

三维重建山威闻天楼和艺术学院与实验楼及其目标检测

三维重建，目前已成为计算机视觉领域一大热门研究方向，并且在医学影像、文物遗迹重建、VR等领域有着非常大的应用潜力。三维重建的主要任务是凭借单张或者多张图片以及如深度等其他信息，将2D图像转换为3D模型。

在计算机视觉中，三维重建是指根据单视图或者多视图的图像重建图像重建三维信息的过程。由于单视频的信息不完全，因此三维重建需要利用经验知识。而多视图的三维重建(类似人的双目定位)相对比较容易，其方法是先对摄像机进行标定，即计算出摄像机的图象坐标系与世界坐标系世界坐标系)的关系，然后利用多个二维图象中的信息重建出三维信息。

常见的三维重建表达方式共分为四类：深度图、点云、体素、网格。在深度图中，图的每个像素值代表的是物体到相机xy平面的距离。点云是点的数据集，点可以包含多样的信息，包括坐标、颜色、类别、时间等等。

本文主要针对项目需要对山东大学（威海）的闻天楼和艺术学院与科学实验楼进行三维重建，并根据三维重建结果进行边缘检测、面积识别以及目标检测等操作。其中，两栋教学楼的三维重建主要采用SFM加MVS的方法；对窗户的识别与教学楼面积体积计算主要采用边缘识别及大津算法等；最后，对多个事物种类的目标检测，本文首先选择热度较高的yolo算法进行训练，最后再针对SSD算法进行了自主训练集拍摄整理并且作出了可视化处理。

下面是对三维建模的预先展示，闻天楼：



艺术学院与科学实验楼：



一、基于深度学习对两建筑物进行三维重建

本章旨在使用深度学习技术对闻天楼以及艺术学院与实验楼进行完整三维重建，无死角、无畸变。

本文所用的两个方法：SFM和MVS。其中SFM是MVS的上游。一个可以理解为稀疏重建，另一个可以理解为稠密重建。SFM给MVS算好了输入视角的位姿、内参、稀疏点以及它们的共视关系，MVS再利用这些信息以及彩色图来估计深度图以及做最后的操作。

STEP 1 获得点云信息

1. 特征提取

特征提取的目的，是从图片的EXIF提取相机参数。基本上，Exif文件格式与JPEG 文件格式相同。Exif按照JPEG的规格在JPEG中插入一些图像/数字相机的信息数据以及缩略图像。于是你能通过与JPEG兼容的互联网浏览器/图片浏览器/图像处理等一些软件来查看Exif格式的图像文件。就跟浏览通常的JPEG图像文件一样。

COLMAP 实现了不同复杂程度的不同相机模型。如果没有先验已知的固有参数，通常最好使用最简单的相机模型，该模型足够复杂，可以对失真效果进行建模。

可以通过双击模型查看器中的特定图像或导出模型并打开相机.txt文件来检查估计的固有参数。

为了获得最佳的重建效果，可能需要针对问题尝试不同的相机型号。通常，当重建失败并且估计的焦距值/失真系数严重错误时，这是使用过于复杂的相机模型的迹象。相反，如果 COLMAP 使用许多迭代的局部和全局束调整，则表明使用了过于简单的相机模型，无法对失真效果进行完全建模。

2. 特征匹配

在特征匹配中，业内常用的大致是四中方法：

Exhaustive：官方文档中说明Exhaustive耗时最长，图像两两匹配，理论上效果最好。但在实际发现这种方式耗时长而且效果未必好，对于场景位置的判断时不断出现失误。比如局部拍摄时无法确定某些图片的位置，场景相似时会出现误判。

Sequential: 拍照时按照时间顺序，相邻的场景往往在一块，适合使用这种匹配方式。以相邻x张图像匹配，上面的问题就不会遇到。但在切换角度过大，或者拍摄开始和拍摄结束是同一个视角时会有误差。

Spatial: 这种方式是利用了地理位置信息，也就是每张图像必须自带位置信息，例如gps。这种方式需要设置在多大范围内进行匹配，因此需要事先知道大概重建场景大小，以便选择合适参数。

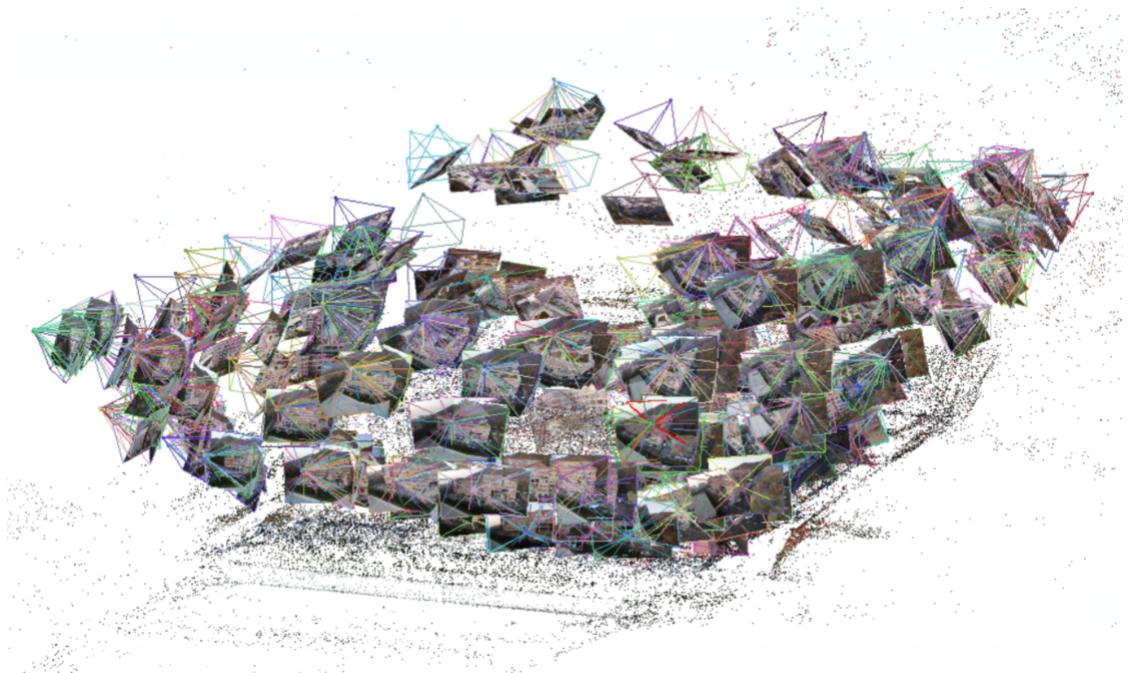
Custom: 自定义匹配方式，根据自身需求进行参数调整。

本文结合实际运行效果、运行速度及自身情况。主要采用前两种匹配方法，以求速度与准确度相结合。

3. 稀疏重建

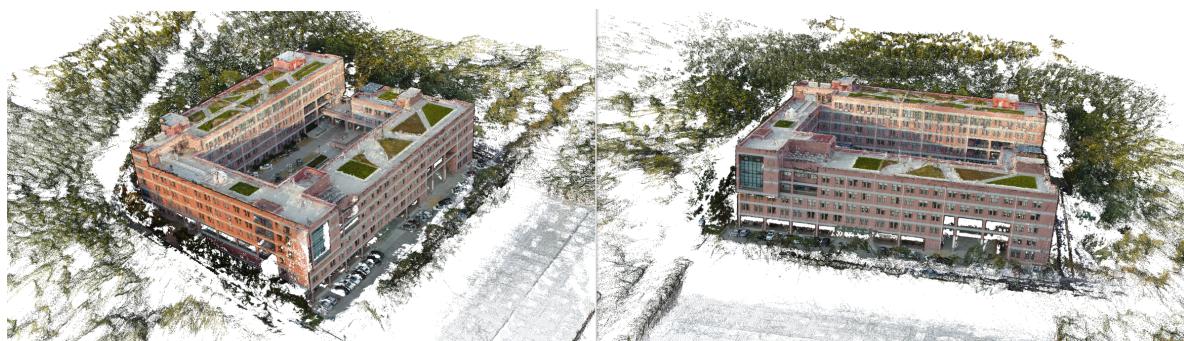
采用增量式SfM (出自openMVG: open Multiple View Geometry library. Basis for 3D computer vision and Structure from Motion) , 逐步增加视角，并进行迭代优化重投影误差，计算不同视图的相机参数、得到场景的稀疏点云和确定不同视图与点云间的可视关系，最后可以得到场景的稀疏点云和各个视角的相机姿态。

如果效果很不好，则重新考虑匹配方式。过程概览如下：

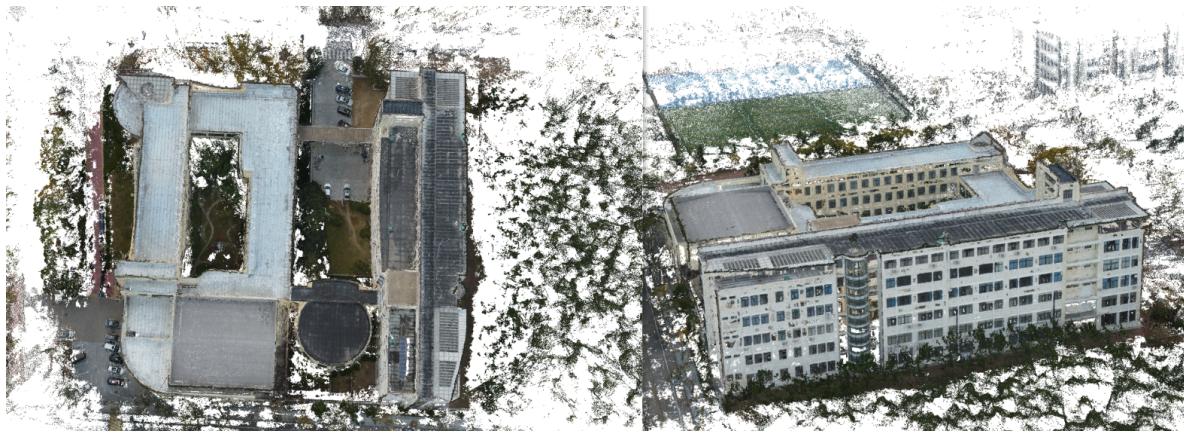


STEP 2.1 基于MVS对点云信息进行三维重建

下面先展示下面三维建模结果如下，闻天楼：



艺术学院与科学实验楼：



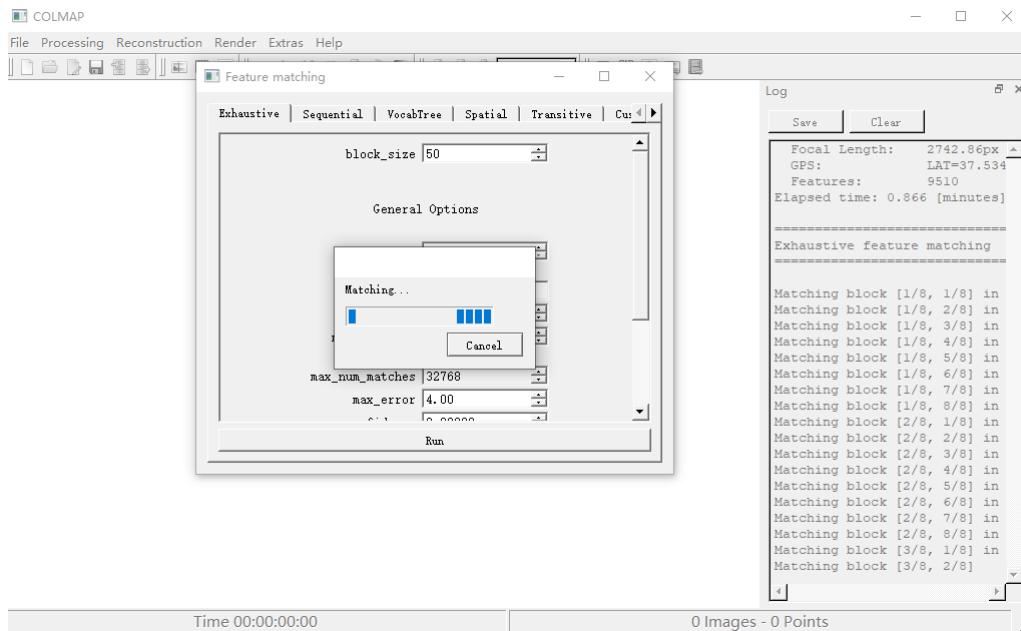
想象，对于在两张图片中的同一个点。现在回到拍摄照片的那一刻，在三维世界中，存在一条光线从照片上这一点，同时穿过拍摄这张照片的相机的成像中心点，最后会到达空间中一个三维点，这个三维点同时也会在另一张照片中以同样的方式投影。

这个过程这样看起来很普通，就如同普通的相机投影而已。但是因为两张图片的原因，他们之间存在联系，这种联系的证明超过了能力范围，但是我们只需要知道，此种情况下，两张照片天然存在了一种约束。

SFM的重建成果是稀疏三维点云，为了进入更加深刻的领域，获得更好的结果，我们进入到MVS。

1. 去畸变

进行深度图估计前的第一步为影像去畸变，在COLMAP中，使用光学一致性和几何一致性联合约束构造匹配代价，带有畸变的影像会导致边缘有较大的视差估计误差。其操作界面如图：



2. 深度估计

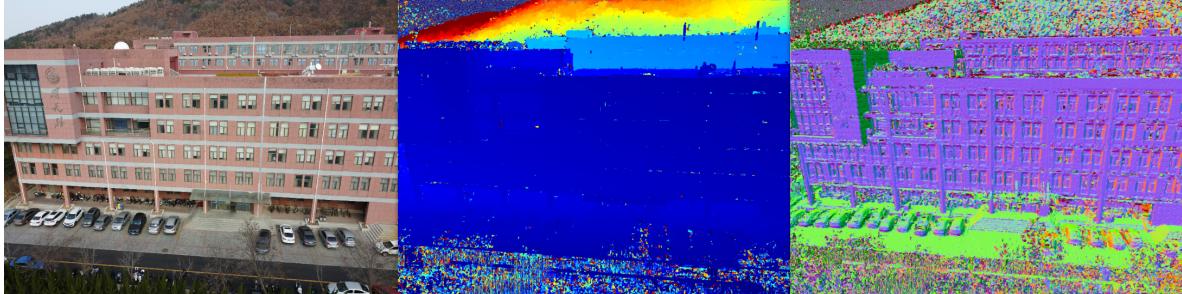
MVS的深度估计一般是通过神经网络直接学习的。网络训练方法是输入代价体和对应深度图真值，利用SoftMax回归每一个像素在深度处的概率，得到一个表示参考影像每个影像沿深度方向置信度的概率体P，以此完成从代价到深度值的学习过程。

当已知概率体时，最简单的方法可以获取参考影像的所有像素在不同深度的概率图，按照赢者通吃原则。直接估计深度图。然而，赢者通吃原则无法在亚像素级别估计深度，造成深度突变、不平滑情况。所以需要沿着概率体的深度方向，以深度期望值作为该像素的深度估计值，使得整个深度图中的不同部分内部较为平滑。

实际上，COLMAP会利用光学一致性（photometric）同时估计视角的深度值和法向量值，并利用几何一致性（geometric）进行深度图优化。深度估计结束后可以得到photometric和geometric下的深度图和法向量图。

观察结果发现，COLMAP可以获得较为精确的深度估计值，但在深度图完整度和深度连续性方面仍存在一些问题，深度图存在较多漏洞。

具体操作时，mainmap, depthmap, normalmap如下：



3. 稠密重建

对估计出来的深度图，首先通过配准进行深度图融合，然后通过下式，按照投影方法进行点云恢复。

$$z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = P \cdot p^w = K(R | t)p^w$$

4. mesh重建

使用Poisson进行mesh重建，得到有颜色的mesh模型，或者通过Delaunay网格得到没有纹理的mesh模型。

(1) Poisson

Poisson重建是一个非常直观的方法。它的核心思想是点云代表了物体表面的位置，其法向量代表了内外的方向。通过隐式地拟合一个由物体派生的指示函数，可以给出一个平滑的物体表面的估计。

给定一个区域M及其边界 ∂M ，指示函数 χ_M 定义为

$$\chi_M(x) = \begin{cases} 1 & x \in M \\ 0 & x \notin M \end{cases}$$

这样，把重构 $S=\partial M$ 的问题转换为重构 χ_M 的问题。

简单列几点：

- Poisson在边缘处的锐度比VRIP要好。这是因为VRIP在大的边缘处TSDF的累加会有平滑效应，而Poisson依据的是法向量，不会引入额外的平滑。
- VRIP是局部方法，每次只更新当前深度图对应的TSDF。Poisson是全局方法。
- 从个人使用经验上看，Poisson对于噪声更加鲁棒一些。点云法向量估计的精度不能太差。
- 如果重建出奇怪的形状（分层、分块），请查看原始点云是否平滑，是否有噪声，调整生成网格的分辨率以适应点云。

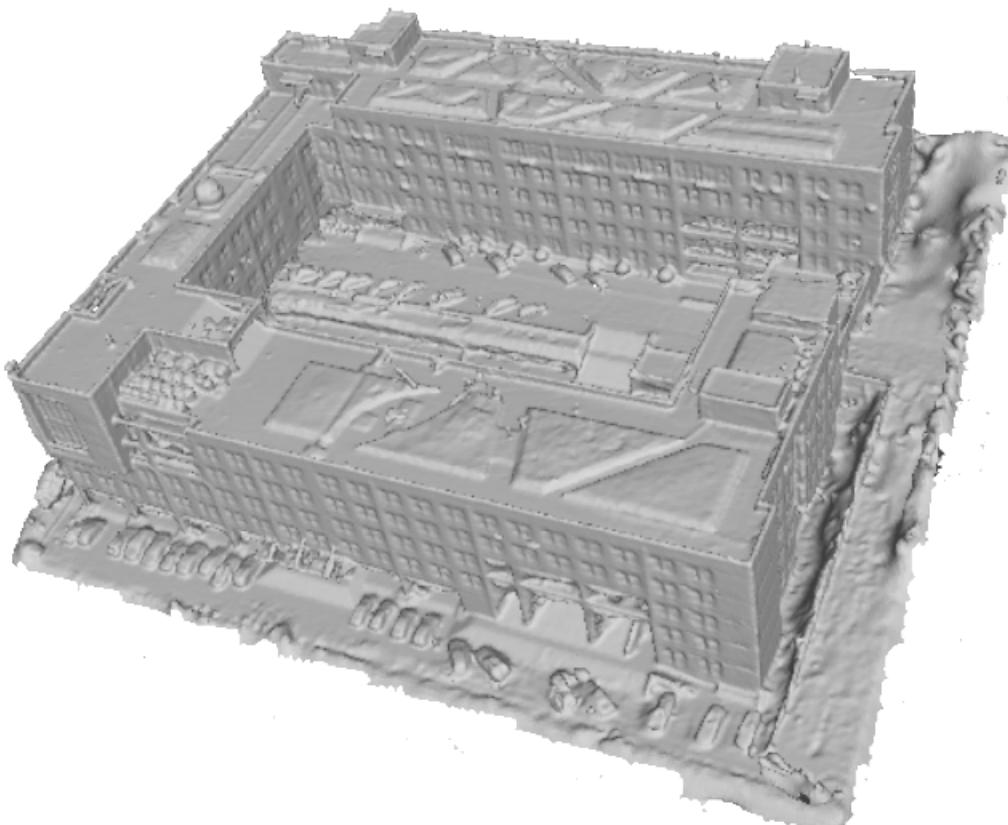
如下展示该算法下的三维模型：



(2) Delaunay

Delaunay 细化算法构建三角形或四面体 ("元素") 的网格，适用于插值、渲染、地形数据库、地理信息系统等应用，以及要求最高的有限元方法偏微分方程的求解。Delaunay 优化算法通过维护 Delaunay 或受约束的 Delaunay 三角测量来运行，该三角测量通过插入其他顶点进行优化，直到网格满足对元素质量和大小的限制。这些算法提供了元素质量、边长和元素大小的空间分级的理论边界。复杂域的拓扑和几何保真度，包括具有内部边界的弯曲域；并在实践中真正令人满意的表现。

如下展示该算法下的三维模型：



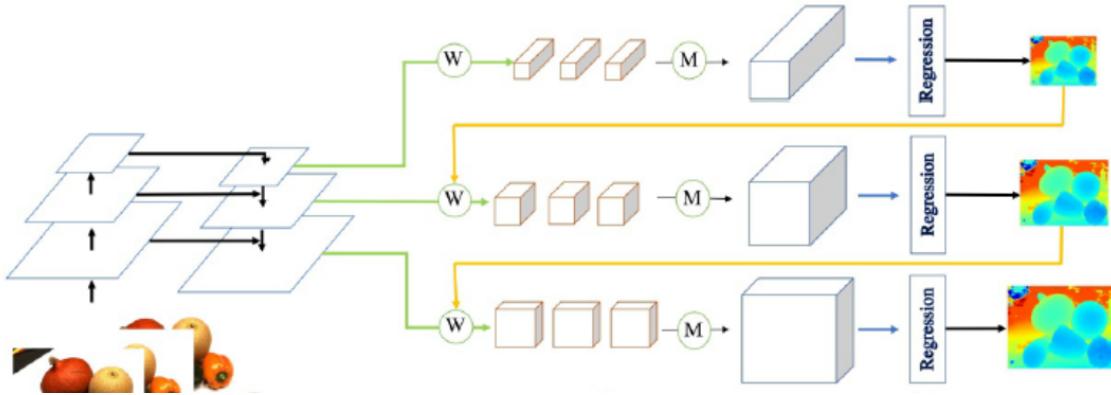
该种模型可用作STEP3中进一步优化。

STEP 2.2 基于Cascade-MVSNet对点云信息进行三维重建

上一节中MVS使用循环神经网络，对将代价体沿深度切成不同的深度图，利用GRU结构进行正则化处理，相比较MVSNet，能减少GPU的消耗。可是循环神经网络会涉及一个遗忘的过程，导致网络不能很好地保留像素周围的纹理信息，所以点云完整度不能得到很好地保留。

为了解决信息保留和GPU消耗的两个问题，Cascade-MVSNet提出一种级联的代价体构造方法，并输出从粗到细的深度估计值。

首先来看其整个网络的架构：



可以看到，整个网络的结构还是沿袭了MVSNet框架，还是以多视图的影像作为输入，然后经过可微分单应性变换形成不同视角下的特征体，在通过代价体构造，形成一个代价体，之后通过回归估计深度值。

与上节方法不同，整个网络利用级联式代价体的构造策略。首先，原始输入影像，利用特征金字塔网络先对原始影像进行降采样，降低特征体的分辨率，使得可以拥有较为精确的深度估计范围。通过初始的MVSNet框架估计出低分辨率下的深度图后，进入下一阶段。

下一阶段开始时，先进行上采样，然后以上一层的深度估计范围作为参考，确定改成的深度估计范围和深度估计间隔。最后输出一个较高分辨率的深度图。

根据稀疏重建给予的先验深度范围，第一阶段的深度估计范围将包含整个场景。所以，会和低分辨率影像建立一个体量较小的代价体（图5）。在之后的阶段中，深度估计范围会进一步缩小。



除了深度估计范围的缩小，深度估计间隔也会进一步缩小，越小的深度估计间隔代表着越精细的深度估计精度。

$$I_k + 1 = I_k \cdot p_k$$

在第 k 个阶段，假设当前的深度估计范围 R_k 和深度平面估计间隔 I_k ，所以对应的深度估计平面数可以表示为：

$$D_k = \frac{R_k}{I_k}$$

精确的深度估计结果，但同时也会提升 GPU 的消耗。同时根据特征金字塔网络的特点，在级联式代价体构造过程中，每个阶段将按照上一阶段的两倍数量，即在每个阶段的图像的分辨率分别是之前的两倍。

下面展现三维建模效果如下：



STEP 3 使用 MeshLab 进行三维重建

在先前的两个方法中，我们输出的结果中有一个.out文件，存储着每个相机的位置及重建出的稀疏点云以及一组(个)ply文件，存储着由稀疏点云重建出的稠密点云。为了进一步消除死角和畸变，让模型更加真实，下面使用Meshlab，过一系列操作可创建出包含纹理的、干净的、高分辨率的网格，并自动计算UV映射及创建纹理图像。

其操作过程如下：

1. 打开bundle.rd.out 和list.txt文件
2. 生成稠密点云代替稀疏点云
3. 网格化，利用Poisson Surface Reconstruction算法由稠密点云生成多边形网格表面。 Octree Depth控制着网格的细节，此值越大细节越丰富但占内存越大运行起来慢，一般设10，可慢慢调大。（其中，在实际网格化过程中，Octree Depth=12）
4. 修复流形边缘，后续的纹理处理要求网格化的模型必须是流形(MANIFOLD)的，因此需删除非流形边。

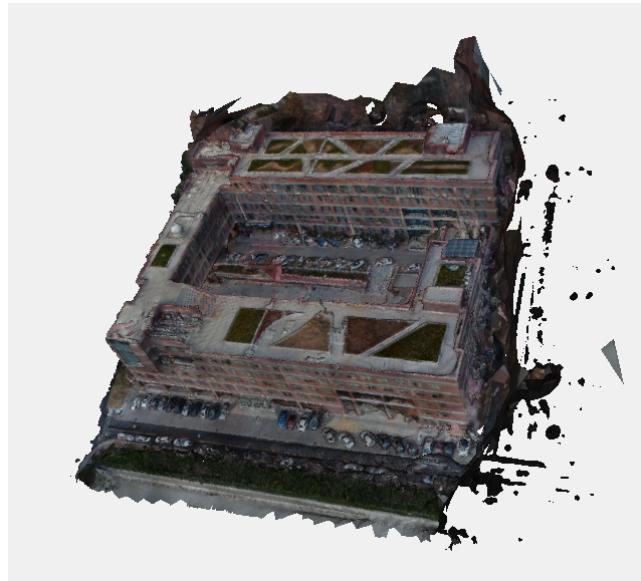


5. 参数化 (Parameterization) , 具体过程为: Filter-> Texture -> Parameterization from registered rasters。以艺术学院与科学实验楼为例, 我们得到了UV映射如下:



6. 投影纹理, 具体过程为: Filter-> Texture -> Project active rasters color to current mesh, filling the texture。该过程可以设置任意分辨率 (512的2的二次方倍:512 / 1024 / 2048 / 4096 / 8192...) 的纹理图。本文生成的分辨率是1024。

下面展示三维建模如下:



算法特点对比

下面直观比较MVS、Cas—MVSNet、MeshLab修复的建模结果，以闻天楼为例：



相比之下MVS运行时间长，对图片数量要求低；同时Cas—MVSNet对图片数量有较大需求；MeshLab修复难度较低，只需要运用开源软件即可得到较理想结果。在本次建模中，MVS重建结果对比之下较为理想。

二、计算机视觉计算建筑楼层数、窗户数量、面积等数据

本章需要结合计算机视觉课程知识，识别并计算出楼层数目、最高高度、最宽宽度、最长长度、窗户数目、窗户面积、整个建筑的体积。

(1) 边缘检测识别教学楼以计算面积

通常，图像中感兴趣区域的位置已知或可以确定。这意味着在相应的边缘图像中关于像素的知识是可用的。在这种情况下，我们可以使用在区域的基础上连接像素的技术，所期望的结果是该区域边界的近似。对这类处理的方法是一种函数近似，这里，我们对已知点拟合一条二维曲线。典型的。我们的兴趣在于快速执行的技术，他可对边界的基本特性产生一个近似，例如端点和凹点。多边形近似尤其有吸引力，因为在保持边界表示相对简单的情况下它们可以捕捉基本形状特性。

这种方法将关注聚焦于物体与背景的边缘像素，在边缘的灰度跳动非常明显，由此得到的灰度直方图将会得到很大的改善。

在这里我们求得边缘的方法主要是梯度算子和拉普拉斯算子。一般来说我们确定阈值 T 是根据，梯度最大值或者拉普拉斯最大值的某百分比来确定的。当有不同需求时，采用不同的占比。

关于边缘改进全局阈值处理基本实现是：

1. 先计算其边界，利用拉普拉斯或者梯度变换都行。
2. 计算变化后边界图像的绝对值

3. 指定一个阈值（一般以百分比的形式指定，比如我指定90%，如果存在有灰度k, 灰度小于K的像素总数占全部像素的90%，那么K就是我们要求的灰度）
4. 对2中计算完的图像进行阈值变换，转化为2值图像
5. 用4中计算得到的二值图像乘原始图像。
6. 计算5中计算的到的图像中灰度大于0的直方图。
7. 对6中得到的直方图进行全局分隔。
8. 提出全局分隔出来的灰度，用该灰度对原始图片进行阈值分割，即可得到结果。

在实际过程中，其实现过程如下：

首先连接测试图像，然后通过Sobel算子计算图像的梯度与角度。下一步进行图像二值化。通过设置梯度门限，再在水平方向和垂直方向上分别计算。进而进行填充空当的过程，在水平二值图像沿x方向进行拓展，再沿垂直方向在y方向上进行填充。对两个不同方向的图像进行逻辑“或”(OR)操作后，通过ImageThinning()函数进行细化操作。结果如下：

如下是将图片转化为灰度图：

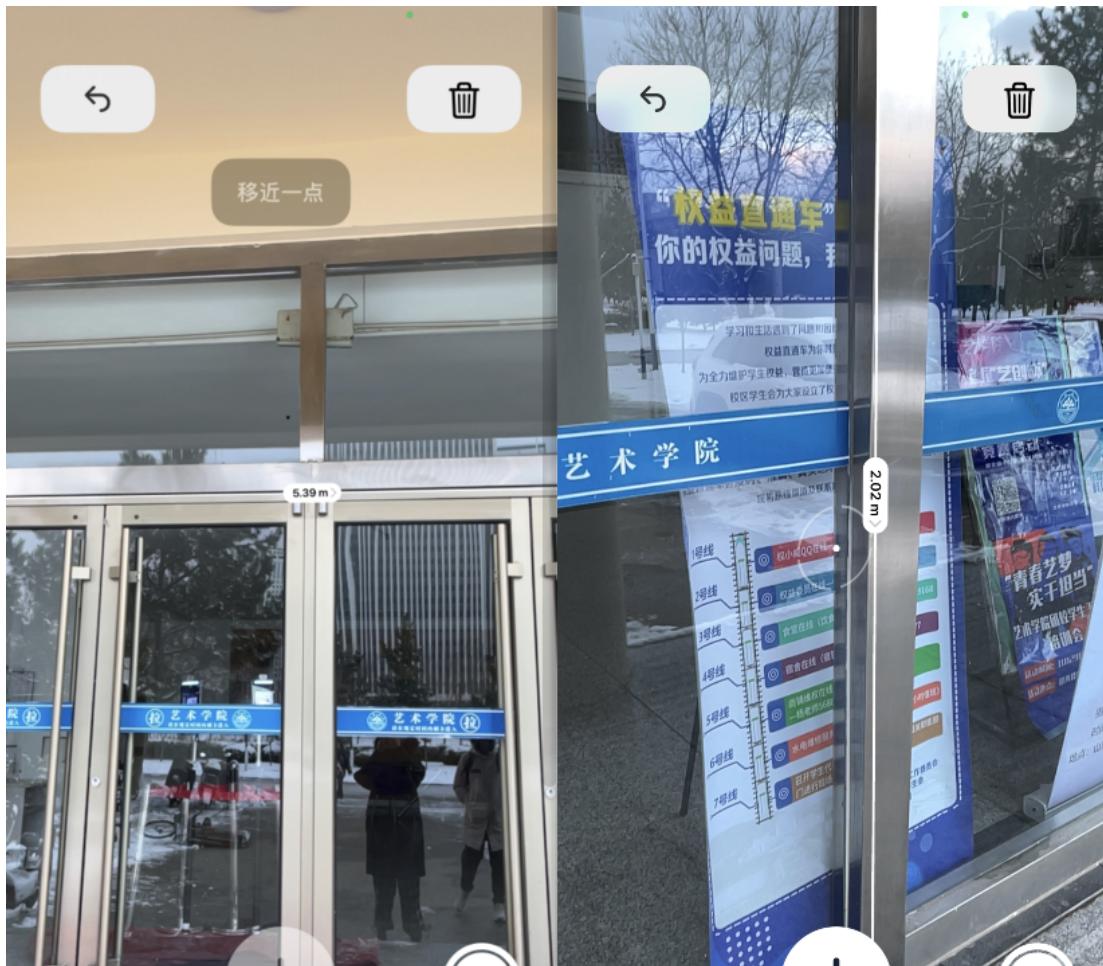


其次是进行边缘检测再进行去噪和填充：



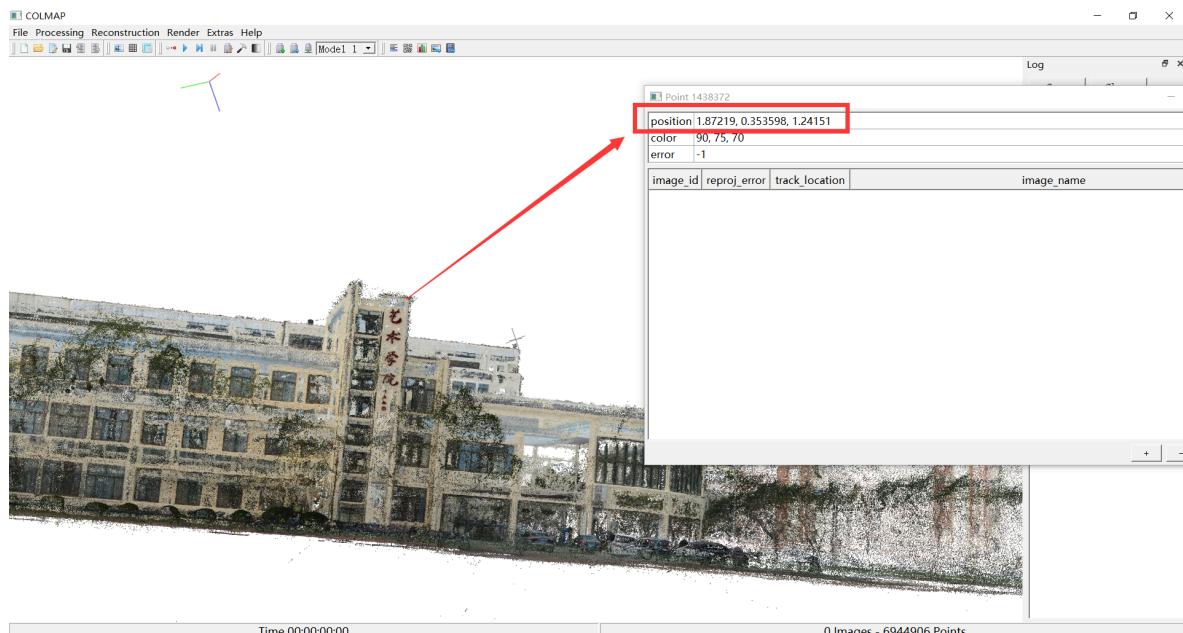
(占地面积识别计算过程与窗户识别类似，这里不加以赘述)

下面进行面积计算，我们以艺术学院大门为基准单位进行测量。通过测距仪得到：



将大门的数据作为面积体积计算中的记准量，放入三维建模中进行计算：得到艺术学院与科学实验楼的占地面积约为：2218平方米（精确到平方米）。加上高度计算，其建筑体积约为：31717立方米。

得到结果后，我们根据第一节三维建模的点云成果进行检验，占地面积差距较为理想（100平方米以内）。如下是点坐标提取：



(2) OpenCV识别三维重建后的模型窗户

首先介绍一下本节的思路：

1. 图像采集(取到图像)：可以用摄像头拍摄或者图片直接导入

2. 图像预处理：对图像进行灰度化
3. 基于灰度的阈值分割：使用局部大津算法进行阈值分割二值化，形态学去噪
4. 图像特征描述及目标分析：使用灰度直方图计算轮廓，并通过面积条件进行筛选
5. 得到最终结果：统计识别的米粒个数，并将米粒用矩形框标记出来，并打上编号

STEP 1 局部大津算法

大津法（Otsu）是图像处理领域里面较为重要的阈值分割方法(局部大津法)适用于处理双峰图像。但大多数开发人员并不熟悉其原理，因此有必要对其进行详细说明与分析。

全自动的全局阈值算法通常由下述几步组成：

1. 预处理输入图像。
2. 获取图像直方图（像素分布）。
3. 计算阈值 T 。
4. 将图像中像素大于 T 的区域替换成白色，其余部分替换成黑色。

可以看到，最主要的步伐在第三步，因此，不同的算法在第三步中有很大的不同。大津法在处理图像直方图后，通过最小化每个类上的方差来分割目标。通常，这种技术为双峰图像产生适当的结果。该类图像的直方图中有两个表达清晰的峰值，代表不同强度值的范围。其核心思想是将图像直方图划分为两个类，并定义一个阈值，使这些类的加权方差最小。该加权方差的计算公式为：

$$\sigma_w^2(t) = w_1(t)\sigma_1^2(t) + w_2(t)\sigma_2^2(t)$$

其中，两个权值是两个类别的概率除以阈值 t ，该阈值在[0, 255]范围内取值，计算公式如下：

$$w_1(t) = \sum_{i=1}^t P(i)$$

$$w_2(t) = \sum_{i=t+1}^I P(i)$$

其中， P 的计算公式如下：

$$P(i) = \frac{n_i}{n}$$

其中， n 为像素的强度。接着，我们要求公式中的两个方差，这两个方差的计算公式如下：

$$\sigma_1^2(t) = \sum_{i=1}^t [i - \mu_1(t)]^2 \frac{P(i)}{w_1(t)}$$

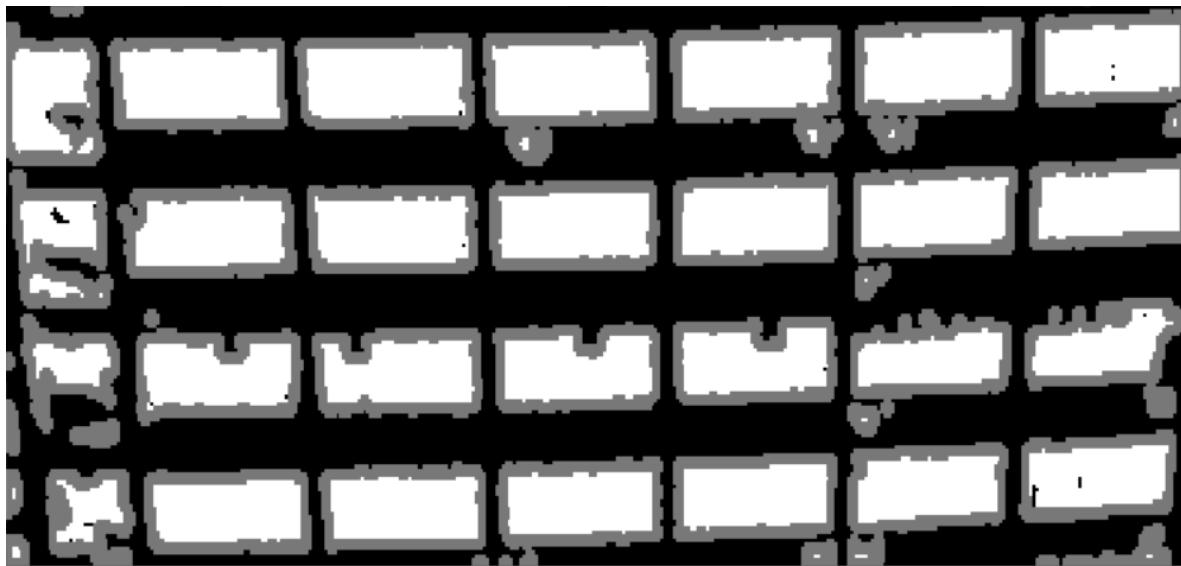
$$\sigma_2^2(t) = \sum_{i=t+1}^I [i - \mu_2(t)]^2 \frac{P(i)}{w_2(t)}$$

可以看到，如果我们想要求得两个方差，我们需要先获得 μ 的值：

$$\mu_1(t) = \sum_{i=1}^t \frac{iP(i)}{w_1(t)}$$

$$\mu_2(t) = \sum_{i=t+1}^I \frac{iP(i)}{w_2(t)}$$

自此，我们就可以得到了所需的加权方差。以下为局部大津算法下的效果图：



STEP 2 图像特征描述及目标分析

检测窗户的目的在于，将照片转化为灰度图，经过处理尽量使窗户全部为白色，其他全部为深色。

主要步骤为：

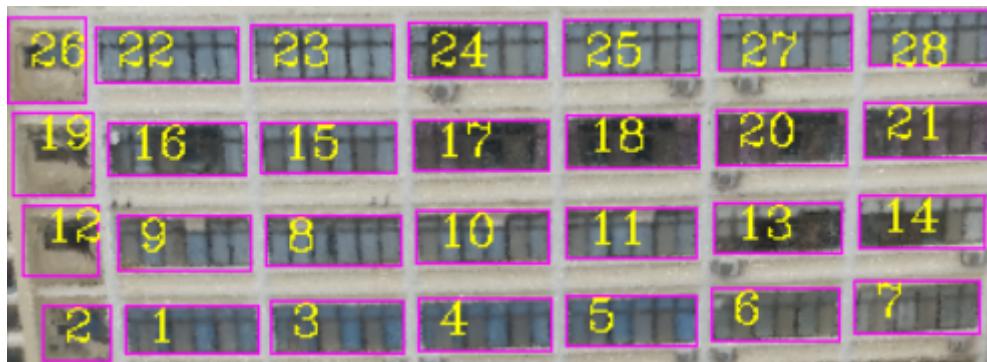
- 1、检测轮廓- cv2.findContours() 函数
- 2、提取轮廓的水平矩形坐标- rect = cv2.boundingRect() 函数
- 3、绘制矩形- cv2.rectangle()函数
- 4、在米粒左上角写上编号-cv2.putText() 函数

具体重要代码如下：

```
for cont in contours:  
    ares = cv2.contourArea(cont)#计算包围性状的面积  
    if ares<80:    #过滤面积小于10的形状  
        continue  
    count+=1      #总体计数加1  
    ares_avrg+=ares  
    print("{}-window:{}".format(count,ares),end=" ") #打印出每个窗户的面积  
    rect = cv2.boundingRect(cont) #提取矩形坐标  
    print("x:{} y:{}".format(rect[0],rect[1]))#打印坐标  
    cv2.rectangle(img,rect,(255,0,255),1)#绘制矩形  
    y=10 if rect[1]<10 else rect[1] #防止编号到图片之外  
    cv2.putText(img,str(count), (rect[0]+8, y+14), cv2.FONT_HERSHEY_COMPLEX,  
    0.6, (0, 255, 255), 1) #在窗户左上角写上编号
```

首先，对图像进行形态学去噪，形态学去噪有一下几种：1.先腐蚀再膨胀，用来消除小物体 2.先膨胀再腐蚀，用于排除小型黑洞 3.就是膨胀图与俯视图之差，用于保留物体的边缘轮廓。4.原图像与开运算图之差，用于分离比邻近点亮一些的斑块。5.闭运算与原图像之差，用于分离比邻近点暗一些的斑块。

其次检测轮廓，寻找轮廓的图像。再通过cv2.findContours()函数返回两个值，一个是轮廓本身，还有一个是每条轮廓对应的属性。进一步，提取轮廓的水平矩形坐标再据此绘制矩形并进行编号：



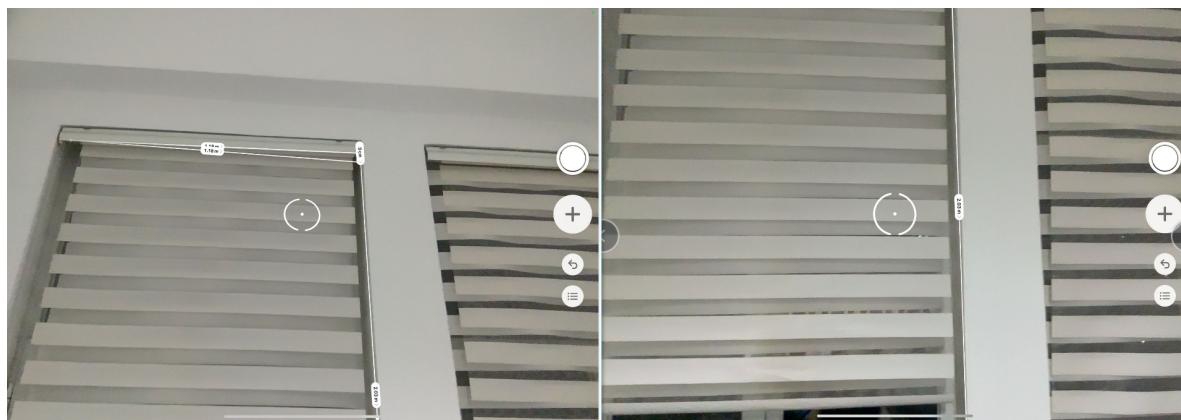
可以清晰分辨，该照片下共有窗户28个。

STEP 3 窗户面积计算

通过照片像素，提取识别出的窗户面积，最后输出面积的总和：

```
C:\WINDOWS\SYST... - X
6-window:891.5 x:281 y:113
7-window:858.5 x:338 y:110
8-window:933.0 x:103 y:84
9-window:938.0 x:44 y:84
10-window:926.0 x:163 y:82
11-window:903.0 x:223 y:81
12-window:378.0 x:6 y:80
13-window:763.5 x:282 y:79
14-window:716.0 x:340 y:76
15-window:1027.0 x:101 y:47
16-window:1023.0 x:40 y:47
17-window:1027.5 x:162 y:46
18-window:1017.5 x:223 y:44
19-window:618.5 x:2 y:43
20-window:995.0 x:283 y:42
21-window:945.5 x:342 y:39
22-window:1133.0 x:35 y:9
23-window:1135.0 x:97 y:8
24-window:1121.5 x:160 y:7
25-window:1082.5 x:222 y:6
26-window:781.0 x:0 y:5
27-window:1053.5 x:284 y:4
28-window:960.0 x:344 y:2
窗户平均面积:8461.5
```

根据测距仪，得到窗户长宽数据：



可以得到，窗户框约1.2米，长约2米。即一扇窗户的面积为2.4平方米。

结合上一步对照片中窗户的识别，计算得到照片中窗户总面积为67.2平方米。再通过所有照片中照下的所有窗户，计算闻天楼所有窗户面积（计算过程中对特殊面积的窗户进行单独计算）为1713.6平方米，其中闻天楼正面左上角的大型玻璃面积为：57.6平方米。（窗户总数为691扇，包括一个大型玻璃）

三、针对楼宇外围树木、垃圾箱等物的目标检测

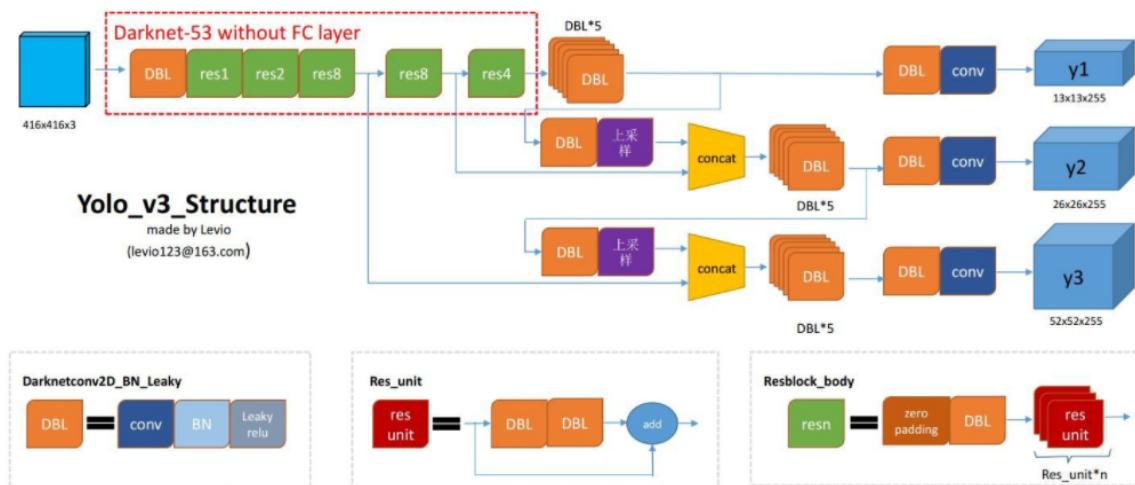
本章目的在于识别并计算楼宇外围物体的数目，比如树木、垃圾箱、灯杆、空调外挂机等等，识别的种类越多越好。根据识别物体在照片中的大小。本章在yolo和SSD两个算法上分别进行了目标检测。

(1) yolo算法下的目标检测

本文采用目前较为前言和完备的yolo_v3算法，其对之前的算法既有保留又有改进。先分析一下yolo_v3上保留的东西：

1. “分而治之”，从yolo_v1开始，yolo算法就是通过划分单元格来做检测，只是划分的数量不一样。
2. 采用“leaky ReLU”作为激活函数。
3. 端到端进行训练。一个loss function搞定训练，只需关注输入端和输出端。
4. 从yolo_v2开始，yolo就用batch normalization作为正则化、加速收敛和避免过拟合的方法，把BN层和leaky relu层接到每一层卷积层之后。
5. 多尺度训练。在速度和准确率之间tradeoff。想速度快点，可以牺牲准确率；想准确率高点儿，可以牺牲一点速度。

其整体框架如下：



1) backbone

整个v3结构里面，是没有池化层和全连接层的。前向传播过程中，张量的尺寸变换是通过改变卷积核的步长来实现的，比如stride=(2, 2)，这就等于将图像边长缩小了一半(即面积缩小到原来的1/4)。
backbone都会将输出特征图缩小到输入的1/32。所以，通常都要求输入图片是32的倍数。

下面是该算法的blackbone：

Yolo v3 base net

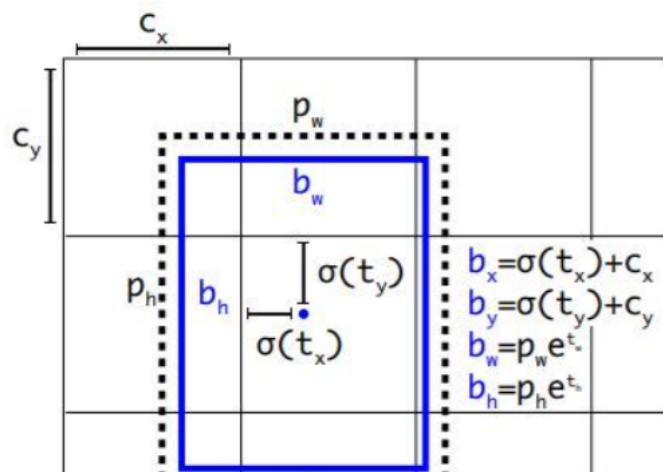
Type	Filters	Size	Output
Convolutional	32	3×3	256×256
Convolutional	64	$3 \times 3 / 2$	128×128
1x	32	1×1	
Convolutional	64	3×3	
Residual			128×128
Convolutional	128	$3 \times 3 / 2$	64×64
Convolutional	64	1×1	
2x	128	3×3	
Residual			64×64
Convolutional	256	$3 \times 3 / 2$	32×32
Convolutional	128	1×1	
8x	256	3×3	
Residual			32×32
Convolutional	512	$3 \times 3 / 2$	16×16
Convolutional	256	1×1	
8x	512	3×3	
Residual			16×16
Convolutional	1024	$3 \times 3 / 2$	8×8
Convolutional	512	1×1	
4x	1024	3×3	
Residual			8×8
Avgpool			Global
Connected			1000
Softmax			

2) Bounding Box Prediction

b-box预测手段是v3论文中提到的又一个亮点。其通过**直接预测相对位置**，预测出b-box中心点相对于网格单元左上角的相对坐标。

$$\begin{aligned}
 b_x &= \sigma(t_x) + c_x \\
 b_y &= \sigma(t_y) + c_y \\
 b_w &= p_w e^{t_w} \\
 b_h &= p_h e^{t_h} \\
 \Pr(\text{object}) * \text{IOU}(b, \text{object}) &= \sigma(t_o)
 \end{aligned}$$

大致过程如下图展示：



3) loss function

我们知道，在目标检测任务里，有几个关键信息是需要确定的：

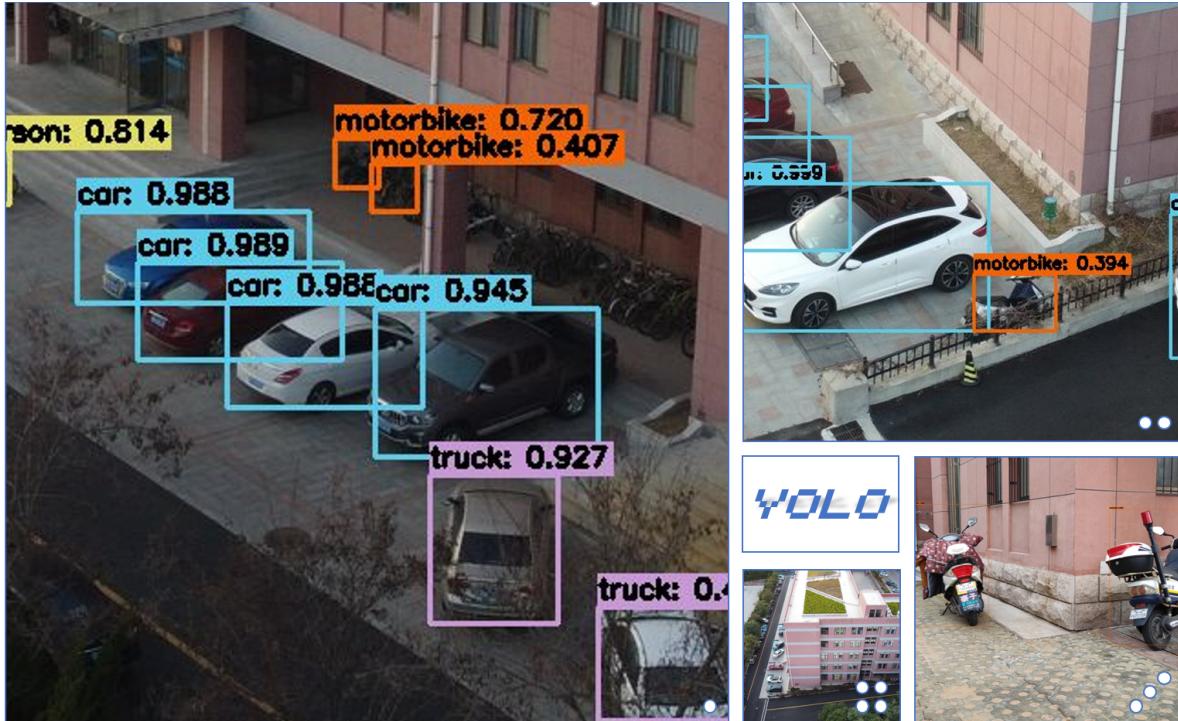
$$(x, y), (w, h), \text{class}, \text{confidence}$$

根据关键信息的特点可以分为上述四类，损失函数应该由各自特点确定。最后加到一起就可以组成最终的loss_function了，也就是一个loss_function搞定端到端的训练。

在实际运算过程中，模型的所有代码为pytorch版本，且在矩池云的GPU服务器上训练测试，所以代码中路径的配置均以矩池云中的路径配置为标准（/mnt），这里的文件夹中包含用官方COCO数据集训练完的完整的网络数据。详情见使用说明（另附）。

COCO数据集提供了八十种物体种类的区分数据，用此数据集训练好的yolo模型在闻天楼和艺术学院楼内可以识别八类专有种类标签：person , bicycle , car , motorbike , traffic light , fire hydrant , stop sigh , bench。

其实际算法部分效果展示如下：



因为还有一些物体需要识别，例如空调外机、井盖、捕虫器、路障、垃圾桶等等，所以需要自己拍摄图片进行标注和训练，考虑到SSD需要的xml文件可以直接由LabelImg软件标注生成，所以后面用SSD进行单独数据集的训练。

为了识别教学楼外的其他特征标志物，比如捕蚊器。我们需要自主拍摄并标注训练集。但是yolo算法下对图片标注要求较复杂。所以我们寻找了不影响模型准确度的近似算法。如下我们将常见的几种目标检测算法在使用两种权威数据集下的准确度以及运算时间进行比较：

VOC2007	YOLO_v2	YOLO_v3	SSD	RFB	FSSD
Darknet53		<u>79.3%</u>	<u>77.3%</u>	<u>79.5%</u>	<u>81.0%</u>
Darknet19	<u>78.4%</u>		<u>76.1%</u>	<u>78.4%</u>	<u>81.0%</u>
Resnet50			<u>79.7%</u>	<u>81.2%</u>	
VGG16			<u>76.0%</u>	<u>80.5%</u>	<u>77.8%</u>
MobilenetV1	<u>74.7%</u>	<u>78.2%</u>	<u>72.7%</u>	<u>73.7%</u>	<u>78.4%</u>
MobilenetV2	<u>72.0%</u>	<u>75.8%</u>	<u>73.2%</u>	<u>73.4%</u>	<u>76.7%</u>

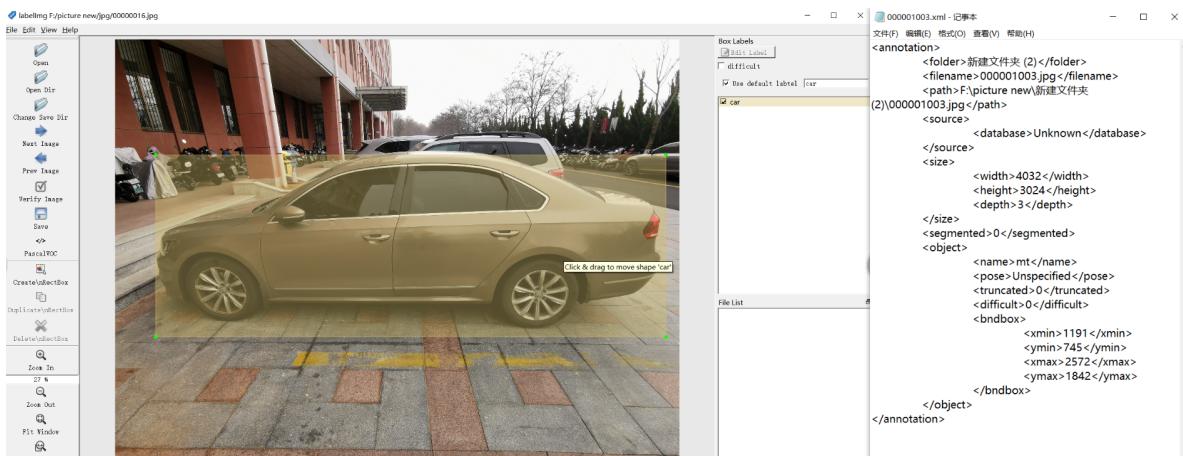
COCO2017	YOLO_v2	YOLO_v3	SSD	RFB	FSSD
Darknet53		<u>27.3%</u>	<u>21.1%</u>	<u>22.8%</u>	<u>25.8%</u>
Darknet19	<u>21.6%</u>		<u>20.6%</u>	<u>22.4%</u>	<u>26.4%</u>
Resnet50			<u>25.1%</u>	<u>26.5%</u>	<u>27.2%</u>
VGG16			<u>25.4%</u>	<u>25.5%</u>	<u>27.2%</u>
MobilenetV1	<u>21.5%</u>	<u>25.7%</u>	<u>18.8%</u>	<u>19.1%</u>	<u>24.2%</u>
MobilenetV2	<u>20.4%</u>	<u>24.0%</u>	<u>18.5%</u>	<u>18.5%</u>	<u>22.2%</u>

Net InferTime* (fp32)	YOLO_v2	YOLO_v3	SSD	RFB	FSSD
Darknet53		5.11ms	4.32ms	6.63ms	4.41ms
Darknet19	1.64ms		2.21ms	4.57ms	2.29ms
Resnet50			3.60ms	6.04ms	3.85ms
VGG16			1.75ms	4.20ms	1.98ms
MobilenetV1	2.02ms	3.31ms	2.80ms	3.84ms	2.62ms
MobilenetV2	3.35ms	4.69ms	4.05ms	5.26ms	4.00ms

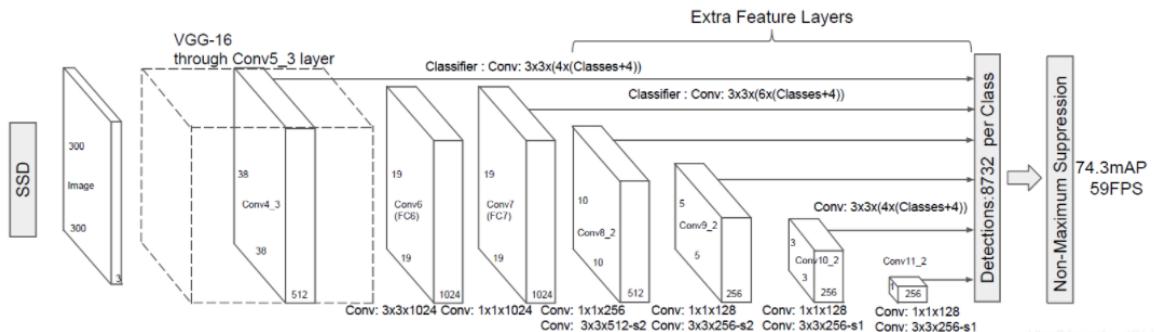
可见，在多个网络模型下，SSD和yolo算法的准确度相差不大，但SSD算法的耗时较小。可见使用SSD算法有一定的意义。并且我们发现SSD算法可以通过软件直接生成训练集，所以我们采用SSD算法尝试训练自己的网络模型。

(2) SSD算法下的目标检测

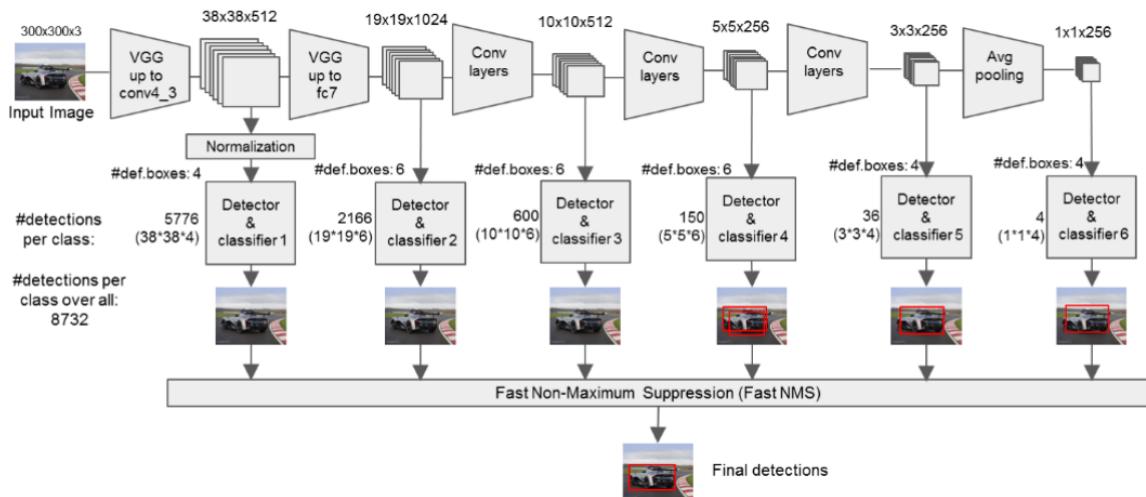
为了更好地、更有针对性地识别目标，我们对校园内的各个目标检测物进行拍摄并通过labelling生成训练集。如下是labelling标记过程及效果：



下面介绍一下SSD算法，其整体框架是：



以下是样例展示：



具体落实步骤如下：

1. 输入一幅图片 (300x300)，将其输入到预训练好的分类网络中来获得不同大小的特征映射，修改了传统的VGG16网络；
2. 将VGG16的FC6和FC7层转化为卷积层，如图1上的Conv6和Conv7；去掉所有的Dropout层和FC8层；添加了Atrous算法 (hole算法) ；将Pool5从2x2-S2变换到3x3-S1；
3. 抽取Conv4_3、Conv7、Conv8_2、Conv9_2、Conv10_2、Conv11_2层的feature map，然后分别在这些feature map层上面的每一个点构造6个不同尺度大小的bbox，然后分别进行检测和分类，生成多个bbox；
4. 将不同feature map获得的bbox结合起来，经过NMS (非极大值抑制) 方法来抑制掉一部分重叠或者不正确的bbox，生成最终的bbox集合 (即检测结果)

1) 多尺度特征映射

SSD算法中使用到了conv4_3,conv_7, conv8_2,conv7_2,conv8_2,conv9_2,conv10_2,conv11_2这些大小不同的feature maps，其目的是为了能够准确的检测到不同尺度的物体，因为在低层的feature map，感受野比较小，高层的感受野比较大，在不同的feature map进行卷积，可以达到多尺度的目的。

我们将一张图片输入到一个卷积神经网络中，经历了多个卷积层和池化层，我们可以看到在不同的卷积层会输出不同大小的feature map（这是由于pooling层的存在，它会将图片的尺寸变小），而且不同的feature map中含有不同的特征，而不同的特征可能对我们的检测有不同的作用。总的来说，浅层卷积层对边缘更加感兴趣，可以获得一些细节信息，而深层网络对由浅层特征构成的复杂特征更感兴趣，可以获得一些语义信息，对于检测任务而言，一幅图像中的目标有复杂的有简单的，对于简单的patch我们利用浅层网络的特征就可以将其检测出来，对于复杂的patch我们利用深层网络的特征就可以将其检测出来，因此，如果我们同时在不同的feature map上面进行目标检测，理论上面应该会获得更好的检测效果。

2) Default box

在特征图的每个位置预测K个bbox，对于每一个bbox，预测C个类别得分，以及相对于Default box的4个偏移量值，这样总共需要 $(C+4) \times K$ 个预测器，则在 $m \times n$ 的feature map上面将会产生 $(C+4) \times K \times m \times n$ 个预测值。

SSD中的Default box和Faster-rcnn中的anchor机制很相似。就是预设一些目标预选框，后续通过softmax分类+bounding box regression获得真实目标的位置。对于不同尺度的feature map 上使用不同的Default boxes。如上图所示，我们选取的feature map包括38x38x512、19x19x1024、10x10x512、5x5x256、3x3x256、1x1x256，Conv4_3之后的feature map默认的box是4个，我们在38x38的这个平面上的每一点上面获得4个box，那么我们总共可以获得38x38x4=5776个；同理，我们依次将FC7、Conv8_2、Conv9_2、Conv10_2和Conv11_2的box数量设置为6、6、6、4、4，那么我们可以获得的box分别为2166、600、150、36、4，即我们总共可以获得8732个box，然后我们将这些box送入NMS模块中，获得最终的检测结果。

Default box生成规则：

以feature map上每个点的中点为中心 ($offset=0.5$)，生成一系列同心的Default box（然后中心点的坐标会乘以step，相当于从feature map位置映射回原图位置）

使用m(SSD300中m=6)个不同大小的feature map 来做预测，最底层的 feature map 的 scale 值为 $S_{min}=0.2$ ，最高层的为 $S_{max}=0.95$ ，其他层通过下面的公式计算得到

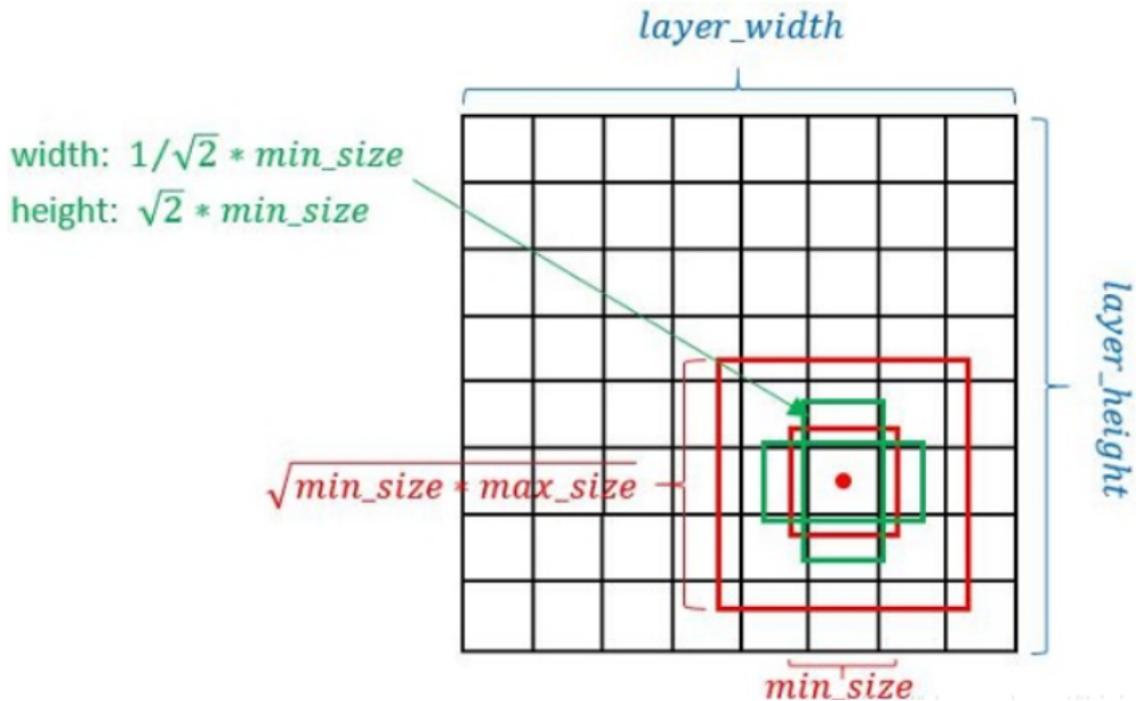
$$s_k = s_{min} + \frac{s_{max} - s_{min}}{m - 1}(k - 1), \quad k \in [1, m]$$

使用不同的ratio值， $[1, 2, 3, 1/2, 1/3]$ ，通过下面的公式计算 default box 的宽度w和高度h：

$$\text{width } (w_k^a = s_k \sqrt{a_r}) \text{ and height } (h_k^a = s_k / \sqrt{a_r})$$

而对于ratio=0的情况，指定的scale如下所示，即总共有 6 中不同的 default box：

$$s'_k = \sqrt{s_k s_{k+1}}$$



3) Loss

loss函数分为两部分：计算相应的default box与目标类别的confidence loss以及相应的位置回归：

$$L(x, c, l, g) = \frac{1}{N} (L_{\text{conf}}(x, c) + \alpha L_{\text{loc}}(x, l, g))$$

其中N是match到Ground Truth的default box数量；而alpha参数用于调整confidence loss和location loss之间的比例，默认alpha=1。

位置回归则是采用 **Smooth L1 loss**，loss函数为：

$$\begin{aligned} L_{\text{loc}}(x, l, g) &= \sum_{i \in \text{Pos}}_{m \in \{cx, cy, w, h\}} x_{ij}^k \text{smoothL1}(l_i^m - \hat{g}_j^m) \\ \hat{g}_j^{cx} &= \left(g_j^{cx} - d_i^{cx} \right) / d_i^w \\ \hat{g}_j^w &= \log \left(\frac{g_j^w}{d_i^w} \right) \hat{g}_j^h = \log \left(\frac{g_j^{cy}}{d_i^h} \right) \end{aligned}$$

confidence loss是典型的**softmax loss**：

$$L_{\text{conf}}(x, c) = - \sum_{i \in \text{Pos}}^N x_{ij}^p \log(\hat{c}_i^p) - \sum_{i \in \text{Neg}} \log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

我们尝试识别以下十种物体：空调外机，车，路障，电动车，自行车，井盖，消防栓，捕蝇器，树，垃圾桶。 ('ac','car','rb','ev','bike','mc','fh','mt','tree','tc')

其实际算法部分效果如下：



分析结果发现：

- 相较于COCO数据集，因为拍摄的图片中首先每类物体样式少，其次像素较专业的摄像机针对性不强，每张图片中的噪点较多，对于官方的SSD算法训练难度大，最后又因为标注标准不如COCO数据集专业，导致模型在ac, fh等物体识别中效果较差。同时，物体拍摄不清晰或像素较小时，识别效果较差。



- 因为学校中的“树”差距较大，缺乏像COCO数据集那样大而全面的数据图片集，导致训练过程中SSD算法未能找到优良的特征；且大多时候树连成一片，对标注工作造成了很大的困扰，极大地受到了噪音的影响。从而影响了模型训练，所以对tree的识别正确率低。
- 在学校拍摄自行车和电动车时因为两者现实中差距较小，外加上种类繁多，且多扎堆放置，甚至部分车增置了防风罩，导致ev和bike的识别正确率低。相较于上节的yolo模型，识别效果相比差距大。

针对上述问题总结如下，训练的网络因下列几种原因导致效果很差：

- 物体本身种类较多，且受拍摄数量影响，无法根据有限的图片拾取有用的特征信息。
- 受拍摄环境、拍摄设备等因素影响，所建立的数据集图片质量较COCO数据集差距较大。
- 所拍图片中背景中噪音太大，因为标注时是用矩形框选图片，往往会导致很多背景被网络认为是物体的一部分，从而导致训练偏差增大。这也是数据集建立中最大的问题。

改进方向：

下一步可以在权威的数据集网站中查找数据集，而不是用自己所拍摄的图片，过程中要尽量寻找多而优的照片，尽量寻找背景噪音小的图片进行标注，尽可能减少模型训练中不必要的误差，尽量最终数据集效果与COCO数据集类似，使模型最后训练效果更好。

参考资料

- 1.colmap相关资料：<https://colmap.github.io/cameras.html>
- 2.EXIF相关资料：<https://blog.csdn.net/libins/article/details/50973498>
- 3.openMVG相关资料：<https://github.com/openMVG/openMVG>
- 4.mesh重建：<https://www.cnblogs.com/luyb/p/5730932.html>
- 5.Delaunay网格：<https://people.eecs.berkeley.edu/~jrs/meshbook.html>
- 6.CasMVSNet：https://github.com/kwea123/CasMVSNet_pl
- 7.对照片中窗户的识别：https://gitee.com/huangjiezhou/xiaohuang_code_warehouse/tree/master/opencv%E6%A3%80%E6%B5%8B%E7%B1%B3%E7%B2%92
- 8.yolo算法：https://blog.csdn.net/leviopku/article/details/82660381?ops_request_misc=%7B%22request%5Fid%22%3A%22163971116116780269840928%22%2C%22scm%22%3A%2220140713.130102334..%22%7D&request_id=163971116116780269840928&biz_id=0&utm_medium=distribut_e.pc_search_result.none-task-blog-2~all~top_positive~default-3-82660381.pc_search_result_contr ol_group&utm_term=yolo&spm=1018.2226.3001.4187
- 9.SSD算法：<https://arxiv.org/abs/1512.02325>