

عنوان:	OpenCVSharp #1
نویسنده:	وحید نصیری
تاریخ:	۱۵:۱۰ ۱۳۹۴/۰۳/۱۱
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 مختلف مانند Windows, Mac, Linux, Android و iOS بکار می‌رود.

محصول کننده‌های OpenCV مخصوص دات نت

تا امروز محصول 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 محصول 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 چندسکوپی 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 نت 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 نت OpenCvSharp را دریافت می‌کنید. در این حالت نیاز است به سایت [OpenCV](#) مراجعه و بسته‌های کامپایل شده‌ی آن را [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 X64 و X86 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 نت نیست](#). بنابراین DLL‌های آن باید بسته به معماری CPU جاری، انتخاب شوند. یعنی اگر برنامه‌ی دات نت 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 X64 و X86 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 X64 و X86 وجود دارند که حاوی DLL‌های اصلی OpenCV می‌باشند. در این پوشه‌ها اگر برای مثال فایلی به نام msvcpr120.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:

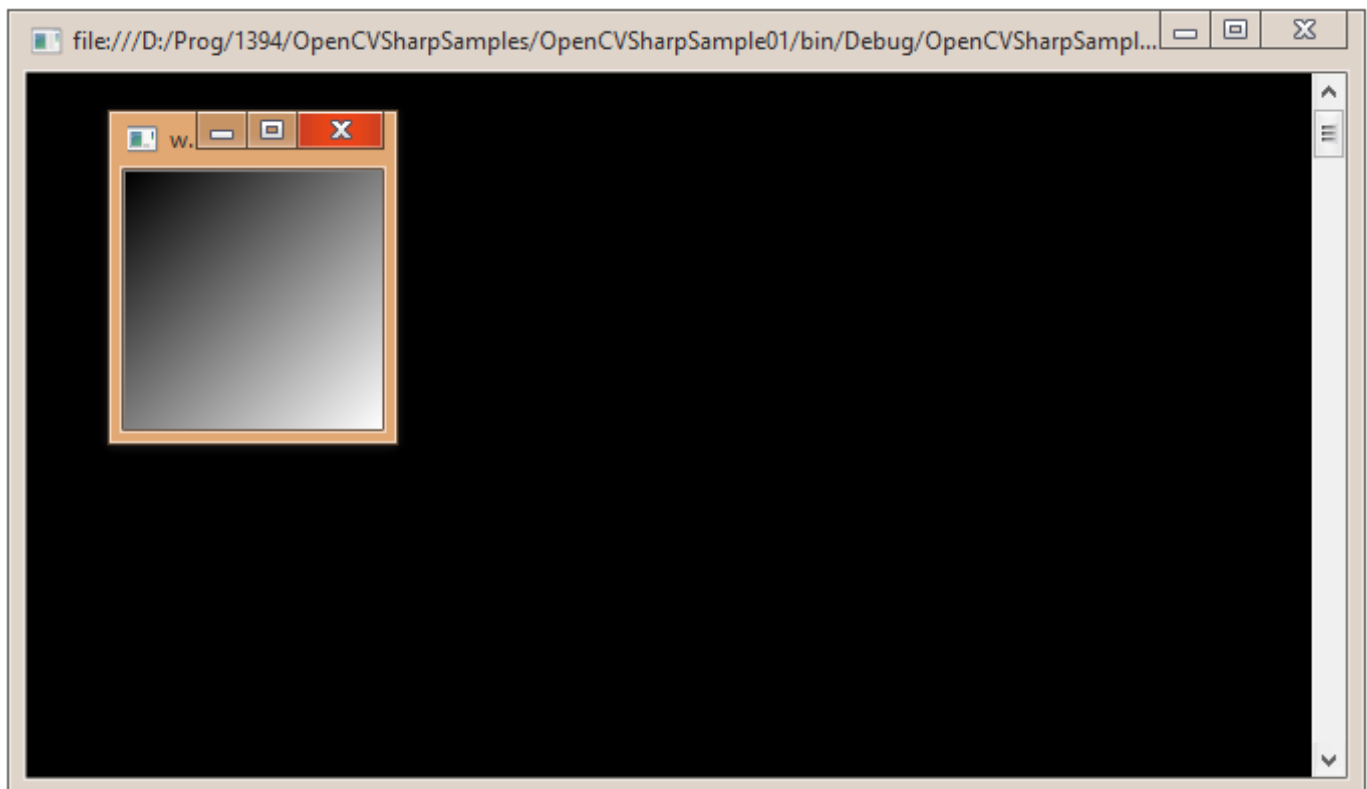
```
using OpenCvSharp;

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.

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");

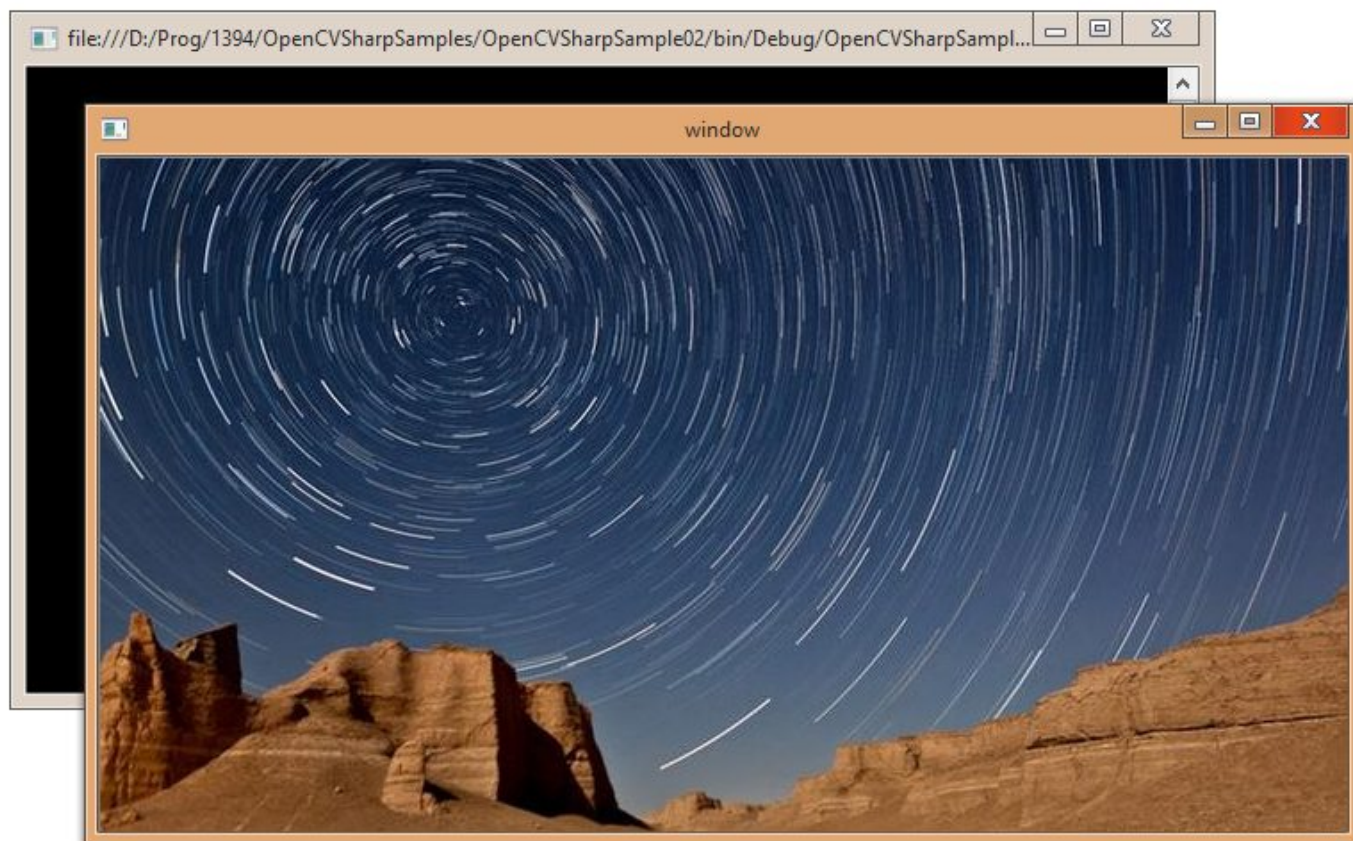
            Cv.NamedWindow("window");
            Cv.ShowImage("window", img);

            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 آن را 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.

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 آن را 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.

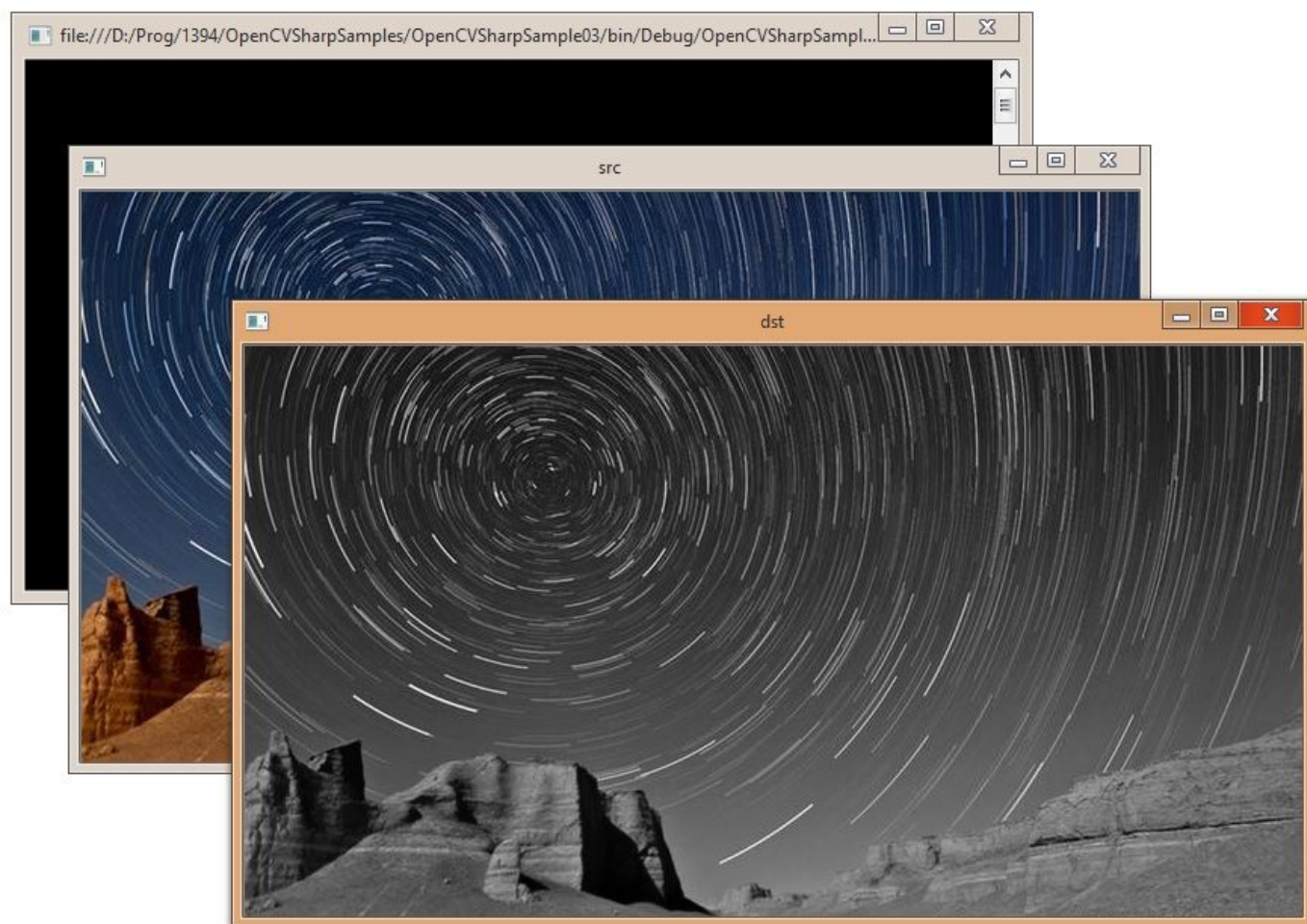
تهیه‌ی یک نمونه‌ی سیاه و سفید از تصویر اصلی در OpenCVSharp

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 عموماً 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.

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, 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;
}
```

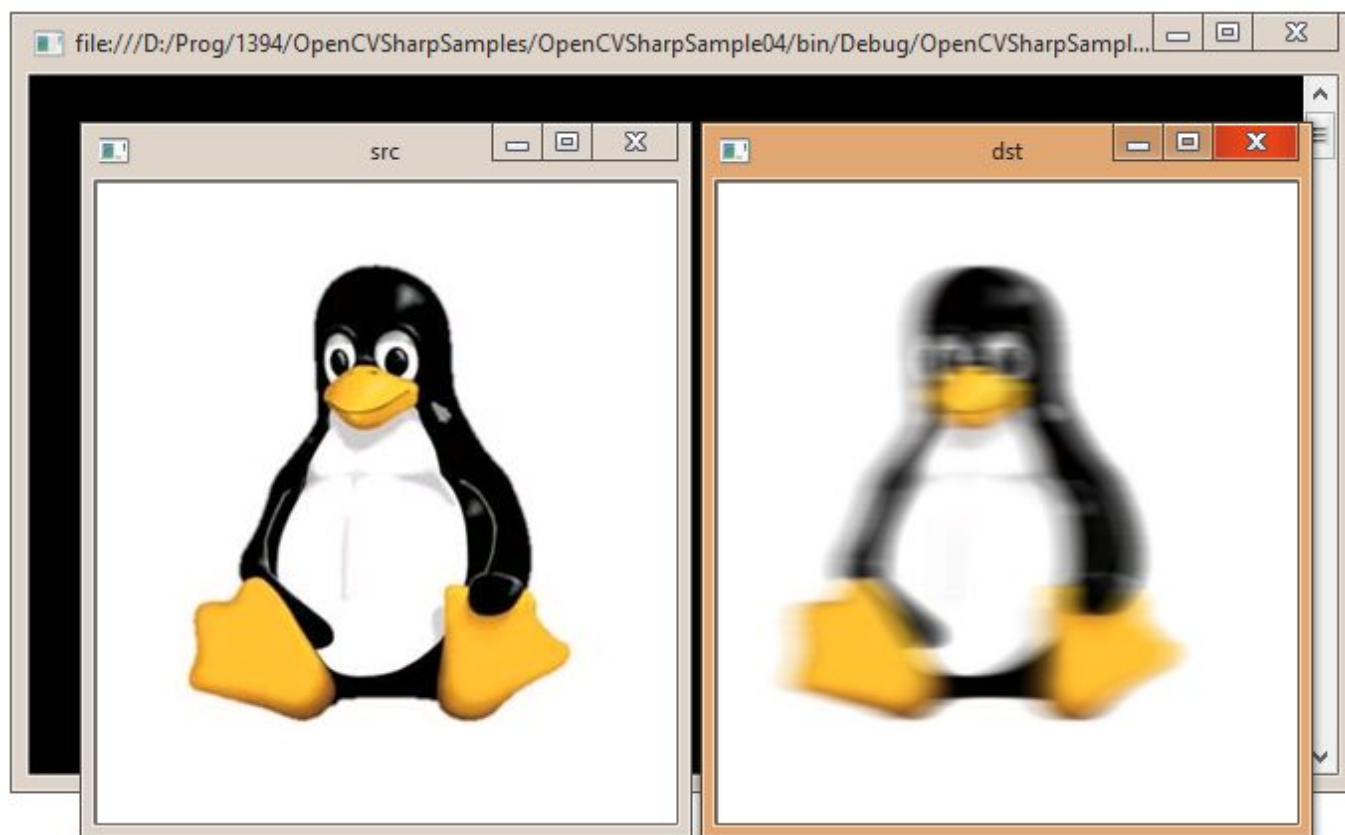
معادل OpenCVSharp آن به صورت ذیل خواهد بود:

```
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, 2, 2, 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.

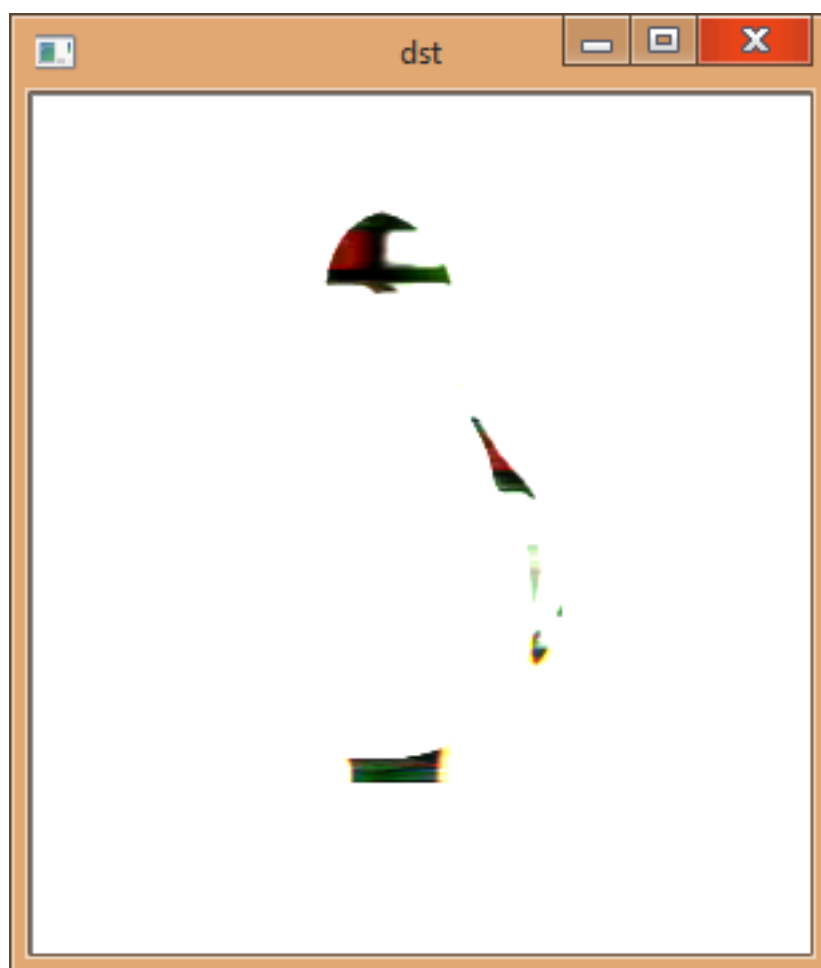
در این مثال آرایه‌ی `data`، یک فیلتر را تعریف می‌کند که در اینجا، حالت یک بردار را دارد تا یک ماتریس. برای تبدیل آن به ماتریس، از شیء `CvMat` استفاده خواهد شد که آنرا تبدیل به ماتریسی با یک ردیف و 21 ستون خواهد کرد. در اینجا از نام کرنل استفاده شده‌است. کرنل در OpenCV به معنای ماتریسی از داده‌ها با یک 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، تقریباً 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:

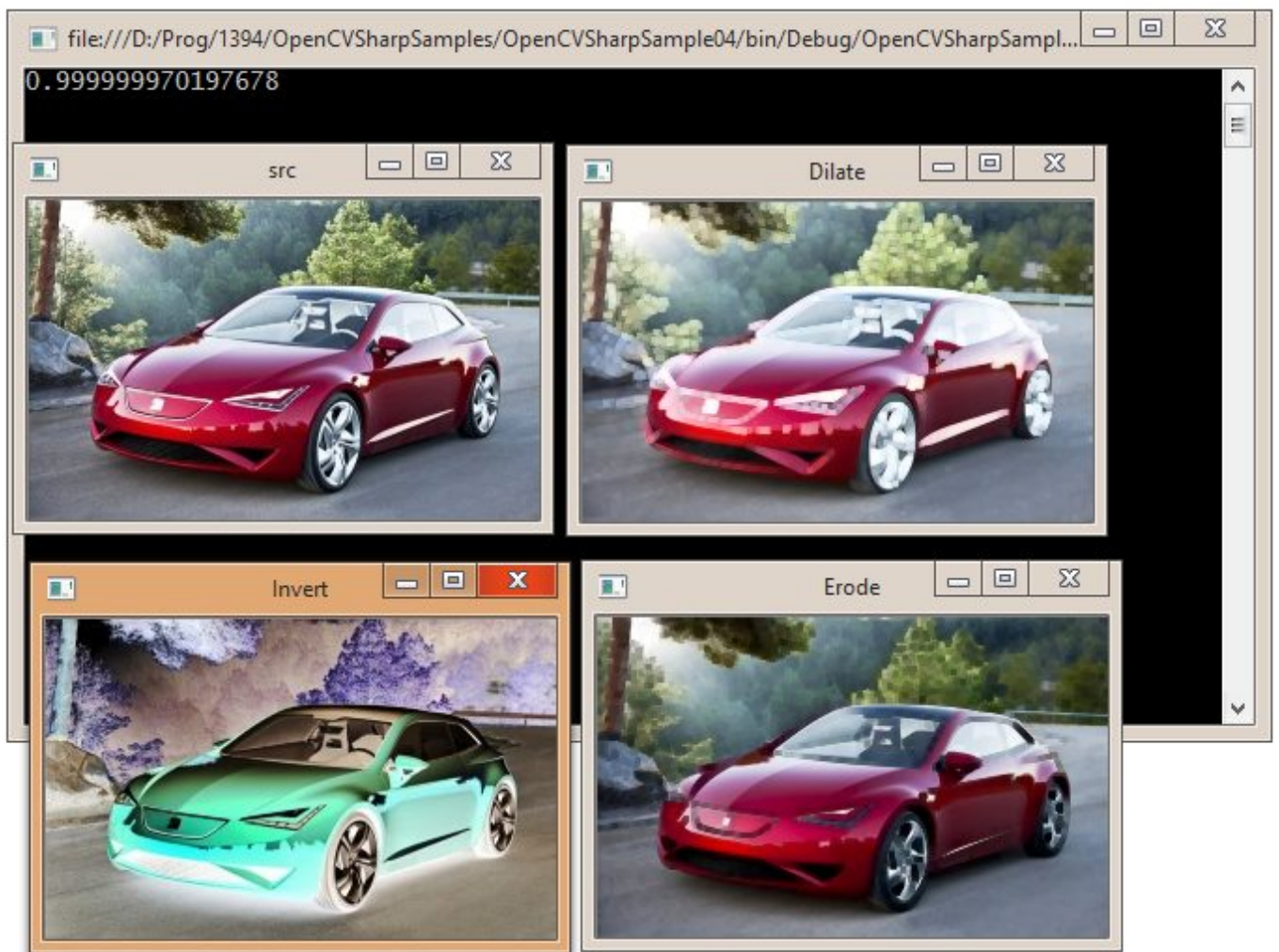
$$dst(x, y) = \sum_{\substack{0 \leq x' < kernel.cols, \\ 0 \leq y' < kernel.rows}} kernel(x', y') * src(x + x' - anchor.x, y + y' - anchor.y)$$

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

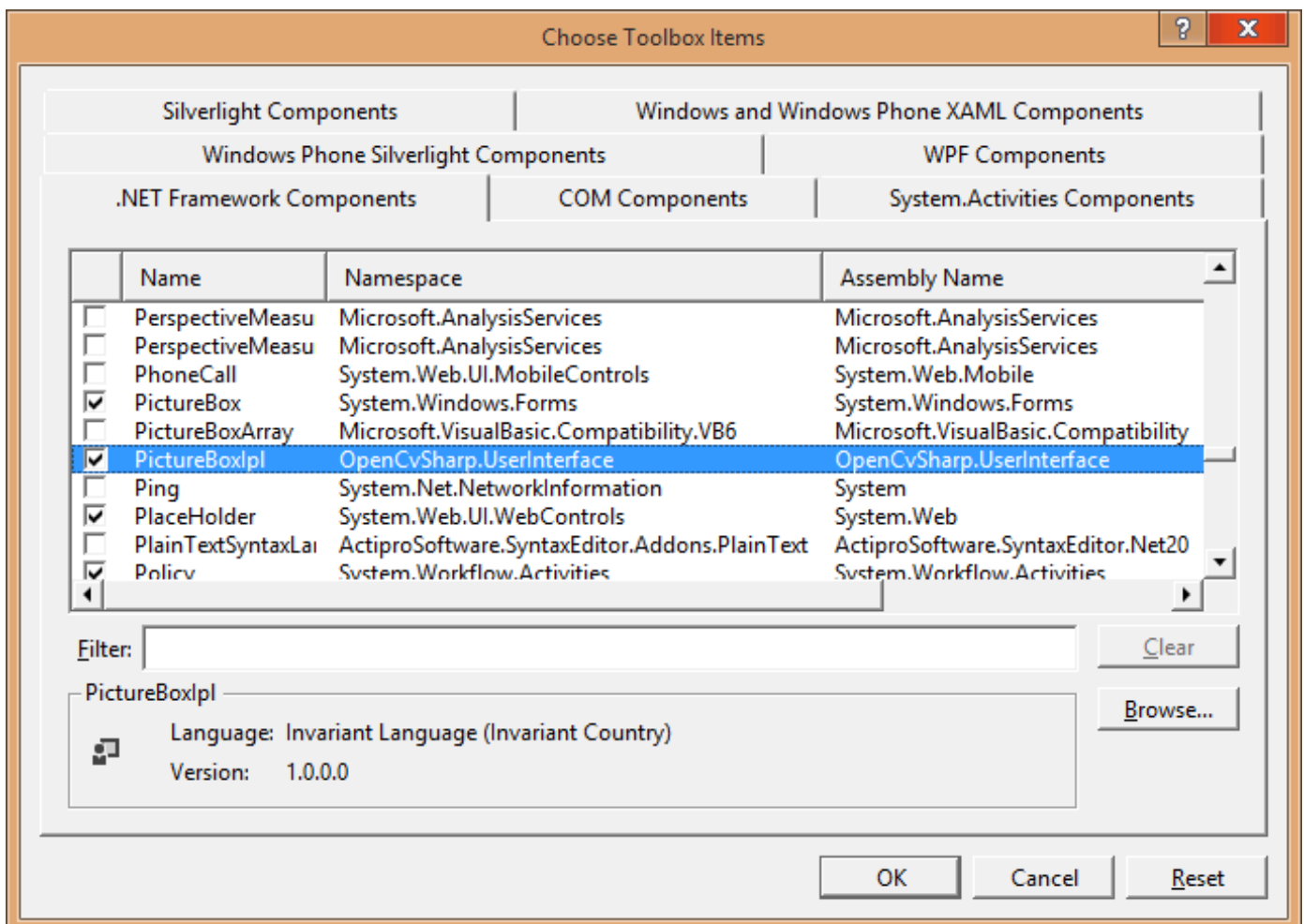


کدهای کامل این مثال را [از اینجا](#) می‌توانید دریافت کنید.

استفاده از پنجره‌ی native خود OpenCV، روش مرسوم‌ی است در زبان‌های مختلف برنامه‌نویسی 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 `ToolBox -> Choose items` 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 (خصوصاً برای نمایش اطلاعات رسیده‌ی از دوربین یا WebCam)، از متد RefreshIplImage استفاده کنید. این متد بجای ایجاد یک شیء جدید، تنها ناحیه‌ی موجود را مجدداً 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);
```

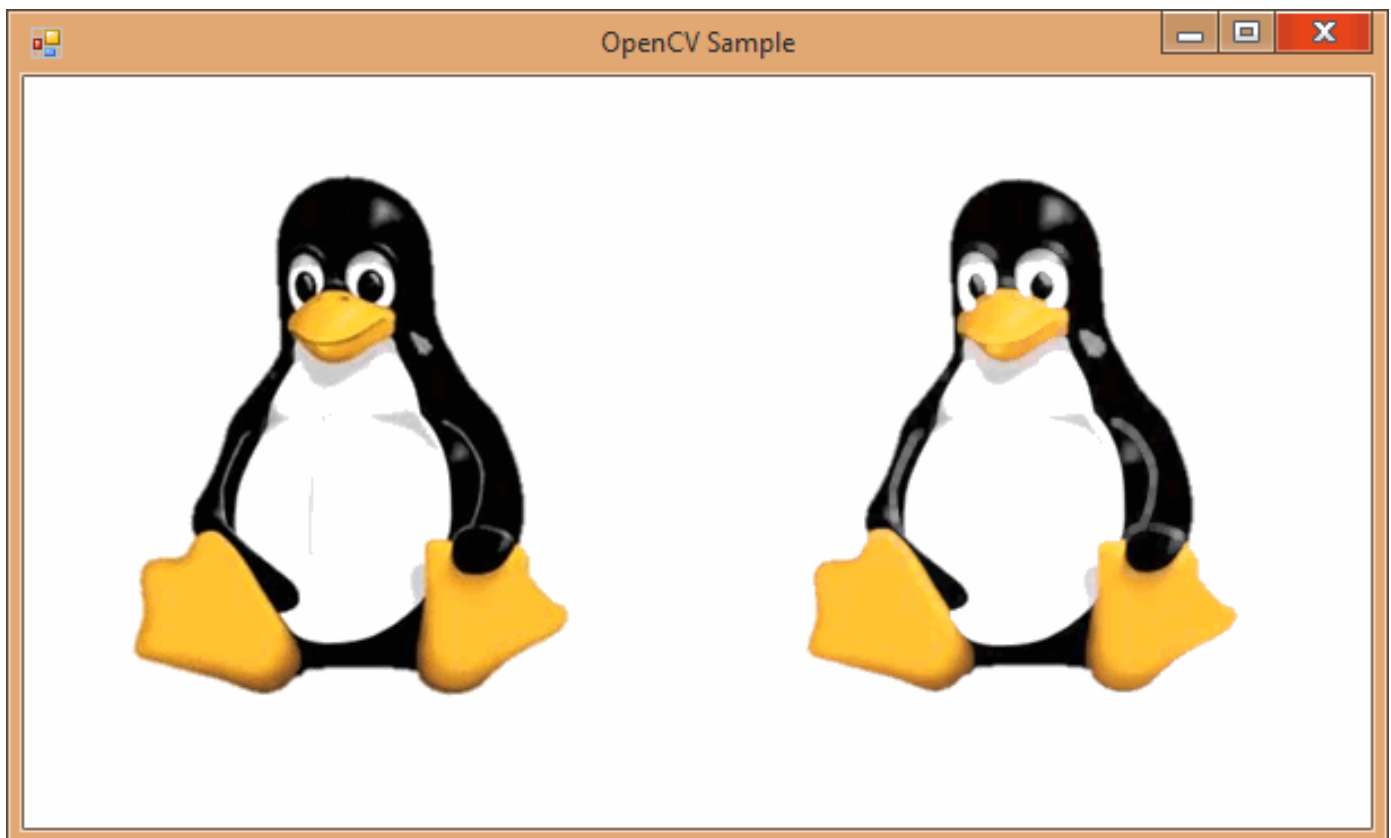
تنها نکته‌ای که در اینجا جدید است، استفاده از متد الحاقی ToBitmap می‌باشد که در کلاس BitmapConverter کتابخانه‌ی OpenCVSharp تعریف شده‌است. به این ترتیب تصویر با فرمت OpenCV، به یک Bitmap دات نت تبدیل می‌شود. اکنون می‌توان این بیت‌مپ را برای مثال به یک Picture box استاندارد انتساب داد و یا حتی متد Save 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 صرفاً ناحیه‌ی نمایشی مجدداً 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 opencv 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:	۱۰:۴۵ ۱۳۹۴/۰۳/۱۶
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 نت 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 پیش‌بینی 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 هر بار یک frame از ویدیو را بازگشت می‌دهد و می‌توان آن را در یک حلقه، تا زمانی که 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 دقیقاً برای یک چنین مواردی طراحی شده‌است تا سر بار 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();
}
```

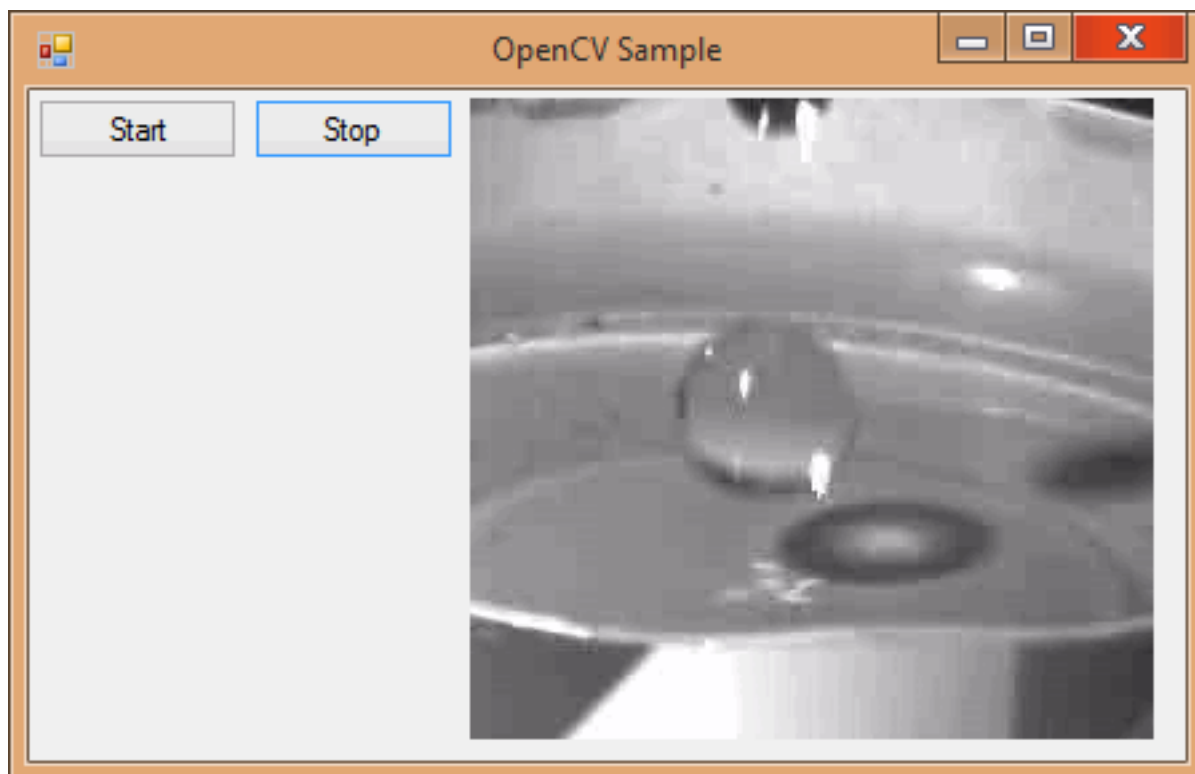
این روش هر چند کار می‌کند اما همانند روش استفاده از متد رخدادران Application Do Idle که صرفاً در زمان بیکاری برنامه فراخوانی می‌شود، سبب خواهد شد تا تعدادی فریم را از دست دهید، همچنین با CPU Usage بالایی نیز مواجه شوید. روش بعدی، استفاده از یک تایمر است که Interval آن بر اساس نرخ فریم‌های ویدیو تنظیم 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 آن را 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 مستقیماً 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 مجدداً 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: ۹:۵۴ ۱۳۹۴/۰۳/۱۷

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: ۱۰:۱۳ ۱۳۹۴/۰۳/۱۷

- 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 زمانی که 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 `pictureBoxIp11.Invoke` غیر ضروری است و سربار بی‌جهتی را به سیستم تحمیل می‌کند.
- به صورت خلاصه در حین استفاده‌ی از `BackgroundWorker`:
- متد `DoWork` بر روی `ThreadPool` اجرا می‌شود (ترد آن با ترد UI یکی نیست)
- متدهای `ReportProgress` و `ReportError` 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.

عنوان:	OpenCVSharp #7
نویسنده:	وحید نصیری
تاریخ:	۱۳:۱۵ ۱۳۹۴/۰۳/۱۷
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 صرفاً 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 در نظر 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 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 نت `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 مستقیماً 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:

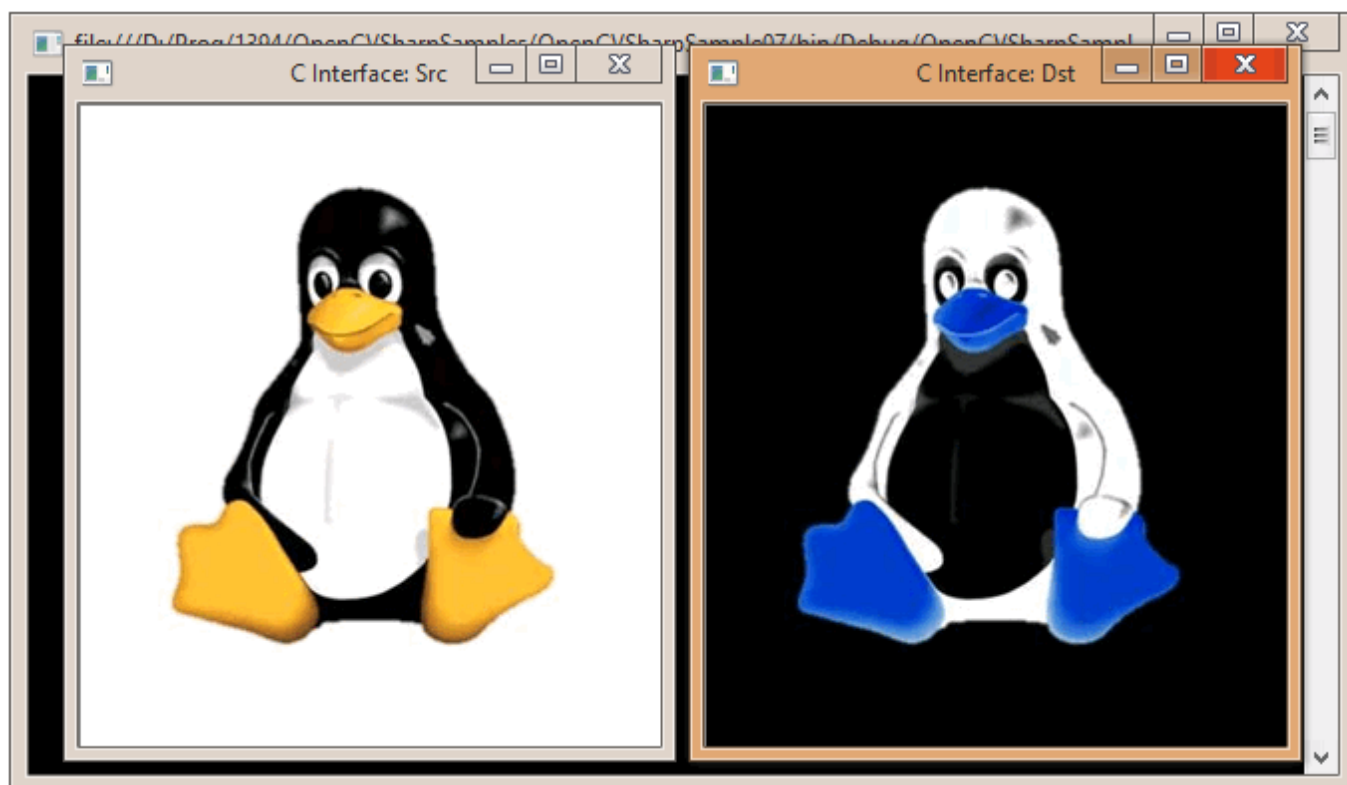
```
using (var src = new IplImage(@"..\..\Images\Penguin.Png", LoadMode.AnyDepth | LoadMode.AnyColor))
using (var dst = new IplImage(src.Size, src.Depth, src.NChannels))
{
    for (var y = 0; y < src.Height; y++)
    {
        for (var x = 0; x < src.Width; x++)
        {
            CvColor pixel =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



شبهه 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 اینترنت‌فیس ++C، [در Wiki](#) این کتابخانه مطالعه نمائید.

کدهای کامل این مثال را [از اینجا](#) می‌توانید دریافت کنید.

بررسی morphology (ریخت شناسی) تصاویر

به تصویر زیر دقت کنید:



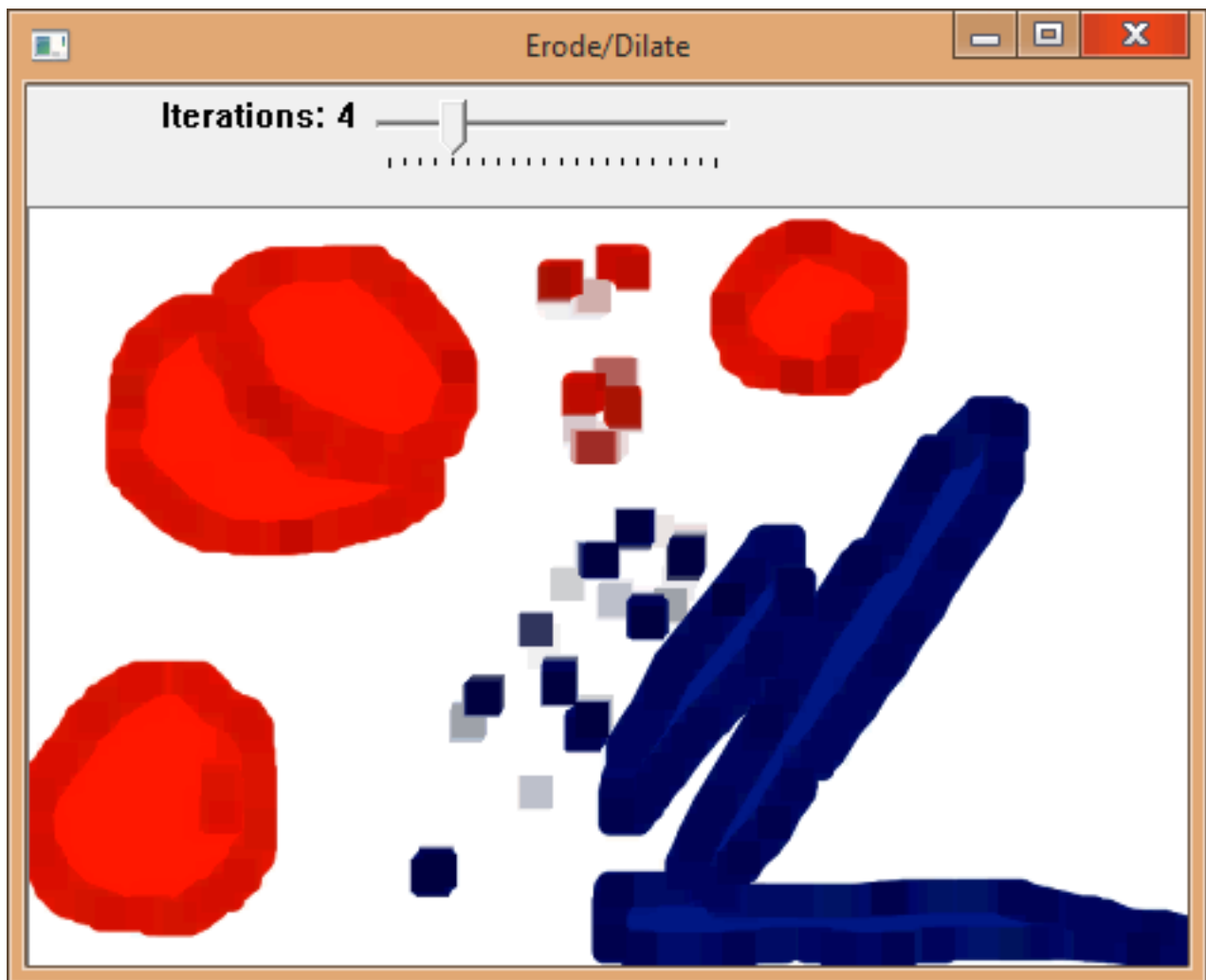
فرض کنید در اینجا قصد دارید تعداد توپ‌های قرمز را شمارش کنید. از دیدگاه یک انسان، شاید سه توپ قرمز قابل مشاهده باشد. اما از دیدگاه یک برنامه، توپ وسطی به دو توپ تفسیر خواهد شد و همچنین نویزهای قرمزی که بین توپ‌ها در صفحه وجود دارند نیز شمارش می‌شوند. بنابراین بهتر است پیش از پردازش این تصویر، ریخت شناسی آن را 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](#)

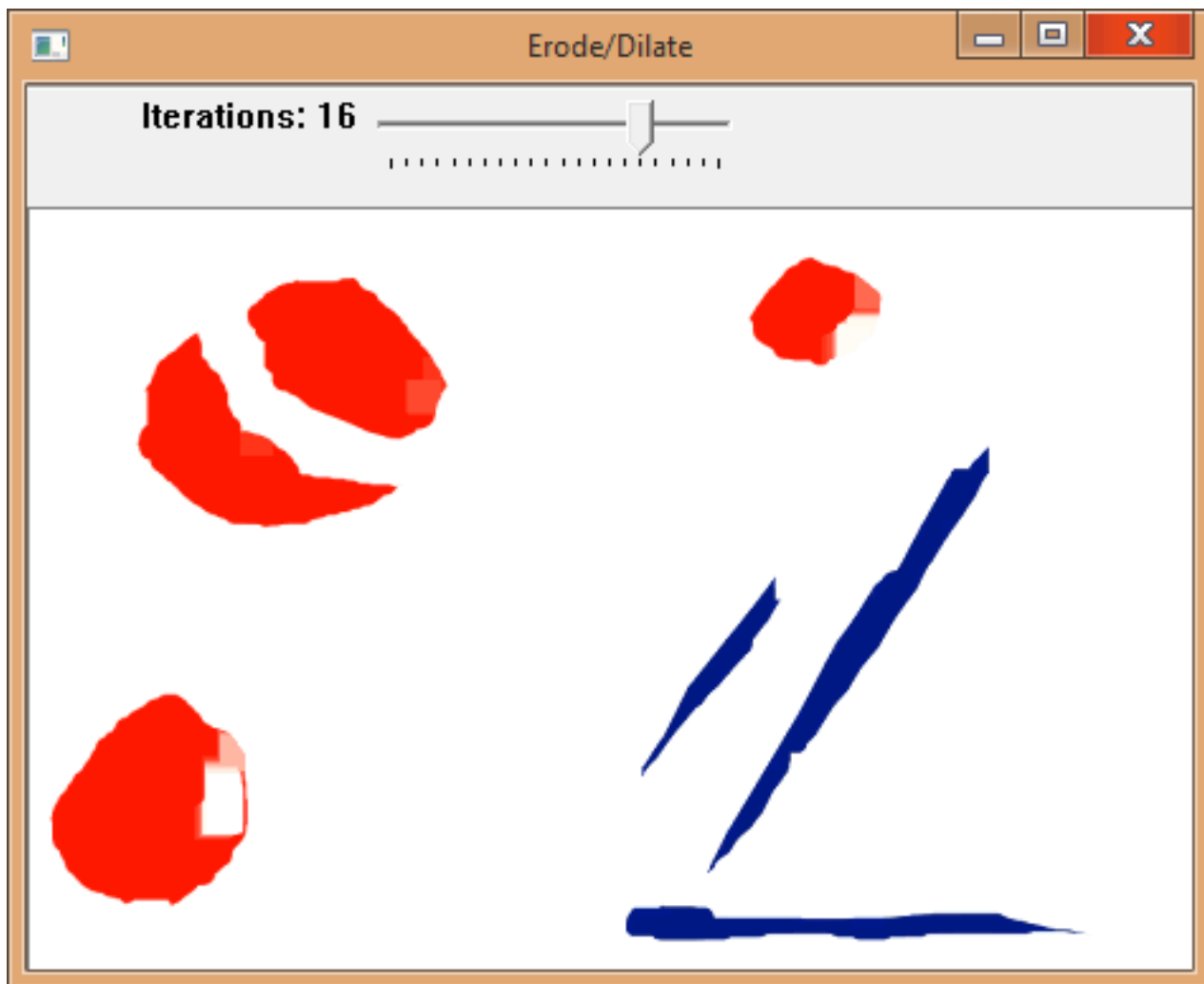
تحلیل/فرسایش یا erosion سبب می‌شود تا نواحی تیره‌ی تصویر «رشد» کنند.



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

[Cv2.Dilate \(2\)](#)

اتساع یا dilation سبب خواهد شد تا نواحی روشن تصویر «رشد» کنند.

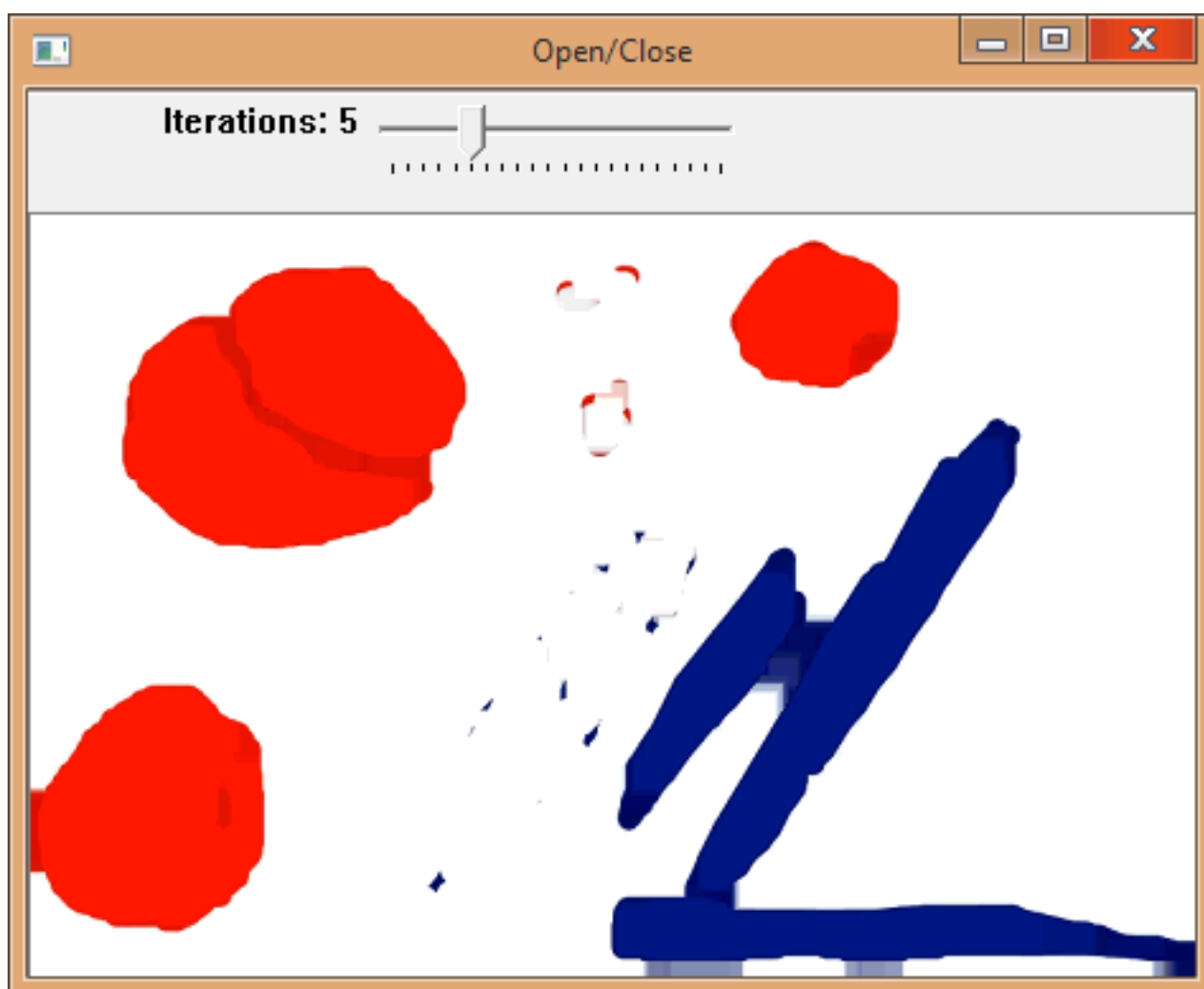


بکارگیری فیلتر Dilate سبب شده است تا نویزهای تصویر محو شوند و اشیاء به هم پیوسته از هم جدا گردند.

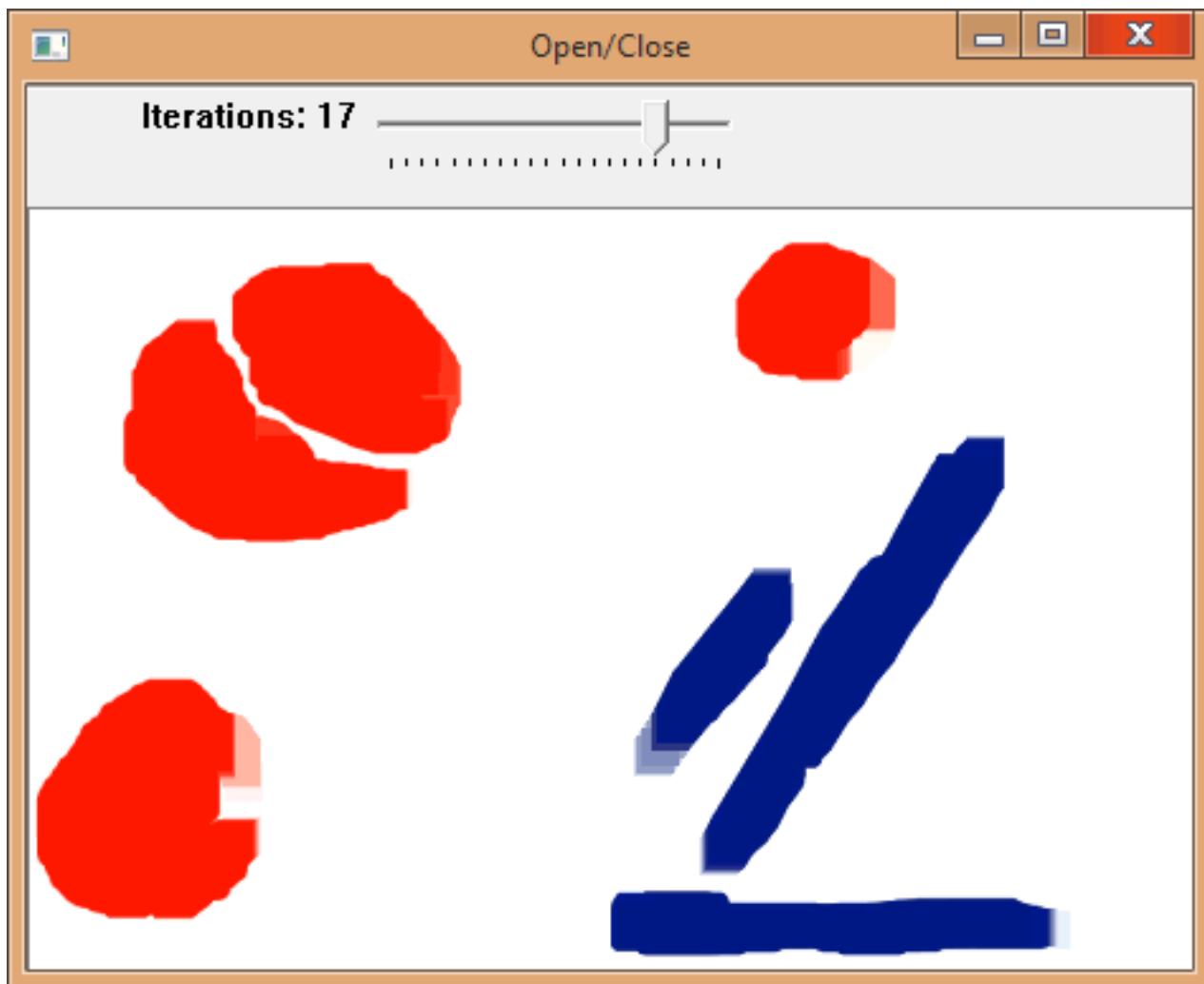
[Cv2.MorphologyEx](#) (3)

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، مشخص‌کننده‌ی میزان تحلیل و یا اتساع نهایی هستند.

Structuring element shapes

Rect



Cross



Ellipse



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 API C++](#) 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، چندسکوپی است. بدیهی است اگر قرار بود چنین کاری را صرفاً با یک برنامه‌ی دات نتی انجام داد، می‌شد قسمت ایجاد پنجره و tracker آن را 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.

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();
                }
            }
        }

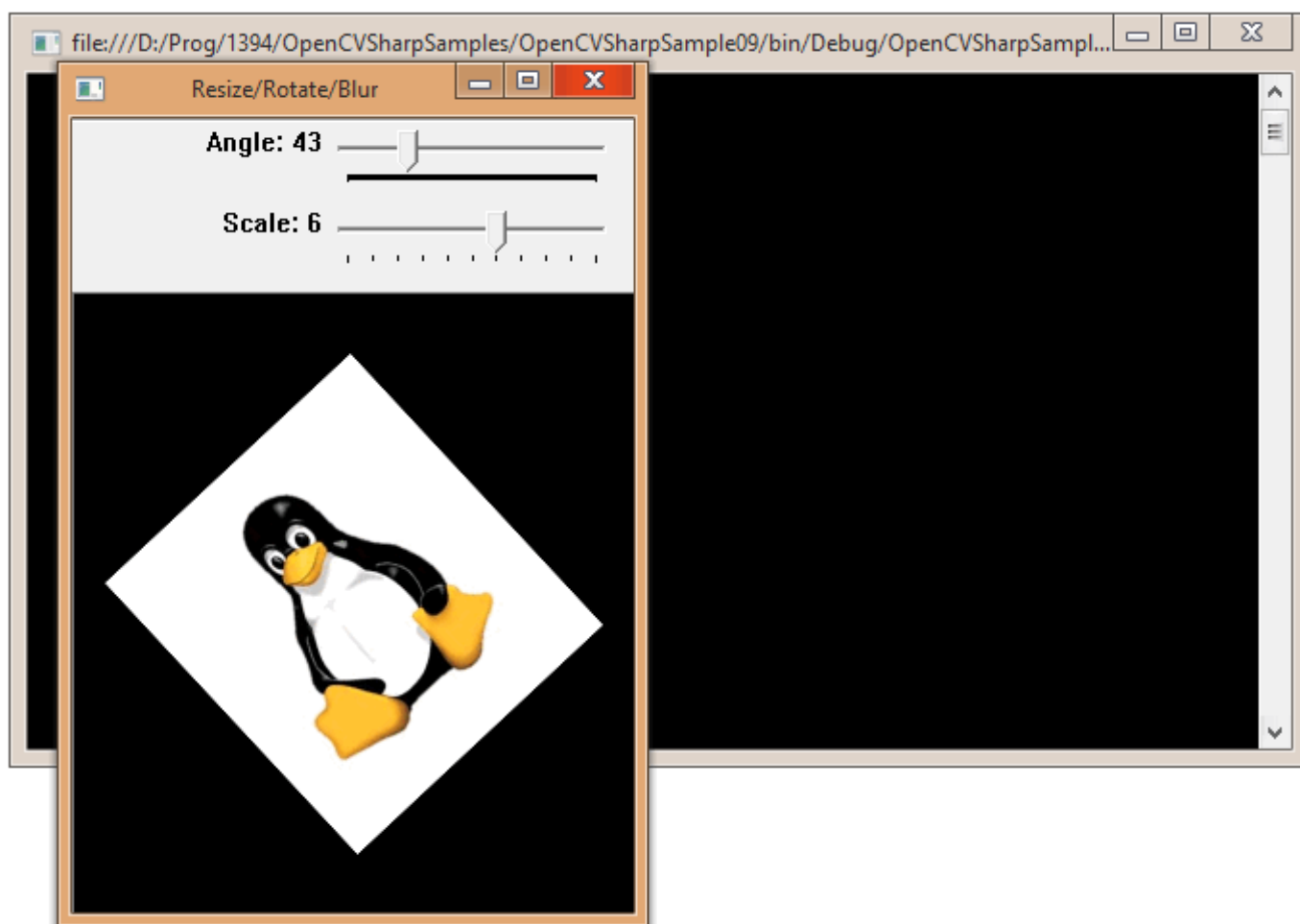
        private static void rotateImage(double angle, double scale, Mat src, Mat dst)
```

```

    {
        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 API C++ 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 صرفاً 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,

```



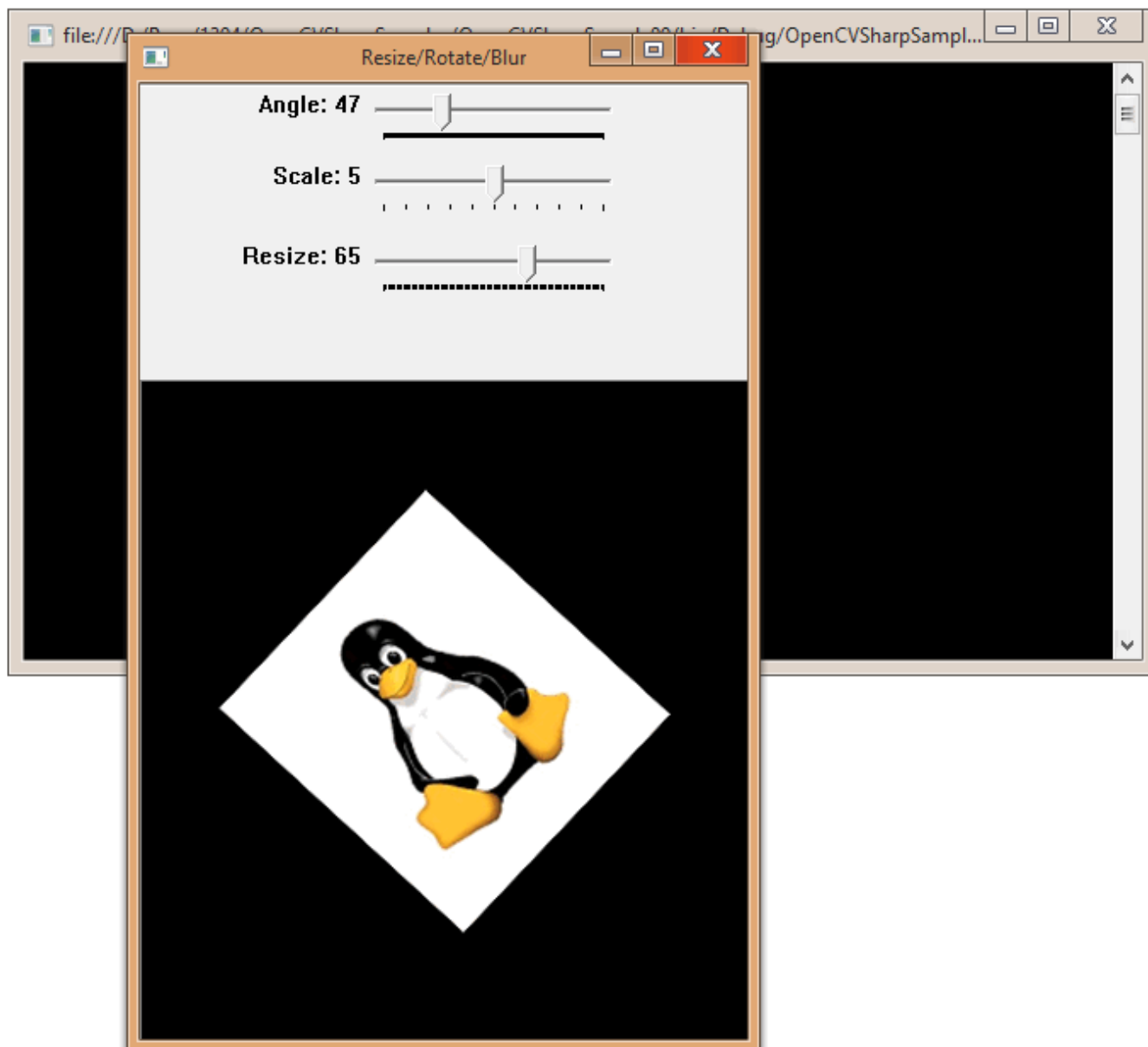
```
callback: pos =>
{
    Cv2.Resize(src, dst,
        new Size(src.Width + pos, src.Height + pos),
        interpolation: Interpolation.Cubic);
    window.Image = dst;
});
```

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

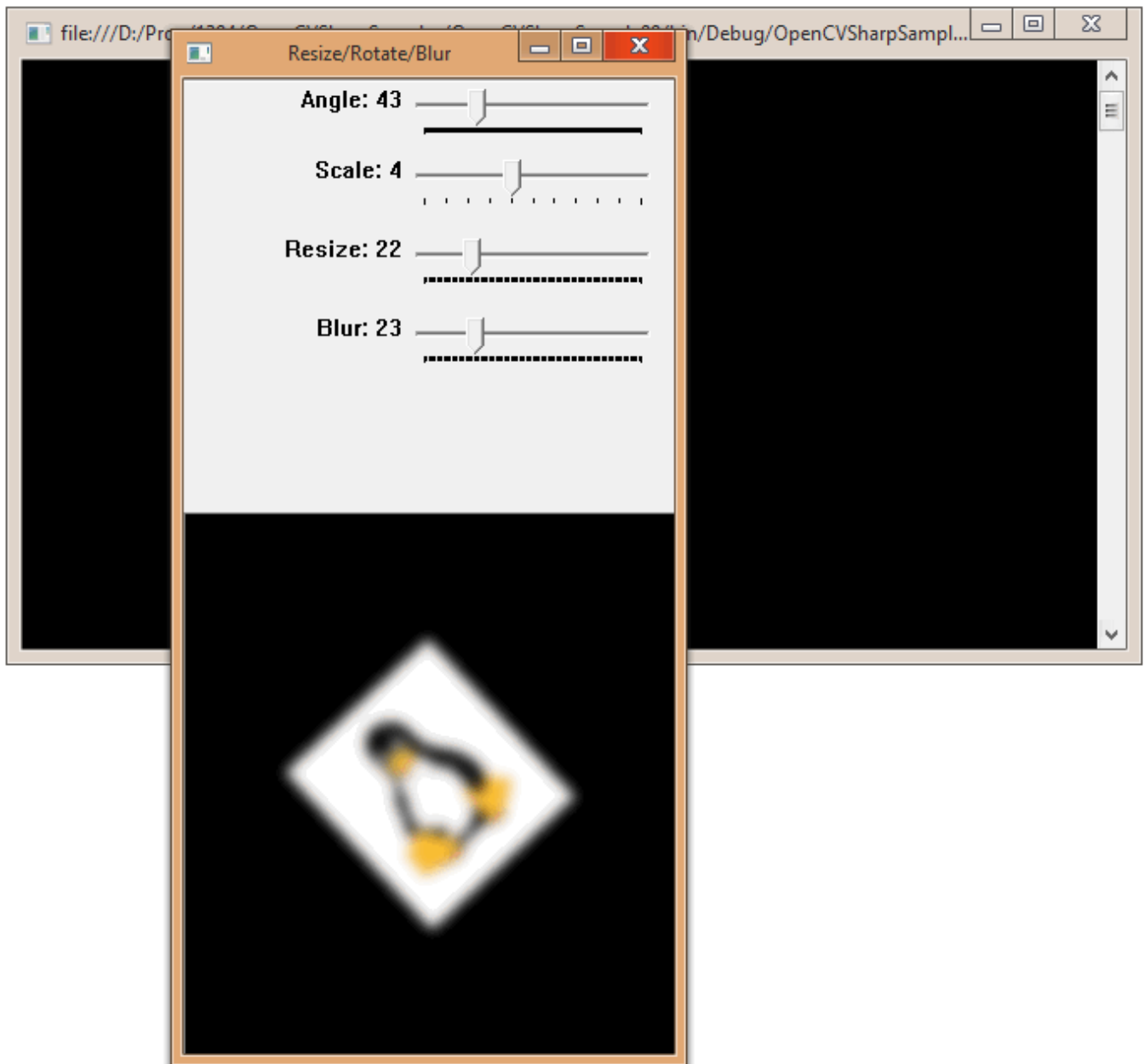
در OpenCV با استفاده از متدهای [GaussianBlur](#) و یا [medianBlur](#) ، می‌توان تصاویر را مات کرد که نمونه‌ای از آن‌را در ادامه ملاحظه می‌کنید:

```
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.

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::cvtColor](#) که قادر است هیستوگرام تعدادی آرایه را محاسبه کند و در API C++ 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 آن را 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 brightnessTracker = sourceWindow.CreateTracker(
                name: "Brightness", value: brightness, max: 200,
                callback: pos =>
                {
                    brightness = pos;
                    updateImageCalculateHistogram(sourceWindow, histogramWindow, src, brightness,
contrast);
                });
        }
    }
}
```

```

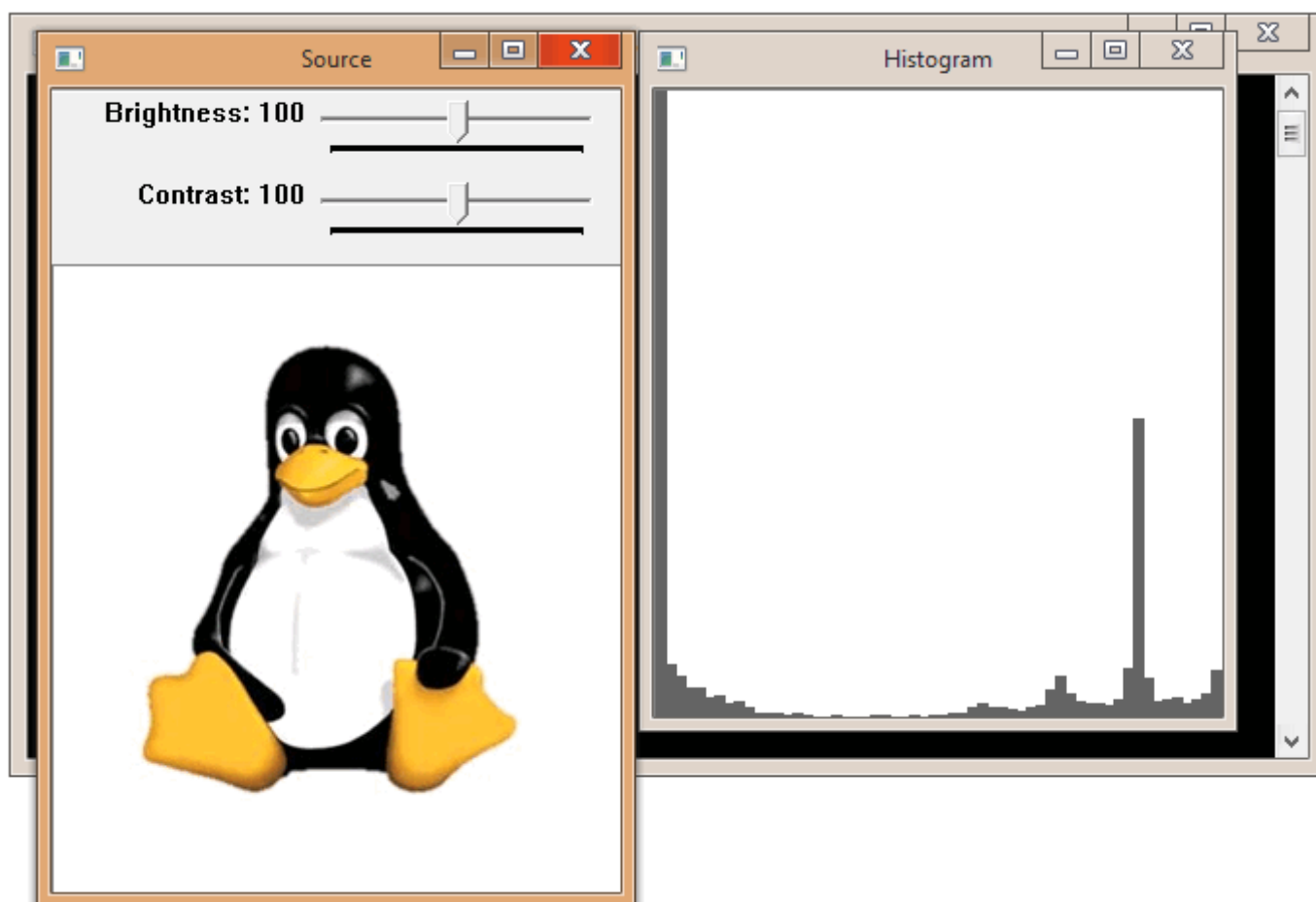
var contrastTrackbar = sourceWindow.CreateTrackbar(
    name: "Contrast", value: contrast, max: 200,
    callback: pos =>
    {
        contrast = pos;
        updateImageCalculateHistogram(sourceWindow, histogramWindow, src, brightness,
contrast);
    });

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.

```
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);

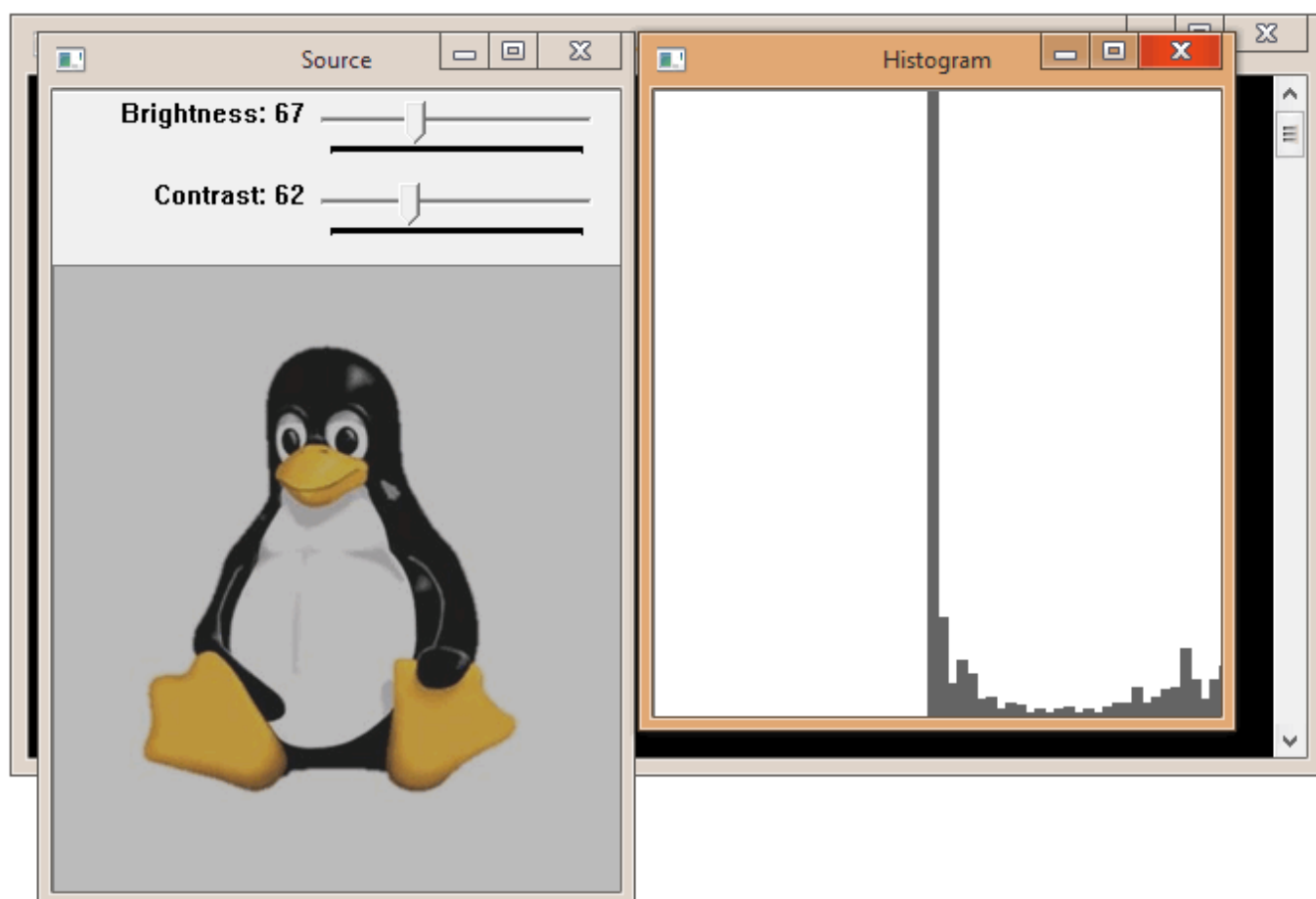
            var color = Scalar.All(100);

            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 اندازه‌گیری کنیم. پارامتر `ranges` مشخص می‌کند که مقادیر اندازه‌گیری شده باید در چه بازه‌ایی جمع‌آوری 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 `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.

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 عموماً 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
```



```

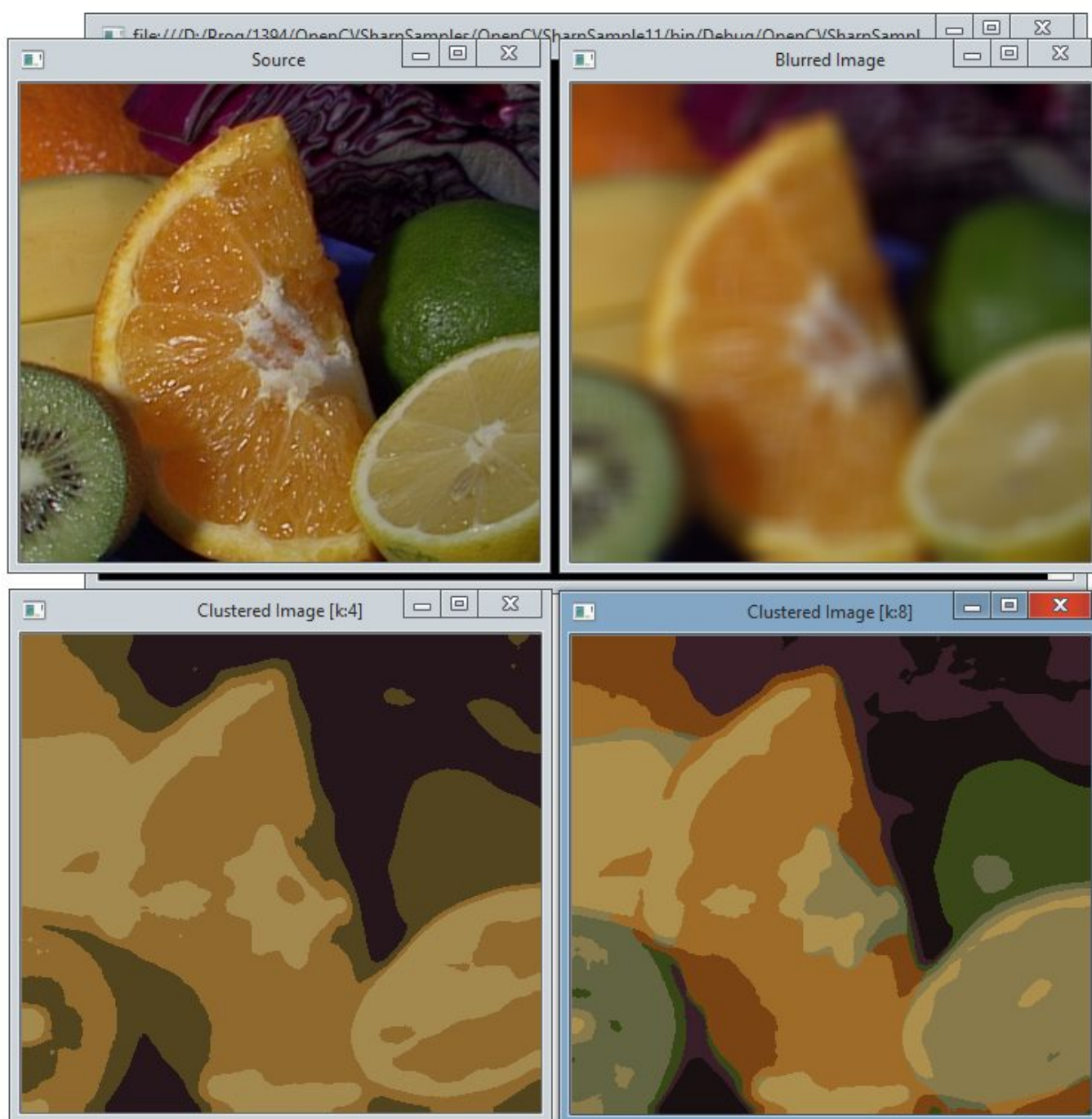
    {
        Item0 = (byte)(centers.At<float>(clusterIndex, 0)), // B
        Item1 = (byte)(centers.At<float>(clusterIndex, 1)), // G
        Item2 = (byte)(centers.At<float>(clusterIndex, 2)) // R
    };
    clusteredImage.Set(size / src.Cols, size % src.Cols, newPixel);
}

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 صرفاً 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 $M \times N$ بردار، به k قسمت تقسیم شوند.

متد `Reshape` تصویر اصلی $M \times N \times 3$ را به یک ماتریس $K \times 3$ تبدیل می‌کند که در آن $K = M \times N$ 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.

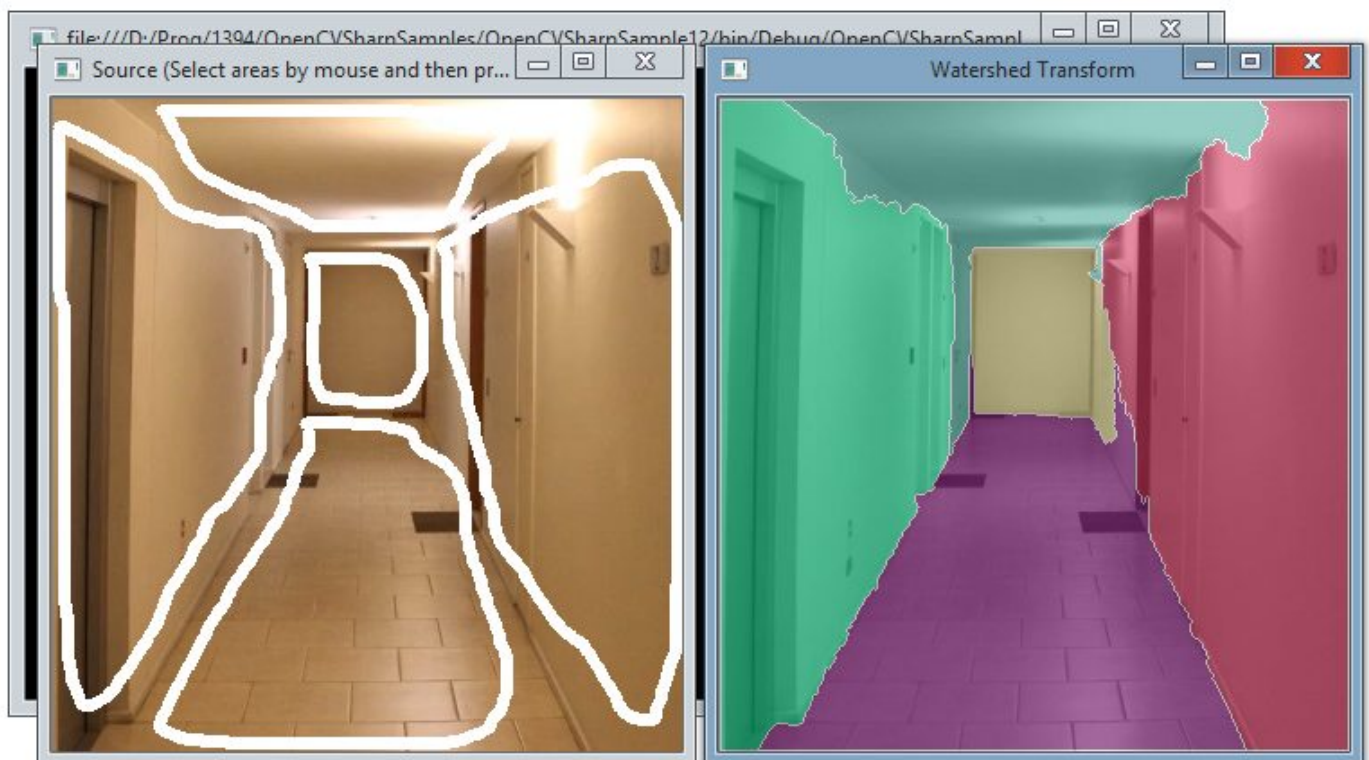
پارامتر `attempts` مشخص می‌کند که این الگوریتم چندبار باید اجرا شود تا بهترین میزان فشردگی و کاهش رنگ حاصل شود.

- پس از پایان عملیات `k-means` نیاز است تا اطلاعات آن مجدداً 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.

قطعه بندی (segmentation) تصویر با استفاده از الگوریتم watershed

در تصویر ذیل، تصویر یک راهرو را مشاهده می‌کنید که توسط ماوس قطعه بندی شده است (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 شده است. 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 شده است. 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 شده است. 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;
HierarchyIndex[] 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 کرده است، دریافت می‌کند. سپس کانتورهای آن را استخراج خواهد کرد. کانتورها [در مثال‌های اصلی OpenCV](#) با vector مشخص شده‌اند. در اینجا (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);

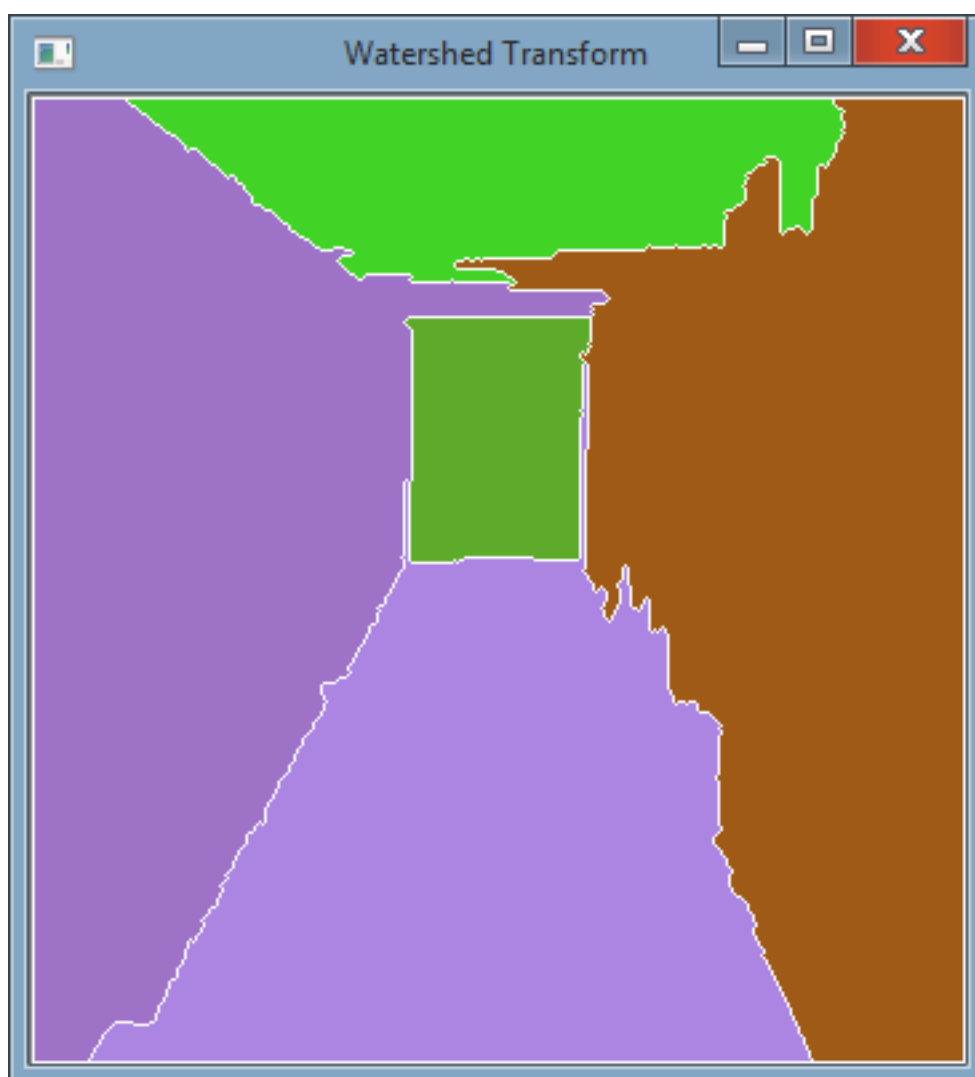
var watershedImage = new Mat(markers.Size(), MatType.CV_8UC3);

// 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

```

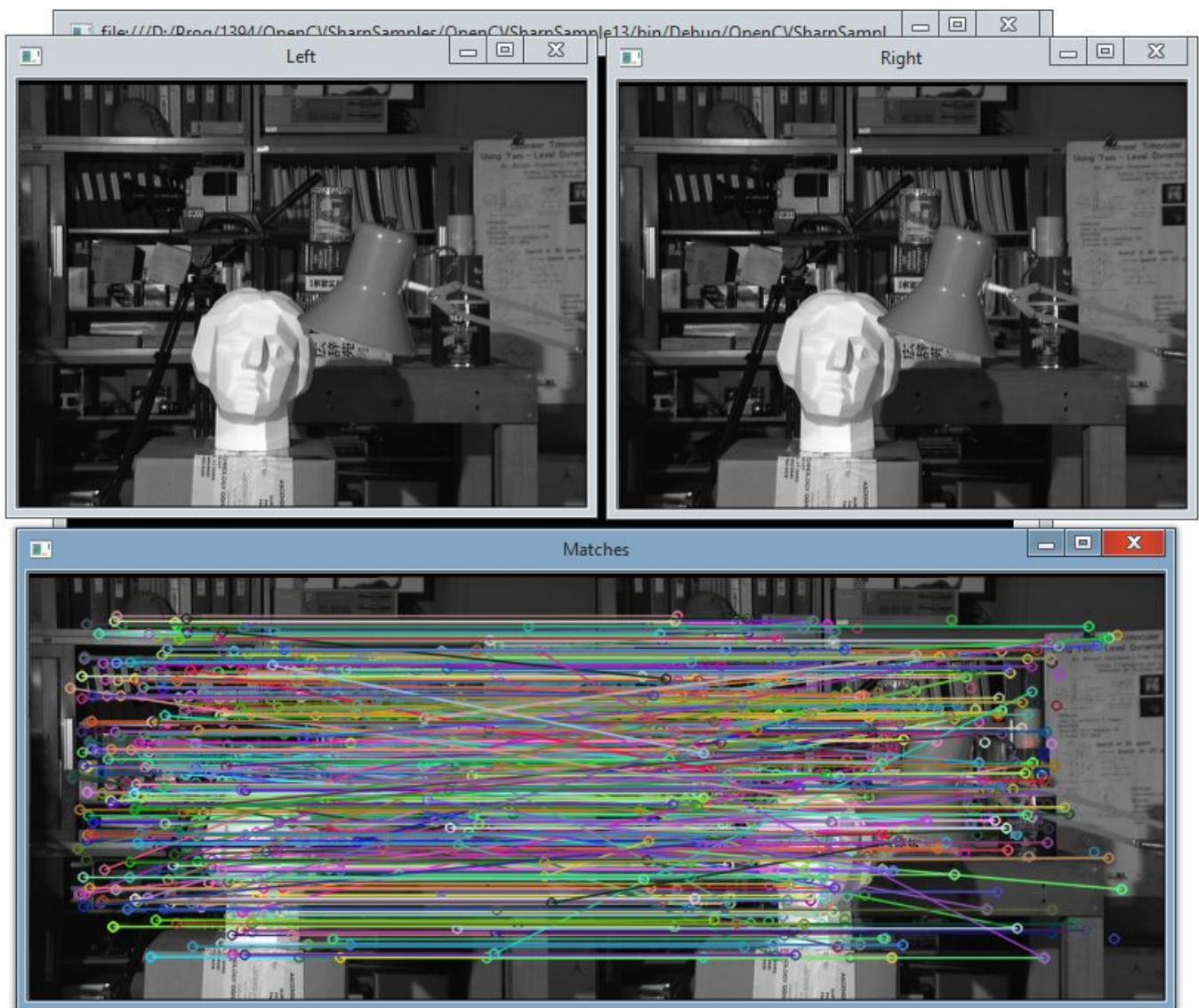
متد `Cv2.TheRNG` یک تولید کننده‌ی اعداد تصادفی توسط `OpenCV` است و متد `Uniform` آن شبیه به متد `Next` کلاس `Random` دات نت عمل می‌کند. به نظر این کلاس تولید اعداد تصادفی، آنچنان هم تصادفی عمل نمی‌کند. به همین جهت از کلاس `Random` دات نت استفاده شد. در اینجا به ازای تعداد کانتورهای ترسیم شده، یک رنگ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

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 استفاده هستند. برای مثال در اینجا از الگوریتم SURF آن استفاده شده است. 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 این کار را انجام می‌شود.

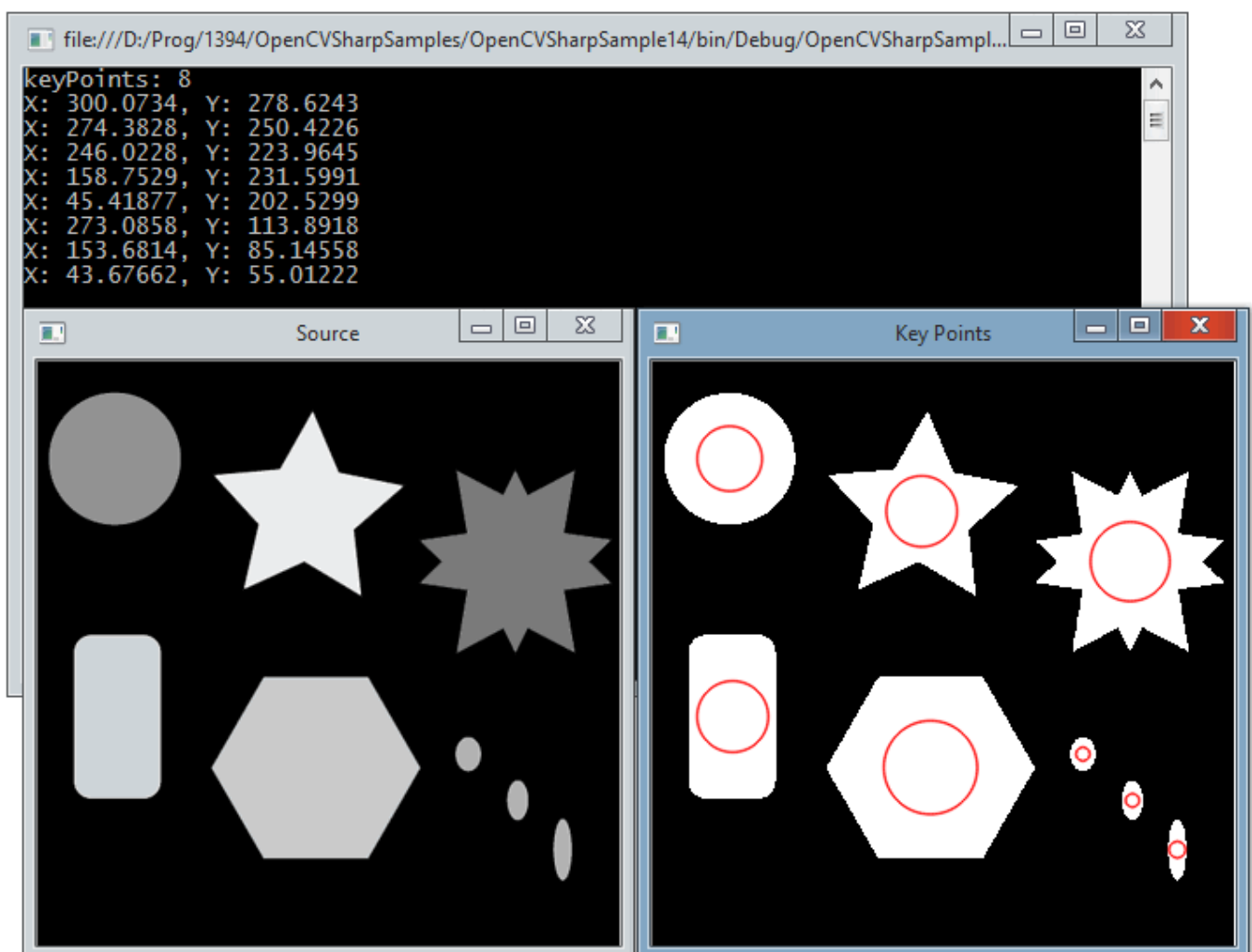
اگر علاقمند هستید که با ریز جزئیات ریاضی الگوریتم‌های استفاده شده نیز آشنا شوید، سری مطالب [descriptors](#) را دنبال نمائید.

کدهای کامل این مثال را [از اینجا](#) می‌توانید دریافت کنید.

تشخیص BLOBs در تصویر به کمک OpenCV

BLOB یا B inary L arge OB ject به معنای گروهی از نقاط به هم پیوسته‌ای 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، نمونه‌ای باینری سیاه و سفید آن. سپس عملیات تشخیص Blobs بر روی تصویر اصلی انجام شده و نتیجه‌ی نهایی که مشخص‌کننده‌ی 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\cv1b1.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.FromRgba(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

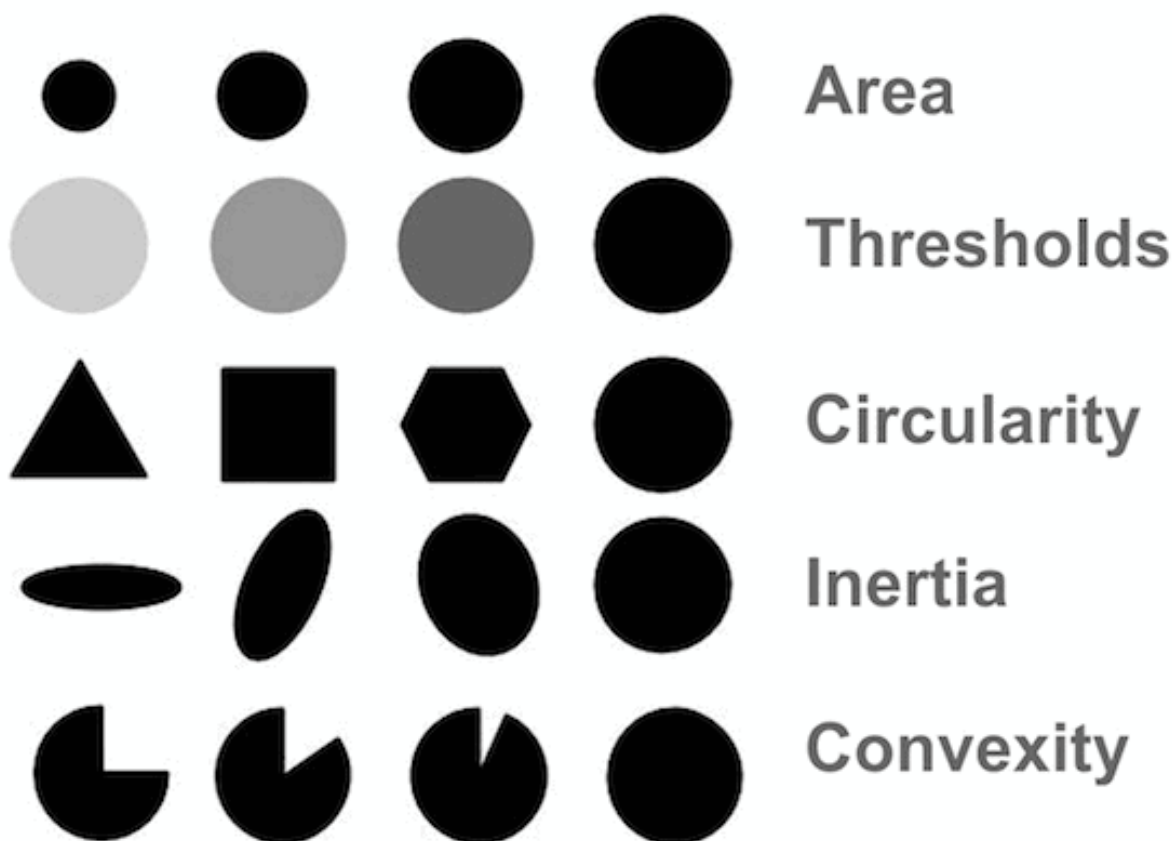
در اینجا ابتدا تصویر منبع به صورتی متداول باگذاری شده است. سپس توسط متد `CvtColor` به نمونه‌ای سیاه و سفید و به کمک متد `Threshold` به یک تصویر باینری تبدیل شده است. اگر به تصویر ابتدای بحث دقت کنید، اشیاء موجود در منبع مفروض، رنگ‌های متفاوتی دارند؛ اما در تصویر نهایی تولید شده، تمام

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.

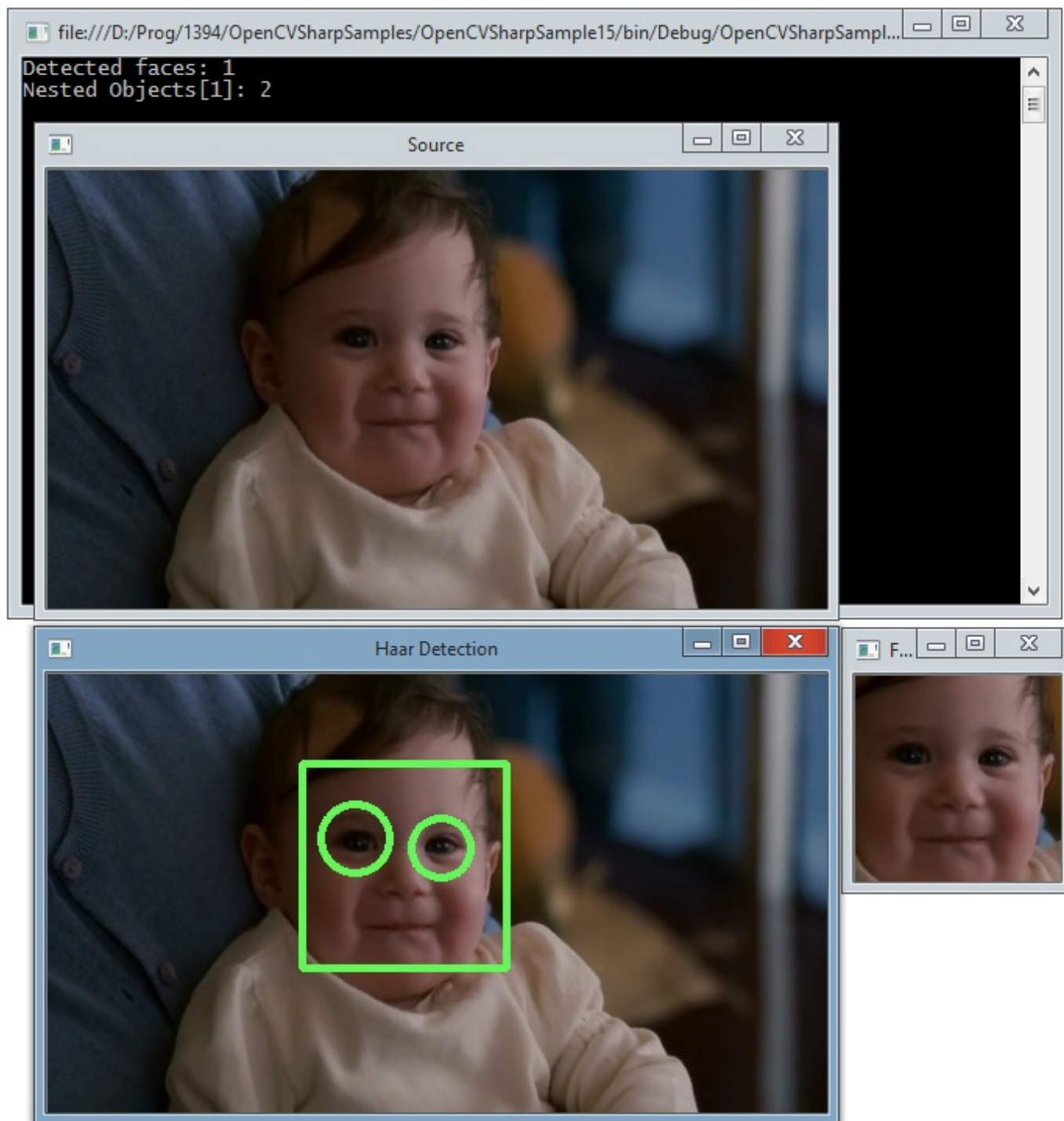
در ادامه متد `simpleBlobDetector.Detect` بر روی تصویر باینری بهبود داده شده، فراخوانی گشته‌است. خروجی آن نقاط کلیدی اشیاء یافت شده‌است. این نقاط را می‌توان توسط متد `DrawKeypoints` ترسیم کرد که حاصل آن دواپر قرمز رنگ موجود در مرکز اشیاء یافت شده‌ی در تصویر سمت راست هستند.

کدهای کامل این مثال را [از اینجا](#) می‌توانید دریافت کنید.

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 دو فایل جهت سهولت کار، به همراه مثال این بحث نیز ارائه شده‌اند).

فایل `haarcascade_frontalface_alt.xml` اصطلاحاً 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 شده است و سپس متد `Cv2.EqualizeHist` بر روی آن فراخوانی گشته است. این متد وضوح تصویر را جهت یافتن الگوها بهبود می بخشد.

سپس فایل `xml` یاد شده ی در ابتدای بحث را توسط کلاس `CascadeClassifier` بارگذاری می کنیم:

```
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 فایل های XML اطلاعات نحوه ی تشخیص چهره و اعضای آن، با فراخوانی متد `DetectMultiScale`، کار تشخیص چهره و استخراج آن از `grayImage` انجام خواهد شد. در اینجا `minSize`، اندازه ی حداقل چهره ی مدنظر است که قرار هست تشخیص داده شود. نواحی کوچکتر از این اندازه، به عنوان نویز در نظر گرفته خواهند شد و پردازش نمی شوند. خروجی این متد، مستطیل ها و نواحی یافت شده ی مرتبط با چهره های موجود در تصویر هستند. اکنون می توان حلقه ای را تشکیل داد و این نواحی را برای مثال با مستطیل های 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 ناحیه ی تشخیص داده شده، نحوه ی استخراج تصویر آن ناحیه را هم در سطر `var detectedFaceImage` مشاهده می کنید.

همچنین اگر علاقمند باشیم تا در این ناحیه ی یافت شده، چشمان شخص را نیز استخراج کنیم، می توان 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 آن را 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.