Dlib姿态估计——旋转矩阵与欧拉角互转

2017年12月25日 11:04:41　从程序猿到程序员　阅读数：3916

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/u012525096/article/details/78890463

原文地址：https://www.learnopencv.com/rotation-matrix-to-euler-angles/

简介

在这篇文章中，我将分享将一个3×3旋转矩阵转换成欧拉角的代码，反之亦然。3D旋转矩阵可以让你的头旋转。 我知道这是一个坏的双关语，但真相有时可能是非常小的！

一个旋转矩阵有三个自由度，数学家已经行使了他们的创造自由，以每个想象的方式来表示3D旋转——或使用三个数字、或使用四个数字、或使用一个3×3矩阵。 还有很多不同的方式用三个数字表示一个旋转或用四个数字的一些方法来表示旋转。

例如，3D中的旋转可以表示为三个角度，可以将其表示在X、Y、Z三个轴上。但是，这三个角度也可以表示到Z，Y和X轴上（表示方法不同）。这些角度被称为欧拉角或Tait–Bryan角。在原始欧拉角公式中，通过围绕Z，X轴和再对Z轴（或者对于Y-X-Y或Z-Y-Z）的连续旋转来描述旋转。当旋转被指定为围绕三个不同的轴（例如X-Y-Z）的旋转时，它们应该被称为泰特—布赖恩（Tait-Bryan）角，但是流行的术语仍然是欧拉角，所以我们也将它们称为欧拉角。有六种可以用Tait-Bryan角度描述旋转的方法——X-Y-Z，X-Z-Y，Y-Z-X，Y-X-Z，Z-X-Y，Z-Y-X。现在你在想，选择很简单。 让我们选择X-Y-Z。 对 么？ 答案是错误的。 行业标准是Z-Y-X，因为它对应于yaw偏航，pitch俯仰和roll滚转。

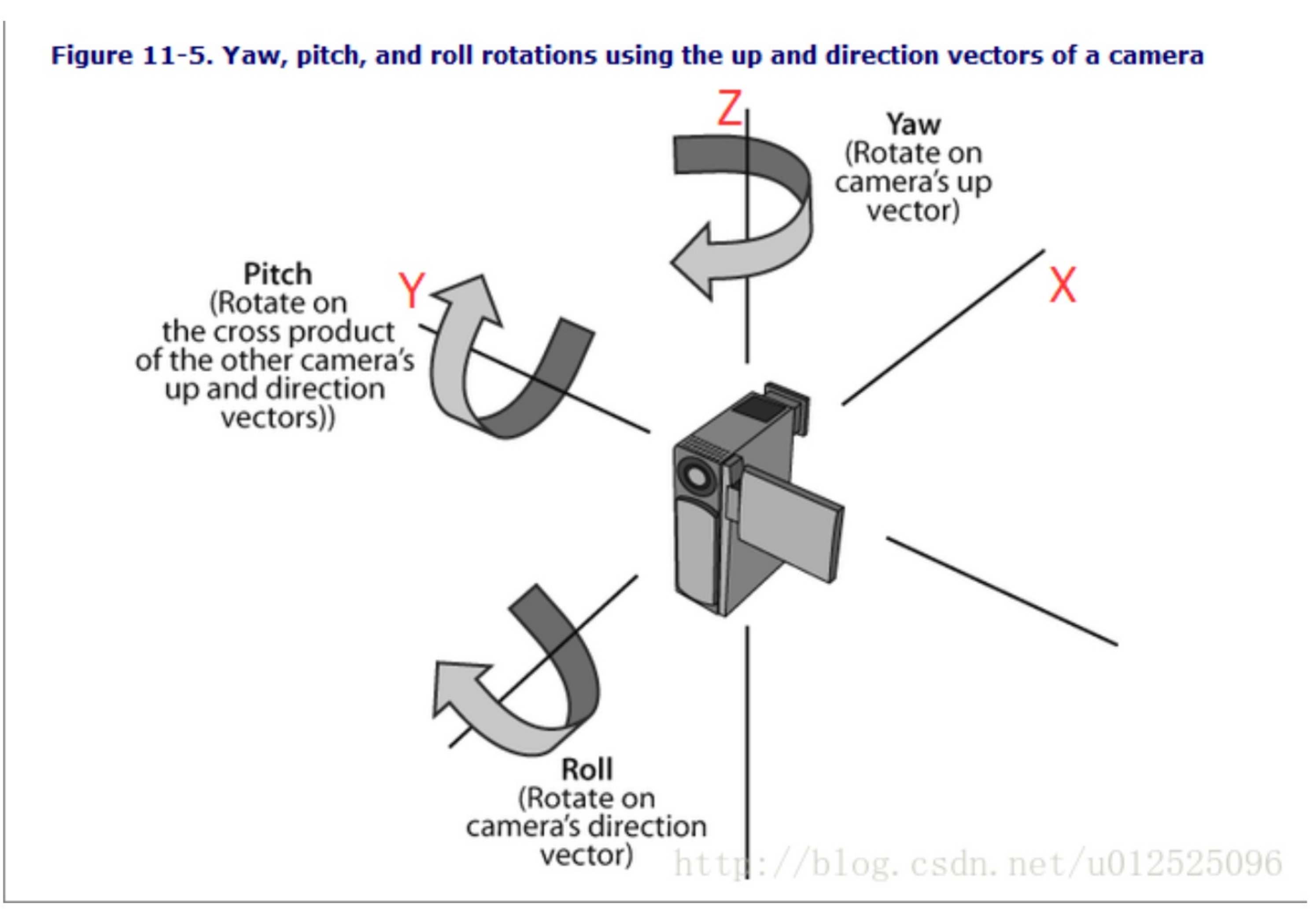
Figure 11-5. Yaw, pitch, and roll rotations using the up and direction vectors of a camera

yaw偏航角

pitch俯仰角

roll滚转角

定义旋转矩阵时还有其他的含糊之处。给定一个点（x，y，z），你可以把这个点想象成一个行向量$[x，y，z]$或一个列向量$[x，y，z]^T$。如果使用行向量，则必须对3×3旋转矩阵进行右乘；如果使用列向量表示，则用3×3旋转矩阵左乘该列向量。这两个旋转矩阵是不一样的（它们是彼此的转置）。

我的观点是没有标准的方法来将旋转矩阵转换成欧拉角。所以，我决定（几乎）与MATLAB实现的`rotm2euler.m`一致。唯一的区别是他们返回的欧拉角z轴是第一个和x轴是最后一个（Z-Y-X）。我的代码先返回x（X-Y-Z）。

欧拉角->旋转矩阵

对于3D旋转的最简单的方法是以轴角形式思考。任何旋转都可以由一个旋转轴来定义，一个角度可以描述旋转的量。比方说，你想旋转一个点或一个参考框架绕x轴旋转θ_x度。与该旋转对应的旋转矩阵由下式给出：

$$\mathbf{R_x}=\begin{bmatrix}1&0&0\\0&\cos(\theta_x)&-\sin(\theta_x)\\0&\sin(\theta_x)&(\cos\theta_x)\end{bmatrix}$$

θ_y和θ_z关于y和z轴的旋转可以写成：

$$\mathbf{R_y}=\begin{bmatrix}\cos(\theta_y)&0&\sin(\theta_y)\\0&1&0\\-\sin(\theta_y)&0&(\cos\theta_y)\end{bmatrix}$$

$$\mathbf{R_z}=\begin{bmatrix}\cos(\theta_z)&-\sin(\theta_z)&0\\\sin(\theta_z)&(\cos\theta_z)&0\\0&0&1\end{bmatrix}$$

关于任意轴的旋转$\mathbf{R}$可以用关于Z，Y和最后X轴的连续旋转来写，使用下面显示的矩阵乘法。

$$\mathbf{R}=\mathbf{R_z}\mathbf{R_y}\mathbf{R_x}$$

在这个公式中，θ_x，θ_y和θ_z是欧拉角。给定这三个角度，首先找到$\mathbf{R_x}$，$\mathbf{R_y}$和$\mathbf{R_z}$，然后将它们相乘得到$\mathbf{R}$，就可以很容易地找到旋转矩阵。

C++代码

```cpp
// Calculates rotation matrix given euler angles.
Mat eulerAnglesToRotationMatrix(Vec3f &theta)
{
    // Calculate rotation about x axis
    Mat R_x = (Mat_<double>(3,3) <<
               1,       0,              0,
               0,       cos(theta[0]),   -sin(theta[0]),
               0,       sin(theta[0]),   cos(theta[0])
               );

    // Calculate rotation about y axis
    Mat R_y = (Mat_<double>(3,3) <<
               cos(theta[1]),    0,      sin(theta[1]),
               0,               1,      0,
               -sin(theta[1]),   0,      cos(theta[1])
               );

    // Calculate rotation about z axis
    Mat R_z = (Mat_<double>(3,3) <<
               cos(theta[2]),    -sin(theta[2]),      0,
               sin(theta[2]),    cos(theta[2]),       0,
               0,               0,                  1);


    // Combined rotation matrix
    Mat R = R_z * R_y * R_x;

    return R;
}
```

在OpenCV中将旋转矩阵转换为欧拉角

将旋转矩阵转换成欧拉角是有点棘手的。该解决方案在大多数情况下不是唯一的。使用上一节中的代码，可以验证与欧拉角$[0.1920, 2.3736, 1.170]$（或$[11, 136, 64]$)(度)）和$[-2.9496, 0.7679，\ -2.0246]$（或$[-169, 44, -116]$(度))实际上是相同的，尽管欧拉角看起来非常地不同。下面的代码显示了给定旋转矩阵的欧拉角的一种方法。下面代码的输出应该与MATLAB的$rotm2euler.m$的输出完全匹配，但是x和z的顺序是交换的(Z-Y-X)。

C++

```cpp
// Checks if a matrix is a valid rotation matrix.
bool isRotationMatrix(Mat &R)
{
    Mat Rt;
    transpose(R, Rt);
    Mat shouldBeIdentity = Rt * R;
    Mat I = Mat::eye(3,3, shouldBeIdentity.type());

    return  norm(I, shouldBeIdentity) < 1e-6;

}

// Calculates rotation matrix to euler angles
// The result is the same as MATLAB except the order
// of the euler angles ( x and z are swapped ).
Vec3f rotationMatrixToEulerAngles(Mat &R)
{

    assert(isRotationMatrix(R));

    float sy = sqrt(R.at<double>(0,0) * R.at<double>(0,0) +  R.at<double>(1,0) * R.

    bool singular = sy < 1e-6; // If

    float x, y, z;
    if (!singular)
    {
        x = atan2(R.at<double>(2,1) , R.at<double>(2,2));
        y = atan2(-R.at<double>(2,0), sy);
        z = atan2(R.at<double>(1,0), R.at<double>(0,0));
    }
    else
    {
        x = atan2(-R.at<double>(1,2), R.at<double>(1,1));
        y = atan2(-R.at<double>(2,0), sy);
        z = 0;
    }
    return Vec3f(x, y, z);
}
```