1. **计算最优的粒子**

optimize 调用了 score 这个函数 （计算粒子得分）

在score 函数里，首先计算障碍物的坐标phit，然后将phit转换成网格坐标iPhit

计算光束上与障碍物相邻的非障碍物网格坐标pfree,pfrree由phit沿激光束方向移动一个网格的距离得到，将pfree转换成网格坐标ipfree（增量，并不是实际值）

在iphit 及其附近8个（m\_kernelSize:default=1）栅格（pr,对应自由栅格为pf）搜索最优可能是障碍物的栅格。

最优准则： pr 大于某一阈值，pf小于该阈值，且pr栅格的phit的平均坐标与phit的距离bestMu最小。

得分计算： s +=exp(-1.0/m\_gaussianSigma\*bestMu\*besMu) 参考NDT算法,距离越大，分数越小，分数的较大值集中在距离最小值处，符合正态分布模型

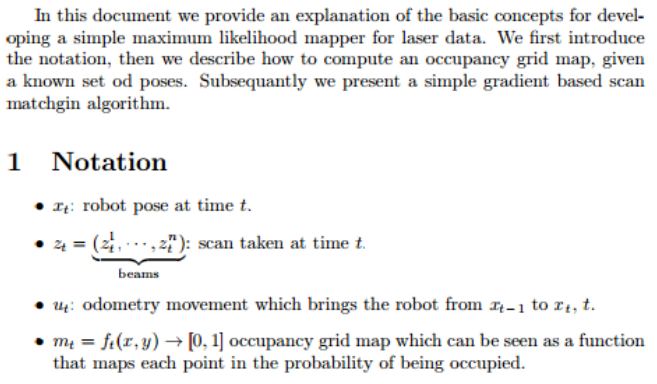
至此 score 函数结束并返回粒子（currentPose）得分，然后回到optimize函数

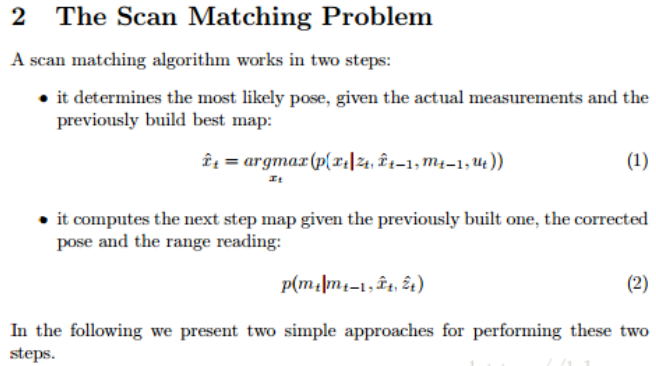
optimize 干的事就是 currentPose 的位姿进行微调，前、后、左、右、左转、右转 共6次，然后选取得分最高的位姿，返回最终的得分

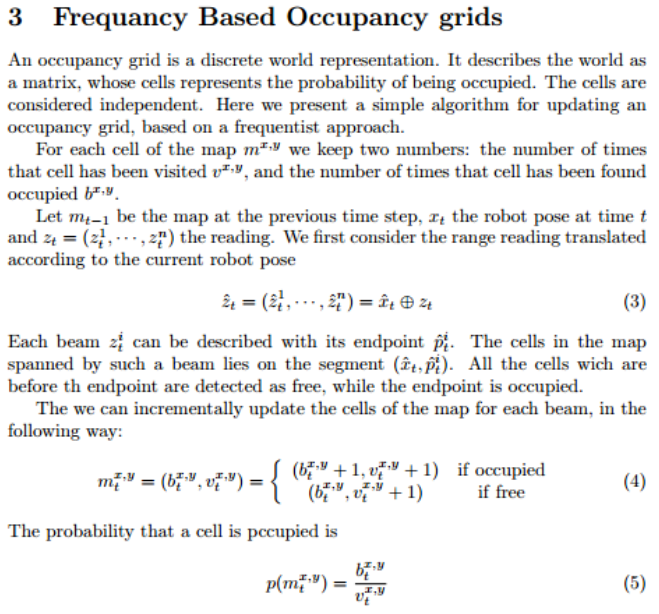
\*/

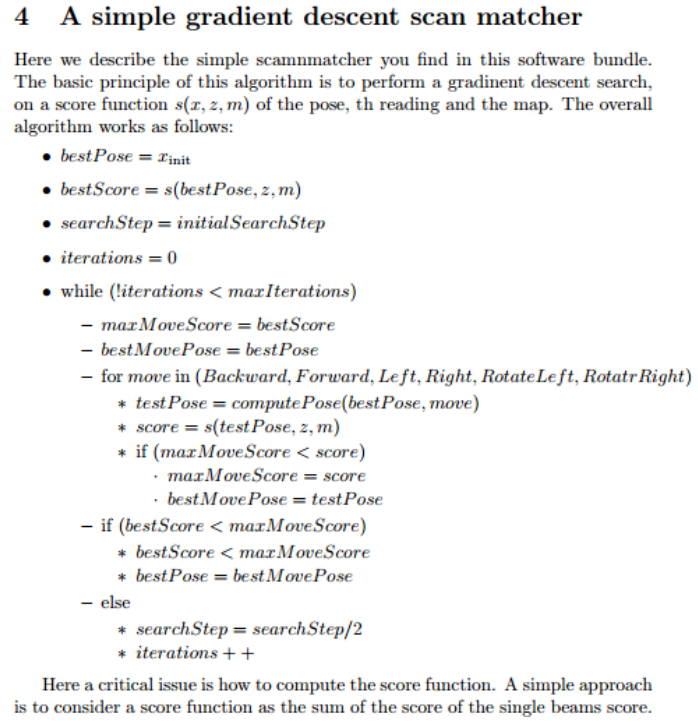
1. **An incomplete scan matching tutorial**

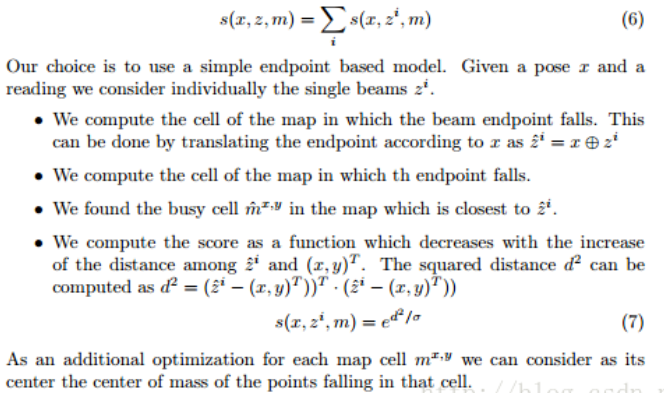
Gmapping











1. **分层存储器的结构**

内存按照双层的结果进行存储

存储体结构函数

PointAccumulator

struct PointAccumulator{

typedef point<float> FloatPoint;

/\* before

PointAccumulator(int i=-1): acc(0,0), n(0), visits(0){assert(i==-1);}

\*/

/\*after begin\*/

PointAccumulator(): acc(0,0), n(0), visits(0){}

PointAccumulator(int i): acc(0,0), n(0), visits(0){assert(i==-1);}

/\*after end\*/

inline void update(bool value, const Point& p=Point(0,0));

/\* 从世界坐标转换到地图坐标，需要做平均值，这样更加精确 \*/

inline Point mean() const

{

return 1./n\*Point(acc.x, acc.y);

}

inline operator double() const

{

return visits?(double)n\*SIGHT\_INC/(double)visits : -1;

}

inline void add(const PointAccumulator& p) {acc=acc+p.acc; n+=p.n; visits+=p.visits; }

static const PointAccumulator& Unknown();

static PointAccumulator\* unknown\_ptr;

FloatPoint acc;

int n, visits;

inline double entropy() const;

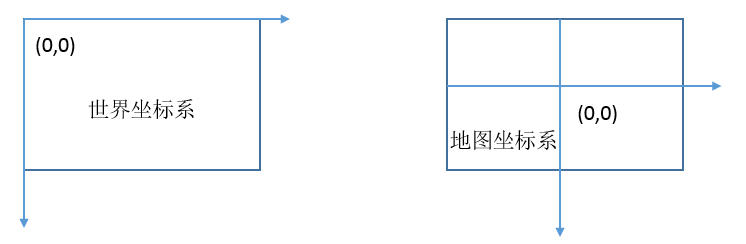
};

m\_storage 是一个双层的结构体 HierarchicalArray2D

1. **地图**

世界坐标

地图坐标，它们的初始位置是不一样的



1. **需要知道的参数**