毕设所用的算法

一、多相机标定：

标定球算法原理：

（1）特征点提取-棋盘格型角点

针对类似黑白直角棋盘格点，此方法比 Harris points 和 Shi-Tomasi points 更鲁棒。

使用下列两类滤波模板，一为直角，一为 45° 角，每类有 A,B,C,D 四种模板与图像进行卷积，滤波器如下图 2-3 所示。

某一像素点是否角点候选点的可能性值 c 是所有滤波模板响应组合的最大值，定义如下:

$$
\begin{aligned}
&\mathrm{c} = \max(S_1^1, S_2^1, S_1^2, S_2^2) \\
&S_1^i = \min(\min(f_A^i, f_B^i) - u, u - \min(f_C^i, f_D^i)) \\
&S_2^i = \min(u - \min(f_A^i, f_B^i), \min(f_C^i, f_D^i) - u) \\
&u = 0.25(f_A^i + f_B^i + f_C^i + f_D^i)
\end{aligned}
\tag{2-1}
$$

公式有错，应为：$S_1^i=\min(\min(f_A^i-u, f_B^i-u), \min(u-f_C^i, u-f_D^i))$,

$S_2^i=\min(\min(u-f_A^i, u-f_B^i),\min(f_C^i-u, f_D^i-u))$

其中，$S^i_{1,2}$ 是第 i 套模板中两种滤波器的可能性值;

f^i_X 是某一像素对于第 i 套模板的核心 X 的滤波响应;

此方法优点，四种滤波模板，任一种滤波模板的响应不理想，都会导致角点候选点可能性值 C 小，从而能去除很多不是棋盘格点形状的角点。

对候选点进行非极大值抑制后，使用候选点局部梯度统计量进行验证。

梯度统计量描述如下：

使用图像的 Sobel 算子响应计算图像局部梯度强度 score_gradient;

兴趣点候选分数score = score_gradient*C;

利用对称几何结构删除误检点；

特征点亚像素值修正cvFindCornerSubPix()。

%MATLAB 程序

```
% filter image according with current template
   img_corners_a1 = conv2(img,template.a1,'same');
   img_corners_a2 = conv2(img,template.a2,'same');
   img_corners_b1 = conv2(img,template.b1,'same');
   img_corners_b2 = conv2(img,template.b2,'same');
   % compute mean
   img_corners_mu = (img_corners_a1+img_corners_a2+img_corners_b1+img_corners_b2)/4;
```

```
% case 1: a=white, b=black
img_corners_a = min(img_corners_a1-img_corners_mu,img_corners_a2-img_corners_mu);
img_corners_b = min(img_corners_mu-img_corners_b1,img_corners_mu-img_corners_b2);
img_corners_1 = min(img_corners_a,img_corners_b);
% case 2: b=white, a=black
img_corners_a = min(img_corners_mu-img_corners_a1,img_corners_mu-img_corners_a2);
img_corners_b = min(img_corners_b1-img_corners_mu,img_corners_b2-img_corners_mu);
img_corners_2 = min(img_corners_a,img_corners_b);
% update corner map
%img_cornersL = max(img_cornersL,img_corners_1);
img_cornersU = max(img_cornersU,img_corners_1);
img_cornersU = max(img_cornersU,img_corners_2);
```

相关常用算法

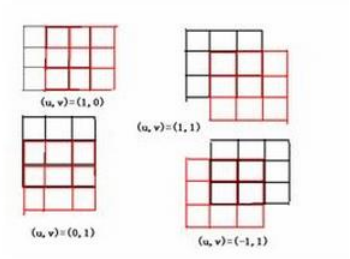
1、 Harris 角点检测

检测原理

Harris角点检测算子是于1988年由CHris Harris & Mike Stephens提出来的。在具体展开之前，不得不提一下Moravec早在1981就提出来的Moravec角点检测算子。

1.Moravec角点检测算子

Moravec角点检测算子的思想其实特别简单，在图像上取一个W*W的“滑动窗口”，不断的移动这个窗口并检测窗口中的像素变化情况E。像素变化情况E可简单分为以下三种：A 如果在窗口中的图像是什么平坦的，那么E的变化不大。B 如果在窗口中的图像是一条边，那么在沿这条边滑动时E变化不大，而在沿垂直于这条边的方向滑动窗口时，E的变化会很大。 C 如果在窗口中的图像是一个角点时，窗口沿任何方向移动E的值都会发生很大变化。



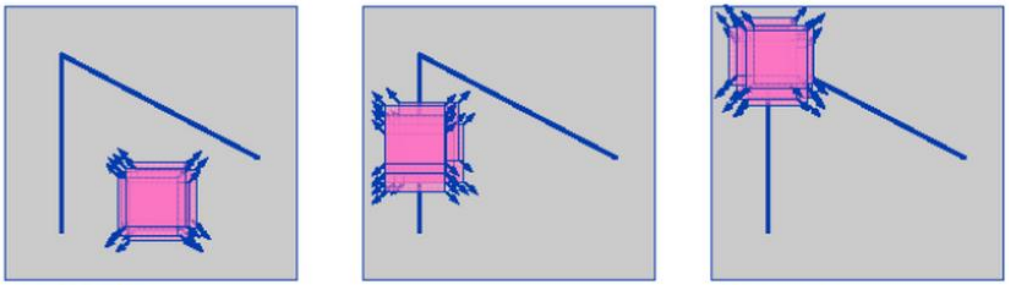
上图就是对Moravec算子的形象描述。用数学语言来表示的话就是：

$$E_{x,y} = \sum_{u,v} w_{u,v} \mid I_{x+u,y+v} - I_{u,v} \mid^2$$

其中（x，y）就表示四个移动方向（1，0）（1，1）（0，1）（-1，1），E就是像素的变化值。Moravec算子对四个方向进行加权求和来确定变化的大小，然和设定阈值，来确定到底是边还是角点。

基本原理：

角点是一幅图像上最明显与重要的特征，对于一阶导数而言，角点在各个方向的变化是最大的，而边缘区域在只是某一方向有明显变化。一个直观的图示如下：



平坦区域
在所有方向没有
明显梯度变化

边缘区域
在某个方向有明显
梯度变化

角度边缘
在各个方向梯度值
有明显变化

数学原理：

基本数学公式如下：

$$E(u,v)=\sum_{x,y} w(x,y)[I(x+u,y+v)-I(x,y)]^2$$

其中W(x, y)表示移动窗口，I(x, y)表示像素灰度值强度，范围为0～255。根据泰勒级数计算一阶到N阶的偏导数，最终得到一个Harris矩阵公式：

$$M=\sum_{x,y} w(x,y)\begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}$$

窗口功能，计算权重W(x，y)取值为1

图像每个像素点X方向的强度Ix，Y方向强度Iy得到的Harris矩阵

根据Harris的矩阵计算矩阵特征值λ_1, λ_2，然后计算Harris角点响应值：

$$R=\det M-k(trace\,M)^2$$

$$\det M=\lambda_1\lambda_2$$

$$trace\,M=\lambda_1+\lambda_2$$

其中K为系数值，通常取值范围为0.04 ~ 0.06之间。

harris 算法详细步骤

第一步：计算图像 X 方向与 Y 方向的一阶高斯偏导数 Ix 与 Iy
第二步：根据第一步结果得到 Ix^2 , Iy^2 与 Ix*Iy 值
第三步：高斯模糊第二步三个值得到 Sxx, Syy, Sxy
第四部：定义每个像素的 Harris 矩阵，计算出矩阵的两个特质值
第五步：计算出每个像素的 R 值
第六步：使用 3X3 或者 5X5 的窗口，实现非最大值压制

缺点：

A 它对尺度很敏感，不具备几何尺度不变性
B 提取的角点是像素级的

2、 goodFeaturesToTrack()+cornerSubPix()

一提到角点检测，最常用的方法莫过于Harris角点检测，OpenCV中也提供了Harris角点检测的接口，即cv::cornerHarris()，但是Harris角点检测存在很多缺陷（如角点是像素级别的，速度较慢等），因此我们这里将介绍opencv中的另一个功能更为强大的函数——cv::goodFeaturesToTrack()，它不仅支持Harris角点检测，也支持Shi Tomasi算法的角点检测。但是，该函数检测到的角点依然是像素级别的，若想获取更为精细的角点坐标，则需要调用cv::cornerSubPix()函数进一步细化处理，即亚像素。

1、cv::goodFeaturesToTrack()角点检测

cv::goodFeaturesToTrack()的具体调用形式如下：

```
[cpp] view plain  copy  print ?
01. void cv::goodFeaturesToTrack(
02.     cv::InputArray image, // 输入图像（CV_8UC1 CV_32FC1）
03.     cv::OutputArray corners, // 输出角点vector
04.     int maxCorners, // 最大角点数目
05.     double qualityLevel, // 质量水平系数（小于1.0的正数，一般在0.01-0.1之间）
06.     double minDistance, // 最小距离，小于此距离的点忽略
07.     cv::InputArray mask = noArray(), // mask=0的点忽略
08.     int blockSize = 3, // 使用的邻域数
09.     bool useHarrisDetector = false, // false ='Shi Tomasi metric'
10.     double k = 0.04 // Harris角点检测时使用
11. );
```

第一个参数是输入图像（8位或32位单通道图）。

第二个参数是检测到的所有角点，类型为vector或数组，由实际给定的参数类型而定。如果是vector，那么它应该是一个包含cv::Point2f的vector对象；如果类型是cv::Mat,那么它的每一行对应一个角点，点的x、y位置分别是两列。

第三个参数用于限定检测到的点数的最大值。

第四个参数表示检测到的角点的质量水平（通常是0.10到0.01之间的数值，不能大于1.0）。

第五个参数用于区分相邻两个角点的最小距离（小于这个距离得点将进行合并）。

第六个参数是mask，如果指定，它的维度必须和输入图像一致，且在mask值为0处不进行角点检测。

第七个参数是blockSize，表示在计算角点时参与运算的区域大小，常用值为3，但是如果图像的分辨率较高则可以考虑使用较大一点的值。

第八个参数用于指定角点检测的方法，如果是true则使用Harris角点检测，false则使用Shi Tomasi算法。

第九个参数是在使用Harris算法时使用，最好使用默认值0.04。

2、cv::cornerSubPix()亚像素角点检测

前面已经提及，cv::goodFeaturesToTrack()提取到的角点只能达到像素级别，在很多情况下并不能满足实际的需求，这时，我们则需要使用cv::cornerSubPix()对检测到的角点作进一步的优化计算，可使角点的精度达到亚像素级别。

具体调用形式如下：

```
[cpp] view plain  copy  print  ?
01. void cv::cornerSubPix(
02.     cv::InputArray image, // 输入图像
03.     cv::InputOutputArray corners, // 角点（既作为输入也作为输出）
04.     cv::Size winSize, // 区域大小为 NXN; N=(winSize*2+1)
05.     cv::Size zeroZone, // 类似于winSize，但是总具有较小的范围，Size(-1,-1)表示忽略
06.     cv::TermCriteria criteria // 停止优化的标准
07. );
```

第一个参数是输入图像，和cv::goodFeaturesToTrack()中的输入图像是同一个图像。

第二个参数是检测到的角点，即是输入也是输出。

第三个参数是计算亚像素角点时考虑的区域的大小，大小为NXN; N=(winSize*2+1)。

第四个参数作用类似于winSize，但是总是具有较小的范围，通常忽略（即Size(-1, -1)）。

第五个参数用于表示计算亚像素时停止迭代的标准，可选的值有cv::TermCriteria::MAX_ITER 、cv::TermCriteria::EPS（可以是两者其一，或两者均选），前者表示迭代次数达到了最大次数时停止，后者表示角点位置变化的最小值已经达到最小时停止迭代。二者均使用cv::TermCriteria()构造函数进行指定。

3、 SUSAN

（2）计算球心图像坐标-霍夫变换圆检测

Hough 变换（霍夫变换）主要用来识别已知的几何形状，最常见的比如直线、线段、圆形、椭圆、矩形等。如果要检测比较复杂的曲线图形，就需要利用广义霍夫变换。

霍夫变换的原理是根据参数空间的统计规律进行参数估计。

具体说来就是，将直角坐标系中的图形(x,y)变换到参数空间(k1,...,kn)，对直角坐标系中的每一个像素点，计算它在参数空间里的所有可能的参数向量。处理完所有像素点后，把出现次数（频率）最多的（一个或几个）参数向量的坐标作为参数代入直角坐标方程，即检测到的图形方程。

直线检测

1.图像二值化，待检测的线变为黑色，背景置为白色。既然是形状检测，这步是必不可少的。

2.

1）在图像中检测直线的问题，其实质是找到构成直线的所有的像素点。那么问题就是从找到直线，变成找到符合 y=kx+b 的所有(x,y)的点的问题。

2）将 y=kx+b 进行坐标系变换，由 x-y 坐标系变换到 k-b 坐标系，即：变成 b=-xk+y，这样表示为过点（k,b）的直线束。可以这样理解，在图像 x-y 坐标系中的一点 A(x,y)，对应到 k-b 坐标系下为一直线（此时，x 值，y 值可以理解为 k-b 坐标系下直线的斜率和截距）。

3）如果在 k-b 坐标系下两条直线相交，那意味着什么呢? 假设这两条相交的直线由 x-y 坐标系下的两个点 A(x1,y1)和 B(x2,y2)确定，那么这意味着，存在一组 k, b 值使得由 k, b 值确定的直线经过 A 点和 B 点。

4）推而广之，在 x-y 坐标系下的边缘点，A 点、B 点、C 点、D 点、E 点、F 点.....对应确定了 k-b 坐标系下的很多条直线，并且这些直线都相交到一点 O(k',b')呢？那就意味着，根据该交点 O(k',b')确定的直线 y=k'x+b'通过了 A 点、B 点、C 点、D 点、E 点、F 点....。

由第四步也许可以在 k-b 坐标系下得到很多个交点，那么在局部统计各个交点处相交的直线个数，相交的直线个数越多，说明由该交点确定的直线通过的边界点越多，就在原图中越有可能是真实存在的直线。统计过后会得到若干个局部最大值，也就得到了若干条直线。

实际在使用这一原理的时候，不是采用直线的斜率和截距公式，因为直线的斜率可能不存在。而是用类似极坐标的方式表示，ρ=xcosϑ+ysinϑ。

*假设直线的参数方程为 p=x*cosa+y*sina，对于直线上的某个点(x,y)来说，变换到参数空间的坐标就是(p,a)，而且这条直线上的所有点都对应于(p,a)。对于一个固定点(x,y)来说，经过它的直线系可以表示为 p=(x^2+y^2)^1/2*sin(a+b)，其中 tanb=x/y，对应参数空间里的一条正弦曲线。也就是说，图像中的一条直线对应参数空间的一点，图像中的一点对应参数空间的一条正弦曲线。*

*3.把参数空间分割为 n*m 个格子，得到参数矩阵，矩阵元(pi,aj)的初始值均为 0，用来对参数计数。计数值代表这个参数是最终结果的可能性，计数值越大，说明落在这条直线上的像素点越多，也就说明它越有可能是我们想找到的参数。p 的范围可以是[0,图像对角线长度]，a 的范围可以是[0,PI/2]（如果取左上角为原点的话），但要包含整个图像。*

4.按照栅格顺序扫描图像，遇到黑色像素就做如下操作：

pi 的 i 从 0 取到 n-1，对每一个 pi，把它和像素点的坐标(x,y)代入参数方程，计算得到相应的 ai，如果 ai 在定义域范围内（或者在图像内），将矩阵元(pi,ai)加一。

处理完所有像素后，如果想识别 d 条直线，就在参数矩阵中找到前 d 个数值最大的矩阵元，他们的坐标作为方程参数，在直角坐标系绘制出直线就可以了。

5、OpenCV 中提供了计算霍夫变换的库函数 HoughLines 和 HoughLinesP

检测圆形

圆形的一般性方程表示为$(x-a)^2+(y-b)^2=r^2$。那么就有三个自由度圆心坐标 a,b,和半径 r。这就意味着需要更多的计算量，而 OpenCV 中提供的 cvHoughCircle()函数里面可以设定半径 r 的取值范围，相当于有一个先验设定，在每一个 r 来说，在二维空间内寻找 a 和 b 就可以了，能够减少计算量。

具体步骤如下：

1）.对输入图像进行边缘检测，获取边界点，即前景点。

2）.假如图像中存在圆形，那么其轮廓必定属于前景点（此时请忽略边缘提取的准确性）。

3）.同霍夫变换检测直线一样，将圆形的一般性方程换一种方式表示，进行坐标变换。由 x-y 坐标系转换到 a-b 坐标系。写成如下形式$(a-x)^2+(b-y)^2=r^2$。那么 x-y 坐标系中圆形边界上的一点对应到 a-b 坐标系中即为一个圆。

4）.那 x-y 坐标系中一个圆形边界上有很多个点，对应到 a-b 坐标系中就会有很多个圆。由于原图像中这些点都在同一个圆形上，那么转换后 a,b 必定也满足 a-b 坐标系下的所有圆形的方程式。直观表现为这许多点对应的圆都会相交于一个点，那么这个交点就可能是圆心(a,b)。

5）.统计局部交点处圆的个数，取每一个局部最大值，就可以获得原图像中对应的圆形的圆心坐标(a,b)。一旦在某一个 r 下面检测到圆，那么 r 的值也就随之确定。

霍夫变换的一个好处就是不需要图像中出现完整的圆，只要落在一个圆上的像素数量足够多，就能正确识别。

使用霍夫变换检测圆与直线时候,一定要对图像进行预处理,灰度化以后,提取图像的边缘使用非最大信号压制得到一个像素宽的边缘, 这个步骤对霍夫变换非常重要.否则可能导致霍夫变换检测的严重失真.

相关常用算法

基于 RANSAC 的圆检测

RANSAC(Random Sample Consensus)随机抽样一致性，略不同于霍夫圆变换那种基于投票的策略，这是一种对观测数据进行最大化模型检验的方法。下面来简单介绍一下它的原理：

最小二乘法通常用在线性拟合参数中，但一旦最小二乘法输入的观测数据中包含有大量分散的干扰点时，它拟合出来的效果可能并不好。Ransac 的思路是随机通过几个点用最小二乘法给出一个假设的直线，然后计算在直线内的 inliers 和在直线范围外的 outliers。对所有可能的直线中找出 inliers 数目最多的那个，也就能找到最好的直线。

RANSAC 几点要注意的：

① 只有 outliers%<50%时，得到的模型才是有保证的。

②inliers 的阈值 δ 跟我们期望的模型抗噪能力相关，阈值越大，抗噪能力越弱，通常我们选取 3 个像素偏差的高斯模型作为噪声模型。

③ 重复 1-3 步骤的次数跟模型的 outliers 占比和我们所需要多高的置信度有关，可以用下面公式表示：

$$S=\log(1-P)\log(1-pk)$$

其中，S 是所需最小试验的次数，P 是置信度，p 是 inliers 占的百分比数，k 是随机采样的数目。

RANSAC 圆检测思路是：

1. 用 Canny 提取边缘点， 用 distanceTransform 得到距离边缘点的距离图；
2. 随机抽取三个不同的点解方程，三个方程三个未知数，有解；
3. 将 2 得到的圆周上的点与 1 中对应位置的点进行比较，看是否属于 inliers，随后输出百分比；
4. 找出最大百分比对应的圆就是 RANSAC 得到的圆。

5、比较霍夫变换跟 RANSAC：鲁棒性来说，霍夫变换要稳定一点；
速度来说，霍夫变换要快，而且其所需时间变化不大，100ms 左右能够完成；
RANSAC 跟 HoughTranform 的参数调节都很麻烦，相对来说，霍夫变换更加简单一点；
RANSAC 拟合的程度可能会更高，但受到 outliers%<50%这个条件限制。
所以综合来说，HoughTransform 的应用更广，效率更高，某些情况下，它不能很好地找到合理的圆，这时可以将 RANSAC 加进去进行优化，可能精度会高很多。

（3）基于 RANSAC 思想的同心球面点匹配（即求出旋转矩阵 R）

RANSAC 算法的输入是一组观测数据（往往含有较大的噪声或无效点），一个用于解释观测数据的参数化模型以及一些可信的参数。RANSAC 通过反复选择数据中的一组随机子集来达成目标。被选取的子集被假设为局内点，并用下述方法进行验证： 有一个模型适应于假设的局内点，即所有的未知参数都能从假设的局内点计算得出。

- 用 1 中得到的模型去测试所有的其它数据，如果某个点适用于估计的模型，认为它也是局内点。
- 如果有足够多的点被归类为假设的局内点，那么估计的模型就足够合理。
- 然后，用所有假设的局内点去重新估计模型（譬如使用最小二乘法），因为它仅仅被初始的假设局内点估计过。
- 最后，通过估计局内点与模型的错误率来评估模型。
- 上述过程被重复执行固定的次数，每次产生的模型要么因为局内点太少而被舍弃，要么因为比现有的模型更好而被选用。

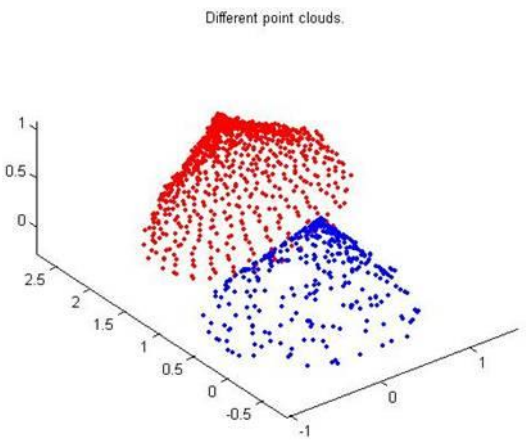
1、 分别从两幅图像（图 1、图 2）中随机取两组点对（即四个点），计算旋转矩阵 R0，将图 2 的点经过 R0 进行变换后，在图 1 搜索匹配点（最小 x,y 坐标差值小于阈值视为匹配成功），计数匹配成功点的数目，该数目大于阈值的则表明 R0 正确；利用成功匹配的点再次计算旋转矩阵 R；

相关常用算法

（一） ICP 算法（Iterative Closest Point 迭代最近点）

ICP（Iterative Closest Point 迭代最近点）算法是一种点集对点集配准方法，如下图 1

如下图，假设 PR（红色块）和 RB（蓝色块）是两个点集，该算法就是计算怎么把 PB 平移旋转，使 PB 和 PR 尽量重叠，建立模型的



（图 1）

ICP 是改进自**对应点集配准算法**的。对应点集配准算法是假设一个理想状况，将一个模型点云数据 X（如上图的 PB）利用四元数$\overrightarrow{q_R} = [q_0 q_1 q_2 q_3]^t$旋转，并平移$\overrightarrow{q_T} = [q_4 q_5 q_6]^t$得到点云 P（类似于上图的 PR）。而对应点集配准算法主要就是怎么计算出 qR 和 qT 的，知道这两个就可以匹配点云了。

但是对应点集配准算法的前提条件是计算中的点云数据 PB 和 PR 的元素一一对应，这个条件在现实里因误差等问题，不太可能实线，所以就有了 ICP 算法

ICP 算法有较多的数学公式和概念，数学公式总归看起来费劲，这里只简要的理解下其算法步骤：

两个点集 P1，P2，每一步迭代，都朝着距离最小的目标进行。

a. 筛选点对：由 P1 中的点，在 P2 中搜索出其最近的点，组成一个点对；找出两个点集中所有的点对。点对集合相当于进行有效计算的两个新点集。

b. 根据点集对，即两个新点集，计算两个重心。

c. 由新点集，计算出下一步计算的旋转矩阵 R，和平移矩阵 t(其实来源于重心的差异)。

d. 得到旋转矩阵和平移矩阵 Rt，就可以计算点集 P2 进行刚体变换之后的新点集 P2`，由计算 P2 到 P2`的距离平方和，以连续两次距离平方和之差绝对值，作为是否收敛的依据。若小于阈值，就收敛，停止迭代。

e. 重复 a-e，直到收敛或达到既定的迭代次数。

--其中，计算旋转矩阵 R 时，需要矩阵方面的运算。

由新的点集，每个点到重心的距离关系，计算正定矩阵 N，并计算 N 的最大特征值及其最大特征向量；其特征向量等价于旋转的四元数（且是残差和最小的旋转四元数），将四元数就可以转换为旋转矩阵。

（4）优化结果——SBA 稀疏光束平差法

光束平差（BA）几乎不变地用作每个基于特征的多视图重建视觉算法的最后一步，以获得最佳的 3D 结构和运动（即相机矩阵）参数估计。提供初始估计，BA 通过最小化观察和预测图像点之间的再投影误差来同时精细化运动和结构。最小化通常在 Levenberg-Marquardt（LM）算法的帮助下进行。然而，由于大量的未知数有助于最小化再投影误差，当应用于在 BA 的上下文中定义的最小化问题时，LM 算法（例如 MINPACK 的 lmder）的通用实现导致高计算成本。

幸运的是，对于不同 3D 点和相机的参数之间缺乏交互作用导致显示稀疏块结构的基本正态方程（点击这里作为示例）。 sba 通过采用导致相当大的计算增益的 LM 算法的定制的稀疏变体来利用这种稀疏性。 sba 是通用的，因为它授予用户对描述相机和 3D 结构的参数的定义的完全控制。因此，它几乎可以支持多视图重建问题的任何表现/参数化，诸如任意投影相机，部分或完全本征校准的相机，来自固定 3D 点的外部取向（即姿态）估计，内部参数的细化等。用户必须做的来适应 sba 到任何这样的问题是为它提供用于计算估计的图像投影和他们的雅可比的问题和参数化的适当的例程。用于计算分析雅可比行列式的程序可以手动编码，用支持符号分化的工具（例如，枫木）生成，或者使用自动分化技术获得。还有一种替代方法是在有限差分的帮助下近似雅可比矩阵。此外，sba 包括用于检查用户提供的雅可比的一致性的例程。据我们所知，sba 是第一个也是目前唯一一个以自由软件形式发布的软件包。

二、多目立体视觉脚型重建及测量

（1） 图像校正(极线约束)

（2） 特征点提取-斑点中心提取

1. 什么是斑点

斑点通常是指与周围有着颜色和灰度差别的区域。在实际地图中，往往存在着大量这样的斑点，如一颗树是一个斑点，一块草地是一个斑点，一栋房子也可以是一个斑点。由于斑点代表的是一个区域，相比单纯的角点，它的稳定性要好，抗噪声能力要强，所以它在图像配准上扮演了很重要的角色。

2、OpenCV2 斑点检测 Simpleblobdetector 的接口——blob 特征检测原理与实现

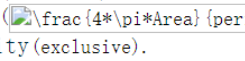
直观上来看，blob 特征就是一团，一坨东西，它并不一定是圆形的，总之它就是那么一团独立存在的特征。

OpenCV 实现的算法如下：

```
class SimpleBlobDetector : public FeatureDetector
{
```

```
public:
struct Params
{
    Params();
    float thresholdStep;
    float minThreshold;
    float maxThreshold;
    size_t minRepeatability;
    float minDistBetweenBlobs;

    bool filterByColor;
    uchar blobColor;

    bool filterByArea;
    float minArea, maxArea;

    bool filterByCircularity;
    float minCircularity, maxCircularity;

    bool filterByInertia;
    float minInertiaRatio, maxInertiaRatio;

    bool filterByConvexity;
    float minConvexity, maxConvexity;
};

SimpleBlobDetector(const SimpleBlobDetector::Params &parameters = SimpleBlobDetector::Params());

protected:
    ...
};
```

1）对[minThreshold, maxThreshold)区间，以 thresholdStep 为间隔，做多次二值化。
2）对每张二值图片，使用 findContours()提取连通域并计算每一个连通域的中心。
3）根据 2 得到的中心，全部放在一起。一些很接近的点［由 theminDistBetweenBlobs 控制多少才算接近］被归为一个 group,对应一个 bolb 特征..
4）从 3 得到的那些点,估计最后的 blob 特征和相应半径，并以 key points 返回。

- **By color**. This filter compares the intensity of a binary image at the center of a blob to blobColor. If they differ, the blob is filtered out. UseblobColor = 0 to extract dark blobs and blobColor = 255 to extract light blobs.
- **By area**. Extracted blobs have an area between minArea (inclusive) and maxArea (exclusive).
- **By circularity**. Extracted blobs have circularity () between minCircularity (inclusive) and maxCircularity(exclusive).
- **By ratio of the minimum inertia to maximum inertia**. Extracted blobs have this ratio between minInertiaRatio (inclusive) andmaxInertiaRatio (exclusive).
- **By convexity**. Extracted blobs have convexity (area / area of blob convex hull) between minConvexity (inclusive) and maxConvexity(exclusive).

你可以设置提取特征的方法，有上面 5 个选项。默认提取黑色圆形的 blob 特征。

的特征点。下面我们就来分析一下该算法。

首先通过一系列连续的阈值把输入的灰度图像转换为一个二值图像的集合，阈值范围为[T_1,T_2]，步长为t，则所有阈值为：

T_1，T_1+t，T_1+2t，T_1+3t，……，T_2　　（1）

第二步是利用Suzuki提出的算法通过检测每一幅二值图像的边界的方式提取出每一幅二值图像的连通区域，我们可以认为由边界所围成的不同的连通区域就是该二值图像的斑点；

第三步是根据所有二值图像斑点的中心坐标对二值图像斑点进行分类，从而形成灰度图像的斑点，属于一类的那些二值图像斑点最终形成灰度图像的斑点，具体来说就是，灰度图像的斑点是由中心坐标间的距离小于阈值T_b的那些二值图像斑点所组成的，即这些二值图像斑点属于该灰度图像斑点；

最后就是确定灰度图像斑点的信息——位置和尺寸。位置是属于该灰度图像斑点的所有二值图像斑点中心坐标的加权和，即公式2，权值q等于该二值图像斑点的惯性率的平方，它的含义是二值图像的斑点的形状越接近圆形，越是我们所希望的斑点，因此对灰度图像斑点位置的贡献就越大。尺寸则是属于该灰度图像斑点的所有二值图像斑点中面积大小居中的半径长度。

$$[X,Y]=\frac{\sum_i q_i[x_i,y_i]}{\sum_i q_i}$$　　（2）

在第二步中，并不是所有的二值图像的连通区域都可以认为是二值图像的斑点，我们往往通过一些限定条件来得到更准确的斑点。这些限定条件包括颜色，面积和形状，斑点的形状又可以用圆度，偏心率，或凸度来表示。

对于二值图像来说，只有两种斑点颜色——白色斑点和黑色斑点，我们只需要一种颜色的斑点，通过确定斑点的灰度值就可以区分出斑点的颜色。

连通区域的面积太大和太小都不是斑点，所以我们需要计算连通区域的面积，只有当该面积在我们所设定的最大面积和最小面积之间时，该连通区域才作为斑点被保留下来。

圆形的斑点是最理想的，任意形状的圆度C定义为：

$$C=\frac{4\pi S}{p^2}$$　　（3）

其中，S和p分别表示该形状的面积和周长，当C为1时，表示该形状是一个完美的圆形，而当C为0时，表示该形状是一个逐渐拉长的多边形。

偏心率是指某一椭圆轨道与理想圆形的偏离程度，长椭圆轨道的偏心率高，而近于圆形的轨道的偏心率低。圆形的偏心率等于0，椭圆的偏心率介于0和1之间，而偏心率等于1表示的是抛物线。直接计算斑点的偏心率较为复

杂，但利用图像矩的概念计算图形的惯性率，再由惯性率计算偏心率较为方便。偏心率E和惯性率I之间的关系为：

$$E^2 + I^2 = 1 \quad (4)$$

因此圆形的惯性率等于1，惯性率越接近1，圆形的程度越高。

最后一个表示斑点形状的量是凸度。在平面中，凸形图指的是图形的所有部分都在由该图形切线所围成的区域的内部。我们可以用凸度来表示斑点凹凸的程度，凸度V的定义为：

$$V = \frac{S}{H} \quad (5)$$

其中，H表示该斑点的凸壳面积

在计算斑点的面积，中心处的坐标，尤其是惯性率时，都可以应用图像矩的方法。

下面我们就介绍该方法。

矩在统计学中被用来反映随机变量的分布情况，推广到力学中，它被用来描述空间物体的质量分布。同样的道理，如果我们将图像的灰度值看作是一个二维的密度分布函数，那么矩方法即可用于图像处理领域。设$f(x,y)$是一幅数字图像，则它的矩M_{ij}为：

$$M_{ij} = \sum_x \sum_y x^i y^j f(x,y) \quad (6)$$

对于二值图像的来说，零阶矩M_{00}等于它的面积。图形的质心为：

$$\{\bar{x}, \bar{y}\} = \{\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}}\} \quad (7)$$

图像的中心矩μ_{pq}定义为：

$$\mu_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q f(x, y) \quad (8)$$

一阶中心矩称为静矩，二阶中心矩称为惯性矩。如果仅考虑二阶中心矩的话，则图像完全等同于一个具有确定的大小、方向和离心率，以图像质心为中心且具有恒定辐射度的椭圆。图像的协方差矩阵为：

$$\mathrm{cov}[f(x,y)] = \begin{bmatrix} \mu'_{20} & \mu'_{11} \\ \mu'_{11} & \mu'_{02} \end{bmatrix} = \begin{bmatrix} \frac{\mu_{20}}{\mu_{00}} & \frac{\mu_{11}}{\mu_{00}} \\ \frac{\mu_{11}}{\mu_{00}} & \frac{\mu_{02}}{\mu_{00}} \end{bmatrix} \quad (9)$$

该矩阵的两个特征值λ_1和λ_2对应于图像强度（即椭圆）的主轴和次轴：

$$\lambda_1 = \frac{\mu'_{20}+\mu'_{02}}{2} + \frac{\sqrt{4\mu'^2_{11}+(\mu'_{20}-\mu'_{02})^2}}{2} \quad (10)$$

$$\lambda_2 = \frac{\mu'_{20}+\mu'_{02}}{2} - \frac{\sqrt{4\mu'^2_{11}+(\mu'_{20}-\mu'_{02})^2}}{2} \quad (11)$$

而图像的方向角度θ为：

$$\theta = \frac{1}{2}\arctan\left(\frac{2\mu'_{11}}{\mu'_{20}-\mu'_{02}}\right) \quad (12)$$

图像的惯性率I为：

$$I = \frac{\lambda_2}{\lambda_1} = \frac{\mu'_{20}+\mu'_{02}-\sqrt{4\mu'^2_{11}+(\mu'_{20}-\mu'_{02})^2}}{\mu'_{20}+\mu'_{02}+\sqrt{4\mu'^2_{11}+(\mu'_{20}-\mu'_{02})^2}} = \frac{\mu_{20}+\mu_{02}-\sqrt{4\mu^2_{11}+(\mu_{20}-\mu_{02})^2}}{\mu_{20}+\mu_{02}+\sqrt{4\mu^2_{11}+(\mu_{20}-\mu_{02})^2}} \quad (13)$$

3、课题具体实现:

保证光照均匀充分，使用 Simpleblobdetector：：detect()提取特征点，并根据红蓝颜色阈值分为两类点

```
/*************************************************特征点检测**/
SimpleBlobDetector::Params params1;
params1.minThreshold = 20;
params1.maxThreshold = 240;
params1.thresholdStep = 5;
params1.minArea = 10;
params1.maxArea = 300;
//params1.minConvexity = 0.5f;
//params1.minInertiaRatio = 0.05f;
//params1.filterByCircularity = 1;    //斑点圆度的限制变量，默认是不限制
//params1.minCircularity = 0.5f;    //斑点的最小圆度
//params1.minRepeatability = 2;

vector<KeyPoint> kpt1, kpt2, kptr1, kptr2;
Featuredetect(img1r, kpt1, kptr1, params1);
Featuredetect(img2r, kpt2, kptr2, params1);
```

```
void Featuredetect(IplImage* img1r, vector<KeyPoint> &key_p
{
    Mat img = Mat::Mat(img1r);
    Mat imghsv(img.size(),CV_32FC3);
    cvtColor(img,imghsv,CV_BGR2HSV);

    //SimpleBlobDetector::Params params;
    SimpleBlobDetector detector(params);
    vector<KeyPoint> key_points;
    detector.detect(img, key_points);
```

（3）种子点匹配-极线约束(在图像校正时实现)+SAD

3.1 极线约束

3.2 SAD

SAD 算法：SAD 算法是一种最简单的匹配算法，该算法快速、但并不精确,通常用于多级处理的初步筛选。

用公式表示为：SAD(u,v) = Sum{|Left(u,v) - Right(u,v)|} 选择最小值

此 种方法就是以左目图像的源匹配点为中心，定义一个窗口 D，其大小为（2m+1） (2n+1)，统计其窗口的灰度值的和，然后在右目图像中逐步计算其左右窗口的灰度和的差值，最后搜索到的差值最小的区域的中心像素即为匹配点。

基本流程：

1.构造一个小窗口，类似与卷积核。

2.用窗口覆盖左边的图像，选择出窗口覆盖区域内的所有像素点。

3.同样用窗口覆盖右边的图像并选择出覆盖区域的像素点。

4.左边覆盖区域减去右边覆盖区域，并求出所有像素点差的绝对值的和。

5.移动右边图像的窗口，重复 3，4 的动作。（这里有个搜索范围，超过这个范围跳出）

6.找到这个范围内 SAD 值最小的窗口，即找到了左边图像的最佳匹配的像素块。

SAD算法

介绍

绝对误差和算法（Sum of Absolute Differences，简称SAD算法）。实际上，SAD算法与MAD算法思想几乎是完全一致，只是其相似度测量公式有一点改动（计算的是子图与模板图的L1距离），这里不再赘述。

$$D(i,j)=\sum_{s=1}^{M}\sum_{t=1}^{N}|S(i+s-1,j+t-1)-T(s,t)|$$

相关常用算法：

常见的基于灰度的模板匹配算法

1、MAD 算法：平均绝对差算法（**Mean Absolute Differences**，简称 **MAD** 算法），它是 **Leese** 在 1971 年提出的一种匹配算法。是模式识别中常用方法，该算法的思想简单，具有较高的匹配精度和较少的计算量，广泛用于图像匹配。
设 **S(x,y)**是大小为 **mxn** 的搜索图像，*T(x,y)*时 **MxN** 的模板图像，

算法思路

在搜索图**S**中，取以**(i,j)**为左上角，**MxN**大小的子图，计算其与模板图相似度；在所有能够取到的子图中，找到与模板图最相似的子图作为最终结果。**MAD**算法的相似性测度公式如下。显然，平均绝对差**D(i,j)**越小，表明越相似，故只需找到最小的**D(i,j)**即可确定子图位置：

$$D(i,j)=\frac{1}{M\times N}\sum_{s=1}^{M}\sum_{t=1}^{N}|S(i+s-1,j+t-1)-T(s,t)|$$

其中：$1\le i\le m-M+1$，$1\le j\le n-N+1$

算法评价：

优点：

①思路简单，容易理解（子图与模板图对应位置上，灰度值之差的绝对值总和，再求平均，实质：是计算的是子图与模板图的L1距离的平均值）。

②运算过程简单，匹配精度高。

缺点：

①运算量偏大。

②对噪声非常敏感。

SSD算法

误差平方和算法（**Sum of Squared Differences**，简称**SSD**算法），也叫差方和算法。实际上，**SSD**算法与**SAD**算法如出一辙，只是其相似度测量公式有一点改动（计算的是子图与模板图的**L2**距离）。这里不再赘述。

$$D(i,j)=\sum_{s=1}^{M}\sum_{t=1}^{N}[S(i+s-1,j+t-1)-T(s,t)]^2$$

MSD算法

平均误差平方和算法（**Mean Square Differences**，简称**MSD**算法），也称均方差算法。实际上，**MSD**之余**SSD**，等同于**MAD**之余**SAD**（计算的是子图与模板图的**L2**距离的平均值），故此处不再赘述。

$$D(i,j)=\frac{1}{M\times N}\sum_{s=1}^{M}\sum_{t=1}^{N}[S(i+s-1,j+t-1)-T(s,t)]^2$$

NCC算法

归一化积相关算法（**Normalized Cross Correlation**，简称**NCC**算法），与上面算法相似，依然是利用子图与模板图的灰度，通过归一化的相关性度量公式来计算二者之间的匹配程度。

$$R(i,j)=\frac{\sum_{s=1}^{M}\sum_{t=1}^{N}\left|S^{i,j}(s,t)-E\left(S^{i,j}\right)\right|\cdot|T(s,t-E(T))|}{\sqrt{\sum_{s=1}^{M}\sum_{t=1}^{N}[S^{i,j}(s,t)-E(S^{i,j})]^2\cdot\sum_{s=1}^{M}\sum_{t=1}^{N}[T(s,t)-E(T)]^2}}$$

其中，$E\left(S^{i,j}\right)$、$E(T)$分别表示**(i,j)**处子图、模板的平均灰度值。

SSDA算法

序贯相似性检测算法（Sequential Similiarity Detection Algorithm，简称SSDA算法），它是由Barnea和Sliverman于1972年，在文章《A class of algorithms for fast digital image registration》中提出的一种匹配算法，是对传统模板匹配算法的改进，比MAD算法快几十到几百倍。

与上述算法假设相同：S(x,y)是mxn的搜索图，T(x,y)是MxN的模板图，$S_{i,j}$是搜索图中的一个子图（左上角起始位置为(i,j)）。显然：$1 \le i \le m-M-1,\ 1 \le j \le n-N-1$

SSDA算法描述如下：

①定义绝对误差：

$$\varepsilon(i,j,s,t) = \left| S_{i,j}(s,t) - \overline{S_{i,j}} - T(s,t) + \bar{T} \right|$$

其中，带有上划线的分别表示子图、模板的均值：

$$\overline{S_{i,j}} = E(S_{i,j}) = \frac{1}{M \times N} \sum_{s=1}^{M} \sum_{t=1}^{N} S_{i,j}(s,t)$$

$$\bar{T} = E(T) = \frac{1}{M \times N} \sum_{s=1}^{M} \sum_{t=1}^{N} T(s,t)$$

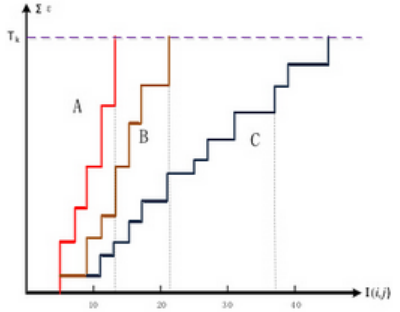
实际上，绝对误差就是子图与模板图各自去掉其均值后，对应位置之差的绝对值。

②设定阈值Th；

③在模板图中**随机**选取**不重复**的像素点，计算与当前子图的绝对误差，将误差累加，当误差累加值超过了Th时，记下累加次数H，所有子图的累加次数H用一个表R(i,j)来表示。SSDA检测定义为：

$$R(i,j) = \left\{ H \left| \min_{1 \le H \le M \times N} \left[\sum_{h=1}^{H} \varepsilon(i,j,s,t) \ge Th \right] \right. \right\}$$

下图给出了A、B、C三点的误差累计增长曲线，其中A、B两点偏离模板，误差增长得快；C点增长缓慢，说明很可能是匹配点（图中Tk相当于上述的Th，即阈值；I(i,j)相当于上述R(i,j)，即累加次数）。



④在计算过程中，随机点的累加误差和超过了阈值（记录累加次数H）后，则放弃当前子图转而对下一个子图进行计算。遍历完所有子图后，选取最大R值所对应的(i,j)子图作为匹配图像【若R存在多个最大值（一般不存在），则取累加误差最小的作为匹配图像】。

由于随机点累加值超过阈值Th后便结束当前子图的计算，所以不需要计算子图所有像素，大大提高了算法速度；为进一步提高速度，可以先进行粗配准，即：隔行、隔离的选取子图，用上述算法进行粗糙的定位，然后再对定位到的子图，用同样的方法求其8个邻域子图的最大R值作为最终配准图像。这样可以有效的减少子图个数，减少计算量，提高计算速度。

SATD算法

hadamard变换算法（Sum of Absolute Transformed Difference，简称SATD算法），它是经hadamard变换再对绝对值求和算法。hadamard变换等价于把原图像Q矩阵左右分别乘以一个hadamard变换矩阵H。其中，hardamard变换矩阵H的元素都是1或-1，是一个正交矩阵，可以由MATLAB中的hadamard(n)函数生成，n代表n阶方阵。

SATD算法就是将模板与子图做差后得到的矩阵Q，再对矩阵Q求其hadamard变换（左右同时乘以H，即HQH），对变换都得矩阵求其元素的绝对值之和即SATD值，作为相似度的判别依据。对所有子图都进行如上的变换后，找到SATD值最小的子图，便是最佳匹配。

（3） 种子点三角化- delaunay 三角剖分

对图 1 种子点进行 Delaunay 三角剖分，根据（2）中种子点的匹配关系，对图 2 种子点建立对应三角剖分。

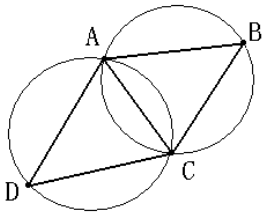
【定义】三角剖分：假设 V 是二维实数域上的有限点集，边 e 是由点集中的点作为端点构成的封闭线段，E 为 e 的集合。那么该点集 V 的一个三角剖分 T=(V,E)是一个平面图 G，该平面图满足条件：

1.除了端点，平面图中的边不包含点集中的任何点。
2.没有相交边。
3.平面图中所有的面都是三角面，且所有三角面的合集是散点集 V 的凸包。
在实际中运用的最多的三角剖分是 Delaunay 三角剖分，它是一种特殊的三角剖分。先从 Delaunay 边说起：
【定义】Delaunay 边：假设 E 中的一条边 e（两个端点为 a,b），e 若满足下列条件，则称之为 Delaunay 边：存在一个圆经过 a,b 两点，圆内(注意是圆内，圆上最多三点共圆)不含点集 V 中任何其他的点，这一特性又称空圆特性。
【定义】Delaunay 三角剖分：如果点集 V 的一个三角剖分 T 只包含 Delaunay 边，那么该三角剖分称为 Delaunay 三角剖分。
【定义】假设 T 为 V 的任一三角剖分，则 T 是 V 的一个 Delaunay 三角剖分，当前仅当 T 中的每个三角形的外接圆的内部不包含 V 中任何的点。

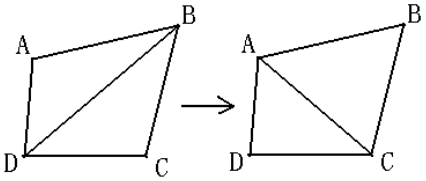
1.3.Delaunay三角剖分的准则

要满足Delaunay三角剖分的定义，必须符合两个重要的准则：

1、空圆特性：Delaunay三角网是唯一的（任意四点不能共圆），在Delaunay三角形网中任一三角形的外接圆范围内不会有其它点存在。如下图所示：



2、最大化最小角特性：在散点集可能形成的三角剖分中，Delaunay三角剖分所形成的三角形的最小角最大。从这个意义上讲，Delaunay三角网是"最接近于规则化的"的三角网。具体的说是指在两个相邻的三角形构成凸四边形的对角线，在相互交换后，六个内角的最小角不再增大。如下图所示：



Delaunay 剖分所具备的优异特性：
1.最接近：以最近的三点形成三角形，且各线段(三角形的边)皆不相交。
2.唯一性：不论从区域何处开始构建，最终都将得到一致的结果。
3.最优性：任意两个相邻三角形形成的凸四边形的对角线如果可以互换的话，那么两个三角形六个内角中最小的角度不会变大。
4.最规则：如果将三角网中的每个三角形的最小角进行升序排列，则 Delaunay 三角网的排列得到的数值最大。
5.区域性：新增、删除、移动某一个顶点时只会影响临近的三角形。
6.具有凸多边形的外壳：三角网最外层的边界形成一个凸多边形的外壳。

2.1 Lawson算法

逐点插入的Lawson算法是Lawson在1977提出的，该算法思路简单，易于编程实现。基本原理为：首先建立一个大的三角形或多边形，把所有数据点包围起来，向其中插入一点，该点与包含它的三角形三个顶点相连，形成三个新的三角形，然后逐个对它们进行空外接圆检测，同时用Lawson设计的局部最优化过程LOP进行优化，集通过交换对角线的方法来保证所形成的三角网为Delaunay三角网。

上述基于散点的构网算法理论严密、唯一性好，网格满足空圆特性，较为理想。由其逐点插入的构网过程可知，遇到非Delaunay边时，通过删除调整，可以构造形成新的Delaunay边。在完成构网后，增加新点时，无需对所有点进行重新构网，只需要对新点的影响三角形范围进行局部联网，且局部联网的方法简单易行。同样，点的删除、移动也可快速动态地进行。但在实际应用当中，这种构网算法当点集较大时网速度也较慢，如果点集范围是非凸区域或者存在内环，则会产生非法三角形。

2.2 Bowyer-Watson算法

目前采用逐点插入方式生成的Delaunay三角网的算法主要基于Bowyer-Watson算法，Bowyer-Watson算法的主要步骤如下：

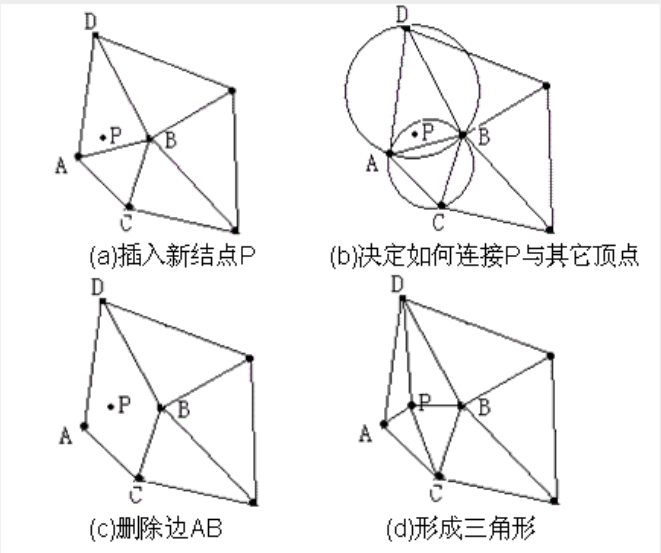
1）建立初始三角网格：针对给定的点集V,找到一个包含该点集的矩形R,我们称R为辅助窗口。连接R的任意一条对角线，形成两个三角形，作为初始Delaunay三角网格。

2）逐点插入：假设目前已经有一个Delaunay三角网格T,现在在它里面再插入一个点P,需要找到该点P所在的三角形。从P所在的三角形开始，搜索该三角形的邻近三角形，并进行空外接圆检测。找到外接圆包含点P的所有的三角形并删除这些三角形，形成一个包含P的多边形空腔，我们称之为Delaunay空腔。然后连接P与Delaunay腔的每一个顶点，形成新的Delaunay三角网格。

3）删除辅助窗口R:重复步骤2），当点集V中所有点都已经插入到三角形网格中后，将顶点包含辅助窗口R的三角形全部删除。

在这些步骤中，快速定位点所在的三角形、确定点的影响并构建Delaunay腔的过程是每插入一个点都会进行的。随着点数的增加，三角形数目增加很快，因此缩短这两个过程的计算时间，是提高算法效率的关键。

算法执行图示如下：



2、Delaunay剖分的算法

Delaunay剖分是一种三角剖分的标准，实现它有多种算法。

表 1 几种 Delaunay 生成算法时间复杂度比较[7]

	算法	一般情况	最坏情况
分	Lewis 和 Robinson（1987）	O(N log N)	$O(N^2)$
治	lee 和 Schachlter（1980）	O(N log N)	O(N log N)
算	Dwyer(1987)	O(N log log N)	O(N log N)
法	Chew(1989)	O(N log N)	O(N log N)
逐	Lawson(1977)	$O(N^{4/3})$	$O(N^2)$
点	lee 和 Schachlter（1980）	$O(N^{3/2})$	$O(N^2)$
插	Bowyer(1981)	$O(N^{3/2})$	$O(N^2)$
入	Watson(1981)	$O(N^{3/2})$	$O(N^2)$
法	Sloan(1987)	$O(N^{5/4})$	$O(N^2)$
三	Green 和 Sibson(1987)	$O(N^{3/2})$	$O(N^2)$
角	Brassel 和 Reif(1979)	$O(N^{3/2})$	$O(N^2)$
网	Macullagh 和 Ross(1980)	$O(N^{3/2})$	$O(N^2)$
生	Mirante 和 Weigarten(1982)	$O(N^{3/2})$	$O(N^2)$
成			
法			

由表可以看出，三角网生成法的时间效率最低，分治算法的时间效率最高，逐点插入法效率居中。由于区域生长法本质的缺陷，导致其效率受限，这种方法在80年代中期以后已经很少使用。分治算法时间效率相对较高，但是由于其递归执行，所以需要较大的内存空间，导致其空间效率较低。此外，分治法的数据处理及结果的优化需要的工作量也比较大。逐点插入算法实现简单，时间效率比较高，而运行占用的空间也较小，从时间效率和空间效率综合考虑，性价比最高，因而应用广泛。

1977年，Lawson提出了逐点插入法建立Delaunay三角网的算法思想。之后Lee和Schachlter,Bowyer,Watson,Sloan,先后进行了发展和完善。他们的算法在初始化三角网的建立方法、定位点所在三角形的过程、以及插入的过程方面各具特点。

(4) 三角约束

For each feature $\mathbf{P_i} \in \mathcal{P}_A$ (*e.g.* the feature marked by dashed red circle) in $\triangle_{abc}$, the relationship between $\mathbf{P_i}$ and the vertexes of $\triangle_{abc}$ is

$$\mathbf{P_i} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}), \tag{1}$$

where β and γ are the scale coefficients of the vector $(\mathbf{b} - \mathbf{a})$ and $(\mathbf{c} - \mathbf{a})$ respectively. Fortunately, the three vertexes of $\triangle_{abc}$ are known and the parameters $\mathbf{K}$ can be computed easily by

$$\mathbf{K} = \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} x_a & x_b & x_c \\ y_a & y_b & y_c \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}, \tag{2}$$

where $\alpha = 1 - \beta - \gamma$. The same parameters for the relationship between the estimated point $\mathbf{P}_e$ (the black point in Fig. 2 (b)) and the vertexes of $\triangle_{a'b'c'}$ in I_B hold true if the triangles are true positive correspondence. As a consequence, we estimate the coordinates of $\mathbf{P}_e$ via

$$\begin{bmatrix} x_e \\ y_e \\ 1 \end{bmatrix} = \begin{bmatrix} x_{a'} & x_{b'} & x_{c'} \\ y_{a'} & y_{b'} & y_{c'} \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix}. \tag{3}$$

To be more robust to noises and distortions, we define the area around the $\mathbf{P}_e$ within R pixels (in our experiments, R = 3) as candidate area and the features in this area as candidate features (the green '△'s in Fig.

The similarity score between the $\mathbf{P}_i$ and the candidate feature $\mathbf{C}_j$ is measured by

$$\mathbf{s}_j = 1.5^{-(dist_j/R)^2} \times \mathbf{D}_i^T \mathbf{D}_{c_j}, (j = 1, 2, ..., |\mathcal{C}|), \quad (4)$$

where $dist_j$ is the Euclidean distance between the $\mathbf{C}_j$ and the $\mathbf{P}_i$, $\mathbf{D}_i$ and $\mathbf{D}_{c_j}$ are the descriptors for the $\mathbf{P}_i$ and the $\mathbf{C}_j$ respectively, and $|\circ|$ stands for the cardinality of a set. If the maximum score of all the features in $\mathcal{C}$ is greater than a predefined threshold τ, the corresponding feature pair is considered as temporary match, and there is at most one match for every feature. In our experiments, we use $\tau = 0.4$ for all the evaluations.

After processing all the features from $\mathcal{P}_A$, there is a set $\mathcal{T}$ containing all the temporary matches for $\mathcal{P}_A$ and $\mathcal{P}_B$. The temporary matches are accepted as final matches if they satisfy

$$|\mathcal{T}| > \lambda \min\{|\mathcal{P}_A|, |\mathcal{P}_B|\}, \tag{5}$$

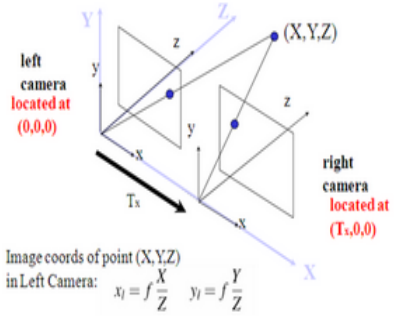
where $\lambda = 0.3$ in our experiments. Otherwise, the triangle pair $\triangle_{abc}$ and $\triangle_{a'b'c'}$ and the temporary matches between them are discarded. This straightforward strategy is based on the observation that if the pair of the triangles is true correspondence, they will approximately even perfectly satisfy the Triangle-Constraint due to the accurate vertexes, or they will be greatly different. And if all the relative triangles of a vertex are discarded, the vertex is then removed from seed points. This strategy is able to reduce wrong matches in the Triangle-Constraint step and filter out the false positive matches survived from the Bi-matching step.

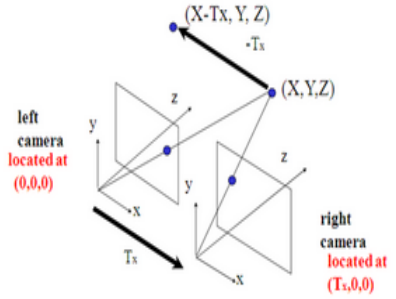
The Triangle-Constraint Measurement is invariant to translation, rotation and scale transformations due to the accurate features obtained from the Bi-matching step. It is also invariant to affine transformation and robust to partially perspective transformation, since the Triangle-Constraint itself is affine invariant.

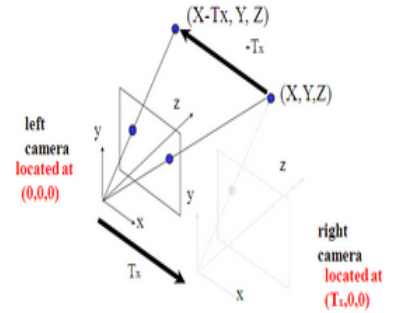
（5）三维点坐标重建

5.1 视差计算

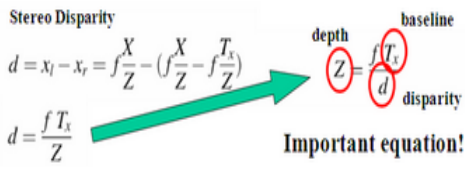
首先需要搞清楚一个非常重要的概念，视差，搞清楚视差，后面的就简单了，老生常谈的问题我不想多说，网上那些一大堆，我希望给大家的是一些明了的东西







这三幅图看明白了就行，其实视差确实很简单，但很多人都没去理清楚，第一幅图是三维世界的一个点在两个相机的成像，我们可以相信的是，这两个在各自相机的相对位置基本不可能是一样的，而这种位置的差别，也正是我们眼睛区别3D和2D的关键，将右边的摄像机投影到左边，怎么做呢？因为他的坐标和左边相机的左边相距Tx（标定测出来的外参数），所以它相当于在左边的相机对三维世界内的（x-tx,y,z）进行投影，所以这时候，一个完美的形似三角形就出来，这里视差就是d=x-x'，



得到视差以后，再用相似三角形.....也就得到了深度也就是距离啦。

5.2 三维点坐标计算

（6）泊松重建

1.泊松曲面重建简介

激光扫描设备的发展使得被测物体更多细节的数据获取成为可能。根据采样数据的模型重建，在许多实际应用中具有实际意义，如在汽车、航空等工业领域中，复杂外形产品的设计仍需要根据手工模型，采用逆向工程的手段建立产品的数字化模型；根据测量数据建立人体以及骨骼和器官的计算机模型在医学、定制生产等方面都有重要意义。

关于采样数据的重构有基于组合结构和基于隐函数两类方法。

基于组合结构的方法，如 Delaunay triangulations, alpha shapes 或Voronoi diagrams 这些方法通过建立三角形网格插值所有或大多数点。当存在噪声点时，所产生的表面往往是锯齿状，因此需要平滑或对数据进行处理（refit to the points in subsequent processing）。

隐函数方法则通过定义分段函数，定义模型内部的值大于零，模型外部它的值小于零，然后提取值为零的等值面，这类方法可以直接地重构逼近表面，如基于快速傅立叶转换和径向基函数(R BF s) 的重构方法都属于隐函数重构方法。隐函数方法分为全局方法和局部方法（暂不赘述）。

泊松曲面重建属于隐函数方法实现。泊松表面重建的算法融合了全局和局部方法的优点，采取隐性拟合的方式，通过求解泊松方程来取得点云模型所描述的表面信息代表的隐性方程，通过对该方程进行等值面提取，从而得到具有几何实体信息的表面模型。优点在于，重建出的模型具有水密性的封闭特征，具有良好的几何表面特性和细节特性。

经过屏蔽的泊松表面重建算法，在原有泊松表面重建算法的基础上对输入的点云进行插值约束（引入点集的约束和梯度的约束），将等值面提取的输入方程由原始的泊松方程转化为屏蔽泊

为防止屏蔽因子经过尺度变换造成错误信息，加入了相关条件约束。

2.泊松曲面重建的数学基础

泊松曲面重建基于泊松方程。泊松方程是一个比较常见的偏微分方程，在很多领域被应用，如高动态范围图像的调和映射、图像区域的无缝编辑、流体力学、网格编辑等，多重网格泊松方法已应用于高效GPU计算。

由梯度关系得到采样点和指示函数的积分关系，根据积分关系利用划分块的方法获得点集的向量场，计算指示函数梯度场的逼近，构成泊松方程。根据泊松方程使用矩阵迭代求出近似解，采用移动立方体算法提取等值面，对所测数据点集重构出被测物体的模型，泊松方程在边界处的误差为零，因此得到的模型不存在假的表面框。采用隐函数的泊松方程进行表面重构是利用泊松方程在边界处没有误差的特点。

直接计算梯度场会引起向量场在表面边缘的无穷大值。因此首先用平滑滤波卷积指示函数，然后求平滑函数的梯度场。

高斯散度理论：平滑指示函数的梯度等于平滑表面法向场得到的向量场。

由于曲面未知，无法直接计算表面积分，把采样点集划分为小的区域块，通过对所有块的积分求和近似计算。知道向量场V后，可求指示函数。但向量场V不可积，使用最小平方逼近理论求解，应用散度算子得到泊松方程。

泊松表面重建一次性把所有的点都考虑在内，因此对噪声点有很好的弹性。泊松仿佛允许的层次结构支持局部的基函数，因此对稀疏线性系统的情况有很好的支持。在此基础上描述了多尺度的空间自适应算法，其时间和空间复杂度同重建模型的大小成正比。

整个算法的步骤包括对具有法向量信息的输入点云信息的预处理，对全局问题离散化，对离散化后的子数据求解，求解泊松问题后的等值面提取，以及后期优化处表面重建过程：

1、定义八叉树。使用八叉树结构存储点集，根据采样点集的位置定义八叉树，然后细分八叉树使每个采样点都落在深度为D的叶节点；

2、设置函数空间：对八叉树的每个节点设置空间函数F，所有节点函数F的线性和可以表示向量场V，基函数F采用了盒滤波的n维卷积；

3、创建向量场：均匀采样的情况下，假设划分的块是常量，通过向量场V逼近指示函数的梯度。采用三次条样插值（三线插值）；

4、求解泊松方程：方程的解采用拉普拉斯矩阵迭代求出；

5、提取等值面：为得到重构表面，需要选择阈值获得等值面；先估计采样点的位置，然后用其平均值进行等值面提取，然后用移动立方体算法得到等值面。

（7）测量方法

三、基础知识：

3.1 相机标定

棋盘格法是摄像机标定中常用的一种方法，在使用该方法时需要对棋盘格的角点进行检测。**OpenCV**中封装了一个专门用于棋盘格角点检测的函数即cv::findChessboardCorners()，同时，也提供了一个专门用于绘制棋盘格角点的函数cv::drawChessboardCorners()。下面将对这两个函数进行详细的介绍。

1、cv::findChessboardCorners()棋盘格角点检测

该函数的具体调用形式如下：

```
[cpp] view plain  copy  print ?
01. bool cv::findChessboardCorners( // 如果找到角点则返回true
02.     cv::InputArray image, // 输入的棋盘格图像（8UC1或8UC3）
03.     cv::Size patternSize, // 棋盘格内部角点的行、列数
04.     cv::OutputArray corners, // 输出的棋盘格角点
05.     int flags = cv::CALIB_CB_ADAPTIVE_THRESH
06.     | cv::CALIB_CB_NORMALIZE_IMAGE
07. );
```

第一个参数是输入的棋盘格图像（可以是8位单通道或三通道图像）。

第二个参数是棋盘格内部的角点的行列数（注意：不是棋盘格的行列数，如下图棋盘格的行列数分别为4、8，而内部角点的行列数分别是3、7，因此这里应该指定为cv::Size(3, 7)）。

第三个参数是检测到的棋盘格角点，类型为std::vector<cv::Point2f>。

第四个参数flag，用于指定在检测棋盘格角点的过程中所应用的一种或多种过滤方法，可以使用下面的一种或多种，如果都是用则使用OR：

cv::CALIB_CB_ADAPTIVE_THRESH：cv::findChessboardCorners()默认的阈值化处理基于平均亮度，如果该标志指定，则使用自适应滤波（自适应滤波见

OpenCV3中的阈值化操作——cv::threshold()与cv::adaptiveThreshold()详解）。

cv::CALIB_CB_NORMALIZE_IMAGE：阈值化前使用cv::equalizeHist()进行直方图均衡化处理。

cv::CALIB_CB_FILTER_QUADS：

cv::CALIB_CV_FAST_CHECK：

当然，找到的角点还需要使用cv::cornerSubPix()进行精度上的优化（具体请参考OpenCV3中的角点检测——cv::goodFeaturesToTrack()与cv::cornerSubPix()详解）。

2、cv::drawChessboardCorners()棋盘格角点的绘制

cv::drawChessboardCorners()的具体调用形式如下：

```cpp
[cpp] view plain  copy  print ?
01. void cv::drawChessboardCorners(
02.     cv::InputOutputArray image, // 棋盘格图像（8UC3）即是输入也是输出
03.     cv::Size patternSize, // 棋盘格内部角点的行、列数
04.     cv::InputArray corners, // findChessboardCorners()输出的角点
05.     bool patternWasFound // findChessboardCorners()的返回值
06. );
```

第一个参数是棋盘格图像（8UC3）。

第二个参数是棋盘格内部角点的行、列，和cv::findChessboardCorners()指定的相同。

第三个参数是检测到的棋盘格角点。

第四个参数是cv::findChessboardCorners()的返回值。

阈值化操作在图像处理中是一种常用的算法，比如图像的二值化就是一种最常见的一种阈值化操作。opencv2和opencv3中提供了直接阈值化操作cv::threshold()和自适应阈值化操作cv::adaptiveThreshold()两种阈值化操作接口，这里将对这两个接口进行介绍和对比。

1、直接阈值化——cv::threshold()

阈值化操作的基本思想是，给定一个输入数组和一个阈值，数组中的每个元素将根据其与阈值之间的大小发生相应的改变。opencv3中支持这一操作的接口是cv::threshold()，具体调用方法如下：

```cpp
[cpp] view plain  copy  print ?
01. double cv::threshold(
02.     cv::InputArray src, // 输入图像
03.     cv::OutputArray dst, // 输出图像
04.     double thresh, // 阈值
05.     double maxValue, // 向上最大值
06.     int thresholdType // 阈值化操作的类型
07. );
```

如下表所示，每一种阈值化操作类型，对应着一种源图像上每一个像素点与阈值thresh之间比较操作。根据源图像像素和阈值之间的大小关系，目标像素可能被置为0、原像素值、或者设定的最大值maxValue。

Threshold type	Operation
cv::THRESH_BINARY	$DST_I = (SRC_I > thresh) ? MAXVALUE : 0$
cv::THRESH_BINARY_INV	$DST_I = (SRC_I > thresh) ? 0 : MAXVALUE$
cv::THRESH_TRUNC	$DST_I = (SRC_I > thresh) ? THRESH : SRC_I$
cv::THRESH_TOZERO	$DST_I = (SRC_I > thresh) ? SRC_I : 0$
cv::THRESH_TOZERO_INV	$DST_I = (SRC_I > thresh) ? 0 : SRC_I$

另外，在opencv3中cv::threshold()函数还支持一种特殊的阈值化操作方式，即Otsu算法。该算法的主要思想是，在进行阈值化时，考虑所有可能的阈值，分别计算低于阈值和高于阈值像素的方差，使下式最小化的值作为阈值：

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

其中，两类像素方差的权值由两类像素的个数决定。这种阈值化的结果相对来说比较理想，可以避免寻找合适阈值的操作，但是这种方式运算量较大，费时。处理的结果如下：

但是，直接阈值化操作是一种一刀切的方式，对于亮度分布差异较大的图像，常常无法找到一个合适的阈值。如下所示，对棋盘格进行二值化操作，由于图像右上角区域和图像下部的亮度差异较为大，无法找到一个合适的阈值，将棋盘上的所有棋盘格给区分开来。

针对于上述情况，我们需要一种改进的阈值化算法，即自适应阈值化。

2、自适应阈值化——cv::adaptiveThreshold()

自适应阈值化能够根据图像不同区域亮度分布的，改变阈值，具体调用方法如下：

```
[cpp] view plain  copy  print  ?
01. void cv::adaptiveThreshold(
02.     cv::InputArray src, // 输入图像
03.     cv::OutputArray dst, // 输出图像
04.     double maxValue, // 向上最大值
05.     int adaptiveMethod, // 自适应方法，平均或高斯
06.     int thresholdType // 阈值化类型
07.     int blockSize, // 块大小
08.     double C // 常量
09. );
```

cv::adaptiveThreshold()支持两种自适应方法，即cv::ADAPTIVE_THRESH_MEAN_C（平均）和cv::ADAPTIVE_THRESH_GAUSSIAN_C（高斯）。在两种情况下，自适应阈值T(x, y)。通过计算每个像素周围bxb大小像素块的加权均值并减去常量C得到。其中，b由blockSize给出，大小必须为奇数；如果使用平均的方法，则所有像素周围的权值相同；如果使用高斯的方法，则（x,y）周围的像素的权值则根据其到中心点的距离通过高斯方程得到。

3.2 常用颜色空间

3.3 常见的聚类算法

2.1 k-means聚类算法

k-means是划分方法中较经典的聚类算法之一。由于该算法的效率高，所以在对大规模数据进行聚类时被广泛应用。目前，许多算法均围绕着该算法进行扩展和改进。

k-means算法以k为参数，把n个对象分成k个簇，使簇内具有较高的相似度，而簇间的相似度较低。k-means算法的处理过程如下：首先，随机地 选择k个对象，每个对象初始地代表了一个簇的平均值或中心;对剩余的每个对象，根据其与各簇中心的距离，将它赋给最近的簇;然后重新计算每个簇的平均值。 这个过程不断重复，直到准则函数收敛。通常，采用平方误差准则，其定义如下：

$$E=\sum_{i=1}^{k}\sum_{p\in C_i}|p-m_i|^2$$

这里E是数据库中所有对象的平方误差的总和，p是空间中的点，mi是簇Ci的平均值[9]。该目标函数使生成的簇尽可能紧凑独立，使用的距离度量是欧几里得距离,当然也可以用其他距离度量。k-means聚类算法的算法流程如下：

输入：包含n个对象的数据库和簇的数目k；

输出：k个簇，使平方误差准则最小。

步骤：

(1) 任意选择k个对象作为初始的簇中心；

(2) repeat；

(3) 根据簇中对象的平均值，将每个对象(重新)赋予最类似的簇；

(4) 更新簇的平均值，即计算每个簇中对象的平均值；

(5) until不再发生变化。

2.2 层次聚类算法

根据层次分解的顺序是自底向上的还是自上向下的，层次聚类算法分为凝聚的层次聚类算法和分裂的层次聚类算法。

凝聚型层次聚类的策略是先将每个对象作为一个簇，然后合并这些原子簇为越来越大的簇，直到所有对象都在一个簇中，或者某个终结条件被满足。绝大多数层次聚类属于凝聚型层次聚类，它们只是在簇间相似度的定义上有所不同。四种广泛采用的簇间距离度量方法如下：

最小距离：

$$d_{\min}(c_i,c_j)=\min_{p\in c_i,p'\in c_j}|p-p'|$$

最大距离：

$$d_{\max}(c_i,c_j)=\max_{p\in c_i,p'\in c_j}|p-p'|$$

平均值的距离：

$$d_{\text{mean}}(c_i,c_j)=|m_i-m_j|$$

平均距离：

$$d_{\text{avg}}(c_i,c_j)=\frac{1}{n_i n_j}\sum_{p\in c_i}\sum_{p'\in c_j}|p-p'|$$

这里，$|p-p'|$是两个对象p和p'之间的距离，m_i是簇c_i的平均值，n_i是簇c_i中对象的数目。

这里给出采用最小距离的凝聚层次聚类算法流程：

(1) 将每个对象看作一类，计算两两之间的最小距离；

(2) 将距离最小的两个类合并成一个新类；

(3) 重新计算新类与所有类之间的距离；

(4) 重复(2)、(3)，直到所有类最后合并成一类。

2.3 SOM聚类算法

SOM神经网络[11]是由芬兰神经网络专家Kohonen教授提出的，该算法假设在输入对象中存在一些拓扑结构或顺序，可以实现从输入空间(n维)到输出平面(2维)的降维映射，其映射具有拓扑特征保持性质,与实际的大脑处理有很强的理论联系。

SOM网络包含输入层和输出层。输入层对应一个高维的输入向量，输出层由一系列组织在2维网格上的有序节点构成，输入节点与输出节点通过权重向量连接。 学习过程中，找到与之距离最短的输出层单元，即获胜单元，对其更新。同时，将邻近区域的权值更新，使输出节点保持输入向量的拓扑特征。

算法流程：

(1) 网络初始化，对输出层每个节点权重赋初值；

(2) 将输入样本中随机选取输入向量，找到与输入向量距离最小的权重向量；

(3) 定义获胜单元，在获胜单元的邻近区域调整权重使其向输入向量靠拢；

(4) 提供新样本、进行训练；

(5) 收缩邻域半径、减小学习率、重复，直到小于允许值，输出聚类结果。

2.4 FCM聚类算法

1965年美国加州大学柏克莱分校的扎德教授第一次提出了'集合'的概念。经过十多年的发展，模糊集合理论渐渐被应用到各个实际应用方面。为克服非此即彼的分类缺点，出现了以模糊集合论为数学基础的聚类分析。用模糊数学的方法进行聚类分析，就是模糊聚类分析[12]。

FCM算法是一种以隶属度来确定每个数据点属于某个聚类程度的算法。该聚类算法是传统硬聚类算法的一种改进。

设数据集 $X=\{x_1,x_2,\cdots x_n\}$，它的模糊 c 划分可用模糊矩阵 $\boldsymbol{U}=[u_{ij}]$ 表示，矩阵 $\boldsymbol{U}$ 的元素 u_{ij} 表示第 $j(j=1,2,\cdots,n)$ 个数据点属于第 $i(i=1,2,\cdots,c)$ 类的隶属度，u_{ij} 满足如下条件：

$$\forall j,\sum_{i=1}^{c}u_{ij}=1;\forall i,ju_{ij}\in[0,1];\forall i,\sum_{j=1}^{n}u_{ij}>0$$

目前被广泛使用的聚类准则为取类内加权误差平方和的极小值，即：

$$(\min)J_m(U,V)=\sum_{j=1}^{n}\sum_{i=1}^{c}u_{ij}^m d_{ij}^2(x_j,v_i)$$

其中 V 为聚类中心，m 为加权指数，

$d_{ij}(x_j,v_i)=\|v_i-x_j\|$

算法流程：

(1) 标准化数据矩阵；

(2) 建立模糊相似矩阵，初始化隶属矩阵；

(3) 算法开始迭代，直到目标函数收敛到极小值；

(4) 根据迭代结果，由最后的隶属矩阵确定数据所属的类，显示最后的聚类结果。

3.1 试验数据

实验中，选取专门用于测试分类、聚类算法的国际通用的UCI数据库中的IRIS[13]数据集，IRIS数据集包含150个样本数据，分别取自三种不同 的莺尾属植物setosa、versicolor和virginica的花朵样本,每个数据含有4个属性，即萼片长度、萼片宽度、花瓣长度，单位为cm。 在数据集上执行不同的聚类算法，可以得到不同精度的聚类结果。

3.2 试验结果说明

文中基于前面所述各算法原理及算法流程，用matlab进行编程运算，得到表1所示聚类结果。

表 1 三种聚类方法的实验对比结果

聚类方法	聚错样本数	运行时间/s	平均准确度/(%)
k-means	17	0.146 001	89
层次聚类	51	0.128 744	66
FCM	12	0.470 417	92
SOM	22	5.267 283	86

如表1所示，对于四种聚类算法，按三方面进行比较：(1)聚错样本数：总的聚错的样本数，即各类中聚错的样本数的和；(2)运行时间：即聚类整个 过程所耗费的时间，单位为s；(3)平均准确度：设原数据集有k个类,用ci表示第i类，ni为ci中样本的个数，mi为聚类正确的个数,则mi/ni为 第i类中的精度，则平均精度为：

$$\mathrm{avg}=\frac{1}{k}\sum_{i=1}^{k} m_i/n_i$$

3.3 试验结果分析

四种聚类算法中，在运行时间及准确度方面综合考虑，k-means和FCM相对优于其他。但是，各个算法还是存在固定缺点：k-means聚类算法的初 始点选择不稳定，是随机选取的，这就引起聚类结果的不稳定，本实验中虽是经过多次实验取的平均值，但是具体初始点的选择方法还需进一步研究；层次聚类虽然 不需要确定分类数，但是一旦一个分裂或者合并被执行，就不能修正，聚类质量受限制；FCM对初始聚类中心敏感，需要人为确定聚类数，容易陷入局部最优 解；SOM与实际大脑处理有很强的理论联系。但是处理时间较长，需要进一步研究使其适应大型数据库。