专 业 学 位 硕 士 学 位 论 文

基于 RGB-D 的 SLAM 定位与 GPU 三维建图方法研究

Research on SLAM Location and GPU 3D Mapping Method Based on RGB-D

作 者 姓 名：王 振 龙

工 程 领 域：软 件 工 程

学　　号：31617067

指 导 教 师：周宽久 教授

完 成 日 期：2018 年 3 月 25 日

大连理工大学

Dalian University of Technology

大连理工大学学位论文独创性声明

作者郑重声明: 所呈交的学位论文，是本人在导师的指导下进行研究工作所取得的成果。尽我所知，除文中已经注明引用内容和致谢的地方外，本论文不包含其他个人或集体已经发表的研究成果，也不包含其他已申请学位或其他用途使用过的成果。与我一同工作的同志对本研究所做的贡献均已在论文中做了明确的说明并表示了谢意。

若有不实之处，本人愿意承担相关法律责任。

学位论文题目: _____

作 者 签 名 : _____ 日期: _____年____月____日

摘　要

传统传感器的 SLAM 一般需要在环境放置识别标志，基于视觉的 SLAM 系统可以在不改变环境的情况下实现定位与建图需求，并且更加灵活多变，适应复杂环境。但是基于视觉的 SLAM 系统在处理传感器的图片信息和进行三维建模时会消耗大量的算力，很难在保证实时性的同时保证准确性。

而本文正是在此基础上提出了一个采用 RBG-D 图像序列的同步定位与三维建图方法，在系统的实时性和系统的准确性方面也进行了相关算法的比较分析与优化选择。本文可分为基于特征的视觉里程计设计和拓扑图优化的 GPU 三维建图。其一，对相邻的两张 RGB-D 图像使用 ORB 特征提取算法进行特征提取，特征匹配算法使用的是 FLAAN，可以进行特征匹配，使用基于距离筛选的策略和 RANSAC 算法进行特征点匹配对的筛选优化，最后使用 PNP 计算相机位姿。当输入 RGB-D 图像序列时，则可以实时输出相机位姿。其二，根据上一部分中的相机位姿，采用拓扑图存储相机运动信息，使用闭环检测优化的方式减小相机移动过程中产生的累积误差，同时根据相机移动距离和相机方向来筛选关键帧减小点云噪声。

最后使用 GPU 加速点云间坐标转换，然后将点云图转换为八叉树地图用以存储以定位需求。八叉树地图是一种基于概率估计的三维地图，可以大大缩小地图的存储空间。经过试验，本文所设计的系统能够满足同步定位和地图创建的实时性需求和精确性需求，使用 GPU 对三维建模的速度有很大提升。

关键词：RGB-D；特征检测；八叉树；SLAM

Research on SLAM Location and GPU 3D Mapping Method Based on RGB-D

Abstract

The traditional sensor's SLAM usually needs to place identification marks in the environment. The vision based SLAM system can realize the requirement of positioning and mapping without changing the environment, and is more flexible and adaptable to the complex environment. But the visual based SLAM system will consume a lot of computing power when dealing with the image information of the sensor and modeling the 3D modeling. It is difficult to ensure the accuracy while ensuring the real-time.

On the basis of this, this thesis puts forward a method of synchronous positioning and 3D mapping using RBG-D image sequence. It also makes comparative analysis and Optimization on the real-time and system accuracy of the system. This thesis can be divided into the features of visual odometry based design and topology optimization of GPU 3D mapping. First, two adjacent RGB-D images are extracted by ORB feature extraction algorithm, and the feature matching algorithm is FLAAN, which can match the feature. The distance selection strategy and RANSAC algorithm are used to optimize the matching of feature points. Finally, PNP is used to calculate the position and posture of the camera. When the RGB-D image sequence is input, the position and posture of the camera can be output in real time. Secondly, according to the position and posture of the camera in the previous part, the motion information of the camera is stored by the topology graph, and the cumulative error generated in the camera movement is reduced by the closed loop detection optimization. At the same time, the camera moving distance and the camera direction are used to screen the key frame to reduce the cloud noise.

Finally, we use GPU to accelerate point cloud coordinate transformation, and then convert point cloud to octree map to store location requirements. Octree map is a 3D map based on probability estimation, which can greatly reduce the storage space of map. The system designed in this thesis can meet the real-time requirements and precision requirements of synchronous positioning and map creation, and the speed of 3D modeling is greatly improved by using GPU.

Key Words：RGB-D; Feature Detection; Octree; SLAM

目　　录

1 绪论

1.1 研究背景及意义

当今时代，科技日新月异，从第一次机械时代开始，机器取代手工劳动的进展越来越快，人类进一步的解放双手的需求变得越来越迫切，而现在的第四次互联网智能时代正在进行。移动机器人自主导航问题一直都是该领域的难点和重点，实现移动机器人准确并且实时定位是众多学者、研究者的目标。在场景比较小的环境中，研究者可以通过在场景中预设标志实现移动机器人自动识别避障，可是在现实生活中，大部分环境是未知的，对于这种未知环境，在不改变环境的情况下，我们就需要移动机器人能够主动地从环境中获取障碍物信息，从而创建地图用以导航避障。

在室外环境中，机器人（或者自动驾驶汽车）可以使用 GPS 进行粗略地定位，但是机器人也只能够知道我们所处的相对于地球的方位，而机器人难以了解周围的物理环境，所以会导致处处碰壁、寸步难行。同时在室内环境下，各种家用服务机器人能够在小范围内进行自身定位及导航，但是目前大部分室内机器人的定位导航是基于提前输入室内模型，或者在室内的角点做标记的方式实现的。

本论文实现了一种基于 RGB-D 的视觉 SLAM 系统，该系统无需提前在室内做标记，更加灵活多变，能够适应复杂环境；同时，本论文使用 GPU 加速三维建模的过程，实现更快的三维建模速度。并且本论文实现使用八叉树地图来存储地图信息，减小了地图文件的存储消耗，而且实现了地图定位的功能。同时三维重建在医学、增强现实等领域也发挥着重要的作用。

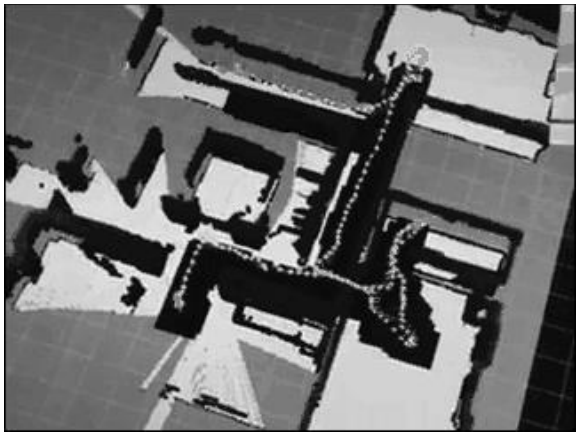

图 1. 1　SLAM 技术在无人机上的应用

Fig. 1.1　Application of SLAM technology in UAV

图 1.1 是宾夕法尼亚大学教授 Kumar 做的一个无人机的 2D SLAM，图中灰色区域为未扫描区域，白色区域为空白位置，黑色区域为无人机扫描出的房屋墙壁，黑色区域中的白点为无人机飞行路径。无人机从起点起飞扫描屋内环境，对屋内物体墙壁进行建模，同时判断路径逐步将整个屋子扫描完毕。

1.2 SLAM 问题概述

相对于室外环境，室内环境更加复杂，要求的精度更高，容错率更低。从二十世纪八十年代开始，室内定位问题凸显出来，得到人们的密切关注，一批批室内定位的方法层出不穷，其中 SLAM 技术涌现出来。SLAM 技术是一个跨越多学科的技术，它需要传感器采集环境数据，然后对环境数据进行处理，将处理结果进行构建三维模型，这里面每个步骤都有很多方案实现。其中在传感器方面，摄像头和激光雷达被采用的最多。

为了实现 SLAM 技术，这里有四个方面必须考虑：

（1） 我们用什么来表示三维模型，点云图或者八叉树图，基于它们之间的特点和我们所做系统的需求，我们要有选择性地选择。

（2） 获取环境信息问题，我们用什么传感器去感知周围环境，是单目摄像头，双目摄像头还是 RGB-D 相机，亦或是激光雷达等等。

（3） 从环境信息到三维模型的转换问题，由于传感器的不同，获得的环境信息结构不一样，例如单目摄像头获得的是单幅单帧的图片，双目是单帧双幅图片，而 RGB-D 是单帧 RGB 和深度图，至于激光雷达，则是环境中每个点到传感器的距离。

（4） 定位问题和建图问题，我们怎么从环境信息转换到三维模型，由于传感器不能够一次获得全部场景的信息，我们怎么将多个部分的信息整合成整个环境的三维模型，以及随之而来的误差优化相关的问题。

在 SLAM 的研究中，还有很多问题等待我们去解决，例如用于减少误差的回环检测、姿态计算等等。由于 SLAM 算法在运算过程中会消耗大量资源导致实时性受到挑战，所以部分对实时性要求很高但是对功耗没有要求的方案会选择在 GPU 上运行 SLAM 算法。

1.3 基于 RGB-D 相机的 SLAM 方法研究现状

而在同步定位与地图创建理论之初，也就是上世纪六十年代末期，学者 Nilsson[1]团队设计了一架采用同时定位与地图创建[2]的机器人，它叫做 Shakey。至此，同时定位与地图创建理论得到广大学者认识，开始有越来越多研究机构，学校学者关注该领域，并在许多领域发挥出巨大价值。

Kinect 相机是微软公司 2010 年 6 月针对 Xbox360 发布的一款体感设备，它包含了

一个 RGB 相机用于收集彩色信息，一个红外发射头一个红外发射头用于收集深度信息。Kinect 相机如图 1.2 所示。

图 1.2　Kinect 相机

Fig. 1.2　Kinect Camera

2010 年华盛顿大学的 Peter Henry、Michael Krainin 等研究人员与英特尔实验室的 Dieter Fox、 Xiaofeng Ren[3]在早些时候就已经提出了一种在室内环境下使用深度相机进行三维模型构建的方法。这种方法不仅能够很好地处理阴暗位置的信息，而且能够实时输出结果。RGB-D ICP 最早在这个方法上得到应用，它采用 SIFT 特征算法进行特征提取和描述符计算，同时该方法还对相邻帧之间的运动变换使用了 TORO 工具和闭环检测。接着 2012 年，Peter Henry[4] 等人对之前的算法进行了改进，使用 FAST[5] 特征检测算法代替了 FAST 特征检测算法，并使用了 Calonder 描述符提取方法，并且使用 RANSAC 进行运动变换估计的运算，用于减小误差，采用闭环检测，当产生闭环时使用稀疏光束平差法（Sparse Bundle Adjustment，SBA）进行全局路径优化。并且加入了 ICP 预选机制，当特征点匹配数目较少时采用 ICP 算法，当特征点匹配数目较多时采用一般方法，这个方法对 RGB-D 的实时性帮助很大。

2011 年德国弗莱堡大学的 Nikolas Engelhard[6]等人使用 Kinect 相机开发了一套 RGB-D SLAM 系统。这套系统采用 Surf 特征检测算法提取 RGB 图片的特征和描述符，经过特征匹配，找到相邻两幅图片之间的匹配点，根据匹配点的坐标从深度图中读取匹配点处的深度信息。这样两幅图之间相同的特征点之间就会有对应关系，我们得到了一组三维坐标点对的集合。使用运动变换估计 RANSAC（Random Sample Consensus, 随机采样一致性）求得相机在两帧之间的变换关系，将变换关系输入 Hogman 位姿图用以位姿优化，将优化后的帧输出点云构成三维地图。

2012 年 Nikolas Engelhard [7]等人改进了之前的设计，他们将 Hogman 位姿图优化方

法替换为全局位姿图优化方法（Global Pose Graph Optimization,G2O），G2O 是一个开源的基于图的非线性误差函数的框架，通过这个框架的优化的得到优化后的位姿路径，得到一个全局的点云图，最后将点云图转换为八叉树图 OctoMap（Octree-based Mapping）。

2013 年，Felix Endres [8]等人更新之前的设计，提出了 环境评估模型（Environment Measurement Model，EMM），这个模型可以验证运动变换估计 RANSAC 和 ICP 求解转换方程的正确性。此外，他们团队还有更多的 RGB-D SLAM 方法。

2011 年，Huang A S, Bachrach A [10]等人采用 SBA 算法和 PRF 算法（最小化重影误差法）生成三维地图。文献[11,12]采用最小光度误差法和深度信息对三维点云除噪声，得到更准确的两个帧间的变换关系，由于文献[11]采用了定位技术，其获得的位姿很准确，适合复杂的室内环境。文献[12]引入了能量方程，由于刚体无需考虑形变，采用动力驱动的机器人被约束在能量守恒定律下，通过这种方式可以减小误差。由于特征检测，特征匹配，路径优化，姿态计算等都需要消耗大量算力，文献 [13]采用 GPU 加速的方式，将计算量大的程序一直到 GPU 上运行，得到了精度、质量、稠密度都很高的三维模型。

还有其他个人研究者和团队研究者都提出了自己的 RGB-D SLAM 算法方案，如文献 [14-18]所示。还有其他个人研究者和团队研究者提出的很多三维建图的算法，如文献 [19-23]所示。

1.4 本文主要工作

本论文的主要内容是围绕着使用 Kinect 相机生成的 RGB-D 图像生成三维模型，这种基于 RGB-D 的视觉 SLAM 方法。Kinect 会同时生成 RGB 图像和深度图，我们根据相邻两幅 RGB 图像的偏差计算出相机位姿变化，依次迭代就可生成全局的相机位姿变化曲线，根据优化后的位姿曲线取合适的关键帧生成点云的三维模型。最后考虑到整个系统的实时性能和准确性能，本文对特征检测、特征匹配等算法进行了比较选择，最后的实验比较验证了方案的可行性。本文的论文结构具体如下：

第二章介绍实现 SLAM 技术时需要的一些底层数学知识，其中包括相机的成像模型和相机标定，同时由于相机成像是二维的，本文还会介绍从 2D 到 3D 的转换过程和方法，为后续章节提供理论支持。介绍了一种基于概率估计的三维地图构建方法八叉树。基于第四章优化后的位姿选择关键帧，将关键帧代表的地图转换为八叉树地图。由于八叉树地图是一种基于概率估计的三维建图方法，它能够很好的消除大部分噪声。

第三章介绍了一个基于图像特征的 SLAM 过程，这个过程是 SLAM 系统的雏形，已经可以基本实现的 SLAM 的功能。包括特征提取过程、特征匹配过程、运动变换估计

算法、相机位姿计算等。在特征提取算法方面，之前的研究者或者应用者已经有了很多成熟的方案，包括，SIFT、SURF、FAST、Harris、ORB 等，本文会针对对其中的几种特征提取算法进行比较，最后使用 ORB 特征检测算法。在特征匹配算法方面经典的有 Brute Force 算法和 FLANN 算法两种，本文会进行一些介绍及描述。运动变换估计使用的是 RANSAC（随机采样一致性），相机位姿计算目前使用较多的有两种方式，一种是 PNP 的方法，一种是 ICP 的方法，两种方法计算过程不一样但是思想有相似的地方。

第四章介绍了一种基于拓扑图优化的相机位姿估计和本文中拓扑图优化的相机位姿估计的实现方式。首先介绍拓扑图优化问题的来源发展。针对 SLAM 问题中一定存在的误差问题，采用拓扑图优化的相机位姿估计是一种很好的方式，这个误差不仅出现在观测的时候，而且在进行运算和特征检测时也会产生误差，由于相机一直在移动，相机目前的位姿都有相机变化的位姿和上一个位姿叠加得到，所以不可避免会产生累计误差。基于相机基于拓扑图优化的相机位姿估计就是为了解决这个问题。本文会介绍 G2O 这种提供精确修正误差函数的框架和本文中它的应用。介绍了 GPU 加速三维建图的过程，并说明了 GPU 加速的架构以及加速的方法。

第五章介绍了本文系统的组成、软件依赖、实验设计、实验结果及实验分析。

2 SLAM 系统的相关技术综述

2.1 相机成像模型

2.1.1 针孔成像模型

将相机模型看得简单些就是光学成像模型，目前的光学成像模型有两种，一种是线性模型还有一种是非线性模型。现实生活中的透镜模型多是非线性模型，一个一般的透镜成像示意图如图 2.1 所示。

其中 u 为物体与透镜中心之间的距离， f 为焦距，v 为相机底片与透镜中心的距离。三者之间的关系是：

$$\frac{1}{f}=\frac{1}{u}+\frac{1}{v} \tag{2.1}$$

相机的镜头一般是一组透明镜，和主光轴平行的光纤穿过透镜会在某一点上汇聚，那么这个平行光线汇聚的点就是焦点，焦距 f 就是汇聚点焦点到透明镜中心的距离。对于数码相机来说，它的镜头就相当于凸透镜，底片也即感光元件会被放置在这个凸透镜的焦点旁边，如果我们近似认为感光元件与透镜中心的距离就是焦距的话，那么这个模型就是小孔成像模型。小孔成像模型如图 2.2 所示。

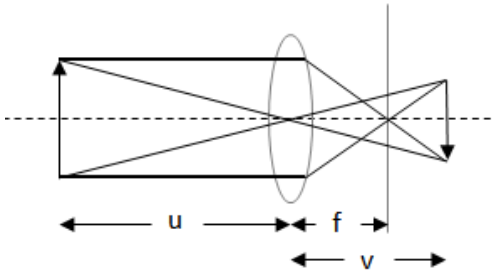

图 2.1 透镜成像原理示意图

Fig. 2.1 Lens principle diagram

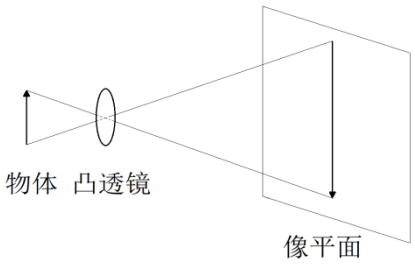

图 2.2 小孔成像模型

Fig. 2.2 Small imaging model

2.1.2 坐标系变换关系

小孔成像模型在相机成像中应用十分广泛。由于在小孔成像模型下，我们可以将求解相机参数的过程看成对线性方程组求解的过程，这样，物体的空间坐标和图像坐标可以通过线性关系进行转化。与相机小孔成像相关的四个坐标系，他们分别为相机坐标系、图像坐标系、世界坐标系，他们如图 2.3 所示。在三维空间中的点 M 经过相机坐标系，在图像坐标系上留下的点为 m，也即 m 就是相机拍照时现实世界中物体 M 在照片中的

成像。

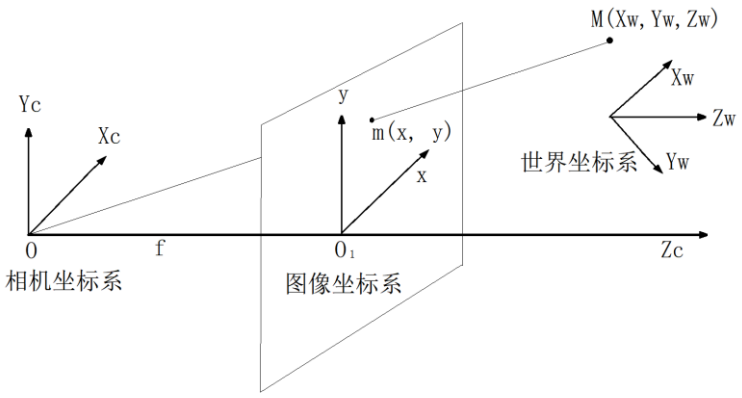

图 2.3　坐标系间的关系图

Fig. 2.3　Sitting relationship

世界坐标系：由于需要一个坐标系来表示我们现实世界的物体位置，而世界坐标系就是这个坐标系，他是我们现实世界的基准，用以描述数码相机和在这个现实三维世界中的任意物体，我们可以用坐标（X_w, Y_w, Z_w）表示任何在现实世界中的物体的坐标值。

相机坐标系（光心坐标系）：相机的光心为坐标系 O 点，并且坐标系的垂直轴依次平行于图像坐标系的两个垂直轴，可以用（X_c, Y_c, Z_c）表示处于该坐标系下点的坐标值。

图像坐标系：图像坐标系的 O 点为图像的的中点，两条坐标轴分别平行于图像的两条边，可以用（x, y）表示处于该坐标系下点的坐标值。图像坐标系的单位是国际距离单位（例如毫米，米）用以表示图像中像素的位置。

像素坐标系：像素坐标系的坐标 O 点也在 CCD 平面上，它为 CCD 平面的左上角顶点。同图像坐标系类似，像素坐标系的 X 轴和 Y 轴也和图像平面的垂直边平行，差别在于像素坐标系中的每个点只代表一个像素点，一般为感光原件最小单位，它和距离没有直接关系。我们可以用（u, v）来表示像素在该坐标系下的位置。由于数码相机使用感光元件获取光信息，感光元件的大小为 $M \times N$，所以在这个像素坐标系下，像素的数目为 M 和 N 的积。我们使用一个数组存储每个像素点的信息。在数码相机中，感光元件将光信息转换为数字信息存储在每一个像素中，所以像素坐标系也就是一个基于像素的图片坐标系。

像素坐标系和图像坐标系之间的转换关系如图 2.4 所示。

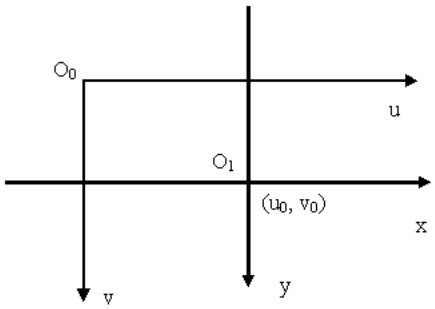

图 2.4　图像坐标系与像素坐标系的关系图

Fig. 2.4　The relationship chat of image coordinate and pixel coordinate

他们之间的转换关系为：

$$u = \frac{x}{dx} + u_0 \qquad v = \frac{y}{dy} + v_0 \tag{2.2}$$

采用齐次坐标系再用矩阵形式将上式表示为：

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} 1/dx & 0 & u_0 \\ 0 & 1/dy & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2.3}$$

图像坐标原点在像素坐标系中的坐标可以表示为(u_0, v_0)，dx 表示每个像素点在图片坐标系下所占的 x 轴距离，dy 表示每个像素点在图片坐标系下所占的 y 轴距离。

可以通过下式将图像坐标系转换为相机坐标系：

$$x = \frac{fX_c}{Z_c} \qquad y = \frac{fY_c}{Z_c} \tag{2.4}$$

其中 f 为焦距（像平面与相机坐标系原点的距离）。可以用齐次坐标系和矩阵表示上述关系：

$$Z_c \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} \tag{2.5}$$

将相机坐标系转换到世界坐标系：

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} = \begin{pmatrix} R & t \\ 0^r & 1 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \tag{2.6}$$

其中 R 是一个 3×3 的正交的旋转的矩阵，t 是三维的平移向量，将式（2.2）、式

（2.3）、式（2.4）综合起来可以得到：

$$Z_c\begin{pmatrix}u\\v\\1\end{pmatrix}=\begin{pmatrix}\frac{1}{dx}&0&u_0\\0&\frac{1}{dy}&v_0\\0&0&1\end{pmatrix}\begin{pmatrix}f&0&0&0\\0&f&0&0\\0&0&1&0\end{pmatrix}\begin{pmatrix}R&t\\0^r&1\end{pmatrix}\begin{pmatrix}X_w\\Y_w\\Z_w\\1\end{pmatrix}$$

$$=\begin{pmatrix}a_x&0&u_0&0\\0&a_y&v_0&0\\0&0&1&0\end{pmatrix}\begin{pmatrix}R&t\\0^r&1\end{pmatrix}\begin{pmatrix}X_w\\Y_w\\Z_w\\1\end{pmatrix} \tag{2.7}$$

$$=KM_1\begin{pmatrix}X_w\\Y_w\\Z_w\\1\end{pmatrix}=M\begin{pmatrix}X_w\\Y_w\\Z_w\\1\end{pmatrix}=\begin{pmatrix}m_{11}&m_{12}&m_{13}&m_{14}\\m_{21}&m_{22}&m_{23}&m_{24}\\m_{31}&m_{32}&m_{33}&m_{34}\end{pmatrix}\begin{pmatrix}X_w\\Y_w\\Z_w\\1\end{pmatrix}$$

其中：

$$a_x=f/dx \qquad a_y=f/dy$$

$$K=\begin{pmatrix}a_x&0&u_0&0\\0&a_y&v_0&0\\0&0&1&0\end{pmatrix}\quad M_1=\begin{pmatrix}R&t\\0^r&1\end{pmatrix} \tag{2.8}$$

a_x 是图像水平轴的尺度因子，a_y 是图像垂直轴的尺度因子。

K 是相机的内部参数矩阵，它的主要参数只有主点坐标和焦距等，而这些参数是被相机内部固定的结构所决定的。由于相机坐标系和世界坐标系的相对位置决定了 M_1 中的旋转矩阵和平移向量，因此 M_1 也被称为数码相机的外部的参数矩阵，M 叫做投影矩阵，而 R 和 t 就被叫做外部的参数，相机标定的过程就是确定相机内参和外参的过程。

2.2 相机标定

2.2.1 相机内参与畸变参数

（1） 相机内外参

相机的内参数是六个分别为：$1/dx$、$1/dy$、r、u_0、v_0、f。

dx 和 dy 表示：X 轴方向和 Y 轴方向的一个像素在图片坐标系中分别占的长度大小，即一个像素代表的实际物理值的大小，其是实现图像物理坐标系与像素坐标系转换的关键。u_0，v_0 代表的是将像素坐标系的原点经过转换到图片坐标系为图片坐标系的原点。

相机的外参数是 6 个：三个轴的旋转参数分别为（ω、δ、θ）,然后把每个轴的 3×3 旋转矩阵进行组合（即先矩阵之间相乘），得到集合三个轴旋转信息的 R，其大小还是 3×3；T 的三个轴的平移参数（T_x、T_y、T_z）。R、T 组合成成的 3×4 的矩阵，其是转换到标定纸坐标的关键。其中绕 X 轴旋转 θ，则其如图 2.5 所示：

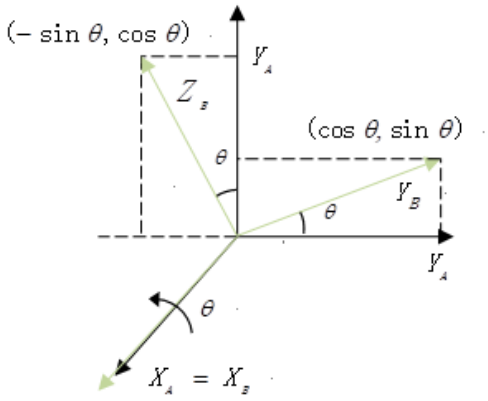

图 2.5　相机外参数

Fig. 2.5　External parameters of a camera

相机的旋转矩阵如式 2.9 所示。

$$R_{rotx}(\theta)=\begin{pmatrix}1 & 0 & 0\\ 0 & cos\theta & sin\theta\\ 0 & sin\theta & cos\theta\end{pmatrix} \tag{2.9}$$

（2） 畸变参数

k_1, k_2, k_3 径向畸变系数，p_1, p_2 是切向畸变系数。径向畸变发生在相机坐标系转图像物理坐标系的过程中。由于感光单元平面和透镜没有完全平行，那么在制作相机的阶段中就会发生切向畸变。光纤距离透镜中心越近，则投影越直，反之越弯曲，径向畸变分为桶形的畸变和枕形的畸变两种。

当成像仪与摄像机粘连在一起的时候，往往或造成透镜与图像平面不平行，这个时候就会造成切向畸变。他们的转换关系如式 2.7 所示。

2.2.2　相机的标定方法与意义

我们进行的机器视觉实验或应用，和测量图片的过程中需要建立摄像机成像的数学模型，用来确定空间中物体的大小、形状、位置或者多个物体之间的位置关系等。而这个数学模型必须要用到拍摄照片的相机的镜头参数，而我们通过实验手段并加以计算求解参数，这个过程我们通常称之为相机标定。

（1） 相机标定的意义

不管是图像测量，亦或是机器人视觉的应用，相机参数标定至始至终都是很重要的环节，标定参数的好坏直接影响到系统的精度和算法是否稳定运行以及最后结果的正确性。因此，相机标定必须做好，它是后面所有工作的前提，如何提高标定相机参数时的精度就成为了科研人员的目标所在，这个目标就是测量并计算出相机的内外参和畸变数据。

（2） 张正友标定法

张正友标定方法是目前求解相机内参和畸变系数的经典方法，也是最受欢迎的方法。采集至少两张不同角度的平面标定板图像，相机标定是三维场景重构中必不可少的步骤，目的是提取出二维图像中包含的计量信息。相机标定方法分为两种：一种是传统的摄影测量标定方法，另一种是自标定方法；张正友标定法是一种在自标定方法和传统的摄影测量标定方法之间的新的并且有效方法；方法相较于其他方法更便捷和精度，稳定；张正友标定方法的透镜模型如式 2.7 所示，张正友法一般公式如式 2.10 所示。

$$sm' = A[R|t]M'$$

$$s\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.10}$$

2.3 3D 空间位置表示

2.3.1 2D 姿态描述

2D 空间中，物体的位姿可用两个平移量 t_x, t_y 加一个旋转角 θ 表示，如图 2.6 所示。此时，设机器人坐标系 $O'-X'-Y'$ 下某点的坐标为$[x_r, y_r]^T$，对应在世界坐标系$O-X-Y$下为$[x_w, y_w]^T$。

那么推得：

$$\begin{cases} x_w = x_r cos\theta - y_r sin\theta + t_x \\ y_w = x_r sin\theta + y_r cos\theta + t_y \end{cases} \tag{2.11}$$

则有：

$$x_w = \begin{pmatrix} R_{2\times2} & t_{2\times1} \\ 0^T_{1\times2} & I_{1\times1} \end{pmatrix} X_r \tag{2.12}$$

为便于理解，我们在矩阵下方标出了它的维数。可以看到使用齐次坐标满足了线性关系，记作：

$$x_w = T_{w,r} x_r \tag{2.13}$$

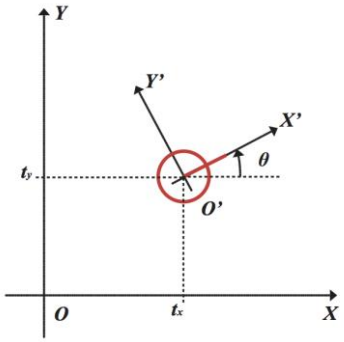

图 2.6　2D 平面物体位姿表示

Fig. 2.6　Pose representation of a plane object

其中 $T_{w,r}$ 表示从世界坐标系到机器人坐标系的变换矩阵。我们也可以轻松地写出反向的变换矩阵：

$$x_r = T_{r,w} x_w = T_{w,r}^{-1} x_w = \begin{pmatrix} R^{-1} & -R^{-1}t \\ 0^T & 1 \end{pmatrix} x_w \tag{2.14}$$

既然如此，我们就可用 T 表示机器人的位姿，那么机器人在时刻 t 的位姿就可以记作 T_t。当然，从存储上来讲，存储 T 是不经济的。在 2D 运动中，它有九个变量，但实际自由度只有三个。所以我们可以只存储位移矢量 t 与旋转角 θ，而在需要计算的时候再构建出 T。称 2D 欧几里得变换，它对矩阵乘法构成群(群是一个集合加一种运算，且运算在该集合上满足封闭性、结合律、有单位元和逆元)，该群记作 $SE(2)$。相应的，二维旋转构成二维旋转群或称特殊正交群 $SO(2)$。

2.3.2　3D 变换

3D 的旋转可以由旋转矩阵、欧拉角、四元数等若干种方式描述，它们也统称为三维旋转群 $SO(3)$。而 3D 的变换即旋转加上位移，视为 $SE(3)$。为了和 2D 变换统一起见，我们首先介绍旋转矩阵表示法。

（1） 旋转矩阵描述

旋转矩阵是一种 3×3 的正交矩阵，它对变换的描述十分类似于 2D 情形。参照上一节的数学符号，我们有：

$$X_w = \begin{pmatrix} R_{3\times3} & t_{3\times1} \\ 0_{1\times3}^T & I_{1\times1} \end{pmatrix} X_r \tag{2.15}$$

这里 R 为 3D 的旋转矩阵，同样的，t 为 3D 的平移矢量。

由于 3D 旋转都可以归结成按照某个单位向量 n 进行大小为 θ 的旋转。所以，已知某个旋转时，可以推导出对应的旋转矩阵。该过程由罗德里格斯公式表明，由于过程比较复杂，我们在此不作赘述，只给出转换的结果：

$$R(n,\theta)=\begin{pmatrix} n_x^2(1-c\theta)+c\theta & n_xn_y(1-c\theta)+n_zs\theta & n_xn_z(1-c\theta)-n_ys\theta \\ n_xn_y(1-c\theta)-n_zs\theta & n_y^2(1-c\theta)+c\theta & n_yn_z(1-c\theta)+n_xs\theta \\ n_xn_z(1-c\theta)+n_ys\theta & n_yn_z(1-c\theta)-n_xs\theta & n_z^2(1-c\theta)+c\theta \end{pmatrix} \tag{2.16}$$

这里 $n=[n_x,n_y,n_z]^T, c\theta=cos\theta, s\theta=sin\theta$ 公式虽然较为复杂，但实际写成程序后，只需知道旋转方向和角度后即可完成计算。另一件有趣的事是，如果用

$$n^{\wedge}=\begin{pmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_z \\ -n_y & n_x & 0 \end{pmatrix} \tag{2.17}$$

表示与 n 对应的一个反对称矩阵，那么有：

$$R(n,\theta)=cos\theta I+(1-cos\theta)nn^T+sin\theta n^{\wedge}=exp(\theta n^{\wedge}) \tag{2.18}$$

根据此式，我们也可以从任意给定的旋转矩阵，求出对应的转轴与转角。关于转角 θ，我们对上式两边求矩阵的逆，可得：

$$\begin{aligned} tr(R)&=cos\theta tr(I)+(1-cos\theta)tr(nn^T)+sin\theta tr(n^{\wedge}) \\ &=3cos\theta+(1-cos\theta) \\ &=1+2cos\theta \end{aligned} \tag{2.19}$$

因此：

$$\theta=arccos(\frac{1}{2}[tr(R)-1]) \tag{2.20}$$

关于转轴 n，由于旋转轴上的向量在旋转后不发生改变，说明 $Rn=n$ 。

因此，只要求此方程的解向量即可。这也说明 n 是 R 特征值为 1 的一个特征向量。也即在 3D 时，旋转和平移仍可用转移矩阵 T 来描述，其结构也与 2D 类似。而 $T_{4\times4}$ 构成了三维欧氏变换群 $SE(3)$。注意到 T 虽然有 16 个变量，但真正的自由度只有 6 个，其中 3 个旋转，3 个位移。

旋转矩阵描述是一种比较适合数学推导及计算机实现的方式，但它不符合人类的思维方式。当你看到一个 3×3 的矩阵时，很难想象物体实际上发生了怎样的旋转。反之，给定一个旋转方式，人也很难直接写出矩阵的数值。所以，为了便于人类理解，人们还使用了其他方法来表示三维旋转。

（2） 欧拉角

欧拉角是一种广为使用的姿态描述方式，以直观见长。在最常用的欧拉角表达方式中，我们把旋转分解成沿三个轴转动的量：滚转角－俯仰角－偏航角(roll-pitch-yaw)。它的好处是十分的直观，且只有三个参数描述。缺点是会碰到著名的万向锁问题：在俯仰为±90∘时，表达某个姿态的形式不唯一。此外，它也不易于插值和迭代。

这里不详细介绍欧拉角，因为它在 SLAM 中用处也很有限。我们既不会在数学推导中使用它，也不会在程序中用欧拉角表示机器人的姿态。不过，在您想验证自己算法输出的姿态是否有错时，转换成欧拉角能让你快速辨认结果是否有错。

（3） 四元数

与欧拉角相比，四元数更加容易迭代，并且结构紧凑，求解过程不会出现奇异解。本文使用的 G2O 库中对相机姿态描述的方式就是四元数，还有众多移动机器人项目，和网上非常出名的 SLAM 数据集等都采用了四元数村粗姿态。

如果用一个矩阵来表示物体的三维旋转，那么我们需要一个 3×3 的矩阵，需要的空间为 9，如果使用四元数的话只需要使用 4 个空间就够了，分别存储 *x, y, z, w*，其中 *w* 为尺度，用来让 *x, y, z* 可以表示出-∞—+∞，当 w=0 是，表示的即是正负无穷。如果用 3×3 的矩阵，由于只是用了三个空间，另外的 6 个空间都空着，那么这样必会产生奇异值，而使用四元数时，由于四个空间均得到利用并且唯一，所以不会产生奇异值。

四元数是 Hamilton 找到的一种扩展的复数。一个四元数拥有一个实部和三个虚部（故事上说他原先找了很久带两个虚部的，结果怎么也找不到，最后豁然开朗找到了三虚部的四元数）：

$$q = q_0 + q_1 i + q_2 j + q_3 k \tag{2.21}$$

其中 i, j, k 为四元数的三个虚部。这三个虚部满足关系式：

$$\begin{cases} i^2 = j^2 = k^2 = -1 \\ ij = k, ji = -k \\ jk = i, kj = -i \\ ki = j, ik = -j \end{cases} \tag{2.22}$$

由于它的这种特殊表示形式，有时人们也用一个标量和一个向量来表达四元数：

$$q = [s, v], s = q^0 \in \mathbb{R}, v = [q_1, q_2, q_3] \in \mathbb{R}^3. \tag{2.23}$$

这里，标量 *s* 称为四元数的实部，而向量 *v* 称为它的虚部。如果一个四元数虚部为 0，称之为实四元数。反之，若它的实部为 0，称之为虚四元数。

四元数可以表示三维空间中任意一个旋转。与旋转矩阵中类似，我们仍假设某个旋转是绕单位向量 $n = \left[n_x, n_y, n_z\right]^T$ 进行了角度为 θ 的旋转，那么这个旋转的四元数形式为：

$$q=[cos\frac{\theta}{2},n_x sin\frac{\theta}{2},n_y sin\frac{\theta}{2},n_z sin\frac{\theta}{2}]^T \tag{2.24}$$

式（2.24）中的四元数被称为单位四元数。

在复数域$\mathbb{C}$，我们可以用一个复数$e^{i\theta}$表示 2*D* 的旋转，类似的，3*D* 空间也可以用单位四元数表示旋转。假设一个空间三维点$v=[x,y,z]\in\mathbb{R}^3$，以及一个由旋转轴和夹角 *n*, θ 指定的旋转，下面讨论如何用四元数表示它们。

把三维空间点用一个虚四元数来描述：

$$p=[0,x,y,z]=[0,v] \tag{2.25}$$

参照式 2.24，用另一个四元数 *q* 表示这个旋转：

$$q=[cos\frac{\theta}{2},nsin\frac{\theta}{2}]. \tag{2.26}$$

那么，旋转后的点 p' 即可表示为这样的乘积：

$$p'=qpq^{-1} \tag{2.27}$$

可以验证，计算结果的实部为$n^T(n\times v)=0$，故计算结果为纯虚四元数。其虚部的三个分量表示旋转后 3*D* 点的坐标。

由于任意单位四元数都可表示为一个 3*D* 旋转，即 *SO*(3)中的元素，我们可以找到一个旋转矩阵与之对应。最简单的方式是由四元数 *q* 解出旋转角 θ 和旋转轴 *n*，但那样要计算一个 *arccos* 函数，代价较大。实际上这个计算是可以通过一定的计算技巧绕过的。为省略篇幅，我们直接给出四元数到旋转矩阵的转换方式。

设四元数$q=q_0+q_1 i+q_2 j+q_3 k$，对应的旋转矩阵 *R* 为：

$$R=\begin{pmatrix}1-2q_2^2-2q_3^2 & 2q_1q_2+2q_0q_3 & 2q_1q_3-2q_0q_2\\ 2q_1q_2-2q_0q_3 & 1-2q_1^2-2q_3^2 & 2q_2q_3+2q_0q_1\\ 2q_1q_3+2q_0q_2 & 2q_2q_3-2q_0q_1 & 1-2q_1^2-2q_2^2\end{pmatrix} \tag{2.28}$$

反之，由旋转矩阵到四元数的转换如（式 2.29）。假设矩阵为 $R=\{m_{ij}\}, i,j\in[1,2,3]$，其对应的四元数 *q* 由式（2.29）给出：

$$\begin{aligned}q_0&=\frac{\sqrt{tr(R)+1}}{2}, q1=\frac{m_{23}-m_{32}}{4q0}\\ q_2&=\frac{m_{31}-m_{13}}{4q_0}, q3=\frac{m_{12}-m_{21}}{4q_0}\end{aligned} \tag{2.29}$$

由于 *q* 和−*q* 表示同一个旋转，事实上一个 *R* 的四元数表示并不是惟一的，存在其他三种与上式类似的计算方式。实际编程中，当 q_0 接近 0 时，其余三个分量会非常大，导致解不稳定，此时会考虑使用剩下的几种方式计算。

① 相似变换：相似变换比欧氏变换多了一个自由度，它允许物体进行自由地缩放。旋转部分多了一个缩放因子 s，它在 x,y,z 三个坐标上形成均匀的缩放。由于含有缩放，相似变换不再保持图形的面积不变。

$$Ts=\begin{pmatrix} sR & t \\ 0^T & 1 \end{pmatrix} \tag{2.30}$$

② 仿射变换：与欧氏变换不同的是，仿射变换只要求 A 是一个可逆矩阵，而不必是正交矩阵。在仿射变换下，直线的夹角会发生改变，但平行性质不变。

$$T_A=\begin{pmatrix} A & t \\ 0^T & 1 \end{pmatrix} \tag{2.31}$$

③ 射影变换：由于采用齐坐标，当 $v\neq 0$ 时，我们可以对整个矩阵除以 v 得到一个右下角为 1 的矩阵；否则，则得到右下角为 0 的矩阵。它左上角为可逆矩阵 A，右上为平移 t，左下缩放 a^T。

$$T_P=\begin{pmatrix} A & t \\ a^T & v \end{pmatrix} \tag{2.32}$$

表 2.1 总结了目前讲到的几种变换的性质。注意在“不变性质”中，从上到下是有包含关系的。例如，欧氏变换除了保体积之外，也具有保平行、相交等性质。

表 2.1　常见变换性质比较

Tab. 2.1　Comparison of common transformation properties

变换名称	自由度	不变性质
欧式变换	6	体积
相似变换	7	体积
仿射变换	12	平行性、体积比
射影变换	15	接触平面的相交和相切

2.4　三维地图

2.4.1　点云图

点云图，顾名思义就是无数个点组成云，就和无数个小水珠组成天上的云一样。对于本文，由于采集的照片是 RGB-D，每一帧都自带 RGB 图和深度图，故只由一帧就可以生成点云图如图 4.1 所示。通过第三章计算的位姿，我们可以将多帧合成一个点云图，为了减小误差和冗余信息，第四章使用了关键帧策略，再使用回环检测，最后生成的点

云图和八叉树图。

本论文中使用的点云库是 PCL（Point Cloud Library）。PCL 库是一个基于 C++语言的开源库，已经有大量的研究人员和开发人员付出了很多努力。PCL 库能够进行绝大部分点云相关运算和操作，可以实现点云的获取，滤波、分割等操作，还可以对点云进行配准运算、检索操作、特征提取、识别操作、追踪运算、曲面重建、可视化行为等。PCL 库支持目前市面上的绝大部分操作系统，甚至大部分嵌入式系统。Opencv 解决的是 2*D* 图片处理的问题，那么 PCL 解决的就是三维立体的问题

PCL 能解决的问题：

（1） CAD/CAM、逆向工程

（2） 激光遥感测量

（3） 虚拟现实、人机交互虚拟现实技术（简称 VR）

PCL 包括多个子模块库。最重要的 PCL 模块库有：过滤器 Filters、特征 Features、关键点 Keypoints、注册 Registration、Kd 树 Kd-tree、八叉树 Octree、切分 Segmentation、Sample Consensus、Surface、Range Image、文件读写 I/O、Visualization、通用库 Common、Search。本文中使用了 Filters、Common、Visualization、IO 等库。

2.4.2 八叉树地图

八叉树可以用来描述三维空间的物体，这种图是一种树状的数据结构。类比于二叉树，八叉树就是每个父节点最多有八个子节点。由于八叉树建图的过程就是将一个正方向不断细分的过程，很容易知道，根节点是最大正方形的中心，将最大正方形分成八个小正方形（根节点的八个子节点），可知八个小正方形的体积之和等于大正方形。使用八叉树用来表现三维空间中物体在 1980 年后才开始流行起来。

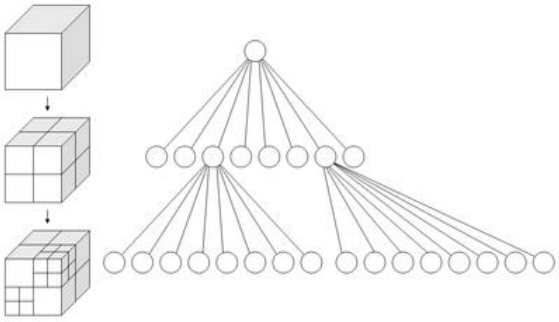

图 2. 7　八叉树

Fig. 2.7　Octree

（1） 八叉树三维数据结构

如图 2.8 所示，假设我们要在三维空间使用八叉树表示一个物体 *V*，首先，我们设定一个正方形 *C*，满足在 *V* 中的任何点都在 *C* 内，那么 *C* 就可以当做八叉树的根节点。在分辨率还没达到要求的情况下，我们需要将正方形 *C* 继续等分成八个正方形，如图中的 0-7，由图 2.8 所示，八叉树的根节点 *C* 的八个子节点分别任 0 到 7。接下来依次对八个子节点进行判断，若存在 *V* 中的点都不在某个子节点中，那么该子节点为空，表示该子节点的任意子节点均为空，则该子节点不用继续递归；若存在该子节点的点全部在 *V* 内，则该子节点为 1，表示该子节点的任意子节点均为 1，所以该子节点可以不用继续递归；否者继续对该子节点进行八等分，直到子节点的正方体体积小于或等于分辨率要求，终止递归。

从上面生成八叉树三维结构的步骤中我们可以知道，最后的八叉树会有三种状态：空，1，非空非 1。非空非 1 状态，的意思是物体的边缘在该子节点内，也即若将该子节点继续细分，那么他的子节点可能会出现空和 1 两种状态。

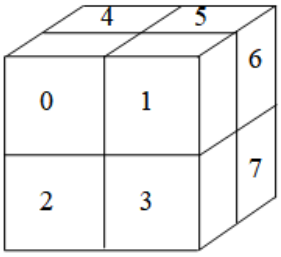

图 2.8　某个子立方体被八等分

Fig. 2.8　A sub cube is divided by eight

（2） 八叉树的存贮结构

按照存储类型区分的八叉树可以分为：线性八叉树、规则八叉树、一对八式八叉树。不同的八叉树优缺点各异，综合来看，一对八式八叉树在优缺点的平衡方面最优秀。

① 规则八叉树

规则八叉树采用链表存储方式，使用固定的数据结构存储每个几点，每个节点结构体有 9 个成员，其中一个成员为当前节点的类型，分为全部占据，空、部分占据。另外八个成员存储八个字节点的指针。假设用一个字节表示节点状态，用 4 个字节存储指针，那么规则八叉树的空间利用率为 1/(1+4×8)=3.03%，可知，使用规则八叉树空间浪费很严重。优点是结构简单。

② 线性八叉树

线性八叉树采用深度遍历的方式将所有节点依次存储，优点是空间利用率非常高，

缺点是查找操作时间复杂度很大。

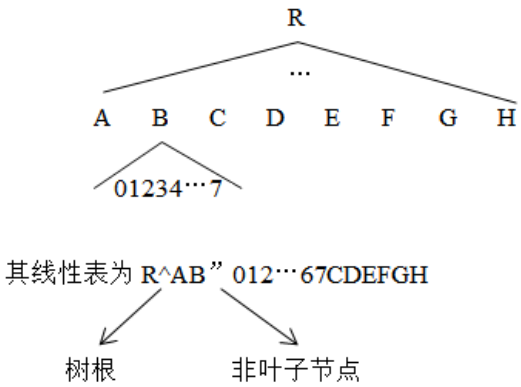

图 2.9 线性八叉树

Fig. 2.9 Linear octree

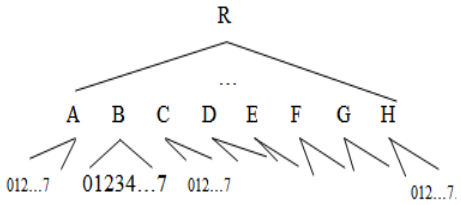

图 2.10 一对八式的八叉树

Fig. 2.10 One to eight octree

③ 一对八式的八叉树

一对八式八叉树采用线性存储方式，规定每个节点的子节点数目必须为八个，所有叶子节点在同一层，也即，一对八式八叉树是一个完全八叉树的结构。从存储效率方面来说，一对八式八叉树的存储利用率不稳定，当每个叶子节点都为中间状态时，则利用率达到最大 100%，比线性八叉树的存储效率还高，当效率最低是，也即八叉树一根枝丫走到头，这时的存储效率最低，八叉树层数越高，存储效率越低，当层数 n 趋近无穷大时，存储效率满足公式 $\eta=\lim_{n\to\infty}\frac{n}{(8^n-1)/7}=\lim_{n\to\infty}\frac{7n}{8^n-1}=0$ ，所以使用大正方形来存储一个点，会造成空间的极大浪费。现实存储过程中应该尽量让叶子节点的数目最多。

（3） 八叉树实现（深度遍历）

① 首先设定八叉树的精度，即叶子节点的尺寸；

② 找到一个正方体，使该正方体能完全包含所有场景，则该节点为根节点，所有节点初始状态均为灰色（非空非占）；

③ 判断该节点是否为空，若为空则将该节点状态置为空；判断该节点是否被场景全部包含，若是，则将该节点状态置为占用；

④ 判断是否达到精度要求，若达到，则跳转到步骤⑥，否则继续向下执行；

⑤ 进行深度递归，将当前节点等分为 8 个子节点。跳转到步骤③；

⑥ 判断该节点是否为根节点，若为根节点则停止递归，否者判断该节点是否有下个子节点，若有则对下个子节点，执行步骤③；否则返回父节点，跳转到步骤⑥；

3 基于图像特征的视觉 SLAM 设计

3.1 图像特征检测与描述子获取

3.1.1 SIFT 算法

SIFT 特征算法从深层来说，它对图片中一些局部外观特别的地方很敏感，同时，图片的旋转与缩放都不会影响特征点的提取，并且该算法在微视角、噪声、光线的变化方面，容错率也很高，不会因为上述有一点变化导致结果发生骤变。我们可以根据物体的特征，构建一个庞大的数据库，只要数据库够大到容纳所有的物体特征，那么我们在获取某一个物体后将其特征与数据库比较就能够很轻松地识别出该物体是什么。同时当物体没有全部显露出来时，理论上只需要露出来的部分采集出三个特征就可以计算出该物体的方位和位置，从而看出对这些物体 SIFT 算法也有很好的识别结果。随着电脑运行速度的加快，特征数据库也朝着小型化发展，使用 SIFT 对物体进行提取特征并辨识越来越快，目前已经可以接近实时运算。使用 SIFT 提取的特征数目多，所以它适用于数据库比较大的情况。

（1） SIFT 算法的优点有：

① SIFT 特征是图片的局部的特征，对图片的的尺度进行缩放，图片旋转角度改变，亮度变化都不会影响 SIFT 特征，同时在图片进行仿射变换或者视角发生变化甚至有噪声的情况下，SIFT 依然能有良好的表现。

② 独特性好，由于基于 SIFT 提取的特征点数目多，所以携带的图片信息更多，在特征数据库够大的情况下，FIST 能够更快更准确地完成匹配。

③ 多量性，物体即使很少，使用 SIFT 算法也能够生成大量的特征向量；

④ 高速性，可以自由选择 SIFT 算法的步骤，进行优化后 SIFT 匹配算法几乎可以完成实时任务；

⑤ 可扩展性，在与其他形式的特征向量结合方面，SIFT 也表现优异。

（2） SIFT 算法的缺点有：

① 相对 FAST 和 SURF 等算法而言，SIFT 的时间消耗仍然太多。

② 在某些特殊情况下的的特征点数目很少。

③ 如果图像模糊，或者物体的边缘光滑，SIFT 提取的特征点不够正确。

3.1.2 SURF 算法

SURF 算法拥有 SIFT 算法的准确性、多量性等特点，同时在实时性方面 SURF 算

法也有很大的进步，避免了 SIFT 算法因为计算复杂、实时性不高，同时 SURF 还改进了 SIFT 的特征提取和描述子计算方法。

SURF 在求特征值的过程中使用 Henssian 阵的方法还是很稳定的，但是由于 SURF 算法在求解主方向的过程是基于对关键点邻近的区域，所以这可能会导致该方法找到的主方向与实际主方向有偏差。由于后面计算特诊向量，以及匹配特征点都会使用这个主方向，那么，误差会被进一步放大，从而导致匹配失败率升高。并且如果 SURF 算法图像金字塔的层数稀疏，是的尺度偏差过大，而在计算特征向量的过程中需要使用这个尺度，那么由进一步放大了误差，所以为了处理这个问题，我们可以取出适中密度的层。然后进行插值运算。

3.1.3 ORB 算法

ORB(Oriented FAST and Rotated BRIEF)是一种快速特征点提取和描述的算法，本论文中使用 ORB 进行特征的快速提取和描述子的快速计算。

（1） 特征检测

ORB 基于 FAST（features from accelerated segment test）算法来检测特征点。FAST 核心思想就是找出那些卓尔不群的点，即拿一个点跟它周围的点比较，如果它和其中大部分的点都不一样就可以认为它是一个特征点。

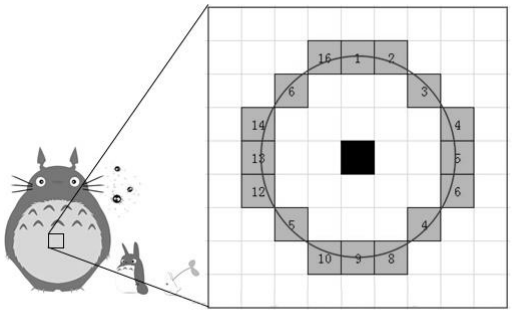

图 3.1 FAST 特征检测算法

Fig. 3.1 FAST algorithm

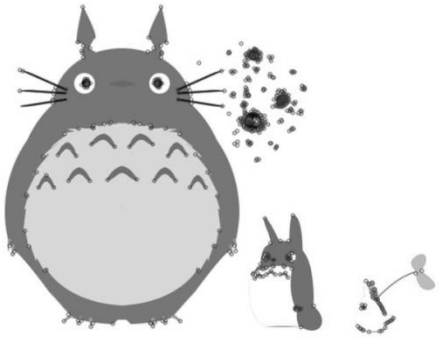

图 3.2 FAST 算法找到的特征点

Fig. 3.2 The feature points found by the FAST algorithm

FAST 具体计算过程：

① 先从图片中随机选择一个点，将这个点的密度值（灰度）设定为 F；

② 然后给定一个合适的阈值 K，我们找出两个不同的点，方法是比较这两个点的密度值，若密度值比阀值小时则认为两个点是相同的；

③ 同理将步骤②拓展到图 3.1 所示的 16 个像素中，分别计算这 16 个像素的相同与否；

④ 首先假设不同点个数 N 为 12，然后排查步骤 3 中的 16 个像素，如果有连续的 N 个点和中心的像素点 P 不一样，那么我们认为中心像素点就是一个特征点；

⑤ 我们现在提出一个高效的测试，来快速排除一大部分非特征点的点。该测试仅仅检查在位置 1、9、5 和 13 四个位置的像素（首先检查 1 和 9，看它们是否和点 P 相同。如果是，再检查 5 和 13）。如果是一个角点，那么上述四个像素点中至少有 3 个应该和点相同。如果都不满足，那么不可能是一个角点。

（2） 计算特征描述子

ORB 算法中计算描述子的方法采用的是 BRIEF 算法，BRIEF 计算描述子的方法是计算特征点周围 N 个点对的比较结果，这 N 个点选择遵循一定的规则。如图 3.3 所示。

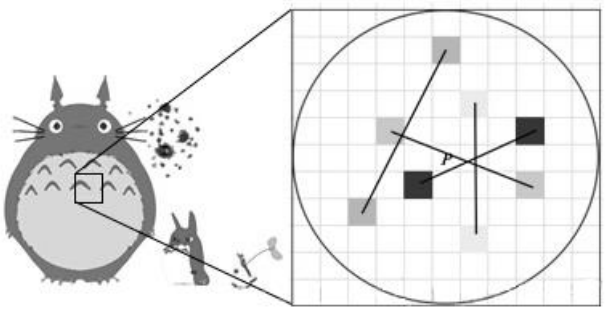

图 3.3　BRIEF 算法的描述子

Fig. 3.3　The descriptor of the BRIEF algorithm

BRIEF 具体计算过程：

① 如图 3.3，取一个正方形范围的像素，像素的内切圆圆心为特征点 P

② 设定一种选择特征点周围点对的规则，根据这个规则选择特征点 P 周围的 N 个点对，N 的方位越大描述子结果越精确，当 N 等于 4 时，点对的选取结果如图 3.3 所示并且上面四个点对可以表示为下面形式：

$$P_1(A,B)、P_2(A,B)、P_3(A,B)、P_4(A,B) \tag{3.1}$$

③ 定义操作 T ，其中 I_A 表示点 A 的灰度

$$T(P(A,B))=\begin{cases}1 & I_A > I_B \\ 0 & I_A \le I_B\end{cases} \tag{3.2}$$

④ 分别对已选取的点对进行 T 操作，再将最后的结果组合起来就得到描述子。假如：

$$T(P_1(A,B))=1 \quad T(P_2(A,B))=0$$
$$T(P_3(A,B))=1 \quad T(P_4(A,B))=1 \qquad (3.3)$$

则最终的描述子为：1011。

3.1.4 三种方案实验比较

基于时间空间的限制，SIFT 算法在准确性上技高一筹，但是在实时性上稍逊一筹，本论文做了以下比较。实验平台及条件：Intel i5-2520M CPU@2.5GHz * 4，Ubuntu16.04 系统，内核版本 Linux version 4.13.0-32，Opencv3.1 库，特征点提取算法比较如表 3.1 所示。

表 3.1 三种算法提取的特征点数目比较

Tab. 3.1 Comparison of the number of feature points extracted by three algorithms

ImageNo	SIFT	SURF	ORB
1	2414	4126	500
2	4295	8129	500
3	3404	4784	500
4	1639	2802	500
5	1510	1484	497
6	191	187	195
7	4899	7523	500
8	163	168	287
9	1884	2413	500
10	21.52	17.4	0.97

表 3.2 三种算法速度比较

Tab. 3.2 Speed comparison of three algorithms

ImageNo	SIFT	SURF	ORB
eiffel-1,9.jpg	2.77	3.22	0.11

表 3.1 是通过一些图片集测的单单是 feature detect 的时间，接下来通过一对图片看看 feature detect 和 compute feature descriptor 总共花费的时间开销（秒），如表 3.2 所示。可以看到计算 descriptor 的开销还是很大的，这里仅仅两张图片，当图片数目很多时，系统的计算开销将向计算描述符偏斜。根据上面的数据，可以得出结论，ORB 算法在特征提取和描述子计算上比 SIFT 算法和 SURF 算法的实时性都要好，三种方法在本论文

中的应用比较如图 3.4 所示。其中左上为原图，右上为 SIFT 方法提取的特征点，左下为 SURF 方法提取的特征点，右下为 ORB 方法提取的特征点。

图 3.4　SIFT、SURF、ORB 提取特征点比较

Fig. 3.4　Comparison of extracting feature points by SIFT、SURF、ORB

从图 3.4 中可以直观地看出，SIFT 和 SURF 算法提取的特征点相对 ORB 算法多一点，并且 SIFT 算法提取的特征点多是在物体的边界处；而 SURF 算法对物体的边界更加敏感，在方向性上表现得比 SIFT 更加敏感；至于 ORB 算法，由于论文中 ORB 最多只能提取 500 个特征点，所以从图中看出，该算法优先将特征更加明显（圆圈更大）的特征点取出，而一些特征相对弱的则直接抛弃。

从上面的比较可以得出结论：

（1） 在特征点提取数目方面，ORB 可以自由设置，相对而言比 SIFT 和 SURF 更灵活，速度更快。当 ORB 最多提取特征点为 500 左右时，特征点数目完全够用。

（2） 在特征点提取质量方面，虽然 ORB 算法的提取数目相对于其他两种算法少一些，但是 ORB 算法的特征点方向性更高，特征表现更加明显，所以 ORB 提取的特征点质量更高。

（3） 特征点提取和描述子计算方面，ORB 算法的速度比 SIFT、SURF 快一个数量级，这对于 SLAM 系统的实时性有很大的帮助。

所以，ORB 方法速度快，特征点数目足够，故在本实验中选择 ORB 方法提取特征点和计算特征点描述子。

3.2 图像匹配算法

3.2.1 FLANN 算法

Muja 和 Lowe [24]与 2009 年提出近似最近邻（Fast Library for Approximate Nearest Neighbor, FLANN）算法，该算法的实现是以 *K* 均值树或者 KD-TREE 搜索树为基础，该算法能够向使用者提供优化的索引类型和检索参数，依据就是提交给算法的对映射的精度、数据集的分布特点以及空间时间资源的消耗的要求，并且当空间维度较高时，该算法进行最近邻查找不会受到局部敏感哈希[25]的影响。FLAAN 算法的核心内容是如何找到每一个实例点的邻居点，并且该算法将 *N* 维的实数向量空间 R_n 作为它的特征空间。若我们将 *Dp* 看做特征点 *p* 的特征分向量，把 *Dq* 看做特征点 *q* 的特征分向量，那么式（3.4）表示的就是 *d*(*p*, *q*)的欧式距离。

$$d(p,q) = < Dp - Dq \bullet Dq - Dp > \qquad (3.4)$$

3.2.2 BF 算法

BF 算法也即 Brute Force 算法，既暴力算法，这种算法的思路很简单，就是将两个串的每个元素进行比较，若第一个串的个数为 *M*，第二串的个数为 *N*，那个 BF 算法的时间复杂度就是 *O*(*M*·*N*)。

3.2.3 两种匹配算法实验比较及改进方法

以上面特征提取算法提取的特征点（如图 3.4 所示）为基础，分别使用 FLANN 算法和 BF 算法来进行特征点匹配。匹配结果如图 3.5 和图 3.6 所示，从图片可以得出，BF 算法和 FLANN 算法匹配的特征数目差不多，匹配质量上 FLANN 算法相比 BF 算法并没有很大的优势。两种方法的匹配速度经实验可得，如图 3.7 所示。从比较结果中可以的出，BF 算法和 FLANN 算法提取的特征点数目均为 1364，但是 FLANN 算法比 BF 算法快了一个数量级，其中 FLANN 算法所用时间为 0.030976 秒而 BF 算法所用时间为 0.103139 秒。所以本论文采用 FLANN 算法匹配特征点。但是无论是 FLANN 算法还是 BF 算法，它们的匹配结果还是有很多错误的匹配，本论文使用一种基于距离判断的筛选方法，把距离太大的去掉，准则是：去掉大于十倍最小距离的匹配。筛选后的匹配结果如图 3.8 所示。

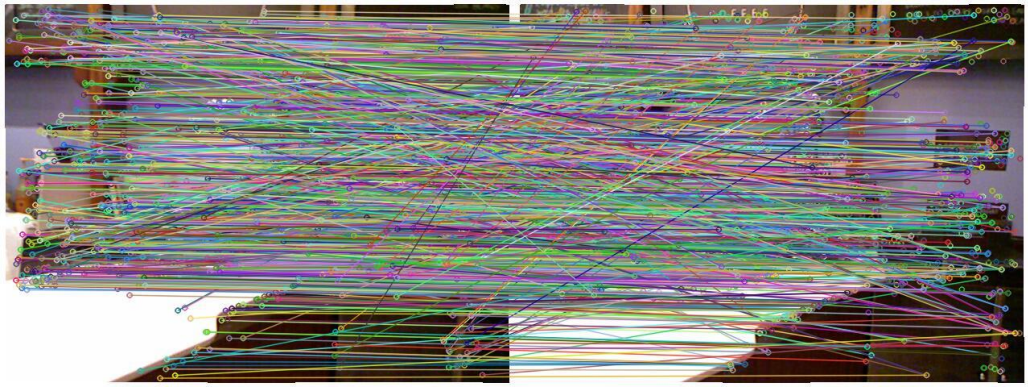

图 3.5　BF 算法匹配结果

Fig. 3.5　Matching results of BF algorithm

图 3.6　Flann 算法匹配结果

Fig. 3.6　Matching results of FLANN algorithm

图 3.7　FLANN 匹配与 BF 匹配结果比较

Fig. 3.7　Comparison of matching results between FLANN and BF

从图 3.8 中可以看出，经过距离筛选的优化错误的匹配大大减少，证实这种基于距离筛选的方法有一定的效果。同时由图 3.7 所示，经过距离筛选后的匹配 good matchs 的数目为 339，也即，匹配数目减少了三倍。但是，由于只是基于距离筛选，对距离一

样的错误匹配，这种筛选策略无能为力。

图 3.8　BF 算法经距离筛选后的匹配结果

Fig. 3.8　Matching results after distance screening by BF algorithm

3.3　基于方向的距离筛选策略

在对 RGB 照片进行特征特提取和描述子计算之后，我们需要根据计算得出的描述子进行特征匹配，根据匹配的特征点来计算相机的相对旋转与位移。

尽管如此，不管是经过 BF 特征匹配还是 FLANN 特征匹配，匹配后的匹配特征对还存在大量的错误匹配，由于本论文使用的是 FLANN 算法，所以这里只展示 FLANN 算法的匹配结果。FLANN 的匹配结果如图 3.6 所示。将图 3.6 模型化，可以得到图 3.9。

图 3.6 中特征匹配结果中大部分的错误匹配包含两个特征点距离很远，以及两个特征点之间的连线与正确匹配的两个特征点之间的连线角度很大。可以知道，图 3.9 中，最左边的两个特征点的特征匹配是距离远的，而图 3.9 中最左边第三和第四两个特征点分别有一条匹配角度特别大，很明显上面的这四条匹配是错误的匹配，我们需要将其去除。

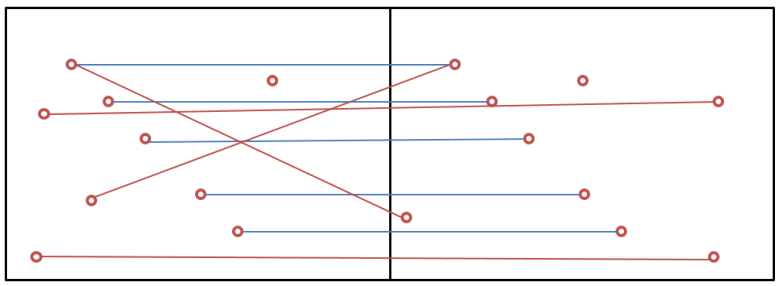

图 3.9　特征匹配结果模型化

Fig. 3.9　Modeling of feature matching results

表 3.3　基于方向的距离筛选策略算法

Tab.3.3　Direction based distance screening strategy algorithm

基于方向的距离筛选策略
for (auto m : matches)
if（minDis < m.dis）
minDis = m.dis
for (auto m : matches)
if (m.dis > minDis * M）
matches.erase(m)
for (auto m : matches)
If |m.query.X –m.train.X| > X.Max　OR　| m.query.Y – m.train.Y | > Y.Max
matches.erase(m)

图 3.10　基于方向的距离筛选结果

Fig. 3.10　Direction based distance screening results

只经过距离筛选的匹配结果如图 3.8 所示。可以看出，匹配结果中仍然有方向角度很大的错误匹配，如果将其用来计算相机位姿将会造成很大的误差。

本文提出基于方向的距离筛选策略，原理是：首先进行方向判断，去除方向偏差比较大的匹配，然后进行距离筛选，去除距离大于最小距离一定倍数的匹配。

基于方向的距离筛选策略的结果如图 3.10 所示。

从图 3.10 可以直观地看出，那些方向性很差的匹配已经被筛选掉了，这证实基于方向的距离筛选策略是有效的。

3.4　基于 PNP 的相机位姿计算

3.4.1　计算方法

基于 PNP 方法计算相机姿态需要提前获知相邻两帧照片中同一点的映射关系，也

即上一节中介绍的特征点匹配。但是经过以上简单的匹配，再加上优化后的距离筛选，还是会有错误的匹配，基于此，在计算 PNP 之前，一般会采用一种方法除去噪声，在本文中采用的方法是 RANSAC（Random Sample Consensus，随机采样一致性），这里不对 RANSAC 过多叙述，Opencv 库中可以直接调用。

图 3.11　RANSAC 优化匹配结果

Fig. 3.11　RANSAC optimization of matching results

经过 RANSAC 优化，如图 3.11 所示，和只经过距离筛选的图 3.8 比较，可以得出结论，RANSAC 优化后可以得到近似完美的匹配。将图 3.10 与图 3.11 比较可得，经过 RANSAC 的筛选，错误匹配进一步减少，依据为图 3.11 的匹配对数相对图 3.10 更加少。下面介绍 PNP 算法计算相机位姿的实现过程。

这里我们假设，我们得到的匹配结果是完美的，我们根据这些完美的匹配计算这两张图像之间的转换问题。它的模型如式 3.5 所示：

$$\begin{pmatrix} u \\ v \end{pmatrix} = C\left(Rp + t\right) \tag{3.5}$$

其中，R 为相机的姿态矩阵，C 为相机的标定矩阵，标定可根据 2.2 节得方法标定。矩阵 C 在相机出厂就已固定，R 随着相机的运动而变化。u、v 为图片坐标系下的点的坐标。我们已知 u、v、C、p，在 Opencv 中最少需要 4 组 $\begin{bmatrix} u & v \end{bmatrix}^T$ 和 p，即可求出旋转矩阵 R 和转移矩阵 t。

Opencv 中提供了一个 PNP 的函数。详细的 API 说明请参考 Opencv2.4.9（http://docs.opencv.org/master/），本论文中用于计算 PNP 的函数的输入参数：

其中输入参数 cameraMatrix 就是相机的标定矩阵 R。

$$R=\begin{pmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{pmatrix} \tag{3.6}$$

输入参数 objectPoints 为第一个帧的三维点。以某个特征点在图像坐标系的坐标[*x*, *y*]、该特征点在深度图中的深度和相机的标定矩阵，可以求得该特征点在三维坐标系下的坐标，也即 objectPoints。

输入参数 imagePoints 为第二个帧的图像点。与第一个帧中该特征点匹配的第二个特征点的坐标。

输出 rvec 为旋转向量，输出 tvec 为平移向量。

输出 inliers 为经过 RANSAC 优化后匹配的特征点对。

iterationsCount 为 RANSAC 运行时的迭代次数。

Confidence 为算法的置信区间。使用时，先要根据第一帧的深度图将第一帧经 RANSAC 筛选的特征点转换到三维空间，作为输入参数 objectPoints。

3.4.2 算法实现

PNP 算法在本论文的代码实现过程如代码所示，根据前面几节的设计，我们已经得到了两帧图片的特征点和特征匹配，由这两张相邻的图片的特征点和特征匹配，调用 Opencv 的 solvePnPRansac 函数，可求得相邻两帧图片的旋转矩阵和位移矩阵，分别存储在 rvec 和 tvec 中。

经过 PNP 计算后的输出结果如图 3.7 所示。我们这里采用 FLANN 算法进行匹配，最后计算得出的旋转矩阵和转移矩阵分别为：

R = [-0.02377374055725811, 0.03303522040324439, 0.01460124962106053]^T

t = [0.0216568507466905, 0.01049981188660019, 0.01206187917807096]^T

从终端的输出可以看出，经过距离筛选和 RANSAC 算法的优化后的匹配比较如表 3.3 所示。

由表 3.4 可知，经过距离筛选，特征点匹配对变为原来的三分之一，经过 RANSAC 优化，特征点匹配对变为距离筛选后的三分之一。

表 3.4 距离优化和 RANSAC 优化的结果比较

Tab. 3.4 Comparison of optimization results by distance and RANSAC

优化方法	无优化	基于方向的距离筛选	RANSAC
匹配对数	1364	320	109

4 拓扑图优化及 GPU 加速

4.1 拓扑图优化问题

第三章介绍了如何从两帧相邻的图片中计算出相机的空间位姿变化，根据空间的位姿变化矩阵，我们可以将第一帧图片的三维模型经过位姿转换与第二帧图片的三维模型合并，从而得到第一帧和第二帧合并后的三维模型，依次迭代，我们可以跟随相机变化，实时输出合并后的三维模型。基于前面的理论，合并后的三维模型都是基于之前相机拍下的图片，以及对该图片的处理过程，但是在以上的每个过程中，都会产生一些误差。我们可以预见，无论多小的误差，在达到足够大的迭代次数时，它将变得不可忽视，与此同时，经过实验得出，未经过优化的三维模型会产生很大的累积误差。

下面分别展示三张点云图，图 4.1 是由一张 RGB 图和一张深度图直接生成的三维点云图的截图。直接由一张 RGB 图和一张深度图生成的点云图没有累积误差，所以从图 4.1 中可以看出物体的位置是正确的。从图 4.2 和 4.3 的我们可以清楚看出，未加入回环检测的点云图存在很多的错误点，表现在：图 4.2 中，图左边的人影模糊，房间墙面模糊，不能够区分出墙的位置，而图 4.3 中人体清晰，可以清晰分出墙面。图中有多个人影是因为人在空间内走动，在此我们不关注。

图 4. 1　一张 RGB 图和一张深度图生成的点云图

Fig. 4.1　A point cloud generated by a depth map and a RGB graph

图 4.2　未加回环检测时生成的点云图

Fig. 4.2　A point cloud generated in the absence of loop detection

图 4.3　加入回环检测时生成的点云图

Fig. 4.3　A point cloud generated by loop detection

4.2　拓扑图优化原理

4.2.1　拓扑图优化问题建模

在一般的 SLAM 系统中，三维建模优化部分的实现方法是拓扑图优化，而这部分优化问题又可以转换成一个非线性最小二乘法问题，一般的非线性最小二乘问题的描述如式（4.1）。

根据式（4.1），x_k 代表的是相机的每个时刻的位置及姿态，拓扑图由节点和边组成，

节点就是相机每个时刻的位置和姿态，边为位姿之间的联系，从相邻的两个节点出发，讨论拓扑图优化理论。若两个相邻节点的位姿分别为 x_i 和 x_j，那么这两个节点间的相位位姿为 $x_k=(x_i, x_j)$。相机从 x_i 位姿到 x_j 位姿时，我们观测并且计算得到的结果为 z_k，那么最后我们的误差为 $e_k=(x_k, z_k)$。e_k 越大表示误差越大。拓扑图优化过程也是是 e_k 趋近为零的过程。

$$F(x)=\sum_{k\in\mathbb{C}}\underbrace{e_k^T(x_k,z_k)W_k e_k(x_k,x_k)}_{F_k} \tag{4.1}$$

$$x^*=\arg\min_x F(x)$$

如图 4.4 所示，由于从位姿 x_i 到位姿 x_j 需要用一条边连接起来，z_k 就是那条边，同时由于误差的存在，Ω_k 就是转换过程中的误差范围。

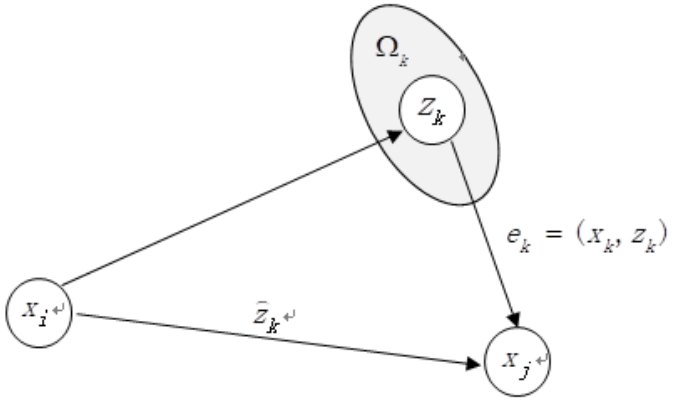

图 4.4　拓扑图结构

Fig. 4.4　The structure of topological graph

由于从节点 x_i 到节点 x_j 必定有一条边，也即转换过程，而 $x_k=(x_i, x_j)$，所以可以将这个转换函数写成 $\hat{z}_k=h_k(x_k)$ ，那么误差函数可以写成下面这种形式：

$$e_k(x_k,z_k)=h_k(x_k)-z_k \tag{4.2}$$

为了方便下面的表示，对于上面的符号做以下简单化：

$$e_k(x_k,z_k)\triangleq e_k(x_k)\triangleq e_k(x) \tag{4.3}$$

4.2.2　非线性最小二乘问题计算

若相机的初始位姿已知，并且这个位姿是比较合适的，那么可以进行 $\breve{x}$ 处的误差函数的一节泰勒展开，将误差以一阶泰勒线性表示出来，如式（4.4）。这个方法是采

用高斯-牛顿和 *LM* 等经典的方法，所用的原型是式（4.1）。

$$\begin{aligned} e(\breve{x}_k + \triangle x_k) &= e_k(\breve{x} + \triangle x) \\ &= e_k + J_k \triangle x \end{aligned} \tag{4.4}$$

将 $e_k(x)$在 $\breve{x}$ 处展开，可以得到一个雅克比矩阵，这个矩阵就是 J_k，并且有公式 $e_k(x) \overset{\triangle}{=} e_k(\breve{x})$。

$$J_k = \frac{\partial e_k(x)}{\partial x} \tag{4.5}$$

将所有的参数重新组成一个状态向量可以得到 $x=(x_1,\ldots,x_n)^T$。

将方程（4.4）替换到公式（4.1）中的 F_k 中，得到：

$$\begin{aligned} F_k(\breve{x} + \triangle x) &= e_k(\breve{x} + \triangle x)^T \Omega_k e_k(\breve{x} + \triangle x) \\ &= (e_k + J_k \triangle x)^T \Omega_k (e_k + J_k \triangle x) \\ &= \underbrace{e_k^T \Omega_k e_k}_{c_k} + 2\underbrace{e_k^T \Omega_k J_k}_{b_k} \triangle x + \triangle x^T \underbrace{J_k^T \Omega_k J_k}_{H_k} \triangle x \\ &= c_k + 2b_k \triangle x + \triangle x^T H_k \triangle x \end{aligned} \tag{4.6}$$

上述等式进行了局部展开，可以将公式（4.1）中的 $F(x)$继续往下推导，从而得到：

$$\begin{aligned} F_k(\breve{x} + \triangle x) &= \sum_{k \in \mathbb{C}} F_k(\breve{x} + \triangle x) \\ &= \sum_{k \in \mathbb{C}} (c_k + 2b_k \triangle x + \triangle x^T H_k \triangle x) \\ &= c + 2b^T \triangle x + \triangle x^T H \triangle x \end{aligned} \tag{4.7}$$

在上式中，令 $c = \sum c_k$ ，$b = \sum b_k$ ，$H = \sum H_k$ ，则 $F_k(\breve{x} + \triangle x)$ 为Δx 函数的二次形式，所以，只要让Δx 在求解微分后的值为零，那么这个时候的Δx 就是最小值Δx^*，可以用下面方式来表述：

$$H \triangle x = -b \tag{4.8}$$

矩阵 H 是系统的信息矩阵，论文中称 H 为系数矩阵，认为 b 是一个系数向量。构建之后的 H 矩阵当且仅当在变量相交接的地方并且必须有约束条线相连的情况下才存在非零值，所以 H 矩阵就是个稀疏矩阵。基于此，我们可以用稀疏 Cholesky 分解、PCG 算法等方法来求解公式（4.8）。研究者已经研究出很多更高效的稀疏 Cholesky 分解算法，并且将他们放在网上供大家一起学习使用，例如 CHOKMOD[26]或者 CSparse。所以，我们通过在之前的假设的基础上对其进行计算得到下面的增量，可以得到下面线性化公式的解：

$$x^* = \breve{x} + \triangle x^* \tag{4.9}$$

经典的高斯-牛顿算法对公式（4.7）进行线性化运算、对公式（4.8）进行求解运算和更新公式（4.9）的值是采用的方法都是迭代。每次迭代初始阶段，需要将上一次迭代的结果当做本次迭代初始条件。在高斯-牛顿算法的非线性化方面，LM 对其进行了优化，并且为了可以让函数收敛结果更有效，对优化过程增加了阻尼系数和后退过程。和高斯-牛顿算法比较来看，LM 算法是在给公式增加一个阻尼系数后，在对公式进行运算，这个过程如式（4.10）所示：

$$(H + \lambda I)\triangle x^* = -b \tag{4.10}$$

其中，λ 是阻尼系数，λ 与Δx 呈负相关关系，这种性质有利于算法寻找最优的步长。LM 算法还有一个优点，那就是能够在算法运行过程中调整阻尼系数，通过比较每次迭代的结果进行动态调整阻尼系数。这个调整过程可以形容成：若求解出来的误差慢慢收敛则继续缩小阻尼系数，若误差增大则增大阻尼系数。文献[27]中对 LM 算法进行了详细的推导和验证。

上面的计算过程的前提是最小化多变量函数，这是他的经典的求解的方法。由于在 SLAM 问题中，我们无法知道 x 是否处于欧式空间中，而对于上面的求解方式来说，预先假设 x 在欧式空间中是必须的，所以 SLAM 问题求的解可能不会是最优的解。对此，需要一种方法能够将这种方法拓展到费欧式空间中，用以求解本文中的 SLAM 问题，这样用来解决拓扑图的优化问题。

4.2.3 计算非欧式空间中的最小二乘问题

基于本文的 SLAM 问题的相机姿态变换不实在欧式空间的框架下，为了解决这个问题，本文采用一种流行空间中计算误差函数最小值的方法。流行空间是一种比欧式空间含义更广泛的数学空间，他的局部有欧式空间的特性。

在本文的 SLAM 问题中，相机的位姿变换是有旋转矩阵 α_i 和平移向量 t_i 组成，我们可以将 t_i 看作一个欧式空间，但是 α_i 属于群论，α_i 在 $SO(2)$ 或者 $SO(3)$的范围框架下。同过将欧式空间和群论进行参数化描述，我们可以使用四元数和旋转矩阵来表示旋转。将公式（4.10）采用参数化的描述方式，会使得他的约束条件破坏，并且经过这个描述方式是的该方法的自由度更高从而增加了复杂度，就是误差更大。通常上面这些问题可以使用欧拉角等最小描述法表示旋转的方式来解决，但是这样奇异解的问题有重新显露出来。

上面叙述的问题可以通过流行空间解决，方法是将上面的算法在流行空间的框架下进行计算，这里我们会重新定义一个运算符$\oplus$，这个运算符的作用是，将欧式空间中的

部分局部增量通过映射的手段放置到流行空间中，表示为$\Delta x \to x \oplus \Delta x$。使用这个运算符计算误差函数的表达方式可以写成下面这个形式：

$$\begin{aligned} \breve{e}_k(\vartriangle \breve{x}_k) &\triangleq e_k(\breve{x}_k \oplus \vartriangle \breve{x}_k) \\ &= e_k(\breve{x}_k \oplus \vartriangle \breve{x}_k) \\ &= \breve{e}_k + \breve{J}_k \vartriangle \breve{x} \end{aligned} \tag{4.11}$$

其中，$\breve{x}$ 在刚开始的过参数化的空间里，$\breve{x}$ 代表的是起始点，

其中，$\breve{x}$ 在最初的过参数化空间当中，$\breve{x}$ 表示起始点，用$\Delta \breve{x}$ 代表 $\breve{x}$ 邻近范围的无限趋近于零的变化量。$SO(3)$可以用单位四元数表示。我们可以使用 $\vartriangle \breve{x}^T = (\vartriangle t^{-T} q^{-T})$ 代表增量$\Delta \breve{x}$，从前面的叙述中可以得知$\Delta \breve{x}$ 是一个 6 维的向量，由于 t 为平移向量，与 $\breve{x}$ 类似，$\Delta \bar{t}$ 代表的是平移变换的 t 邻近范围的无限趋近于零的变化量。那么旋转变换就可以表示成 $q^{-T} = (\vartriangle q_x, \vartriangle q_y, \vartriangle q_z)^T$。反过来说，四元数 $\bar{q}$ 的 $\breve{x}^T = (\breve{t}^{-T} \bar{q}^{-T})$ 写法也可以用来表示旋转变换。所以有下面这种对运算符$\oplus$的解释方法，首先在一个全四元数 $\vartriangle y$ 中加入一个 $\vartriangle \breve{y}$，然后给 $\bar{x}$ 增加增量 $\vartriangle x^T = (\vartriangle t^T \vartriangle q^T)$。上面叙述的过程是可以在最小化误差方程的阶段使用运算符$\oplus$表示的。那么，雅克比矩阵 J_k 可以有下面的表示方法：

$$\begin{aligned} J_k &= \left. \frac{\partial e_k(\breve{x} \oplus \vartriangle \breve{x})}{\partial \vartriangle \breve{x}} \right|_{\vartriangle \breve{x}=0} \\ &= (0 \ldots 0 \quad J_{k_1} \ldots J_{k_2} \ldots J_{k_q} \quad 0 \ldots 0) \end{aligned} \tag{4.12}$$

鉴于在之前求解 $\breve{e}_k$ 的过程中，$\vartriangle \breve{x}_{k_i} \in \vartriangle \breve{x}_k$ 存在其中。所以公式（4.12）可以使用下面的公式表达出来：

$$J_k = \left. \frac{\partial e_k(\breve{x} \oplus \vartriangle \breve{x})}{\partial \vartriangle \breve{x}} \right|_{\vartriangle \breve{x}=0} \tag{4.13}$$

使用公式（4.13），我们能够得出下面增量的表示方法：

$$\breve{H} \vartriangle \breve{x}^* = -\breve{b} \tag{4.14}$$

因为 $\vartriangle \breve{x}^*$ 处于欧式空间中，我们可以通过运算符$\oplus$，并且 $\breve{x}$ 为中心将 $\vartriangle \breve{x}^*$ 投影回原来的空间。所以，公式（4.14）可以写成这个样子：

$$x^* = \breve{x} \oplus \vartriangle \breve{x}^* \tag{4.15}$$

从上面得出，在流行空间计算最小化的问题目前还有两个研究方向，其一是使用公式（4.14）的结论，在估计一个初始值的情况下，首先通过欧式空间计算一组增量，然后在使用公式（4.15），在流行空间的框架下进行初始估计值和增量的叠加操作。这里，我们只需要定义一个运算符$\otimes$，同时计算出与这个运算符相对应的雅克比矩阵 $\breve{J}_{k_i}$，那么

就能很轻松地在流行空间中使用欧式空间的相关结论，在本文中，可以使用拓扑结构的图优化问题。本文使用 G2O 的优化库，由于 G2O 库内部集成有在流行空间的雅克比矩阵求解方法，所以通过调用 G2O 库就能够轻松实现图优化。

4.3 拓扑图优化实现

4.3.1 关键帧筛选策略

我们可以很容易地从 RGB-D 相机输出的 RGB 图片和 Depth 图片得到与当前帧相对应的三维点云图，上一章已经计算得出相邻两帧间的旋转矩阵和平移向量，通过将上一帧的三维点云图经过旋转和平移叠加到下一帧的三维点云图，依次迭代就可以得到全部的点云图。但是，目前这种做法还有些许问题：其一，RGB-D 相机每秒钟可以输出几十帧图片，由于相机运动不会很快，所以相邻两帧间会有很多重合的信息，如果将每一帧都计算并加入到三维点云图，这时候会消耗大量的计算资源，同时由于很多重合，对系统的精度提升并不大；其二，前面章节提到相机拍照以及提取特征点、描述子、计算位姿变化时会有累积误差，如果将每一帧都加入三维点云图，会照成误差被放大。

针对上面的问题，本文采用了一种关键帧筛选的策略来进行场景构建，他的主要思想是在确保不丢失点云信息的情况下，使用尽量少的图像帧来还原整个三维模型，被选中的帧就是关键帧。这样做有两个好处，其一是加快了三维重建速度，其二是减少了图像间累积误差的影响。本论文使用一种基于范数的关键帧筛选策略。

本文中筛选关键帧的流程是：

（1） 设置相邻两关键帧间的最小不一致性 NormMin 和最大不一致性 NormMax；

（2） 将第一帧作为关键帧加入关键帧序列 KF；

（3） 从帧序列中按序取出一帧，与 KF 末尾的关键帧比较；

（4） 若该帧与 KF 最后一帧的不一致性在 NormMin 和 NormMax 的范围内，那么将该帧加入关键帧序列尾部；

（5） 判断该帧是否为最后一帧；

（6） 是最后一帧，结束；

（7） 跳转到第（3）步。

相邻两帧间的关系如图 4.5 所示。

根据图 4.5 中的转换关系，我们可以列出下面的式子：

$$\dot{L} = R * L + t \tag{4.16}$$

其中 L 为上一帧，$\dot{L}$ 为当前帧，R 为旋转矩阵，t 为平移向量。

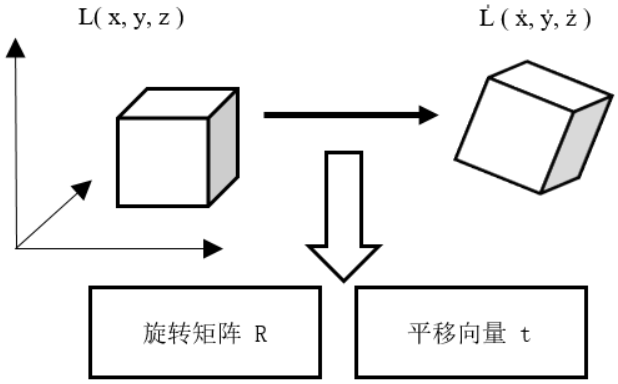

图 4.5　相邻两帧之间转换关系图

Fig. 4.5　Conversion diagram between two adjacent frames

由于比较的相邻两帧间的变换关系由旋转和平移两部分组成，可以通过下式将两帧之间的变化转换为一个量来表示。

$$Norm = min(\| R \|, 2\pi - \| R \|) + \| t \| \quad (4.17)$$

|| R ||表示 R 的范数。由于我们视物体的方向向量从(1,0,0)变化到(0,1,0)和从(1,0,0)变化到(0,0,1)的旋转变化量是一样的，尽管 3D 空间会有无数个圆，但这里可以取最大的旋转变化为 π。

本论文可以通过改变 NormMin 和 NormMax 的值来决定我们选取关键帧的密度。一般而言，通过 NormMin 来减少点云的稠密度，通过 NormMax 来减少误差帧。

在调试的时候，我们需要注意一点，由于每次比较都是将当前帧与关键帧的最后一帧进行比较，可以知道，两张相同的帧的 Norm 值为 0。当把该帧加入关键帧末尾时，下一帧往往因为 NormMin 的限制被剔除，若 NormMax 设置过小，可能发生在下下帧的时候 Norm 值大于 NormMax，从而导致接下来的所有帧都被判定为误差帧。

4.3.2　优化实现

通过关键帧的筛选，减少了累积误差和冗余信息，这些关键帧的位姿可以作为拓扑图上的节点，每个节点都有位置信息和方向信息，系统每增加一个节点，就给拓扑图增加了一条边，每次增加一个节点后的新的拓扑图就是一个待优化的拓扑图，调用 G2O 库函数执行优化就可以优化该拓扑图。

我们通过相机将现实中的物体拍下来，记录在照片中，设三维中物体（图 4.6 的三维点估计位置）应该在照片中的点为 *A*（图 4.6 的二维特征点），实际上因为重投影误差，利用相机矩阵进行运算后的物体位置为点 *B*（图 4.6 的重投影位置），那么点 *B* 到 *A* 与三维中物体直线的距离即为重投影误差。

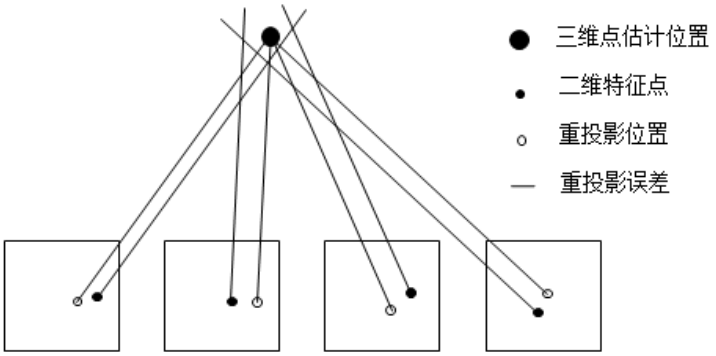

图 4.6　当两幅关键帧信息相似时发生闭环

Fig. 4.6　The loop closure occurs when two keyframes have similar information

在这里，我们使用 i 表示第几帧图像，j 表示图像中的第几个特征点，假设有 M 个帧图片，每张图片上有 N 个特征点，那么可以用下面的式子表示：

$$\begin{aligned} z_1 &= \left\{ z_1^1, z_1^2, ..., z_1^N \right\} \\ z_2 &= \left\{ z_2^1, z_2^2, ..., z_2^N \right\} \\ &\vdots \\ z_M &= \left\{ z_M^1, z_M^2, ..., z_M^N \right\} \end{aligned} \tag{4.18}$$

三维点（物体，相机）到二维点（投影）的转换关系如式（4.19）所示：

$$d_j \begin{pmatrix} z_i^j \\ 1 \end{pmatrix} = K\left(R_i X^j + t \right) \tag{4.19}$$

在这里，K 是摄像机的内部参数的矩阵，d 是 RGB 图像中每个像素点的深度信息，R 是相机的旋转矩阵，t 是相机的平移向量，d 可以通过深度图获得。所以可以通过下面式子定义重投影：

$$\frac{1}{d_j} K\left(R_i X^j + t \right) - \begin{pmatrix} z_i^j & 1 \end{pmatrix}^T \tag{4.20}$$

将重投影误差最小化后，可以表示成下面这样：

$$\min_{X,R,t} \sum_{i=0}^{M} \sum_{j=0}^{N} \left\| \frac{1}{d_j} K\left(R_i X^j + t \right) - \begin{pmatrix} z_i^j & 1 \end{pmatrix}^T \right\|^2 \tag{4.21}$$

在理想环境中，如果我们将拓扑图中的每个节点用 X 表示，T 表示节点间的转换关系，也即边，那么一个闭环内的所有边的积将为单位阵，也即经过这些边的转换节点又回到了原点，用数学表示为：

$$T_0 T_1 \ldots T_n = I \tag{4.22}$$

产生等式（4.22）的原因是产生回环的原因是两个节点是同一个节点，也即当时相机所处的姿态和位置都一样，但是由于观测和计算时候的偏差，导致最后结果两个节点不在一个位置，只有当满足闭环条件是，式（4.22）才能成立。若我们已知有闭环发生，并且人为地告诉优化器哪两个节点是相同的，那么，就可以通过优化的方法找到一个尽量符合实际的相机位姿拓扑图曲线。

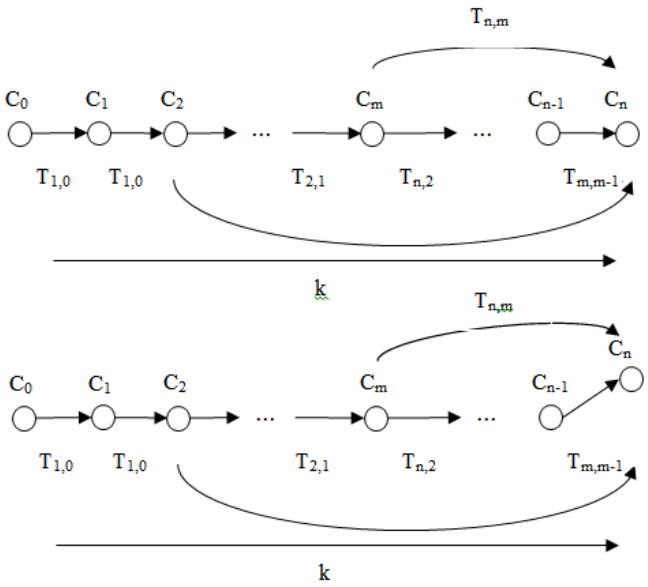

图 4.7　拓扑图优化过程

Fig. 4.7　The process of topology optimization

如图 4.7 所示，已知 C_m 和 C_n 发生了回环，优化器会将 C_m 和 C_n 拉近。

如式（4.1）所示，设 x_k 为摄像机的位置和姿态信息，在这里，我们定义一个消耗函数：

$$x_k = (t^T,\ r^T)^T \tag{4.23}$$

等式（4.23）中的 t 是摄像机的平移向量，他是一个三维的向量，表示为 $t=(t_x, t_y, t_z)$。r 是摄像机的旋转矩阵，等式中的 r 是一个四元数，也即一个四维向量，则 x_k 是一个七维向量，在 G2O 的存储文件中，就是以 x_k 这种方式存储节点信息。通过这种表示方法，在求解相机位姿的过程中不会出现奇异解。同理Δx_k 可以表示为。

$$\triangle x_k = (\triangle t^T, \triangle r^T)^T \tag{4.24}$$

由于 r 四元数有一个数固定不变，所以Δr 可以认为只有三维，而Δt 不变，还是三

维。

至此，我们已经得到了摄像机位姿拓扑图中的节点和边，下面我们需要一种运算符来实现节点通过边到另一个节点的运算过程，同样还需要这个运算符的逆，在这里，我们设这个运算符为⊕，他的逆为⊙。

$$\begin{aligned} x_k \oplus_{\Delta} x_k &= x_k \oplus_{\Delta} x_{k(7\times1)} = \begin{pmatrix} t \\ r \end{pmatrix} \oplus \begin{pmatrix} \Delta t \\ \Delta r \end{pmatrix} = normalize\left(\begin{bmatrix} t + r \bullet \Delta t \\ r \bullet \Delta r \end{bmatrix} \right) \\ x_a \odot x_b &= \begin{pmatrix} t_a \\ t_b \end{pmatrix} \odot \begin{pmatrix} t_b \\ t_a \end{pmatrix} = normalize\left(\begin{bmatrix} conj(r_b) \bullet (conj(r_b) \bullet (-t_b)) + conj(r_b) \bullet t_a \\ cong(r_b) \bullet r_a \end{bmatrix} \right) \end{aligned} \tag{4.25}$$

根据邻近的两个节点之间的边，我们就可以通过式（4.23），求得节点在三维空间的位姿变换。

$$h_{i,i+1}(x_i, x_{i+1}) = x_{i+1} \odot x_i \tag{4.26}$$

至此，误差 $e(x_k, z_k)$就表示成：

$$e(x_i, x_j) = e(x_i, x_{i+1}, z_{ij}) = z_{i,i+1} \odot h_{i,i+1}(x_i, x_{i+1}) \tag{4.27}$$

经过上面的推演验证，已将将 SLAM 拓扑图优化问题转变成非线性最小二乘法问题，接下来就可以使用 G2O 工具进行优化。

4.4 基于 GPU 的三维场景构建

前面几章已经详细介绍了从 RGB-D 相机到最后三维建图过程。在实现该 SLAM 快速建图的过程中，仍有下面几个地方限制了该 RGB-D SLAM 系统的速度：特征计算与描述子计算、特征匹配、相机位姿计算、回环检测、三维建图。

本文使用 GPU 加速三维建图的过程。三维建图在本文中的步骤为：特征提取——描述子计算——图像匹配——位姿计算——点云叠加——点云转换为八叉树。本文中使用 GPU 的过程是点云叠加的过程，因为已经计算出相机的位姿，将两个点云图叠加的时候需要将每个点都用旋转矩阵和平移向量计算一遍，这里会非常耗时间，并且都是重复的计算，所以考虑在 GPU 上运行点云旋转和平移的过程，再将计算完成的点云返回给 CPU 转换为八叉树。

前面的研究者已经将 GPU 运用于 SLAM 系统中，他们使用 GPU 来运行特征算法和使用 GPU 进行滤波。本论文使用 GPU 对 SLAM 系统中三维建图过程进行并行加速。

在本论文的 RGB-D SLAM 系统中，获得关键帧后，需要将关键帧转换为单帧点云，在拓扑图优化之后，将转换完成的单帧点云根据优化后的位姿进行叠加，合成全局点云。本论文使用 GPU 对点云叠加的过程进行加速。

点云叠加的过程其实就是一个坐标系转换的过程，转换矩阵可以通过优化后的位姿

提取。叠加所有帧其实就是将每一帧的每个点乘以该帧的转换矩阵的过程。本论文使用 GPU 并行算法来完成这个过程。本论文 GPU 算法原理图如图 4.8 所示。

图 4.8 中，灰色块为关键帧，空白块为未加入关键帧序列的帧。在本论文的设计中，为每个关键帧分配一个线程，当关键帧数目超出 65535 时使用其他 Block 分配线程。由图可知，对每帧图片来说，它的叠加过程是并行的。

由于 GPU 运算只能使用 GPU 内存，所以，需要先为 GPU 分配足够多的内存。本论文中设计的 GPU 函数共有四个指针参数，分别为：allFrameVertexXYZ、frameVertexRange、matrix4f 和 frameSize。它们的意义分别为：所有关键帧的所有点的坐标、每一个关键帧点坐标在 allFrameVertexXYZ 中的范围、每一个关键帧的转换矩阵及关键帧数目。它们需要分配的空间大小如表 4.1 所示。

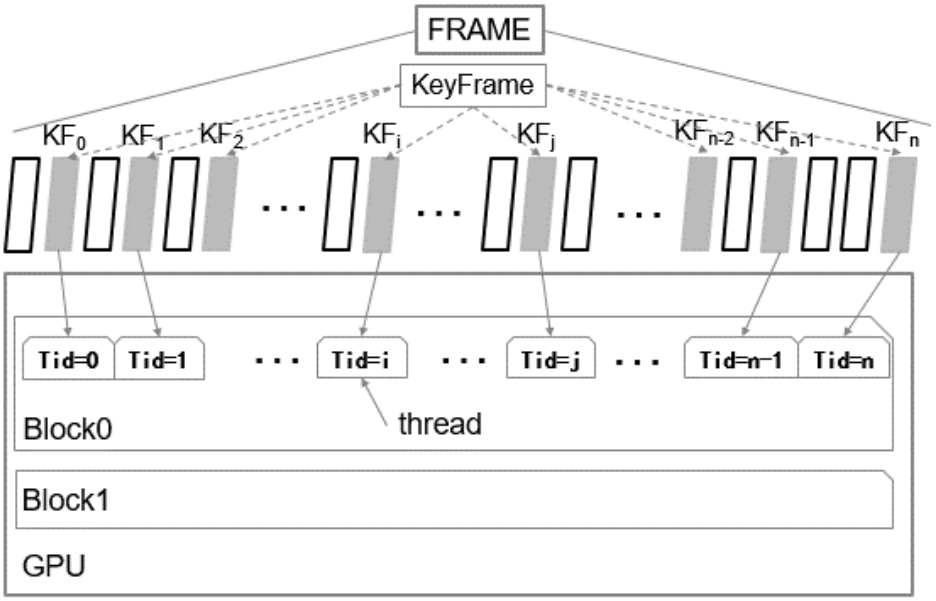

图 4.8　GPU 算法原理图

Fig. 4.8　Schematic diagram of GPU algorithm

表 4.1　每个参数需要分配空间大小

Table. 4.1　Each parameter needs to be allocated space size

参数名称	类型	分配空间大小
allFrameVertexXYZ	Float *	frameVertexRange[n-1].tail + 1
frameVertexRange	Int *	N*2
matrix4f	Float *	N*12
frameSize	Int	1

虽然转换矩阵是一个 4×4 的矩阵，但是第 4 行只有第 4 列为常数 1,所以在存储的时候可以不用存储第四行，于是变量 matrix4f 需要分配的空间大小为 N*12；变量 frameVertexRange 中需要按顺序存储每一帧的头指针和尾指针，所以它的大小为 N*2；变量 allFrameVertexXYZ 的大小可以通过变量 frameVertexRange 来获得，只要知道最后一帧的尾指针的位置即可知道 allFrameVertexXYZ 需要分配的空间大小。

为了使 GPU 运算过程更快，需要减少访问存储的时间。由于计算机顺序访问时比随机访问快很多，所以在分配内存空间时应当连续分配坐标数据。

通过分配内存，将数据拷贝进 GPU 内存就可以使算法在 GPU 上运行，GPU 运行结束需要将结果拷贝出来。

5　实验设计及结果

前面几章已经详细叙述了特征选取、描述子计算、图像匹配、回环检测、三维显示等步骤的实现过程。这一章将讲述上面内容的实现过程以及通过一些实验手段来论述本文的设计，本章会对每一个流程进行叙述及实验结果比较。

5.1　系统组成

本文的系统有以上讲到的部分组成，本系统是一个基于 RGB-D 的 SLAM 系统。系统组成如图 5.1 所示。本论文的 SLAM 系统分为前端和后端两个大部分，其中前端包含系统输入、图片处理、特征匹配四部分，后端包括三维建图和位姿优化两部分。最后的目的就是生成三维建图中的八叉树图。

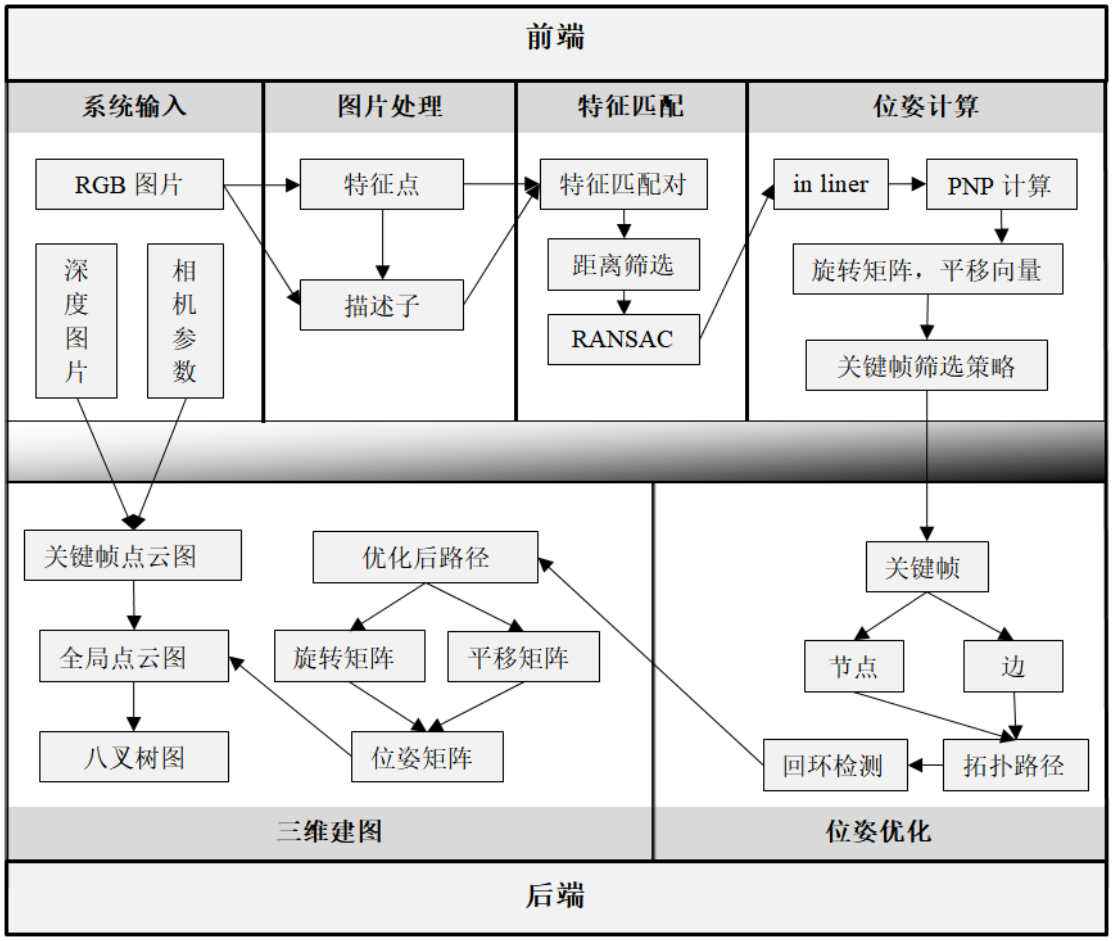

图 5.1　本论文 SLAM 系统组成图

Fig. 5.1　The composition diagram of the SLAM system in this thesis

其中，本论文使用了基于 ORB 的特征提取和描述子计算算法，ORB 算法的特征提取部分使用的是 FAST 算法，计算描述子部分使用的是 BRIEF 算法，在特征匹配阶段使用了基于 FLAAN 的匹配算法可以更快速地匹配特征点，同时使用了基于特征点距离匹配的筛选算法和 RANSA 算法，能够更加准确地悬着出特征点对，位姿计算使用的是 PNP 算法，使用 PNP 算法可以将特征点对作为输入，输出旋转矩阵和平移向量，然后通过关键帧筛选策略筛选出关键帧，根据关键帧的旋转矩阵和平移向量得到拓扑图的节点和边，通过回环检测和 G2O 的路径优化得到优化后的路径，根据优化后的路径获得关键帧相机的位姿信息从而将关键帧生成的点云合并生成全局点云图，最后将全局点云图转换为八叉树。

5.2 软件依赖

本 RGB-D SLAM 系统运行在联想 V480 的便携计算机中（笔记本计算机），CPU 为 i5-2520M，GPU 为 GT 630M，其中本系统的软件依赖如下。

计算机的操作系统：Ubuntu 16.04.3 LTS。

Linux 内核版本：4.13.0-32-generic(build@lgw01-and64-004)。

GCC 版本：5.4.0。OpenCV 版本：2.4.9。PCL 版本：1.8.0。G2O 版本：20170730。OCTOMAP 版本：1.9.0。CUDA 版本：8.0。显卡驱动版本：nvidia-384。QT 版本：qt4。

5.3 实验设计及实验结果

（1） 系统输入

系统的输入 Kinect 的 RGB 图片和 depth 图片，其中一帧的图片如图 5.2 所示。由于原始深度图片在肉眼看来都是黑色，将深度图色调调整为 1500，饱和度调整为 0。

图 5.2 Kinect 相机输入的一帧 RGB 图片和 Depth 图片

Fig. 5.2 A frame of RGB and Depth pictures entered by the Kinect camera

左边的是 RGB 图片，右边的是深度图片，由于深度图片中存储的只是深度的值，所以看起来乌黑一片，其实里面有像素点的深度信息。

（2） 图片处理

将图 5.2 的 RGB 图进行特征点提取和描述子计算

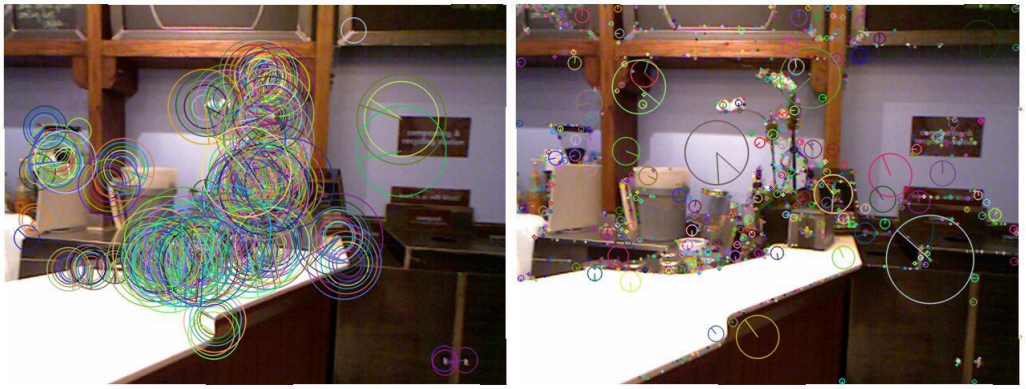

图 5. 3　特征提取 ORB（左）SIFT（右）

Fig. 5.3　Feature extraction ORB (left) SIFT (right)

本系统使用 ORB 算法进行特征提取，这里仅将 ORB 与 SIFT 进行比较，详细的必将在第 3.1 节进行了详细的关于特征点数量以及速度的比较。

（3） 特征匹配

将图 5.3 提取的特征点进行匹配，并使用距离筛选和 RANSAC 优化得到准确的特征匹配对。

本文使用 FLAAN 匹配算法，FLAAN 匹配算法能够在匹配道足够的特征点对数后立即停止匹配，实效性比 BF 方法好，在 FLANN 匹配后再进行基于方向的距离筛选，可以看出最后在进行 RANSAC 优化，得到图 5.4 下的图片，其中 RANSAC 优化后的特征点对已经比初始特征点对正确性提高了很多，特征点对也少了很多，减少了计算量。

根据图 5.4，可以得出下面的结论：

① FLANN 算法能够提取出大量的、足够数目的特征点对，但其中有很多时错误的匹配；

② 使用优化的基于方向的距离筛选的算法，能够去掉大量的错误匹配，仍然不能够去除距离和正确匹配差不多的错误匹配；

③ 使用 RANSAC 算法对距离筛选后的特征点对对再次筛选，能够得到近乎完美的匹配。

由于 RANSAC 算法比较耗时，所以这里先使用基于方向的距离筛选。

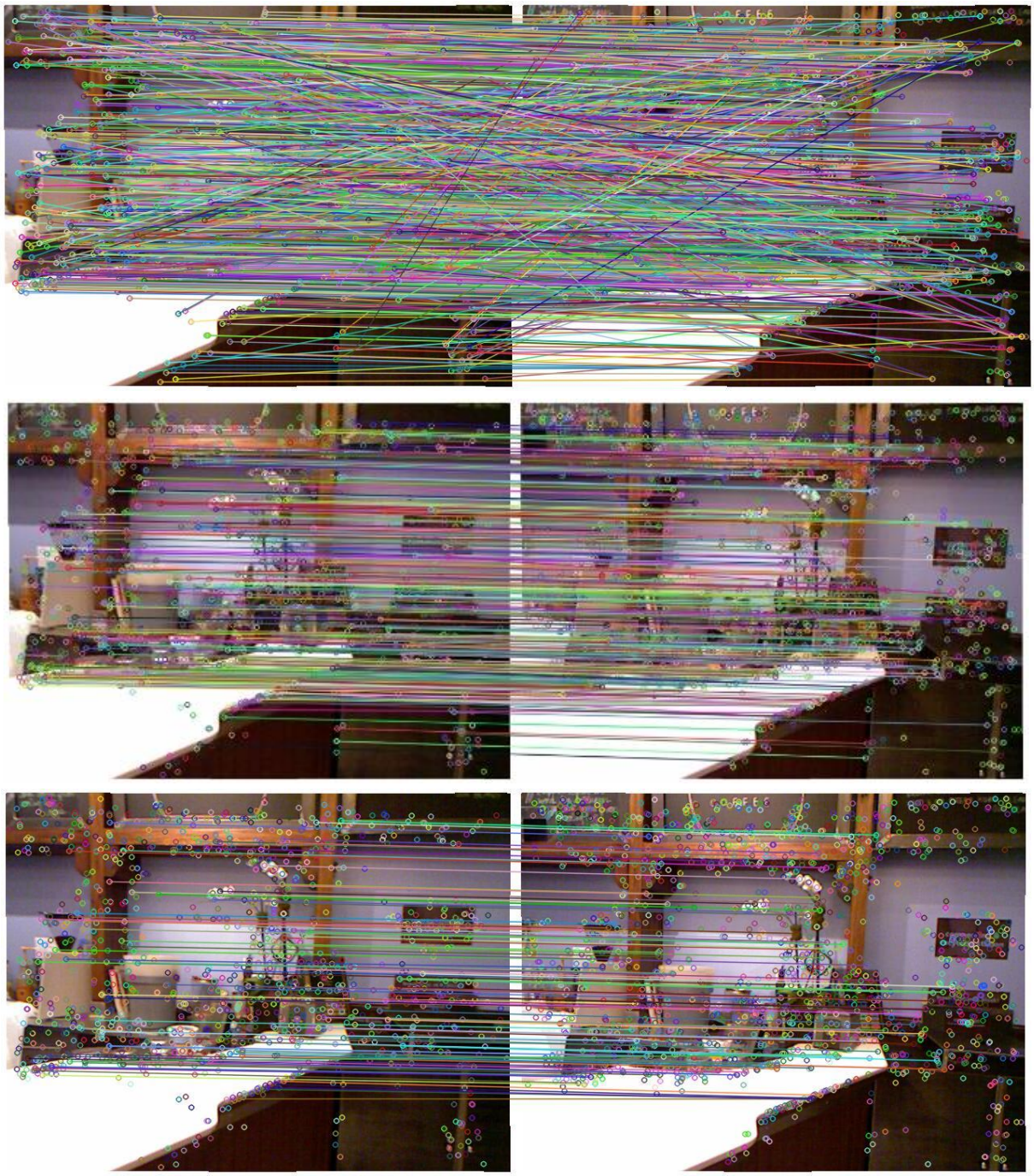

图 5.4　特征匹配结果 FLAAN（上）、距离筛选（中）、RANSAC（下）

Fig. 5.4　Feature matching results of FLAAN (upper), distance screening (middle), RANSAC (below)

（4） 位姿计算

将计算得出的特征点对经过 RANSAC 优化后，进行 PNP 计算可以得出相机的旋转矩阵和平移向量。

```
Key points of two images: 1364, 1728
Find total 1364 matches.
min dis = 0.00816368
good matches=339
inliers: 110
R=[-0.024237218415694;
  0.03317233303199228;
  0.01489879248415617]
t=[0.02194903839922979;
  0.009013527929348213;
  0.01426638136833981]
```

```
Key points of two images: 1364, 1728
Find total 1364 matches.
Costed time by Flann: 0.030976 s.
min dis = 0.00816368
good matches=339
inliers: 109
R=[-0.02377374055725811;
  0.03303522040324439;
  0.01460124962106053]
t=[0.0216568507466905;
  0.01049981188660019;
  0.01206187917807096]
```

图 5.5　两次位姿计算结果

Fig. 5.5　Two position calculation results

（5） 位姿优化

根据图 5.5 中的计算得到的旋转矩阵 R 和平移向量 t，可以求得当前帧与关键帧之间的偏差 F，其中 F=function(R,t)，F 是以 R，t 为参数的输出。本文根据比较 F 判断当前帧与关键帧的偏差是否在合理范围，若超出范围则认为误差太大，若 F 太小则认为两帧太相似，重合太多，也舍弃，最后只将位置和角度合适的帧插入关键帧序列。

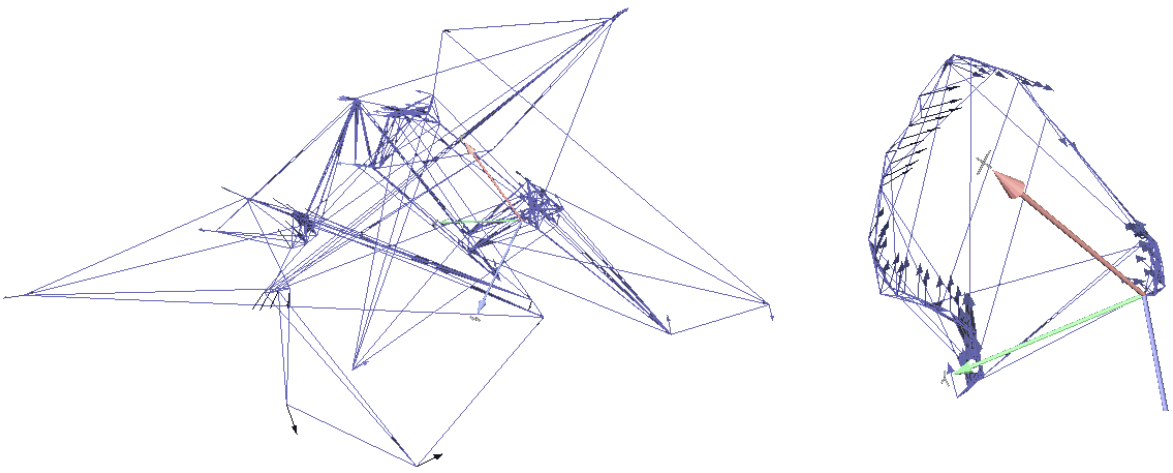

图 5.6　未优化和优化后的位姿图

Fig. 5.6　Pose map without optimization and optimization

从图 5.6 可以得出，未经过 G2O 优化的路径大体上正确，但是会有一些点偏离太远，经过 G2O 优化后，可以将偏离的点向正确的位置拉近，得到正确的位姿拓扑图。

（6） 三维建图

根据位姿优化后的相机位姿图生成下面的三维模型。

图 5.7　本文设计系统生成的三维图，点云图（左）、八叉树图（右）

Fig. 5.7　This thesis designs the three-dimensional graph generated by the system, the point cloud (left), the octree (right)

（7） GPU 加速

SLAM 系统在合成点云时需根据两帧之间的旋转矩阵和位移向量转换点云图，点云图中的每个点都需要转换，这里有大量相同的数学运算，所以本论文将这一部分放在 GPU 上执行。本论文设置 GPU 版与 CPU 版进行比较，其中系统的运行环境为：CPU 为 i5-2520M 双核四线程；GPU 为 GT 630M 流处理器 93 个。比较的结果如表 5.1 所示。

表 5.1　GPU 与 CPU 点云叠加时间比较

Table. 5.1　Comparison between GPU and CPU point cloud superposition time

序号	CPU		GPU	
	时间(us)	关键帧数目	时间(us)	关键帧数目
1	1708265	167	226235	162
2	1365736	149	200016	138
3	1100439	112	168422	112
4	715894	84	140176	80
5	575507	56	130485	60
6	284041	28	109692	33
7	181432	21	82389	19
8	95346	11	55721	12
9	33147	4	46301	4
10	25770	3	50163	3
11	16923	2	42146	1

从表 5.1 可以看出，不管是 CPU 还是 GPU，随着关键帧的增加，进行点云叠加所需的时间都在增加。画出 CPU 与 GPU 点云叠加曲线如图 5.8 所示。

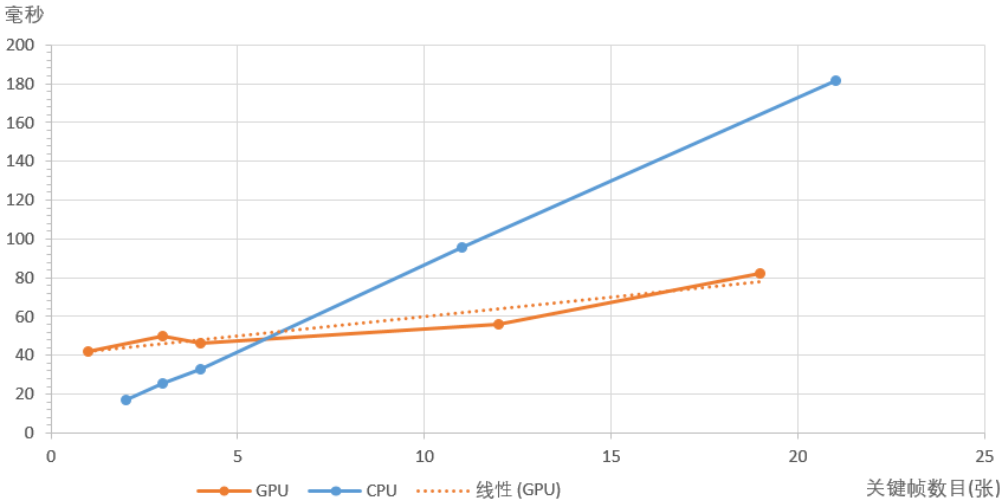

图 5. 8　CPU 与 GPU 点云叠加时间曲线

Fig. 5.8　CPU and GPU point cloud superposition time curve

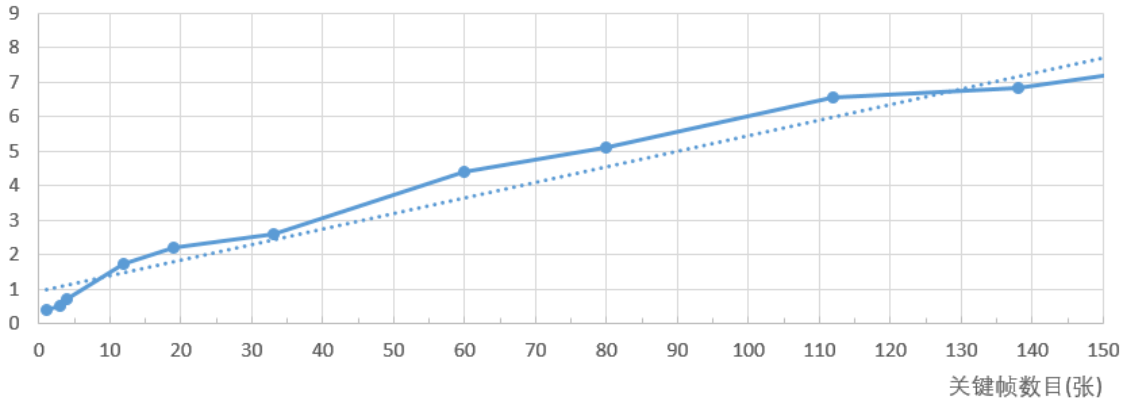

图 5. 9　GPU/CPU 点云叠加速度比

Fig. 5.9　GPU/CPU point cloud superimposed acceleration ratio

由图 5.8 可知，CPU 的点云叠加时间与关键帧数目近似呈正比关系，而 GPU 的点云叠加时间与关键帧数目呈正相关。经过分析，可以知道，由于使用 GPU 进行并行运算需要先将数据拷贝进 GPU，同时还有 GPU 初始化等操作，所以当关键帧为 0 时，GPU 所花的时间也不会为 0。从图中可以看出，当关键帧数目为 6 时，GPU 与 CPU 所用的时间一致，当关键帧数目大于 6 时，GPU 比 CPU 快。同理更直观地做出 GPU/CPU 点云叠加速度比曲线，如图 5.9 所示。从图 5.9 中可以看出，随着关键帧数目的增加，GPU/CPU 点云叠加速度比的值越来越大，验证了 GPU 的并行性，同时也证实了本论文使用 GPU 进行点云叠加可以加速三维建图过程。

结　论

本文设计的 SLAM 系统是以 RGB-D 相机生成的照片为基础，运用一系列图像处理算法和空间转换运算求得相机位姿，并依此合成点云，最后生成三维的八叉树模型。同时本系统分为前端和后端两个大部分，前端又分为系统输入、图像处理、特征匹配、位姿运算四个部分，后端分为位姿优化和三维建图。

前端部分，完成的任务是将 RGB-D 相机的图片处理后生成关键帧为后面的位姿优化做准备。这里只需要用到 RGB 图片，使用 ORB 算法进行图片的特征提取和描述子计算。在提取特征子方面，ORB 能够比 SIFT 和 SURF 快，同时 ORB 提取的特征子数目和质量也足够，可以说 ORB 算法速度和精度已经完全满足了本系统的使用。在特征匹配方面，本论文使用的是 FLAAN 算法，FLAAN 算法相比于 BF 算法能更快地匹配足够数目的特征匹配对。经过匹配的特征点对非常多，并且还有很多错误的匹配，本文加入了基于距离的筛选策略和 RANSAC 优化策略，距离筛选能够初步筛选特征点对。从本饰演的是剧中可得，距离筛选能够将特征匹配对数减少 3 倍，筛选掉的里面大部分都是错误匹配，而 RANSAC 优化又能够进一步优化匹配的特征点对。得到了筛选后的匹配点对，就可以根据相邻两帧之间的空间关系，结合之前标定的相机参数，使用 PNP 计算出相机在相邻两帧间的相对运动关系。这里加入一个关键帧策略，保证关键帧序列中的帧满足后面三维建图的要求，又要保证不会冗余太多信息，拖慢系统的速度。

后端部分，根据前端计算优化得到的关键帧和关键帧的位姿，以关键帧的 6D 姿态为节点，关键帧之间的转换关系为边，建立拓扑图。同时比对关键帧，找出其中的回环，将其作为边加入拓扑图。使用 G2O 优化工具，设定迭代次数就可以得到优化后的位姿拓扑图。优化后的节点包含的信息有优化后的该关键帧的空间位置和相机方向信息。将位置与方向重新组成位姿矩阵就可以用来进行点云图的拼接。关键帧的点云图由相机矩阵和深度图片组成，同时，如果需要彩色信息，就要加入 RGB 图。生成关键帧点云图后，利用关键帧的位姿进行点云图的旋转平移，最后将所有关键帧点云叠加，就得到了全局的点云图，然后将点云图转换为八叉树图。由于在进行点云旋转时需要将每个点进行旋转，这里会耗费大量时间，所以本文使用 GPU 运行这部分代码。

本文使用了 ORB 算法、FLANN、关键帧策略、GPU、距离筛选、RANSAC 算法、拓扑图优化保证了系统的效率和准确性。但是本系统不能够实时输出三维模型这在 AR、导航等应用方面有局限性。同时关键帧策略的设计也不尽人意，这个策略很依靠人为设定的参数，不能够自适应系统，同时本系统没有考虑运动场景。

参 考 文 献

[1] Nils J N. A mobius automation: an applicaton of artificial intelligence techniques [C]. International Joint Conference on Artificial Intelligence, Washington, DC, 1969: 509-520.

[2] Durrant-Whyte H, Bailey T. Simultaneous localization and mapping: part I [J]. Robotics& Automation Magazine, 2006, 13(2):99-110.

[3] Henry P, Krainin M, Herbst E, et al. RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments[M]. Experimental Robotics. Springer Berlin Heidelberg, 2014:647-663.

[4] Henry P, Krainin M, Herbst E, et al. RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments[J]. International Journal of Robotics Research, 2013, 31(5):647-663.

[5] Edward C, Vincent L, Pascal F. KeyPoint Signatures for Fast Learning and Recognition[C]. European Conference on computer Vision, Marseille, France, 2008:58-71.

[6] Zhao Z, Chen X. Building 3D semantic maps for mobile robots using RGB-D camera[J]. Intelligent Service Robotics, 2016, 9(4):1-13.

[7] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, W. Burgard. An Evaluation of the RGB-D SLAM System[C]. In Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA), 2012.

[8] Endres F, Hess J, Sturm J, et al. 3-D Mapping With an RGB-D Camera[J]. IEEE Transactions on Robotics, 2017, 30(1):177-187.

[9] Bay H, Tuytelaars R,Gool L V.SURF: Speeded Up Robust Features[J].Computer Vision & Image Understanding, 2006,110(3):404-417.

[10] Huang A S, Bachrach A, Henry P, et al.Visual odometry and mapping for autonomous flight using an RGB-D camera [C]. International Symposium on Robotics Research, Flagstaff, Arizona, USA, 2011:231-243.

[11] Audras C, comportA, Meilland M, et al. Real-time dense appearance-based SlAM for RGB-D sensors [C]. Australasion Conference on Rootics and Automation, Melbourne, Australia, 2011:156-163.

[12] Steinbrucker F, Sturm J, Cremers D. Real-time visual odometry from dense RGB-D images [C]. International Conference on Computer Vision, Barcelona, Spain, 2011:719-722.

[13] Whelan T, Johannsson H, Kaess M, et al. Robust real-time visual odometry for dense RGB-D mapping [C]. International Conference on Robotics and Automation, Karlsruhe, Germany, 2013:5724-5731.

[14] T. Whelan, H. Johannsson, M. Kaess, J. Leonard, and J. Mc Donald, Robust real-time visual odometry for dense RGB-D mapping [C], presented at the IEEE Int. Conf. Robotics Automation, Karlsruhe, Germany, May 2013:5724-5731.

[15] Hu G, Huang S, Zhao L, et al. A robust RGB-D SLAM algorithm[C]. Ieee/rsj International Conference on Intelligent Robots and Systems. IEEE, 2012:1714-1719.

[16] Zeng M, Zhao F, Zheng J, et al. Octree-based fusion for realtime 3D reconstruction[J]. Graphical Models, 2013, 75(3):126-136.

[17] Whelan T, Johannsson H, Kaess M, et al. Robust real-time visual odometry for dense RGB-D mapping [C]. IEEE International Conference on Robotics and Automation. IEEE, 2013:5724-5731.

[18] Kerl C, Sturm J, Cremers D. Robust odometry estimation for RGB-D cameras [C]. IEEE International Conference on Robotics and Automation. IEEE, 2013:3748-3754.
[19] Grisetti G, Kummerle R, Stachniss C, et al. Hierarchical optimization on manifolds for online 2D and 3D mapping [C]. IEEE International Conference on Robotics and Automation. IEEE, 2010:273-278.
[20] Izadi S, Newcombe R A, Kim D, et al. KinectFusion:real-time dynamic 3D surface reconstruction and interaction [C]. ACM SIGGRAPH. ACM, 2011:1-1.
[21] Konolige K, Agrawal M. FrameSLAM: From Bundle Adjustment to Real-Time Visual Mapping[J]. IEEE Transactions on Robotics, 2008, 24(5):1066-1077.
[22] Newcombe R A, Izadi S, Hilliges O, et al. KinectFusion: Real-time dense surface mapping and tracking [C]//IEEE International Symposium on Mixed and Augmented Reality. IEEE, 2012:127-136.
[23] Rusinkiewicz S, Levoy M. Efficient variants of the ICP algorithm [C]. International Conference on 3-D Digital Imaging and Modeling, 2001. Proceedings. IEEE, 2002:145-152.
[24] Muja M, Lowe D G. Fast approximate nearest neighbors with automatic algorithm configuration [C]//Proceedings of IEEE Conference on Computer Vision Theory and Applications. Lisbon, Portugal: IEEE Computer Society, 2009:331-340.
[25] Chum O, Philbin J, Zisserman A. Near duplicate image detection: min-hash and tf-idf weighting [C]//Proceedings of the 19th British Machine Vision Conference. London, UK, 2008: 493-502.
[26] Yangqing C, Timothy A D, William W H, et al. Algorithm 887:CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate [J]. ACM Transactions on Mathematical Software, 2008. 35(3):1-14.
[27] More J J. The Levenberg-Marquardt algorithm: Implementation and theory [M]. In: Numerical analysis, 1978:105-116.
[28] 沈大为.基于 RGB-D 图像序列的实时相机定位与三维场景建图方法[D].大连:大连理工大学, 2016.
[29] 夏文玲.基于 Kinect 与单目视觉 SLAM 的实时三维重建算法的实现[D].长沙:湖南大学, 2013.
[30] 丁洁琼.基于 RGB-D 的 SLAM 算法研究[D].西安:西安电子科技大学, 2014.
[31] 龙超.基于 Kinect 和视觉词典的三维 SLAM 算法研究[D].杭州:浙江大学, 2016.
[32] 贾松敏,王可,郭兵,李秀智.基于 RGB-D 相机的移动机器人三维 SLAM[J].华中科技大学学报(自然科学版),2014,42(01):103-109.
[33] 赵矿军.基于 RGB-D 摄像机的室内三维彩色点云地图构建[J].哈尔滨商业大学学报(自然科学版),2018,34(01):66-74.
[34] 王旒军,陈家斌,余欢,朱汇申.RGB-D SLAM 综述[J].导航定位与授时,2017,4(06):9-18.
[35] 于宁波,王石荣,徐昌.一种基于 RGB-D 的移动机器人未知室内环境自主探索与地图构建方法[J].机器人,2017,39(06):860-871.
[36] 许晓东,陈国良,李晓园,周文振,杜珊珊.一种采用图像特征匹配技术的 RGB-D SLAM 算法[J].测绘通报,2017(10):48-51.
[37] 熊军林,王婵.基于 RGB-D 图像的具有滤波处理和位姿优化的同时定位与建图[J].中国科学

技术大学学报,2017,47(08):665-673.
[38] 李弋星,刘士荣,仲朝亮,王坚.基于改进关键帧选择的 RGB-D SLAM 算法[J].大连理工大学学报,2017,57(04):411-417.
[39] 付梦印,吕宪伟,刘彤,杨毅,李星河,李玉.基于 RGB-D 数据的实时 SLAM 算法[J].机器人,2015,37(06):683-692.
[40] 梅峰,刘京,李淳秡,王兆其.基于 RGB-D 深度相机的室内场景重建[J].中国图象图形学报,2015,20(10):1366-1373.
[41] 于宁波,李元,徐昌,刘景泰.一种基于 RGB-D SLAM 的移动机器人定位和运动规划与控制方法[J].系统科学与数学,2015,35(07):838-847.
[42] 朱笑笑,曹其新,杨扬,陈培华.基于 RGB-D 传感器的 3D 室内环境地图实时创建[J].计算机工程与设计,2014,35(01):203-207.

致　　谢

时光荏苒，两年时间初看很长，但回首却又像一瞬间，自己已长大很多，学到了很多东西，但是也错过了很多。人不就是这样在得到与失去中成长吗！社会已经够匆忙，我们没有太多的时间去思考人生，但是身边总会有那些人，在我们成长的路上给我们一杯水、一个馒头、一杯水或者指引我们前进的方向。我们不会停止成长，不管是走向成熟或者是苍老，不管是走向光明或者黑暗，可身边总有那些人。我要感谢他们，感谢他们出现在自己的人生旅途，感谢他们陪着自己成长，感谢他们给我们指引方向。

首先，我要感谢我的导师周宽久教授，在这两年间，在学术方面他给予了学生细心的教导，教会我们如何研究问题。从入学到现在，周宽久教授在我课程学习、项目、论文的选题、研究点拓展、论文修改方面给予了悉心指导和建议。现在论文已经写完了，我要衷心地感谢周宽久教授，谢谢周宽久教授的付出。

然后，我要感谢已经毕业的潘杰师兄、石国辉师兄、梁浩然师兄等，他们在我学习和做项目的过程中给我无私的指导和建议，祝他们工作顺利；还要感谢姜献昭、刘鑫、王闯、张云帆、刘宝星、崔帅飞、赵昆雨同学，感谢他们陪我走过研究生的两年，马上就要毕业，祝他们未来一路顺风；同时还要感谢孙兆鹏、于泽伟、尹世冲、靳卓毅师弟，谢谢他们一路上对自己的支持，祝他们学业顺利。

最后，我要感谢我的家人。谢谢他们在这二十年来对我的默默支持和无私奉献，因为他们我才能走到现在。

感谢所有出现在我生命中的的美好的人和事。

大连理工大学学位论文版权使用授权书

本人完全了解学校有关学位论文知识产权的规定，在校攻读学位期间论文工作的知识产权属于大连理工大学，允许论文被查阅和借阅。学校有权保留论文并向国家有关部门或机构送交论文的复印件和电子版，可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印、或扫描等复制手段保存和汇编本学位论文。

学位论文题目：_____

作 者 签 名：_____ 日期：____年____月____日

导 师 签 名：_____ 日期：____年____月____日