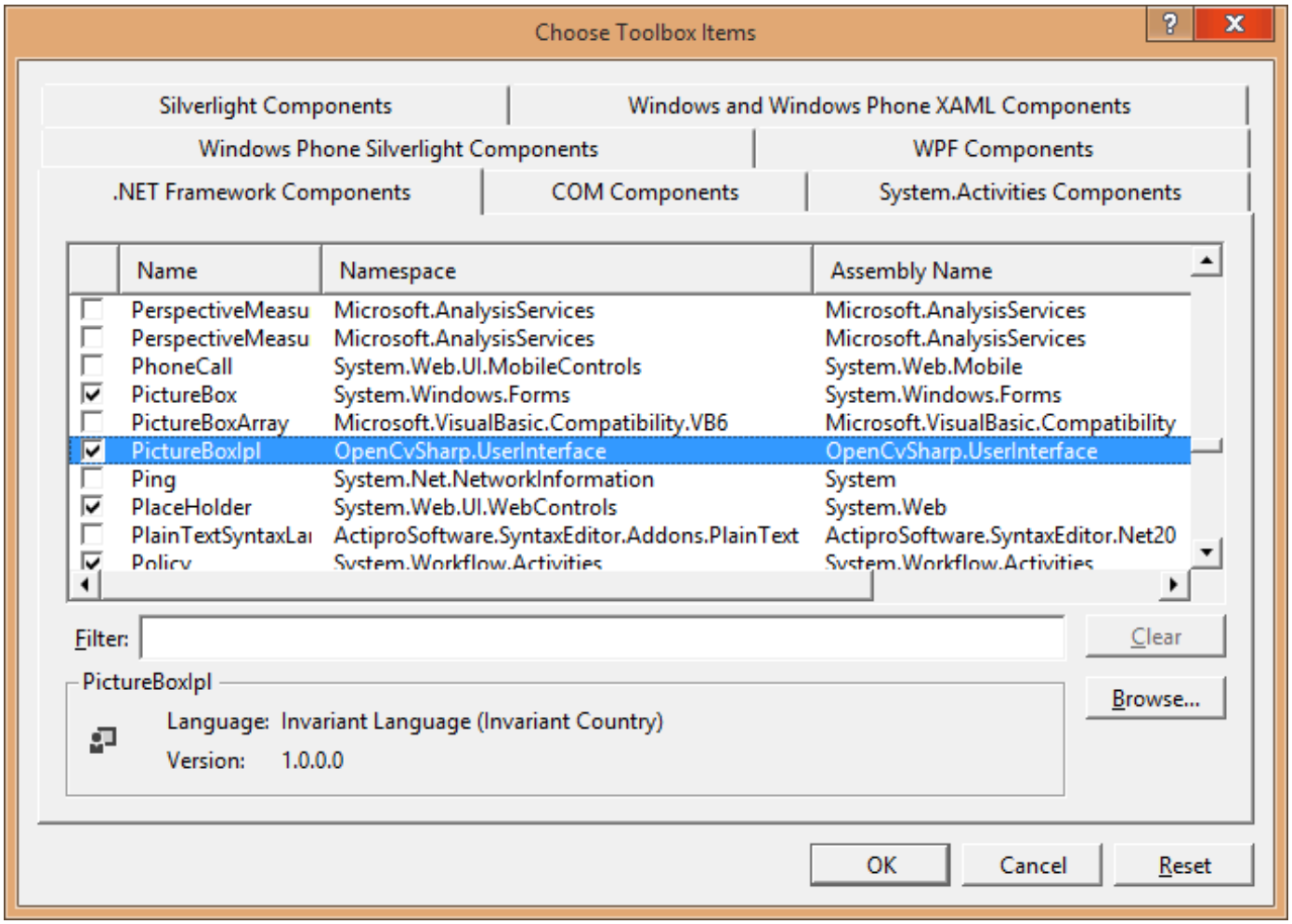
اندازه‌ی کرنل برای کار با بسیاری از متدهای دیگر باید یک عدد فرد باشد (در مورد عدد 21) و ... مربعی هست. در حقیقت یک ماتریس یک سطری با 21 ستون است (شیء CvMat تعریف شده از آن).

عنوان: OpenCVSharp #5
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۱۵ ۱۱:۴۰
آدرس: www.dotnettips.info
گروه‌ها: OpenCV

استفاده از پنجره‌ی native خود OpenCV، روش مرسومی است در زبان‌های مختلف برنامه نویسی که از OpenCV استفاده می‌کنند و این پنجره مستقل است از سکوی کاری مورد استفاده. اما شاید در دات نت علاقمند باشید که نتیجه‌ی عملیات را در یک picture box استاندارد نمایش دهید. در ادامه، تبدیل تصاویر OpenCV را به فرمت دات نت، در دو قالب برنامه‌های WinForms و همچنین WPF، بررسی خواهیم کرد.

استفاده از OpenCVSharp در برنامه‌های WinForms به کمک PictureBoxIpl

یکی از اسمبلی‌های کتابخانه‌ی OpenCVSharp را که در پوشه‌ی bin برنامه می‌توان مشاهده کرد،
OpenCvSharp.UserInterface.dll نام دارد. این اسمبلی حاوی یک picture box جدید به نام PictureBoxIpl است که می‌تواند تصاویری را با فرمت IplImage، دریافت کند.



می‌توانید این picture box ویژه را از طریق منوی Choose items -> ToolBox و سپس صفحه‌ی دیالوگ فوق، به نوار ابزار WinForms اضافه کرده و از آن استفاده کنید و یا می‌توان با کدنویسی نیز به آن دسترسی یافت:

```
using (var iplImage = new IplImage(@"..\..\Images\Penguin.png", LoadMode.Color))
```

```
{
    Cv.Dilate(iplImage, iplImage);

    var pictureBoxIpl = new OpenCvSharp.UserInterface.PictureBoxIpl
    {
        ImageIpl = iplImage,
        AutoSize = true
    };
    flowLayoutPanel1.Controls.Add(pictureBoxIpl);
}
```

در اینجا تصویر مورد نظر را توسط کلاس IplImage بارگذاری کرده و سپس برای نمونه فیلتر Dilate را به آن اعمال کرده‌ایم. سپس وهله‌ی جدیدی از کنترل PictureBoxIpl ایجاد و خاصیت ImageIpl آن، به تصویر بارگذاری شده، تنظیم و در آخر این picture box با کدنویسی به صفحه اضافه شده‌است.

یک نکته

هر نوع تغییری به iplImage پس از انتساب آن به خاصیت ImageIpl، نمایش داده نخواهد شد. برای به حداقل رساندن سربار ایجاد اشیاء جدید (خصوصا برای نمایش اطلاعات رسیده‌ی از دوربین یا WebCam)، از متد RefreshIplImage استفاده کنید. این متد بجای ایجاد یک شیء جدید، تنها ناحیه‌ی موجود را مجددا ترسیم خواهد کرد و بسیار سریع است:

```
pictureBoxIpl.RefreshIplImage(iplImage);
```

استفاده از OpenCVSharp در برنامه‌های WinForms به کمک PictureBox

اگر نخواهید از کنترل جدید PictureBoxIpl استفاده کنید، می‌توان از همان Picture box استاندارد WinForms نیز کمک گرفت:

```
Bitmap bitmap;
using (var iplImage = new IplImage(@"..\..\Images\Penguin.png", LoadMode.Color))
{
    bitmap = iplImage.ToBitmap(); // BitmapConverter.ToBitmap()
}

var pictureBox = new PictureBox
{
    Image = bitmap,
    ClientSize = bitmap.Size
};

flowLayoutPanel1.Controls.Add(pictureBox);
```

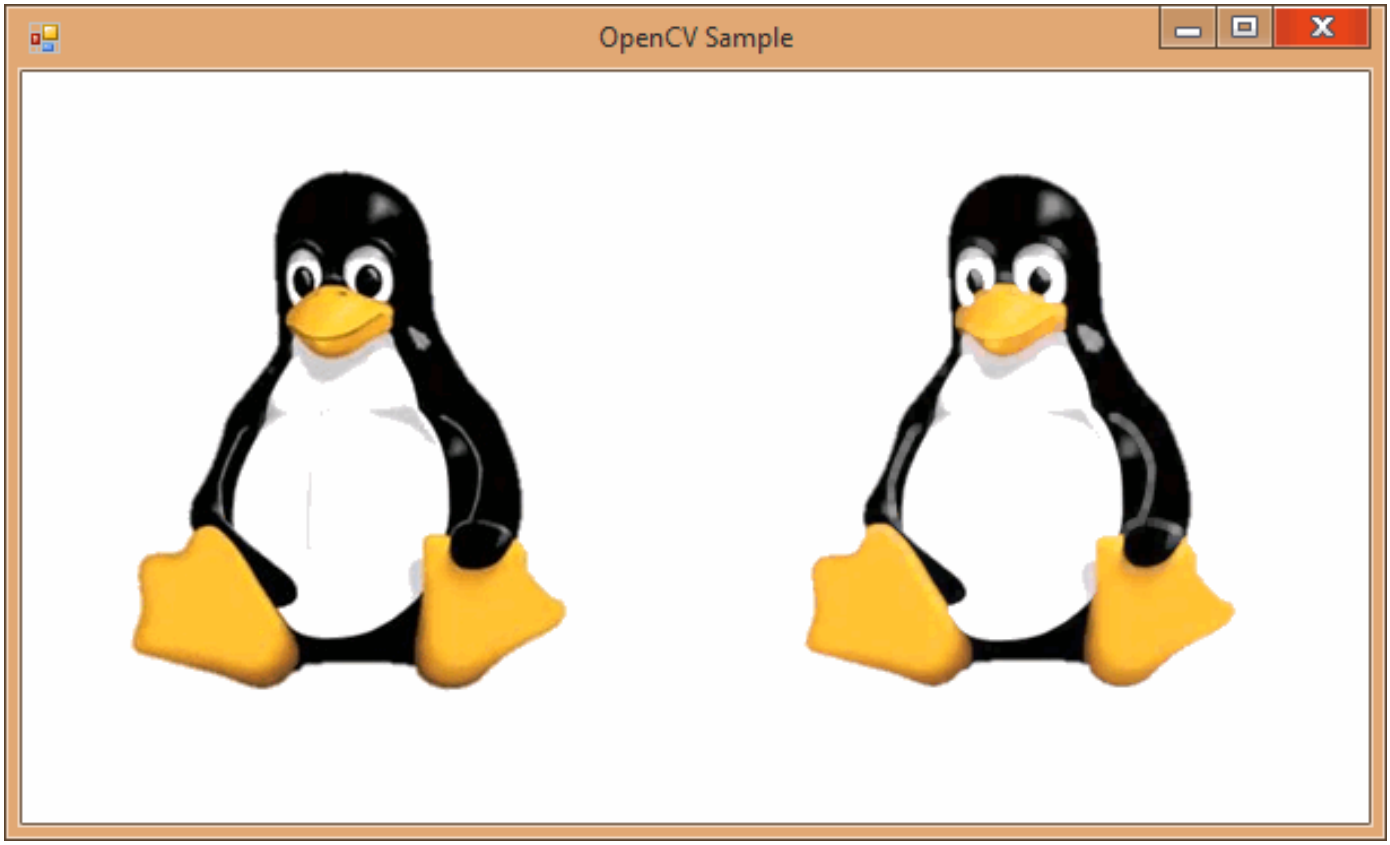
تنها نکته‌ای که در اینجا جدید است، استفاده از متد الحاقی ToBitmap می‌باشد که در کلاس BitmapConverter کتابخانه‌ی OpenCVSharp تعریف شده‌است. به این ترتیب تصویر با فرمت OpenCV، به یک Bitmap دات نتی تبدیل می‌شود. اکنون می‌توان این بیت‌مپ را برای مثال به یک Picture box استاندارد انتساب داد و یا حتی متد Save آن‌را فراخوانی کرد و آن‌را بر روی دیسک سخت، ذخیره نمود.

یک نکته

در اینجا نیز برای به حداقل رسانی به روز رسانی‌های بعدی picture box بهتر است از متد ToBitmap به شکل زیر کمک گرفت:

```
iplImage.ToBitmap(dst: (Bitmap)pictureBox.Image);
```

به این ترتیب سربار وهله سازی یک شیء جدید Bitmap حذف خواهد شد و صرفا ناحیه‌ی نمایشی مجددا ترسیم می‌شود.



استفاده از OpenCVSharp در برنامه‌های WPF

در WPF می‌توان با استفاده از متد الحاقی ToWriteableBitmap کلاس BitmapConverter، فرمت IplImage را به منبع تصویر یک کنترل تصویر استاندارد، تبدیل کرد:

```
using System.Windows.Media;
using OpenCvSharp;
using OpenCvSharp.Extensions;

namespace OpenCVSharpSample05Wpf
{
    public partial class MainWindow
    {
        public MainWindow()
        {
            InitializeComponent();
            loadImage();
        }

        private void loadImage()
        {
            using (var iplImage = new IplImage(@"..\..\Images\Penguin.png", LoadMode.Color))
            {
                Cv.Dilate(iplImage, iplImage);

                Image1.Source = iplImage.ToWriteableBitmap(PixelFormats.Bgr24);
            }
        }
    }
}
```

کدهای کامل WPF و WinForms این مطلب برای دریافت.

نظرات خوانندگان

نویسنده: علی ساری
تاریخ: ۱۳۹۴/۰۳/۱۶ ۸:۳۰

سلام؛ ممنون از مطلب خوبتون. من از کتابخانه‌های opncv و emgu برای خوندن تصویر از وب کم و پردازش تصویر استفاده میکنم. مشکل من سرعت پایین برنامه در رزولوشن بالای وب کم است هر چه رزولوشن بالاتر میره سرعت برنامه من کمتر میشه. مثلا تو یه سیستم corei7 با ram 8 gig من 61 درصد استفاده از cpu دارم و فیلم نمایش داده شده در برنامه از محیط واقعی عقب‌تر است. من در رویداد Application.Idle فرم این کد را قرار دادم:

```
public void Application_Idle(object sender, EventArgs e)
        {
            if (_capture != null)
            {
                try
                {
                    frame = _capture.QueryFrame();
                    Pic.Image = frame.ToBitmap();
                    //frame.ToBitmap(dst: (Bitmap)Pic.Image);
                }
                catch (NullReferenceException excpt)
                {
                    MessageBox.Show(excpt.Message);// you can also show any suitable message
                }
            }
        }
```

شما فرمودید برای به حداقل رسانی به روز رسانی‌های بعدی picture box بهتر است از متد ToBitmap به شکل زیر کمک گرفت:

```
iplImage.ToBitmap(dst: (Bitmap)pictureBox.Image);
```

ولی dst رو فرم نمیشناسه و اینکه آیا رویداد idle رویداد مناسبی برای این کار هست؟ نظرتون درباره سرعت پایین برنامه من چیه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۱۶ ۱۰:۴۷

« نمایش ویدیو و اعمال فیلتر بر روی آن »

عنوان: OpenCVSharp #6
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۱۶ ۱۰:۴۵
آدرس: www.dotnettips.info
گروه‌ها: OpenCV

نمایش ویدیو و اعمال فیلتر بر روی آن

در قسمت قبل با نحوه‌ی نمایش تصاویر OpenCV در برنامه‌های دات نتی آشنا شدیم. در این قسمت قصد داریم همان نکات را جهت پخش یک ویدیو توسط OpenCVSharp بسط دهیم.

روش‌های متفاوت پخش ویدیو و یا کار با یک Capture Device

OpenCV امکان کار با یک WebCam، دوربین و یا فیلم‌های آماده را دارد. برای این منظور کلاس CvCapture در OpenCVSharp پیش بینی شده‌است. در اینجا قصد داریم جهت سهولت پیگیری بحث، یک فایل avi را به عنوان منبع CvCapture معرفی کنیم:

```
using (var capture = new CvCapture(@"..\..\Videos\drop.avi"))
{
    var image = capture.QueryFrame();
}
```

روش کلی کار با CvCapture را در اینجا ملاحظه می‌کنید. متد QueryFrame هربار یک frame از ویدیو را بازگشت می‌دهد و می‌توان آن‌را در یک حلقه، تا زمانیکه image نال بازگشت داده نشده، ادامه داد. همچنین برای نمایش آن نیز می‌توان از یکی از روش‌های مطرح شده ، مانند picture box استاندارد یا PictureBoxIpl (روش توصیه شده) استفاده کرد. اگر از PictureBoxIpl استفاده می‌کنید، متد pictureBoxIpl1.RefreshIplImage آن دقیقا برای یک چنین مواردی طراحی شده‌است تا سربار نمایش تصاویر را به حداقل برساند.
در اینجا اولین روشی که جهت به روز رسانی UI به نظر می‌رسد، استفاده از متد Application.DoEvents است تا UI فرصت داشته باشد، تعداد فریم‌های بالا را نمایش دهد و خود را به روز کند:

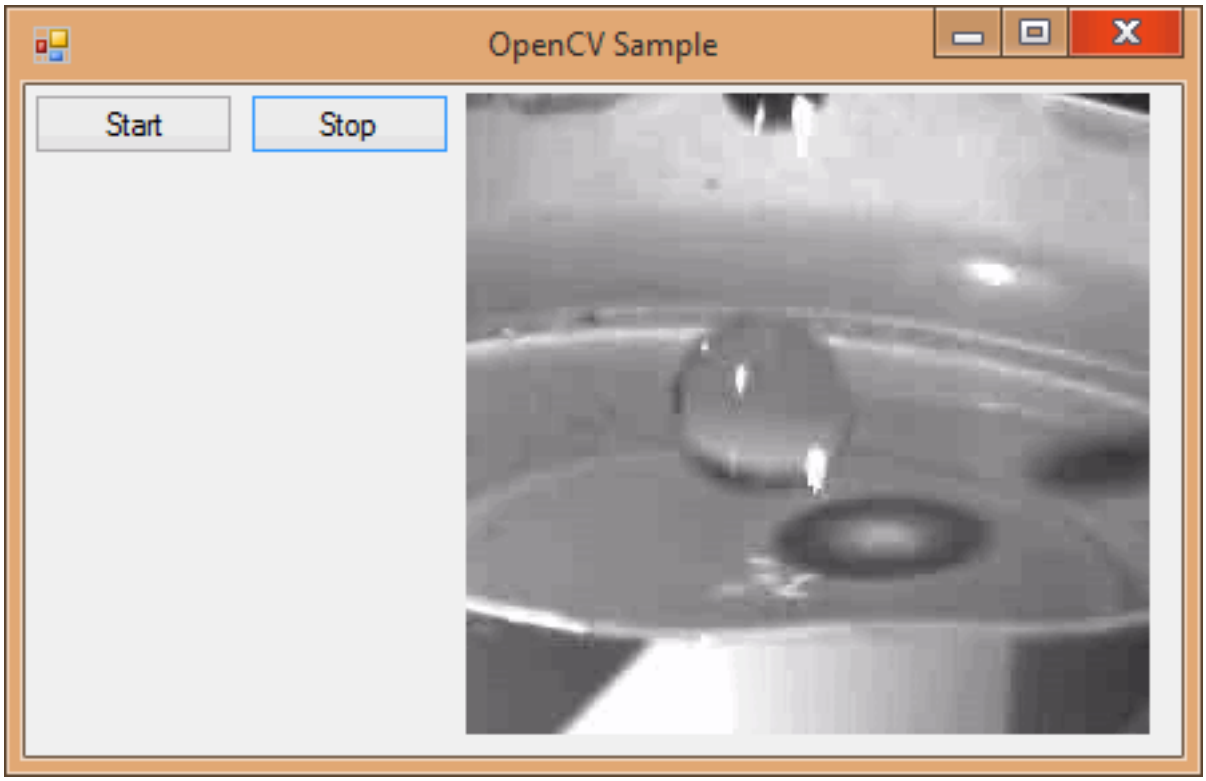
```
IplImage image;
while ((image = Capture.QueryFrame()) != null)
{
    _pictureBoxIpl1.RefreshIplImage(image);

    Thread.Sleep(interval);
    Application.DoEvents();
}
```

این روش هرچند کار می‌کند اما همانند روش استفاده از متد رخدادگردان Application Do Idle که صرفا در زمان بیکاری برنامه فراخوانی می‌شود، سبب خواهد شد تا تعدادی فریم را از دست دهید، همچنین با CPU Usage بالایی نیز مواجه شوید.
روش بعدی، استفاده از یک تایمر است که Interval آن بر اساس نرخ فریم‌های ویدیو تنظیم شده‌است:

```
timer = new Timer();
timer.Interval = (int)(1000 / Capture.Fps);
timer.Tick += Timer_Tick;
```

این روش بهتر است از روش DoEvents و به خوبی کار می‌کند؛ اما باز هم کار دریافت و همچنین پخش فریم‌ها، در ترد اصلی برنامه انجام خواهد شد.
روش بهتر از این، انتقال دریافت فریم‌ها به تردی جداگانه و پخش آن‌ها در ترد اصلی برنامه است؛ زیرا نمی‌توان GUI را از طریق یک ترد دیگر به روز رسانی کرد. برای این منظور می‌توان از BackgroundWorker دات نت کمک گرفت. رخداد DoWork آن در تردی جداگانه و مجزای از ترد اصلی برنامه اجرا می‌شود، اما رخداد ProgressChanged آن در ترد اصلی برنامه اجرا شده و امکان به روز رسانی UI را فراهم می‌کند.

استفاده از BackgroundWorker جهت پخش ویدیو به کمک OpenCVSharp



ابتدا دو دکمه‌ی Start و Stop را به فرم اضافه خواهیم کرد (شکل فوق).
سپس در زمان آغاز برنامه، یک PictureBoxIpl را به فرم جاری اضافه می‌کنیم:

```
private void FrmMain_Load(object sender, System.EventArgs e)
{
    _pictureBoxIpl1 = new PictureBoxIpl
    {
        AutoSize = true
    };
    flowLayoutPanel1.Controls.Add(_pictureBoxIpl1);
}
```

و یا همانطور که در قسمت پیشین نیز عنوان شد، می‌توانید این کنترل را به نوار ابزار VS.NET اضافه کرده و سپس به سادگی آن‌را روی فرم قرار دهید.

در دکمه‌ی Start، کار آغاز BackgroundWorker انجام خواهد شد:

```
private void BtnStart_Click(object sender, System.EventArgs e)
{
    if (_worker != null && _worker.IsBusy)
    {
        return;
    }

    _worker = new BackgroundWorker
    {
        WorkerReportsProgress = true,
        WorkerSupportsCancellation = true
    };
    _worker.DoWork += workerDoWork;
    _worker.ProgressChanged += workerProgressChanged;
    _worker.RunWorkerCompleted += workerRunWorkerCompleted;
```

```
    _worker.RunWorkerAsync();

    BtnStart.Enabled = false;
}
```

در اینجا یک سری خاصیت را مانند امکان لغو عملیات، جهت استفاده‌ی در دکمه‌ی Stop، به همراه تنظیم رخدادگردان‌هایی جهت دریافت و نمایش فریم‌ها تعریف کرده‌ایم. کدهای این روال‌های رخدادگردان را در ادامه ملاحظه می‌کنید:

```
private void workerDoWork(object sender, DoWorkEventArgs e)
{
    using (var capture = new CvCapture(@"..\..\Videos\drop.avi"))
    {
        var interval = (int)(1000 / capture.Fps);

        IplImage image;
        while ((image = capture.QueryFrame()) != null &&
                _worker != null && !_worker.CancellationPending)
        {
            _worker.ReportProgress(0, image);
            Thread.Sleep(interval);
        }
    }
}

private void workerProgressChanged(object sender, ProgressChangedEventArgs e)
{
    var image = e.UserState as IplImage;
    if (image == null) return;

    Cv.Not(image, image);
    _pictureBoxIpl1.RefreshIplImage(image);
}

private void workerRunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
{
    _worker.Dispose();
    _worker = null;
    BtnStart.Enabled = true;
}
```

متد workerDoWork کار دریافت فریم‌ها را در یک ترد مجزای از ترد اصلی برنامه به عهده دارد. این فریم‌ها توسط متد ReportProgress به متد workerProgressChanged جهت نمایش نهایی ارسال خواهند شد. این متد در ترد اصلی برنامه اجرا می‌شود و در اینجا کار با UI، مشکلی را به همراه نخواهد داشت و برنامه کرش نمی‌کند. اگر در متد workerDoWork کار به روز رسانی UI را مستقیما انجام دهیم، چون ترد اجرایی آن، با ترد اصلی برنامه یکی نیست، برنامه بلافاصله کرش خواهد کرد.
متد workerRunWorkerCompleted در پایان کار نمایش ویدیو، به صورت خودکار فراخوانی شده و در اینجا می‌توانیم دکمه‌ی Start را مجددا فعال کنیم.
همچنین در حین نمایش ویدیو، با کلیک بر روی دکمه‌ی Stop، می‌توان درخواست لغو عملیات را صادر کرد:

```
private void BtnStop_Click(object sender, System.EventArgs e)
{
    if (_worker != null)
    {
        _worker.CancelAsync();
        _worker.Dispose();
    }
    BtnStart.Enabled = true;
}
```

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید.

نظرات خوانندگان

نویسنده: علی ساری
تاریخ: ۱۳۹۴/۰۳/۱۷ ۹:۵۴

من از کدهای زیر استفاده کردم و در نهایت این خطا را در خط Application.Run(new Form1()); گرفتم

```
An unhandled exception of type 'System.Reflection.TargetInvocationException' occurred in mscorlib.dll

Additional information: Exception has been thrown by the target of an invocation.
```

کدهایی که در برنامه نوشتم:

```
private void button1_Click(object sender, EventArgs e)
        {
            if (_worker != null && _worker.IsBusy)
            {
                return;
            }

            _worker = new BackgroundWorker
            {
                WorkerReportsProgress = true,
                WorkerSupportsCancellation = true
            };
            _worker.DoWork += workerDoWork;
            _worker.ProgressChanged += workerProgressChanged;
            _worker.RunWorkerCompleted += workerRunWorkerCompleted;
            _worker.RunWorkerAsync();
        }
```

```
private void workerDoWork(object sender, DoWorkEventArgs e)
        {
            //var interval = (int)(1000 / _capture.Fps);
            Image image;
            while ((image = _capture.QueryFrame().ToBitmap()) != null &&
                    _worker != null && !_worker.CancellationPending)
            {
                _worker.ReportProgress(0, image);
                //Thread.Sleep(interval);

                Thread.Sleep(10);
            }
        }
        private void workerProgressChanged(object sender, ProgressChangedEventArgs e)
        {
            var image = e.UserState as Image;
            if (image == null) return;

            //Cv.Not(image, image);
            //_pictureBoxIpl1.RefreshIplImage(image);
            //_pictureBoxIpl1.Image=image;

            _pictureBoxIpl1.Invoke(new EventHandler(delegate
            {
                _pictureBoxIpl1.Image = image;
            }));


        }

        private void workerRunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
        {
            _worker.Dispose();
            _worker = null;
        }
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۱۷ ۱۰:۱۳

- وجود Thread Sleep با مقداری که در مطلب فوق عنوان شده، ضروری هست. از این جهت که اساسا رابط کاربری معمولی ویندوز، قابلیت پردازش تعداد عظیمی از پیام‌های رسیده را ندارد و باید در این بین به آن فرصت داد. بحث DirectShow مجموعه‌ی Direct-X متفاوت است و طراحی اختصاصی آن برای یک چنین کارهایی است. اما در اینجا نمی‌توانید UI معمولی را با سیلی از داده‌ها و پیام‌های به روز رسانی، مدفون کنید.
- استفاده از متد capture.QueryFrame().ToBitmap اشتباه هست. از این جهت که خروجی capture.QueryFrame می‌تواند نال باشد. بنابراین این تبدیل را باید در داخل حلقه انجام دهید و نه در زمانیکه قصد دارید تصویری را دریافت کنید. شرط موجود در حلقه (مانند مثال اصلی مطلب)، بررسی نال نبودن این فریم دریافتی است. بنابراین اگر نال باشد، حلقه پایان خواهد یافت.
- همانطور که در متن عنوان شد، متد workerProgressChanged در ترد اصلی یا همان ترد UI اجرا می‌شود. بنابراین فراخوانی pictureBoxIpl1.Invoke غیر ضروری است و سربار بی‌جهتی را به سیستم تحمیل می‌کند.
به صورت خلاصه در حین استفاده‌ی از BackgroundWorker:
- متد رخداد گردان DoWork بر روی ThreadPool اجرا می‌شود (ترد آن با ترد UI یکی نیست)
- متدهای رخدادگردان گزارش پیشرفت کار و اتمام کار، بر روی ترد UI اجرا می‌شوند. بنابراین امکان دسترسی به عناصر UI در این متدها، بدون مشکلی وجود دارد.

نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۴/۰۴/۰۵ ۱۱:۶

2 نکته و یک تجربه کوچک درباره نمایش ویدیو با خواندن اطلاعات از WebCam :
-اول اینکه اگر خواستید لیست از وب کم‌های سیستم تون داشته باشید از کد زیر استفاده کنید (البته برای استفاده از آن به DirectShow.Net dll نیاز دارید)

```
private void LoadCameras()
        {
            List<string> data = new List<string>();
            List<KeyValuePair<int, string>> ListCamerasData = new List<KeyValuePair<int, string>>();
            //-> Find systems cameras with DirectShow.Net dll
            DsDevice[] _SystemCamereas = DsDevice.GetDevicesOfCat(FilterCategory.VideoInputDevice);
            int _DeviceIndex = 0;
            foreach (DirectShowLib.DsDevice _Camera in _SystemCamereas)
            {
                ListCamerasData.Add(new KeyValuePair<int, string>(_DeviceIndex, _Camera.Name));
                data.Add(_Camera.Name);
                _DeviceIndex++;
            }

            CameraList.ItemsSource = data;
        }
```

-دوم اینکه برای نسبت دادن وب کم به CvCapture از متد CvCapture.FromCamera(cameraIndex) استفاده می‌کنیم :

```
using (CvCapture capture = CvCapture.FromCamera(cameraIndex))
            {
                //var interval = (int)(1000 / capture.Fps);
                IplImage image;
                while (_worker != null && !_worker.CancellationPending)
                {
                    if ((image = capture.QueryFrame()) != null)
                    {
                        _worker.ReportProgress(0, image);
                        Thread.Sleep(10);
                    }
                }
            }
```

این رو هم بگم که همین روش رو با بکارگیری محصور کننده Emgu انجام دادم و سرعت پایین‌تری نسبت به OpenCvSharp داشت.

و یک سوال : چرا در حین کار با وب کم مقدار خروجی capture.Fps یا همان frames per second مقدار صفر را بر می‌گرداند؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۴/۰۵ ۱۳:۱۵

برای کار با وب کم و دوربین دیجیتال، این مقدار را باید محاسبه کرد (شمارش تعداد فریم دریافتی در طی حداقل 3 ثانیه):

```
private static double getFps(CvCapture capture)
{
    double counter = 0;
    double seconds = 0;
    var watch = Stopwatch.StartNew();
    while (capture.QueryFrame() != null)
    {
        counter++;
        seconds = watch.ElapsedMilliseconds / (double)1000;
        if (seconds >= 3)
        {
            watch.Stop();
            break;
        }
    }
    var fps = counter / seconds;
    return fps;
}
```

و بعد برای استفاده :

```
using (var capture = CvCapture.FromCamera(index: 0))
{
    var fps = getFps(capture);
    capture.SetCaptureProperty(CvConst.CV_CAP_PROP_FPS, fps);
    var interval = (int)(1000 / fps);
```

نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۴/۰۴/۰۵ ۱۴:۴۲

با تشکر / راهکار خوبی بود.
در ضمن چون چند لحظه زمان می‌برد تا وب کم راه اندازی شود من قبل از کد زیر

```
while (capture.QueryFrame() != null) {
...
 }
```

کد زیر رو هم اضافه کردم تا شمارنده ثانیه بعد از دریافت تصویر شروع شود.

```
while (capture.QueryFrame() == null) { }
```

عنوان: **OpenCVSharp #7**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۱۷ ۱۳:۱۵
آدرس: www.dotnettips.info
گروه‌ها: OpenCV

معرفی اینترفیس C++ کتابخانه‌ی OpenCVSharp

اینترفیس یا API زبان C کتابخانه‌ی OpenCV مربوط است به نگارش‌های 1x این کتابخانه و تمام مثال‌هایی را که تاکنون ملاحظه کردید، بر مبنای همین اینترفیس تهیه شده بودند. اما از OpenCV سری 2x، این اینترفیس صرفا جهت سازگاری با نگارش‌های قبلی، نگهداری می‌شود و اینترفیس اصلی مورد استفاده، API جدید C++ آن است. به همین جهت کتابخانه‌ی OpenCVSharp نیز در فضای نام OpenCvSharp.CPlusPlus و توسط اسمبلی OpenCvSharp.CPlusPlus.dll، امکان دسترسی به این API جدید را فراهم کرده‌است که در ادامه نکات مهم آن‌را بررسی خواهیم کرد.

تبدیل مثال‌های اینترفیس C به اینترفیس C++

مثال « تبدیل تصویر به حالت سیاه و سفید » قسمت سوم را درنظر بگیرید. این مثال به کمک اینترفیس C کتابخانه‌ی OpenCV کار می‌کند. معادل تبدیل شده‌ی آن به اینترفیس C++ به صورت ذیل است:

```
// Cv2.ImRead
using (var src = new Mat(@"..\..\Images\Penguin.Png", LoadMode.AnyDepth | LoadMode.AnyColor))
using (var dst = new Mat())
{
    Cv2.CvtColor(src, dst, ColorConversion.BgrToGray);

    // How to export
    using (var bitmap = dst.ToBitmap()) // => OpenCvSharp.Extensions.BitmapConverter.ToBitmap(dst)
    {
        bitmap.Save("gray.png", ImageFormat.Png);
    }

    using (new Window("BgrToGray C++: src", image: src))
    using (new Window("BgrToGray C++: dst", image: dst))
    {
        Cv2.WaitKey();
    }
}
```

نکاتی را که باید در اینجا مدنظر داشت:
- بجای IplImage، از کلاس Mat استفاده شده‌است.
- برای ایجاد Clone یک تصویر نیازی نیست تا پارامترهای خاصی را به Mat دوم (همان dst) انتساب داد و ایجاد یک Mat خالی کفایت می‌کند.
- اینبار بجای کلاس Cv اینترفیس C، از کلاس Cv2 اینترفیس C++ استفاده شده‌است.
- متد الحاقی ToBitmap نیز که در کلاس OpenCvSharp.Extensions.BitmapConverter قرار دارد، با نمونه‌ی Mat سازگار است و به این ترتیب می‌توان خروجی معادل دات نتی Mat را با فرمت Bitmap تهیه کرد.
- بجای CvWindow، در اینجا باید از Window سازگار با Mat، استفاده شود.
- new Mat معادل Cv2.ImRead است. بنابراین اگر مثال C++ ایی را در اینترنت یافتید:

```
cv::Mat src = cv::imread ("foo.jpg");
cv::Mat dst;
cv::cvtColor (src, dst, CV_BGR2GRAY);
```

معادل متد imread آن همان new Mat کتابخانه‌ی OpenCVSharp است و یا متد Cv2.ImRead آن.

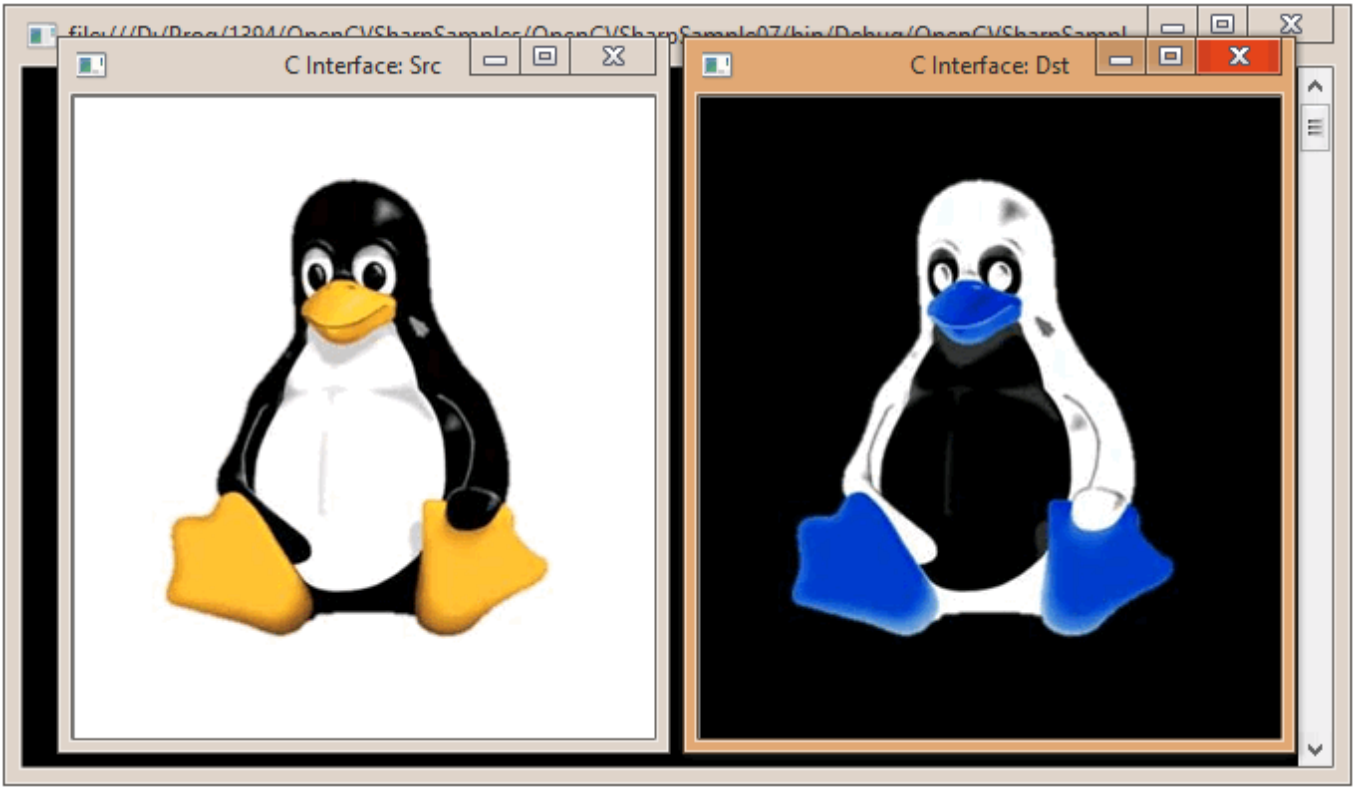
کار مستقیم با نقاط در OpenCVSharp

متدهای ماتریسی OpenCV، فوق العاده در جهت سریع اجرا شدن و استفاده‌ی از امکانات سخت افزاری و پردازش‌های موازی، بهینه سازی شده‌اند. اما اگر قصد داشتید این متدهای سریع را با نمونه‌هایی متداول و نه چندان سریع جایگزین کنید، می‌توان مستقیما با نقاط تصویر نیز کار کرد. در ادامه قصد داریم کار فیلتر توکار Not را که عملیات معکوس سازی رنگ نقاط را انجام می‌دهد، شبیه سازی کنیم.

در اینجا نحوه‌ی دسترسی مستقیم به نقاط تصویر بارگذاری شده را توسط اینترفیس C، ملاحظه می‌کنید:

```
using (var src = new IplImage(@"..\..\Images\Penguin.Png", LoadMode.AnyDepth | LoadMode.AnyColor))
using (var dst = new IplImage(src.Size, src.Depth, src.NChannels))
{
    for (var y = 0; y < src.Height; y++)
    {
        for (var x = 0; x < src.Width; x++)
        {
            CvColor pixel = src[y, x];
            dst[y, x] = new CvColor
            {
                B = (byte)(255 - pixel.B),
                G = (byte)(255 - pixel.G),
                R = (byte)(255 - pixel.R)
            };
        }
    }

    // [C] Accessing Pixel
    // https://github.com/shimat/opencvsharp/wiki/%5BC%5D-Accessing-Pixel

    using (new CvWindow("C Interface: Src", image: src))
    using (new CvWindow("C Interface: Dst", image: dst))
    {
        Cv.WaitKey(0);
    }
}
```

IplImage امکان دسترسی به نقاط را به صورت یک آرایه‌ی دو بعدی میسر می‌کند. خروجی آن از نوع CvColor است که در اینجا از هر عنصر آن، 255 واحد کسر خواهد شد تا فیلتر Not شبیه سازی شود. سپس این رنگ جدید، به نقطه‌ای معادل آن در تصویر خروجی انتساب داده می‌شود.
روش ارائه شده‌ی در اینجا یکی از روش‌های دسترسی به نقاط، توسط اینترفیس C است. سایر روش‌های ممکن را در Wiki آن می‌توانید مطالعه کنید.



شبیه به همین کار را می‌توان به نحو ذیل توسط اینترفیس C++ کتابخانه‌ی OpenCVSharp نیز انجام داد:

```
// Cv2.ImRead
using (var src = new Mat(@"..\..\Images\Penguin.Png", LoadMode.AnyDepth | LoadMode.AnyColor))
using (var dst = new Mat())
{
    src.CopyTo(dst);

    for (var y = 0; y < src.Height; y++)
    {
        for (var x = 0; x < src.Width; x++)
        {
            var pixel = src.Get<Vec3b>(y, x);
            var newPixel = new Vec3b
            {
                Item0 = (byte)(255 - pixel.Item0), // B
                Item1 = (byte)(255 - pixel.Item1), // G
                Item2 = (byte)(255 - pixel.Item2) // R
            };
            dst.Set(y, x, newPixel);
        }
    }

    // [Cpp] Accessing Pixel
    // https://github.com/shimat/opencvsharp/wiki/%5BCpp%5D-Accessing-Pixel

    //Cv2.NamedWindow();
    //Cv2.ImShow();
    using (new Window("C++ Interface: Src", image: src))
    using (new Window("C++ Interface: Dst", image: dst))
    {
        Cv2.WaitKey(0);
    }
}
```

ابتدا توسط کلاس Mat، کار بارگذاری و سپس تهیه‌ی یک کپی، از تصویر اصلی انجام می‌شود. در ادامه برای دسترسی به نقاط تصویر، از متد Get که خروجی آن از نوع Vec3b است، استفاده خواهد شد. این بردار دارای سه جزء است که بیانگر اجزای رنگ نقطه‌ی مدنظر می‌باشند. در اینجا نیز 255 واحد از هر جزء کسر شده و سپس توسط متد Set، به تصویر خروجی اعمال خواهند

شد.
می‌توانید سایر روش‌های دسترسی به نقاط را توسط اینترفیس C++، در Wiki این کتابخانه مطالعه نمائید.

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید.

عنوان: OpenCVSharp #8
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۱۹ ۱۲:۲۵
آدرس: www.dotnettips.info
گروه‌ها: OpenCV

بررسی morphology (ریخت شناسی) تصاویر

به تصویر زیر دقت کنید:

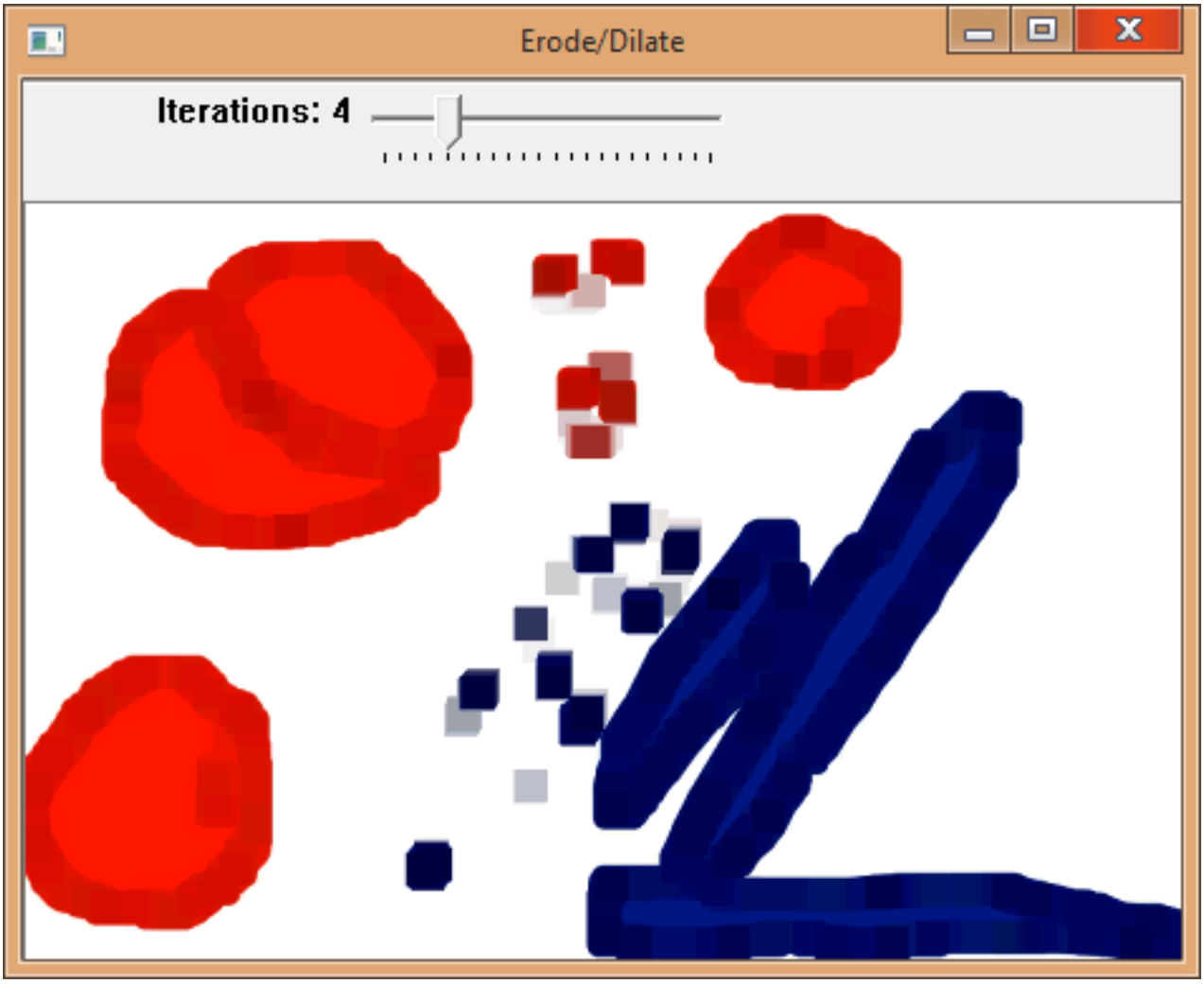
فرض کنید در اینجا قصد دارید تعداد توپ‌های قرمز را شمارش کنید. از دیدگاه یک انسان، شاید سه توپ قرمز قابل مشاهده باشد. اما از دیدگاه یک برنامه، توپ وسطی به دو توپ تفسیر خواهد شد و همچنین نویزهای قرمزی که بین توپ‌ها در صفحه وجود دارند نیز شمارش می‌شوند. بنابراین بهتر است پیش از پردازش این تصویر، ریخت شناسی آن‌را بهبود بخشید. برای مثال توپ وسطی را یکی کرد،حفره‌های توپ‌های دیگر را پوشاند و یا نویزهای قرمز را حذف نمود. به علاوه خطوط آبی رنگی را که با یکدیگر تماس یافته‌اند نیز می‌خواهیم اندکی از هم جدا کنیم.

متدهایی که مورفولوژی تصاویر را تغییر می‌دهند

در OpenCV سه متد یا فیلتر مهم، کار تغییر مورفولوژی تصاویر را انجام می‌دهند:

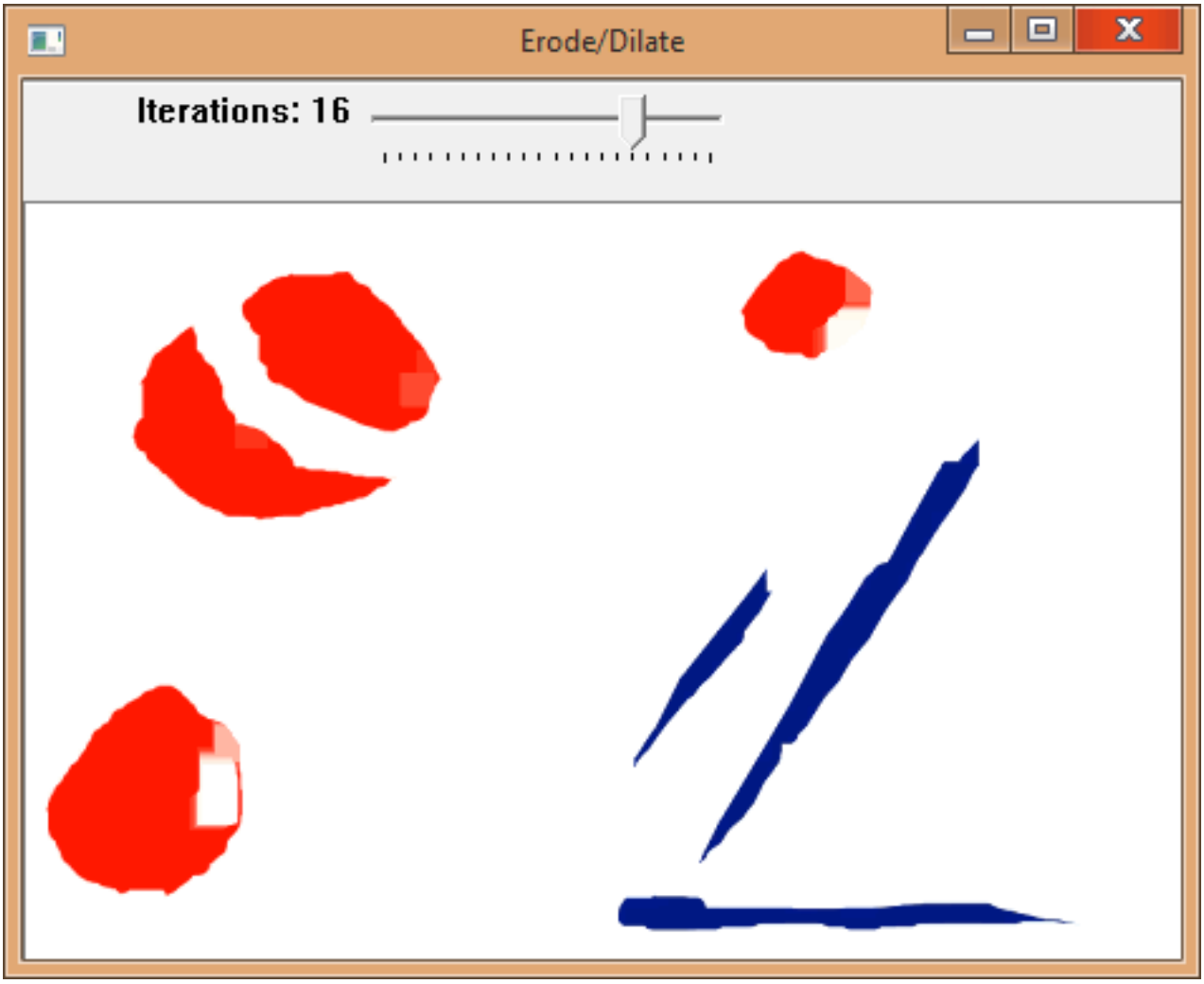
1) Cv2.Erode

تحلیل/فرسایش یا erosion سبب می‌شود تا نواحی تیره‌ی تصویر «رشد» کنند.



در اینجا فیلتر Erode کار یکی کردن اجزای جدای توپ‌های رنگی را انجام داده‌است.

2) Cv2.Dilate
اتساع یا dilation سبب خواهد شد تا نواحی روشن تصویر «رشد» کنند.

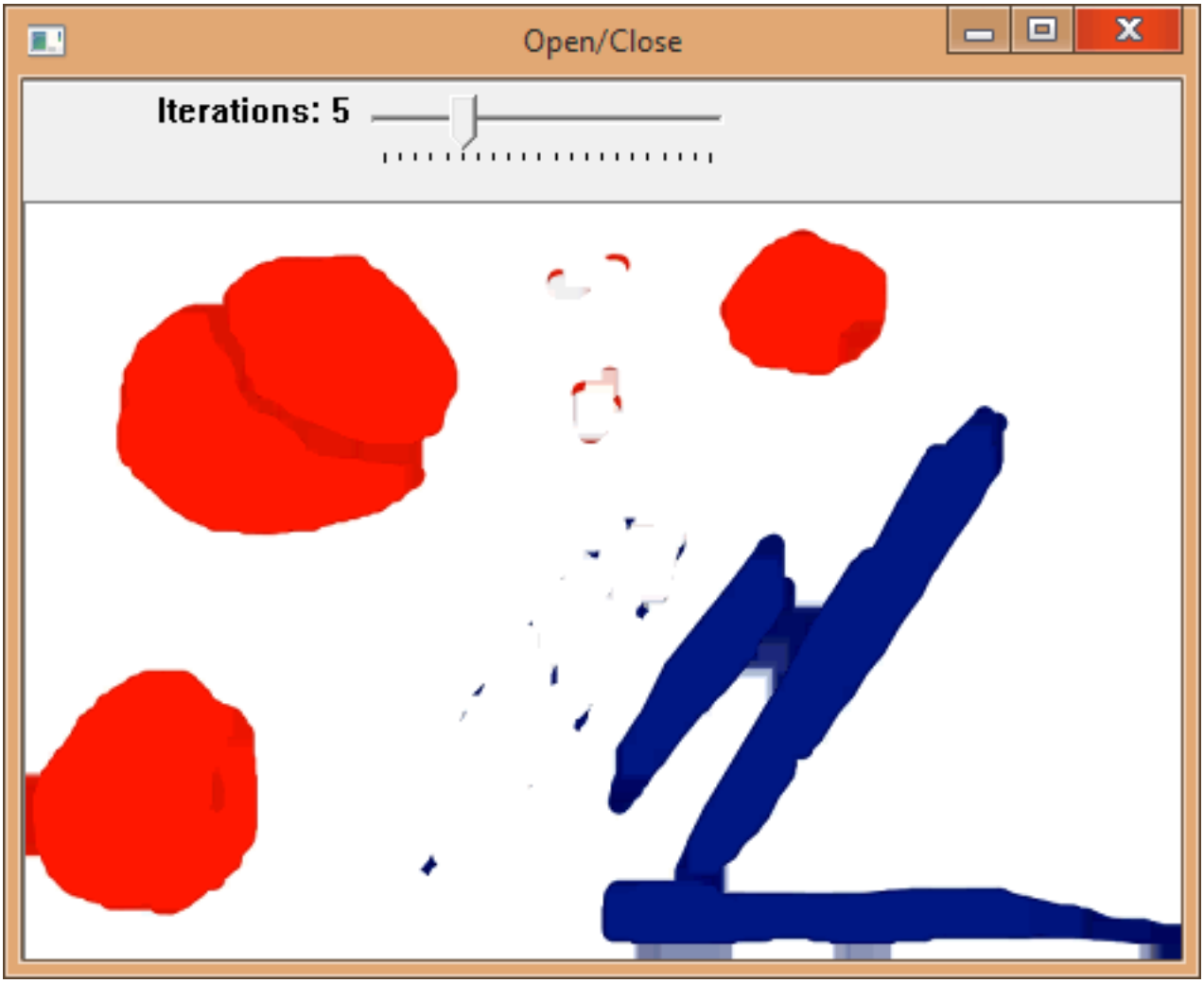


بکارگیری فیلتر Dilate سبب شده‌است تا نویزهای تصویر محو شوند و اشیاء به هم پیوسته از هم جدا گردند.

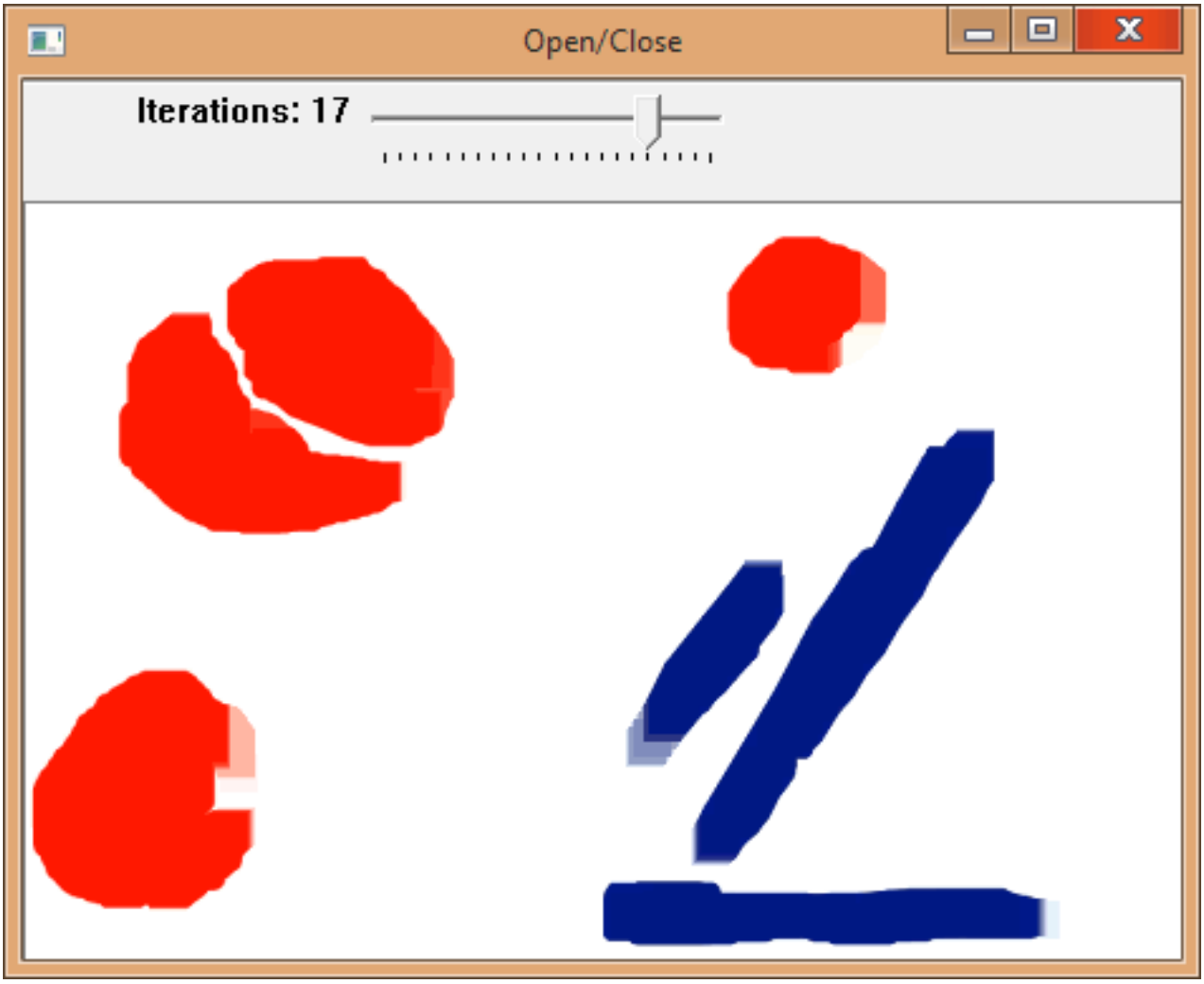
3) Cv2.MorphologyEx

کار این متد انجام اعمال پیشرفته‌ی مورفولوژی بر روی تصاویر است و در اینجا ترکیبی از erosion و dilation، با هم انجام می‌شوند.

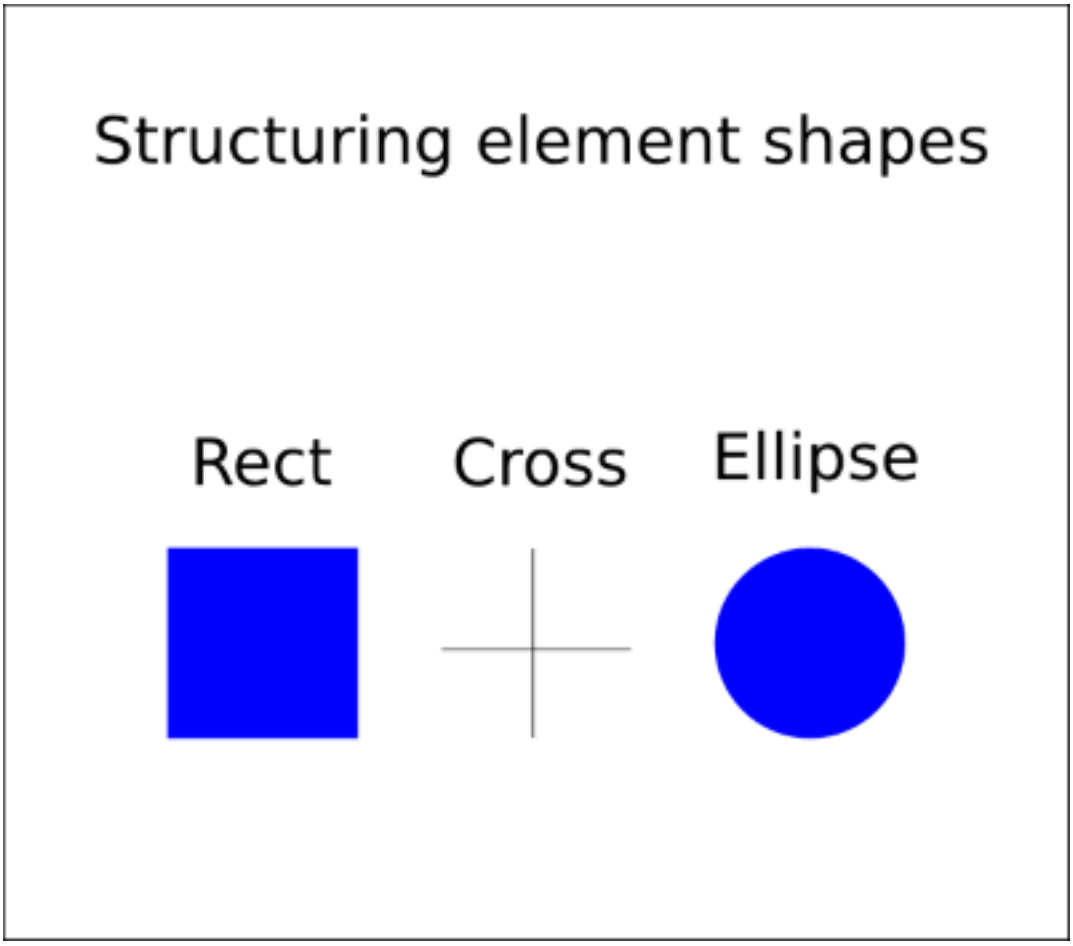
اگر پارامتر سوم آن به MorphologyOperation.Open تنظیم شود، ابتدا erosion و سپس dilation انجام خواهد شد:



و اگر این پارامتر به MorphologyOperation.Close مقدار دهی شود، ابتدا dilation و سپس erosion انجام می‌شود:



در تمام این حالات، پارامتر آخر که Structuring Element نام دارد، یکی از مقادیر اشیاء مستطیل، به علاوه و بیضی را می‌تواند داشته باشد. این اشیاء و اندازه‌ی آن‌ها، مشخص کننده‌ی میزان تحلیل و یا اتساع نهایی هستند.



استفاده از متدهای مورفولوژی در عمل

در اینجا مثالی را از نحوه‌ی بکارگیری متدهای اتساع و فرسایش، ملاحظه می‌کنید:

```
using (var src = new Mat(@"..\..\Images\cvmorph.Png", LoadMode.AnyDepth | LoadMode.AnyColor))
using (var dst = new Mat())
{
    src.CopyTo(dst);

    var elementShape = StructuringElementShape.Rect;
    var maxIterations = 10;

    var erodeDilateWindow = new Window("Erode/Dilate", image: dst);
    var erodeDilateTrackbar = erodeDilateWindow.CreateTrackbar(
        name: "Iterations", value: 0, max: maxIterations * 2 + 1,
        callback: pos =>
        {
            var n = pos - maxIterations;
            var an = n > 0 ? n : -n;
            var element = Cv2.GetStructuringElement(
                    elementShape,
                    new Size(an * 2 + 1, an * 2 + 1),
                    new Point(an, an));
            if (n < 0)
            {
                Cv2.Erode(src, dst, element);
            }
            else
            {
                Cv2.Dilate(src, dst, element);
            }

            Cv2.PutText(dst, (n < 0) ?
                string.Format("Erode[{0}]", elementShape) :
                string.Format("Dilate[{0}]", elementShape),
```

```
                new Point(10, 15), FontFace.HersheyPlain, 1, Scalar.Black);
            erodeDilateWindow.Image = dst;
        });

    Cv2.WaitKey();
    erodeDilateWindow.Dispose();
}
```

اینترفیس به کار گرفته شده، همان C++ API است و در اینجا ابتدا یک تصویر و کپی آن تهیه می‌شوند. سپس پنجره‌ی سازگار با C++ API ایجاد شده و به این پنجره یک شیء tracker اضافه می‌شود. این tracker یا slider، چندسکویی است. بدیهی است اگر قرار بود چنین کاری را صرفا با یک برنامه‌ی دات نتی انجام داد، می‌شد قسمت ایجاد پنجره و tracker آن‌را حذف کرد و بجای آن‌ها از یک picture box و یک slider به همراه مدیریت روال رخدادگردان تغییر مقادیر slider کمک گرفت. اما در اینجا تنها جهت آشنایی با این امکانات توکار OpenCV، از همان متدهای native آن استفاده شده‌است. در این مثال، روال رخدادگردان تغییر مقادیر tracker یا slider، یک function pointer است که معادل آن در دات نت یک delegate می‌باشد که در پارامتر callback متد ایجاد tracker قابل مشاهده است. هر بار که مقدار tracker تغییر می‌کند، مقدار pos را به callback خود ارسال خواهد کرد. از این مقدار جهت ایجاد شیء ساختاری و همچنین انتخاب بین حالات اتساع و فرسایش، کمک گرفته شده‌است.
کار متد PutText، نوشتن یک متن ساده بر روی پنجره‌ی native مربوط به OpenCV است.

همچنین در ادامه کدهای بکارگیری متد MorphologyEx را که کار ترکیب اتساع و فرسایش را با هم انجام می‌دهد، ذکر شده‌است و نکات بکارگیری آن همانند مثال اول بحث است:

```
using (var src = new Mat(@"..\..\Images\cvmorph.Png", LoadMode.AnyDepth | LoadMode.AnyColor))
    using (var dst = new Mat())
    {
        src.CopyTo(dst);

        var elementShape = StructuringElementShape.Rect;
        var maxIterations = 10;

        var openCloseWindow = new Window("Open/Close", image: dst);
        var openCloseTrackbar = openCloseWindow.CreateTrackbar(
            name: "Iterations", value: 0, max: maxIterations * 2 + 1,
            callback: pos =>
            {
                var n = pos - maxIterations;
                var an = n > 0 ? n : -n;
                var element = Cv2.GetStructuringElement(
                        elementShape,
                        new Size(an * 2 + 1, an * 2 + 1),
                        new Point(an, an));

                if (n < 0)
                {
                    Cv2.MorphologyEx(src, dst, MorphologyOperation.Open, element);
                }
                else
                {
                    Cv2.MorphologyEx(src, dst, MorphologyOperation.Close, element);
                }

                Cv2.PutText(dst, (n < 0) ?
                    string.Format("Open/Erosion followed by Dilation[{0}]", elementShape)
                    : string.Format("Close/Dilation followed by Erosion[{0}]", elementShape),
                    new Point(10, 15), FontFace.HersheyPlain, 1, Scalar.Black);
                openCloseWindow.Image = dst;
            });

        Cv2.WaitKey();
        openCloseWindow.Dispose();
    }
```

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید.

عنوان: OpenCVSharp #9
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۲۰ ۱۷:۳۰
آدرس: www.dotnettips.info
گروه‌ها: OpenCV

تغییر اندازه، و چرخش تصاویر

در OpenCV با استفاده از مفهومی به نام affine transform ، امکان تغییر اندازه و همچنین چرخش تصاویر میسر می‌شود. در اینجا، تصویر در یک ماتریس دو در سه ضرب می‌شود تا انتقالات یاد شده، انجام شوند.

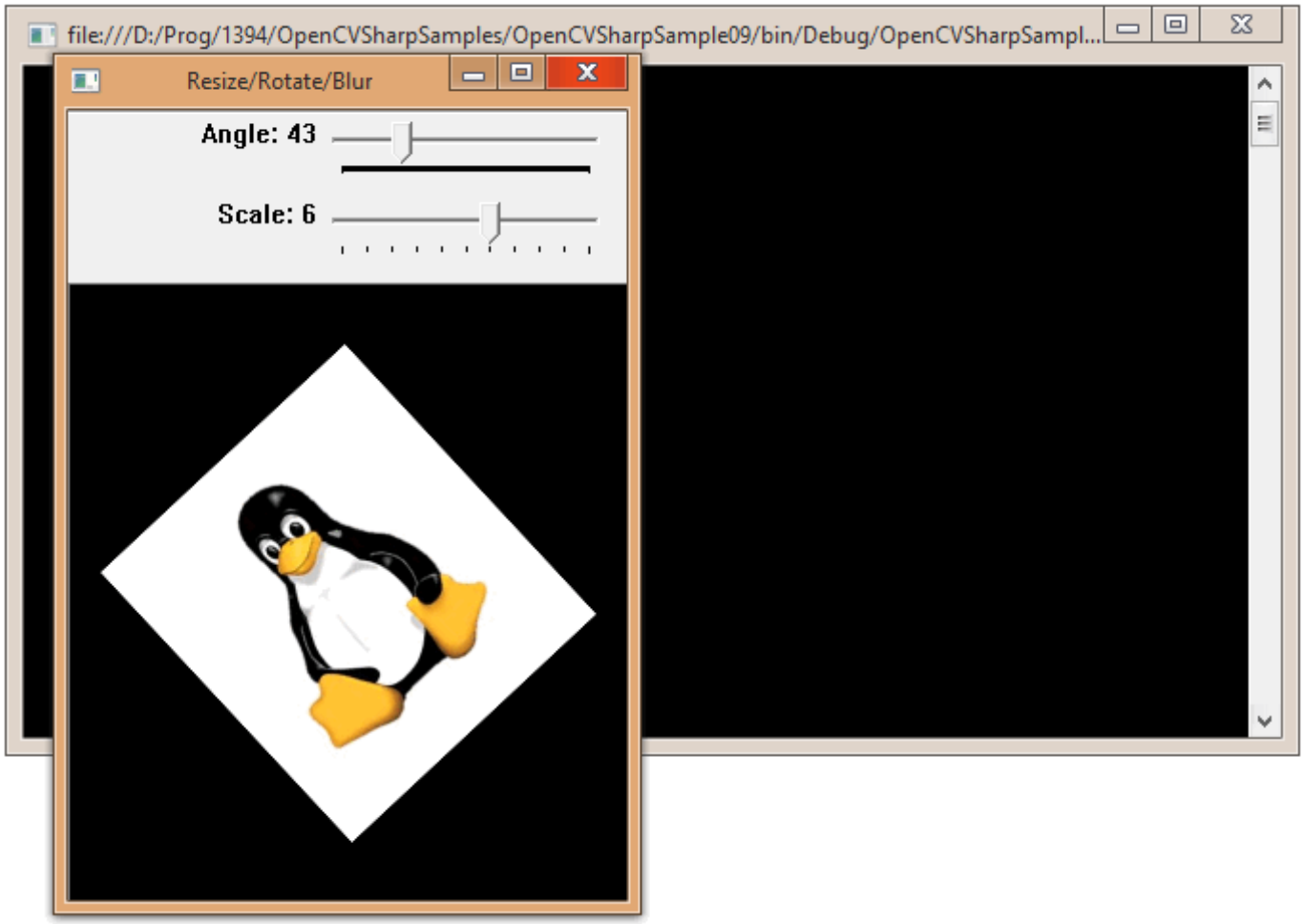
```
private static void rotateImage(double angle, double scale, Mat src, Mat dst)
{
    var imageCenter = new Point2f(src.Cols / 2f, src.Rows / 2f);
    var rotationMat = Cv2.GetRotationMatrix2D(imageCenter, angle, scale);
    Cv2.WarpAffine(src, dst, rotationMat, src.Size());
}
```

متد فوق کار چرخش تصویر مبدا (src) را به تصویر مقصد (dst) انجام می‌دهد. این عملیات توسط متد WarpAffine مدیریت شده و مهم‌ترین پارامتر آن، پارامتر سوم آن است که ماتریس تعریف کننده‌ی انتقالات تعریف شده توسط متد GetRotationMatrix2D است. در اینجا مرکز مشخص شده، زاویه و مقیاس، نحوه‌ی چرخش را تعریف می‌کنند.
برای مشاهده‌ی بهتر تاثیر پارامترهای مختلف در اینجا، به مثال ذیل دقت کنید:

```
using OpenCvSharp;
using OpenCvSharp.CPlusPlus;

namespace OpenCVSharpSample09
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var src = new Mat(@"..\..\Images\Penguin.Png", LoadMode.AnyDepth |
LoadMode.AnyColor))
            using (var dst = new Mat())
            {
                src.CopyTo(dst);

                using (var window = new Window("Resize/Rotate/Blur",
                                               image: dst, flags: WindowMode.AutoSize))
                {
                    var angle = 0.0;
                    var scale = 0.7;

                    var angleTrackbar = window.CreateTrackbar(
                        name: "Angle", value: 0, max: 180,
                        callback: pos =>
                        {
                            angle = pos;
                            rotateImage(angle, scale, src, dst);
                            window.Image = dst;
                        });

                    var scaleTrackbar = window.CreateTrackbar(
                        name: "Scale", value: 1, max: 10,
                        callback: pos =>
                        {
                            scale = pos / 10f;
                            rotateImage(angle, scale, src, dst);
                            window.Image = dst;
                        });

                    angleTrackbar.Callback.DynamicInvoke(0);
                    scaleTrackbar.Callback.DynamicInvoke(1);

                    Cv2.WaitKey();
                }
            }
        }

        private static void rotateImage(double angle, double scale, Mat src, Mat dst)
```

```
        {
            var imageCenter = new Point2f(src.Cols / 2f, src.Rows / 2f);
            var rotationMat = Cv2.GetRotationMatrix2D(imageCenter, angle, scale);
            Cv2.WarpAffine(src, dst, rotationMat, src.Size());
        }
    }
}
```

با این خروجی:



در این مثال، مانند مطلب قسمت قبل ، ابتدا یک پنجره‌ی سازگار با C++ API ایجاد شده و سپس دو tracker به آن اضافه شده‌اند. این trackers کار دریافت ورودی اطلاعات را از کاربر به عهده دارند (دریافت مقادیر زاویه‌ی چرخش و مقیاس) و مقادیر دریافتی از آن‌ها، در نهایت به متد rotateImage ارسال می‌شوند. این متد کار چرخش و تغییر مقیاس تصویر اصلی را انجام داده و نتیجه را به تصویر dst کپی می‌کند. در آخر تصویر dst در پنجره به روز شده و نمایش داده می‌شود.

تغییر اندازه‌ی تصاویر

اگر صرفا قصد تغییر اندازه‌ی تصاویر را دارید (بدون چرخش آن‌ها)، متد ویژه‌ای به نام Resize برای این منظور تدارک دیده شده‌است:

```
var resizeTrackbar = window.CreateTrackbar(
    name: "Resize", value: 1, max: 100,
```

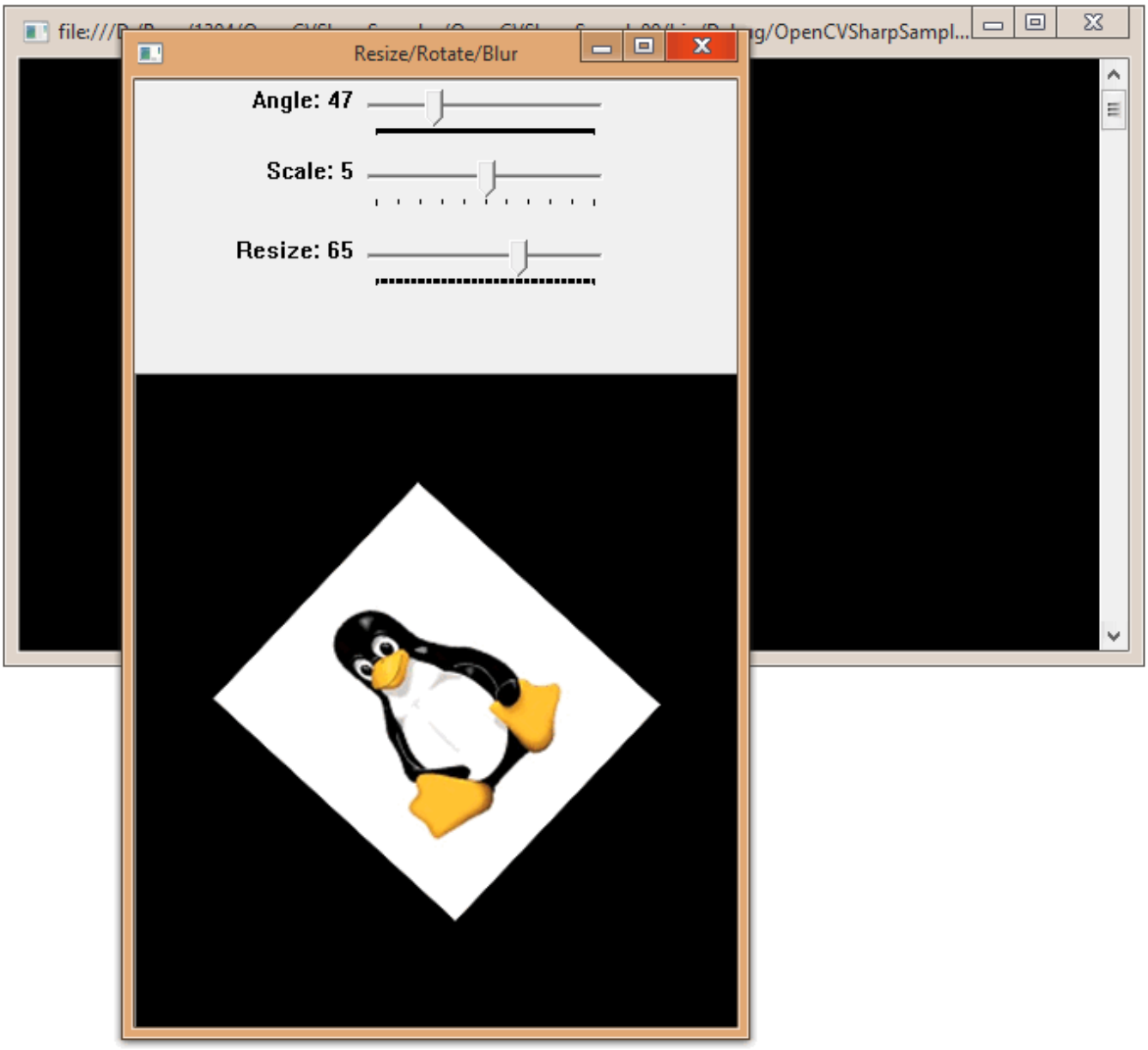
```
    callback: pos =>
    {
        Cv2.Resize(src, dst,
            new Size(src.Width + pos, src.Height + pos),
            interpolation: Interpolation.Cubic);
        window.Image = dst;
    });
```

در اینجا یک tracker دیگر به پنجره‌ی اصلی اضافه شده و توسط آن کار تعیین تغییر اندازه‌ی تصویر انجام می‌شود. نکته‌ی مهم این متد، امکان تعیین الگوریتم تغییر اندازه است که برای مثال در اینجا از Interpolation.Cubic استفاده شده‌است (احتمالا با این نام‌ها در برنامه‌های معروف کار با تصاویر، مانند فتوشاپ آشنایی دارید).

اگر می‌خواهید مقادیر پارامترهای چرخشی تصویر نیز در اینجا اعمال شوند، می‌توان به نحو ذیل عمل کرد:

```
var resizeTrackbar = window.CreateTrackbar(
    name: "Resize", value: 1, max: 100,
    callback: pos =>
    {
        rotateImage(angle, scale, src, dst);
        Cv2.Resize(dst, dst,
            new Size(src.Width + pos, src.Height + pos),
            interpolation: Interpolation.Cubic);
        window.Image = dst;
    });
```

در این کد ابتدا تصویر اصلی چرخش یافته و سپس در متد Resize از این تصویر چرخش یافته، به عنوان src استفاده می‌شود (هر دو پارامتر متد Resize به dst تنظیم شده‌اند).



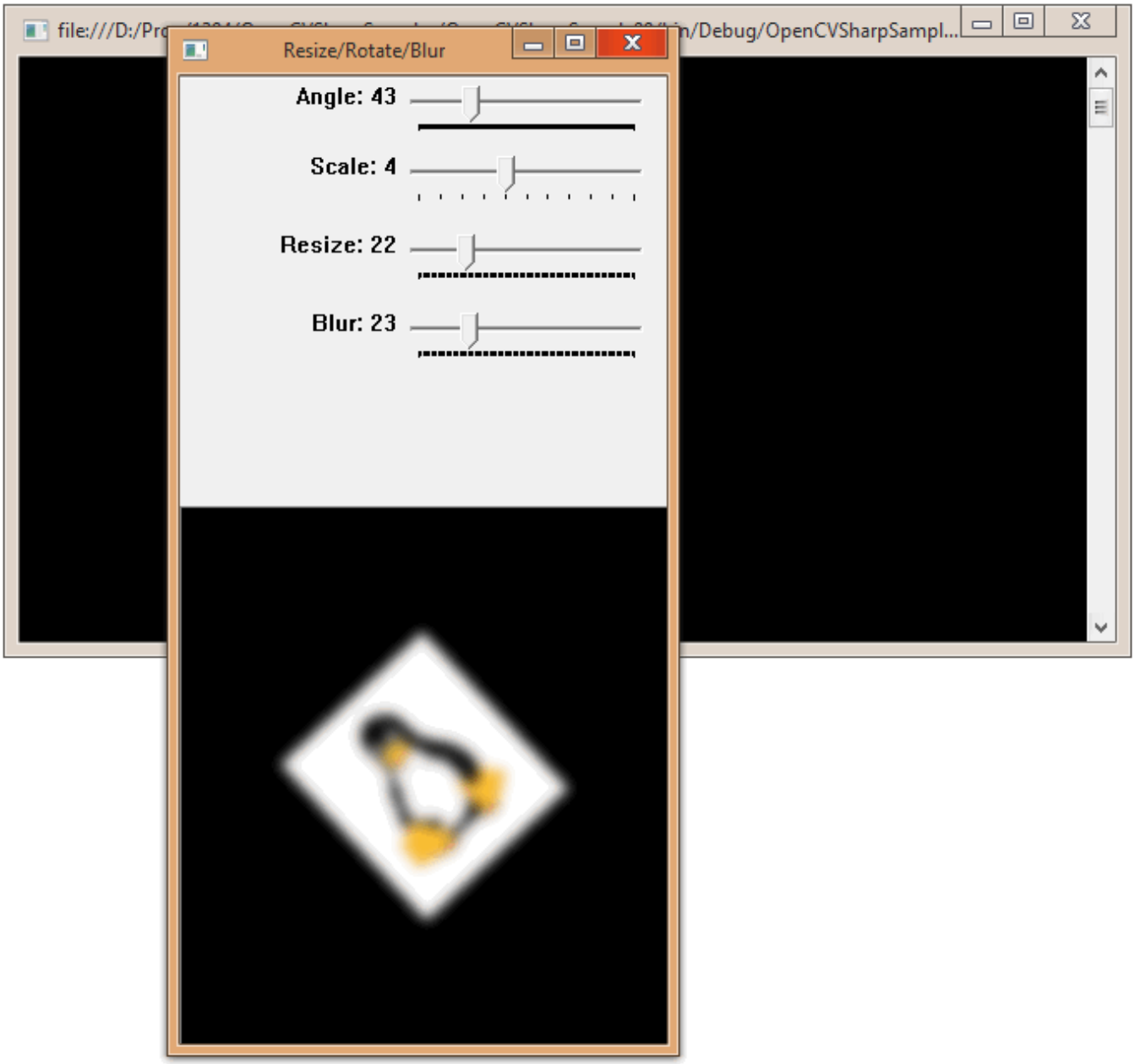
مات کردن تصاویر

در OpenCV با استفاده از متدهای GaussianBlur و یا medianBlur ، می‌توان تصاویر را مات کرد که نمونه‌ای از آن‌را در ادامه ملاحظه می‌کنید:

```
var blurTrackbar = window.CreateTrackbar(
   name: "Blur", value: 1, max: 100,
   callback: pos =>
   {
       if (pos % 2 == 0) pos++;

       rotateImage(angle, scale, src, dst);
       Cv2.GaussianBlur(dst, dst, new Size(pos, pos), sigmaX: 0);
       window.Image = dst;
   });
```

در اینجا ابتدا تصویر اصلی به متد چرخش تصویر ارسال شده و نتیجه‌ی آن در متد GaussianBlur استفاده خواهد شد. اندازه‌ی مشخص شده‌ی در این متد باید توسط اعداد فرد تعیین گردد. پارامتر sigmaX به معنای standard deviation در جهت x است و

اگر صفر تعیین شود، برای محاسبه‌ی آن از پارامتر اندازه‌ی تعیین شده کمک گرفته خواهد شد.



کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید.

عنوان: OpenCVSharp #10
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۲۱ ۱۶:۴۵
آدرس: www.dotnettips.info
گروه‌ها: OpenCV

محاسبه و ترسیم Histogram تصاویر

هیستوگرام یک تصویر ، توزیع میزان روشنایی آن تصویر را نمایش می‌دهد و در آن تعداد نقاط قسمت‌های روشن تصویر، ترسیم می‌شوند. محاسبه‌ی هیستوگرام تصاویر در حین دیباگ الگوریتم‌های پردازش تصویر، کاربرد زیادی دارند.
OpenCV به همراه متد توکاری است به نام cv::calcHist که قادر است هیستوگرام تعدادی آرایه را محاسبه کند و در C++ API آن قرار دارد. البته هدف اصلی این متد، انجام محاسبات مرتبط است و در اینجا قصد داریم این محاسبات را نمایش دهیم.

تغییر میزان روشنایی و وضوح تصاویر در OpenCV

همانطور که عنوان شد، کار هیستوگرام تصاویر، نمایش توزیع میزان روشنایی نقاط و اجزای آن‌ها است. بنابراین می‌توان جهت مشاهده‌ی تغییر هیستوگرام محاسبه شده با تغییر میزان روشنایی و وضوح تصویر، از متد ذیل کمک گرفت:

```
private static void updateBrightnessContrast(Mat src, Mat modifiedSrc, int brightness, int contrast)
{
    brightness = brightness - 100;
    contrast = contrast - 100;

    double alpha, beta;
    if (contrast > 0)
    {
        double delta = 127f * contrast / 100f;
        alpha = 255f / (255f - delta * 2);
        beta = alpha * (brightness - delta);
    }
    else
    {
        double delta = -128f * contrast / 100;
        alpha = (256f - delta * 2) / 255f;
        beta = alpha * brightness + delta;
    }
    src.ConvertTo(modifiedSrc, MatType.CV_8UC3, alpha, beta);
}
```

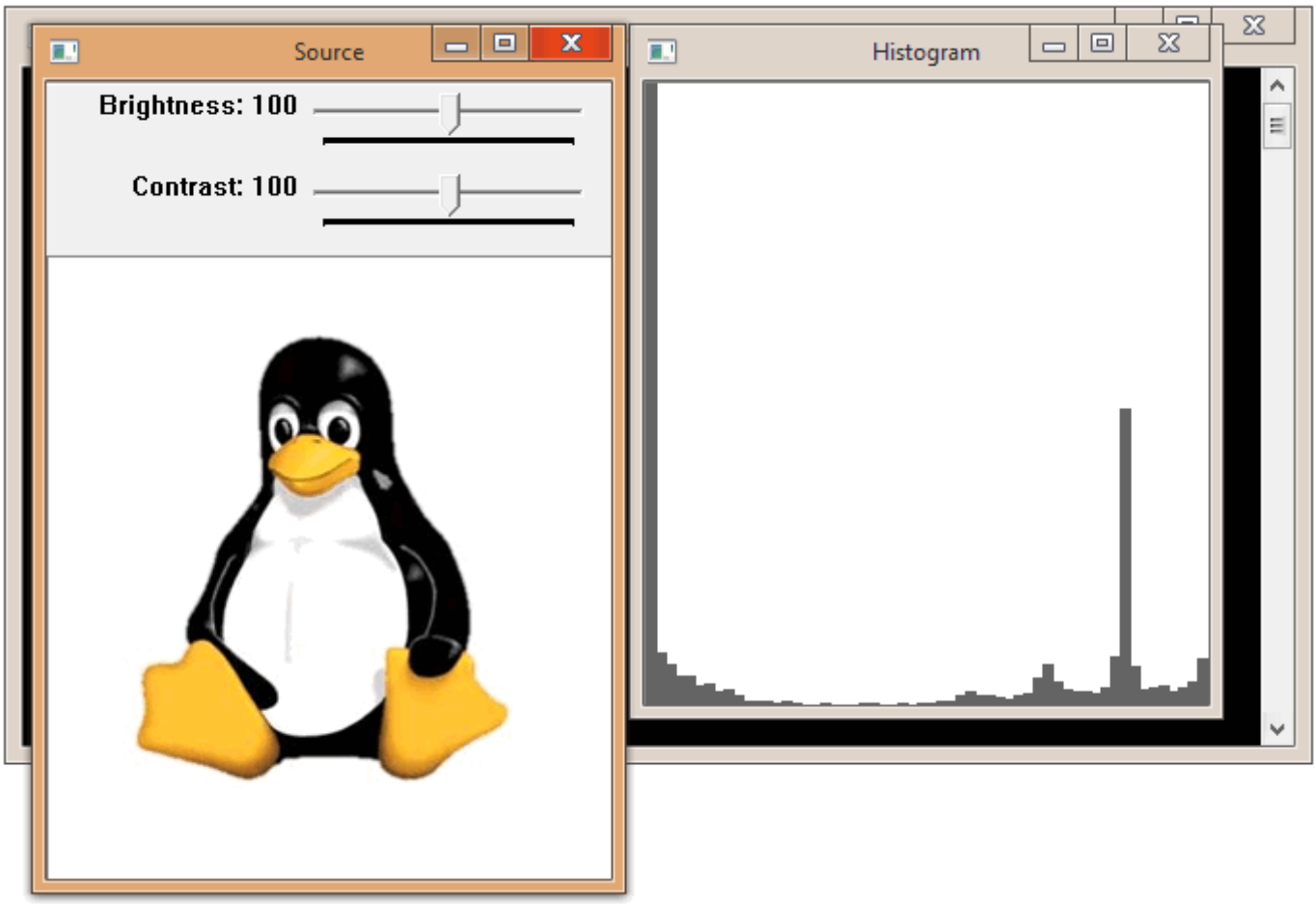
در اینجا src تصویر اصلی است. brightness و contrast، مقادیر میزان روشنایی و وضوح دریافتی از کاربر هستند. این مقادیر را می‌توان به متد ConvertTo ارسال کرد تا src را تبدیل به modifiedSrc نماید و وضوح و روشنایی آن‌را تغییر دهد.

پس از اینکه متد تغییر وضوح تصویر اصلی را تهیه کردیم، می‌توان به پنجره‌ی نمایش تصویر اصلی، دو tracker جهت دریافت brightness و contrast اضافه کرد و به این ترتیب امکان نمایش پویای تغییرات را مهیا نمود:

```
using (var src = new Mat(@"..\..\Images\Penguin.Png", LoadMode.AnyDepth | LoadMode.AnyColor))
{
    using (var sourceWindow = new Window("Source", image: src,
           flags: WindowMode.AutoSize | WindowMode.FreeRatio))
    {
        using (var histogramWindow = new Window("Histogram",
               flags: WindowMode.AutoSize | WindowMode.FreeRatio))
        {
            var brightness = 100;
            var contrast = 100;

            var brightnessTrackbar = sourceWindow.CreateTrackbar(
                    name: "Brightness", value: brightness, max: 200,
                    callback: pos =>
                    {
                        brightness = pos;
                        updateImageCalculateHistogram(sourceWindow, histogramWindow, src, brightness,
contrast);
                    });
```

```
            var contrastTrackbar = sourceWindow.CreateTrackbar(
                name: "Contrast", value: contrast, max: 200,
                callback: pos =>
                {
                    contrast = pos;
                    updateImageCalculateHistogram(sourceWindow, histogramWindow, src, brightness,
contrast);
                });


            brightnessTrackbar.Callback.DynamicInvoke(brightness);
            contrastTrackbar.Callback.DynamicInvoke(contrast);

            Cv2.WaitKey();
        }
    }
}
```

در اینجا src تصویر اصلی است. پنجره‌ی Source کار نمایش تصویر اصلی را به عهده دارد. همچنین به این پنجره، دو tracker اضافه شده‌اند تا کار دریافت مقادیر روشنایی و وضوح را از کاربر، مدیریت کنند.
پنجره‌ی دومی نیز به نام هیستوگرام در اینجا تعریف شده‌است. در این پنجره قصد داریم هیستوگرام تغییرات پویای تصویر اصلی را نمایش دهیم.



روش محاسبه‌ی هیستوگرام تصاویر و نمایش آن‌ها در OpenCVSharp

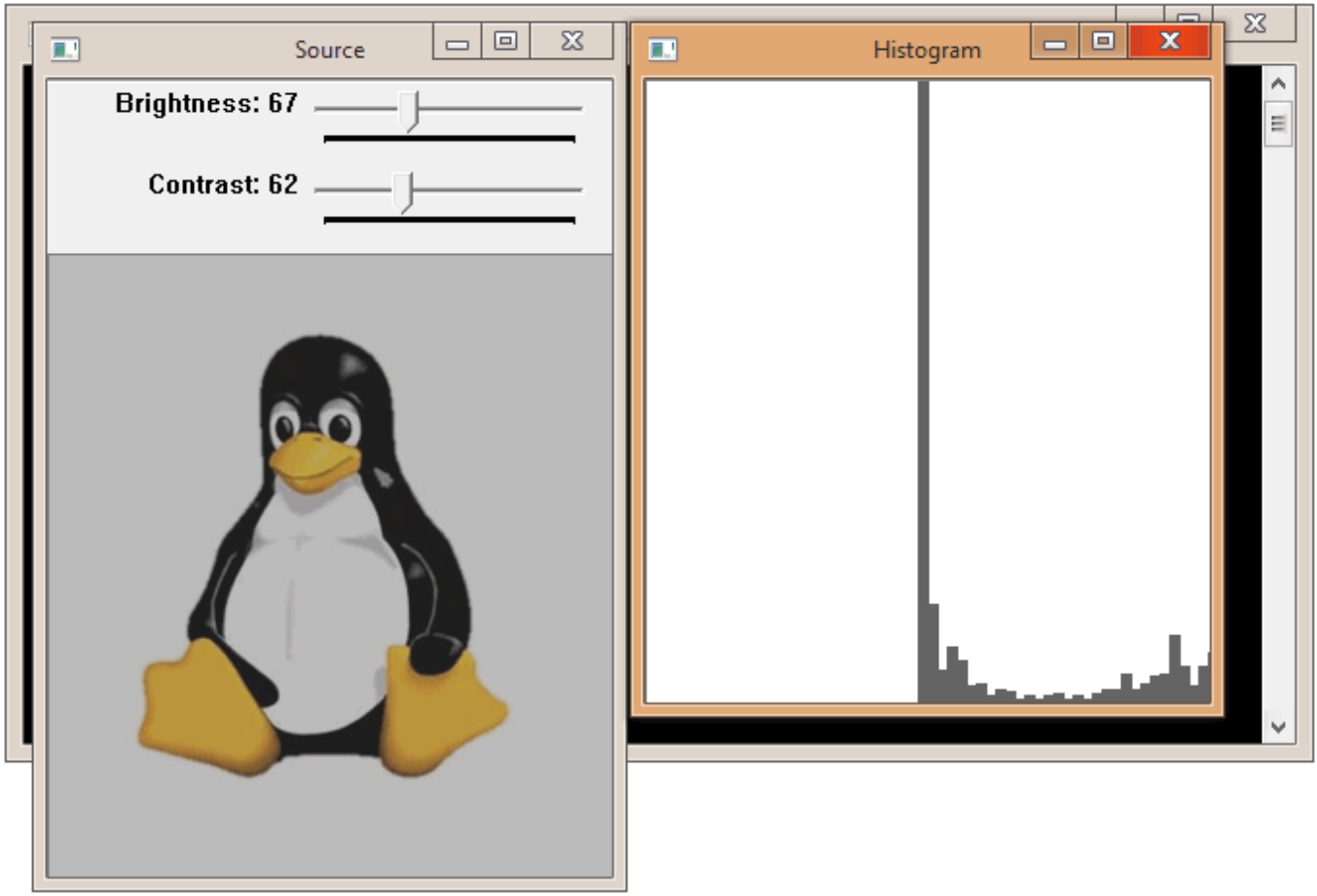
کدهای کامل محاسبه‌ی هیستوگرام تصویر اصلی تغییر یافته (modifiedSrc) و سپس نمایش آن‌را در پنجره‌ی histogramWindow،

در ادامه ملاحظه می‌کنید:

```
private static void calculateHistogram1(Window histogramWindow, Mat src, Mat modifiedSrc)
{
    const int histogramSize = 64;
    int[] dimensions = { histogramSize }; // Histogram size for each dimension
    Rangef[] ranges = { new Rangef(0, histogramSize) }; // min/max

    using (var histogram = new Mat())
    {
        Cv2.CalcHist(
            images: new[] { modifiedSrc },
            channels: new[] { 0 },
            mask: null,
            hist: histogram,
            dims: 1,
            histSize: dimensions,
            ranges: ranges);

        using (var histogramImage = (Mat)(Mat.Ones(rows: src.Rows, cols: src.Cols, type: MatType.CV_8U)
* 255))
        {
            // Scales and draws histogram

            Cv2.Normalize(histogram, histogram, 0, histogramImage.Rows, NormType.MinMax);
            var binW = Cv.Round((double)histogramImage.Cols / histogramSize);

            var color = Scalar.All(100);

            for (var i = 0; i < histogramSize; i++)
            {
                Cv2.Rectangle(histogramImage,
                    new Point(i * binW, histogramImage.Rows),
                    new Point((i + 1) * binW, histogramImage.Rows - Cv.Round(histogram.Get<float>(i))),
                    color,
                    -1);
            }

            histogramWindow.Image = histogramImage;
        }
    }
}
```

معادل متد cv::calcHist، متد Cv2.CalcHist در OpenCVSharp است. این متد آرایه‌ای از تصاویر را قبول می‌کند که در اینجا تنها قصد داریم با یک تصویر کار کنیم. به همین جهت آرایه‌های images، اندازه‌های آن‌ها و بازه‌های min/max این تصاویر تنها یک عضو دارند. خروجی این متد پارامتر hist آن است که توسط یک new Mat تامین شده‌است. مقدار dims به یک تنظیم شده‌است؛ زیرا در اینجا تنها قصد داریم شدت نقاط را اندازه گیری کنیم. پارامتر ranges مشخص می‌کند که مقادیر اندازه گیری شده باید در چه بازه‌ایی جمع آوری شوند.

پس از محاسبه‌ی هیستوگرام، یک تصویر خالی پر شده‌ی با عدد یک را توسط متد Mat.Ones ایجاد می‌کنیم. این تصویر به عنوان منبع تصویر هیستوگرام نمایش داده شده، مورد استفاده قرار می‌گیرد. سپس نیاز است اطلاعات محاسبه شده، در مقیاسی قرار گیرند که قابل نمایش باشد. به همین جهت با استفاده از متد Normalize، آن‌ها را در مقیاس و بازه‌ی ارتفاع تصویر، تغییر اندازه خواهیم داد. سپس به کمک متد مستطیل، خروجی آرایه هیستوگرام را در صفحه، با رنگ خاکستری مشخص شده توسط متد Scalar.All ترسیم خواهیم کرد.



همانطور که در این تصویر ملاحظه می‌کنید، با کدرتر شدن تصویر اصلی، هیستوگرام آن، توزیع روشنایی کمتری را نمایش می‌دهد.

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید.

عنوان: OpenCVSharp #11
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۲۴ ۱:۲۵
آدرس: www.dotnettips.info
گروه‌ها: OpenCV

خوشه بندی تصویر به کمک الگوریتم K-Means توسط OpenCV

الگوریتم k-Means clustering را می‌توان به کمک یک مثال بهتر بررسی کرد. فرض کنید شرکت منسوجاتی قرار است پیراهن‌های جدیدی را به بازار ارائه کند. بدیهی است برای فروش بیشتر، بهتر است پیراهن‌هایی را با اندازه‌های متفاوتی تولید کرد تا برای عموم مردم مفید باشد. اما ... برای این شرکت مقرون به صرفه نیست تا برای تمام اندازه‌های ممکن، پیراهن تولید کند. بنابراین اندازه‌های اشخاص را در سه گروه کوچک، متوسط و بزرگ تعریف می‌کند. این گروه بندی را می‌توان توسط الگوریتم k-means clustering نیز انجام داد و به کمک آن به سه اندازه‌ی بسیار مناسب رسید تا برای عموم اشخاص مناسب باشد. حتی اگر این سه گروه ناکافی باشند، این الگوریتم می‌تواند تعداد خوشه بندی‌های متغیری را دریافت کند تا بهینه‌ترین پاسخ حاصل شود. [برای مطالعه بیشتر]

ارتباط الگوریتم k-means clustering با مباحث پردازش تصویر، در پیش پردازش‌های لازمی است که جهت سرفصل‌هایی مانند تشخیص اشیاء، آنالیز صحنه، ردیابی و امثال آن ضروری هستند. از الگوریتم خوشه بندی k-means عموما جهت مفهومی به نام Color Quantization یا کاهش تعداد رنگ‌های تصویر استفاده می‌شود. یکی از مهم‌ترین مزایای این کار، کاهش فشار حافظه و همچنین بالا رفتن سرعت پردازش‌های بعدی بر روی تصویر است. همچنین گاهی از اوقات برای چاپ پوسترها نیاز است تعداد رنگ‌های تصویر را کاهش داد که در اینجا نیز می‌توان از این الگوریتم استفاده کرد.

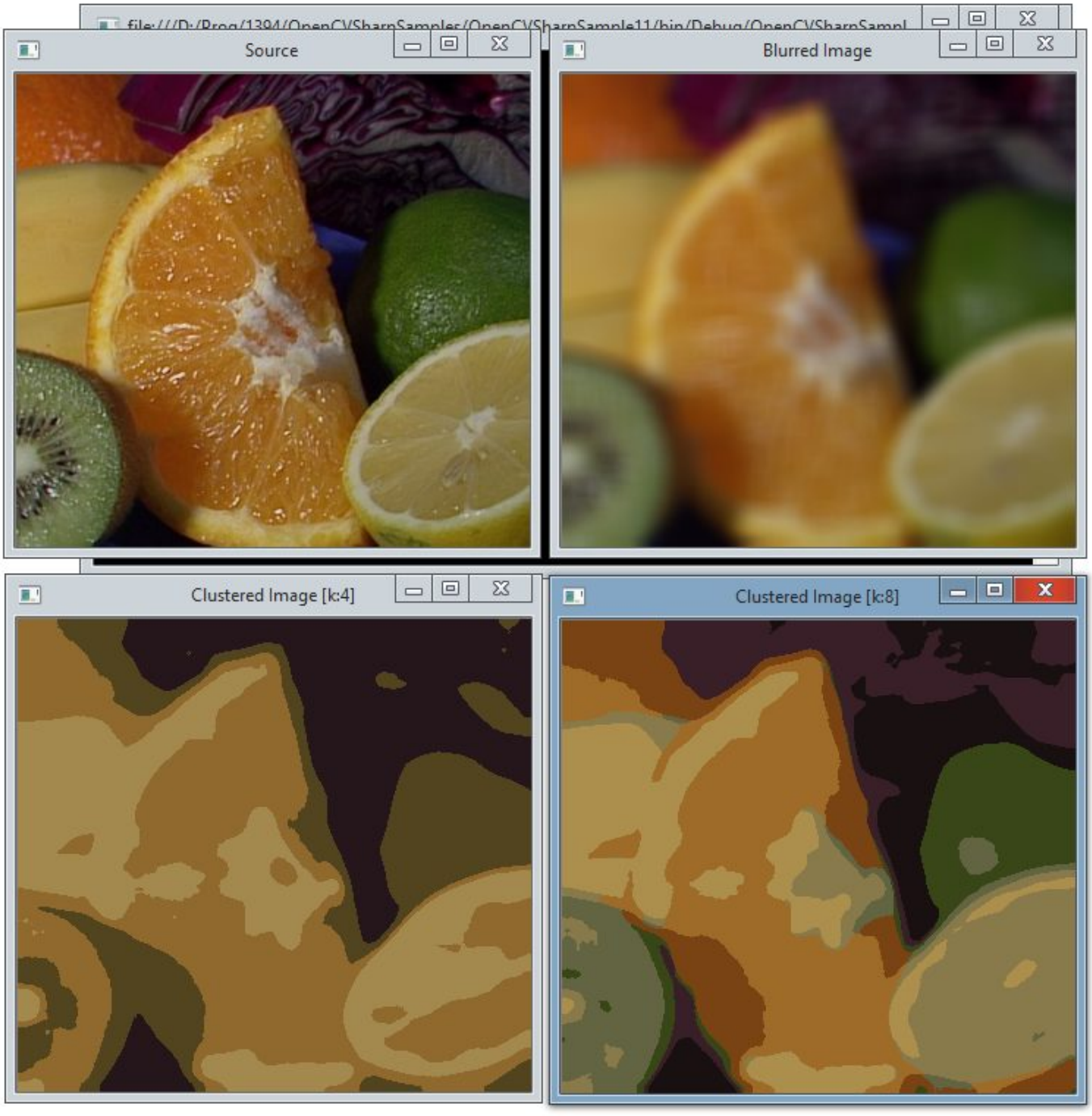
پیاده سازی الگوریتم خوشه بندی K-means

در ادامه کدهای بکارگیری متد kmeans کتابخانه‌ی OpenCV را به کمک OpenCVSharp مشاهده می‌کنید:

```
var src = new Mat(@"..\..\Images\fruits.jpg", LoadMode.AnyDepth | LoadMode.AnyColor);
Cv2.ImShow("Source", src);
Cv2.WaitKey(1); // do events

Cv2.Blur(src, src, new Size(15, 15));
Cv2.ImShow("Blurred Image", src);
Cv2.WaitKey(1); // do events

// Converts the MxNx3 image into a Kx3 matrix where K=MxN and
// each row is now a vector in the 3-D space of RGB.
// change to a Mx3 column vector (M is number of pixels in image)
var columnVector = src.Reshape(cn: 3, rows: src.Rows * src.Cols);

// convert to floating point, it is a requirement of the k-means method of OpenCV.
var samples = new Mat();
columnVector.ConvertTo(samples, MatType.CV_32FC3);

for (var clustersCount = 2; clustersCount <= 8; clustersCount += 2)
{
    var bestLabels = new Mat();
    var centers = new Mat();
    Cv2.Kmeans(
        data: samples,
        k: clustersCount,
        bestLabels: bestLabels,
        criteria:
            new TermCriteria(type: CriteriaType.Epsilon | CriteriaType.Iteration, maxCount: 10,
epsilon: 1.0),
        attempts: 3,
        flags: KMeansFlag.PpCenters,
        centers: centers);


    var clusteredImage = new Mat(src.Rows, src.Cols, src.Type());
    for (var size = 0; size < src.Cols * src.Rows; size++)
    {
        var clusterIndex = bestLabels.At<int>(0, size);
        var newPixel = new Vec3b
```

```
        {
            Item0 = (byte)(centers.At<float>(clusterIndex, 0)), // B
            Item1 = (byte)(centers.At<float>(clusterIndex, 1)), // G
            Item2 = (byte)(centers.At<float>(clusterIndex, 2)) // R
        };
        clusteredImage.Set(size / src.Cols, size % src.Cols, newPixel);
    }

    Cv2.ImShow(string.Format("Clustered Image [k:{0}]", clustersCount), clusteredImage);
    Cv2.WaitKey(1); // do events
}

Cv2.WaitKey();
Cv2.DestroyAllWindows();
```

با این خروجی



توضیحات

- ابتدا تصویر اصلی برنامه بارگذاری می‌شود و در یک پنجره نمایش داده خواهد شد. در اینجا متد Cv2.WaitKey را با پارامتر یک، مشاهده می‌کنید. این فراخوانی ویژه، شبیه به متد do events در برنامه‌های WinForms است. اگر فراخوانی نشود، تمام تصاویر پنجره‌های مختلف برنامه تا زمان پایان پردازش‌های مختلف برنامه، نمایش داده نخواهند شد و تا آن زمان صرفا یک یا چند پنجره‌ی خاکستری رنگ را مشاهده خواهید کرد.
- در ادامه متد Blur بر روی این تصویر فراخوانی شده‌است تا مقداری تصویر را مات کند. هدف از بکارگیری این متد در این مثال، برجسته کردن خوشه بندی گروه‌های رنگی مختلف در تصویر اصلی است.
- سپس متد Reshape بر روی ماتریس تصویر اصلی بارگذاری شده فراخوانی می‌شود.
هدف از بکارگیری الگوریتم k-means، انتساب برچسب‌هایی به هر نقطه‌ی RGB تصویر است. در اینجا هر نقطه به شکل یک بردار در فضای سه بعدی مشاهده می‌شود. سپس سعی خواهد شد تا این MxN بردار، به k قسمت تقسیم شوند.
متد Reshape تصویر اصلی MxNx3 را به یک ماتریس Kx3 تبدیل می‌کند که در آن K=MxN است و اکنون هر ردیف آن برداری است در فضای سه بعدی RGB.
- پس از آن توسط متد ConvertTo، نوع داده‌های این ماتریس جدید به float تبدیل می‌شوند تا در متد kmeans قابل استفاده شوند.
- در ادامه یک حلقه را مشاهده می‌کنید که عملیات کاهش رنگ‌های تصویر و خوشه بندی آن‌ها را 4 بار با مقادیر مختلف clustersCount انجام می‌دهد.
- در متد kmeans، پارامتر data یک ماتریس float است که هر نمونه‌ی آن در یک ردیف قرار گرفته‌است. K بیانگر تعداد خوشه‌ها، جهت تقسیم داده‌ها است.
در اینجا پارامترهای labels و centers خروجی‌های متد هستند. برچسب‌ها بیانگر اندیس‌های هر خوشه به ازای هر نمونه هستند. Centers ماتریس مراکز هر خوشه است و دارای یک ردیف به ازای هر خوشه است.
پارامتر criteria آن مشخص می‌کند که الگوریتم چگونه باید خاتمه یابد که در آن حداکثر تعداد بررسی‌ها و یا دقت مورد نظر مشخص می‌شوند.
پارامتر attempts مشخص می‌کند که این الگوریتم چندبار باید اجرا شود تا بهترین میزان فشردگی و کاهش رنگ حاصل شود.
- پس از پایان عملیات k-means نیاز است تا اطلاعات آن مجددا به شکل ماتریسی هم اندازه‌ی تصویر اصلی برگردانده شود تا بتوان آن‌را نمایش داد. در اینجا بهتر می‌توان نحوه‌ی عملکرد متد k-means را درک کرد. حلقه‌ی تشکیل شده به اندازه‌ی تمام نقاط طول و عرض تصویر اصلی است. به ازای هر نقطه، توسط الگوریتم k-means یک برچسب تشکیل شده (bestLabels) که مشخص می‌کند این نقطه متعلق به کدام خوشه و cluster رنگ‌های کاهش یافته است. سپس بر اساس این اندیس می‌توان رنگ این نقطه را از خروجی centers یافته و در یک تصویر جدید نمایش داد.

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید.

نظرات خوانندگان

نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۴/۰۴/۰۷ ۱۹:۶

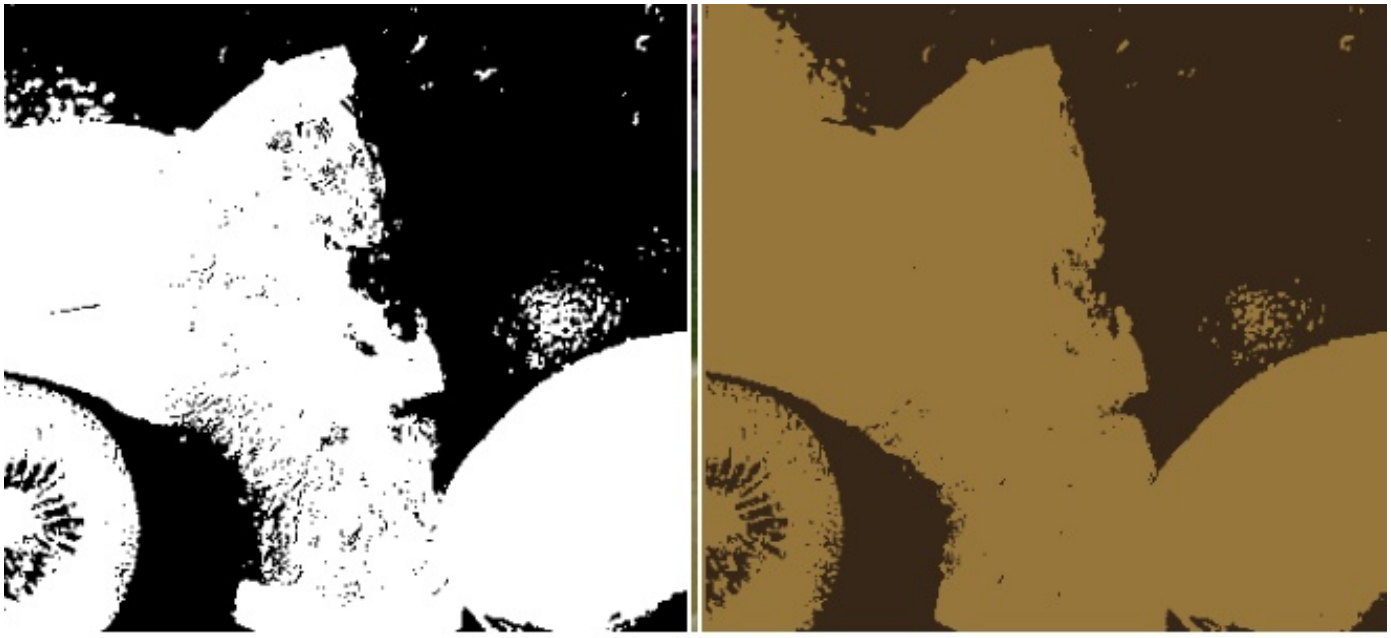
به طور مشخص با توجه به توضیحات بالا می‌توان گفت اگر مقدار k در الگوریتم k-means را برابر 2 در نظر بگیریم، تصویر ما به دو رنگ کوانتیزه می‌شود

```
using (var src = new Mat(@"test.jpg", LoadMode.Color))
            {
                image1.Source = KMeans(src, 2).ToWriteableBitmap();
                image2.Source = Binary(src).ToWriteableBitmap();
            }
```

که تقریبا با تصویر بدست آمده پس از باینری کردن آن معادل است

```
private static Mat Binary(Mat src)
        {
            // Convert the image to Gray
            src = src.CvtColor(ColorConversion.RgbToGray);
            // Convert the Gray to Binary with auto threshold
            Cv2.Threshold(src, src, 0, 255, ThresholdType.Binary | ThresholdType.Otsu);
            // Convert the Gray to Binary with manual threshold
            //Cv2.Threshold(src, src, 127, 255, ThresholdType.Binary);

            return src;
        }

        private static Mat KMeans(Mat src, int k)
        {
            var columnVector = src.Reshape(cn: 3, rows: src.Rows * src.Cols);
            var samples = new Mat();
            columnVector.ConvertTo(samples, MatType.CV_32FC3);

            var bestLabels = new Mat();
            var centers = new Mat();
            Cv2.Kmeans(
                data: samples,
                k: k,
                bestLabels: bestLabels,
                criteria: new TermCriteria(type: CriteriaType.Epsilon | CriteriaType.Iteration,
maxCount: 10, epsilon: 1.0),
                attempts: 3,
                flags: KMeansFlag.PpCenters,
                centers: centers);


            var dst = new Mat(src.Rows, src.Cols, src.Type());
            for (var size = 0; size < src.Cols * src.Rows; size++)
            {
                var clusterIndex = bestLabels.At<int>(0, size);
                var newPixel = new Vec3b
                {
                    Item0 = (byte)(centers.At<float>(clusterIndex, 0)), // B
                    Item1 = (byte)(centers.At<float>(clusterIndex, 1)), // G
                    Item2 = (byte)(centers.At<float>(clusterIndex, 2)) // R
                };
                dst.Set(size / src.Cols, size % src.Cols, newPixel);
            }

            return dst;
        }
```

نتایج :



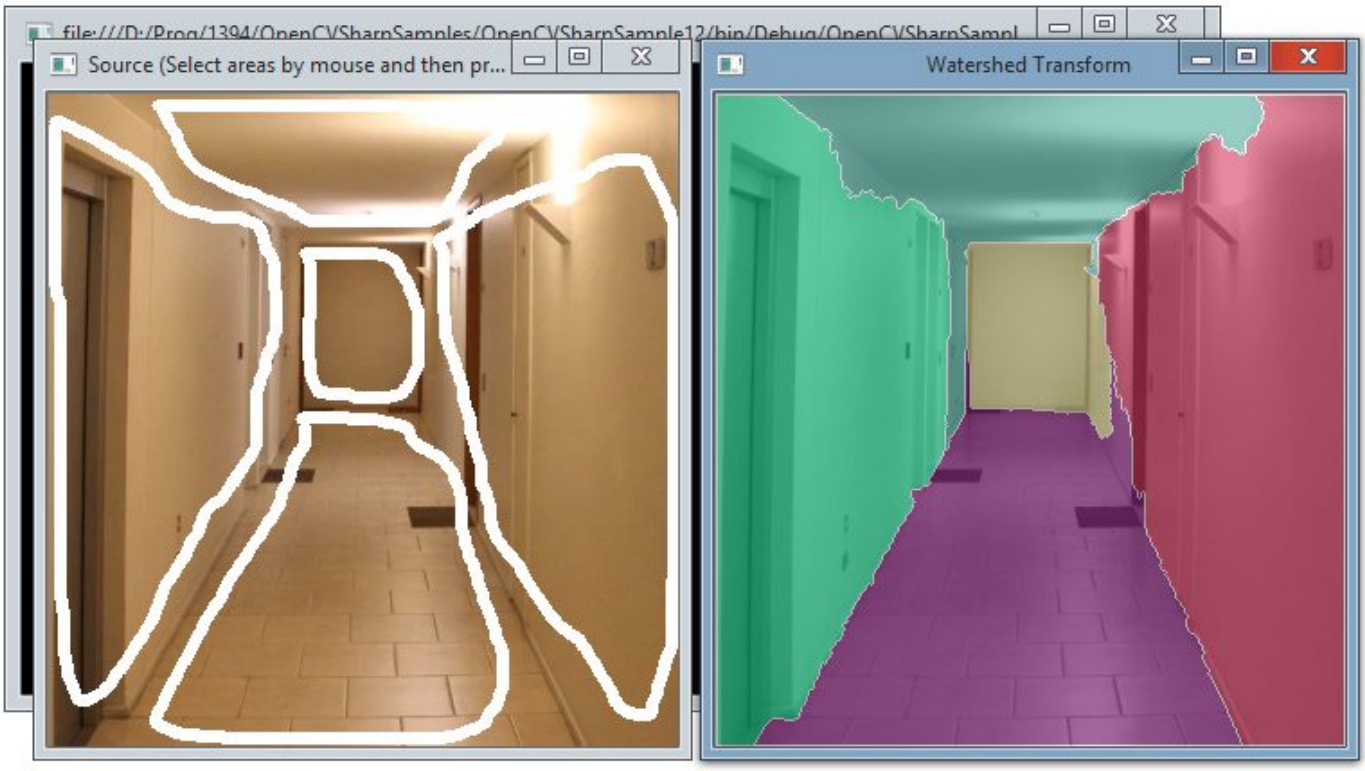
Binary

K-Means

عنوان: OpenCVSharp #12
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۲۵ ۱۱:۵۰
آدرس: www.dotnettips.info
گروه‌ها: OpenCV

قطعه بندی (segmentation) تصویر با استفاده از الگوریتم watershed

در تصویر ذیل، تصویر یک راه‌رو را مشاهده می‌کنید که توسط ماوس قطعه بندی شده‌است (تصویر اصلی یا سمت چپ). تصویر سمت راست، نسخه‌ی قطعه بندی شده‌ی این تصویر به کمک الگوریتم watershed است.



همانطور که در تصویر نیز مشخص است، نمایش هر ناحیه‌ی قطعه بندی شده، شبیه به سیلان آب است که با رسیدن به مرز قطعه‌ی بعدی متوقف شده‌است. به همین جهت به آن watershed (آب پخشان) می‌گویند.

انتخاب نواحی مختلف به کمک ماوس

در اینجا کدهای آغازین مثال بحث جاری را ملاحظه می‌کنید:

```
var src = new Mat(@"..\..\Images\corridor.jpg", LoadMode.AnyDepth | LoadMode.AnyColor);
var srcCopy = new Mat();
src.CopyTo(srcCopy);

var markerMask = new Mat();
Cv2.CvtColor(srcCopy, markerMask, ColorConversion.BgrToGray);

var imgGray = new Mat();
Cv2.CvtColor(markerMask, imgGray, ColorConversion.GrayToBgr);
markerMask = new Mat(markerMask.Size(), markerMask.Type(), s: Scalar.All(0));

var sourceWindow = new Window("Source (Select areas by mouse and then press space)")
{
    Image = srcCopy
};
```

```
var previousPoint = new Point(-1, -1);
sourceWindow.OnMouseCallback += (@event, x, y, flags) =>
{
    if (x < 0 || x >= srcCopy.Cols || y < 0 || y >= srcCopy.Rows)
    {
        return;
    }

    if (@event == MouseEvent.LButtonUp || !flags.HasFlag(MouseEvent.FlagLButton))
    {
        previousPoint = new Point(-1, -1);
    }
    else if (@event == MouseEvent.LButtonDown)
    {
        previousPoint = new Point(x, y);
    }
    else if (@event == MouseEvent.MouseMove && flags.HasFlag(MouseEvent.FlagLButton))
    {
        var pt = new Point(x, y);
        if (previousPoint.X < 0)
        {
            previousPoint = pt;
        }

        Cv2.Line(img: markerMask, pt1: previousPoint, pt2: pt, color: Scalar.All(255), thickness: 5);
        Cv2.Line(img: srcCopy, pt1: previousPoint, pt2: pt, color: Scalar.All(255), thickness: 5);
        previousPoint = pt;
        sourceWindow.Image = srcCopy;
    }
};
```

ابتدا تصویر راه‌رو بارگذاری شده‌است. سپس یک نسخه‌ی سیاه و سفید تک کاناله به نام markerMask از آن استخراج می‌شود. از آن برای ترسیم خطوط انتخاب نواحی مختلف تصویر به کمک ماوس استفاده می‌شود. به علاوه متد FindContours که در ادامه معرفی خواهد شد، نیاز به یک تصویر 8 بیتی تک کاناله دارد (به هر یک از اجزای RGB یک کانال گفته می‌شود).
همچنین این نسخه‌ی سیاه و سفید تک کاناله به یک تصویر سه کاناله برای نمایش رنگ‌های قسمت‌های مختلف قطعه بندی شده، تبدیل می‌شود.
سپس پنجره‌ی نمایش تصویر اصلی برنامه ایجاد شده و در اینجا روال رخدادگردان OnMouseCallback آن به صورت inline مقدار دهی شده‌است. در این روال می‌توان مدیریت ماوس را به عهده گرفت و کار نمایش خطوط مختلف را با فشرده شدن و سپس رها شدن کلیک سمت چپ ماوس انجام داد.
خط ترسیم شده بر روی دو تصویر از نوع Mat نمایش داده می‌شود. تصویر srcCopy، همان تصویر نمایش داده شده‌ی در پنجره‌ی اصلی است و تصویر markerMask، بیشتر جنبه‌ی محاسباتی دارد و در متدهای بعدی OpenCV استفاده خواهد شد.

تشخیص کانتورها (Contours) در تصویر

پس از ترسیم نواحی مورد نظر توسط ماوس، یک سری خطوط به هم پیوسته در شکل قابل مشاهده هستند. می‌خواهیم این خطوط را تشخیص داده و سپس از آن‌ها جهت محاسبات قطعه بندی تصویر استفاده کنیم. تشخیص این خطوط متصل، توسط متدی به نام FindContours انجام می‌شود. کانتورها، قسمت‌های خارجی اجزای متصل به هم هستند.

```
Point[][] contours; //vector<vector<Point>> contours;
HiearchyIndex[] hierarchyIndexes; //vector<Vec4i> hierarchy;
Cv2.FindContours(
    markerMask,
    out contours,
    out hierarchyIndexes,
    mode: ContourRetrieval.CComp,
    method: ContourChain.ApproxSimple);
```

متد FindContours همان تصویر markerMask را که توسط ماوس، قسمت‌های مختلف تصویر را علامتگذاری کرده‌است، دریافت می‌کند. سپس کانتورهای آن را استخراج خواهد کرد. کانتورها در مثال‌های اصلی OpenCV با verctor مشخص شده‌اند. در اینجا (در کتابخانه‌ی OpenCVSharp) آن‌ها را توسط یک آرایه‌ی دو بعدی از نوع Point مشاهده می‌کنید یا شبیه به لیستی از آرایه‌ی نقاط کانتورهای مختلف تشخیص داده شده (هر کانتور، آرایه‌ی از نقاط است). از hierarchyIndexes جهت یافتن و ترسیم این کانتورها

در متد DrawContours استفاده می‌شود.
متد FindContours یک تصویر 8 بیتی تک کاناله را دریافت می‌کند. اگر mode آن CCOMP یا FLOODFILL تعریف شود، امکان دریافت یک تصویر 32 بیتی را نیز خواهد داشت.
پارامتر hierarchy آن یک پارامتر اختیاری است که بیانگر اطلاعات topology تصویر است.
توسط پارامتر Mode، نحوه‌ی استخراج کانتور مشخص می‌شود. اگر به external تنظیم شود، تنها کانتورهای خارجی‌ترین قسمت‌ها را تشخیص می‌دهد. اگر مساوی list قرار گیرد، تمام کانتورها را بدون ارتباطی با یکدیگر و بدون تشکیل hierarchy استخراج می‌کند. حالت ccomp تمام کانتورها را استخراج کرده و یک درخت دو سطحی از آن‌ها را تشکیل می‌دهد. در سطح بالایی مرزهای خارجی اجزاء وجود دارند و در سطح دوم مرزهای حفره‌ها مشخص شده‌اند. حالت و مقدار tree به معنای تشکیل یک درخت کامل از کانتورهای یافت شده‌است.
پارامتر method اگر به none تنظیم شود، تمام نقاط کانتور ذخیره خواهند شد و اگر به simple تنظیم شود، قطعه‌های افقی، عمودی و قطری، فشرده شده و تنها نقاط نهایی آن‌ها ذخیره می‌شوند. برای مثال در این حالت یک کانتور مستطیلی، تنها با 4 نقطه ذخیره می‌شود.

ترسیم کانتورهای تشخیص داده شده بر روی تصویر

می‌توان به کمک متد DrawContours، مرزهای کانتورهای یافت شده را ترسیم کرد:

```
var markers = new Mat(markerMask.Size(), MatType.CV_32S, s: Scalar.All(0));

var componentCount = 0;
var contourIndex = 0;
while ((contourIndex >= 0))
{
    Cv2.DrawContours(
        markers,
        contours,
        contourIndex,
        color: Scalar.All(componentCount + 1),
        thickness: -1,
        lineType: LineType.Link8,
        hierarchy: hierarchyIndexes,
        maxLevel: int.MaxValue);

    componentCount++;
    contourIndex = hierarchyIndexes[contourIndex].Next;
}
```

پارامتر اول آن تصویری است که قرار است ترسیمات بر روی آن انجام شوند. پارامتر کانتور، آرایه‌ای است از کانتورهای یافت شده‌ی در قسمت قبل. پارامتر ایندکس مشخص می‌کند که اکنون کدام کانتور باید رسم شود. برای یافتن کانتور بعدی باید از hierarchyIndexes یافت شده‌ی توسط متد FindContours استفاده کرد. خاصیت Next آن، بیانگر ایندکس کانتور بعدی است و اگر مساوی منهای یک شد، کار متوقف می‌شود. مقدار maxLevel مشخص می‌کند که بر اساس پارامتر hierarchyIndexes، چند سطح از کانتورهای به هم مرتبط باید ترسیم شوند. در اینجا چون به حداکثر مقدار Int32 تنظیم شده‌است، تمام این سطوح ترسیم خواهند شد. اگر پارامتر ضخامت به یک عدد منفی تنظیم شود، سطوح داخلی کانتور ترسیم و پر می‌شوند.

اعمال الگوریتم watershed

در مرحله‌ی آخر، تصویر کانتورهای ترسیم شده را به متد Watershed ارسال می‌کنیم. پارامتر اول آن تصویر اصلی است و پارامتر دوم، یک پارامتر ورودی و خروجی محسوب می‌شود و کار قطعه بندی تصویر بر روی آن انجام خواهد شد.
کار الگوریتم watershed، ایزوله سازی اشیاء موجود در تصویر از پس زمینه‌ی آن‌ها است. این الگوریتم، یک تصویر سیاه و سفید را دریافت می‌کند؛ به همراه یک تصویر ویژه به نام marker. تصویر marker کارش مشخص سازی اشیاء، از پس زمینه‌ی آن‌ها است که در اینجا توسط ماوس ترسیم و سپس به کمک یافتن کانتورها و ترسیم آن‌ها بهینه سازی شده‌است.

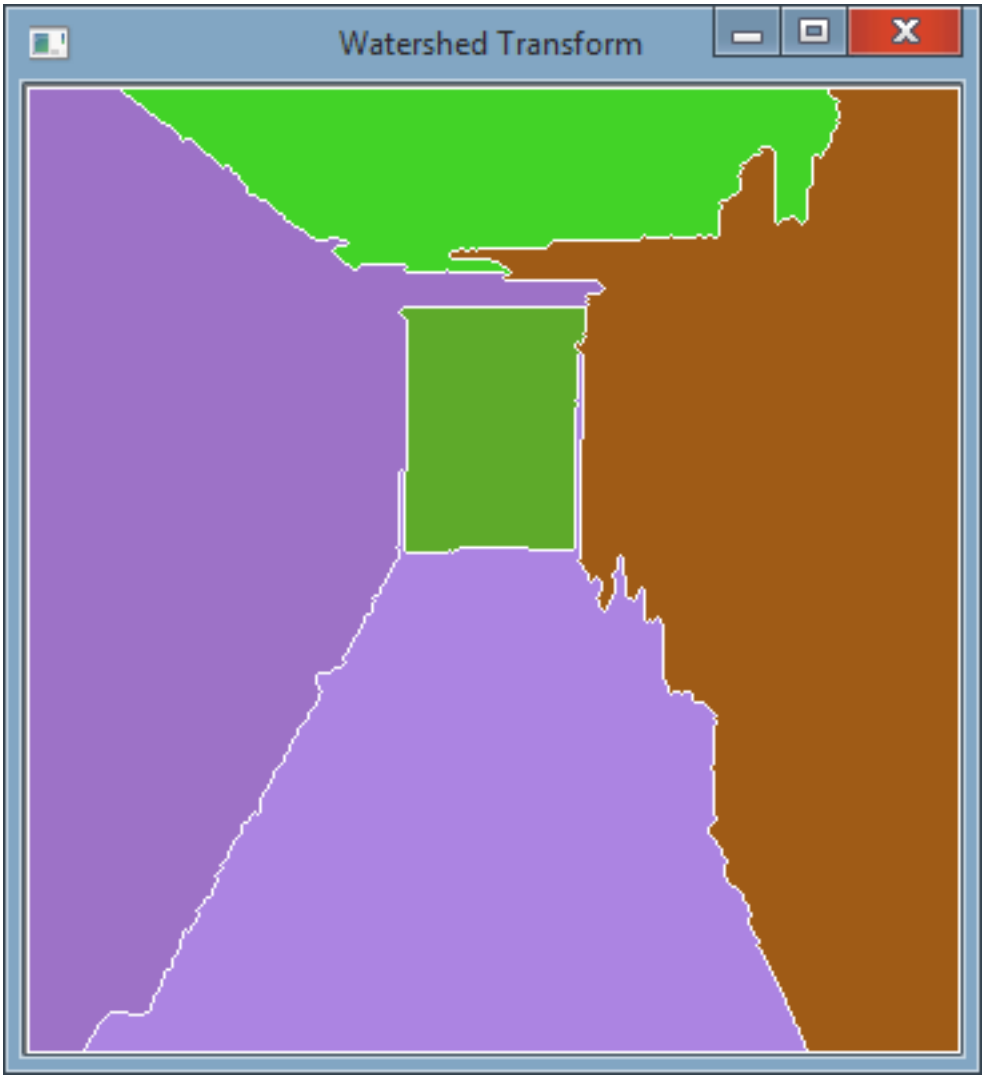
```
var rnd = new Random();
var colorTable = new List<Vec3b>();
for (var i = 0; i < componentCount; i++)
{
```

```
    var b = rnd.Next(0, 255); //Cv2.TheRNG().Uniform(0, 255);
    var g = rnd.Next(0, 255); //Cv2.TheRNG().Uniform(0, 255);
    var r = rnd.Next(0, 255); //Cv2.TheRNG().Uniform(0, 255);

    colorTable.Add(new Vec3b((byte)b, (byte)g, (byte)r));
}

Cv2.Watershed(src, markers);

var watershedImage = new Mat(markers.Size(), MatType.CV_8UC3);

// paint the watershed image
for (var i = 0; i < markers.Rows; i++)
{
    for (var j = 0; j < markers.Cols; j++)
    {
        var idx = markers.At<int>(i, j);
        if (idx == -1)
        {
            watershedImage.Set(i, j, new Vec3b(255, 255, 255));
        }
        else if (idx <= 0 || idx > componentCount)
        {
            watershedImage.Set(i, j, new Vec3b(0, 0, 0));
        }
        else
        {
            watershedImage.Set(i, j, colorTable[idx - 1]);
        }
    }
}

watershedImage = watershedImage * 0.5 + imgGray * 0.5;
Cv2.ImShow("Watershed Transform", watershedImage);
Cv2.WaitKey(1); //do events
```

متد Cv2.TheRNG یک تولید کننده‌ی اعداد تصادفی توسط OpenCV است و متد Uniform آن شبیه به متد Next کلاس Random دات نت عمل می‌کند. به نظر این کلاس تولید اعداد تصادفی، آنچنان هم تصادفی عمل نمی‌کند. به همین جهت از کلاس Random دات نت استفاده شد. در اینجا به ازای تعداد کانتورهای ترسیم شده، یک رنگ تصادفی تولید شده‌است.

پس از اعمال متد Watershed، هر نقطه‌ی تصویر marker مشخص می‌کند که متعلق به کدام قطعه‌ی تشخیص داده شده‌است. سپس به این نقطه، رنگ آن قطعه را نسبت داده و آن‌را در تصویر جدیدی ترسیم می‌کنیم.

در آخر، پس زمینه، با نواحی تشخیص داده ترکیب شده‌اند (watershedImage * 0.5 + imgGray * 0.5) تا تصویر ابتدای بحث حاصل شود. اگر این ترکیب صورت نگیرد، چنین تصویری حاصل خواهد شد:

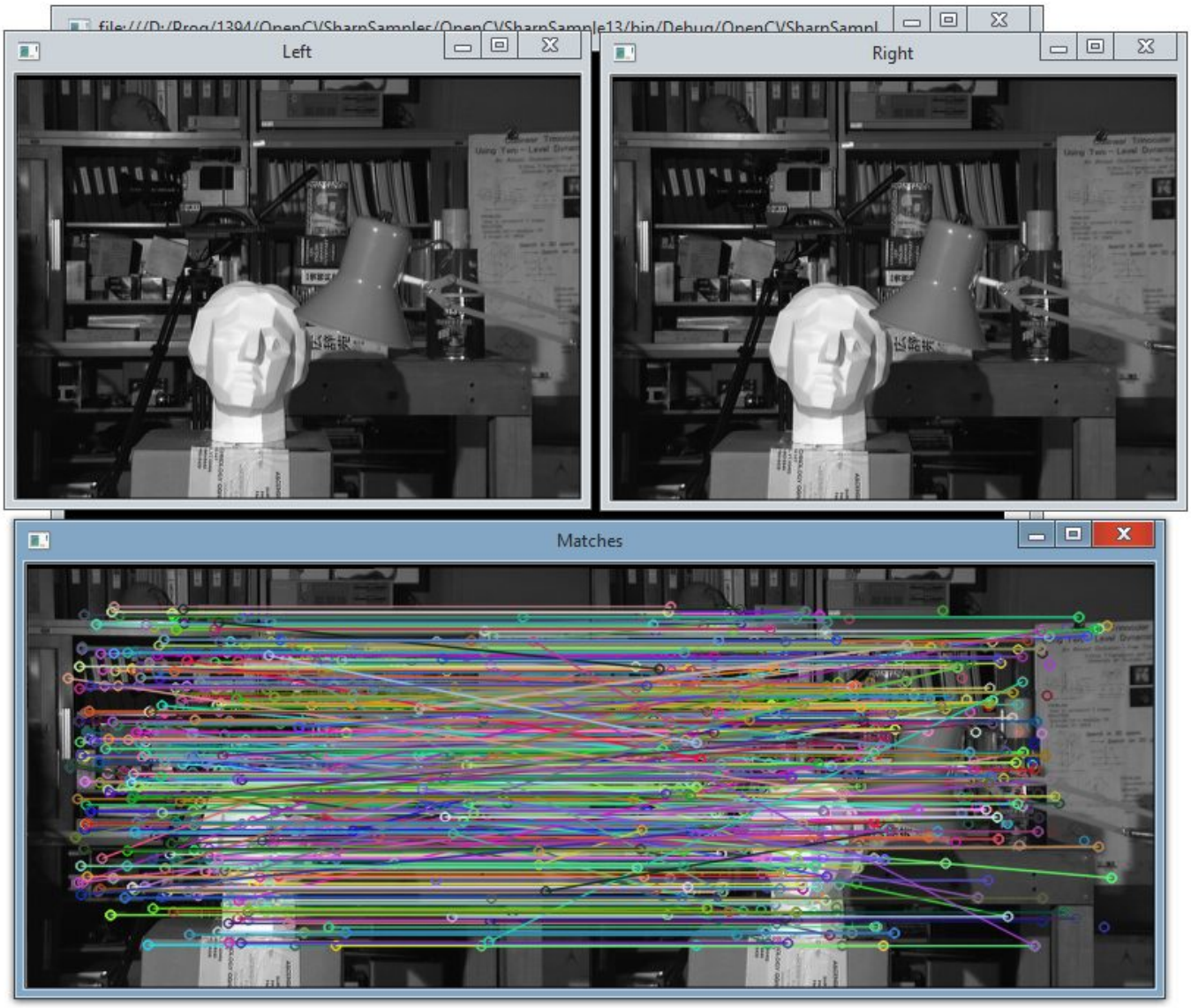


کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید.

عنوان: OpenCVSharp #13
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۲۶ ۱۴:۱۵
آدرس: www.dotnettips.info
گروه‌ها: OpenCV

تشخیص قسمت‌های مشابه تصاویر در OpenCV

در شکل زیر، دو تصویر سمت چپ و راست، اندکی با هم تفاوت دارند و در تصویر سوم، نقاط مشابه یافت شده‌ی توسط OpenCV ترسیم شده‌اند:



کدهای مثال فوق را در ذیل مشاهده می‌کنید:

```
var img1 = new Mat(@"..\..\Images\left.png", LoadMode.GrayScale);
Cv2.ImShow("Left", img1);
Cv2.WaitKey(1); // do events

var img2 = new Mat(@"..\..\Images\right.png", LoadMode.GrayScale);
Cv2.ImShow("Right", img2);
```

```
Cv2.WaitKey(1); // do events


// detecting keypoints
// FastFeatureDetector, StarDetector, SIFT, SURF, ORB, BRISK, MSER, GFTTDetector, DenseFeatureDetector,
SimpleBlobDetector
// SURF = Speeded Up Robust Features
var detector = new SURF(hessianThreshold: 400); //SurfFeatureDetector
var keypoints1 = detector.Detect(img1);
var keypoints2 = detector.Detect(img2);

// computing descriptors, BRIEF, FREAK
// BRIEF = Binary Robust Independent Elementary Features
var extractor = new BriefDescriptorExtractor();
var descriptors1 = new Mat();
var descriptors2 = new Mat();
extractor.Compute(img1, ref keypoints1, descriptors1);
extractor.Compute(img2, ref keypoints2, descriptors2);

// matching descriptors
var matcher = new BFMatcher();
var matches = matcher.Match(descriptors1, descriptors2);

// drawing the results
var imgMatches = new Mat();
Cv2.DrawMatches(img1, keypoints1, img2, keypoints2, matches, imgMatches);
Cv2.ImShow("Matches", imgMatches);
Cv2.WaitKey(1); // do events

Cv2.WaitKey(0);

Cv2.DestroyAllWindows();
img1.Dispose();
img2.Dispose();
```

در ابتدا نیاز به یک تشخیص دهنده یا Detector داریم. در OpenCVSharp، الگوریتم‌ها و کلاس‌های FastFeatureDetector, StarDetector, SIFT, SURF, ORB, BRISK, MSER, GFTTDetector, DenseFeatureDetector, SimpleBlobDetector برای اینکار قابل استفاده هستند. برای مثال در اینجا از الگوریتم SURF آن استفاده شده‌است.
کار این تشخیص دهنده، تشخیص نقاط کلیدی تصاویر است. برای مثال تشخیص گوشه‌ها، لبه‌ها و غیره.
سپس اطلاعات نواحی اطراف هر نقطه‌ی کلیدی را تحت عنوان descriptors استخراج می‌کنیم.
بعد از محاسبه‌ی نقاط کلیدی هر تصویر، اینبار نیاز است این نقاط را بین دو تصویر با هم مقایسه کرد و مشابه‌ها را یافت. برای مثال الگوریتم BFMatcher یک Brute force matcher است که بر اساس اطلاعات نواحی اطراف هر نقطه‌ی کلیدی، سعی در یافتن نقاط مشابه می‌کند.
پس از یافتن نقاط مشابه، نیاز است بر اساس آن‌ها نگاشتی بین دو تصویر صورت گیرد و مشابه‌ها ترسیم شوند. متد DrawMatches این‌کار را انجام می‌شود.

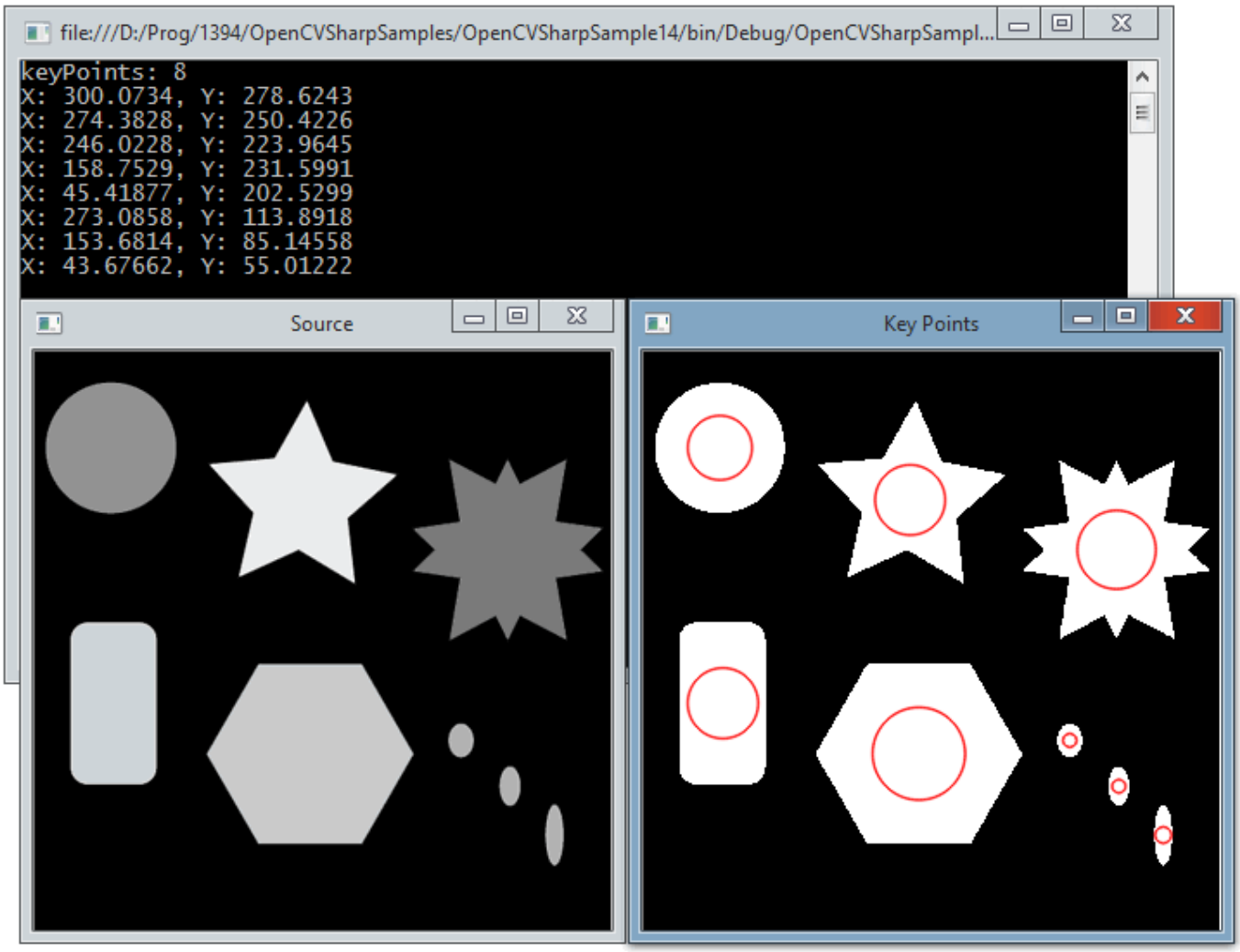
اگر علاقمند هستید که با ریز جزئیات ریاضی الگوریتم‌های استفاده شده نیز آشنا شوید، سری مطالب descriptors را دنبال نمائید.

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید.

عنوان: OpenCVSharp #14
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۲۸ ۱۲:۲۰
آدرس: www.dotnettips.info
گروه‌ها: OpenCV

تشخیص BLOBs در تصویر به کمک OpenCV

BLOB یا **B** inary **L** arge **OB** ject به معنای گروهی از نقاط به هم پیوسته‌ی در یک تصویر باینری هستند. Large در اینجا به این معنا است که اشیایی با اندازه‌هایی مشخص، مدنظر هستند و اشیاء کوچک، به عنوان نویز درنظر گرفته خواهند شد و پردازش نمی‌شوند.
برای نمونه در تصویر ذیل، تصویر سمت چپ، تصویر اصلی است و تصویر سمت راست، نمونه‌ی باینری سیاه و سفید آن. سپس عملیات تشخیص Blobs بر روی تصویر اصلی انجام شده و نتیجه‌ی نهایی که مشخص کننده‌ی اشیاء تشخیص داده شده‌است، با دوایری قرمز در تصویر سمت راست، مشخص هستند.



معرفی کلاس SimpleBlobDetector

در کدهای ذیل، نحوه‌ی کار با کلاس SimpleBlobDetector را مشاهده می‌کنید. این کلاس کار تشخیص Blobs را در تصویر اصلی انجام می‌دهد:

```
var srcImage = new Mat(@"..\..\Images\cvlbl.png");
Cv2.ImShow("Source", srcImage);
Cv2.WaitKey(1); // do events


var binaryImage = new Mat(srcImage.Size(), MatType.CV_8UC1);

Cv2.CvtColor(srcImage, binaryImage, ColorConversion.BgrToGray);

Cv2.Threshold(binaryImage, binaryImage, thresh: 100, maxval: 255, type: ThresholdType.Binary);


var detectorParams = new SimpleBlobDetector.Params
{
    //MinDistBetweenBlobs = 10, // 10 pixels between blobs
    //MinRepeatability = 1,

    //MinThreshold = 100,
    //MaxThreshold = 255,
    //ThresholdStep = 5,

    FilterByArea = false,
    //FilterByArea = true,
    //MinArea = 0.001f, // 10 pixels squared
    //MaxArea = 500,

    FilterByCircularity = false,
    //FilterByCircularity = true,
    //MinCircularity = 0.001f,

    FilterByConvexity = false,
    //FilterByConvexity = true,
    //MinConvexity = 0.001f,
    //MaxConvexity = 10,

    FilterByInertia = false,
    //FilterByInertia = true,
    //MinInertiaRatio = 0.001f,

    FilterByColor = false
    //FilterByColor = true,
    //BlobColor = 255 // to extract light blobs
};
var simpleBlobDetector = new SimpleBlobDetector(detectorParams);
var keyPoints = simpleBlobDetector.Detect(binaryImage);

Console.WriteLine("keyPoints: {0}", keyPoints.Length);
foreach (var keyPoint in keyPoints)
{
    Console.WriteLine("X: {0}, Y: {1}", keyPoint.Pt.X, keyPoint.Pt.Y);
}

var imageWithKeyPoints = new Mat();
Cv2.DrawKeypoints(
        image: binaryImage,
        keypoints: keyPoints,
        outImage: imageWithKeyPoints,
        color: Scalar.FromRgb(255, 0, 0),
        flags: DrawMatchesFlags.DrawRichKeypoints);


Cv2.ImShow("Key Points", imageWithKeyPoints);
Cv2.WaitKey(1); // do events


Cv2.WaitKey(0);

Cv2.DestroyAllWindows();
srcImage.Dispose();
imageWithKeyPoints.Dispose();
```

توضیحات

در اینجا ابتدا تصویر منبع به صورتی متداول باگذاری شده‌است. سپس توسط متد CvtColor به نمونه‌ای سیاه و سفید و به کمک متد Threshold به یک تصویر باینری تبدیل شده‌است.

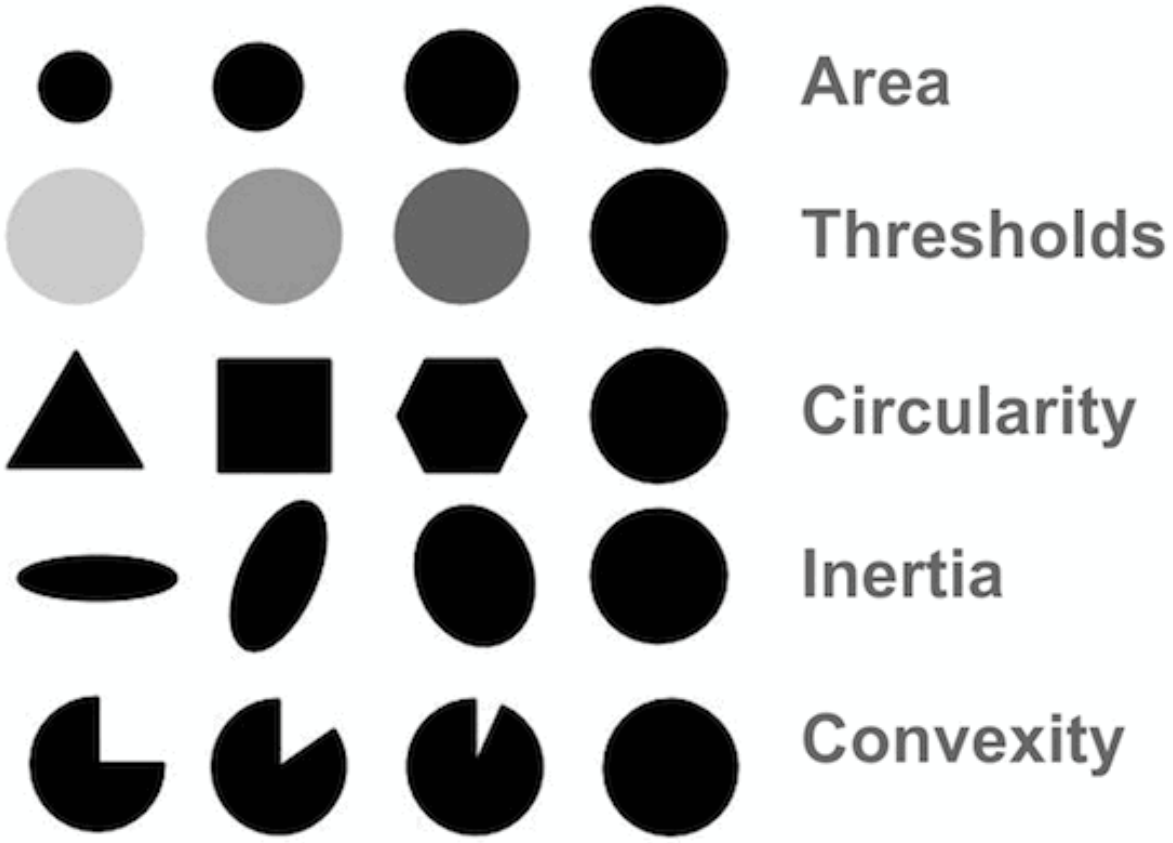
اگر به تصویر ابتدای بحث دقت کنید، اشیاء موجود در منبع مفروض، رنگ‌های متفاوتی دارند؛ اما در تصویر نهایی تولید شده، تمام

آن‌ها تبدیل به اشیایی سفید رنگ شده‌اند. این کار را متد Cv2.Threshold انجام داده‌است، تا کلاس SimpleBlobDetector قابلیت تشخیص بهتری را پیدا کند. تشخیص اشیایی یک دست، بسیار ساده‌تر هستند از تشخیص اشیایی با رنگ‌ها و شدت نور متفاوت.
اگر بخواهیم الگوریتم Threshold را بیان کنیم به pseudo code زیر خواهیم رسید:

```
if src(x,y) > thresh
    dst(x,y) = maxValue
else
    dst(x,y) = 0
```

در اینجا رنگ نقطه‌ی x,y با مقدار thresh (پارامتر سوم متد Cv2.Threshold) مقایسه می‌شود. اگر مقدار آن بیشتر بود، با پارامتر چهارم جایگزین خواهد شد. مقدار 255 در اینجا روشن‌ترین حالت ممکن است؛ اگر خیر با صفر یا تیره‌ترین رنگ، جایگزین می‌شود. به همین دلیل است که در تصویر سمت راست، تمام اشیاء را با روشن‌ترین رنگ ممکن مشاهده می‌کنید.

سپس پارامتر اصلی سازنده‌ی کلاس SimpleBlobDetector مقدار دهی شده‌است. این تنظیمات در تشخیص اشیاء موجود در تصویر بسیار مهم هستند. برای درک بهتر واژه‌هایی مانند Circularity، Convexity، Inertia و امثال آن، به تصویر ذیل دقت کنید:



در اینجا می‌توان حداقل و حداکثر تقعر و تحدب اشیاء و یا حتی حداقل و حداکثر اندازه‌ی اشیاء مدنظر را مشخص کرد.
اگر سازنده‌ی کلاس SimpleBlobDetector به نال تنظیم شود، از مقادیر پیش فرض آن استفاده خواهند شد که در اینجا به معنای تشخیص اشیاء کدر دایره‌ای شکل هستند. به همین جهت در تنظیمات فوق FilterByColor به false تنظیم شده‌است تا اشکال سفید رنگ تصویر اصلی را بتوان تشخیص داد.

در ادامه متد simpleBlobDetector.Detect بر روی تصویر باینری بهبود داده شده، فراخوانی گشته‌است. خروجی آن نقاط کلیدی اشیاء یافت شده‌است. این نقاط را می‌توان توسط متد DrawKeypoints ترسیم کرد که حاصل آن دوایر قرمز رنگ موجود در مرکز اشیاء یافت شده‌ی در تصویر سمت راست هستند.

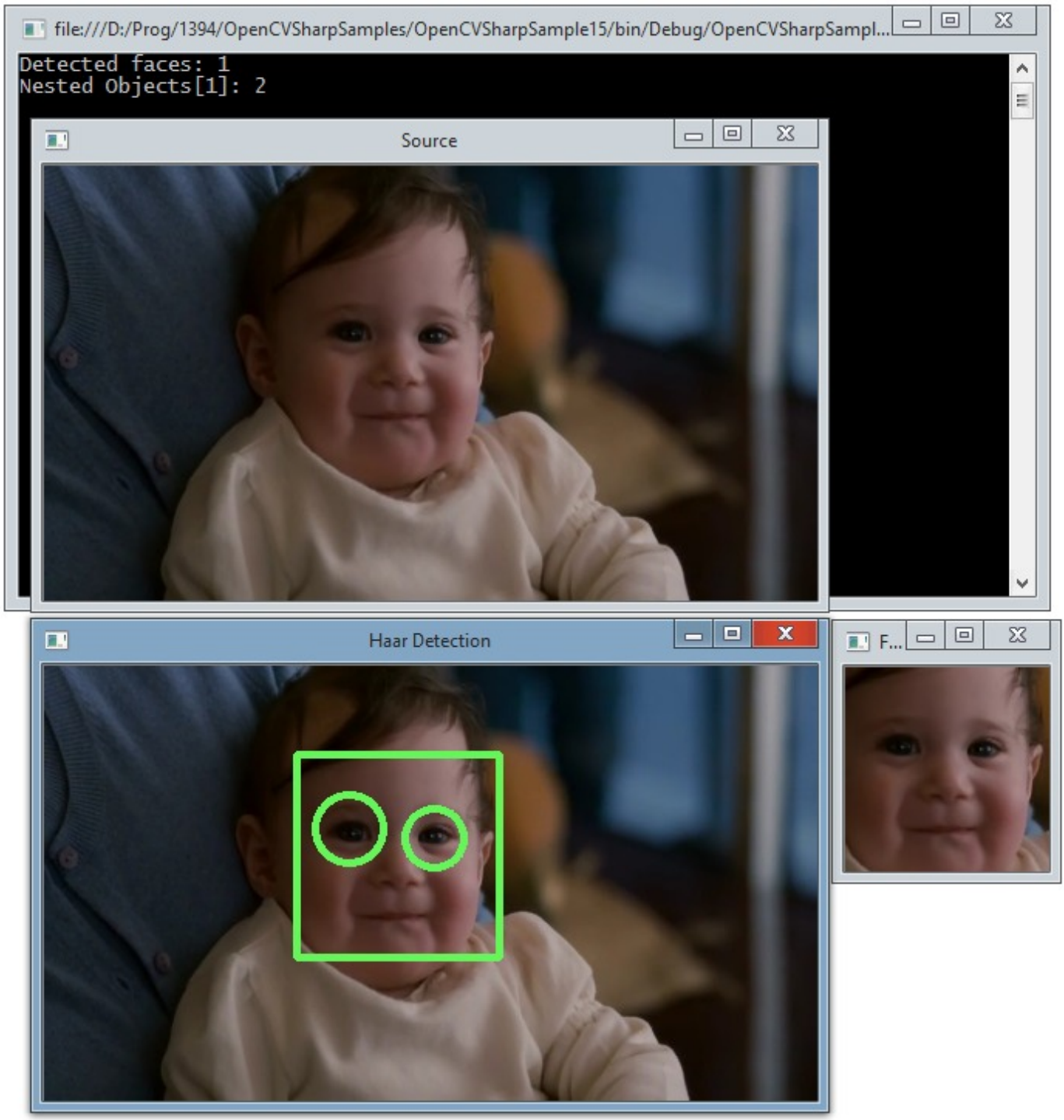
کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید.

عنوان: OpenCVSharp #15
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۲۹ ۱۵:۲۰
آدرس: www.dotnettips.info
گروه‌ها: OpenCV

تشخیص چهره به کمک OpenCV

OpenCV به کمک الگوریتم‌های machine learning (در اینجا Haar feature-based cascade classifiers) و داده‌های مرتبط با آن‌ها، قادر است اشیاء پیچیده‌ای را در تصاویر پیدا کند. برای پیگیری مثال بحث جاری نیاز است کتابخانه‌ی اصلی OpenCV را دریافت کنید؛ از این جهت که به فایل‌های XML موجود در پوشه‌ی opencv\sources\data\haarcascades آن نیاز داریم. در اینجا از دو فایل haarcascade_eye_tree_eyeglasses.xml و haarcascade_frontalface_alt.xml آن استفاده خواهیم کرد (این دوفایل جهت سهولت کار، به همراه مثال این بحث نیز ارائه شده‌اند).

فایل haarcascade_frontalface_alt.xml اصطلاحا trained data مخصوص یافتن چهره‌ی انسان در تصاویر است و فایل haarcascade_eye_tree_eyeglasses.xml کمک می‌کند تا بتوان در چهره‌ی یافت شده، چشمان شخص را نیز با دقت بالایی تشخیص داد؛ چیزی همانند تصویر ذیل:



مراحل تشخیص چهره توسط OpenCVSharp

ابتدا همانند سایر مثال‌های OpenCV، تصویر مدنظر را به کمک کلاس Mat بارگذاری می‌کنیم:

```
var srcImage = new Mat(@"..\..\Images\Test.jpg");
Cv2.ImShow("Source", srcImage);
Cv2.WaitKey(1); // do events

var grayImage = new Mat();
Cv2.CvtColor(srcImage, grayImage, ColorConversion.BgrToGray);
Cv2.EqualizeHist(grayImage, grayImage);
```

همچنین در اینجا جهت بالا رفتن سرعت کار و بهبود دقت تشخیص چهره، این تصویر اصلی به یک تصویر سیاه و سفید تبدیل شده‌است و سپس متد Cv2.EqualizeHist بر روی آن فراخوانی گشته‌است. این متد وضوح تصویر را جهت یافتن الگوها بهبود می‌بخشد.
سپس فایل xml یاد شده‌ی در ابتدای بحث را توسط کلاس CascadeClassifier بارگذاری می‌کنیم:

```
var cascade = new CascadeClassifier(@"..\..\Data\haarcascade_frontalface_alt.xml");
var nestedCascade = new CascadeClassifier(@"..\..\Data\haarcascade_eye_tree_eyeglasses.xml");

var faces = cascade.DetectMultiScale(
    image: grayImage,
    scaleFactor: 1.1,
    minNeighbors: 2,
    flags: HaarDetectionType.Zero | HaarDetectionType.ScaleImage,
    minSize: new Size(30, 30)
    );

Console.WriteLine("Detected faces: {0}", faces.Length);
```

پس از بارگذاری فایل‌های XML اطلاعات نحوه‌ی تشخیص چهره و اعضای آن، با فراخوانی متد DetectMultiScale، کار تشخیص چهره و استخراج آن از grayImage انجام خواهد شد. در اینجا minSize، اندازه‌ی حداقل چهره‌ی مدنظر است که قرار هست تشخیص داده شود. نواحی کوچکتر از این اندازه، به عنوان نویز در نظر گرفته خواهند شد و پردازش نمی‌شوند.
خروجی این متد، مستطیل‌ها و نواحی یافت شده‌ی مرتبط با چهره‌های موجود در تصویر هستند. اکنون می‌توان حلقه‌ای را تشکیل داد و این نواحی را برای مثال با مستطیل‌های رنگی، متمایز کرد:

```
var rnd = new Random();
var count = 1;
foreach (var faceRect in faces)
{
    var detectedFaceImage = new Mat(srcImage, faceRect);
    Cv2.ImShow(string.Format("Face {0}", count), detectedFaceImage);
    Cv2.WaitKey(1); // do events

    var color = Scalar.FromRgb(rnd.Next(0, 255), rnd.Next(0, 255), rnd.Next(0, 255));
    Cv2.Rectangle(srcImage, faceRect, color, 3);

    count++;
}

Cv2.ImShow("Haar Detection", srcImage);
Cv2.WaitKey(1); // do events
```

در اینجا علاوه بر رسم یک مستطیل، به دور ناحیه‌ی تشخیص داده شده، نحوه‌ی استخراج تصویر آن ناحیه را هم در سطر var detectedFaceImage مشاهده می‌کنید.

همچنین اگر علاقمند باشیم تا در این ناحیه‌ی یافت شده، چشمان شخص را نیز استخراج کنیم، می‌توان به نحو ذیل عمل کرد:

```
var rnd = new Random();
var count = 1;
foreach (var faceRect in faces)
{
    var detectedFaceImage = new Mat(srcImage, faceRect);
    Cv2.ImShow(string.Format("Face {0}", count), detectedFaceImage);
    Cv2.WaitKey(1); // do events

    var color = Scalar.FromRgb(rnd.Next(0, 255), rnd.Next(0, 255), rnd.Next(0, 255));
    Cv2.Rectangle(srcImage, faceRect, color, 3);


    var detectedFaceGrayImage = new Mat();
    Cv2.CvtColor(detectedFaceImage, detectedFaceGrayImage, ColorConversion.BgrToGray);
    var nestedObjects = nestedCascade.DetectMultiScale(
        image: detectedFaceGrayImage,
        scaleFactor: 1.1,
        minNeighbors: 2,
        flags: HaarDetectionType.Zero | HaarDetectionType.ScaleImage,
        minSize: new Size(30, 30)
        );
```

```
    Console.WriteLine("Nested Objects[{0}]: {1}", count, nestedObjects.Length);

    foreach (var nestedObject in nestedObjects)
    {
        var center = new Point
        {
            X = Cv.Round(nestedObject.X + nestedObject.Width * 0.5) + faceRect.Left,
            Y = Cv.Round(nestedObject.Y + nestedObject.Height * 0.5) + faceRect.Top
        };
        var radius = Cv.Round((nestedObject.Width + nestedObject.Height) * 0.25);
        Cv2.Circle(srcImage, center, radius, color, thickness: 3);
    }

    count++;
}

Cv2.ImShow("Haar Detection", srcImage);
Cv2.WaitKey(1); // do events
```

کدهای ابتدایی آن همانند توضیحات قبل است. تنها تفاوت آن، استفاده از nestedCascade بارگذاری شده‌ی در ابتدای بحث می‌باشد. این nestedCascade حاوی trained data استخراج چشمان اشخاص، از تصاویر است. پارامتر ورودی آن‌را نیز تصویر سیاه و سفید ناحیه‌ی چهره‌ی یافت شده، قرار داده‌ایم تا سرعت تشخیص چشمان شخص، افزایش یابد.

کدهای کامل این مثال را از اینجا **می‌توانید دریافت کنید.**

عنوان: OpenCVSharp #16
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۳۱ ۱۴:۱۰
آدرس: www.dotnettips.info
گروه‌ها: OpenCV

در قسمت قبل با نحوه‌ی استفاده از یک trained data از پیش آماده شده‌ی تشخیص چهره، آشنا شدیم. در این قسمت قصد داریم با نحوه‌ی تولید این فایل‌های XML آشنا شویم و یک تشخیص دهنده‌ی سفارشی را طراحی کنیم.

طراحی classifier سفارشی تشخیص خودروها

برای طراحی یک تشخیص دهنده‌ی سفارشی مبتنی بر الگوریتم‌های Machine learning، نیاز به تعداد زیادی تصویر داریم. در اینجا از بانک تصاویر خودروهای « UIUC Image Database for Car Detection » استفاده خواهیم کرد. در این بسته، یک سری تصویر positive و negative را می‌توان ملاحظه کرد. تصاویر مثبت، تصاویر انواع و اقسام خودروها هستند (550 عدد) و تصاویر منفی، تصاویر غیر خودرویی (500 عدد)؛ یا به عبارتی، هر تصویری، منهای تصاویر خودرو می‌تواند تصویر منفی باشد.

ایجاد فایل برداری از تصاویر خودروها

در ادامه یک فایل متنی را به نام carImages.txt ایجاد می‌کنیم. هر سطر این فایل چنین فرمتی را خواهد داشت:

```
pos/pos-177.pgm 1 0 0 100 40
```

ابتدا مسیر تصویر مشخص می‌شود. سپس عدد 1 به این معنا است که در این تصویر فقط یک عدد خودرو وجود دارد. 4 عدد بعدی، ابعاد مستطیلی تصویر هستند.
در ادامه فایل متنی دیگری را به نام negativeImages.txt جهت درج اطلاعات تصاویر منفی، ایجاد می‌کنیم. اینبار هر سطر این فایل تنها حاوی مسیر تصویر مدنظر است:

```
neg/neg-274.pgm
```

این دو فایل را می‌توان با استفاده از دو متد ذیل، به سرعت تولید کرد:

```
private static void createCarImagesFile()
{
    var sb = new StringBuilder();
    foreach (var file in new DirectoryInfo(@"..\..\CarData\CarData\TrainImages").GetFiles("*pos-
*.pgm"))
    {
        sb.AppendFormat("{0} {1} {2} {3} {4} {5}{6}", file.FullName, 1, 0, 0, 100, 40,
Environment.NewLine);
    }
    File.WriteAllText(@"..\..\CarsInfo\carImages.txt", sb.ToString());
}

private static void createNegativeImagesFile()
{
    var sb = new StringBuilder();
    foreach (var file in new DirectoryInfo(@"..\..\CarData\CarData\TrainImages").GetFiles("*neg-
*.pgm"))
    {
        sb.AppendFormat("{0}{1}", file.FullName,Environment.NewLine);
    }
    File.WriteAllText(@"..\..\CarsInfo\negativeImages.txt", sb.ToString());
}
```

برای کامپایل اطلاعات فایل‌های تولید شده، نیاز به فایل opencv_createsamples.exe است. این فایل را در پوشه‌ی opencv\build\x86\vc12\bin بسته‌ی اصلی OpenCV می‌توانید پیدا کنید.

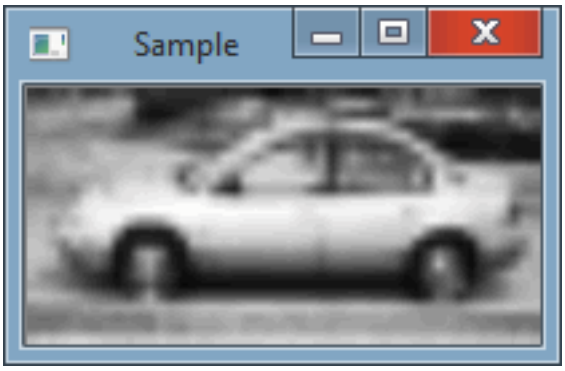
```
opencv_createsamples.exe -info carImages.txt -num 550 -w 48 -h 24 -vec cars.vec
```

پارامترهای این دستور شامل سوئیچ info است؛ به معنای مشخص سازی فایل اطلاعات تصاویر مثبت. سوئیچ num تعداد تصاویر آن‌را تعیین می‌کند و سوئیچ‌های w و h، طول و عرض تصاویر هستند. سوئیچ vec نیز جهت تولید یک فایل vector از این اطلاعات بکار می‌رود.
پس از اجرای این دستور، فایل cars.vec تولید خواهد شد؛ با این خروجی:

```
Info file name: carImages.txt
Img file name: (NULL)
Vec file name: cars.vec
BG  file name: (NULL)
Num: 550
BG color: 0
BG threshold: 80
Invert: FALSE
Max intensity deviation: 40
Max x angle: 1.1
Max y angle: 1.1
Max z angle: 0.5
Show samples: FALSE
Original image will be scaled to:
  Width: $backgroundWidth / 48
  Height: $backgroundHeight / 24
Create training samples from images collection...
Done. Created 550 samples
```

اگر علاقمند هستید که محتویات فایل باینری cars.vec را مشاهده کنید، دستور ذیل را صادر نمائید:

```
"c:\opencv\build\x86\vc12\bin\opencv_createsamples.exe" -vec cars.vec -w 48 -h 24
```



در این پنجره‌ی باز شده، تصاویر بعدی و قبلی را می‌توان با دکمه‌های arrow صفحه کلید، مشاهده کرد.

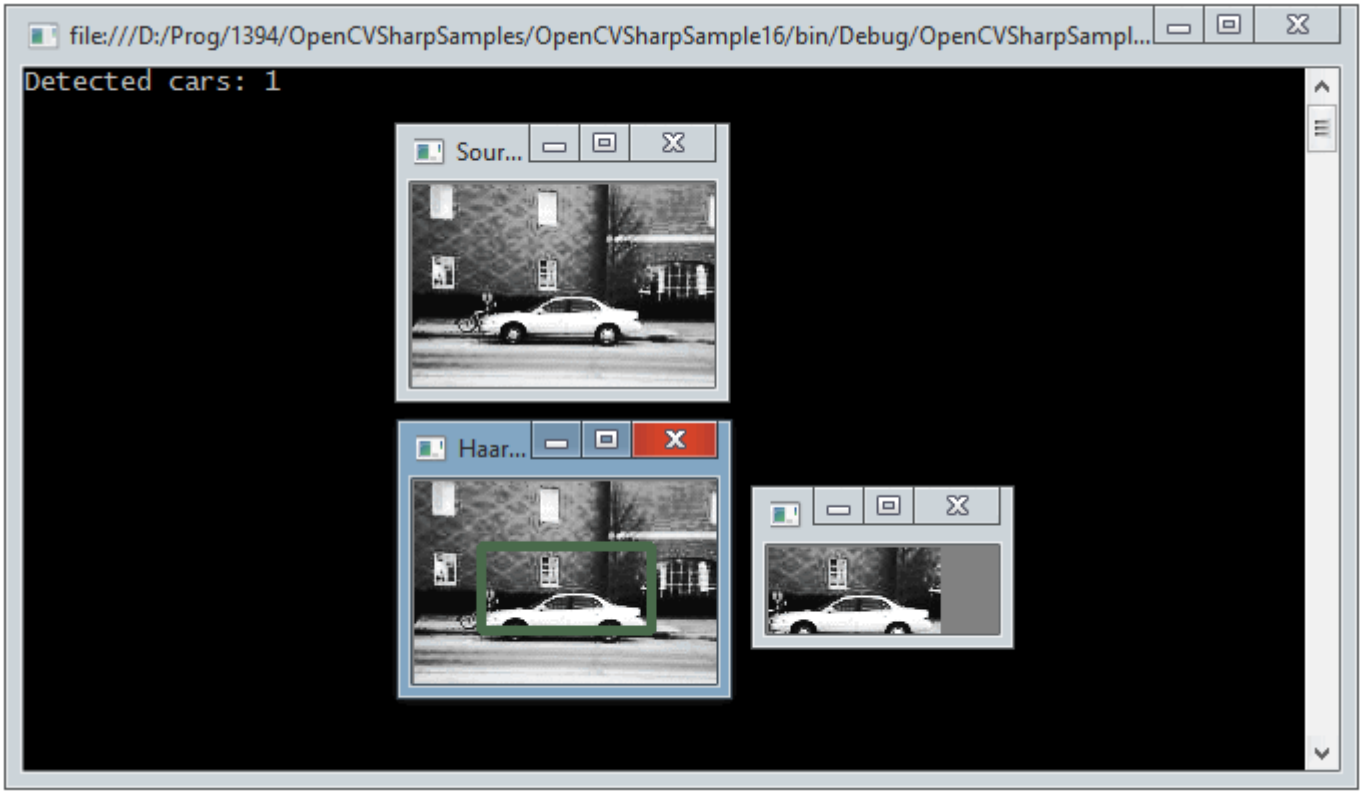
تبدیل فایل برداری تصاویر خودروها به trained data

تا اینجا موفق شدیم بیش از 500 تصویر خودرو را تبدیل به یک فایل برداری سازگار با OpenCV کنیم. اکنون نیاز است، این اطلاعات پردازش شده و trained data مخصوص الگوریتم‌های machine learning تولید شود. این‌کار را توسط برنامه‌ی opencv_traincascade.exe انجام خواهیم داد. این فایل نیز در پوشه‌ی opencv\build\x86\vc12\bin بسته‌ی اصلی OpenCV موجود است.
دستور ذیل در پوشه‌ی data، بر اساس اطلاعات برداری cars.vec و همچنین تصاویر منفی مشخص شده‌ی در فایل negativeImages.txt، با تعداد هر کدام 500 عدد (این عدد را توصیه شده‌است که اندکی کمتر از تعداد max موجود مشخص کنیم) و تعداد مراحل 2 (هر چقدر این تعداد مراحل بیشتر باشد، فایل نهایی تولید شده دقت بالاتری خواهد داشت؛ اما تولید آن به زمان

بیشتری نیاز دارد) اجرا می‌شود. در اینجا featureType به LBP یا Local binary Pattern، تنظیم شده‌است. این الگوریتم از Haar cascade سریعتر است.

```
"E:\opencv\bin\opencv_traincascade.exe" -data data -vec cars.vec -bg negativeImages.txt -numPos 500 -numNeg 500 -numStages 2 -w 48 -h 24 -featureType LBP
```

خروجی اجرای این دستور را می‌توانید در پوشه‌ی data با نام cascade.xml، پیدا کنید. پس از آن، روش استفاده‌ی از این فایل، با مطلب تشخیص چهره تفاوتی ندارد.



کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید.

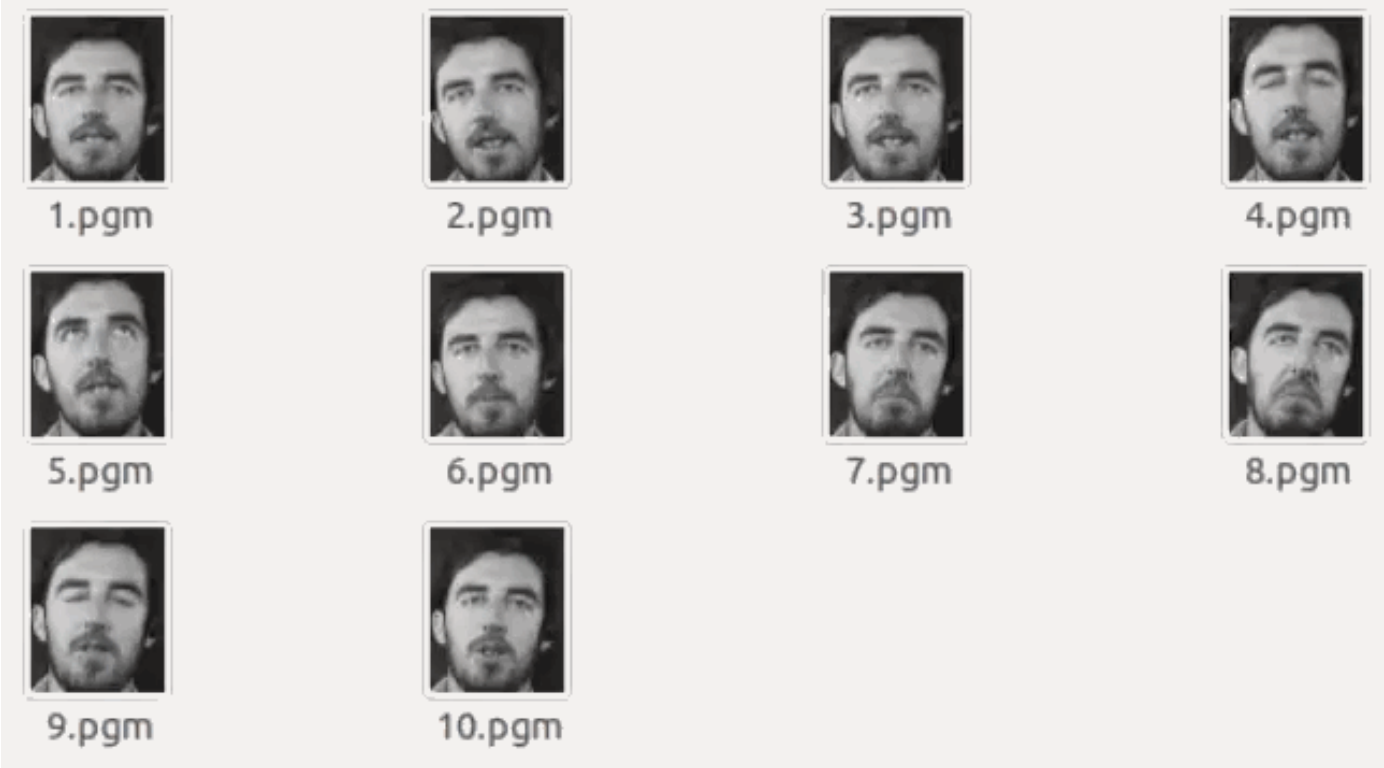
عنوان: OpenCVSharp #17
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۴/۰۱ ۱۵:۴۵
آدرس: www.dotnettips.info
گروه‌ها: OpenCV

تشخیص اشخاص به کمک OpenCV

فرض کنید قصد دارید یک سیستم حضور غیاب مبتنی بر تشخیص چهره را طراحی کنید. قسمت استخراج چهره، از تصویر کلی رسیده را بررسی کردیم . اما در ادامه چگونه تشخیص دهیم که این چهره متعلق به چه شخصی است؟ با توجه به اینکه تصویر چهره‌ی یک شخص می‌تواند از زوایای مختلفی تهیه شود و یا حتی حالات روحی منعکس شده‌ی در صورت نیز در تغییر بیت و بایت‌های تصویر چهره مؤثر هستند.

بانک اطلاعاتی تصاویر چهره‌های اشخاص

در اینجا از تصاویر « The Database of Faces » استفاده خواهیم کرد. این مجموعه شامل تصاویر 40 شخص، در 10 حالت مختلف است.



برای بارگذاری این تصاویر و استفاده‌ی از آن‌ها در الگوریتم FisherFaceRecognizer نیاز به ساختار ذیل است:

```
public class ImageInfo
{
    public Mat Image { set; get; }
    public int ImageGroupId { set; get; }
    public int ImageId { set; get; }
}
```

در اینجا Image، محتوای تصویر انتخابی است. مقدار ImageGroupId مساوی مقدار عددی نام پوشه‌ی تصاویر منهای یک، تنظیم

می‌شود. برای مثال پوشه‌ی s1 به گروه صفر تنظیم می‌شود. ImageId نیز به یک مقدار خود افزایش یابنده معادل شماره‌ی جاری تصویر، تنظیم می‌گردد؛ به این صورت:

```
var images = new List<ImageInfo>();

var imageId = 0;
foreach (var dir in new DirectoryInfo(@"..\..\Images").GetDirectories())
{
    var groupId = int.Parse(dir.Name.Replace("s", string.Empty)) - 1;
    foreach (var imageFile in dir.GetFiles("*.pgm"))
    {
        images.Add(new ImageInfo
        {
            Image = new Mat(imageFile.FullName, LoadMode.GrayScale),
            ImageId = imageId++,
            ImageGroupId = groupId
        });
    }
}
```

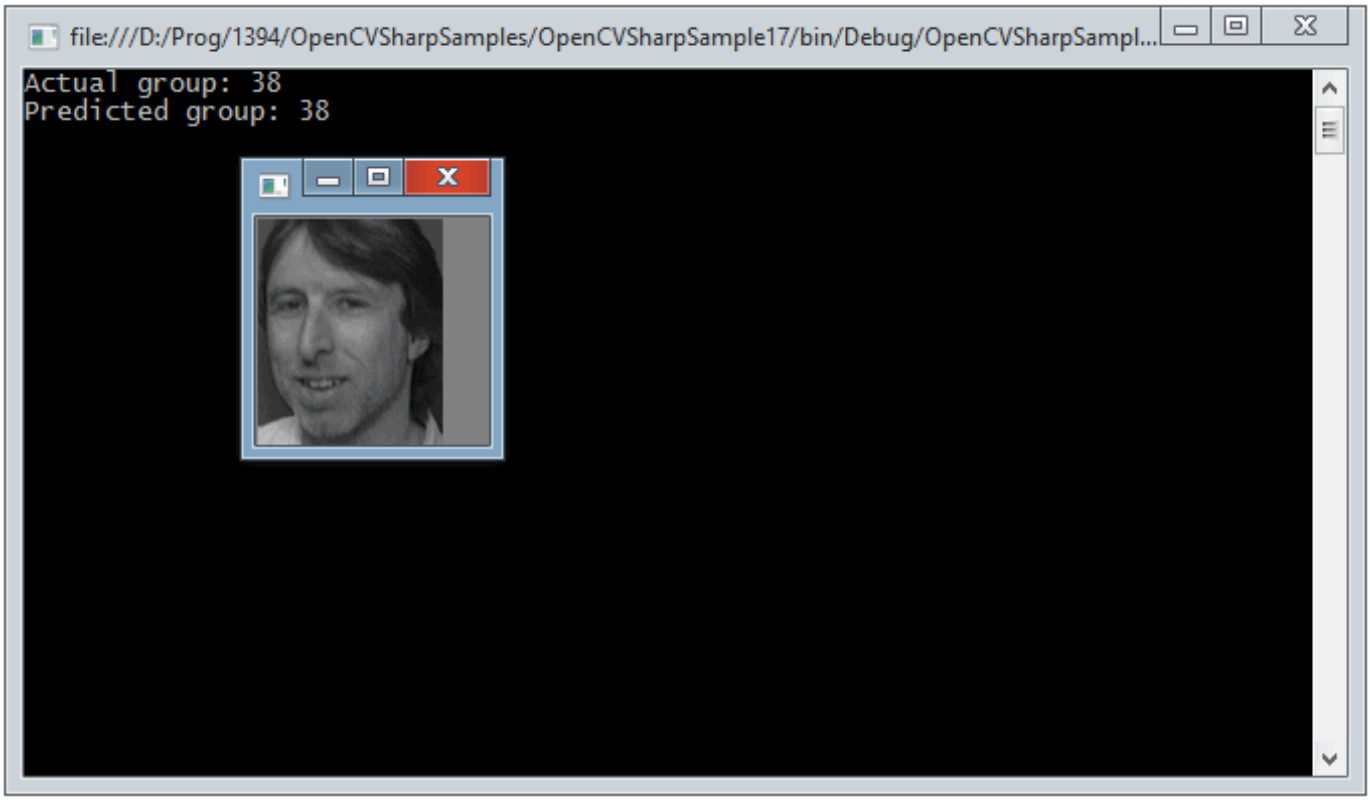
ابتدا پوشه‌های دیتابیس تصاویر یافت شده و سپس از نام هر پوشه یک شماره‌ی گروه (یا شماره‌ی شخص) استخراج می‌شود. سپس تصاویر این پوشه به لیست تصاویر اصلی اضافه خواهند شد.

تشخیص یک چهره‌ی اتفاقی

پس از تشکیل لیست تصاویر، اکنون کار با الگوریتم FisherFaceRecognizer به نحو ذیل خواهد بود:

```
var model = FaceRecognizer.CreateFisherFaceRecognizer();
model.Train(images.Select(x => x.Image), images.Select(x => x.ImageGroupId));

var rnd = new Random();
var randomImageId = rnd.Next(0, images.Count - 1);
var testSample = images[randomImageId];

Console.WriteLine("Actual group: {0}", testSample.ImageGroupId);
Cv2.ImShow("actual", testSample.Image);

var predictedGroupId = model.Predict(testSample.Image);
Console.WriteLine("Predicted group: {0}", predictedGroupId);
```

پارامتر اول متد Train، لیست تصاویر است و پارامتر دوم، لیست شماره گروه‌های متناظر با هر تصویر است که در اینجا به عنوان برچسب نیز نامگذاری شده‌است.
سپس با استفاده از کلاس Random، یک تصویر اتفاقی انتخاب می‌شود.
اکنون این تصویر اتفاقی به متد Predict ارسال شده و نتیجه‌ی آن، شماره گروه چهره‌ی تشخیص داده شده‌است. به این ترتیب می‌توان تشخیص داد که یک تصویر مفروض ورودی، متعلق به چه شخصی (یا در اینجا گروه یا برچسب) است.



کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید.

نظرات خوانندگان

نویسنده: م منفرد
تاریخ: ۱۳۹۴/۰۴/۰۱ ۱۸:۴۴

با این اوصاف، اگر عکسی از حالت مثلا خنده فرد در بانک ثبت شده است و ما حالت عصبانیت ایشان را داشته باشیم، قابل تشخیص است؟

نویسنده: صادق ایمانی
تاریخ: ۱۳۹۴/۰۴/۰۱ ۲۰:۱۴

خیلی ممنون؛ من خودم چند ماه قبل داشتم یه سیستم تشخیص و شناسایی چهره با کمک این سایت مینوشتم... که خوب متاسفانه نتیجه خوبی نگرفتم به این خاطر که بعضی چهره‌ها رو به اشتباه تشخیص می‌داد... البته تعداد گروههای عکس هایی که من Train کردم زیاد بود شاید به این خاطر اشتباه تشخیص می‌داد... فک کنم در حدود 50 گروه بود که در هر گروه 10 عکس برای Train وجود داشت.
به نظر شما بهترین Library برای تشخیص و شناسایی چهره چیه؟ و اینکه این درست است که اگر تعداد گروههای Train زیاد باشه احتمال تشخیص صحیح کم میشه یا اینکه من یه جای کار اشتباه کردم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۴/۰۱ ۲۰:۲۲

یک عکس کافی نیست. این‌ها الگوریتم‌های machine learning هستند و متد Train ایی که اینجا بکار گرفته شده، دقیقا به همین منظور است. هر چقدر تعداد نمونه‌های بیشتری از یک شخص داشته باشید (مثل تصویر ابتدای بحث یا مطلب تهیه trained data)، دقت کار بیشتر می‌شود.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۴/۰۱ ۲۰:۲۵

در مطلب تهیه trained data اشاره‌ای به این موضوع شده‌است. اگر تعداد stage معرفی شده بیشتر شود، دقت بیشتر خواهد شد و به همین ترتیب، تهیه‌ی داده‌ی آموزش داده شده کندتر می‌شود. همچنین در اینجا الگوریتم‌های زیادی هم برای آزموش دادن داده‌ها وجود دارند؛ نمونه‌ای که بکار گرفته شد LBP یا Local binary Pattern بود که بسیار سریع هست. روش‌های دیگر دقت بالاتری دارند اما کند هستند. درکل باید روی الگوریتم‌ها و تعداد stages بیشتر بررسی کنید. به علاوه الگوریتم‌های FaceRecognizer دیگری هم وجود دارند که نیاز به بررسی بیشتری دارند. در مطلب جاری فقط الگوریتم فیشر این سری بررسی شد.

عنوان: OpenCVSharp #18
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۴/۰۴ ۱۰:۵۰
آدرس: www.dotnettips.info
گروه‌ها: OCR, OpenCV

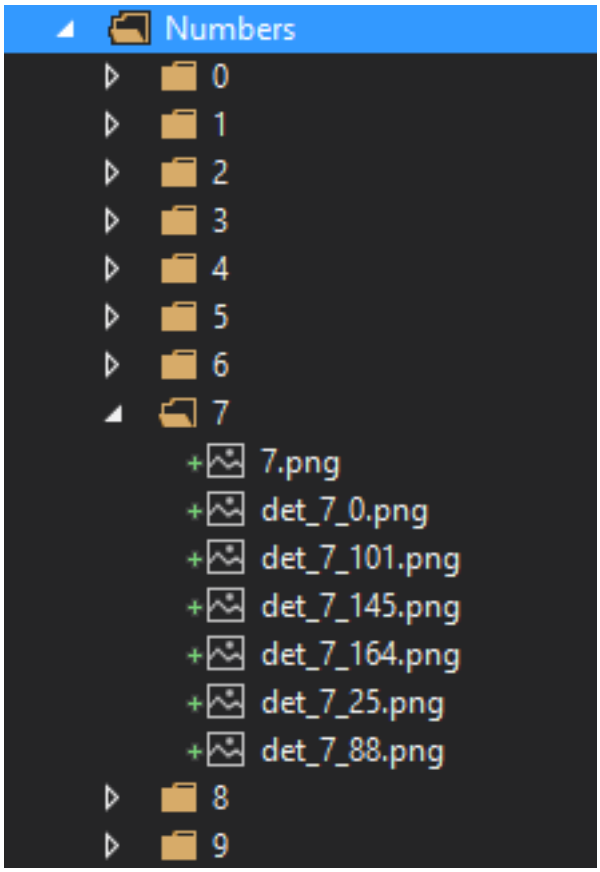
ساخت یک OCR ساده تشخیص اعداد انگلیسی به کمک OpenCV

این مطلب را می‌توان به عنوان جمع بندی مطالبی که تاکنون بررسی شدند درنظر گرفت و در اساس مطلب جدیدی ندارد و صرفا ترکیب یک سری تکنیک است؛ برای مثال:

چطور یک تصویر را به نمونه‌ی سیاه و سفید آن تبدیل کنیم؟
کار با متد Threshold جهت بهبود کیفیت یک تصویر جهت تشخیص اشیاء
تشخیص کانتورها (Contours) و اشیاء موجود در یک تصویر
آشنایی با نحوه‌ی گروه بندی تصاویر مشابه و مفاهیمی مانند برچسب‌های تصاویر که بیانگر یک گروه از تصاویر هستند.

تهیه تصاویر اعداد انگلیسی جهت آموزش دادن به الگوریتم CvKNearest

در اینجا نیز از یکی دیگر از الگوریتم‌های machine learning موجود در OpenCV به نام CvKNearest برای تشخیص اعداد انگلیسی استفاده خواهیم کرد. این الگوریتم نزدیک‌ترین همسایه‌ی اطلاعاتی مفروض را در گروهی از داده‌های آموزش داده شده‌ی به آن پیدا می‌کند. خروجی آن شماره‌ی این گروه است. بنابراین نحوه‌ی طبقه‌ی بندی اطلاعات در اینجا چیزی شبیه به شکل زیر خواهد بود:



مجموعه‌ای از تصاویر 0 تا 9 را جمع آوری کرده‌ایم. هر کدام از پوشه‌ها، بیانگر اعدادی از یک خانواده هستند. این تصویر را با

فرمت ذیل جمع آوری می‌کنیم:

```
public class ImageInfo
{
    public Mat Image { set; get; }
    public int ImageGroupId { set; get; }
    public int ImageId { set; get; }
}
```

به این ترتیب

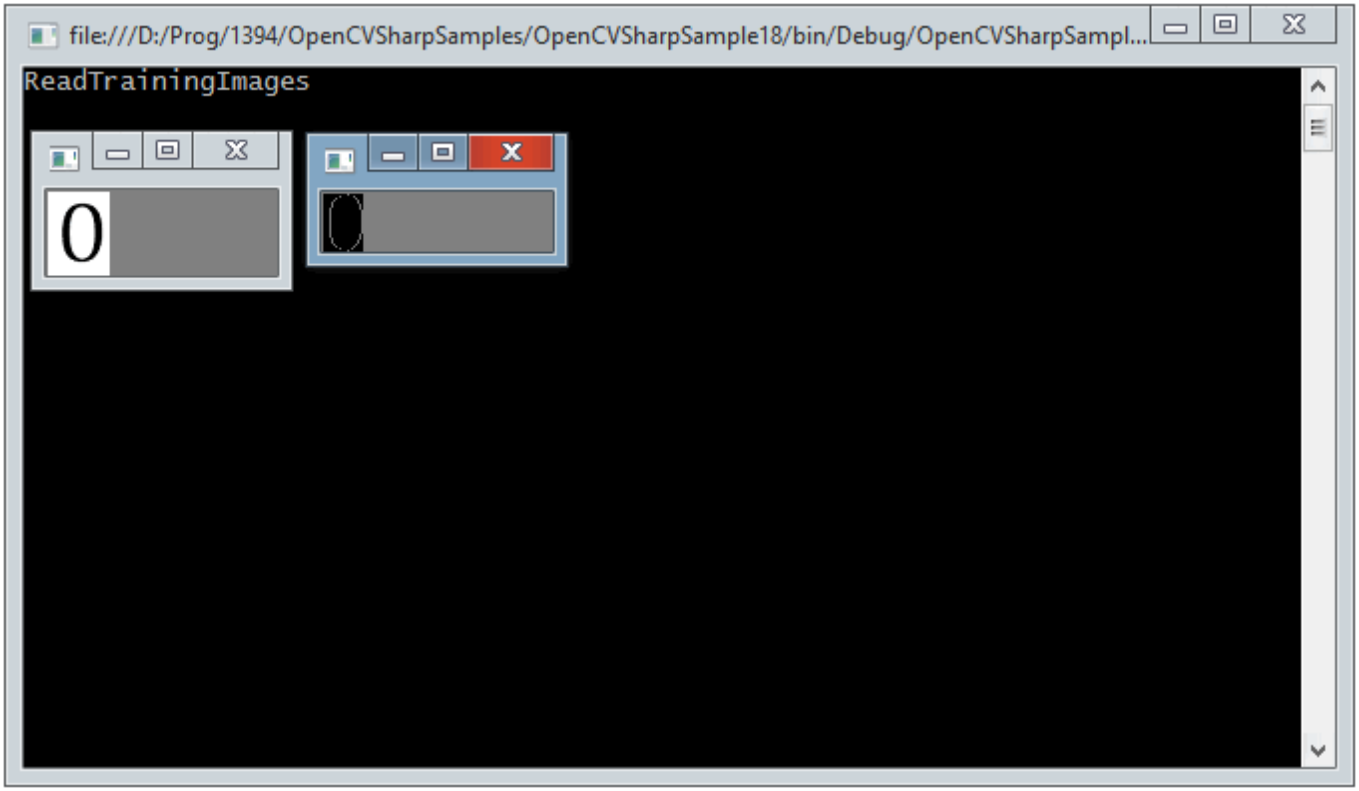
```
public IList<ImageInfo> ReadTrainingImages(string path, string ext)
{
    var images = new List<ImageInfo>();

    var imageId = 1;
    foreach (var dir in new DirectoryInfo(path).GetDirectories())
    {
        var groupId = int.Parse(dir.Name);
        foreach (var imageFile in dir.GetFiles(ext))
        {
            var image = processTrainingImage(new Mat(imageFile.FullName, LoadMode.GrayScale));
            if (image == null)
            {
                continue;
            }

            images.Add(new ImageInfo
            {
                Image = image,
                ImageId = imageId++,
                ImageGroupId = groupId
            });
        }
    }

    return images;
}
```

در متد خواندن تصاویر آموزشی، ابتدا پوشه‌های اصلی مسیر Numbers تصویر ابتدای بحث دریافت می‌شوند. سپس نام هر پوشه، شماره‌ی گروه تصاویر موجود در آن پوشه را تشکیل خواهد داد. به این نام در الگوریتم‌های machine leaning، کلاس هم گفته می‌شود. سپس هر تصویر را با فرمت سیاه و سفید بارگذاری کرده و به لیست تصاویر موجود اضافه می‌کنیم. در اینجا از متد processTrainingImage نیز استفاده شده‌است. هدف از آن بهبود کیفیت تصویر دریافتی جهت کار تشخیص اشیاء است:

```
private static Mat processTrainingImage(Mat gray)
{
    var threshImage = new Mat();
    Cv2.Threshold(gray, threshImage, Thresh, ThresholdMaxVal, ThresholdType.BinaryInv); // Threshold to
find contour

    Point[][] contours;
    HiearchyIndex[] hierarchyIndexes;
    Cv2.FindContours(
        threshImage,
        out contours,
        out hierarchyIndexes,
        mode: ContourRetrieval.CComp,
        method: ContourChain.ApproxSimple);

    if (contours.Length == 0)
    {
        return null;
    }

    Mat result = null;

    var contourIndex = 0;
    while ((contourIndex >= 0))
    {
        var contour = contours[contourIndex];

        var boundingRect = Cv2.BoundingRect(contour); //Find bounding rect for each contour
        var roi = new Mat(threshImage, boundingRect); //Crop the image
```

```
        //Cv2.ImShow("src", gray);
        //Cv2.ImShow("roi", roi);
        //Cv.WaitKey(0);

        var resizedImage = new Mat();
        var resizedImageFloat = new Mat();
        Cv2.Resize(roi, resizedImage, new Size(10, 10)); //resize to 10X10
        resizedImage.ConvertTo(resizedImageFloat, MatType.CV_32FC1); //convert to float
        result = resizedImageFloat.Reshape(1, 1);

        contourIndex = hierarchyIndexes[contourIndex].Next;
    }

    return result;
}
```

عملیات صورت گرفته‌ی در این متد را با تصویر ذیل بهتر می‌توان توضیح داد:



ابتدا تصویر اصلی بارگذاری می‌شود؛ همان تصویر سمت چپ. سپس با استفاده از متد Threshold، شدت نور نواحی مختلف آن یکسان شده و آماده می‌شود برای تشخیص کانتورهای موجود در آن. در ادامه با استفاده از متد FindContours، شیء مرتبط با عدد جاری یافت می‌شود. سپس متد Cv2.BoundingRect مستطیل دربرگیرنده‌ی این شیء را تشخیص می‌دهد (تصویر سمت راست). بر این اساس می‌توان تصویر اصلی ورودی را به یک تصویر کوچکتر که صرفا شامل ناحیه‌ی عدد مدنظر است، تبدیل کرد.
در ادامه برای کار با الگوریتم CvKNearest نیاز است تا این تصویر بهبود یافته را تبدیل به یک ماتریس یک بعدی کردی که روش انجام کار توسط متد Reshape مشاهده می‌کنید.
از همین روش پردازش و بهبود تصویر ورودی، جهت پردازش اعداد یافت شده‌ی در یک تصویر با تعداد زیادی عدد نیز استفاده خواهیم کرد.

آموزش دادن به الگوریتم CvKNearest

تا اینجا تصاویر گروه بندی شده‌ای را خوانده و لیستی از آن‌ها را مطابق فرمت الگوریتم CvKNearest تهیه کردیم. مرحله‌ی بعد، معرفی این لیست به متد Train این الگوریتم است:

```
public CvKNearest TrainData(IList<ImageInfo> trainingImages)
{
    var samples = new Mat();
    foreach (var trainingImage in trainingImages)
    {
        samples.PushBack(trainingImage.Image);
    }

    var labels = trainingImages.Select(x => x.ImageGroupId).ToArray();
    var responses = new Mat(labels.Length, 1, MatType.CV_32SC1, labels);
    var tmp = responses.Reshape(1, 1); //make continuous
    var responseFloat = new Mat();
    tmp.ConvertTo(responseFloat, MatType.CV_32FC1); // Convert  to float


    var kNearest = new CvKNearest();
    kNearest.Train(samples, responseFloat); // Train with sample and responses
    return kNearest;
}
```

متد Train دو ورودی دارد. ورودی اول آن یک تصویر است که باید از طریق متد PushBack کلاس Mat تهیه شود. بنابراین لیست تصاویر اصلی را تبدیل به لیستی از Matها خواهیم کرد.
سپس نیاز است لیست گروه‌های متناظر با تصاویر اعداد را تبدیل به فرمت مورد انتظار متد Train کنیم. در اینجا صرفا لیستی از اعداد صحیح را داریم. این لیست نیز باید تبدیل به یک Mat شود که روش انجام آن در متد فوق بیان شده‌است. کلاس Mat سازنده‌ی مخصوصی را جهت تبدیل لیست اعداد، به همراه دارد. این Mat نیز باید تبدیل به یک ماتریس یک بعدی شود که برای این منظور از متد Reshape استفاده شده‌است.

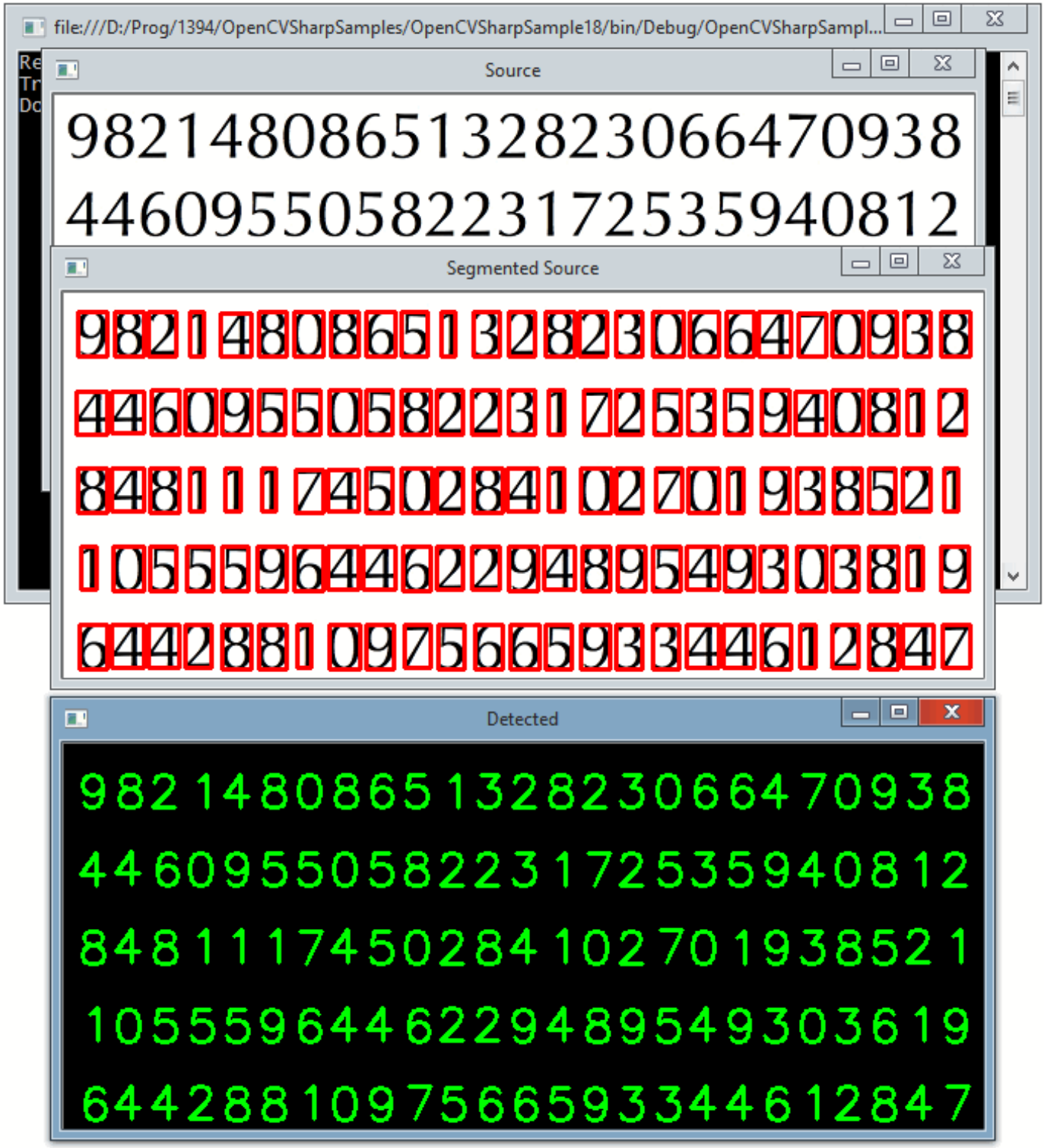
انجام عملیات OCR نهایی

پس از تهیه‌ی لیستی از تصاویر و آموزش دادن آن‌ها به الگوریتم CvKNearest، تنها کاری که باید انجام دهیم، یافتن اعداد در تصویر نمونه‌ی مدنظر و سپس معرفی آن به متد FindNearest الگوریتم CvKNearest است. روش انجام اینکار بسیار شبیه است به روش معرفی شده در متد processTrainingImage که پیشتر بررسی شد:

```
public void DoOCR(CvKNearest kNearest, string path)
{
    var src = Cv2.ImRead(path);
    Cv2.ImShow("Source", src);

    var gray = new Mat();
    Cv2.CvtColor(src, gray, ColorConversion.BgrToGray);

    var threshImage = new Mat();
    Cv2.Threshold(gray, threshImage, Thresh, ThresholdMaxVal, ThresholdType.BinaryInv); // Threshold to
find contour


    Point[][] contours;
    HiearchyIndex[] hierarchyIndexes;
    Cv2.FindContours(
        threshImage,
        out contours,
        out hierarchyIndexes,
        mode: ContourRetrieval.CComp,
        method: ContourChain.ApproxSimple);

    if (contours.Length == 0)
    {
        throw new NotSupportedException("Couldn't find any object in the image.");
    }

    //Create input sample by contour finding and cropping
    var dst = new Mat(src.Rows, src.Cols, MatType.CV_8UC3, Scalar.All(0));

    var contourIndex = 0;
```

```
    while ((contourIndex >= 0))
    {
        var contour = contours[contourIndex];

        var boundingRect = Cv2.BoundingRect(contour); //Find bounding rect for each contour

        Cv2.Rectangle(src,
            new Point(boundingRect.X, boundingRect.Y),
            new Point(boundingRect.X + boundingRect.Width, boundingRect.Y + boundingRect.Height),
            new Scalar(0, 0, 255),
            2);

        var roi = new Mat(threshImage, boundingRect); //Crop the image

        var resizedImage = new Mat();
        var resizedImageFloat = new Mat();
        Cv2.Resize(roi, resizedImage, new Size(10, 10)); //resize to 10X10
        resizedImage.ConvertTo(resizedImageFloat, MatType.CV_32FC1); //convert to float
        var result = resizedImageFloat.Reshape(1, 1);


        var results = new Mat();
        var neighborResponses = new Mat();
        var dists = new Mat();
        var detectedClass = (int)kNearest.FindNearest(result, 1, results, neighborResponses, dists);

        //Console.WriteLine("DetectedClass: {0}", detectedClass);
        //Cv2.ImShow("roi", roi);
        //Cv.WaitKey(0);

        //Cv2.ImWrite(string.Format("det_{0}_{1}.png",detectedClass, contourIndex), roi);

        Cv2.PutText(
            dst,
            detectedClass.ToString(CultureInfo.InvariantCulture),
            new Point(boundingRect.X, boundingRect.Y + boundingRect.Height),
            0,
            1,
            new Scalar(0, 255, 0),
            2);

        contourIndex = hierarchyIndexes[contourIndex].Next;
    }

    Cv2.ImShow("Segmented Source", src);
    Cv2.ImShow("Detected", dst);

    Cv2.ImWrite("dest.jpg", dst);

    Cv2.WaitKey();
}
```

این عملیات به صورت خلاصه در تصویر ذیل مشخص شده‌است:



ابتدا تصویر اصلی که قرار است عملیات OCR روی آن صورت گیرد، بارگذاری می‌شود. سپس کانتورها و اعداد موجود در آن تشخیص داده می‌شوند. مستطیل‌های قرمز رنگ در برگیرنده‌ی این اعداد را در تصویر دوم مشاهده می‌کنید. سپس این کانتورهای یافت شده را که شامل یکی از اعداد تشخیص داده شده‌است، تبدیل به یک ماتریس یک بعدی کرده و به متد FindNearest ارسال می‌کنیم. خروجی آن نام گروه یا پوشه‌ای است که این عدد در آن قرار دارد. در همینجا این خروجی را تبدیل به یک رشته کرده و در تصویر سوم با رنگ سبز رنگ نمایش می‌دهیم.

بنابراین در این تصویر، پنجره‌ی segmented image، همان اشیاء تشخیص داده شده‌ی از تصویر اصلی هستند.

پنجره‌ی با زمینه‌ی سیاه رنگ، نتیجه‌ی نهایی OCR است که نسبتا هم دقیق عمل کرده‌است.

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید.

نظرات خوانندگان

نویسنده: مهدی سعیدی فر
تاریخ: ۱۳۹۴/۰۴/۰۵ ۱۵:۴۰

امکانش هست که با استفاده از این نکات، بتوان حروف پلاک یک ماشین را تشخیص داد؟
برای مثال از تک تک حروف ممکن برای یک پلاک، عکس گرفته و این روش را برای آن پیاده سازی کرد و یا کلا روش پیاده سازی آن متفاوت است؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۴/۰۵ ۱۶:۶

- مرحله‌ی اول، بیرون کشیدن مستطیل شماره پلاک خودرو از داخل یک عکس کلی است؛ چیزی شبیه به مطلب « تشخیص چهره ». اگر به پوشه‌ی دیتا OpenCV مراجعه کنید، فایل xml تشخیص مستطیل شماره پلاک خودروهای روسی را دارد؛ فایل‌های haarcascade_licence_plate_rus_16stages.xml و haarcascade_russian_plate_number.xml. نحوه‌ی استفاده‌ی از این فایل‌ها، دقیقا همانند مطلب تشخیص چهره‌است. برای تشخیص شماره پلاک‌های ایرانی، باید از روش کلی مطرح شده در مطلب « طراحی classifier سفارشی تشخیص خودروها » استفاده کنید. یک سری عکس تهیه کنید و بعد فایل XML آن‌را استخراج کنید.
- مرحله‌ی دوم، با مطلب جاری تفاوتی ندارد:

ابتدا اصل پلاک باید تشخیص داده شود (همان مطلب تشخیص چهره با یک فایل XML مناسب). بعد بهبود کیفیت تصویر پلاک و آماده سازی آن برای استخراج کانتورها است. سپس این اشیاء یافت شده را به الگوریتم مثلا CvKNearest ارسال و شماره‌ی گروه هر کانتور را دریافت می‌کنید (روش OCR مطلب جاری).

یک نکته‌ی تکمیلی

فایل‌های XML یافتن مستطیل شماره پلاک‌های چند کشور مختلف را در پروژه‌ی openalpr می‌توانید پیدا کنید. این پروژه از OpenCV برای تشخیص پلاک و سپس از Tesseract OCR برای انجام کار OCR نهایی استفاده می‌کند (Tesseract OCR یک OCR سورس باز تهیه شده توسط گوگل است).

نویسنده: امیران
تاریخ: ۱۳۹۴/۰۴/۰۷ ۹:۵۰

او سی آر tesseract از موتور leptonica برای پردازش تصاویر استفاده می‌کند. opencv معروفتر است. بنچمارکی برای مقایسه وجود دارد؟
در مقاله عنوان کردید برای بهبود کیفیت از threshold استفاده می‌کنیم در مقالات قبلی در همین زمینه بحثی راجع به morphology داشتید آیا راه حل نهایی ترکیبی از این دو است؟ مثلا برای متون خطی قدیمی ماشین تحریر با کیفیت پائین می‌توان از ترکیب این دو استفاده نمود؟
یکی از معضلات حل نشده در زمینه ocr فارسی، متون دست نویس است. راه حلی برای آن با استفاده از سلسله مطالب جاری می‌توان یافت یا حداقل مسیری برای حل آن؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۴/۰۷ ۱۰:۰

مسلما یک برنامه‌ی OCR قوی باید دارای قسمتی به نام کالیبره کردن باشد و در اینجا می‌توان انواع و اقسام الگوریتم‌ها را برای رسیدن به بهترین نتیجه ترکیب کرد. برای مثال در مطلب فوق اگر پارامترهای متد threshold را تغییر دهید، دقت OCR متفاوت خواهد بود.
در پروژه‌ی نهایی بحث جاری، یک پوشه‌ی اعداد دست نویس انگلیسی هم هست که از آن می‌توان برای آموزش دادن به الگوریتم‌های machine learning مطرح شده استفاده کرد.

نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۴/۰۴/۰۷ ۱۹:۳۱

مجموعه داده بزرگ HODA شامل ارقام فارسی که توسط دانشگاه تربیت مدرس ایجاده شده از وب سایت فارسی او سی آر قابل دریافت است .

- مشخصات و روند جمع آوری این مجموعه داده در سال 2007 میلادی در مجله Pattern Recognition Letters منتشر شد
- این مجموعه شامل 102,352 نمونه عدد فارسی است که از 12,000 فرم مربوط به آزمون ورودی کاردانی به کارشناسی و کارشناسی ارشد جمع آوری شده است.

عنوان: **ساخت یک بارکدخوان با استفاده از OpenCV و ZXing.Net**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۴/۱۴ ۱۰:۳۵
آدرس: www.dotnettips.info
گروه‌ها: OpenCV, Barcodes

فرض کنید می‌خواهیم بارکد این قبض را یافته و سپس عدد متناظر با آن‌را در برنامه بخوانیم.

مراحل کار به این صورت هستند:

بارگذاری تصویر و چرخش آن در صورت نیاز

ابتدا تصویر بارکد دار را بارگذاری کرده و آن‌را تبدیل به یک تصویر سیاه و سفید می‌کنیم:

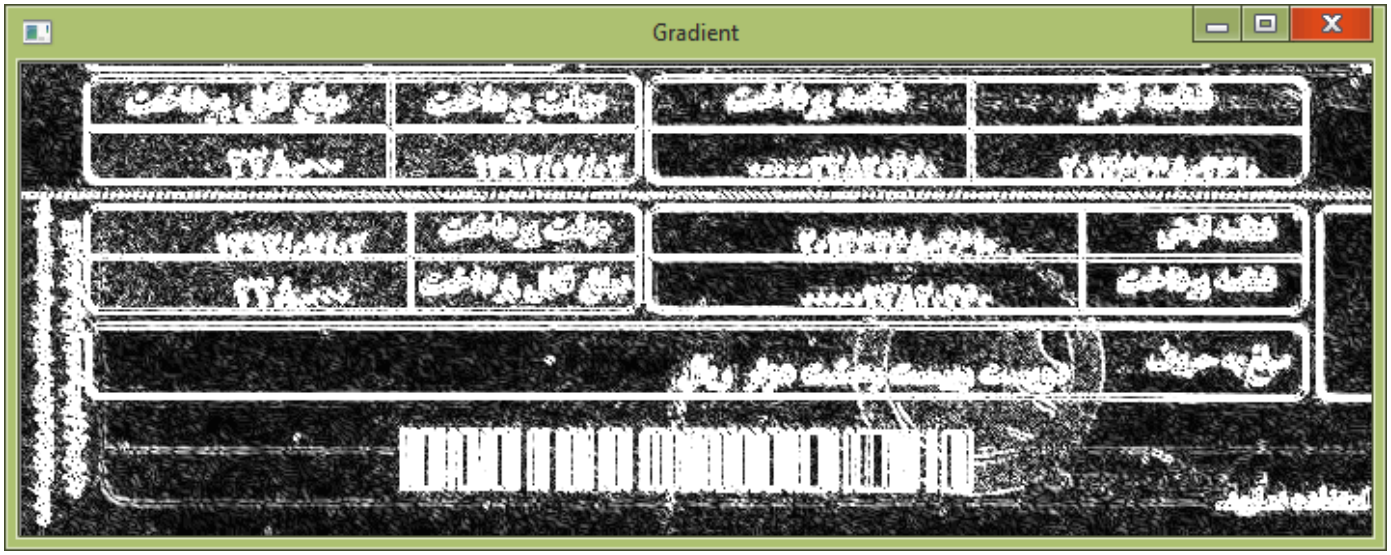
```
// load the image and convert it to grayscale
var image = new Mat(fileName);

if (rotation != 0)
{
    rotateImage(image, image, rotation, 1);
}

if (debug)
{
    Cv2.ImShow("Source", image);
    Cv2.WaitKey(1); // do events
}

var gray = new Mat();
var channels = image.Channels();
if (channels > 1)
{
    Cv2.CvtColor(image, gray, ColorConversion.BgrToGray);
}
else
{
    image.CopyTo(gray);
}
```

در این بین ممکن است بارکد موجود در تصویر، دقیقا در زاویه‌ای که در تصویر ابتدای بحث قرار گرفته‌است، وجود نداشته باشد؛ مثلا منهای 90 درجه، چرخیده باشد. به همین جهت می‌توان از متد چرخش تصویر مطلب « تغییر اندازه، و چرخش تصاویر » ارائه شده در قسمت نهم این سری استفاده کرد.

تشخیص گرادیان‌های افقی و عمودی

یکی از روش‌های تشخیص بارکد، استفاده از روشی است که در تشخیص خودرو قسمت 16 بیان شد. تعداد زیادی تصویر بارکد را تهیه و سپس آن‌ها را به الگوریتم‌های machine learning جهت تشخیص و یافتن محدوده‌ی بارکد موجود در یک تصویر، ارسال کنیم. هرچند این روش جواب خواهد داد، اما در این مورد خاص، قسمت بارکد، شبیه به گرادیانی از رنگ‌ها است. کتابخانه‌ی OpenCV برای یافتن این نوع گرادیان‌ها دارای متدی است به نام Sobel :

```
// compute the Scharr gradient magnitude representation of the images
// in both the x and y direction
var gradX = new Mat();
Cv2.Sobel(gray, gradX, MatType.CV_32F, xorder: 1, yorder: 0, ksize: -1);
//Cv2.Scharr(gray, gradX, MatType.CV_32F, xorder: 1, yorder: 0);

var gradY = new Mat();
Cv2.Sobel(gray, gradY, MatType.CV_32F, xorder: 0, yorder: 1, ksize: -1);
//Cv2.Scharr(gray, gradY, MatType.CV_32F, xorder: 0, yorder: 1);

// subtract the y-gradient from the x-gradient
var gradient = new Mat();
Cv2.Subtract(gradX, gradY, gradient);
Cv2.ConvertScaleAbs(gradient, gradient);

if (debug)
{
    Cv2.ImShow("Gradient", gradient);
    Cv2.WaitKey(1); // do events
}
```



ابتدا درجه‌ی شدت گرادیان‌ها در جهت‌های x و y محاسبه می‌شوند. سپس این شدت‌ها از هم کم خواهند شد تا بیشترین شدت گرادیان موجود در محور x حاصل شود. این بیشترین شدت‌ها، بیانگر نواحی خواهند بود که احتمال وجود بارکدهای افقی در آن‌ها بیشتر است.

کاهش نویز و یکی کردن نواحی تشخیص داده شده

در ادامه می‌خواهیم با استفاده از متدهای تشخیص کانتور (قسمت 12)، نواحی با بیشترین شدت گرادیان افقی را پیدا کنیم. اما تصویر حاصل از قسمت قبل برای اینکار مناسب نیست. به همین جهت با استفاده از متدهای کار با مورفولوژی تصاویر، این نواحی

گرادیانی را یکی می‌کنیم (قسمت 8).

```
// blur and threshold the image
var blurred = new Mat();
Cv2.Blur(gradient, blurred, new Size(9, 9));

var threshImage = new Mat();
Cv2.Threshold(blurred, threshImage, thresh, 255, ThresholdType.Binary);

if (debug)
{
    Cv2.ImShow("Thresh", threshImage);
    Cv2.WaitKey(1); // do events
}


// construct a closing kernel and apply it to the thresholded image
var kernel = Cv2.GetStructuringElement(StructuringElementShape.Rect, new Size(21, 7));
var closed = new Mat();
Cv2.MorphologyEx(threshImage, closed, MorphologyOperation.Close, kernel);

if (debug)
{
    Cv2.ImShow("Closed", closed);
    Cv2.WaitKey(1); // do events
}


// perform a series of erosions and dilations
Cv2.Erode(closed, closed, null, iterations: 4);
Cv2.Dilate(closed, closed, null, iterations: 4);

if (debug)
{
    Cv2.ImShow("Erode & Dilate", closed);
    Cv2.WaitKey(1); // do events
}
```

این سه مرحله را در تصاویر ذیل مشاهده می‌کنید:

ابتدا با استفاده از متد Threshold، تصویر را به یک تصویر باینری تبدیل خواهیم کرد. در این تصویر تمام نقاط دارای شدت رنگ کمتر از مقدار thresh، به مقدار حداکثر 255 تنظیم می‌شوند.

سپس با استفاده از متدهای تغییر مورفولوژی تصویر، قسمت‌های مجاور به هم را می‌بندیم و یکی می‌کنیم. این مورد در یافتن اشیاء احتمالی که ممکن است بارکد باشند، بسیار مفید است.

متدهای Erode و Dilate در اینجا کار حذف نویزهای اضافی را انجام می‌دهند؛ تا بهتر بتوان بر روی نواحی بزرگتر یافت شده، تمرکز کرد.

یافتن بزرگترین ناحیه‌ی به هم پیوسته‌ی موجود در یک تصویر

تمام این مراحل را انجام دادیم تا بتوانیم بزرگترین ناحیه‌ی به هم پیوسته‌ای را که احتمال می‌رود بارکد باشد، در تصویر تشخیص دهیم. پس از این آماده سازی‌ها، اکنون با استفاده از متد یافتن کانتورها، تمام نواحی یکی شده را یافته و بزرگترین مساحت ممکن را به عنوان بارکد انتخاب می‌کنیم:

```
//find the contours in the thresholded image, then sort the contours
//by their area, keeping only the largest one

Point[][] contours;
HiearchyIndex[] hierarchyIndexes;
Cv2.FindContours(
    closed,
    out contours,
    out hierarchyIndexes,
    mode: ContourRetrieval.CComp,
    method: ContourChain.ApproxSimple);

if (contours.Length == 0)
{
    throw new NotSupportedException("Couldn't find any object in the image.");
}

var contourIndex = 0;
var previousArea = 0;
var biggestContourRect = Cv2.BoundingRect(contours[0]);
while ((contourIndex >= 0))
{
    var contour = contours[contourIndex];

    var boundingRect = Cv2.BoundingRect(contour); //Find bounding rect for each contour
    var boundingRectArea = boundingRect.Width * boundingRect.Height;
    if (boundingRectArea > previousArea)
    {
        biggestContourRect = boundingRect;
        previousArea = boundingRectArea;
    }

    contourIndex = hierarchyIndexes[contourIndex].Next;
}


var barcode = new Mat(image, biggestContourRect); //Crop the image
Cv2.CvtColor(barcode, barcode, ColorConversion.BgrToGray);

Cv2.ImShow("Barcode", barcode);
Cv2.WaitKey(1); // do events
```

حاصل این عملیات یافتن بزرگترین ناحیه‌ی گرادیانی به هم پیوسته‌ی موجود در تصویر است:

خواندن مقدار متناظر با بارکد یافت شده

خوب، تا اینجا موفق شدیم، محل قرارگیری بارکد را تصویر پیدا کنیم. مرحله‌ی بعد خواندن مقدار متناظر با این تصویر است. برای این منظور از کتابخانه‌ی سورس بازی به نام http://zxingnet.codeplex.com استفاده خواهیم کرد. این کتابخانه قادر است بارکد بسازد و همچنین تصاویر بارکدها را خوانده و مقادیر متناظر با آن‌ها را استخراج کند. برای نصب آن می‌توان از دستور ذیل استفاده کرد:

```
PM> Install-Package ZXing.Net
```

پس از نصب این کتابخانه‌ی بارکدساز و بارکد خوان، اکنون تنها کاری که باید صورت گیرد، ارسال تصویر بارکد جدا شده‌ی توسط OpenCV به آن است:

```
private static string getBarcodeText(Mat barcode)
{
    // `ZXing.Net` needs a white space around the barcode
    var barcodeWithWhiteSpace = new Mat(new Size(barcode.Width + 30, barcode.Height + 30),
MatType.CV_8U, Scalar.White);
    var drawingRect = new Rect(new Point(15, 15), new Size(barcode.Width, barcode.Height));
    var roi = barcodeWithWhiteSpace[drawingRect];
    barcode.CopyTo(roi);

    Cv2.ImShow("Enhanced Barcode", barcodeWithWhiteSpace);
    Cv2.WaitKey(1); // do events

    return decodeBarcodeText(barcodeWithWhiteSpace.ToBitmap());
}

private static string decodeBarcodeText(System.Drawing.Bitmap barcodeBitmap)
{
    var source = new BitmapLuminanceSource(barcodeBitmap);

    // using http://zxingnet.codeplex.com/
    // PM> Install-Package ZXing.Net
    var reader = new BarcodeReader(null, null, ls => new GlobalHistogramBinarizer(ls))
    {
        AutoRotate = true,
        TryInverted = true,
        Options = new DecodingOptions
```

```
        {
            TryHarder = true,
            //PureBarcode = true,
            /*PossibleFormats = new List<BarcodeFormat>
                    {
                        BarcodeFormat.CODE_128
                        //BarcodeFormat.EAN_8,
                        //BarcodeFormat.CODE_39,
                        //BarcodeFormat.UPC_A
                    }*/
        }
    };

    //var newhint = new KeyValuePair<DecodeHintType, object>(DecodeHintType.ALLOWED_EAN_EXTENSIONS, new
Object());
    //reader.Options.Hints.Add(newhint);

    var result = reader.Decode(source);
    if (result == null)
    {
        Console.WriteLine("Decode failed.");
        return string.Empty;
    }

    Console.WriteLine("BarcodeFormat: {0}", result.BarcodeFormat);
    Console.WriteLine("Result: {0}", result.Text);


    var writer = new BarcodeWriter
    {
        Format = result.BarcodeFormat,
        Options = { Width = 200, Height = 50, Margin = 4},
        Renderer = new ZXing.Rendering.BitmapRenderer()
    };
    var barcodeImage = writer.Write(result.Text);
    Cv2.ImShow("BarcodeWriter", barcodeImage.ToMat());

    return result.Text;
}
```

چند نکته را باید در مورد کار با ZXing.Net بخاطر داشت؛ وگرنه جواب نمی‌گیرید:

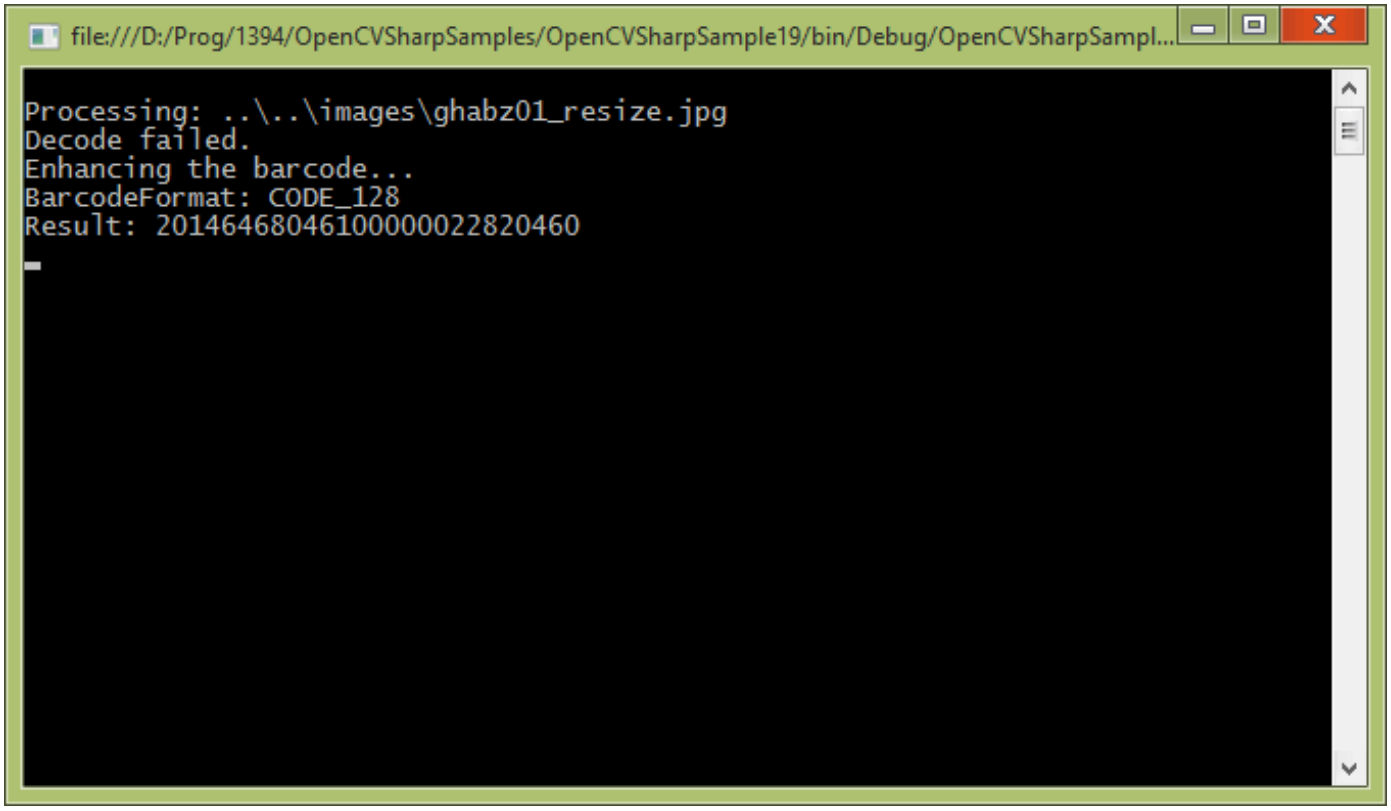
الف) این کتابخانه حتما نیاز دارد تا تصویر بارکد، در یک حاشیه‌ی سفید در اختیار او قرار گیرد. به همین جهت در متد getBarcodeText، ابتدا تصویر بارکد یافت شده، به میانه‌ی یک مستطیل سفید رنگ بزرگ‌تر کپی می‌شود.

ب) برای تبدیل Mat به Bitmap مورد نیاز این کتابخانه می‌توان از متد الحاقی ToBitmap استفاده کرد (قسمت 7).

ج) پس از آن وهله‌ای از کلاس BarcodeReader آماده شده و در آن پارامترهایی مانند بیشتر سعی کن (TryHarder) و اصلاح درجه‌ی چرخش تصویر (AutoRotate) تنظیم شده‌اند.

د) بارکدهای موجود در قبض‌های ایران عموما بر اساس فرمت CODE_128 ساخته می‌شوند. بنابراین برای خواندن سریعتر آنها می‌توان PossibleFormats را مقدار دهی کرد. اگر این مقدار دهی صورت نگیرد، تمام حالت‌های ممکن بررسی می‌شوند.

در آخر کار این متد، از متد Writer آن نیز برای تولید بارکد مشابهی استفاده شده‌است تا بتوان بررسی کرد این دو تا چه اندازه به هم شبیه هستند.



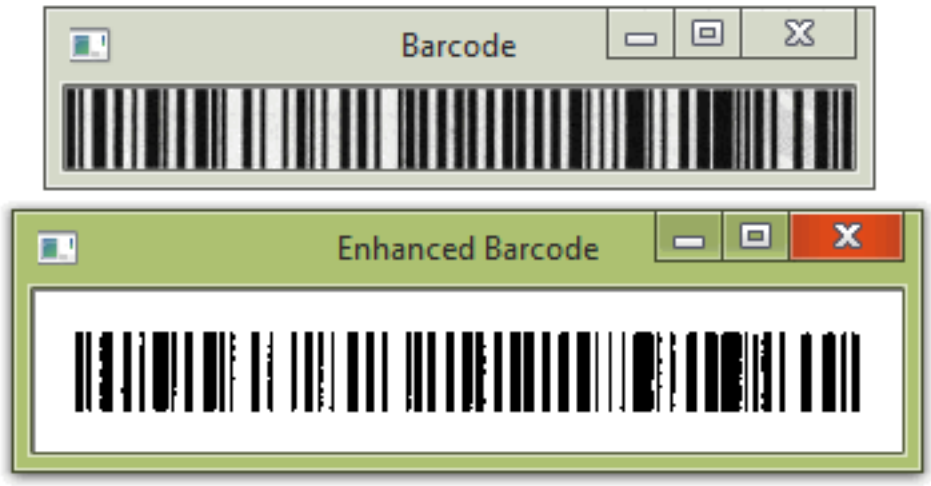
همانطور که مشاهده می‌کنید، عدد تشخیص داده شده، با عدد شناسه‌ی قبض و شناسه‌ی پرداخت تصویر ابتدای بحث یکی است.

بهبود تصویر، پیش از ارسال آن به متد Decode کتابخانه‌ی ZXing.Net

در تصویر قبلی، سطر decode failed را هم ملاحظه می‌کنید. علت اینجا است که اولین سعی انجام شده، موفق نبوده است؛ چون تصویر تشخیص داده شده، بیش از اندازه نویز و حاشیه‌ی خاکستری دارد. می‌توان این حاشیه‌ی خاکستری را با دوبار اعمال متد Threshold از بین برد:

```
var barcodeClone = barcode.Clone();
var barcodeText = getBarcodeText(barcodeClone);

if (string.IsNullOrWhiteSpace(barcodeText))
{
    Console.WriteLine("Enhancing the barcode...");
    //Cv2.AdaptiveThreshold(barcode, barcode, 255,
        //AdaptiveThresholdType.GaussianC, ThresholdType.Binary, 9, 1);
    //var th = 119;
    var th = 100;
    Cv2.Threshold(barcode, barcode, th, 255, ThresholdType.ToZero);
    Cv2.Threshold(barcode, barcode, th, 255, ThresholdType.Binary);
    barcodeText = getBarcodeText(barcode);
}

Cv2.Rectangle(image,
    new Point(biggestContourRect.X, biggestContourRect.Y),
    new Point(biggestContourRect.X + biggestContourRect.Width, biggestContourRect.Y +
biggestContourRect.Height),
    new Scalar(0, 255, 0),
    2);

if (debug)
{
    Cv2.ImShow("Segmented Source", image);
    Cv2.WaitKey(1); // do events
}

Cv2.WaitKey(0);
```

```
Cv2.DestroyAllWindows();
```



اعداد یافت شده، دقیقا از روی تصویر بهبود یافته‌ی توسط متدهای Threshold خوانده شده‌اند و نه تصویر ابتدایی یافت شده. بنابراین به این موضوع نیز باید دقت داشت.

کدهای کامل این مثال را از اینجا **می‌توانید دریافت کنید.**

عنوان: **Accord.NET #1**
نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۴/۰۵/۲۲ ۱۹:۳۰
آدرس: www.dotnettips.info
گروه‌ها: OpenCV, Accord.NET, Machine Learning, Artificial Intelligence

Accord.NET کتابخانه‌ای است متن‌باز و بسیار کارآمد که در آن توابع بسیار زیادی در حوزه‌ی تحلیل آماری (statistical analysis)، یادگیری ماشین (machine learning)، پردازش تصویر (Image processing) و بینایی ماشین (computer vision) قرار گرفته‌اند تا در برنامه‌های .NET. ایی مورد استفاده قرار گیرند.

چارچوب Accord.NET توسط آقای سزار سوزا بر پایه کتابخانه‌ی مشهور و محبوب AForge.NET (که توسط آقای اندرو کریلو ایجاد شده بود) بنا شده و البته ابزارهای جدید زیادی به همراه یک محیط کامل برای محاسبات علمی (scientific computing) در .NET. به آن اضافه شده است.

این چارچوب متشکل از چندین کتابخانه است که می‌توان آن را از طریق NuGet دریافت و نصب کرد.

کتابخانه‌های Accord.NET را می‌توان به سه دسته‌ی کلی تقسیم کرد :

1. محاسبات علمی (scientific computing)

1.1. Accord.Math	جهت کار با ماتریس‌ها عددی تجزیه ماتریس‌ها (decomposition matrix) الگوریتم‌های بهینه سازی عددی برای مسائل محدود و نامحدود توابع و ابزارهای خاص جهت استفاده در کاربردهای علمی
1.2. Accord.Statistics	شامل توابعی جهت توزیع‌های احتمال (probability distributions) آزمایش فرضیات (hypothesis testing) مدل‌های آماری (statistical models) و توابعی شامل : رگرسیون خطی، مدل پنهان مارکوف (Hidden Markov Models)، آنالیز اجزای اساسی (Principal Component Analysis) و خیلی از تکنیک‌های مرتبط دیگر.
1.3. Accord.MachineLearning	شامل دسته بندهای معروف از جمله : ماشین برداری پشتیبان - Support Vector Machines درخت تصمیم - Decision Trees مدل نیو بیز - Naive Bayesian models K-means مدل ترکیبی گوسین - Gaussian Mixture models و الگوریتم‌های متداول دیگری مانند : Ransac, Cross-validation و Grid-Search
1.4. Accord.Neuro	شامل الگوریتم‌های معروف در حوزه شبکه‌های عصبی مصنوعی مانند: لونبرگ مارکوارت - Levenberg-Marquardt Parallel Resilient Back-propagation شبکه باور عمیق - Deep Belief Networks ماشین بولتزمن - Restrictured Boltzmann Machines و تعدادی از شبکه‌های عصبی دیگر

2. پردازش تصویر و سیگنال

2.1. Accord.Imaging	شامل آشکارسازهای نقاط از جمله Harris, SURF, FAST و FREAK فیلترهایی برای تصاویر توابعی جهت انطباق (matching) و دوخت (stitching) تصاویر استخراج ویژگی‌های خوبی مانند - هیستوگرام گرادیان‌های شیب‌گرا و یا هاگ (Histograms of Oriented Gradients) و ویژگی‌های توصیفی بافتی هارلیک (Haralick’s textural)
2.2. Accord.Vision	تشخیص و ردیابی بی‌درنگ چهره توابعی برای تشخیص، ردیابی و تبدیل اشیایی که در جریانی(streams) از تصاویر هستند
2.3. Accord.Audio	شامل توابعی جهت پردازش صدا از جمله اسپکتروم آنالایزر

3. سایر کتابخانه‌های پشتیبانی

3.1. Accord.Controls	شامل نمودار هیستوگرام، پلات‌ها و نمایشگرها و نمودارهایی برای داده‌های جدولی جهت کاربردهای علمی.
3.2. Accord.Controls.Imaging	شامل ابزاری برای نمایش سریع تصاویر برای برنامه‌های Windows Forms
3.3. Accord.Controls.Audio	شامل کنترل‌های Windows Forms برای نمایش شکل موج صوت و اطلاعات آن
3.4. Accord.Controls.Vision	شامل اجزاء و کنترل‌های Windows Forms برای ردیابی حرکات سر، صورت، دست و سایر کارهای مرتبط با بینایی ماشین

اگر با مفاهیم یادگیری ماشین و هوش موصنوعی کمتر آشنا هستید و در این قسمت کمی کلمات تخصصی به کار رفته نگران نباشید؛ در مطالب آتی به صورت کاربردی به استفاده‌ی از آنها خواهیم پرداخت.

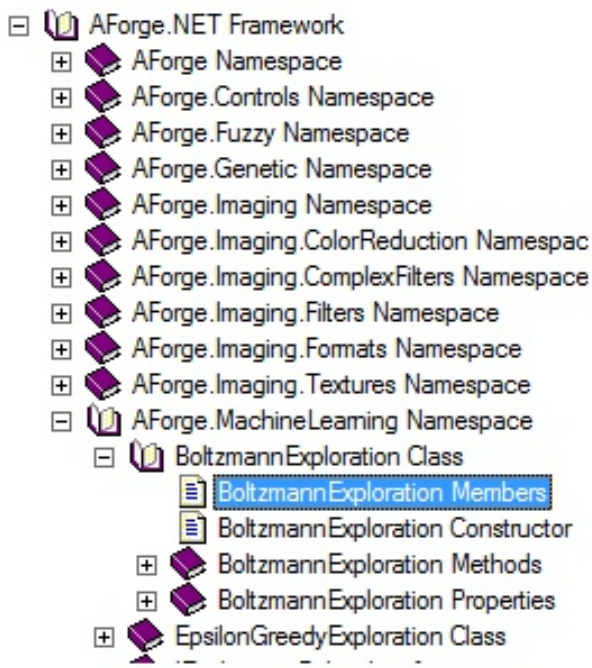
نظرات خوانندگان

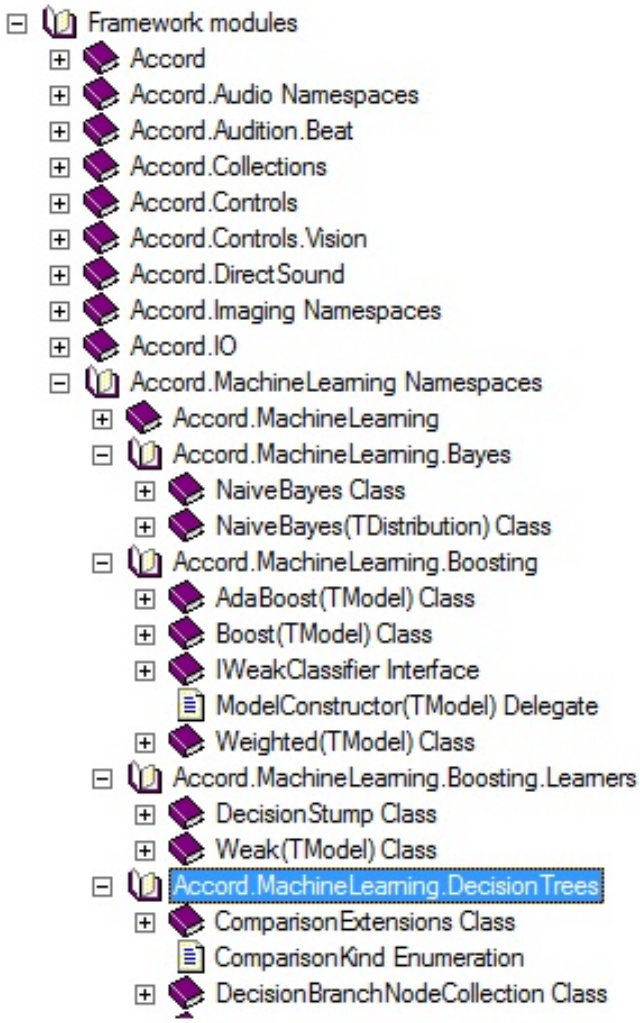
نویسنده: مصطفی عسگری
تاریخ: ۱۳۹۴/۰۵/۲۴ ۹:۵۰

مزایای این framework نسبت به AForge.NET چیست؟

نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۴/۰۵/۲۴ ۱۷:۱۹

Accord.NET در حقیقت یک توسعه ای برای AForge.NET است. و چنانچه می‌خواهید از آکورد استفاده کنید بایستی ابتدا AForge.NET نصب نمایید.
AForge.NET یک کتابخانه بسیار عالی است اما در هر کدام از فضای نام هایش نقص هایی وجود دارد که در آکورد دات نت به آن افزوده شده است؛ به عنوان مثال در درختواره فضای نام MachineLearning مستندات دو پروژه مشاهده می‌کنیم که بسیاری از مفاهیم یادگیری ماشین از جمله : دسته بند نیو بیز، بوستینگ، بگینگ، درخت تصمیم، انواع مختلف اعتبارسنجی‌ها و ... در Accord.NET گنجانده شده است.





نویسنده: زواری

تاریخ: ۱۳۹۴/۰۵/۲۴ ۱۸:۱۶

همچنین یک توسعه دیگر بنام Accord.NET Extensions وجود دارد که توسط دکتر " دارکو یوریچ " برای آکورد نوشته شده است و ادعا می‌کند که با پیاده سازی "اساس شی تصویر" بعنوان آرایه محلی دات نت (مانند متلب)، سرعت پردازش‌ها بیشتر کرده است.

نویسنده: زواری
تاریخ: ۱۳۹۴/۰۵/۲۴ ۱۹:۰

اینو هم اضافه کنم که اگر نیاز هست با دات نت، پروژه پردازش تصویر بنویسیم؛ بهتر هست تا از فریم ورکهای فوق استفاده کنیم.
یک برنامه خیلی ابتدایی Emgu-V.S.-Aforge-V.S.-WICInterop.rar برای مقایسه سرعت بین "Emgu"، "AForge" و "WIC" نوشتم؛ به این ترتیب که یک تصویر خیلی بزرگ (حدود 10 مگابایت، تصویر یک نقشه) رو پویش میکنند. برای این پویش "aforg" دو ثانیه، "emgu" پنج و WIC بیست ثانیه زمان سپری شد.
درضمن ادعای برتری " Accord.NET Extensions Framework " رو هم بصورت مستند میتونید در لینک " Introducing Portable #Generic Image Library for C " مشاهده کنید.