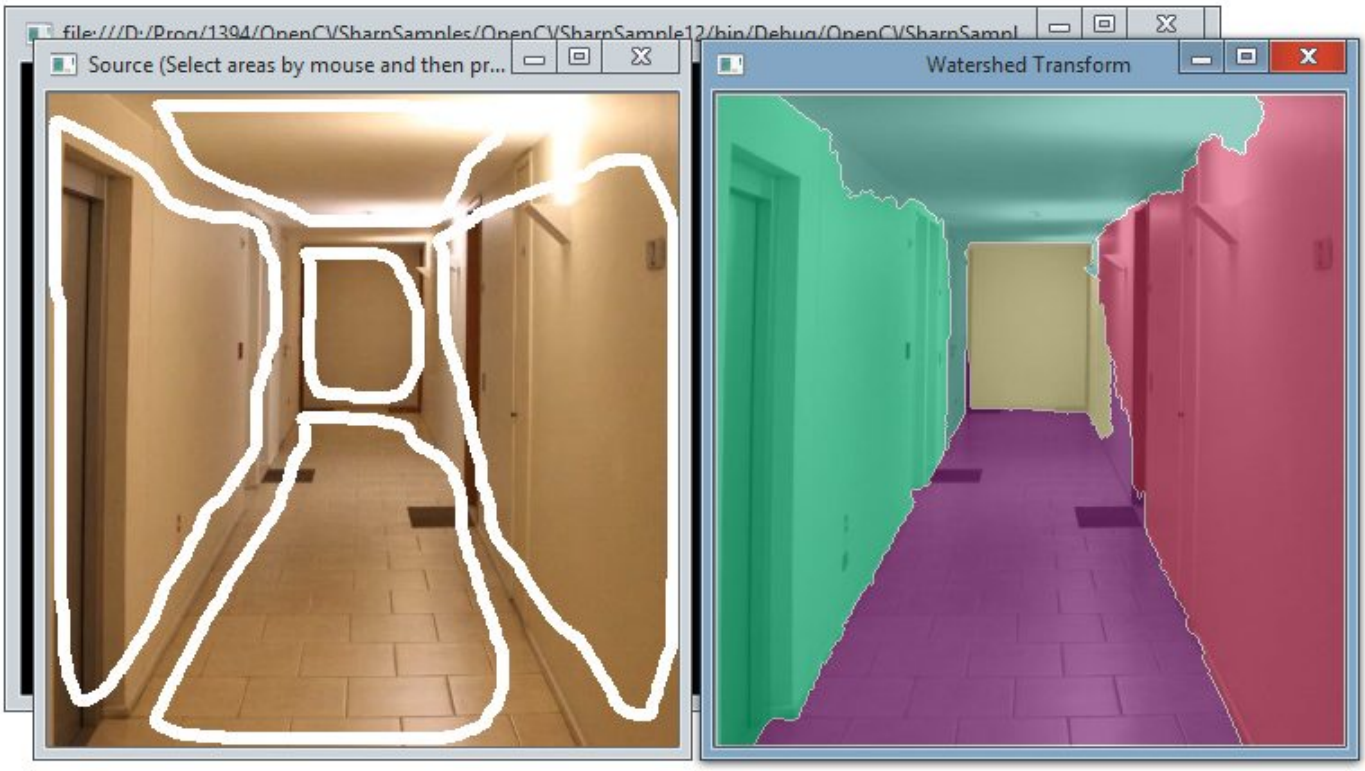
عنوان: OpenCVSharp #12
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۳/۲۵ ۱۱:۵۰
آدرس: www.dotnettips.info
گروه‌ها: OpenCV

### 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.

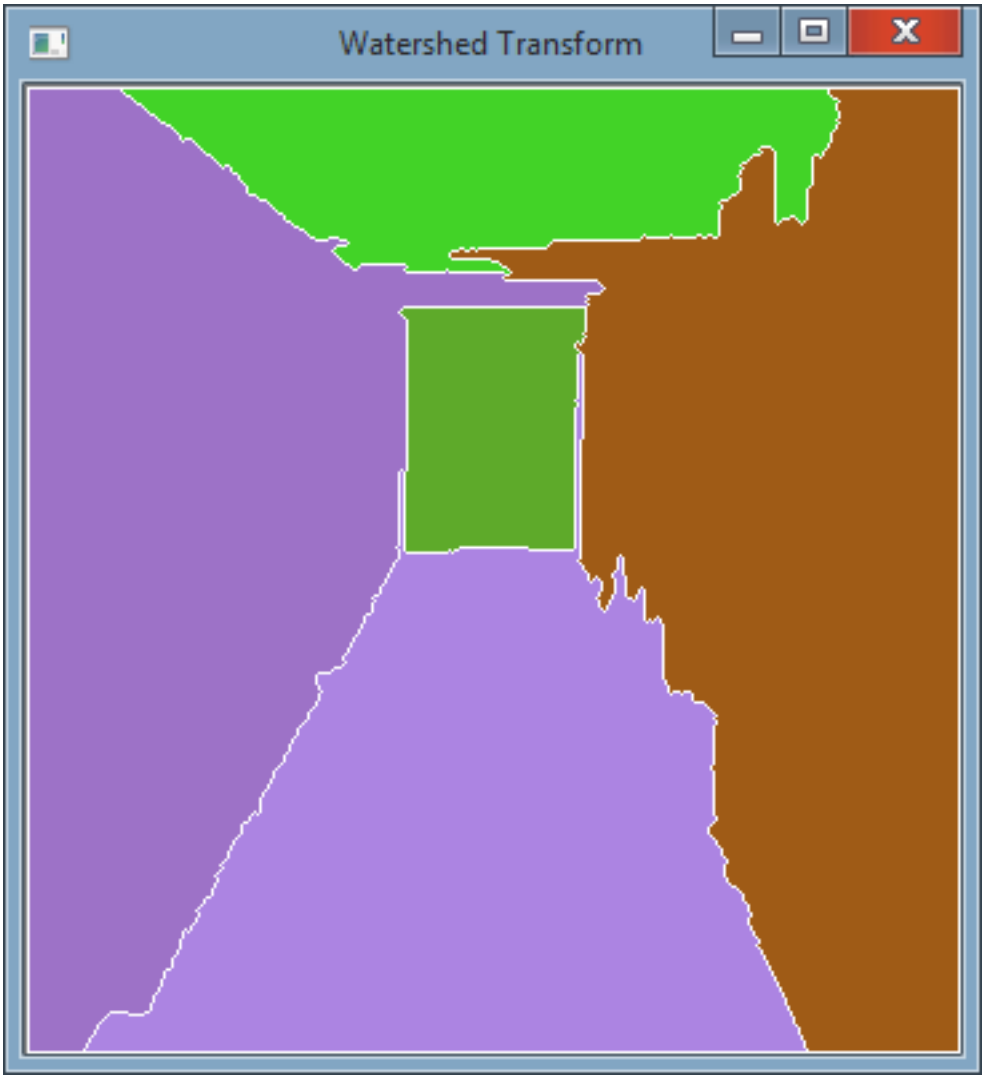
```
var rnd = new Random();
var colorTable = new List<Vec3b>();
for (var i = 0; i < componentCount; i++)
{
```

```
    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.**