# 【opencv实践】仿射变换和透视变换

原创 周旋 周旋机器视觉 2020年03月23日 15:12

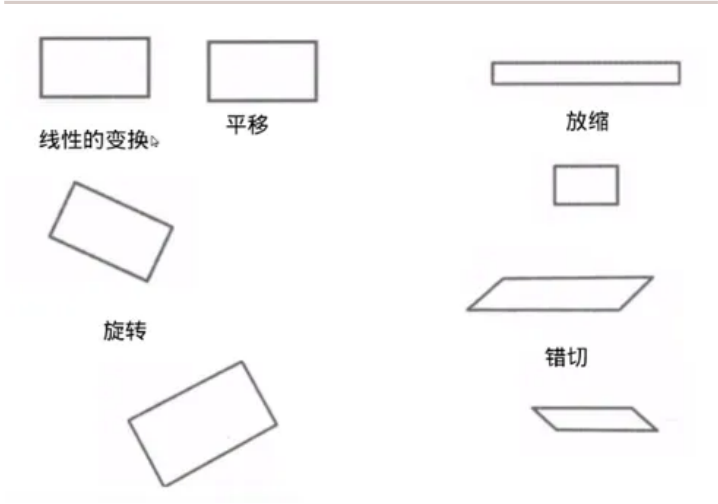
**点击上方"蓝色小字"关注我哦~**

上面这副图就是我们今天要处理的了，我们想把它从拍照视角变成鸟瞰图，这是机器人导航中的常用手段，以便在该平面上进行规划和导航。

这种变换常常用到透视变换，但我们今天在讲解透视变换时，需要普及一下其他的变换，包括**平移**，**旋转**，**错切**，**放缩**，以及**仿射变换**。

## 综述

所有复杂的东西，都是由基本的组成的。所以我们需要先了解一下基础的变换有哪些：




## 平移

我们对矩形（图像）平移，需要怎么做？

对每一个像素点坐标平移。可以让每一个像素点的x,y坐标都加一个变量。

矩阵形式表示：

$$\begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} X + T_x \\ Y + T_y \\ 1 \end{bmatrix}$$

等式左边[X,Y,1]是像素坐标的齐次形式。等式右边是平移之后的坐标。

## 放缩

进行放缩，就是将矩形（图像）放缩n倍，也就是长宽各乘一个变量。

$$\begin{bmatrix} T_x & 0 & 0 \\ 0 & T_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} T_x X \\ T_y Y \\ 1 \end{bmatrix}$$

## 旋转

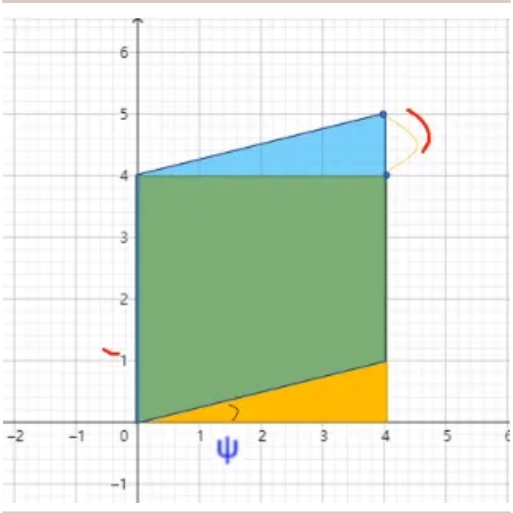
对矩形（图片）进行旋转，关于旋转的数学推导在后面仿射会介绍：

$$\begin{bmatrix} \cos\Phi & -\sin\Phi & 0 \\ \sin\Phi & \cos\Phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_0 \\ Y_0 \\ 1 \end{bmatrix} = \begin{bmatrix} X_1 \\ Y_1 \\ 1 \end{bmatrix}$$

## 错切

前面的都比较直观，那错切是什么呢？

我们可以看下矩形关于y方向的错切：




看图就很直观了，那数学表达呢？

$$\begin{bmatrix} 1 & 0 & 0 \\ \tan\psi & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y + \tan\psi X \\ 1 \end{bmatrix}$$

x轴上的错切就是同理了，公式如下：

$$\begin{bmatrix} 1 & \tan\phi & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} X + \tan\phi Y \\ Y \\ 1 \end{bmatrix}$$

然后两者和起来，就如下了：

$$\begin{bmatrix} 1 & \tan\phi & 0 \\ \tan\psi & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} X + \tan\phi Y \\ Y + \tan\psi X \\ 1 \end{bmatrix}$$

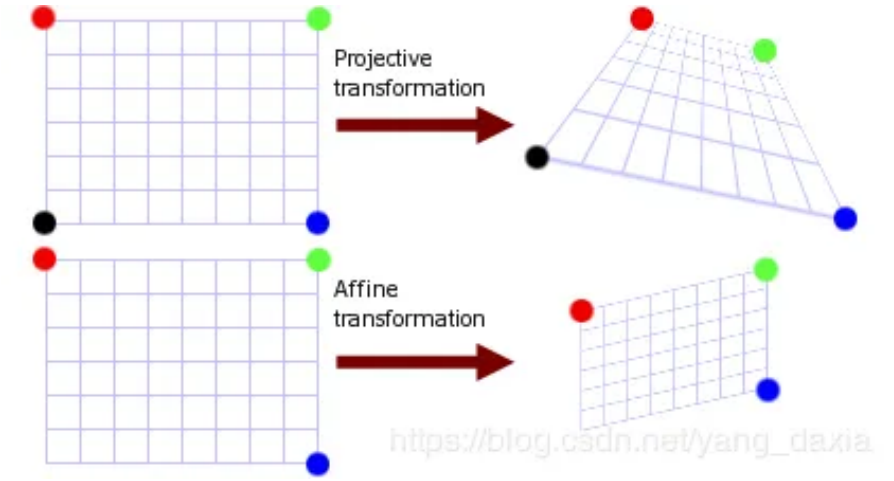
好了，到此我们就了解了这四种变换了，那仿射变换是什么呢？可以看下图公式：

$$\underbrace{\begin{bmatrix} 1 & 0 & -x \\ 0 & 1 & -y \\ 0 & 0 & 1 \end{bmatrix}}_{\text{平移 Translate}} \underbrace{\begin{bmatrix} cos\Theta & -sin\Theta & 0 \\ sin\Theta & cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{旋转 Rotate}} \underbrace{\begin{bmatrix} W & 0 & 0 \\ 0 & H & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{放缩}} \underbrace{\begin{bmatrix} 1 & \tan\phi & 0 \\ \tan\psi & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{错切 Shear}} \underbrace{\begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix}}_{\text{平移}} = \begin{bmatrix} a_0 & a_1 & b_1 \\ a_2 & a_3 & b_2 \\ 0 & 0 & 1 \end{bmatrix}$$

**等式右边就是仿射变换矩阵，是由原图像平移，旋转，放缩，错切之后得来的。**

在书上往往将仿射变换和透视变换放一起讲，这两者各是什么呢？

在刚学仿射变换和透视变换时，我是有些分不清的。印象最深刻的就是下图：



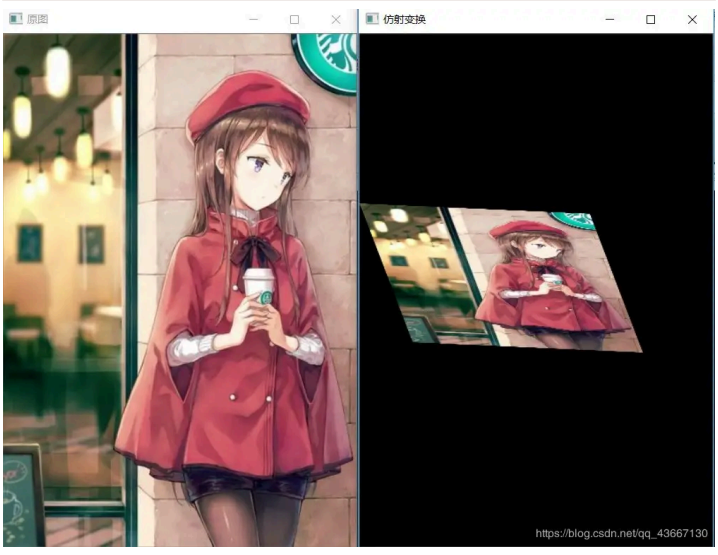

可以看到，仿射变换（下）是将矩形变换成**平行四边形**（即变换后各边依旧平行），而透视变换（上）可以变换成**任意不规则四边形**。

这样看来，好像仿射变换是透视变换的子集。

那到底是不是呢？其实是的。仿射变换属于线性变换，而透视变换则不仅仅是线性变换。仿射变换可以看做是透视变换的一种特例。

直观上感受，我们可以认为：

仿射变换是单纯对图片进行缩放，倾斜和旋转，因此图片不论如何变化，线之间的平行性是不变的。如下图。

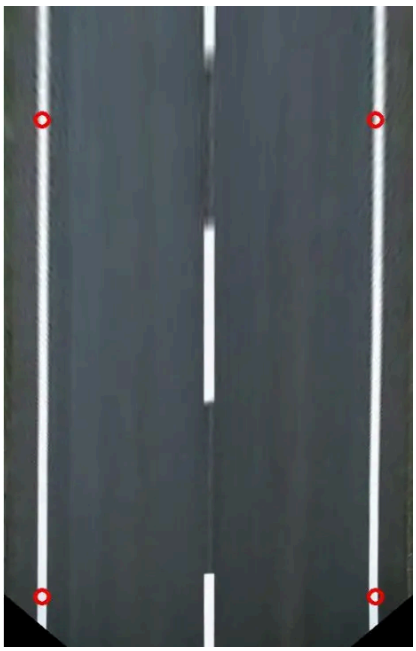



可以感受到，右图是可以通过左图平移，旋转，错切，缩放之后得来。

而透视变换，则是当观察者的视角发生变化时物体发生的透视变换，此转换允许造成透视形变。

我们看下图的公路，近处宽远处窄，就是因为视角的原因，

而我们本文要做的，就是将视角改为鸟瞰，从而得到类似下图的鸟瞰图：



## 仿射变换原理

前文已经说了，仿射变换是单纯对图片进行平移，缩放，倾斜和旋转，而这几个操作都不会改变图片线之间的平行关系。

opencv中给出了仿射变换的函数接口：

```
warpAffine(
   InputArray  src,     输入图像
   OutputArray  dst,     输出图像
   InputArray  M,       仿射计算矩阵
   Size    dsize,     输出图像大小
   int      flags = INIET_LINEAR,    插值方法
   int      borderMode = BORDER_CONSTANT,
   const Scalar&  borderValue = Scalar()
   );
```

这个函数很好理解，输入一个图像，输出这个图像的仿射变换。

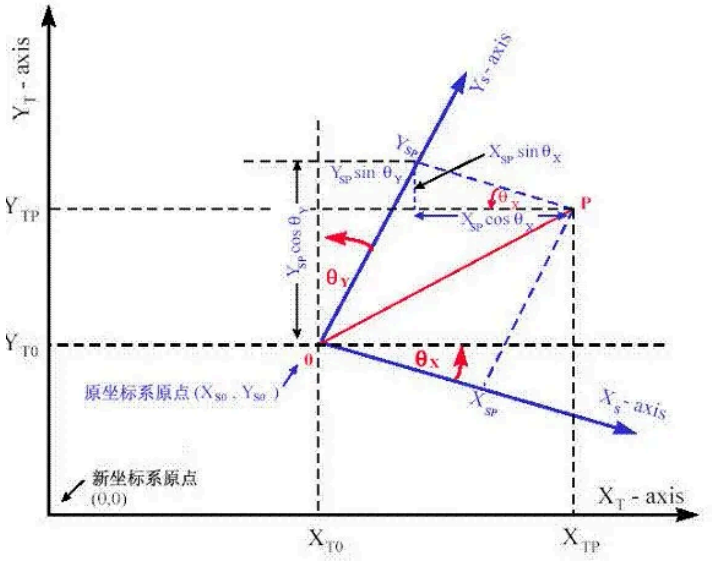
但第三个参数需要我们输入**2*3的仿射计算矩阵**，这是什么鬼？

我们先看一下仿射计算矩阵长什么样子（可以去掉最后一行）：

$$\begin{bmatrix} cos\theta & sin\theta & tx \\ -sin\theta & cos\theta & ty \\ 0 & 0 & 1 \end{bmatrix}$$

我们的输出图像G(x,y) = F(x,y)乘仿射矩阵。

我们可以看下图推导出仿射计算矩阵。




一个点P在原始坐标系下的坐标是(Xsp,Ysp)。然后要完成旋转操作，旋转操作是基于原点的。如何得到旋转之后的点的坐标，这里用到一个技巧：

**坐标系中某个点的旋转可以等价地去旋转坐标轴。**

所以有了上图中以(Xs0,Ys0)为中心的虚线与屏幕水平垂直的坐标系。在这个坐标系中确定P的坐标，和在蓝色坐标系中确定旋转之后P的坐标是等价的。

基于这个结论，我们可以通过简单的立体几何知识确定P在新坐标系中的坐标。P在新坐标系中的X坐标和Y坐标分别是

$$Y_{sp}sin\theta_y + X_{sp}cos\theta_x \qquad Y_{sp}cos\theta_y - X_{sp}sin\theta_x$$

进而我们可以得到：

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta_x & sin\theta_y \\ -sin\theta_x & cos\theta_y \end{bmatrix} \begin{bmatrix} X_{sp} \\ Y_{sp} \end{bmatrix}$$

到此，我们完成了旋转操作，如何平移呢？仅是加一个平移常数的事：

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta_x & sin\theta_y \\ -sin\theta_x & cos\theta_y \end{bmatrix} \begin{bmatrix} X_{sp} \\ Y_{sp} \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

到此，我们的2*3大小的仿射变换便推导出来了。

推导知道了，但如何实现呢？

opencv同样给我们提供了计算仿射矩阵的函数接口：

```
getAffineTransform(
  const Point2f* src,    输入图像的点集
  const Point2f* dst     输出图像的点集
  );
```

这个函数可以计算出我们想要图像变换的矩阵，但需要我们输入至少三对点集，点集是什么鬼？为什么是至少三对？

我们可以看到上面公式里有六个变量，因此自然需要至少列六个等式才可计算出该矩阵。

因此我们需要找输入图像和输出图像上一一对应的三对点（3个x,y对应计算式）来作为输入。

这样，我们就可以进行仿射变换啦。

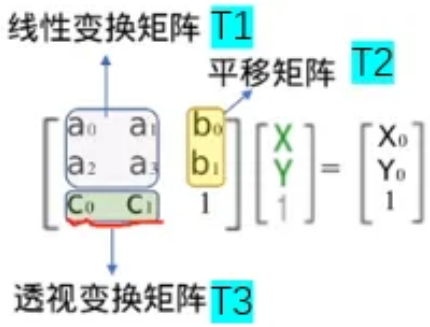
## 透视变换原理

我们说仿射变换是在二维空间中的旋转，平移和缩放。而透视变换则是在三维空间中视角的变化。

opencv中同样给出了透视变换的函数接口：

```
void warpPerspective(
InputArray src,   输入图像
OutputArray dst,   输出图像
InputArray M,    输入透视变换矩阵M
Size dsize,
int flags=INTER_LINEAR,
int borderMode=BORDER_CONSTANT,
const Scalar& borderValue=Scalar()
);
```

和仿射变换基本相同，不同的是输入透视变换矩阵M大小为3*3：




上面矩阵的未知量比仿射变换的矩阵多了一个透视变换矩阵T3（两个未知量），因此我们需要给下面计算透视变换矩阵的函数提供四对以上的点来求解：

```
Mat cv::getPerspectiveTransform (
const Point2f src[],     输入图像点集
const Point2f dst[],     输出图像点集
);
```

T1为线性变换完成旋转，错切和放缩，T2完成平移操作。T3就是设了两个变量来表示映射关系。

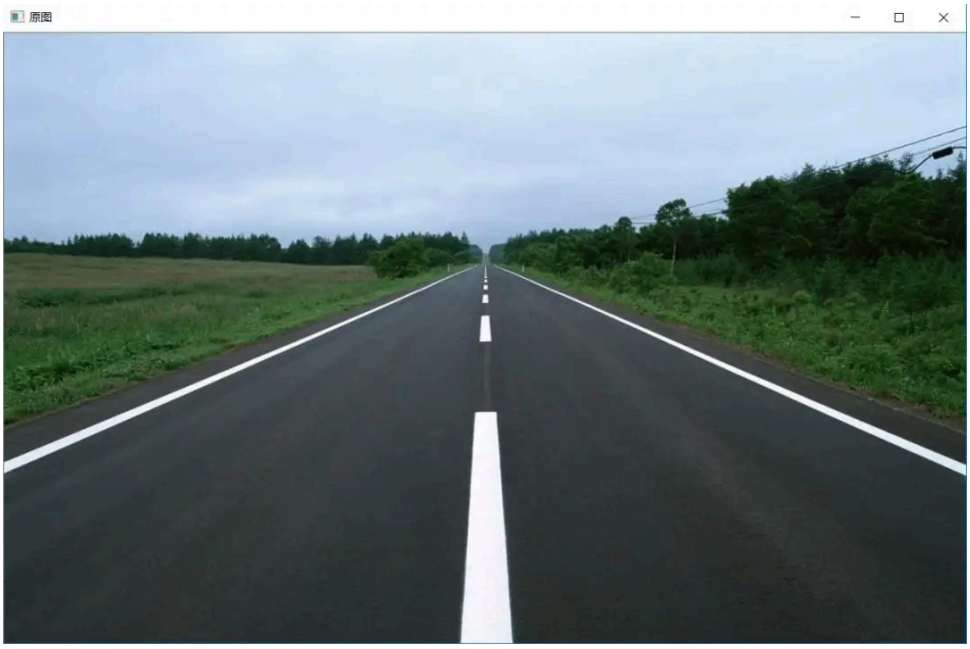
## 编程实现

理解了透视变换的原理后，我们就着手来实现了（代码可以顺次复制即可运行）：

**首先是读取原图片并显示啦：**

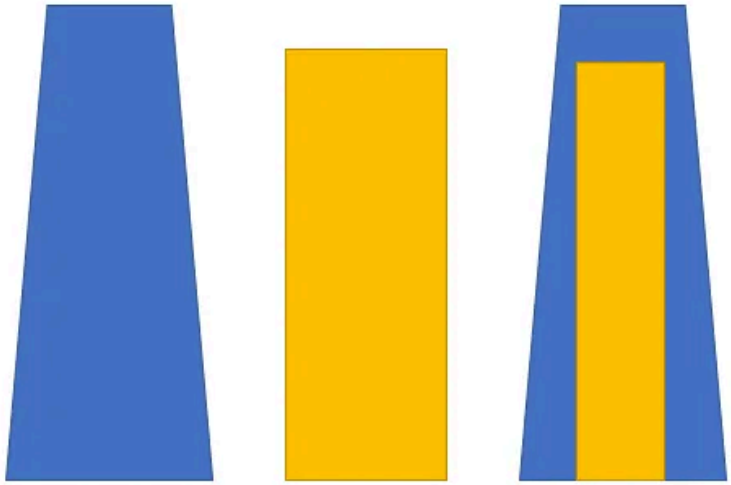
```
#include <opencv2/opencv.hpp>
#include <iostream>
using namespace std;
using namespace cv;
int main()
{
  Mat dstImage,srcImage = imread("road.png");
  cout<<srcImage.size;  //674 x 1020

  imshow("原图", srcImage);
  waitKey();
  return 0;
}
```




然后我们需要选取原图上的四个点，并计算出该四对点变换后的位置。

如何选点？我们可以选两边白条的四个定点。那变换后的位置就需要我们自己估算了，如下图：



我们希望将蓝色的透视变换为黄色的。

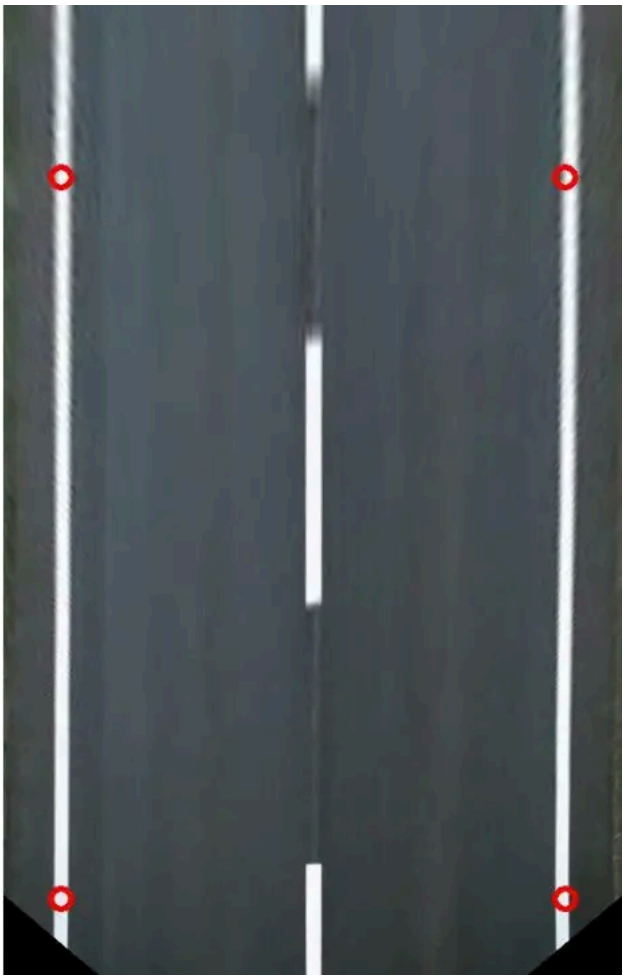
```
Point2f imgPts[4], objPts[4];  //透视前和透视后
//原坐标
imgPts[0].x = 20 * 1020 / 230; imgPts[0].y = 95 * 647 / 145;
imgPts[1].x = 210 * 1020 / 230; imgPts[1].y = 95 * 647 / 145;
imgPts[2].x = 90 * 1020 / 230; imgPts[2].y = 65 * 647 / 145;
imgPts[3].x = 140 * 1020 / 230; imgPts[3].y = 65 * 647 / 145;
//透视后坐标
int road_w = 540; //将透视变换的图片大小改变一下
int road_h = 850;
objPts[0].x = 50;  objPts[0].y = 780;
objPts[1].x = 490; objPts[1].y = 780;
objPts[2].x = 50 ; objPts[2].y = 150;
objPts[3].x = 490; objPts[3].y = 150;
```


我们选取了如图四个点，首先计算透视变换矩阵：

```
//计算透视变换矩阵
Mat H = getPerspectiveTransform(imgPts, objPts);
```

然后进行透视变换：

```
//进行透视变换
warpPerspective(srcImage, dstImage, H, srcImage.size());
//画出透视变换后的四个点
circle(dstImage, objPts[0], 9, Scalar(0, 0, 255), 3);
circle(dstImage, objPts[1], 9, Scalar(0, 0, 255), 3);
circle(dstImage, objPts[2], 9, Scalar(0, 0, 255), 3);
circle(dstImage, objPts[3], 9, Scalar(0, 0, 255), 3);
imshow("变换后", dstImage);
```



这样，我们就得到鸟瞰图啦。

本文中的部分公式截图来自下面视频的PPT：

```
https://www.bilibili.com/video/av97686119
```

这个视频也是介绍仿射变换和透视变换的，大家可以和本文对比着看。

**本文如有表述错误的地方，还望批评指正！**

往期推荐

🔗

**【opencv实践】图像增强基本操作**

**【opencv】带你再学一遍直方图**

**【opencv小项目】深入理解回调函数**


点个在看，就可以评论啦~

上一篇
【opencv实践】图像增强基本操作

下一篇
【opencv实践】你确定真的了解寻找轮廓函数吗？【RM大符识别】