项目名称：基于标记的AR的OpenCV实现及在Unity中的应用学习

作者： 史昀捷

指导老师： 李春

**使用方法：**

PC：（由于PC没有可使用的摄像头，所以PC端采用读取现成图片的方法。若PC有可使用的摄像头，PC端可直接使用安卓端的代码。）

1. Unity项目下Resource内放入需要读取的图片；
2. 在CamSetup.cs第53行修改所要读取图片的文件名；
3. Unity中运行场景。

Android：

1. 安装APK文件夹中的.apk文件，并允许使用摄像机；
2. 将摄像头对准Marker即可。

**算法解析：**

**OpenCv部分：**

1. **findMarkers：**

寻找标识是对图片进行分析得到图中所有标识，标识需要满足一下条件：

1. 足够大的闭合的凸四边形
2. 有完整的边框和合理的ID。

具体步骤如下：

1. 使用cv:: findContour() 对二值化的图片进行搜索，找到所有轮廓。
2. 使用cv:: approxPolyDP() 提取轮廓的顶点，并判断轮廓是否为闭合的凸四边形。
3. 求每个顶点之间的距离，判断所得到的四边形是否足够大。
4. 对相邻的两个四边形进行比较，取其中较大的一个四边形。
5. 将通过2-4测试的四边形放入一个“可能为标识”的集合。
6. 通过cv::getPerspectiveTransform()和cv:: warpPerspective()将“可能为标识”的标识去掉透视投影，得到平面／正面的矩形。
7. 根据海明码求出标识的id，确认标识，并保存到集合中。

每个标记都有一个内部编码。标记被分成7x7的网格，其中内部5x5的网格包含ID信息。其余部分是黑色边界。因此首先需要检查外部黑色边界是否存在，然后读取5x5的网格是否存在有效的标记编码（因为检测出来的标记可能是旋转的，要旋转标记编码来得到有效的标记编码）。

每个标记可以划分成7\*7个方格，黑格子表示0，白格子表示1。这样标记内部将有5个数字，而每个数字由5个bit表示。具体编码方式类似于海明码，3个bit用于校验，2个bit用于存放数据，因此每5个bit可以表达4种数据，而5行这样的编码可以表达4^5=1024个数据。

首先检查四边形轮廓是否完整，即通过统计方块内非零像素值个数，若大于方块内像素个数的一半，则认为该方块是白色的。按行遍历所有轮廓方格，方格大小为100/7，只要有一个轮廓方格被判定为白色，那么整个轮廓就是不完整的，舍弃该Marker

然后，同理识别5\*5编码区域，将0-1编码写入bitMatrix矩阵。由于IPAD拍摄照片存在旋转变化，因此每个矩形方格具有四种旋转状态，即直接从当前矩形区域解码可能是旋转过的图片，不能代表真正的数据。

本文为所有旋转状态下的Marker计算海明距离，选择海明距离最小的作为最终的编码矩阵。海明距离的计算：

hammDistMarker函数中，ids数组的由来。采用3位校验2位数据，因此每个stripe的2位数据将产生4种海明编码。也就是说ids数组列举了Marker中每行数据的所有可能取值。

Marker中的一行表示一个数据，我们把bitMatrix的每一行同ids中的一行数据依次比较，总能寻找到ids中最贴近bitMatrix第x行的一行ids。再把bitMatrix对应的ids值求和，即可得到海明距离。

最后，在确定了Marker的旋转状态后，mat2id函数对该Marker进行解码，即遍历各行，或运算、移位运算得到最终的ID。

确定了Marker的旋转状态后，对四边形顶点按照旋转状态排序，无论相机如何拍摄都使四边形中间的顶点排在第一个。 而后，通过亚像素技术cornerSubPix函数对顶点位置进一步细化。所谓亚像素，两个像素之间，还存在像素，它完全由计算得到。

1. **estimatePosition：**

对每一个标记，求出其相对于相机的转换矩阵。找到上面监测到的标记的位置。

为了将三维模型放置在场景中，需要知道它关于摄像机的姿态。可在直角坐标系中使用欧式空间+变换来表示这个姿态。

三维世界中Marker的位置与其对应的二维投影，遵从以下公式：

**P = A \* [R | T] \* M**

其中，

M表示三维世界中的点；

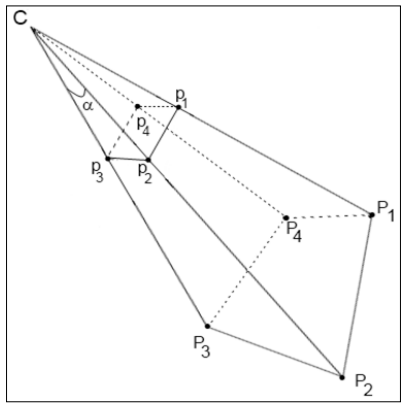
[R|T]表示欧氏变换，是一个3\*4矩阵

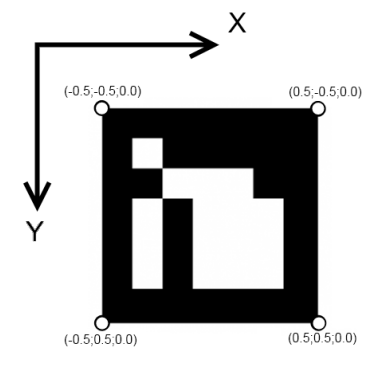
A表示相机参数矩阵，存放相机内部参数

P表示M在二维空间的投影，是一个二维点。

在执行标记检测后，需要知道二维情形下标记的四个角点位置（在屏幕空间中的投影）。

位姿估计就是如何获取矩阵A，参数向量M，以及计算变换矩阵[R|T].具体步骤如下：

在三维空间中，可通过标记角点的精确位置来估计摄像机与标记之间的变换。此操作称为二维到三维的姿态估计。该估计过程会在物体与摄像机之间找到一个欧式空间的变换，该变换仅由旋转矩阵和平移矩阵构成[R|T]。如右图所示，C代表相机中心，P1-P4是现实三维世界中的点，p1-p4是物体在相机图像二维平面上投影的点。姿态估计的目标就是找到p1-p4到P1-P4之间的转换关系。现在p1-p4是已知的，就是在图像平面求到的标识的二维坐标点。而P1-P4， 标记总是方形且所有顶点都在一个平面，可将标记放在XY平面上（Z分量为0），其标记的中心为点（0.0,0.0,0.0)。坐标系统的起始处就是标记的中心（Z轴是与标记平面垂直）。如下图所示：



得到了以上的关系后，即可通过cv:: solvePnP()及可根据相机的内置矩阵的到欧氏变换[R|T]。

1. 确定相机的视界。

使用cv:: calibrationMatrixValue()根据当前图片的大小及相机的内置参数的到当前相机的视界。目的是为了使opencv所使用的相机属性和Unity保持一致。

1. 将以上步骤所的到的结果（标识的ID，标识的变换矩阵，相机的视界）转换为C#可读的字符串传输给Unity C# script。

**Unity部分：**

1. 打开后置摄像头，取出当前相机的纹理。
2. 运用RenderTexture对当前相机的纹理进行缩小，提高效率。
3. 使用GCHandle将缩小后的纹理储存在一块内存中供OpenCV使用
4. 读取由OpenCV处理得到图中得到标识的ID，标识的变换矩阵，相机的视界。
5. 将视界参数应用到Unity相机中；根据标识的ID以及变换矩阵创建相对应的物体。

**参考文献:**

**Unity部分：**

MarkerBased AR Example by Enox Software

<https://www.assetstore.unity3d.com/en/#!/content/29678>

**OpenCV部分：**

基于标记的AR的opencv实现 by chuhang\_zhqr

<http://blog.csdn.net/chuhang_zhqr/article/details/50034669>

<http://blog.csdn.net/chuhang_zhqr/article/details/50036443>

**Android Studio NDK使用教程：**

https://www.youtube.com/watch?v=0lZA90BnI7U