机器人栅格地图优化调研

1. **栅格地图介绍**

机器人通过激光SLAM算法构建好场景地图之后，为了保证机器人完成导航、路劲规划等功能，我们通常会对场景地图进行修改或优化，对应的场景地图通常为栅格地图（xxx.pgm 格式的文件）。

**1.1 栅格地图的属性**

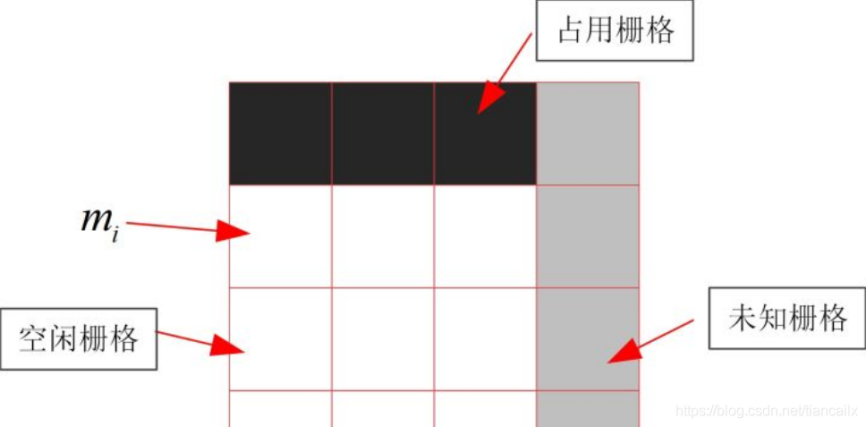
栅格地图是用一个个栅格组成的网格来代表地图. 栅格里可以存储不同的数值, 代表栅格地图的属性：

白色代表空闲，表示可通过区域，其存储的值为 0；

黑色代表占用，表示不可通过区域，其存储的值为 100；

灰色代表未知，表示未知区域，其存储的值为 -1．

栅格地图的示意图如下图所示：



**1.2 栅格地图的数据格式**

机器人通过激光SLAM算法构建好场景地图之后，会生成xxx.pgm和xxx.yaml两个文件。其中栅格地图的信息存储在xxx.pgm格式文件中,地图的路径以及参数设置存储在xxx.yaml文件中：

**1.2.1 yaml格式文件的内容：**

image: /home/xxx/xxx.pgm

resolution: 0.050000

origin: [-10.000000, -10.000000, 0.000000]

negate: 0

occupied\_thresh: 0.65

free\_thresh: 0.196

参数： image：地图的路径信息；

resolution：地图的分辨率，单位是meters/pixel ；

origin：地图的初始位姿；

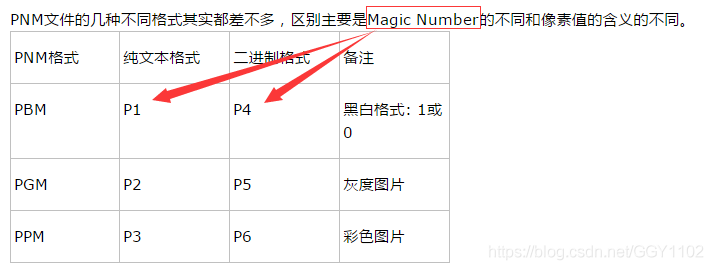
Negate：无论白色或黑色，占用或自由，语义应该是颠倒的；

occupied\_thresh：像素占用率大于这个阈值则认为完全占用；

free\_thresh：像素占用率比该阈值小被则认为完全自由；

**1.2.2 pgm格式文件的内容：**

pgm是便携式灰度图像格式(Portable Gray Map)的缩写，它是灰度图像格式中一种最简单的格式标准。pgm格式文件由两部分组成,文件头部分和数据部分。此外，PGM数据部分还分为P2和P5两种不同的格式，如图所示。



**1.2.2.1 文件头信息:**

1.PGM文件的格式类型;

2.图像的宽度;

3.图像的高度;

4.图像灰度值的可能的最大值;

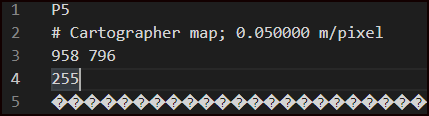
文件头的这四部分信息都是以ASCII码形式存储的,可以直接将PGM格式的文件在记事本中打开查看文件头的信息。

**1.2.2.2 数据部分：**

  数据部分存储每个像素的灰度值，按照图像从上到下，从左到右的顺序依次存储。像素灰度值的表示形式有P2格式和P5格式：

**P5格式：**

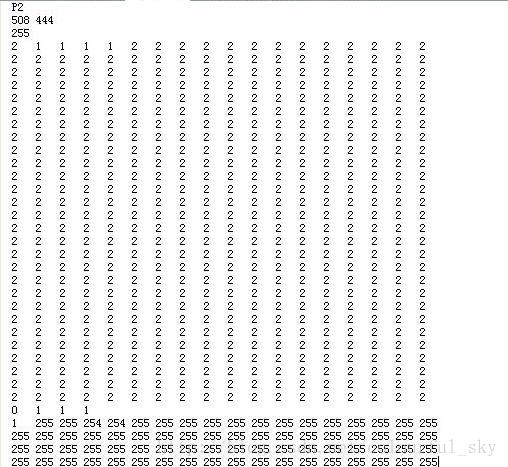
  P5格式的文件，每个像素用二进制表示，且没有行列的标识和区分，从开始第一个像素到最后一个像素都直接依次连续存储，中间没有空格，行与行直之间也没有空格。假设有一幅P5格式图像，灰度值可能的最大值为255，因此该图像每个像素使用一个字节表示即可。比如它的第一行第一列像素值为100，第一行第二列像素值为200，依次后面排。此外，因为是以二进制表示,所以数据部分在记事本中打开后看到的将会是乱码，如图所示：



**P2格式**

  P2格式的文件，每个像素使用字符串来表示。比如一副P2格式图像，灰度值可能的最大值为255，它的第一行第一列像素值为100，那么该图像图像每个像素使用3个ASCII字符表示。第一行第一列数据为ASII表示的”100”，即100的每一个数据位由一个字节表示，百位1用一个字节，十位0用一个字节，各位0用一个字节。

  P2格式和P5格式不同，每个像素数据之间需要用一个空格符分开存储，在图像的每一行数据结束时需要换行。还有一点需要注意，P2格式文件数据部分当数据超过70个字节的时候，会自动换行。也就是说图像的每一行数据末尾，或者存储时超过70个字节时,需要进行换行。



**1.3 栅格地图的数据类型**

在ROS中的数据类型中通过OccupancyGrid(占据网格)的数据类型表示，OccupancyGrid由一个xxx.yaml格式的元数据文件和图片格式的地图数据文件组成。

$ rosmsg show nav\_msgs/OccupancyGrid

std\_msgs/Header header          # 数据的消息头

  uint32 seq                    # 数据的序号

  time stamp                    # 数据的时间戳

  string frame\_id               # 地图的坐标系

nav\_msgs/MapMetaData info       # 地图的一些信息

  time map\_load\_time            # 加载地图的时间

  float32 resolution            # 地图的分辨率，一个格子代表着多少米，一般为0.05，[m/cell]

  uint32 width                  # 地图的宽度，像素的个数, [cells]

  uint32 height                 # 地图的高度，像素的个数, [cells]

  geometry\_msgs/Pose origin     # 地图左下角的格子对应的物理世界的坐标，[m, m, rad]

    geometry\_msgs/Point position

      float64 x

      float64 y

      float64 z

    geometry\_msgs/Quaternion orientation

      float64 x

      float64 y

      float64 z

      float64 w

# 地图数据，优先累加行，从（0,0）开始。占用值的范围为[0,100]，未知为-1。

int8[] data

消息可以分为3个部分，消息头header，地图信息info，地图数据data。

地图信息info储存了地图相关的信息，包括加载地图的时间，地图的分辨率，地图的宽度与高度，以及地图左下角栅格对应的物理坐标．

地图本身是有像素坐标的，其像素坐标系为坐下角为(0,0) 的坐标系．通过左下角栅格对应的物理坐标origin以及分辨率，再通过 像素 \* 分辨率 + origin , 将像素坐标转成物理世界的坐标，从而确定了整个地图的物理坐标．

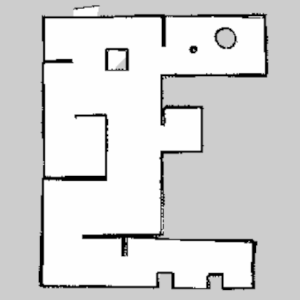
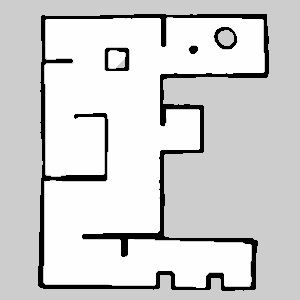
地图数据data是一维的，在赋值之前要首先对这个一维数组进行初始化，数据的大小就是所有像素的个数．

遍历的时候要注意方向，这个数据是以行为主要递增方向的．也就是说遍历的时候要先遍历第一行的所有数据，然后再遍历第二行的所有数据．

1. **地图优化**

通过激光SLAM算法构建好场景地图，栅格地图的创建是基于概率模型计算地图栅格的占用概率；生成map.pgm的栅格地图用于机器人的导航与路径规划等功能，由于激光雷达建图过程中存在精度误差，造成地图中有噪点和边界缝隙。因此，需要对创建后的栅格地图进行优化，消除栅格地图噪点和连接边界缝隙；

如图所示，栅格地图原图与优化后的地图图片；

map.pgm原图 map.pgm优化图片

**2.1优化方法分析**

方法一：从SLAM算法源码处优化边缘的噪点，对噪点进行剔除。

方法二：对创建的栅格地图进行优化，运用数字图像处理（OPENCV）的方法处理地图。

采用方法二利用OPENCV的基础API接口函数，对PGM栅格地图进行优化（边缘不清晰栅格进行优化），具体的优化步骤：

1. 调用OPENCV的API接口函数依赖的库；

#include <opencv2/opencv.hpp>

#include <opencv2/imgproc.hpp>

#include <opencv/highgui.h>

1. 读入map1.pgm的栅格地图，并转化为灰度图片；

    Mat img = imread("map1.pgm",IMREAD\_GRAYSCALE);

1. 对img的灰度图片进行阈值分割、中值滤波、腐蚀、膨胀等处理；

**阈值处理：对栅格地图进行阈值分割**

//阈值处理

dst.create(img.size(), img.type());

    int height = img.rows;

    int width = img.cols;

    int nc = img.channels();    //The method returns the number of matrix channels.

    for (int row = 0; row < height; row++)

    {

        for (int col = 0; col < width; col++)

        {

            if (nc == 1)    //单通道处理

            {

                if(row <= 13)

                {

                    dst.at<uchar>(row, col) = 205;//将灰度图片的值给新的dst矩阵

                   continue;

                }

                int gray = img.at<uchar>(row, col);//通过at索引图像的值

                if(gray !=205 && gray != 254)

                {

                   //dst.at<uchar>(row, col) = 0;//将灰度图片的值给新的dst矩阵

                   if(gray >= 225)

                   {

                        dst.at<uchar>(row, col) = 255;//将灰度图片的值给新的dst矩阵

                   }

                   else if(gray >= 185 && gray < 225)

                   {

                        dst.at<uchar>(row, col) = 205;//将灰度图片的值给新的dst矩阵

                   }

                   else

                   {

                        dst.at<uchar>(row, col) = 0;//将灰度图片的值给新的dst矩阵

                   }

                }

                else

                {

                    dst.at<uchar>(row, col) = gray;//将灰度图片的值给新的dst矩阵

                }

            }

            else if(nc==3)  //3通道处理

            {

                int b = img.at<Vec3b>(row, col)[0];

                int g = img.at<Vec3b>(row, col)[1];

                int r = img.at<Vec3b>(row, col)[2];

                dst.at<Vec3b>(row, col)[0] = b;

                dst.at<Vec3b>(row, col)[1] = g;

                dst.at<Vec3b>(row, col)[2] = r;

            }

        }

    }

    //将处理后的数据显示出来

    namedWindow("dst", WINDOW\_NORMAL);

    imshow("dst", dst);      //将数据输出显示；

imwrite("/home/chen/linux/opencv/opencv\_learning/map\_1/阈值处理.png", dst);

**腐蚀处理：连接栅格地图的边界缝隙并消除噪点**

  // //腐蚀

    Mat structElement = getStructuringElement(MORPH\_RECT, Size(3,3), Point(-1,-1));

    erode(dst, dst1, structElement);

    namedWindow("dst1", WINDOW\_NORMAL);

    imshow("dst1", dst1);

**中值滤波处理：消除栅格地图中的噪点**

    //中值滤波

    medianBlur(dst1, dst2, (3,3));

    namedWindow("dst2", WINDOW\_NORMAL);

    imshow("dst2", dst2);

**膨胀处理：**

    // 膨胀

    structElement = getStructuringElement(MORPH\_RECT, Size(3,3), Point(-1,-1));

    dilate(dst2, dst3, structElement, Point(-1,-1), 1);

    namedWindow("dst3", WINDOW\_NORMAL);

    imshow("dst3", dst3);

1. 将优化后的图片保存；

    imwrite("/home/chen/linux/opencv/opencv\_learning/map\_1/阈值+腐蚀+中值+膨胀处理.png", dst3);

**2.2 优化结果**

利用上述的栅格地图优化方法对三个不同场景的地图进行优化获得的优化结果；

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 图片 | 原图 | 阈值+腐蚀 | 阈值+腐蚀+中值 | 阈值+腐蚀+  中值+膨胀 |
| map1.pgm | 原图 | 阈值+腐蚀处理 | 阈值+腐蚀+中值处理 | 阈值+腐蚀+中值+膨胀处理 |
| map2.pgm | 原图 | 阈值+腐蚀处理 | 阈值+腐蚀+中值处理 | 阈值+腐蚀+中值+膨胀处理 |
| Demo1.pgm | 原图 | 阈值+腐蚀处理 | 阈值+腐蚀+中值处理 | 阈值+腐蚀+中值+膨胀处理 |

**2.3 梳理OPENCV依赖库**

调用基础的API函数处理栅格地图，使用的头文件包括opencv.hpp、imgproc.hpp、hughgui.hpp，所依赖的如下所示；

target\_link\_libraries(map\_1 /usr/lib/x86\_64-linux-gnu/libopencv\_imgcodecs.so.3.2.0)

target\_link\_libraries(map\_1 /usr/lib/x86\_64-linux-gnu/libopencv\_imgproc.so.3.2.0)

target\_link\_libraries(map\_1 /usr/lib/x86\_64-linux-gnu/libopencv\_core.so.3.2.0)

target\_link\_libraries(map\_1 /usr/lib/x86\_64-linux-gnu/libopencv\_highgui.so.3.2.0)

**2.4 房间分割算法**

房间分割算法：ipa\_coverage\_planning/ipa\_room\_segmentation

形态分割：给定地图作为二值图像，迭代地应用二值侵蚀。在每一步之后，算法都会搜索面积在定义范围内的单独区域。如果找到，则保存该区域并将其绘制为黑色，以便无需进一步查看。在所有白色区域都变为黑色之后，算法停止并在图像中绘制片段，然后应用波前算法来填充剩余的空白区域。

距离分割：从给定的图像中计算距离图，为每个白色像素存储它到最近的黑色像素的距离。在这个距离图上，迭代地应用阈值算子，使高于某个阈值的每个像素都变成黑色，而低于它的所有像素都变成白色。在每一步之后，算法搜索给定区域范围内的片段并存储它们，然后增加阈值。在查看了所有可能的阈值之后，算法选择具有最有效段的分割作为分割。之后，它还绘制图像中的片段并应用波前算法。

Voronoi 分割：对于给定的地图，计算修剪后的 voronoi 图。然后将属于该图的点分成不同的邻域，并在该邻域中搜索与黑色像素距离最小的点，称为临界点。这个临界点与两个最接近的黑色像素相连，形成了两段的分离。

语义分割：对于每个白色像素，模拟 360 度激光扫描，导致到该点周围障碍物的距离不同。这些激光扫描然后用于计算不同的特征，这些特征被放入两个 AdaBoost 算法中。这些计算当前点是房间或走廊的概率。根据概率给那个点一个标签。然后对给定地图中的每个白色像素重复相同的过程，从而为房间/走廊产生不同的分离区域，这些区域被视为片段。

Voronoi 随机场分割：首先为给定的地图计算修剪后的 voronoi 图，在该图上，邻域被集中到一个中心点。找到所有点后，就获得了相邻关系，这意味着哪个节点与另一个节点相邻，具体取决于 voronoi 图。这会产生几个派系，从而产生一个条件随机场。这个 CRF 的特征是 AdaBoost 分类器对房间、门口和走廊的弱响应。这些 AdaBoost 分类器使用与语义分割相同的特征。然后在这个 CRF 上进行循环信念传播，产生几个相交的片段。

**2.5 Voronoi房间分割算法**

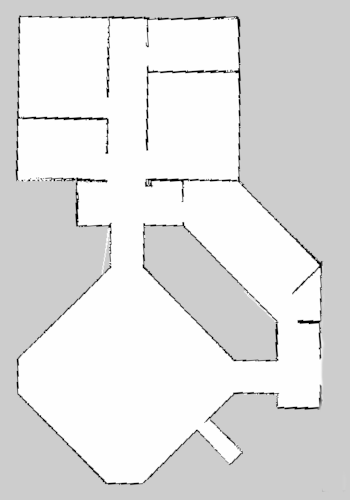
**2.5.1 订阅栅格地图数据（nav\_msgs::OccupancyGrid）**

接收激光SLAM构建的场景地图（栅格地图）如下图所示，图片为pgm格式，接收地图的数据int data[]，数据属性：

白色代表空闲，表示可通过区域，其存储的值为 0；

黑色代表占用，表示不可通过区域，其存储的值为 100；

灰色代表未知，表示未知区域，其存储的值为 -1



栅格地图数据订阅成功后，调用vonoroi算法对地图进行房间分割。

**2.5.2 Voronoi房间分割算法**

Voronoi 分割：对于给定的地图，计算修剪后的 voronoi 图。然后将属于该图的点分成不同的邻域，并在该邻域中搜索与黑色像素距离最小的点，称为临界点。这个临界点与两个最接近的黑色像素相连，形成了两段的分离。

网址：<https://github.com/ipa320/ipa_coverage_planning/tree/melodic_dev>

从房间分割算法：ipa\_room\_segmentation中分离出Voronoi房间分割算法，应用到栅格地图的房间中，解析算法，明确算法的输入参数和输出结果；

Voronoi房间分割算法调用：

void VoronoiSegmentation::segmentMap(const cv::Mat& map\_to\_be\_labeled,

cv::Mat& segmented\_map, double map\_resolution\_from\_subscription,

double room\_area\_factor\_lower\_limit, double room\_area\_factor\_upper\_limit,

int neighborhood\_index, int max\_iterations, double min\_critical\_point\_distance\_factor, double max\_area\_for\_merging, bool display\_map)

**2.5.3 参数配置与解析：**

const float map\_resolution = 0.5;

const cv::Mat& original\_img = map;

cv::Mat segmented\_map;

segmented\_map.create(map.size(), map.type());

double room\_upper\_limit\_voronoi\_ = 1000000;

double room\_lower\_limit\_voronoi\_ = 1.53;

int voronoi\_neighborhood\_index\_ = 280;

int max\_iterations\_ = 150;

double min\_critical\_point\_distance\_factor\_ = 1.6;

double max\_area\_for\_merging\_ = 12.5;

bool publish\_segmented\_map = false;

bool display\_segmented\_map\_ = false;

// 参数 1 ：Mat original\_img 输入图片

// 参数 2 ：Mat segmented\_map 分割后输出的图片

// 参数 3 ：double map\_resolution 图片的分辨率

// 参数 4 ：double room\_lower\_limit\_voronoi\_ 房间面积系数下限 判断值

// 参数 5 ：double room\_upper\_limit\_voronoi\_ 房间面积系数上限

// 参数 6 ：int voronoi\_neighborhood\_index\_ 相邻系数 //310

// 参数 7 ：int max\_iterations\_ 最大迭代次数： //0++

// 参数 8 ：double min\_critical\_point\_distance\_factor\_ 最小临界点距离因子：//1.7

// 参数 9 : double max\_area\_for\_merging\_ 最大合并区域：

// 参数 10 : bool display\_segmented\_map\_ 发布分割地图后的数据

// 参数 11 : bool DEBUG\_DISPLAYS 是否显示分割后的地图

**2.5.4** 房间分割步骤

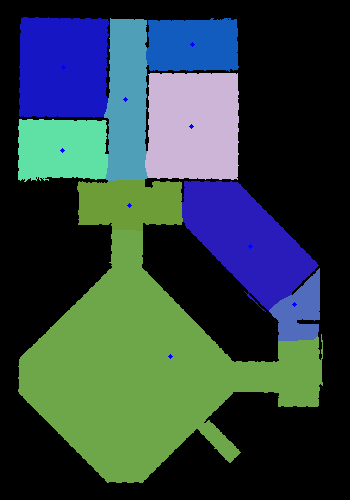
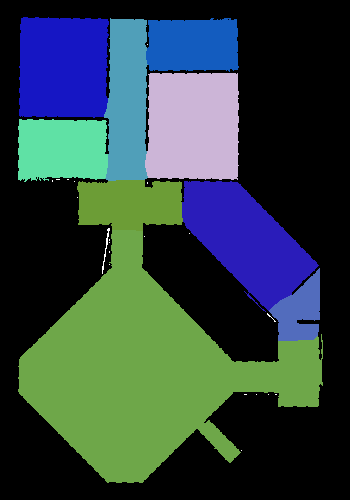
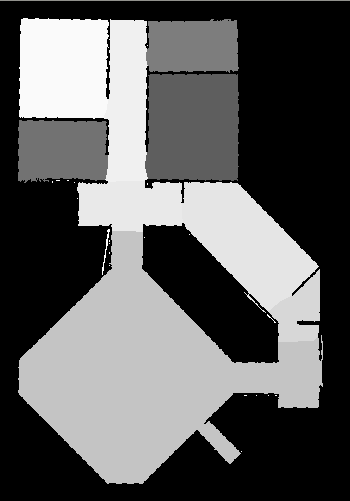
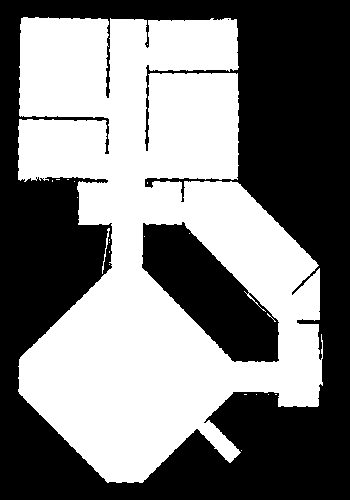
1、Vonoroi分割算法的第一个参数（original\_img）图片输入为二值化地图，将栅格地图转化为二值化地图，空闲区域为白色，墙体和未知区域为黑色，如下图所示：

2、根据地图的分辨率、面积大小、房间大小等因素调整房间分割的参数配置，如2.1节所示；

3、输出Vonoroi分割的地图，对地图的各个房间进行了分割，并且使用不同的灰度等级进行区分，对不同的房间岁不同的房间进行彩色标记；

4、计算每个房间的中心位置并且标记中心位置点区分房间，获得最后的房间地图；

1. 将最终分割完成的地图进行保存，并且发布房间分割后的地图数据；



**2.5 面积计算**

利用房间分割的方法，获取每个房间的大小，计算每个房间占多少个像素，利用每个房间占据的像素个数计算房间的面积；（opencv计算轮廓面积）

像素个数 \* 分辨率 \* 分辨率 = 房间面积（）；

1. **优化后地图发布**
2. **总结**

本文完成机器人栅格地图优化调研，从SLAM算法构建地图出发，获取栅格地图，针对栅格地图中存在的噪点和边界缝隙等问题，提出采用数字图像处理（OPENCV）的方法，调用OPENCV中的基础API接口函数对栅格地图进行优化，之后对地图进行房间分割，实现对房间的规划和管理，并计算出房间的面积，最后完成地图的保存将地图以节点的方式发布出去。