**本周计划：**

1. **学习视觉SLAM，整理算法流程。**
2. **学习SLAM前端，学习图像特征点的提取与匹配，以及计算机器人位姿的相关算法，阅读代码，整理文档。**
3. **学习英语和阅读期刊论文。**

**视觉SLAM**

1. 图像特征点的提取与匹配

1.1. ORB提取特征点

视觉SLAM主要分为视觉前端和优化后端。前端也称为视觉里程计（VO）。它根据相邻图像的信息，估计出粗略的相机运动，给后端较好的初始值。VO的实现方法，按是否需要提取特征，分为特征点法的前端和不提取特征的直接法前端。

基于特征法的前端，长久以来（直到现在）被认为是视觉里程计的主要方法。它运行稳定，对光照、动态物体不敏感，是目前比较成熟的解决方案。

特征点由关键点和描述子两部分组成。关键点是指特征点在图像中的位置，有些特征点还具有朝向、大小等信息。描述子通常是一个向量，按照某种人为设计的方式，描述了关键点周围像素的信息。

SIFT（尺度不变特征变换）是最为经典的一种特征，它充分考虑了在图像变换中出现的光照、尺度、旋转等变化，但随之而来的是极大的计算量。

FAST关键点是计算特别快的一种特征点，它考虑适当降低精度和鲁棒性，提升计算速度。

ORB特征则是目前看来非常具有代表性的图像特征，它改进了FAST检测子不具备方向性的问题，并采用速度极快的二进制描述子BRIEF，使整个图像特征提取的环节大大加速。

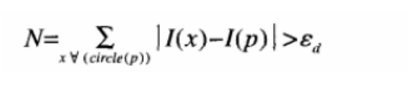
* + 1. ORB特征

提取ORB特征分为两个步骤：

（1）FSAT角点提取：找出图像中的“角点”。相较于原版FAST，ORB中计算了特征点的主方向，为后续的BRIEF描述子增加了旋转不变性。   
（2）BRIEF描述子：对前一步提取出特征点的周围图像区域进行描述。

1.1.2 FAST关键点

FAST是一种角点，主要检测局部像素灰度变化明显的地方，以速度快著称。它的思想是：如果一个像素与它邻域的像素差别较大（过亮或过暗），那它更可能是角点。



其中I（x）为圆周上任意一点的灰度，I（p）为圆心的灰度，Ed为灰度值差得阈值，如果N大于给定阈值，一般为周围圆圈点的四分之三，则认为p是一个特征点。

为了获得更快的结果，还采用了额外的加速办法。如果测试了候选点周围每隔90度角的4个点，应该至少有3个和候选点的灰度值差足够大，否则则不用再计算其他点，直接认为该候选点不是特征点。候选点周围的圆的选取半径是一个很重要的参数，这里为了简单高效，采用半径为3，共有16个周边像素需要比较。为了提高比较的效率，通常只使用N个周边像素来比较，也就是大家经常说的FAST-N。

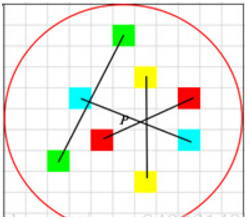
FAST特征点数量很大且不确定，而我们往往希望对图像提取固定数量的特征。因此，在ORB中，对原始的FAST进行了改进。我们可以指定最终要提取的角点数量N，对原始FAST角点分别计算Harris值，然后选取前N个具有最大响应值的角点，作为最终的角点集合。

FAST角点不具有方向信息。而且，由于它取固定半径为3的圆，存在尺度问题：远处看着像是角点的地方，接近后看可能就不是角点了。针对FAST角点不具有方向性和尺度的缺点，ORB添加了尺度和旋转的描述。尺度不变性由构造图像金字塔，并在金字塔的每一层检测角点来实现，而特征的旋转是由灰度质心法实现的。

* + 1. 计算特征描述子

得到特征点后我们需要以某种方式描述这些特征点的属性。这些属性的输出我们称之为该特征点的描述子（Feature DescritorS）.ORB采用BRIEF算法来计算一个特征点的描述子。

BRIEF是一种二进制描述子，它的描述向量由许多个0和1组成，这里的0和1编码了关键点附近两个像素（比如p和q）的大小关系：如果p比q大，则取1，反之就取0。如果我们选取了128个这样的p、q，最后就得到128维由0,1组成的向量。

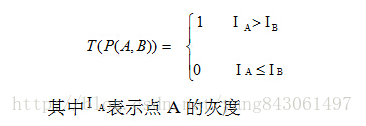


BRIEF算法的核心思想是在关键点P的周围以一定模式选取N个点对，把这N个点对的比较结果组合起来作为描述子。

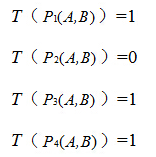
步骤：  
1.以关键点P为圆心，以d为半径做圆O。  
2.在圆O内某一模式选取N个点对。这里为方便说明，N=4，实际应用中N可以取512.  
假设当前选取的4个点对如上图所示分别标记为：

IMG_256

3.定义操作T



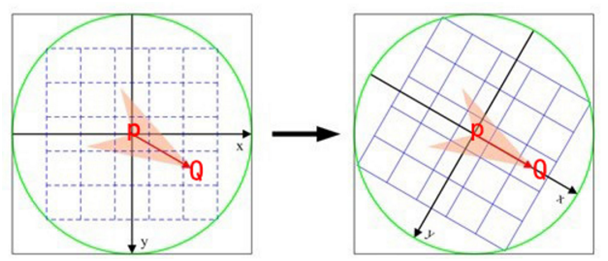
4.分别对已选取的点对进行T操作，将得到的结果进行组合。假如：



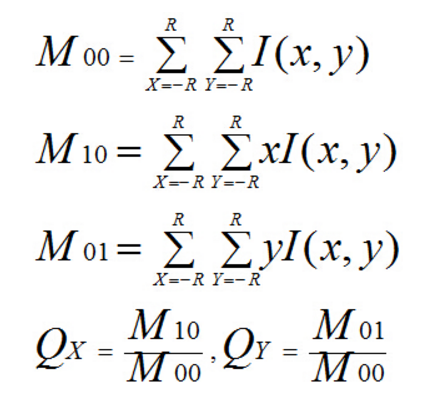
则最终的描述子为：1011

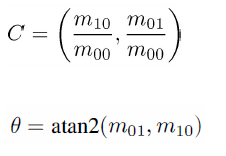
**理想的特征点描述子应该具备的属性：**  
在现实生活中，我们从不同的距离，不同的方向、角度，不同的光照条件下观察一个物体时，物体的大小，形状，明暗都会有所不同。但我们的大脑依然可以判断它是同一件物体。理想的特征描述子应该具备这些性质。即，在大小、方向、明暗不同的图像中，同一特征点应具有足够相似的描述子，称之为描述子的可复现性。  
当以某种理想的方式分别计算描述子时，应该得出同样的结果。即描述子应该对光照（亮度）不敏感，具备尺度一致性（大小 ），旋转一致性（角度）等。  
ORB并没有解决尺度一致性问题，在OpenCV的ORB实现中采用了[图像金字塔](https://www.baidu.com/s?wd=%E5%9B%BE%E5%83%8F%E9%87%91%E5%AD%97%E5%A1%94&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "https://blog.csdn.net/l297969586/article/details/_blank)来改善这方面的性能。ORB主要解决BRIEF描述子不具备旋转不变性的问题。

回顾一下BRIEF描述子的计算过程：  
 在当前关键点P周围以一定模式选取N个点对，组合这N个点对的T操作的结果就为最终的描述子。当我们选取点对的时候，是以当前关键点为原点，以水平方向为X轴，以垂直方向为Y轴建立坐标系。当图片发生旋转时，坐标系不变，同样的取点模式取出来的点却不一样，计算得到的描述子也不一样，这是不符合我们要求的。因此我们需要重新建立坐标系，使新的坐标系可以跟随图片的旋转而旋转。这样我们以相同的取点模式取出来的点将具有一致性。  
打个比方，我有一个印章，上面刻着一些直线。用这个印章在一张图片上盖一个章子，图片上分处直线2头的点将被取出来。印章不变动的情况下，转动下图片，再盖一个章子，但这次取出来的点对就和之前的不一样。为了使2次取出来的点一样，我需要将章子也旋转同一个角度再盖章。（取点模式可以认为是章子上直线的分布情况）  
 ORB在计算BRIEF描述子时建立的坐标系是以关键点为圆心，以关键点和取点区域的形心的连线为X轴建立2维坐标系。



P为关键点。圆内为取点区域，每个小格子代表一个像素。现在我们把这块圆心区域看做一块木板，木板上每个点的质量等于其对应的像素值。根据积分学的知识我们可以求出这个密度不均匀木板的质心Q。计算公式如下。其中R为圆的半径。





我们知道圆心是固定的而且随着物体的旋转而旋转。当我们以PQ作为坐标轴时，在不同的旋转角度下，我们以同一取点模式取出来的点是一致的。这就解决了旋转一致性的问题。  
BRIEF中，采用了9x9的高斯算子进行滤波，可以一定程度上解决噪声敏感问题，但一个滤波显然是不够的。ORB中提出，利用积分图像来解决：在31x31的窗口中，产生一对随机点后，以随机点为中心，取5x5的子窗口，比较两个子窗口内的像素和的大小进行二进制编码，而非仅仅由两个随机点决定二进制编码。（这一步可有积分图像完成）  
特征点的匹配  
ORB算法最大的特点就是计算速度快 。 这首先得益于使用FAST检测特征点，FAST的检测速度正如它的名字一样是出了名的快。再次是使用BRIEF算法计算描述子，该描述子特有的2进制串的表现形式不仅节约了存储空间，而且大大缩短了匹配的时间。

例如特征点A、B的描述子如下。  
A：10101011  
B：10101010  
我们设定一个阈值，比如80%。当A和B的描述子的相似度大于90%时，我们判断A,B是相同的特征点，即这2个点匹配成功。在这个例子中A,B只有最后一位不同，相似度为87.5%，大于80%。则A和B是匹配的。  
我们将A和B进行异或操作就可以轻松计算出A和B的相似度。而异或操作可以借组硬件完成，具有很高的效率，加快了匹配的速度。

特征的匹配是针对特征描述子的进行的，上面提到特征描述子通常是一个向量，两个特征描述子的之间的距离可以反应出其相似的程度，也就是这两个特征点是不是同一个。根据描述子的不同，可以选择不同的距离度量。如果是浮点类型的描述子，可以使用其欧式距离；对于二进制的描述子（BRIEF）可以使用其汉明距离（两个不同二进制之间的汉明距离指的是两个二进制串不同位的个数）。

两个特征点的描述子在向量空间上的距离很近，就可以认为他们是同一个特征点。

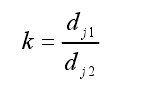
汉明距离是以理查德•卫斯里•汉明的名字命名的。在信息论中，两个等长字符串之间的汉明距离是两个字符串对应位置的不同字符的个数。换句话说，它就是将一个字符串变换成另外一个字符串所需要替换的字符个数。例如：  
1011101 与 1001001 之间的汉明距离是 2。  
2143896 与 2233796 之间的汉明距离是 3。  
"toned" 与 "roses" 之间的汉明距离是 3。  
给予两个任何的字码，10001001和10110001，即可决定有多少个相对位是不一样的。在此例中，有三个位不同。要决定有多少个位不同，只需将xor运算加诸于两个字码就可以，并在结果中计算有多个为1的位。例如：  
10001001  
Xor 10110001  
00111000  
两个字码中不同位值的数目称为汉明距离(Hamming distance) 。

**1.2 特征点的匹配**

有了计算描述子相似度的方法，那么在特征点的集合中如何寻找和其最相似的特征点，这就是特征点的匹配了。最简单直观的方法就是暴力匹配法即对每一个特征点与所有的特征点测量描述子的距离，然后排序，取最近的一个作为匹配点。描述子的距离表述了两个特征之间的相似程度。然而，当特征点很多的时，暴力匹配的运算量就变得很大，特别是我们想匹配一个帧和一张地图的时候。这不符合我们在SLAM中的实时性需求。此时快速近似最近邻算法更加适合于匹配点数量极多的情况。

K近邻匹配，在匹配的时候选择K个和特征点最相似的点，如果这K个点之间的区别足够大，则选择最相似的那个点作为匹配点，通常选择K = 2，也就是最近邻匹配。对每个匹配返回两个最近邻的匹配（另外一帧图像上最近的两个点），如果第一匹配和第二匹配距离比率足够大（向量距离足够远），则认为这是一个正确的匹配，比率的阈值通常在2左右

在计算特征点 x 在另一幅图像中的投影时，计算与特征点Xi 距离最小的两个特征Xj1、Xj2 ，分别为最近点和次近点，距离分别为dj1和dj2 ，

 如果 k 大于设定值的话，则认为最近点Xj1 (第一个近邻点）为匹配点，发现 k 的取值在 0.4~0.65 之间比较合适。

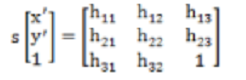
|  |
| --- |
| // 1. 初始化  vector<KeyPoint> keypoints1, keypoints2;  Mat descriptors1, descriptors2;  Ptr<ORB> orb = ORB::create();  // 2. 提取特征点  orb->detect(img1, keypoints1);  orb->detect(img2, keypoints2);  // 3. 计算特征描述符  orb->compute(img1, keypoints1, descriptors1);  orb->compute(img2, keypoints2, descriptors2);  // 4. 对两幅图像的BRIEF描述符进行匹配，使用 knnMatches，Hamming距离作为参考 const float minRatio = 1.f / 1.5f;  const int k = 2;  vector<vector<DMatch>> knnMatches;  matcher->knnMatch(leftPattern->descriptors, rightPattern->descriptors, knnMatches, k);  for (size\_t i = 0; i < knnMatches.size(); i++) {  const DMatch& bestMatch = knnMatches[i][0];  const DMatch& betterMatch = knnMatches[i][1];  float distanceRatio = bestMatch.distance / betterMatch.distance;  if (distanceRatio < minRatio)  matches.push\_back(bestMatch);  } |

将不满足的最近邻的匹配之间距离比率大于设定的阈值（1/1.5）匹配剔除。

**消除误匹配**

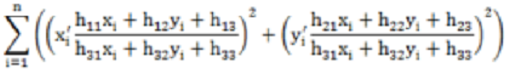
另外还可采用随机采样一致性（RANSAC）来过滤掉错误的匹配，该方法利用匹配点计算两个图像之间单应矩阵，然后利用重投影误差来判定某一个匹配是不是正确的匹配。

RANSAC算法是寻找一个最佳单应性矩阵，矩阵大小为3\*3，目的是寻找最优的参数矩阵，使得满足该矩阵的数据点最多，通常设h33=1，由于单应性矩阵有8个未知参数，所以需要8个线性方程求解，对应到点位置信息上，则至少包含4组匹配点。



其中，（x,y）表示目标图像的角点位置，（x’,y’）为场景图像角点位置，s为尺度函数。

RANSAC算法从匹配数据集中随机抽取出4个样本并保证这4个样本之间不共线，计算出单应性矩阵，然后用这个模型去测试所有的数据，并计算满足这个模型数据点的个数以及重投影误差（代价函数），若此模型为最优模型，则对应的代价函数最小。

 OpenCV中封装了求解单应矩阵的方法findHomography,可以为该方法设定一个重投影误差的阈值，可以得到一个向量mask来指定那些是符合该重投影误差的匹配点对，以此来剔除错误的匹配，代码如下：

|  |
| --- |
| const int minNumbermatchesAllowed = 8;  if (matches.size() < minNumbermatchesAllowed)  return;  //Prepare data for findHomography  vector<Point2f> srcPoints(matches.size());  vector<Point2f> dstPoints(matches.size());  for (size\_t i = 0; i < matches.size(); i++) { srcPoints[i] = rightPattern->keypoints[matches[i].trainIdx].pt; dstPoints[i] = leftPattern->keypoints[matches[i].queryIdx].pt; }  //find homography matrix and get inliers mask  vector<uchar> inliersMask(srcPoints.size());  homography = findHomography(srcPoints, dstPoints, CV\_FM\_RANSAC, reprojectionThreshold, inliersMask);  vector<DMatch> inliers;  for (size\_t i = 0; i < inliersMask.size(); i++){  if (inliersMask[i])  inliers.push\_back(matches[i]);  }  matches.swap(inliers);const int minNumbermatchesAllowed = 8;  if (matches.size() < minNumbermatchesAllowed)  return;  //Prepare data for findHomography  vector<Point2f> srcPoints(matches.size());  vector<Point2f> dstPoints(matches.size());  for (size\_t i = 0; i < matches.size(); i++) {  srcPoints[i] = rightPattern->keypoints[matches[i].trainIdx].pt;  dstPoints[i] = leftPattern->keypoints[matches[i].queryIdx].pt;  }  //find homography matrix and get inliers mask  vector<uchar> inliersMask(srcPoints.size());  homography = findHomography(srcPoints, dstPoints, CV\_FM\_RANSAC, reprojectionThreshold, inliersMask);  vector<DMatch> inliers;  for (size\_t i = 0; i < inliersMask.size(); i++){  if (inliersMask[i]) inliers.push\_back(matches[i]);  }  matches.swap(inliers); |

* 1. **计算机器人的位姿**

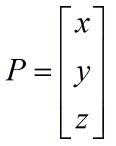
**1.2.1位姿表示**

位姿代表位置和姿态。任何一个刚体在空间坐标系(OXYZ)中可以用位置和姿态来精确、唯一表示其位置状态。

位置：x、y、z坐标

姿态：刚体与OX轴的夹角rx、与OY轴的夹角ry、与OZ轴的夹角rz

假设基坐标系为OXYZ，刚体坐标系为O`X`Y`Z`。对于机器人而言，空间中的任何一个点都必须要用上述六个参数明确指定，即(x,y,z,rx,ry,rz)，即便(x,y,z)都一样，(rx,ry,rz)不同代表机器人以不同的姿态去到达同一个点。

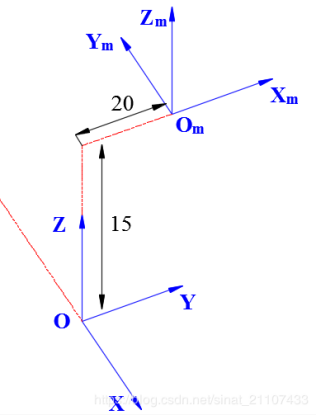
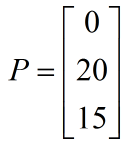
刚体的位置可以用一个3x1的矩阵来表示，即刚体坐标系中心O`在基坐标系中的位置，即。

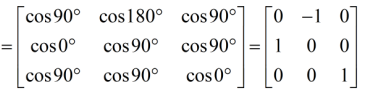
刚体的姿态可以用一个3x3的矩阵来表示，即刚体坐标系在基坐标系中的姿态，即



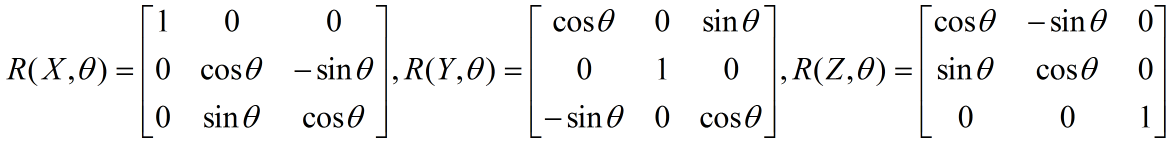
其中，第一列表示刚体坐标系的O`X`轴在基坐标系的三个轴方向上的分量，称为单位主矢量。同理，第二列和第三列分别是刚体坐标系的O`Y`轴和O`Z`轴在基坐标系的三个轴方向上的分量。

举个例子，在下图中，刚体M沿坐标系O中平移了（0,20,15），绕Z轴旋转了90度，因此刚体M在坐标系O的位姿可描述为：

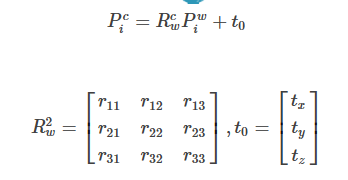
 



根据上面的例子，很容易得到，刚体坐标系绕X轴（Y轴、Z轴）旋转角度θ后的姿态矩阵为：



位姿估计是机器人视觉领域一个基本问题。若已获得合作标志在摄像机坐标系下的坐标值，一般情况可采用奇异值分解或最小二乘的方法求解位姿信息中的旋转矩阵（R）与平移矩阵（t），满足，



其中下标c表示Camera，w表示World。

在得到匹配点之后，并且知道两组匹配点的深度，于是就是3D-3D的匹配这种情况下 R,t 的估计是有解析解（闭式解）的。闭式解可以稍加推导，假如我们有一组配对好的3D点（比如上述我们对两幅RGB\_D图进行了匹配）：



现在想找一个欧式变换R,t，使得：



对于这个问题可以用**迭代最近点（Iterative Closest Point）ICP**算法求解，3D-3D位姿估计中并没有出现相机模型，也就是说，仅考虑两组3D点之间的变换时，和相机没有关系。在激光SlAM中也会碰到ICP 不过由于激光数据特征不够丰富，我们无法知道两个点集之间的匹配关系，只能认为距离最近的两个点为同一个，所以这个方法就是迭代最近点。而在视觉中，特征点为我们提供了较好的匹配关系，所以整个问题就变的简单了，在RGB\_D SLAM中，可以用这种方法估计相机位姿。

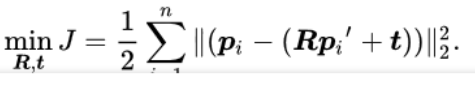
Icp的求解也分为两种，利用线性代数的求解（主要是SVD），以及非线性优化方式求解（类似于Bundle Adjustment）。

**SVD方法**

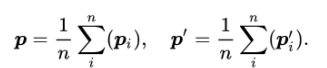
一开始不知道R,t，所以算一个误差再求他的最小化，我么先定义第i 对点的误差项：



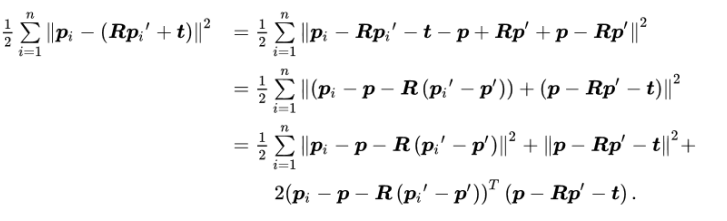
然后，构建最小二乘问题求使误差平方和达到极小的R,t：



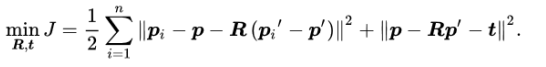
下面来推导他的求解方法。首先，定义两组的质心：



要注意质心是没有下标的。然后，在误差函数里进行如下的处理：



注意到 交叉部分在求和之后是为零的，因此优化目标函数可以简化为：



经过观察可以发现左边只和旋转矩阵有关系，而右边既有R也有t，但只和质心有关系，只要我们得到R，令第二项为零就能得到t.于是就能得到t 。于是ICP算法，可以分为以下三个步骤求解：

1. 计算两组点的质新位置P，P'，然后就散每个点的去质心坐标：



1. 根据以下优化问题计算旋转矩阵：

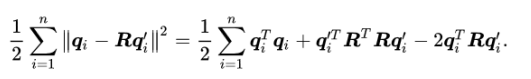


1. 根据第二步的R计算t:

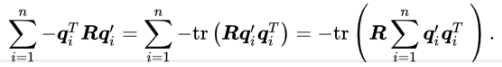


上述可以得到只要得到了旋转矩阵R,平移矩阵是非常容易得到的，那R是怎么得到的呢？

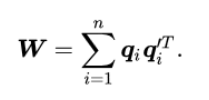
展开关于R的误差项，得：



注意到第一项与R无关，第二项由于，也与R无关，因此实际上优化目标函数变为：



然后介绍怎样用SVD解出上述问题中最优解的R，关于最优解证明比较复杂，这里只讨论结果。首先定义矩阵：



W是一个3×3的矩阵，对W进行SVD分解，得；



其中∑是奇异值组成的对角矩阵，对角线元素从大到小排列，而U和V为对角矩阵。当W满秩时，R为：



解出R后，按上述公式求出t即可。

**1.4图优化**

SLAM问题的处理方法主要分为滤波和图优化两类。滤波的方法中常见的是扩展卡尔曼滤波、粒子滤波、信息滤波等，这类SLAM问题是递增的、实时的处理数据并矫正机器人位姿。比如基于粒子滤波的SLAM的处理思路是假设机器人知道当前时刻的位姿，利用编码器或者IMU之类的惯性导航又能够计算下一时刻的位姿，然而这类传感器有累计误差，所以再将每个粒子的激光传感器数据或者图像特征对比当前建立好的地图中的特征，挑选和地图特征匹配最好的粒子的位姿当做当前位姿，如此往复。

在图优化的方法中（graph-based slam），处理数据的方式就和滤波的方法不同了，它不是在线的纠正位姿，而是把所有数据记下来，最后一次性算账。

所谓的图优化，就是把一个常规的优化问题，以图（Graph）的形式来表述。

图是由顶点（Vertex）和边（Edge）组成的结构，而图论则是研究图的理论。我们记一个图为G={V,E}，其中V为顶点集，E为边集。

顶点就是普通的点，　边是什么呢？一条边连接着若干个顶点，表示顶点之间的一种关系。边可以是有向的或是无向的，对应的图称为有向图或无向图。边也可以连接一个顶点（Unary Edge，一元边）、两个顶点（Binary Edge，二元边）或多个顶点（Hyper Edge，多元边）。最常见的边连接两个顶点。当一个图中存在连接两个以上顶点的边时，称这个图为超图（Hyper Graph）。而SLAM问题就可以表示成一个超图。

SLAM的核心是根据已有的观测数据，计算机器人的运动轨迹和地图。假设在时刻k，机器人在位置xk处，用传感器进行了一次观测，得到了数据zk。传感器的观测方程为：

zk=h(xk)

由于误差的存在，zk不可能精确地等于h(xk)，于是就有了误差：

ek=zk−h(xk)

那么，如果我们以xk为优化变量，以为目标函数，就可以求得xk的估计值，进而得到我们想要的东西了。这实际上就是用优化来求解SLAM的思路。

优化变量xk，观测方程zk=h(xk)等等，它们具体是什么东西呢？

这个取决于我们的参数化（parameterazation）。x可以是一个机器人的Pose（6自由度下为 4×4的变换矩阵T 或者 3自由度下的位置与转角[x,y,θ]，也可以是一个空间点（三维空间的[x,y,z]或二维空间的[x,y]）。

相应的，观测方程也有很多形式，如：

机器人两个Pose之间的变换；

机器人在某个Pose处用激光测量到了某个空间点，得到了它离自己的距离与角度；

机器人在某个Pose处用相机观测到了某个空间点，得到了它的像素坐标；

同样，它们的具体形式很多样化，这允许我们在讨论slam问题时，不局限于某种特定的传感器或姿态表达方式。

在图中，以顶点表示优化变量，以边表示观测方程。由于边可以连接一个或多个顶点，所以我们把它的形式写成更广义的 zk=h(xk1,xk2,…)，以表示不限制顶点数量的意思。

机器人在某个Pose处用相机观测到了某个空间点，得到了它的像素坐标；——Binary Edge，顶点为一个3D Pose：T和一个空间点x=[x,y,z]T，观测数据为像素坐标z=[u,v]T。那么观测方程为：

z=C(Rx+t)

C为相机内参，R,t为旋转矩阵和平移矩阵。

图优化的步骤大概为：

· 选择你想要的图里的节点与边的类型，确定它们的参数化形式；

· 往图里加入实际的节点和边；

· 选择初值，开始迭代；

· 每一步迭代中，计算对应于当前估计值的雅可比矩阵和海塞矩阵；

· 求解稀疏线性方程HkΔx=−bk，得到梯度方向；

· 继续用GN或LM进行迭代。如果迭代结束，返回优化值。

　　实际上，g2o能帮我们做好第3-6步，我们要做的只是前两步而已。

**Gmapping建图**

gmapping是一个比较完善的地图构建开源包，使用激光和里程计的数据来生成二维地图。输入的是激光数据以及机器人当前的位姿。

 1、Gmapping是基于滤波SLAM框架的常用开源SLAM算法。

      2、Gmapping基于RBpf粒子滤波算法，即将定位和建图过程分离，先进行定位再进行建图。

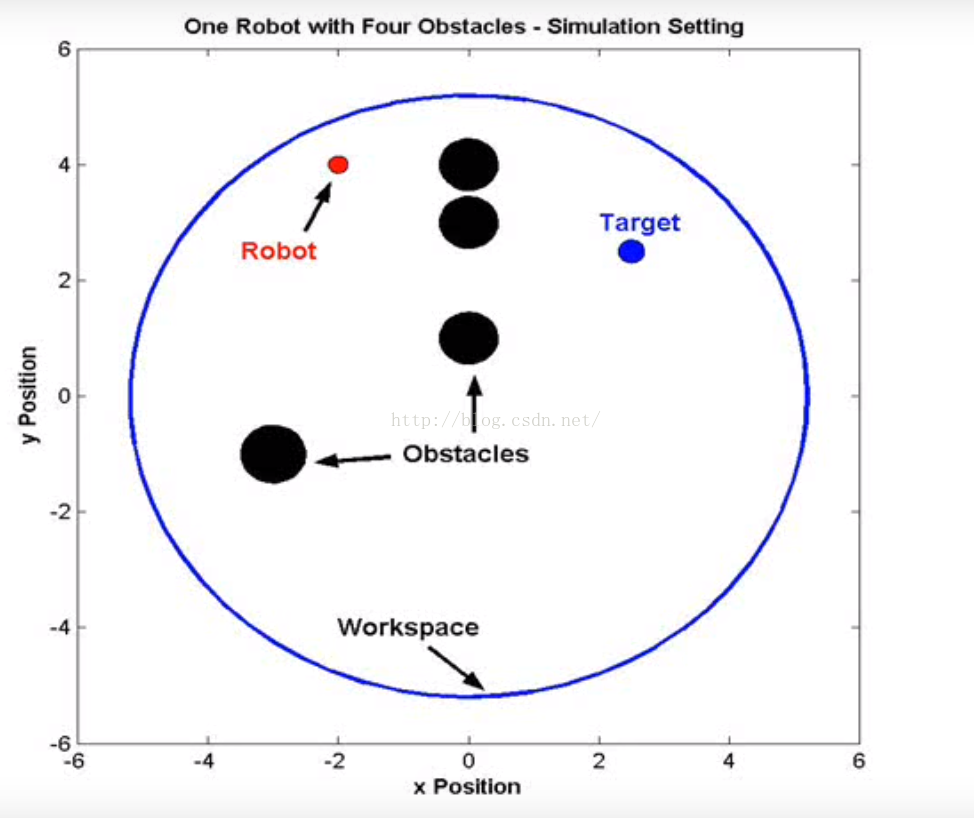
      3、Gmapping在RBpf算法上做了两个主要的改进：改进提议分布和选择性重采样。

 优点：Gmapping可以实时构建室内地图，在构建小场景地图所需的计算量较小且精度较高。相比Hector SLAM对激光雷达频率要求低、鲁棒性高（Hector 在机器人快速转向时很容易发生错误匹配，建出的地图发生错位，原因主要是优化算法容易陷入局部最小值）；而相比Cartographer在构建小场景地图时，Gmapping不需要太多的粒子并且没有回环检测因此计算量小于Cartographer而精度并没有差太多。Gmapping有效利用了车轮里程计信息，这也是Gmapping对激光雷达频率要求低的原因：里程计可以提供机器人的位姿先验。而Hector和Cartographer的设计初衷不是为了解决平面移动机器人定位和建图，Hector主要用于救灾等地面不平坦的情况，因此无法使用里程计。而Cartographer是用于手持激光雷达完成SLAM过程，也就没有里程计可以用。

   缺点：随着场景增大所需的粒子增加，因为每个粒子都携带一幅地图，因此在构建大地图时所需内存和计算量都会增加。因此不适合构建大场景地图。并且没有回环检测，因此在回环闭合时可能会造成地图错位，虽然增加粒子数目可以使地图闭合但是以增加计算量和内存为代价。所以不能像Cartographer那样构建大的地图，虽然论文生成几万平米的地图，但实际我们使用中建的地图没有几千平米时就会发生错误。Gmapping和Cartographer一个是基于滤波框架SLAM另一个是基于优化框架的SLAM，两种算法都涉及到时间复杂度和空间复杂度的权衡。Gmapping牺牲空间复杂度保证时间复杂度，这就造成Gmapping不适合构建大场景地图，试想一下你要构建200乘200米的环境地图，栅格分辨率选择5厘米，每个栅格占用一字节内存，那么一个粒子携带的地图就需要16M内存，如果是100个粒子就需要1.6G内存。如果地图变成500乘500米，粒子数为200个，可能电脑就要崩溃了。翻看Cartographer算法，优化相当于地图中只用一个粒子，因此存储空间比较Gmapping会小很多倍，但计算量大，一般的笔记本很难跑出来好的地图，甚至根本就跑不动。优化图需要复杂的矩阵运算，这也是谷歌为什么还有弄个ceres库出来的原因。

**人工势场法路径规划**

人工势场法是局部路径规划的一种比较常用的方法。这种方法假设机器人在一种虚拟力场下运动。

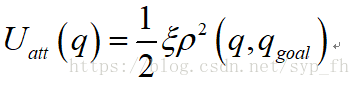


如图所示，机器人在一个二维环境下运动，图中指出了机器人，障碍和目标之间的相对位置。

人工势场包括引力场和斥力场，其中目标点对物体产生引力，引导物体朝向其运动（这一点有点类似于A\*算法中的启发函数h）。障碍物对物体产生斥力，避免物体与之发生碰撞。物体在路径上每一点所受的合力等于这一点所有斥力和引力的和。这里的关键是如何构建引力场和斥力场。下面我们分别讨论一下：

**引力场：**

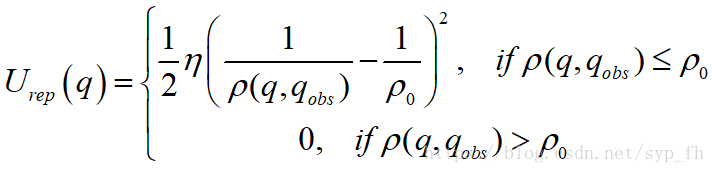
常用的引力函数：

 （1）

这里的ε是尺度因子.ρ(q,q\_goal)表示物体当前状态与目标的距离。引力场有了，那么引力就是引力场对距离的导数（类比物理里面W=FX）：

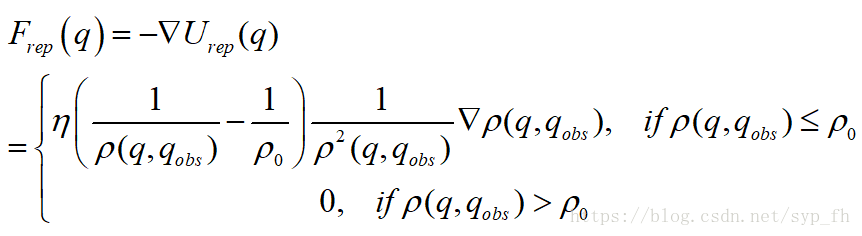
IMG_256 （2）

**斥力场：**

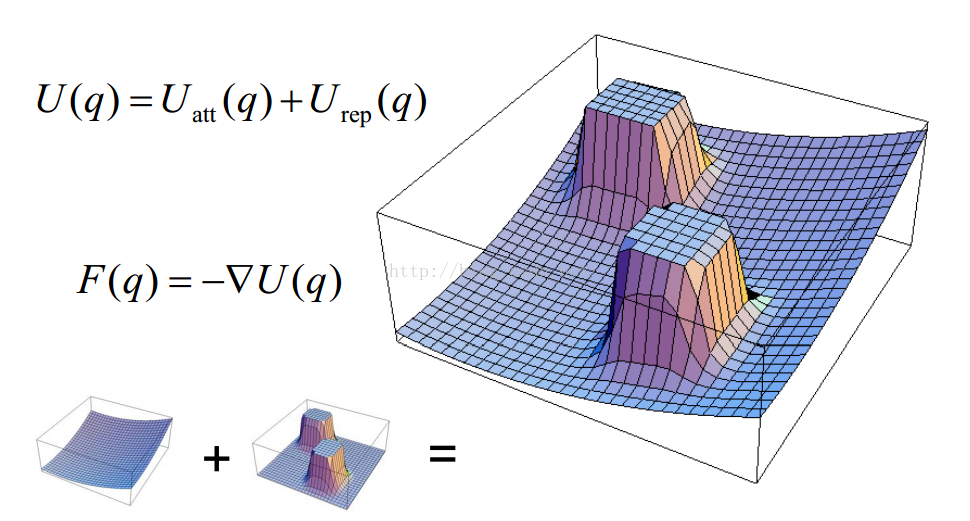
（3）

公式（3）是传统的斥力场公式。公式中η是斥力尺度因子，ρ（q,q\_obs）代表物体和障碍物之间的距离。ρ\_0代表每个障碍物的影响半径。换言之，离开一定的距离，障碍物就对物体没有斥力影响。

斥力就是斥力场的梯度：



总的场就是斥力场合引力场的叠加，也就是U=U\_att+U\_rep,总的力也是对对应的分力的叠加，如下图所示：



二、存在的问题

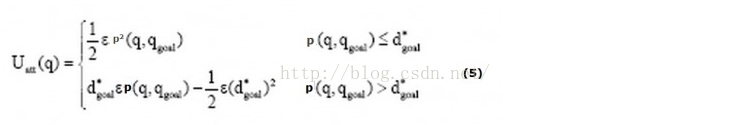
（a） 当物体离目标点比较远时，引力将变的特别大，相对较小的斥力在甚至可以忽略的情况下，物体路径上可能会碰到障碍物

（b）当目标点附近有障碍物时，斥力将非常大，引力相对较小，物体很难到达目标点

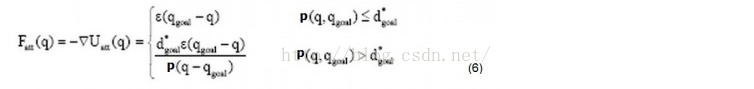
（c）在某个点，引力和斥力刚好大小相等，方向想反，则物体容易陷入局部最优解或震荡

## 三、各种改进版本的人工势场法

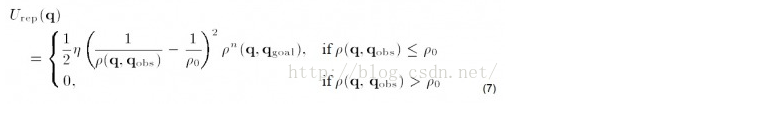
（a）对于可能会碰到障碍物的问题，可以通过修正引力函数来解决，避免由于离目标点太远导致引力过大



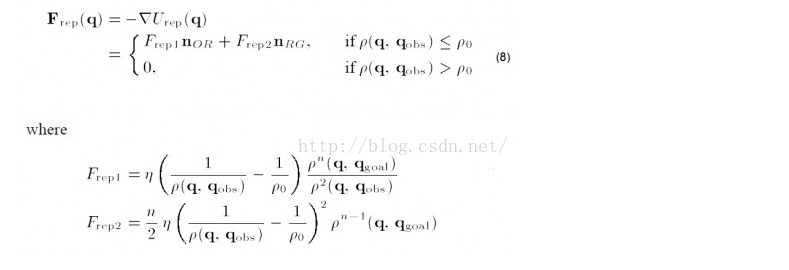
和（1）式相比，（5）式增加了范围限定。d\*\_goal 给定了一个阈值限定了目标和物体之间的距离。对应的梯度也就是引力相应变成：



**（b）**目标点附近有障碍物导致目标不可达的问题，引入一种新的斥力函数：



这里在原有斥力场的基础上，加上了目标和物体距离的影响,（n是正数，我看到有篇文献上n=2）。直观上来说，物体靠近目标时，虽然斥力场要增大，但是距离在减少，所以在一定程度上可以起到对斥力场的拖拽作用相应斥力变成：



（c）局部最优问题是一个人工势场法的一个大问题，这里可以通过加一个随机扰动，让物体跳出局部最优值。类似于梯度下降法局部最优值的解决方案。