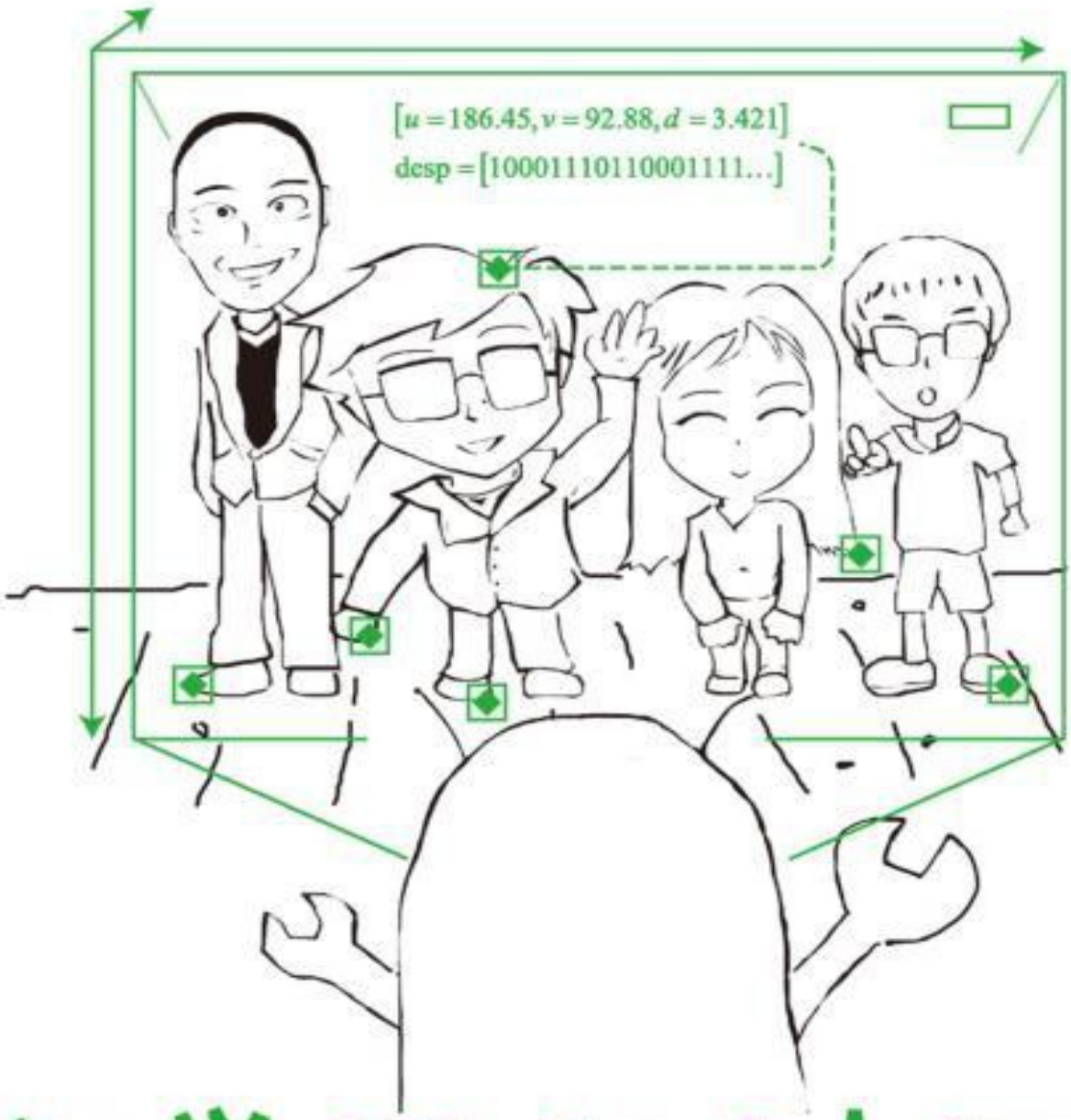
书籍汇总下载 http://t.cn/RDIAj5D

Broadview
www.broadview.com.cn



视觉SLAM十四讲

从理论到实践

高翔 张涛 等著

中国工信出版集团
電子工業出版社

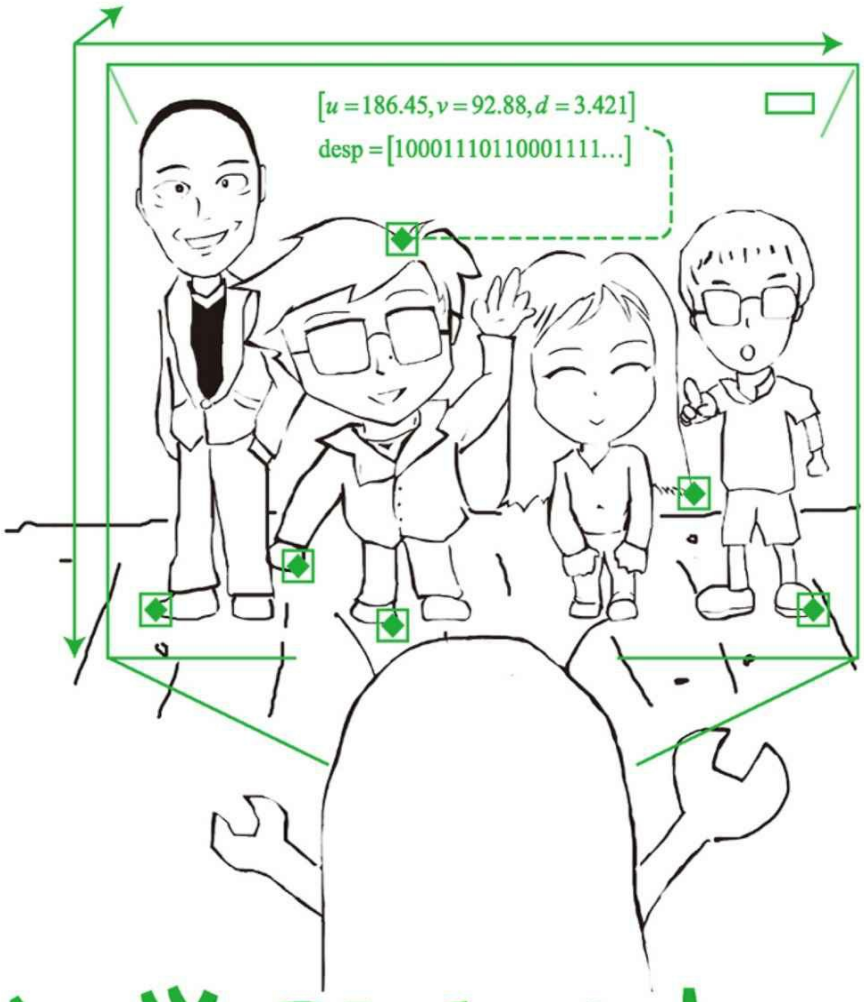
电子书寻找看手相qq37391319 钉钉或微信 pythontesting
书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

Broadview®
www.broadview.com.cn



视觉SLAM十四讲

从理论到实践

高翔 张涛 等著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

电子书寻找看手相qq37391319 钉钉或微信 pythontesting
书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

作者简介



·高翔·

2008年就读清华大学自动化系，2012年免试进入清华大学自动化系攻读博士学位。研究课题为视觉SLAM，兴趣包括计算机视觉与机器学习。曾撰写过与SLAM相关的论文和技术博客。



·张涛·

清华大学自动化系教授、党委书记、副系主任。1995年9月至1999年9月在清华大学自动化系检测技术与自动化装置专业学习，获博士学位。1999年10月至2002年9月在日本国立佐贺大学大学院工学系研究科系统控制专业学习，获博士学位。研究课题包括机器人、航空航天、计算机视觉等。



·刘毅·

华中科技大学图像与人工智能研究所在读博士，本科毕业于武汉理工大学数学系。读博期间专注于图像处理、三维重建、视觉SLAM，以及传感器融合研究和应用。读博期间先后于深圳市大疆创新公司、英特尔中国研究院等单位实习。



·颜沁睿·

比利时荷语鲁汶大学人工智能硕士，电子工程学士（GroupT）。电子科技大学信息显示与光电技术学士。长期致力于研究人工智能技术在机器人领域的应用，包括计算机视觉、机器学习和SLAM。现担任地平线机器人公司智能驾驶部算法工程师。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

视觉SLAM十四讲

从理论到实践

高翔 张涛 刘毅 颜沁睿 著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

电子书寻找看手相qq37391319 钉钉或微信 pythontesting
书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

内容简介

本书系统介绍了视觉SLAM（同时定位与地图构建）所需的基本知识与核心算法，既包括数学理论基础，如三维空间的刚体运动、非线性优化，又包括计算机视觉的算法实现，例如多视图几何、回环检测等。此外，我们还提供了大量的实例代码供读者学习研究，从而更深入地掌握这些内容。

本书可以作为对SLAM感兴趣的研究人员的入门自学材料，也可以作为SLAM相关的高校本科生或研究生课程教材使用。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有，侵权必究。

图书在版编目（**CIP**）数据

视觉SLAM十四讲：从理论到实践/高翔等著.—北京：电子工业出版社，2017.3

ISBN 978-7-121-31104-8

I.①视… II.①高… III.①人工智能-视觉跟踪-研究 IV.①TP18

中国版本图书馆CIP数据核字（2017）第053910号

策划编辑：郑柳洁

责任编辑：白 涛

印 刷：北京季蜂印刷厂

装 订：北京季蜂印刷厂

出版发行：电子工业出版社

北京市海淀区万寿路173信箱 邮编：100036

开 本：720×1000 1/16 印张：25 字数：560千字

版 次：2017年3月第1版

印 次：2017年3月第1次印刷

定 价：75.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

本书咨询联系方式：（010）51260888-819faq@phei.com.cn。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

阅读须知

本书图片

为描述方便，书中介绍图片相关内容时直接使用了对应的色彩名称，对应到纸面上即灰色。此外，部分图片可能需要放大观察，纸面上无法呈现应有效果。为此，书中图片均在博文视点官方网站提供下载。

读者服务

轻松注册成为博文视点社区用户（www.broadview.com.cn），您即可享受以下服务。

•下载资源：本书所提供的示例代码及资源文件均可在下载资源处下载。

•提交勘误：您对书中内容的修改意见可在提交勘误处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。

•与作者交流：在页面下方读者评论处留下您的疑问或观点，与作者和其他读者一同学习交流。

页面入口：http://www.broadview.com.cn/31104



电子书寻找看手相qq37391319 钉钉或微信 pythontesting

书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

目　录

内容简介

阅读须知

第1讲　预备知识

1.1　本书讲什么

1.2　如何使用本书

1.2.1　组织方式

1.2.2　代码

1.2.3　面向的读者

1.3　风格约定

1.4　致谢和声明

第2讲　初识SLAM

2.1　引子：小萝卜的例子

2.2　经典视觉SLAM框架

2.2.1　视觉里程计

2.2.2　后端优化

2.2.3　回环检测

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

2.2.4 建图

2.3 SLAM问题的数学表述

2.4 实践：编程基础

2.4.1 安装Linux操作系统

2.4.2 Hello SLAM

2.4.3 使用cmake

2.4.4 使用库

2.4.5 使用IDE

第3讲 三维空间刚体运动

3.1 旋转矩阵

3.1.1 点和向量，坐标系

3.1.2 坐标系间的欧氏变换

3.1.3 变换矩阵与齐次坐标

3.2 实践：Eigen

3.3 旋转向量和欧拉角

3.3.1 旋转向量

3.3.2 欧拉角

3.4 四元数

3.4.1 四元数的定义

3.4.2 四元数的运算

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

3.4.3 用四元数表示旋转

3.4.4 四元数到旋转矩阵的转换

3.5 *相似、仿射、射影变换

3.6 实践：Eigen几何模块

3.7 可视化演示

第4讲 李群与李代数

4.1 李群与李代数基础

4.1.1 群

4.1.2 李代数的引出

4.1.3 李代数的定义

4.1.4 李代数so(3)

4.1.5 李代数se(3)

4.2 指数与对数映射

4.2.1 SO(3)上的指数映射

4.2.2 SE(3)上的指数映射

4.3 李代数求导与扰动模型

4.3.1 BCH公式与近似形式

4.3.2 SO(3)李代数上的求导

4.3.3 李代数求导

4.3.4 扰动模型（左乘）

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

4.3.5 SE(3)上的李代数求导

4.4 实践：Sophus

4.5 *相似变换群与李代数

4.6 小结

第5讲 相机与图像

5.1 相机模型

5.1.1 针孔相机模型

5.1.2 畸变

5.1.3 双目相机模型

5.1.4 RGB-D相机模型

5.2 图像

5.3 实践：图像的存取与访问

5.3.1 安装OpenCV

5.3.2 操作OpenCV图像

5.4 实践：拼接点云

第6讲 非线性优化

6.1 状态估计问题

6.1.1 最大后验与最大似然

6.1.2 最小二乘的引出

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

6.2 非线性最小二乘

6.2.1 一阶和二阶梯度法

6.2.2 高斯牛顿法

6.2.3 列文伯格—马夸尔特方法

6.2.4 小结

6.3 实践：Ceres

6.3.1 Ceres简介

6.3.2 安装Ceres

6.3.3 使用Ceres拟合曲线

6.4 实践：g2o

6.4.1 图优化理论简介

6.4.2 g2o的编译与安装

6.4.3 使用g2o拟合曲线

6.5 小结

第7讲 视觉里程计1

7.1 特征点法

7.1.1 特征点

7.1.2 ORB特征

7.1.3 特征匹配

7.2 实践：特征提取和匹配

电子书寻找看手相qq37391319 钉钉或微信 pythontesting

书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

7.3 2D-2D：对极几何

7.3.1 对极约束

7.3.2 本质矩阵

7.3.3 单应矩阵

7.4 实践：对极约束求解相机运动

7.5 三角测量

7.6 实践：三角测量

7.6.1 三角测量代码

7.6.2 讨论

7.7 3D-2D：PnP

7.7.1 直接线性变换

7.7.2 P3P

7.7.3 Bundle Adjustment

7.8 实践：求解PnP

7.8.1 使用EPnP求解位姿

7.8.2 使用BA优化

7.9 3D-3D：ICP

7.9.1 SVD方法

7.9.2 非线性优化方法

7.10 实践：求解ICP

电子书寻找看手相qq37391319 钉钉或微信 pythontesting

书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

7.10.1 SVD方法

7.10.2 非线性优化方法

7.11 小结

第8讲 视觉里程计2

8.1 直接法的引出

8.2 光流（OpticalFlow）

8.3 实践：LK光流

8.3.1 使用TUM公开数据集

8.3.2 使用LK光流

8.4 直接法（DirectMethod）

8.4.1 直接法的推导

8.4.2 直接法的讨论

8.5 实践：RGB-D的直接法

8.5.1 稀疏直接法

8.5.2 定义直接法的边

8.5.3 使用直接法估计相机运动

8.5.4 半稠密直接法

8.5.5 直接法的讨论

8.5.6 直接法优缺点总结

电子书寻找看手相qq37391319 钉钉或微信 pythontesting

书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

第9讲　实践：设计前端

9.1　搭建VO框架

9.1.1　确定程序框架

9.1.2　确定基本数据结构

9.1.3　Camera类

9.1.4　Frame类

9.1.5　MapPoint类

9.1.6　Map类

9.1.7　Con fi g类

9.2　基本的VO：特征提取和匹配

9.2.1　两两帧的视觉里程计

9.2.2　讨论

9.3　改进：优化PnP的结果

9.4　改进：局部地图

9.5　小结

第10讲　后端1

10.1　概述

10.1.1　状态估计的概率解释

10.1.2　线性系统和KF

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

10.1.3 非线性系统和EKF

10.1.4 EKF的讨论

10.2 BA与图优化

10.2.1 投影模型和BA代价函数

10.2.2 BA的求解

10.2.3 稀疏性和边缘化

10.2.4 鲁棒核函数

10.2.5 小结

10.3 实践：g2o

10.3.1 BA数据集

10.3.2 g2o求解BA

10.3.3 求解

10.4 实践：Ceres

10.4.1 Ceres求解BA

10.4.2 求解

10.5 小结

第11讲 后端2

11.1 位姿图（PoseGraph）

11.1.1 Pose Graph的意义

11.1.2 Pose Graph的优化

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

11.2 实践：位姿图优化

11.2.1 g2o原生位姿图

11.2.2 李代数上的位姿图优化

11.2.3 小结

11.3 *因子图优化初步

11.3.1 贝叶斯网络

11.3.2 因子图

11.3.3 增量特性

11.4 *实践：gtsam

11.4.1 安装gtsam 4.0

11.4.2 位姿图优化

第12讲 回环检测

12.1 回环检测概述

12.1.1 回环检测的意义

12.1.2 方法

12.1.3 准确率和召回率

12.2 词袋模型

12.3 字典

12.3.1 字典的结构

12.3.2 实践：创建字典

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

12.4 相似度计算

12.4.1 理论部分

12.4.2 实践：相似度的计算

12.5 实验分析与评述

12.5.1 增加字典规模

12.5.2 相似性评分的处理

12.5.3 关键帧的处理

12.5.4 检测之后的验证

12.5.5 与机器学习的关系

第13讲 建图

13.1 概述

13.2 单目稠密重建

13.2.1 立体视觉

13.2.2 极线搜索与块匹配

13.2.3 高斯分布的深度滤波器

13.3 实践：单目稠密重建

13.4 实验分析与讨论

13.4.1 像素梯度的问题

13.4.2 逆深度

13.4.3 图像间的变换

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

13.4.4 并行化：效率的问题

13.4.5 其他的改进

13.5 RGB-D稠密建图

13.5.1 实践：点云地图

13.5.2 八叉树地图

13.5.3 实践：八叉树地图

13.6 *TSDF地图和Fusion系列

13.7 小结

第14讲 SLAM：现在与未来

14.1 当前的开源方案

14.1.1 MonoSLAM

14.1.2 PTAM

14.1.3 ORB-SLAM

14.1.4 LSD-SLAM

14.1.5 SVO

14.1.6 RTAB-MAP

14.1.7 其他

14.2 未来的SLAM话题

14.2.1 视觉+惯性导航SLAM

14.2.2 语义SLAM

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

14.2.3　SLAM的未来

附录A　高斯分布的

附录B　ROS入门

参考文献

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

第1讲　预备知识

1.1　本书讲什么

这是一本介绍视觉SLAM的书，也很可能是第一本以视觉SLAM为主题的中文书。

那么，SLAM是什么？

SLAM是Simultaneous Localization and Mapping的缩写，中文译作“同时定位与地图构建”[1]。它是指搭载特定传感器的主体，在没有环境先验信息的情况下，于运动过程中建立环境的模型，同时估计自己的运动[2]。如果这里的传感器主要为相机，那就称为“视觉SLAM”。

本书的主题就是视觉SLAM。这里我们刻意把许多个定义放到一句话中，是希望读者有一个较明确的概念。首先，SLAM的目的是解决“定位”与“地图构建”这两个问题。也就是说，一边要估计传感器自身的位置，一边要建立周围环境的模型。那么怎么解决呢？这需要用到传感器的信息。传感器能以一定形式观察外部的世界，不过不同传感器观察的方式是不同的。而之所以要花一本书的内容去讨论这个问题，是因为它很难——特别是我们希望实时地、在没有先验知识的情况下进行SLAM。当用相机作为传感器时，我们要做的就是根据一张张连续运动的图像（它们形成了一段视频），从中推断相机的运动，以及周围环境的情况。

这似乎是个很直观的问题。我们自己走进陌生的环境时不就是这么做的吗？

在计算机视觉（ComputerVision）创立之初，人们就想象着有朝一日计算机将和人一样，通过眼睛去观察世界，理解周遭的物体，探索未知的领域——这是一个美妙而又浪漫的梦想，吸引了无数的科研人员日夜为之奋斗[3]。我们曾经以为这件事情并不困难，然而进展却远不如预想的那么顺利。我们眼中的花草树木、虫鱼鸟兽，在计算机中却是那样

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

的不同：它们只是一个个由数字排列而成的矩阵（Matrix）。让计算机理解图像的内容，就像让我们自己理解这些数字一样困难。我们既不了解自己如何理解图像，也不知道计算机该如何理解、探索这个世界。于是我们困惑了很久，直到几十年后的今天，才发现了一点点成功的迹象：通过人工智能（Arti fi cial Intelligence）和机器学习（Machine Learning）技术，计算机渐渐能够辨别出物体、人脸、声音、文字——尽管它所用的方式（概率学建模）与我们是如此不同。另一方面，在SLAM发展了将近30年之后，我们的相机才渐渐开始能够认识到自身的位置，发觉自己在运动——虽然方式还是和我们人类有巨大的差异。不过，至少研究者们已经成功地搭建出种种实时SLAM系统，有的能够快速跟踪自身位置，有的甚至能够进行实时的三维重建。

这件事情确实很困难，但我们已经有了很大的进展。更令人兴奋的是，近年来随着科技的发展，涌现出了一大批与SLAM相关的应用点。在许多地方，我们都希望知道自身的位置：室内的扫地机和移动机器人需要定位，野外的自动驾驶汽车需要定位，空中的无人机需要定位，虚拟现实和增强现实的设备也需要定位。SLAM是那样重要。没有它，扫地机就无法在房间自主地移动，只能盲目地游荡；家用机器人就无法按照指令准确到达某个房间；虚拟现实也将永远固定在座椅之上——所有这些新奇的事物都无法出现在现实生活中，那将多么令人遗憾。

今天的研究者和应用开发人员，逐渐意识到了SLAM技术的重要性。在国际上，SLAM已经有近三十年的研究历史，也一直是机器人和计算机视觉的研究热点。21世纪以来，以视觉传感器为中心的视觉SLAM技术，在理论和实践上都经历了明显的转变与突破，正逐步从实验室研究迈向市场应用。同时，我们又遗憾地发现，至少在国内，与SLAM相关的论文、书籍仍然非常匮乏，让许多对SLAM技术感兴趣的初学者无从一窥门径。虽然SLAM的理论框架基本趋于稳定，但其编程实现仍然较为复杂，有着较高的技术门槛。刚步入SLAM领域的研究者，不得不花很长的时间，学习大量的知识，往往要走过许多弯路才得以接近SLAM技术的核心。

本书全面系统地介绍了以视觉传感器为主体的视觉SLAM技术，我们希望它能（部分地）填补这方面资料的空白。我们会详细地介绍SLAM的理论背景、系统架构，以及各个模块的主流做法。同时，极其重视实践：本书介绍的所有重要算法，都将给出可以运行的实际代码，

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

以求加深读者的理解。之所以这么做，主要是考虑到SLAM毕竟是一项和实践紧密相关的技术。再漂亮的数学理论，如果不能转化为可以运行的代码，那就仍是可望而不可即的空中楼阁，没有实际意义。我们相信，实践出真知，实践出真爱。只有实际地演算过各种算法之后，你才能真正地认识SLAM，真正地喜欢上科研。

自1986年提出以来[4]，SLAM一直是机器人领域的热点问题。关于它的文献数以千计，想要对SLAM发展史上的所有算法及变种做一个完整的说明，是十分困难而且没有必要的。本书中会介绍SLAM所牵涉的背景知识，例如射影几何、计算机视觉、状态估计理论、李群李代数等，并在这些背景知识之上，给出SLAM这棵大树的主干，而略去一部分形状奇特、纹理复杂的枝叶。我们认为这种做法是有效的。如果读者能够掌握主干的精髓，那么自然会有能力去探索那些边缘的、细节的、错综复杂的前沿知识。所以，我们的目的是，让SLAM的初学者通过阅读本书快速地成长为能够探索这个领域边缘的研究者。另一方面，即便你已经是SLAM领域的研究人员，本书也可能有一些你还觉得陌生的地方，可以让你产生新的见解。

目前，与SLAM相关的书籍主要有《概率机器人》（*Probabilistic robotics*）[5]、《计算机视觉中的多视图几何》（*Multiple View Geometry in Computer Vision*）[3]、《机器人学中的状态估计》（*State Estimation for Robotics:A Matrix-Lie-Group Approach*）[6]等。它们内容丰富、论述全面、推导严谨，是SLAM研究者中脍炙人口的经典教材。然而就目前来看，还存在两个重要的问题：其一，这些图书的目的在于介绍基础理论，SLAM只是其应用之一。因此，它们并不能算是专门讲解SLAM的书籍。其二，它们的内容偏重于数学理论，基本不涉及编程实现，导致读者经常出现“书能看懂却不会编程”的情况。而我们认为，只有读者亲自实现了算法，调试了各个参数，才能谈得上真正理解了问题本身。

我们会提及SLAM的历史、理论、算法、现状，并把完整的SLAM系统分成几个模块：视觉里程计、后端优化、建图，以及回环检测。我们将陪着读者一点点实现这些模块中的核心部分，探讨它们在什么情况下有效，什么情况下会出问题，并指导大家在自己的机器上运行这些代码。你会接触到一些必要的数学理论和许多编程知识，会用到Eigen、OpenCV、PCL、g2o、Ceres等库[1]，掌握它们在Linux操作系统中的使

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

用方法。

从写作风格上，我们不想把本书写成枯燥的理论书籍。技术类图书应该是严谨可靠的，但严谨不意味着刻板。一本优秀的技术书应该是生动有趣而易于理解的。如果你觉得“这个作者怎么这么不正经”，敬请原谅，因为我并不是一个非常严肃的人[2]。无论如何，有一件事是可以肯定的：只要你对这门新技术感兴趣，在学习本书的过程中肯定会有所收获！您会掌握与SLAM相关的理论知识，你的编程能力也将有明显的进步。在很多时候，您会有一种“我在陪你一起做科研”的感觉，这正是我所希望的。但愿您能在此过程中发现研究的乐趣，喜欢这种“通过一番努力，看到事情顺利运行”的成就感。

好了，话不多说，祝你旅行愉快！

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

1.2 如何使用本书

1.2.1 组织方式

本书名为“视觉SLAM十四讲”。顾名思义，我们会像在学校里讲课那样，以“讲”作为本书的基本单元。每一讲都对应一个固定的主题，其中会穿插“理论部分”和“实践部分”两种内容。通常是理论部分在前，实践部分在后。在理论部分中，我们将介绍理解算法所必需的数学知识，并且大多数时候以叙述的方式，而不是像数学教科书那样用“定义—定理—推论”的方式，因为我们觉得这样的方式阅读起来更容易一些，尽管有时候显得不那么严谨。实践部分主要是编程实现，讨论程序里各部分的含义及实验结果。看到标题中带有“实践”两个字的章节，你就应该（兴致勃勃地）打开电脑，和我们一起愉快地编写代码了。

值得一提的是，我们只会把与解决问题相关的数学知识放在书里，并尽量保持浅显。虽然我们是工科生，但也要承认，某些做法只要经验上够用，没必要非得在数学上追求完备。只要我们知道这些算法能够工作，并且数学家们告诉了我们在什么情况下可能不工作，那么我们就表示满意，而不追究那些看似完美但实际复杂冗长的证明（当然它们固有自己的价值）。由于SLAM牵涉到了太多数学背景，为了防止使本书变成数学教科书，我们把一些细节上的推导和证明留作习题和补充阅读材料，方便感兴趣的读者进一步阅读参考文献，更深入地掌握相关细节。

每一讲正文之后，我们设计了一些习题。其中，带*号的习题是具有一定难度的。我们强烈建议读者把习题都练习一遍，这对你掌握这些知识很有帮助[3]。

全书内容主要分为两个部分。

1.第一部分为数学基础篇，我们会以浅显易懂的方式，铺垫与视觉SLAM相关的

数学知识，包括：

•第1讲是前言，介绍这本书的基本信息，习题部分主要包括一些自

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

测题。

•第2讲为SLAM系统概述，介绍一个SLAM系统由哪些模块组成，各模块的具体工作是什么。实践部分介绍编程环境的搭建过程以及IDE的使用。

•第3讲介绍三维空间运动，你将接触到旋转矩阵、四元数、欧拉角的相关知识，并且在Eigen当中使用它们。

•第4讲为李群和李代数。即便你现在不懂李代数为何物，也没有关系。你将学到李代数的定义和使用方式，然后通过Sophus操作它们。

•第5讲介绍针孔相机模型以及图像在计算机中的表达。你将用OpenCV来调取相机的内外参数。

•第6讲介绍非线性优化，包括状态估计理论基础、最小二乘问题、梯度下降方法。你会完成一个使用Ceres和g2o进行曲线拟合的实验。

这些就是我们要用到的所有数学知识了，当然，其中还隐含了你以前学过的高等数学和线性代数。我们保证它们看起来都不会很难。当然，若你想进一步深入挖掘，我们会提供一些参考资料供你阅读，那些材料可能会比正文里讲的知识难一些。

2.第二部分为SLAM技术篇。我们会使用第一部分所介绍的理论，讲述视觉SLAM中各个模块的工作原理。

•第7讲为特征点法的视觉里程计。该讲内容比较多，包括特征点的提取与匹配、对极几何约束的计算、PnP和ICP等。在实践中，你将用这些方法去估计两个图像之间的运动。

•第8讲为直接法的视觉里程计。你将学习光流和直接法的原理，然后利用g2o实现一个简单的RGB-D直接法。

•第9讲为视觉里程计的实践章，你将搭建一个视觉里程计框架，综合运用先前学过的知识，实现它的基本功能。这个过程中，你会碰到一些问题，例如优化的必要性、关键帧的选择等。

•第10讲为后端优化，主要为对Bundle Adjustment的深入讨论，包括基本的BA，以及如何利用稀疏性加速求解过程。你将用Ceres和g2o分别书写一个BA程序。

书籍汇总下载 http://t.cn/RDIAj5D

•第11讲主要讲后端优化中的位姿图。位姿图是表达关键帧之间约束的一种更紧凑的形式。你将用g2o和gtsam对一个位姿球进行优化。

•第12讲为回环检测，主要介绍以词袋方法为主的回环检测。你将使用dbow3书写字典训练程序和回环检测程序。

•第13讲为地图构建。我们会讨论如何使用单目进行稠密深度图的估计（以及这是多么不可靠），然后讨论RGB-D的稠密地图构建过程。你会书写极线搜索与块匹配的程序，然后在RGB-D中遇到点云地图和八叉树地图的构建问题。

•第14讲主要介绍当前的开源SLAM项目以及未来的发展方向。相信在阅读了前面的知识之后，你会更容易理解它们的原理，实现自己的新想法。

最后，如果你完全看不懂上面在说什么，那么恭喜你！这本书很适合你！加油！

1.2.2 代码

本书所有源代码均托管在github上：

https://github.com/gaoxiang12/slambook

强烈建议读者下载它们以供随时查看。代码是按章节划分的，比如，第7讲的内容就会放在ch7文件夹中。此外，对于书中用到的一些小型库，会以压缩包的形式放在3rdparty文件夹下。对于像OpenCV那种大中型库，我们会在它们第一次出现时介绍其安装方法。如果你对代码有任何疑问，请单击GitHub上的Issues按钮，提交问题。如果确实是代码出现问题，我们会及时进行修改；即使是你的理解有偏差，我们也会尽可能回复。如果你不习惯使用Git，那么单击右侧包含download字样的按钮将代码下载至本地即可。

1.2.3 面向的读者

本书面向对SLAM感兴趣的学生和研究人员。阅读本书需要一定的基础，我们假设你具备以下知识：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

•高等数学、线性代数、概率论。这些是大部分读者应该在大学本科阶段接触过的基本数学知识。你应当明白矩阵和向量是什么，或者做微分和积分是什么意思。对于SLAM中用到的专业知识，我们会额外加以介绍。

•C++语言基础。由于我们采用C++作为编码语言，所以建议读者至少熟悉这门语言的语法。比如，你应该知道类是什么，如何使用C++标准库，模板类如何使用，等等。我们会避免过多地使用技巧，但有些地方确实无法避免。此外，我们还使用了一些C++11标准的内容，不过，我们会在用到的地方加以解释。

•Linux基础。我们的开发环境是Linux而非Windows，并且只提供Linux下的源程序，不会再提供Windows下的开发方法介绍。我们认为，掌握Linux是一个SLAM研究人员所必需的，请初学者暂时不要问为什么，把本书的知识学好之后相信你会和我们有同样的想法。各种程序库在Linux下的配置都非常便捷，你也会在此过程中体会到Linux的便利。如果读者此前从未使用过Linux，那么最好找一本Linux的教材稍加学习（掌握基本知识即可，一般就是相关图书的前面几章内容）。我们不要求读者具备高超的Linux操作技能，但希望读者至少知道“打开终端，进入代码目录”是如何操作的。本讲的习题里有一些Linux知识自测题，如果你清楚自测题的答案，那么阅读本书代码不会有任何问题。

对SLAM感兴趣但不具备上述知识的读者，可能在阅读本书时会感到困难。如果你不了解C++的基本知识，可以读一点*C++Primer Plus*之类的图书入门；如果你缺少相关的数学知识，也可以先阅读一些相关数学教材补充知识，不过我们认为，对大多数大学本科水平的朋友，读懂本书所需的数学背景肯定是具备了。代码方面，你最好花点时间亲自输入一遍，再调节里面的参数，看看效果会发生怎样的改变。这会对学习很有帮助。

本书可作为SLAM相关课程的教材，亦可作为课外自学材料使用。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

1.3 风格约定

本书既有数学理论介绍，也有编程实现，因此，为方便阅读，对不同内容采用了不同排版方式加以区分。

1.数学公式单独列出，重要的公式还在右侧标了序号，例如：

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}. \tag{1.1}$$

标量使用斜体字（如a），向量和矩阵使用粗斜体（如$\boldsymbol{a}$,$\boldsymbol{A}$）。空心粗体代表特殊集合，如实数集$\mathbb{R}$、整数集$\mathbb{Z}$。李代数部分使用哥特体，如$\mathfrak{se}(3)$。

2.程序代码以方框框出，使用小一些的字号，左侧带有行号。如果程序较长，方框会延续到下一页：

```
1 #include <iostream>
2 using namespace std;
3
4 int main ( int argc, char** argv )
5 {
6     cout<<"Hello"<<endl;
7     return 0;
8 }
```

3.当代码数量较多或有的部分与之前列出的重复，不适合完全列在书中时，我们会仅给出重要片段，并以“片段”二字注明。因此，我们强烈建议读者到GitHub上下载所有源代码，完成练习，以更好地掌握本书知识。

4.由于排版原因，书中展示的代码可能与GitHub中的代码有稍许不同，请以GitHub上的代码为准。

5.我们用到的每个库，在第一次出现的时候会有比较详细的说明，但在后续的使用中则不再赘述。所以，建议读者按章节顺序阅读本书内

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

容。

6.每一讲的开头会列出本讲的内容提要，而末尾会有小结和练习题。引用的参考文献在全书末尾列出。

7.以星号开头的章节是选读部分，读者可以根据兴趣阅读。跳过它们不会对理解后续章节产生影响。

8.文中重要的内容以黑体标出，相信你已经习惯了。

9.我们设计的实验大多数是演示性质的。看懂了它们不代表你已经熟悉整个库的使用。所以我们建议你在课外花一点时间，对本书经常用的几个库进行深入学习。

10.本书的习题和选读内容可能需要你自己搜索额外材料，所以你需要学会使用搜索引擎。

书籍汇总下载 http://t.cn/RDIAj5D

1.4 致谢和声明

在本书漫长的写作过程中，我们得到了许多人的帮助，包括但不限于：

•中科院的贺一家博士为第5讲的相机模型部分提供了材料。

•颜沁睿提供了第7讲的公式推导材料。

•华中科大的刘毅博士为本书第6讲和第10讲提供了材料。

•众多的老师、同学为本书提供了修改意见：肖锡臻、谢晓佳、耿欣、李帅杰、刘富强、袁梦、孙志明、陈昊升、王京、朱晏辰、丁文东、范帝楷、衡昱帆、高扬、李少朋、吴博、闫雪娇、张腾、郑帆、卢美奇、杨楠，等等。在此向他们表示感谢。

此外，感谢我的导师张涛教授一直以来对我的支持和帮助。感谢电子工业出版社郑柳洁编辑的支持。没有他们的帮助，本书不可能以现在的面貌来到读者面前。本书的成书与出版是所有人共同努力的结晶，尽管我没法把他们都列在作者列表中，但是它的出版离不开他们的工作。

本书写作过程中参考了大量文献和论文。其中大部分数学理论知识是前人研究的成果，并非我的原创。一小部分实验设计亦来自各开源代码的演示程序，不过大部分是我自己编写的。此外，也有一些图片摘自公开发表的期刊或会议论文，文中均已注明。未做说明的图像，或为原创，或来自网络，恕不一一列举。如有问题，请与我们联系，我们会在第一时间加以修正。

本书涉及知识点众多，错漏在所难免。如有疑问，欢迎通过电子邮件与我联系。

我的邮箱是：gaoxiang12@mails.tsinghua.edu.cn。

感谢我的爱人刘丽莲女士长期的理解和支持。这本书是献给她的。

习题（基本知识自测题）

1.有线性方程$Ax=b$，若已知A,b，需要求解x，该如何求解？这对A

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

和*b*有哪些要求？提示：从*A*的维度和秩角度来分析。

2.高斯分布是什么？它的一维形式是什么样子？它的高维形式是什么样子？

3.你知道C++中的类吗？你知道STL吗？你使用过它们吗？

4.你以前怎样书写C++程序？（你完全可以说只在Visual C++6.0下写过C++工程，只要你有写C++和C语言的经验就行。）

5.你知道C++11标准吗？其中哪些新特性你听说过或用过？有没有其他的标准？

6.你知道Linux吗？你有没有至少使用过一种（不算安卓），比如Ubuntu？

7.Linux的目录结构是什么样的？你知道哪些基本命令，比如ls,cat等？

8.如何在Ubuntu中安装软件（不打开软件中心的情况下）？这些软件被安装在什么地方？如果只知道模糊的软件名称（比如想要装一个名称中含有eigen的库），应该如何安装它？

9.*花一个小时学习一下Vim，因为你迟早会用它。你可以在终端中输入vimtutor阅读一遍所有内容。我们不需要你非常熟练地操作它，只要能够在学习本书的过程中使用它输入代码即可。不要在它的插件上浪费时间，不要想着把Vim用成IDE，我们只用它做文本编辑的工作。

---

[1]如果你完全没有听说过它们，那么应该感到兴奋，这说明你会从本书中收获很多知识。

[2]你会经常在脚注中发现一些神奇的东西。

[3]它们也可能成为今后相关行业的面试题，或许还能帮你在找工作时留个好印象。

书籍汇总下载 http://t.cn/RDIAj5D

第2讲　初识SLAM

主要目标

1.理解一个视觉SLAM框架由哪几个模块组成，各模块的任务是什么。

2.搭建编程环境，为开发和实验做准备。

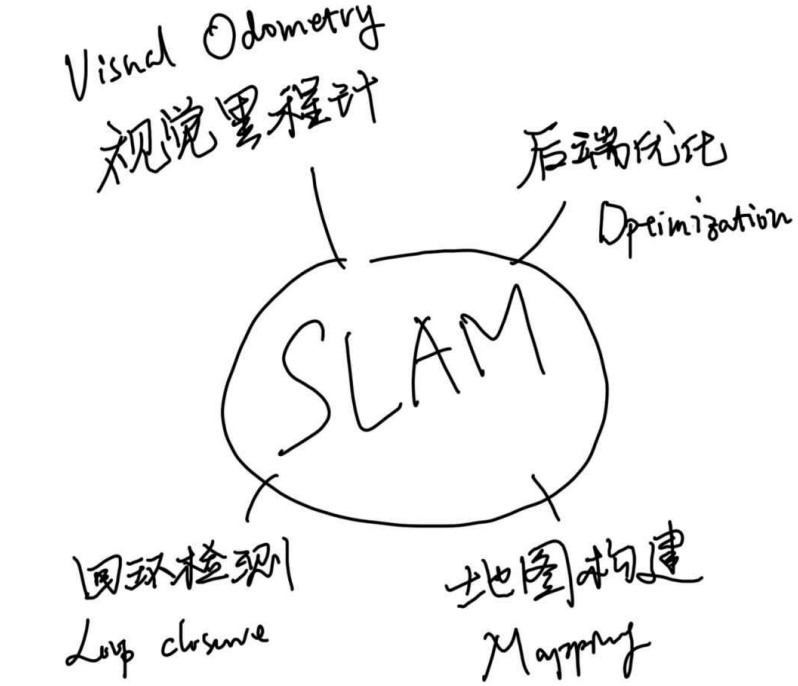
3.理解如何在Linux下编译并运行一个程序，如果程序出了问题，又该如何对它进行调试。

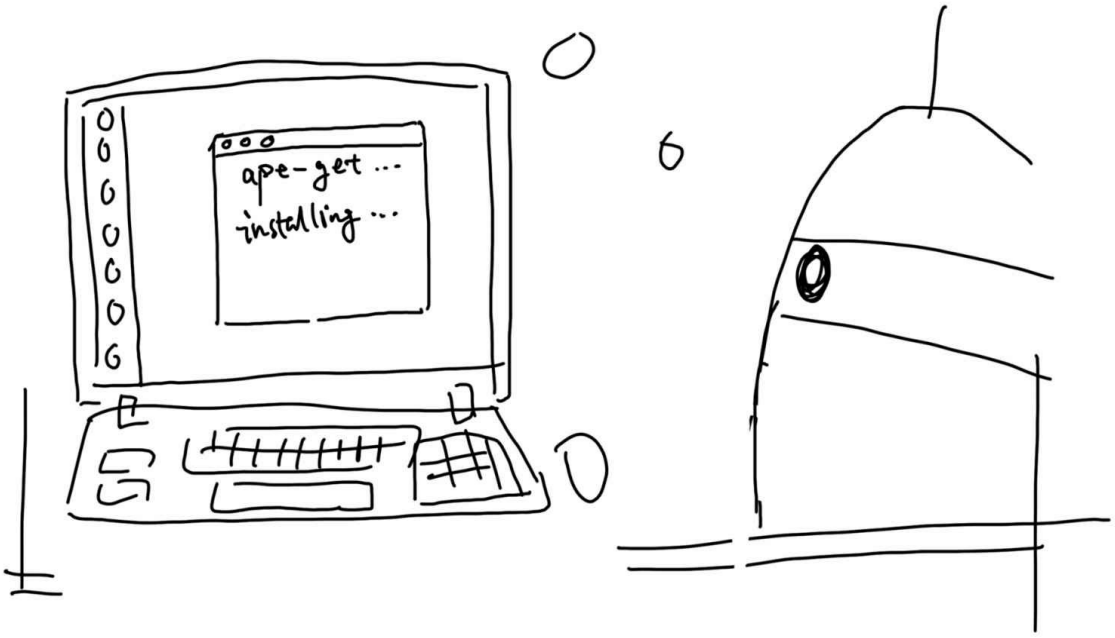
4.掌握cmake的基本使用方法。

本讲概括地介绍一个视觉SLAM系统的结构，作为后续内容的大纲。实践部分介绍环境搭建、程序基本知识，最后完成一个“Hello SLAM”程序。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D





电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

2.1 引子：小萝卜的例子

假设我们组装了一台叫作“小萝卜”的机器人，大概的样子如图2-1所示。

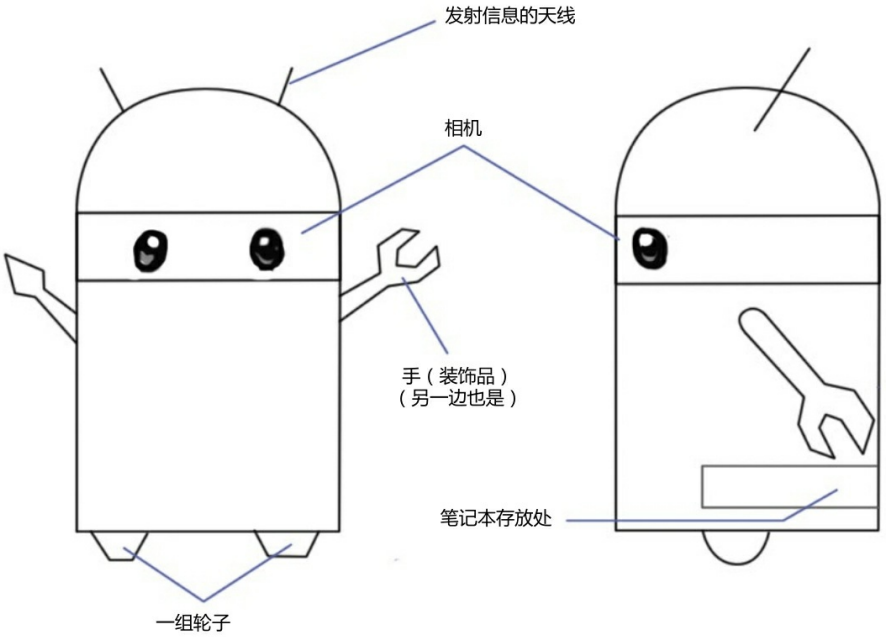


图2-1 小萝卜设计图。左边：正视图；右边：侧视图。设备有相机、轮子、笔记本，手是装饰品。

虽然有点像“安卓”，但它并不是靠安卓系统来计算的。我们把一台笔记本塞进了它的后备箱内（方便我们随时拿出来调试程序）。它能做点什么呢？

我们希望小萝卜具有自主运动能力。虽然世界上也有放在桌面像摆件一样的机器人，能够和人说话或播放音乐，不过一台平板电脑完全可以胜任这些事情。作为机器人，我们希望小萝卜能够在房间里自由移动。不管我们在哪里招呼一声，它都会滴溜溜地走过来。

要移动首先得有轮子和电机，所以我们在小萝卜的下方安装了轮子（足式机器人步态很复杂，我们暂时不考虑）。有了轮子，机器人就能够四处行动了，但不加控制的话，小萝卜不知道行动的目标，就只能四

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

处乱走，更糟糕的情况下会撞上墙造成损毁。为了避免这种情况的发生，我们在它的脑袋上安装了一个相机。安装相机的主要动机，是考虑到这样一个机器人和人类非常相似——从画面上一眼就能看出。有眼睛、大脑和四肢的人类，能够在任意环境里轻松自在地行走、探索，我们（天真地）觉得机器人也能够完成这件事。为了使小萝卜能够探索一个房间，它至少需要知道两件事：

1.我在什么地方？——定位。

2.周围环境是什么样？——建图。

“定位”和“建图”，可以看成感知的“内外之分”。作为一个“内外兼修”的小萝卜，一方面要明白自身的状态（即位置），另一方面也要了解外在的环境（即地图）。当然，解决这两个问题的方法非常多。比方说，我们可以在房间地板上铺设导引线，在墙壁上贴识别二维码，在桌子上放置无线电定位设备。如果在室外，还可以在小萝卜脑袋上安装定位设备（像手机或汽车一样）。有了这些东西之后，定位问题是否已经解决了呢？我们不妨把这些传感器（见图2-2）分为两类。

一类传感器是携带于机器人本体上的，例如机器人的轮式编码器、相机、激光传感器，等等。另一类是安装于环境中的，例如前面讲的导轨、二维码标志，等等。安装于环境中的传感设备，通常能够直接测量到机器人的位置信息，简单有效地解决定位问题。然而，由于它们必须在环境中设置，在一定程度上限制了机器人的使用范围。比方说，有些地方没有GPS信号，有些地方无法铺设导轨，这时该怎么做定位呢？

书籍汇总下载 http://t.cn/RDIAj5D

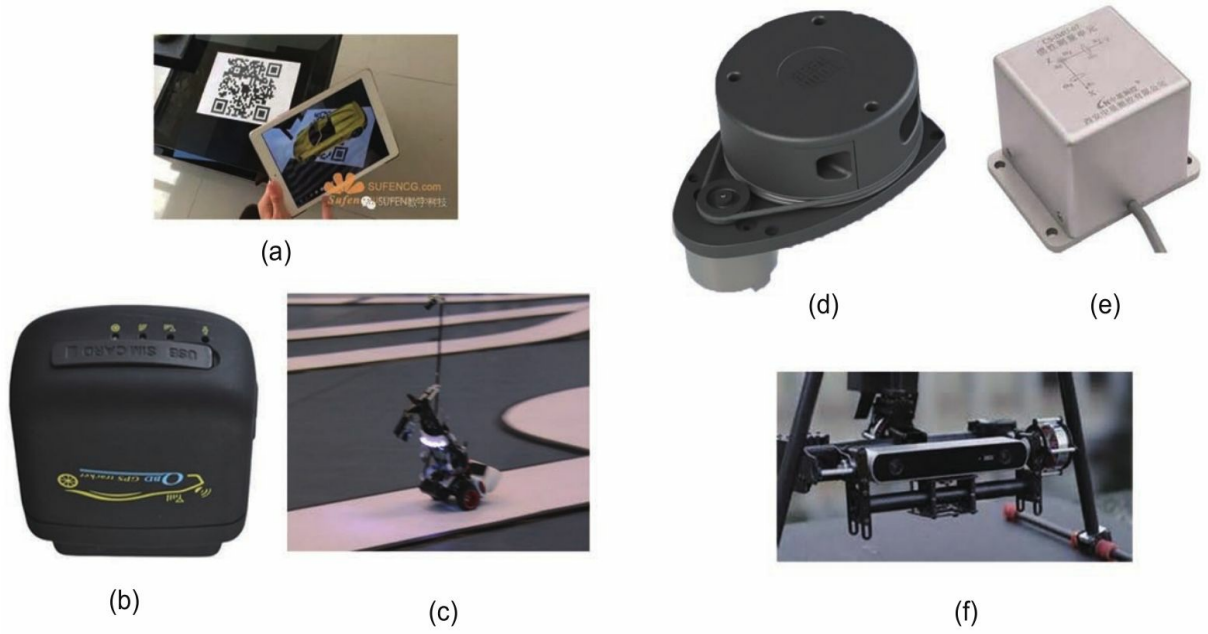


图2-2　一些传感器。（a）利用二维码进行定位的增强现实软件；（b）GPS定位装置；（c）铺设导轨的小车；（d）激光雷达；（e）IMU单元；（f）双目相机。

我们看到，这类传感器约束了外部环境。只有在这些约束满足时，基于它们的定位方案才能工作。反之，当约束无法满足时，我们就没法进行定位了。所以说，虽然这类传感器简单可靠，但它们无法提供一个普遍的、通用的解决方案。相对地，那些携带于机器人本体上的传感器，比如激光传感器、相机、轮式编码器、惯性测量单元（Inertial Measurement Unit，IMU）等，它们测到的通常都是一些间接的物理量而不是直接的位置数据。例如，轮式编码器会测到轮子转动的角度，IMU测量运动的角速度和加速度，相机和激光传感器则读取外部环境的某种观测数据。我们只能通过一些间接的手段，从这些数据推算自己的位置。虽然这听上去是一种迂回战术，但更明显的好处是，它没有对环境提出任何要求，从而使得这种定位方案可适用于未知环境。

回顾前面讨论过的SLAM定义，我们在SLAM中非常强调未知环境。在理论上，我们没法限制小萝卜的使用环境[1]，这意味着我们没法假设像GPS这些外部传感器都能顺利工作。因此，使用携带式的传感器来完成SLAM是我们重点关心的问题。特别地，当谈论视觉SLAM时，我们主要是指如何用相机解决定位和建图问题。

视觉SLAM是本书的主题，所以我们尤其关心小萝卜的眼睛能够做

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

些什么事。SLAM中使用的相机与我们平时见到的单反摄像头并不是同一个东西。它往往更加简单，不携带昂贵的镜头，而是以一定速率拍摄周围的环境，形成一个连续的视频流。普通的摄像头能以每秒钟30张图片的速度采集图像，高速相机则更快一些。按照工作方式的不同，相机可以分为单目相机（Monocular）、双目相机（Stereo）和深度相机（RGB-D）三大类，如图2-3所示。直观看来，单目相机只有一个摄像头，双目有两个，而RGB-D原理较复杂，除了能够采集到彩色图片之外，还能读出每个像素与相机之间的距离。深度相机通常携带多个摄像头，工作原理和普通相机不尽相同，在第5讲会详细介绍其工作原理，此处读者只需有一个直观概念即可。此外，SLAM中还有全景相机[7]、Event相机[8]等特殊或新兴的种类。虽然偶尔能看到它们在SLAM中的应用，不过到目前为止还没有成为主流。从样子上看，小萝卜使用的似乎是双目相机[2]。



图2-3　形形色色的相机：单目、双目和深度相机。

我们来分别看一看各种相机用来做SLAM时有什么特点。

单目相机

只使用一个摄像头进行SLAM的做法称为单目SLAM（Monocular SLAM）。这种传感器结构特别简单，成本特别低，所以单目SLAM非常受研究者关注。你肯定见过单目相机的数据：照片。是的，作为一张照片，它有什么特点呢？

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

照片本质上是拍照时的场景（Scene）在相机的成像平面上留下的一个投影。它以二维的形式反映了三维的世界。显然，这个过程丢掉了场景的一个维度，也就是所谓的深度（或距离）。在单目相机中，我们无法通过单张图片来计算场景中物体与我们之间的距离（远近）。之后我们会看到，这个距离将是SLAM中非常关键的信息。由于我们人类见过大量的图像，形成了一种天生的直觉，对大部分场景都有一个直观的距离感（空间感），它可以帮助我们判断图像中物体的远近关系。比如说，我们能够辨认出图像中的物体，并且知道其大致的大小；比如，近处的物体会挡住远处的物体，而太阳、月亮等天体一般在很远的地方；再如，物体受光照后会留下影子，等等。这些信息都可以帮助我们判断物体的远近，但也存在一些情况会使这种距离感失效，这时我们就无法判断物体的远近及其真实大小了。图2-4所示就是这样一个例子。在这张图像中，我们无法仅通过它来判断后面那些小人是真实的人，还是小型模型。除非我们转换视角，观察场景的三维结构。换言之，在单张图像里，你无法确定一个物体的真实大小。它可能是一个很大但很远的物体，也可能是一个很近但很小的物体。由于近大远小的原因，它们可能在图像中变成同样大小的样子。



图2-4　单目视觉中的尴尬：不知道深度时，手掌上的人是真人还是模型？

由于单目相机拍摄的图像只是三维空间的二维投影，所以，如果真想恢复三维结构，必须改变相机的视角。在单目SLAM中也是同样的原

理。我们必须移动相机，才能估计它的运动（Motion），同时估计场景中物体的远近和大小，不妨称之为结构（Structure）。那么，怎么估计这些运动和结构呢？从生活经验中我们知道，如果相机往右移动，那么图像里的东西就会往左边移动——这就给我们推测运动带来了信息。另一方面，我们还知道：近处的物体移动快，远处的物体则运动缓慢。于是，当相机移动时，这些物体在图像上的运动就形成了视差。通过视差，我们就能定量地判断哪些物体离得远，哪些物体离得近。

然而，即使我们知道了物体远近，它们仍然只是一个相对的值。比如我们在看电影时，虽然能够知道电影场景中哪些物体比另一些大，但无法确定电影里那些物体的“真实尺度”：那些大楼是真实的高楼大厦，还是放在桌上的模型？而摧毁大厦的是真实怪兽，还是穿着特摄服装的演员？直观地说，如果把相机的运动和场景大小同时放大两倍，单目相机所看到的像是一样的。同样地，把这个大小乘以任意倍数，我们都将看到一样的景象。这说明，单目SLAM估计的轨迹和地图将与真实的轨迹和地图相差一个因子，也就是所谓的尺度（Scale）[3]。由于单目SLAM无法仅凭图像确定这个真实尺度，所以又称为尺度不确定性。

平移之后才能计算深度，以及无法确定真实尺度，这两件事情给单目SLAM的应用造成了很大的麻烦。其根本原因是通过单张图像无法确定深度。所以，为了得到这个深度，人们开始使用双目和深度相机。

双目相机和深度相机

使用双目相机和深度相机的目的，在于通过某种手段测量物体与我们之间的距离，克服单目相机无法知道距离的缺点。一旦知道了距离，场景的三维结构就可以通过单个图像恢复出来，也就消除了尺度不确定性。尽管都是为了测量距离，但双目相机与深度相机测量深度的原理是不一样的。双目相机由两个单目相机组成，但这两个相机之间的距离〔称为基线（Baseline）〕是已知的。我们通过这个基线来估计每个像素的空间位置——这和人眼非常相似。我们人类可以通过左右眼图像的差异判断物体的远近，在计算机上也是同样的道理（见图2-5）。如果对双目相机进行拓展，也可以搭建多目相机，不过本质上并没有什么不同。

书籍汇总下载 http://t.cn/RDIAj5D

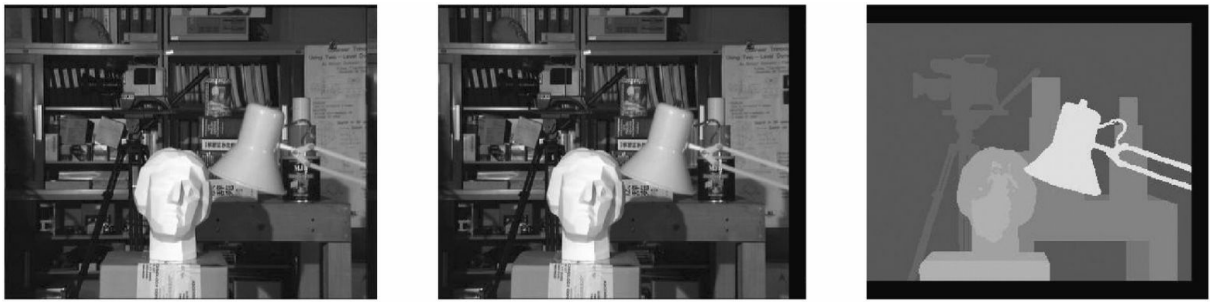


图2-5　双目相机的数据：左眼图像，右眼图像。通过左右眼的差异，能够判断场景中物体与相机之间的距离。

计算机上的双目相机需要大量的计算才能（不太可靠地）估计每一个像素点的深度，相比于人类真是非常笨拙。双目相机测量到的深度范围与基线相关。基线距离越大，能够测量到的就越远，所以无人车上搭载的双目通常会是个很大的家伙。双目相机的距离估计是比较左右眼的图像获得的，并不依赖其他传感设备，所以它既可以应用在室内，亦可应用于室外。双目或多目相机的缺点是配置与标定均较为复杂，其深度量程和精度受双目的基线与分辨率所限，而且视差的计算非常消耗计算资源，需要使用GPU和FPGA设备加速后，才能实时输出整张图像的距离信息。因此在现有的条件下，计算量是双目的主要问题之一。

深度相机（又称RGB-D相机，在本书中主要使用RGB-D这个名称）是2010年左右开始兴起的一种相机，它最大的特点是可以通过红外结构光或Time-of-Flight（ToF）原理，像激光传感器那样，通过主动向物体发射光并接收返回的光，测出物体与相机之间的距离。这部分并不像双目相机那样通过软件计算来解决，而是通过物理的测量手段，所以相比于双目相机可节省大量的计算（见图2-6）。目前常用的RGB-D相机包括Kinect/Kinect V2、Xtion Pro Live、RealSense等。不过，现在多数RGB-D相机还存在测量范围窄、噪声大、视野小、易受日光干扰、无法测量透射材质等诸多问题，在SLAM方面，主要用于室内，室外则较难应用。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D



图2-6　RGB-D数据：深度相机可以直接测量物体的图像和距离，从而恢复三维结构。

我们讨论了几种常见的相机，相信通过以上的说明，你已经对它们有了直观的了解。现在，想象相机在场景中运动的过程，我们将得到一系列连续变化的图像[4]。视觉SLAM的目标，是通过这样的一些图像，进行定位和地图构建。这件事情并没有我们想象的那么简单。它不是某种算法，只要我们输入数据，就可以往外不断地输出定位和地图信息了。SLAM需要一个完善的算法框架，而经过研究者们长期的努力工作，现有这个框架已经定型了。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

2.2 经典视觉SLAM框架

下面来看经典的视觉SLAM框架，如图2-7所示，了解一下视觉SLAM究竟由哪几个模块组成。

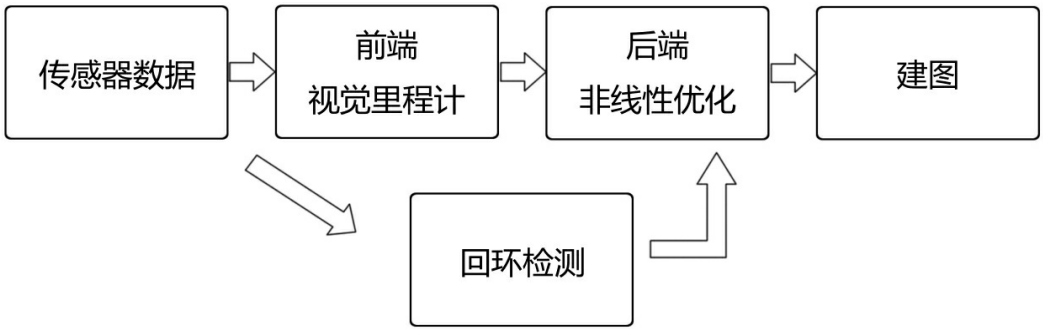


图2-7　整体视觉SLAM流程图。

整个视觉SLAM流程包括以下步骤。

1.传感器信息读取。在视觉SLAM中主要为相机图像信息的读取和预处理。如果是在机器人中，还可能有码盘、惯性传感器等信息的读取和同步。

2.视觉里程计（Visual　Odometry，VO）。视觉里程计的任务是估算相邻图像间相机的运动，以及局部地图的样子。VO又称为前端（Front End）。

3.后端优化（Optimization）。后端接受不同时刻视觉里程计测量的相机位姿，以及回环检测的信息，对它们进行优化，得到全局一致的轨迹和地图。由于接在VO之后，又称为后端（Back End）。

4.回环检测（Loop　Closing）。回环检测判断机器人是否到达过先前的位置。如果检测到回环，它会把信息提供给后端进行处理。

5.建图（Mapping）。它根据估计的轨迹，建立与任务要求对应的地图。

经典的视觉SLAM框架是过去十几年的研究成果。这个框架本身及其所包含的算法已经基本定型，并且已经在许多视觉程序库和机器人程

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

序库中提供。依靠这些算法，我们能够构建一个视觉SLAM系统，使之在正常的工作环境里实时定位与建图。因此，我们说，如果把工作环境限定在静态、刚体，光照变化不明显、没有人为干扰的场景，那么，这个SLAM系统是相当成熟的了[9]。

读者可能还没有理解上面几个模块的概念，下面就来详细介绍各个模块具体的任务。但是，准确理解其工作原理需要一些数学知识，我们将放到本书的第二部分进行。这里读者只需对各模块有一个直观的、定性的理解即可。

2.2.1 视觉里程计

视觉里程计关心的是相邻图像之间的相机运动，最简单的情况当然是两张图像之间的运动关系。例如，当看到图2-8时，我们会自然地反应出右图应该是左图向左旋转一定角度的结果（在视频情况下感觉会更加自然）。我们不妨思考一下：自己是怎么知道“向左旋转”这件事情的呢？人类早已习惯于用眼睛探索世界，估计自己的位置，但又往往难以用理性的语言描述我们的直觉。看到图2-8时，我们会自然地认为，这个场景中离我们近的是吧台，远处是墙壁和黑板。当相机向左转动时，吧台离我们近的部分出现在视野中，而右侧远处的柜子则移出了视野。通过这些信息，我们判断相机应该是向左旋转了。



图2-8　相机拍摄到的图片与人眼反应的运动方向。

但是，如果进一步问：能否确定旋转了多少度，平移了多少厘米？我们就很难给出一个确切的答案了。因为我们的直觉对这些具体的数字并不敏感。但是，在计算机中，又必须精确地测量这段运动信息。所以我们要问：计算机是如何通过图像确定相机的运动的呢？

书籍汇总下载 http://t.cn/RDIAj5D

前面也提过，在计算机视觉领域，人类在直觉上看来十分自然的事情，在计算机视觉中却非常困难。图像在计算机里只是一个数值矩阵。这个矩阵里表达着什么东西，计算机毫无概念（这也正是现在机器学习要解决的问题）。而在视觉SLAM中，我们只能看到一个个像素，知道它们是某些空间点在相机的成像平面上投影的结果。所以，为了定量地估计相机运动，必须先了解相机与空间点的几何关系。

要讲清这个几何关系以及VO的实现方法，需要铺垫一些背景知识。在这里我们先让读者对VO有个直观的概念。现在只需知道，VO能够通过相邻帧间的图像估计相机运动，并恢复场景的空间结构。称它为“里程计”是因为它和实际的里程计一样，只计算相邻时刻的运动，而和再往前的过去的信息没有关联。在这一点上，VO就像一种只有短时间记忆的物种。

现在，假定我们已有了一个视觉里程计，估计了两张图像间的相机运动。那么，只要把相邻时刻的运动“串”起来，就构成了机器人的运动轨迹，从而解决了定位问题。另一方面，我们根据每个时刻的相机位置，计算出各像素对应的空间点的位置，就得到了地图。这么说来，有了VO，是不是就解决了SLAM问题呢？

视觉里程计确实是SLAM的关键，我们也会花大量的篇幅来介绍它。然而，仅通过视觉里程计来估计轨迹，将不可避免地出现累积漂移（Accumulating Drift）。这是由于视觉里程计（在最简单的情况下）只估计两个图像间的运动造成的。我们知道，每次估计都带有一定的误差，而由于里程计的工作方式，先前时刻的误差将会传递到下一时刻，导致经过一段时间之后，估计的轨迹将不再准确（见图2-9）。比方说，机器人先向左转90°，再向右转90°。由于误差，我们把第一个90°估计成了89°。那我们就会尴尬地发现，向右转之后机器人的估计位置并没有回到原点。更糟糕的是，即使之后的估计再准确，与真实值相比，都会带上这-1°的误差。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

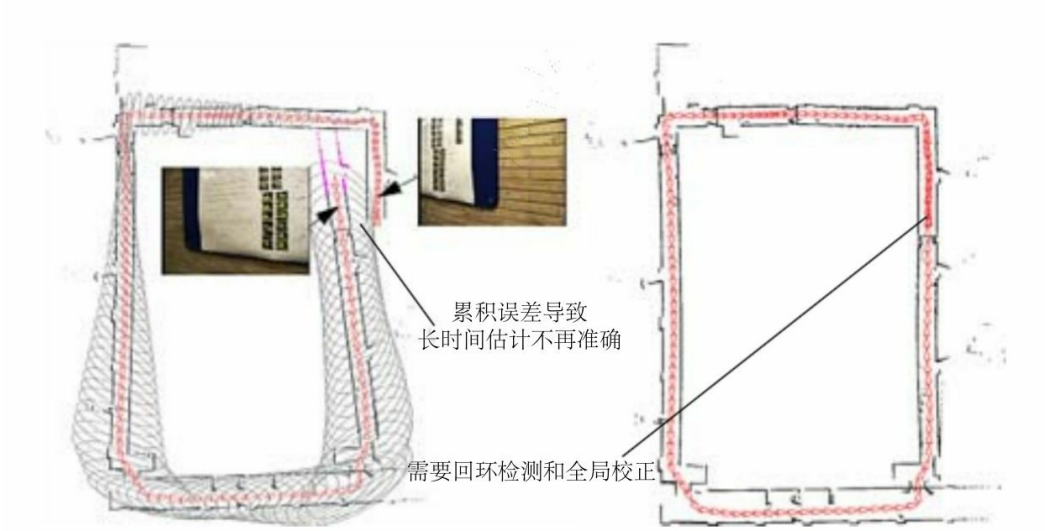


图2-9　累积误差与回环检测的校正结果[10]。

这也就是所谓的漂移（Drift）。它将导致我们无法建立一致的地图。你会发现原本直的走廊变成了斜的，而原本90°的直角变成了歪的——这实在是一件很难令人忍受的事情！为了解决漂移问题，我们还需要两种技术：后端优化[5]和回环检测。回环检测负责把“机器人回到原始位置”的事情检测出来，而后端优化则根据该信息，校正整个轨迹的形状。

2.2.2　后端优化

笼统地说，后端优化主要指处理SLAM过程中噪声的问题。虽然我们很希望所有的数据都是准确的，然而现实中，再精确的传感器也带有一定的噪声。便宜的传感器测量误差较大，昂贵的可能会小一些，有的传感器还会受磁场、温度的影响。所以，除了解决“如何从图像估计出相机运动”之外，我们还要关心这个估计带有多大的噪声，这些噪声是如何从上一时刻传递到下一时刻的，而我们又对当前的估计有多大的自信。后端优化要考虑的问题，就是如何从这些带有噪声的数据中估计整个系统的状态，以及这个状态估计的不确定性有多大——这称为最大后验概率估计（Maximum-a-Posteriori，MAP）。这里的状态既包括机器人自身的轨迹，也包含地图。

相对地，视觉里程计部分有时被称为“前端”。在SLAM框架中，前

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

端给后端提供待优化的数据，以及这些数据的初始值。而后端负责整体的优化过程，它往往面对的只有数据，不必关心这些数据到底来自什么传感器。在视觉SLAM中，前端和计算机视觉研究领域更为相关，比如图像的特征提取与匹配等，后端则主要是滤波与非线性优化算法。

从历史意义上来说，现在我们称为后端优化的部分，很长一段时间直接被称为“SLAM研究”。早期的SLAM问题是一个状态估计问题——正是后端优化要解决的东西。在最早提出SLAM的一系列论文中，当时的人们称它为“空间状态不确定性的估计”（Spatial Uncertainty）[4,11]。虽然有一些晦涩，但也确实反映出了SLAM问题的本质：对运动主体自身和周围环境空间不确定性的估计。为了解决SLAM问题，我们需要状态估计理论，把定位和建图的不确定性表达出来，然后采用滤波器或非线性优化，估计状态的均值和不确定性（方差）。状态估计与非线性优化的具体内容将在第6讲、第10讲和第11讲介绍。让我们暂时跳过它的原理说明，继续往下介绍。

2.2.3 回环检测

回环检测，又称闭环检测（Loop Closure Detection），主要解决位置估计随时间漂移的问题。怎么解决呢？假设实际情况下机器人经过一段时间的运动后回到了原点，但是由于漂移，它的位置估计值却没有回到原点。怎么办呢？我们想，如果有某种手段，让机器人知道“回到了原点”这件事，或者把“原点”识别出来，我们再把位置估计值“拉”过去，就可以消除漂移了。这就是所谓的回环检测。

回环检测与“定位”和“建图”二者都有密切的关系。事实上，我们认为，地图存在的主要意义是让机器人知晓自己到过的地方。为了实现回环检测，我们需要让机器人具有识别到过的场景的能力。它的实现手段有很多。例如像前面说的那样，我们可以在机器人下方设置一个标志物（如一张二维码图片）。它只要看到了这个标志，就知道自己回到了原点。但是，该标志物实质上是一种环境中的传感器，对应用环境做了限制（万一不能贴二维码怎么办？）。我们更希望机器人能使用携带的传感器——也就是图像本身，来完成这一任务。例如，可以判断图像间的相似性来完成回环检测。这一点和人是相似的。当我们看到两张相似的图片时，容易辨认它们来自同一个地方。如果回环检测成功，可以显著

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

地减小累积误差。所以，视觉回环检测实质上是一种计算图像数据相似性的算法。由于图像的信息非常丰富，使得正确检测回环的难度降低了不少。

在检测到回环之后，我们会把“A与B是同一个点”这样的信息告诉后端优化算法。然后，后端根据这些新的信息，把轨迹和地图调整到符合回环检测结果的样子。这样，如果我们有充分而且正确的回环检测，就可以消除累积误差，得到全局一致的轨迹和地图。

2.2.4 建图

建图（Mapping）是指构建地图的过程。地图（见图2-10）是对环境的描述，但这个描述并不是固定的，需要视SLAM的应用而定。

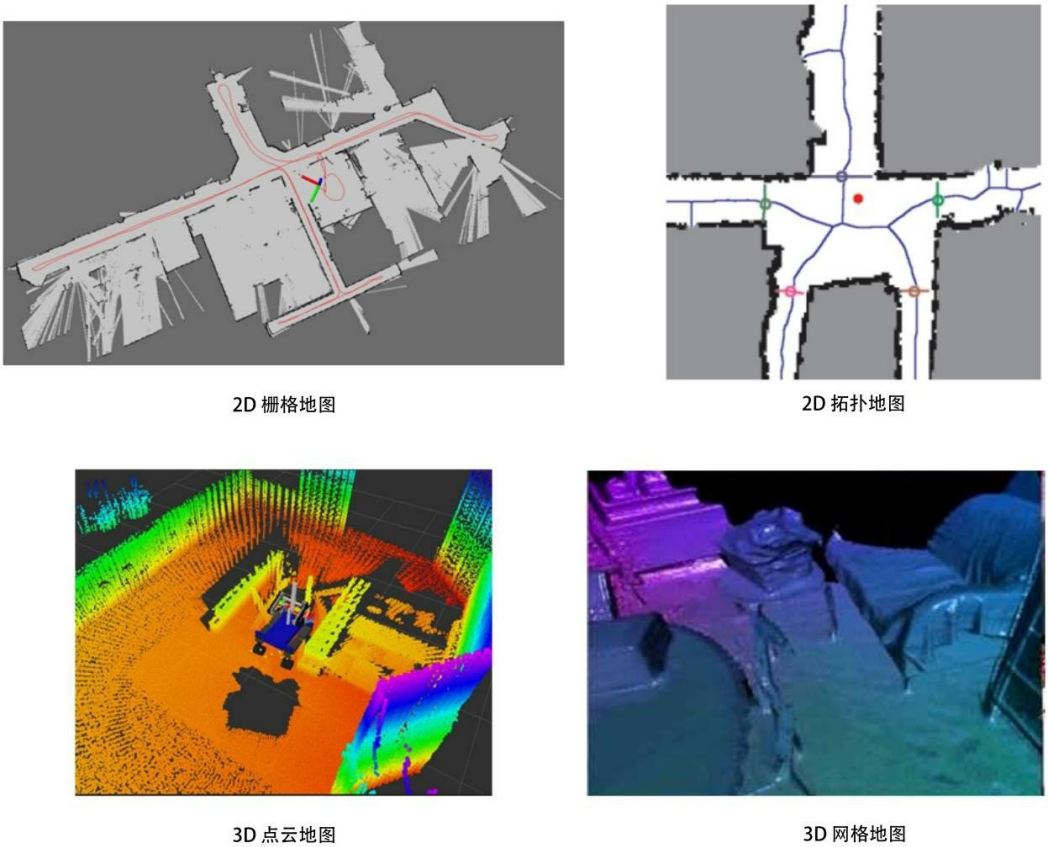


图2-10　形形色色的地图[12]。

对于家用扫地机器人来说，这种主要在低矮平面里运动的机器人，只需要一个二维的地图，标记哪里可以通过，哪里存在障碍物，就够它

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

在一定范围内导航了。而对于一个相机，它有6自由度的运动，我们至少需要一张三维的地图。有些时候，我们想要一个漂亮的重建结果，不仅是一组空间点，还需要带纹理的三角面片。另一些时候，我们又不关心地图的样子，只需要知道“A点到B点可通过，而B点到C点不行”这样的事情。甚至，有时不需要地图，或者地图可以由其他人提供，例如，行驶的车辆往往可以得到已绘制好的当地地图。

对于地图，我们有太多的想法和需求。因此，相比于前面提到的视觉里程计、回环检测和后端优化，建图并没有一个固定的形式和算法。一组空间点的集合也可以称为地图，一个漂亮的3D模型亦是地图，一个标记着城市、村庄、铁路、河道的图片还是地图。地图的形式随SLAM的应用场合而定。大体上讲，可以分为度量地图与拓扑地图两种。

度量地图（Metric Map）

度量地图强调精确地表示地图中物体的位置关系，通常用稀疏（Sparse）与稠密（Dense）对其分类。稀疏地图进行了一定程度的抽象，并不需要表达所有的物体。例如，我们选择一部分具有代表意义的东西，称之为路标（Landmark），那么一张稀疏地图就是由路标组成的地图，而不是路标的部分就可以忽略掉。相对地，稠密地图着重于建模所有看到的东西。对于定位来说，稀疏路标地图就足够了。而用于导航时，则往往需要稠密的地图（否则撞上两个路标之间的墙怎么办？）。稠密地图通常按照某种分辨率，由许多个小块组成。对于二维度量地图是许多个小格子（Grid），而对于三维度量地图则是许多小方块（Voxel）。一般地，一个小块含有占据、空闲、未知三种状态，以表达该格内是否有物体。当查询某个空间位置时，地图能够给出该位置是否可以通过的信息。这样的地图可以用于各种导航算法，如A*、D*[6]等，为机器人研究者所重视。但是我们也看到，这种地图需要存储每一个格点的状态，会耗费大量的存储空间，而且多数情况下地图的许多细节部分是无用的。另一方面，大规模度量地图有时会出现一致性问题。很小的一点转向误差，可能会导致两间屋子的墙出现重叠，使地图失效。

拓扑地图（Topological Map）

相比于度量地图的精确性，拓扑地图则更强调地图元素之间的关

书籍汇总下载 http://t.cn/RDIAj5D

系。拓扑地图是一个图（Graph），由节点和边组成，只考虑节点间的连通性，例如A、B点是连通的，而不考虑如何从A点到达B点。它放松了地图对精确位置的需要，去掉了地图的细节问题，是一种更为紧凑的表达方式。然而，拓扑地图不擅长表达具有复杂结构的地图。如何对地图进行分割形成结点与边，又如何使用拓扑地图进行导航与路径规划，仍是有待研究的问题。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

2.3 SLAM问题的数学表述

通过前面的介绍，读者应该对SLAM中各个模块的组成和主要功能有了直观的了解。但仅仅靠直观印象并不能帮助我们写出可以运行的程序。我们要把它上升到理性层次，也就是用数学语言来描述SLAM过程。我们会用到一些变量和公式，但请读者放心，我们会尽量让它保持足够地清楚。

假设小萝卜正携带着某种传感器在未知环境里运动，怎么用数学语言描述这件事呢？首先，由于相机通常是在某些时刻采集数据的，所以我们也只关心这些时刻的位置和地图。这就把一段连续时间的运动变成了离散时刻t=1,⋯,K当中发生的事情。在这些时刻，用x表示小萝卜自身的位置。于是各时刻的位置就记为x_1,⋯,x_K，它们构成了小萝卜的轨迹。地图方面，我们假设地图是由许多个路标（Landmark）组成的，而每个时刻，传感器会测量到一部分路标点，得到它们的观测数据。不妨设路标点一共有N个，用y_1,⋯,y_N表示它们。在这样的设定中，“小萝卜携带着传感器在环境中运动”，由如下两件事情描述：

1.什么是运动？我们要考虑从k-1时刻到k时刻，小萝卜的位置x是如何变化的。

2.什么是观测？假设小萝卜在k时刻于x_k处探测到了某一个路标y_j，我们要考虑这件事情是如何用数学语言来描述的。

先来看运动。通常，机器人会携带一个测量自身运动的传感器，比如说码盘或惯性传感器。这个传感器可以测量有关运动的读数，但不一定直接就是位置之差，还可能是加速度、角速度等信息。然而，无论是什么传感器，我们都能使用一个通用的、抽象的数学模型：

$$\boldsymbol{x}_k = f\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k, \boldsymbol{w}_k\right). \tag{2.1}$$

这里，u_k是运动传感器的读数（有时也叫输入），w_k为噪声。注意到，我们用一个一般函数f来描述这个过程，而不具体指明f的作用方式。这使得整个函数可以指代任意的运动传感器，成为一个通用的方

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

程，而不必限定于某个特殊的传感器上。我们把它称为运动方程。

与运动方程相对应，还有一个观测方程。观测方程描述的是，当小萝卜在x_k位置上看到某个路标点y_j，产生了一个观测数据$z_{k,j}$。同样，用一个抽象的函数h来描述这个关系：

$$\boldsymbol{z}_{k,j} = h\left(\boldsymbol{y}_j, \boldsymbol{x}_k, \boldsymbol{v}_{k,j}\right). \tag{2.2}$$

这里，$v_{k,j}$是这次观测里的噪声。由于观测所用的传感器形式更多，这里的观测数据z以及观测方程h也有许多不同的形式。

读者或许会说，我们用的函数f,h，似乎并没有具体地说明运动和观测是怎么回事？同时，这里的x,y,z又是什么东西呢？事实上，根据小萝卜的真实运动和传感器的种类，存在着若干种参数化方式（Parameterization）。什么叫参数化呢？举例来说，假设小萝卜在平面中运动，那么，它的位姿[7]由两个位置和一个转角来描述，即$x_k = [x, y, \theta]_k^{\mathrm{T}}$。同时，运动传感器能够测量到小萝卜在任意两个时间间隔位置和转角的变化量$\boldsymbol{u}_k = [\Delta x, \Delta y, \Delta\theta]_k^{\mathrm{T}}$，于是，此时运动方程就可以具体化为这是简单的线性关系。不过，并不是所有的传感器都能直接测量出位移和角度变化，所以也存在着其他形式更加复杂的运动方程，那时我们可能需要进行动力学分析。关于观测方程，比方说小萝卜携带着一个二维激光传感器。我们知道激光传感器观测一个2D路标点时，能够测到两个量：路标点与小萝卜本体之间的距离r和夹角ϕ。记路标点为$y = [p_x, p_y]^{\mathrm{T}}$（为保持简洁，省略了下标），观测数据为$z = [r, \phi]^{\mathrm{T}}$，那么观测方程就具体化为：

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_k = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{k-1} + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix}_k + \boldsymbol{w}_k. \tag{2.3}$$

$$\begin{bmatrix} r \\ \phi \end{bmatrix} = \begin{bmatrix} \sqrt{(p_x - x)^2 + (p_y - y)^2} \\ \arctan\left(\frac{p_y - y}{p_x - x}\right) \end{bmatrix} + \boldsymbol{v}. \tag{2.4}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

考虑视觉SLAM时，传感器是相机，那么观测方程就是“对路标点拍摄后，得到图像中的像素”的过程。这个过程牵涉到相机模型的描述，将在第5讲中详细介绍，这里暂且略过。

可见，针对不同的传感器，这两个方程有不同的参数化形式。如果我们保持通用性，把它们取成通用的抽象形式，那么SLAM过程可总结为两个基本方程：

$$\begin{cases} \boldsymbol{x}_k = f\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k, \boldsymbol{w}_k\right) \\ \boldsymbol{z}_{k,j} = h\left(\boldsymbol{y}_j, \boldsymbol{x}_k, \boldsymbol{v}_{k,j}\right) \end{cases}. \tag{2.5}$$

这两个方程描述了最基本的SLAM问题：当知道运动测量的读数u，以及传感器的读数z时，如何求解定位问题（估计x）和建图问题（估计y）？这时，我们就把SLAM问题建模成了一个状态估计问题：如何通过带有噪声的测量数据，估计内部的、隐藏着的状态变量？

状态估计问题的求解，与两个方程的具体形式，以及噪声服从哪种分布有关。按照运动和观测方程是否为线性，噪声是否服从高斯分布进行分类，分为线性/非线性和高斯/非高斯系统。其中线性高斯系统（Linear Gaussian，LG系统）是最简单的，它的无偏的最优估计可以由卡尔曼滤波器（Kalman Filter，KF）给出。而在复杂的非线性非高斯系统（Non-Linear Non-Gaussian，NLNG系统）中，我们会使用以扩展卡尔曼滤波器（Extended Kalman Filter，EKF）和非线性优化两大类方法去求解。直至21世纪早期，以EKF为主的滤波器方法在SLAM中占据了主导地位。我们会在工作点处把系统线性化，并以预测—更新两大步骤进行求解（见第10讲）。最早的实时视觉SLAM系统即是基于EKF[2]开发的。随后，为了克服EKF的缺点（例如线性化误差和噪声高斯分布假设），人们开始使用粒子滤波器（Particle Filter）等其他滤波器，乃至使用非线性优化的方法。时至今日，主流视觉SLAM使用以图优化（Graph Optimization）为代表的优化技术进行状态估计[13]。我们认为优化技术已经明显优于滤波器技术，只要计算资源允许，通常都偏向于使用优化方法（见第10讲和第11讲）。

相信读者已经对SLAM的数学模型有了大致的了解，然而我们仍需

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

澄清一些问题。首先，要说明机器人位置x是什么。我们方才说位置是有些模糊的。也许读者能够理解，在平面中运动的小萝卜可以用两个坐标加一个转角的形式将位置参数化。然而，虽然我的漫画风格有些二次元，小萝卜在更多时候是一个三维空间里的机器人。我们知道三维空间的运动由3个轴构成，所以小萝卜的运动要由3个轴上的平移，以及绕着3个轴的旋转来描述，一共有6个自由度。那是否意味着随便用一个$\mathbb{R}^6$中的向量就能描述它了呢？我们将发现事情并没有那么简单。对6自由度的位姿[8]，如何表达它，如何优化它，都需要一定篇幅来介绍，这将是第3讲和第4讲的主要内容。随后，我们要说明在视觉SLAM中，观测方程如何参数化。换句话说，空间中的路标点是如何投影到一张照片上的。这需要解释相机的成像模型，我们将在第5讲介绍。最后，当知道了这些信息，怎么求解上述方程？这需要非线性优化的知识，是第6讲的内容。

这些内容组成了本书数学知识的部分。在对它们进行铺垫之后，我们就能仔细讨论视觉里程计、后端优化等更详细的知识了。可以看到，本讲介绍的内容构成了本书的一个提要。如果读者还没有很好地理解上面的概念，不妨回过头再阅读一遍。下面就要开始介绍程序啦！

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

2.4 实践：编程基础

2.4.1 安装Linux操作系统

终于开始令人兴奋的实践环节啦！你是否准备好了呢？为了完成本书的实践环节，我们需要准备一台电脑。你可以使用笔记本或台式机，当然最好是你个人的电脑，因为我们需要在上面安装操作系统进行实验。

我们的程序将以Linux上的C++程序为主。在实验过程中，我们会使用到大量程序库。大部分程序库只对Linux提供了较好的支持，而在Windows上的配置则相对（相当）麻烦。因此，我们不得不假定你已经具备关于Linux的基本知识了（参见上一讲的练习题），包括使用基本的命令，了解软件如何安装。这样我们才无须讲解这些内容。当然，你不必了解如何在Linux下开发C++程序，这正是下面会详细讲解的。

我们先来搭建本书所需的实验环境。作为一本面向初学者的书，我们使用Ubuntu作为开发环境。在Linux的各大发行版中，Ubuntu及其衍生版本一直享有对用户友好的美誉。Ubuntu是一个开源操作系统，它的系统和软件可以在官方网站（http://cn.ubuntu.com）免费下载，并且提供了详细的安装方式说明。同时，清华、中科大等国内各大高校也提供了Ubuntu软件源，使软件的安装十分便捷。对于初学者，建议你使用和我们一样的环境：Ubuntu 14.04。如果你想试试其他口味，那么Ubuntu 16.04、Ubuntu Kylin、Debian 7/8和Linux Mint 17/18也是不错的选择。我们保证书中所有代码在Ubuntu 14.04下经过了良好的测试，但如果你选择其他发行版，则无法确定你是否会遇到问题。你可能需要花费一些时间解决问题（不过你也可以把它们当作锻炼自己的机会）。大体来说，Ubuntu对各种库的支持均较为完善，软件也非常丰富。尽管我们不限制你具体使用哪种Linux发行版，不过在讲解中，我们会以Ubuntu 14.04为例，且主要使用Ubuntu下的命令（例如apt-get），就不谈在其他Linux下怎么操作了。一般情况下，程序在Linux间移植不会非常烦琐。但如果你想在Windows或OS X下使用本书中的程序，则需要有一定的移植经验。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

现在，请大家在自己的PC上安装好Ubuntu 14.04。关于Ubuntu的安装，可以在网上搜到大量教程，只要照做即可，此处略过。最简单的方式是使用虚拟机（见图2-11），但需要占用大量内存（我们的经验是4GB以上）和CPU才能保持流畅；你也可以安装双系统，这样会快一些，但需要一个空白的U盘来作为启动盘。另外，虚拟机软件对外部硬件的支持往往不够好，如果希望使用实际的传感器（双目、Kinect等），则建议你使用双系统来安装Linux。

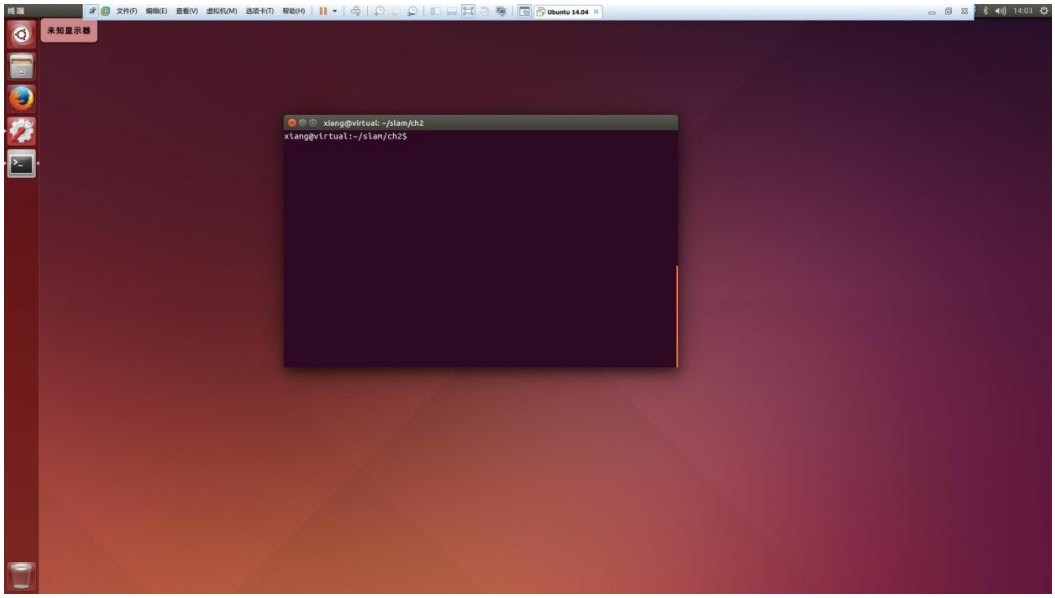


图2-11 一个运行在虚拟机中的Ubuntu 14.04。

关于安装的小提示：

•安装操作系统时请不要选择“安装中下载更新”，并且断开网络连接，这样可以提高安装速度。至于更新可以在系统安装完毕后再装。如果你有SSD硬盘，这个过程大概用时15分钟。

•安装完成后，请务必把软件源设置到离你较近的服务器上，以获得更快的下载速度。例如我们使用清华的软件源通常能以10MB/s的速度安装软件[9]。

现在，假设你已经成功安装好Ubuntu，无论是使用虚拟机还是双系统的方式。如果你还不熟悉Ubuntu，可以试试它的各种软件，体验一下它的界面和交互方式[10]。不过我必须提醒你，特别是新手朋友：不要在Ubuntu的用户界面上花费太多时间！Linux有许多可能浪费时间的地

书籍汇总下载 http://t.cn/RDIAj5D

方，你可能会找到某些小众的软件、一些游戏，甚至会为找一张壁纸花费不少时间。但是请记住，你是用Linux来工作的。特别是在本书中，你是用Linux来学习SLAM的，所以要尽量把时间花在学习SLAM上。

好了，我们选择一个目录，放置本书中SLAM程序的代码。例如，可以将代码放到家目录（/home）的“slambook”下。以后我们将把这个目录称为“代码根目录”。同时，可以另外选择一个目录，把本书的Git代码复制下来，方便做实验时随时对照。本书的代码是按章节划分的。比如，本讲的代码将在slambook/ch2下，下一讲则在slambook/ch3下。所以，现在请读者进入slambook/ch2下（你应该会新建文件夹并进入该文件夹了吧）。

2.4.2 Hello SLAM

我们从最基本的程序开始。与许多计算机类书籍一样，我们来书写一个HelloSLAM程序。不过在做这件事之前，我们先来聊聊程序是什么。

在Linux中，程序是一个具有执行权限的文件。它可以是一个脚本，也可以是一个二进制文件，不过我们不限定它的后缀名（不像Windows那样需要指定成.exe文件）。我们常用的cd、ls等命令，就是位于/bin目录下的可执行文件。而对于其他地方的可执行程序，只要它有可执行权限，那么当我们在终端中输入程序名时，它就会运行。在C++编程时，我们像下面这样用编译器把一个文本文件编译成可执行程序。

slambook/ch2/helloSLAM.cpp

```
#include <iostream>
using namespace std;

int main( int argc, char** argv )
{
    cout<<"Hello SLAM!"<<endl;
    return 0;
}
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

这是一个非常简单的程序。你应该能毫不费力地看懂它，所以这里不多加解释——如果实际情况不是这样，请你先补习一下C++的基本知识。这个程序只是把一个字符串输出到屏幕上而已。你可以用文本编辑器gedit（或Vim，如果你在上一讲学习了Vim）输入这些代码，并保存在上面列出的路径下。现在，我们用编译器g++（g++是一个C++编译器）把它编译成一个可执行文件。输入：

```
g++ helloSLAM.cpp
```

如果顺利，这条命令应该没有任何输出。如果机器上出现“command not found”的错误信息，说明你可能还没有安装g++，请用如下命令进行安装：

```
sudo apt-get install g++
```

如果出现别的错误，请再检查一遍刚才的程序是否输入正确。

刚才这条编译命令把helloSLAM.cpp这个文本文件编译成了一个可执行程序。我们检查当前目录，会发现多了一个a.out文件，而且它具有执行权限（终端里颜色不同）。我们输入./a.out即可运行此程序：

```
% ./a.out
Hello SLAM!
```

如我们所想，这个程序输出“Hello SLAM!”，告诉我们它在正确运行。

请回顾一下我们之前做的事情。在这个例子中，我们用编辑器输入了helloSLAM.cpp的源代码，然后调用g++编译器对它进行编译，得到了可执行文件。g++默认把源文件编译成a.out这个名字的程序（虽然有些古怪，但是可以接受的）。如果我们愿意，也可以指定这个输出的文件名（留作习题）。这是一个极其简单的例子，我们使用了大量的默认参数，几乎省略了所有中间步骤，为的是给读者一个简洁的印象（虽然你可能没有体会到）。下面我们要用cmake来编译这个程序。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

2.4.3 使用cmake

理论上说，任意一个C++程序都可以用g++来编译。但当程序规模越来越大时，一个工程可能有许多个文件夹和源文件，这时输入的编译命令将越来越长。通常一个小型C++项目可能含有十几个类，各类间还存在着复杂的依赖关系。其中一部分要编译成可执行文件，另一部分编译成库文件。如果仅靠g++命令，我们需要输入大量的编译指令，整个编译过程会变得异常烦琐。因此，对于C++项目，使用一些工程管理工具会更加高效。在历史上工程师们曾使用make fi le进行自动编译，但下面要谈的cmake比它更加方便。并且，我们会看到后面提到的大多数库都使用cmake来管理源代码。

在一个cmake工程中，我们会用cmake命令生成一个make fi le文件，然后，用make命令根据这个make fi le文件的内容编译整个工程。读者可能还不知道make fi le是什么东西，不过没关系，我们会通过例子来学习。仍然以上面的helloSLAM.cpp为例，这次我们不是直接使用g++，而是用cmake来制作一个工程，然后再编译它。在slambook/ch2/中新建一个CMakeLists.txt文件，内容如下：

slambook/ch2/CMakeLists.txt

```
# 声明要求的 cmake 最低版本
cmake_minimum_required( VERSION 2.8 )

# 声明一个 cmake 工程
project( HelloSLAM )

# 添加一个可执行程序
# 语法：add_executable( 程序名  源代码文件 )
add_executable( helloSLAM helloSLAM.cpp )
```

CMakeLists.txt文件用于告诉cmake我们要对这个目录下的文件做什么事情。CMake-Lists.txt文件的内容需要遵守cmake的语法。这个示例中，我们演示了最基本的工程：指定一个工程名和一个可执行程序。根据注释，读者应该理解每句话做了些什么。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

现在，在当前目录下（slambook/ch2/），调用cmake对该工程进行分析：

```
cmake .
```

cmake会输出一些编译信息，然后在当前目录下生成一些中间文件，其中最重要的就是MakeFile[11]。由于MakeFile是自动生成的，我们不必修改它。现在，用make命令对工程进行编译：

```
% make
Scanning dependencies of target helloSLAM
[100%] Building CXX object CMakeFiles/helloSLAM.dir/helloSLAM.cpp.o
Linking CXX executable helloSLAM
[100%] Built target helloSLAM
```

编译过程中会输出一个编译进度。如果顺利通过，我们就可以得到在CMakeLists.txt中声明的那个可执行程序helloSLAM。执行它：

```
% ./helloSLAM
Hello SLAM!
```

因为我们并没有修改源代码，所以得到的结果和之前是一样的。请读者想想这种做法和之前直接使用g++编译的区别。这次我们用cmake–make的做法，cmake过程处理了工程文件之间的关系，而make过程实际调用了g++来编译程序。虽然这个过程中多了调用cmake和make的步骤，但我们对项目的编译管理工作，从输入一串g++命令，变成了维护若干个比较直观的CMakeLists.txt文件，这将明显降低维护整个工程的难度。比如，如果想新增一个可执行文件，只需在CMakeLists.txt中添加一行“add_executable”命令即可，而后续的步骤是不变的。cmake会帮我们解决代码的依赖关系，而无须输入一大串g++命令。

现在这个过程中唯一让我们不满的是，cmake生成的中间文件还留在我们的代码文件当中。当想要发布代码时，我们并不希望把这些中间文件一同发布出去。这时我们还需要把它们一个个删除，十分不便。一

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

种更好的做法是让这些中间文件都放在一个中间目录中，在编译成功后，把这个中间目录删除即可。所以，更常见的编译cmake工程的做法如下：

```
mkdir build
cd build
cmake ..
make
```

我们新建了一个中间文件夹“build”，然后进入build文件夹，通过cmake..命令对上一层文件夹，也就是代码所在的文件夹进行编译。这样，cmake产生的中间文件就会生成在build文件夹中，与源代码分开。当发布源代码时，只要把build文件夹删掉即可。请读者自行按照这种方式对ch2中的代码进行编译，然后调用生成的可执行程序（请记得把上一步产生的中间文件删掉）。

2.4.4 使用库

在一个C++工程中，并不是所有代码都会编译成可执行文件。只有带有main函数的文件才会生成可执行程序。而另一些代码，我们只想把它们打包成一个东西，供其他程序调用。这个东西叫作库。

一个库往往是许多算法、程序的集合，我们会在之后的练习中接触到许多库。例如，OpenCV库提供了许多计算机视觉相关的算法，而Eigen库提供了矩阵代数的计算。因此，我们要学习如何用cmake生成库，并且使用库中的函数。现在书写如下libHelloSLAM.cpp文件。

slambook/ch2/libHelloSLAM.cpp

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
//这是一个库文件
#include <iostream>
using namespace std;
void printHello()
{
    cout<<"Hello SLAM"<<endl;
}
```

这个库提供了一个printHello函数，调用此函数将输出一条信息。但是它没有main函数，这意味着这个库中没有可执行文件。我们在CMakeLists.txt里加上如下内容：

```
add_library( hello libHelloSLAM.cpp )
```

这条命令告诉cmake，我们想把这个文件编译成一个叫作“hello”的库。然后，和上面一样，使用cmake编译整个工程：

```
cd build
cmake ..
make
```

这时，在build文件夹中就会生成一个libhello.a文件，这就是我们得到的库。

在Linux中，库文件分成静态库和共享库两种[12]。静态库以.a作为后缀名，共享库以.so结尾。所有库都是一些函数打包后的集合，差别在于静态库每次被调用都会生成一个副本，而共享库则只有一个副本，更省空间。如果想生成共享库而不是静态库，只需使用以下语句即可。

```
add_library( hello_shared SHARED libHelloSLAM.cpp )
```

此时得到的文件就是libhello_shared.so了。

库文件是一个压缩包，里面有编译好的二进制函数。不过，如果仅

书籍汇总下载 http://t.cn/RDIAj5D

有.a或.so库文件，那么我们并不知道里面的函数到底是什么，调用的形式又是什么样。为了让别人（或者自己）使用这个库，我们需要提供一个头文件，说明这些库里都有些什么。因此，对于库的使用者，只要拿到了头文件和库文件，就可以调用这个库了。下面编写libhello的头文件。

slambook/ch2/libHelloSLAM.h

```
#ifndef LIBHELLOSLAM_H_
#define LIBHELLOSLAM_H_
void printHello();
#endif
```

这样，根据这个文件和我们刚才编译得到的库文件，就可以使用printHello函数了。下面写一个可执行程序来调用这个简单的函数：

slambook/ch2/useHello.cpp

```
#include "libHelloSLAM.h"
int main( int argc, char** argv )
{
    printHello();
    return 0;
}
```

然后，在CMakeLists.txt中添加一个可执行程序的生成命令，链接到刚才使用的库上：

```
add_executable( useHello useHello.cpp )
target_link_libraries( useHello hello_shared )
```

通过这两行语句，useHello程序就能顺利使用hello_shared库中的代码了。这个小例子演示了如何生成并调用一个库。请注意，对于他人提供的库，我们也可用同样的方式对它们进行调用，整合到自己的程序中。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

除了已经演示的功能之外，cmake还有许多语法和选项，这里不一一列举。习题中包含了一些cmake的阅读材料，感兴趣的读者可自行阅读。现在，简单回顾一下我们之前做了哪些事：

1.首先，程序代码由头文件和源文件组成。

2.带有main函数的源文件编译成可执行程序，其他的编译成库文件。

3.如果可执行程序想调用库文件中的函数，它需要参考该库提供的头文件，以明白调用的格式。同时，要把可执行程序链接到库文件上。

这几个步骤应该是简单清楚的，但实际操作中你可能会遇上一些问题。比如说，如果代码里引用了库的函数，但忘了把程序链接到库上，会发生什么呢？请试试把CMake-Lists.txt中的链接部分去掉，看看会发生什么情况。你能看懂cmake报告的错误消息吗？

2.4.5 使用IDE

最后，我们来谈谈如何使用集成开发环境（Integrated Development Environment，IDE）。前面的编程完全可以用一个简单的文本编辑器来完成。然而，你可能需要在各个文件间跳来跳去，查询某个函数的声明和实现。当文件很多时，这仍然很烦琐。IDE为开发者提供了跳转、补全、断点调试等很多方便的功能，所以，我们建议读者选择一个IDE进行开发。

Linux下的IDE有很多种。虽然与Windows下的Visual Studio还有一些差距，不过支持C++开发的也有好几种，例如：Eclipse、QtCreator、Code::Blocks、Clion，等等。同样，我们不强制读者使用某种特定的IDE，而仅给出我们的建议。我们使用的是Kdevelop（见图2-12）。它是一个免费软件，在Ubuntu的源中提供了，这意味着你可以用apt-get来安装它。Kdevelop的优点列举如下：

1.支持cmake工程。

2.对C++支持较好（包括11标准）。有高亮、跳转、补全等功能。能自动排版代码。

3.能方便地看到各个文件和目录树。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

4.有一键编译、断点调试等功能。

5.无须付费。

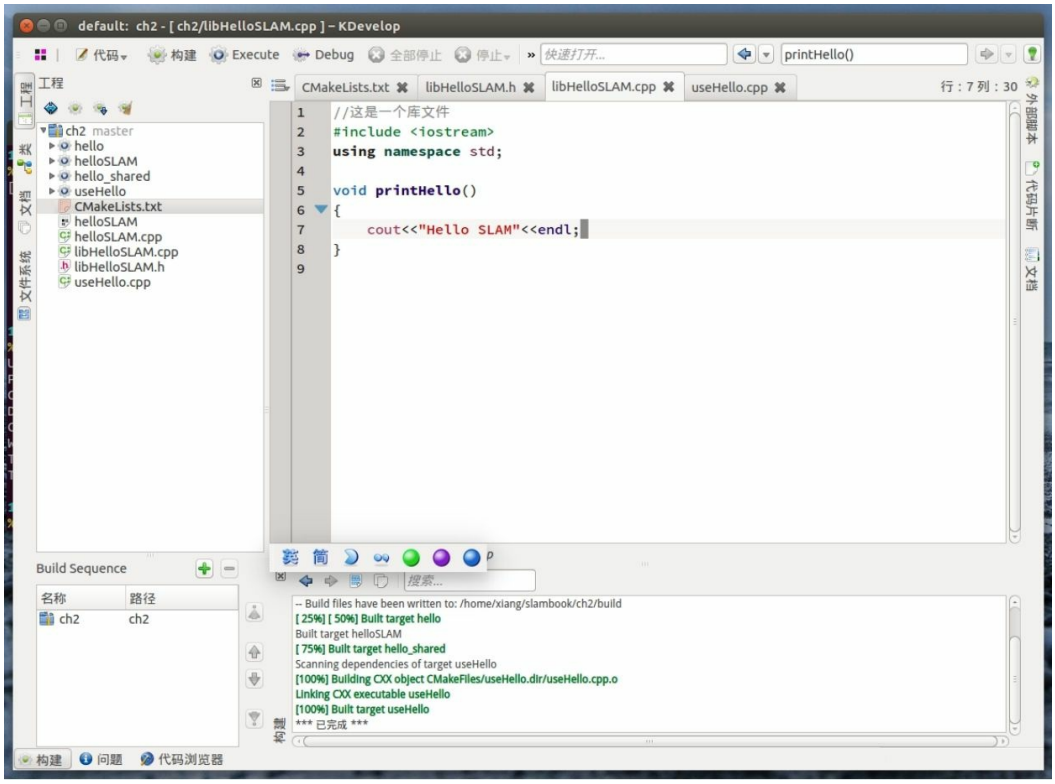


图2-12　Kdevelop界面。

基本上，我们对一个IDE的功能要求它都具备，所以读者不妨尝试一下。有时候你会碰到一些问题，例如对某些模板类的解析响应比较缓慢等，确实它还不够完善。不过相比于其他IDE，它是不错的。

Kdevelop原生支持cmake工程。具体做法是，在终端建立CMakeLists.txt后，用Kdevelop中的“工程→打开/导入工程”打开CMakeLists.txt。软件会询问你几个问题，并且默认建立一个build文件夹，帮你调用刚才的cmake和make命令。只要按下快捷键F8，这些都可以自动完成。图2-12的下面部分就显示了编译信息。

我们把适应IDE的任务交给读者自己来完成，而并不打算在书中进行详细说明。如果你是从Windows转过来的，会觉得它的界面与Visual C++或Visual　Studio挺相似。请用Kdevelop打开刚才的工程然后进行编译，看看它输出什么信息。相信你会觉得比打开终端更方便一些。

书籍汇总下载 http://t.cn/RDIAj5D

不过，本节重点想讲的是如何在IDE中进行调试。在Windows下编程的同学多半会有在Visual Studio下断点调试的经历。不过在Linux中，默认的调试工具gdb只提供了文本界面，对新手来讲不太方便。有些IDE提供了断点调试功能（底层仍旧是gdb），Kdevelop就是其中之一。要使用Kdevelop的断点调试功能，你需要完成以下几件事：

1.在CMakeLists.txt中把工程调为Debug编译模式。

2.告诉Kdevelop你想运行哪个程序。如果有参数，也要配置它的参数和工作目录。

3.进入断点调试界面，就可以单步运行，看到中间变量的值了。

第一步，在CMakeLists.txt中加入下面的命令来设置编译模式：

```
1 set( CMAKE_BUILD_TYPE "Debug" )
```

cmake自带一些编译相关的内部变量，它们可以对编译过程进行更精细的控制。对于编译类型，通常有调试用的Debug模式与发布用的Release模式。在Debug模式中，程序运行较慢，但可以进行断点调试；而Release模式则速度较快，但没有调试信息。我们把程序设置成Debug模式，就能放置断点了。接下来，告诉Kdevelop你想启动哪个程序。

第二步，打开“运行→配置启动器”，然后单击左侧的“Add New→应用程序”。在这一步中，我们的任务是告诉Kdevelop想要启动哪一个程序。如图2-13所示，既可以直接选择一个cmake的工程目标（也就是我们用add_executable指令构建的可执行程序），也可以直接指向一个二进制文件。建议使用第二种方式，根据我们的经验，这样更少出现问题。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

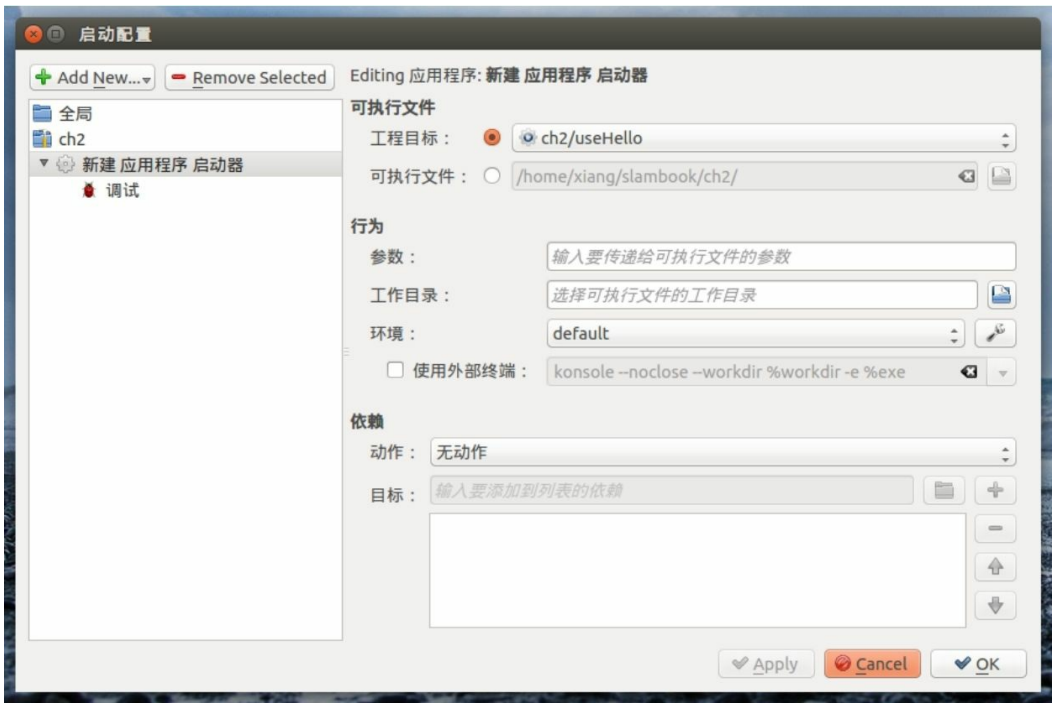


图2-13　启动器设置界面。

在第二栏里，可以设置程序的运行参数和工作目录。有时程序是有运行参数的，它们会作为main函数的参数被传入。如果没有则可以留空，对于工作目录亦是如此。配置好这两项后，可以单击“OK”按钮保存配置结果。

刚才这几步我们配置了一个应用程序的启动项。对于每一个启动项，我们可以单击“Execute”按钮直接启动这个程序，也可单击“Debug”按钮对它进行断点调试。读者可以试着单击“Execute”按钮，查看输出的结果。现在，为了调试这个程序，单击printHello那行的左侧，增加一个断点。然后，单击“Debug”按钮，程序会停留在断点处等待，如图2-14所示。

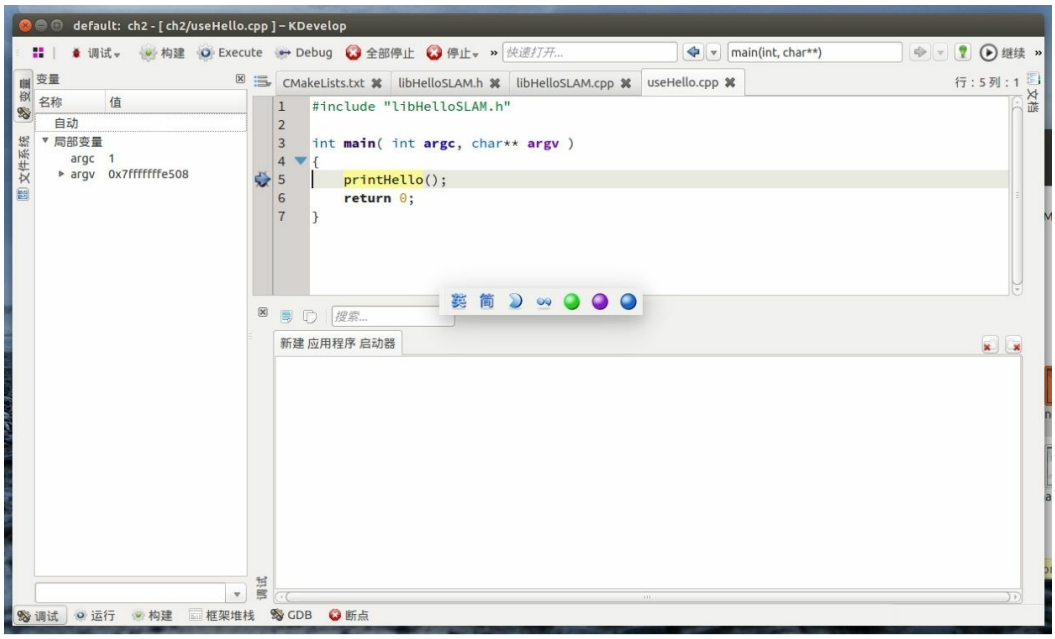


图2-14　调试界面。

调试时，Kdevelop会切换到调试模式，界面会发生一点变化。在断点处，可以用单步运行（F10键）、单步跟进（F11键）、单步跳出（F12键）功能控制程序的运行。同时，可以点开左侧的界面，查看局部变量的值。或者选择“停止”按钮，结束调试。调试结束后，Kdevelop会回到正常的开发界面。

现在你应该熟悉了整个断点调试的流程。今后，如果在程序运行阶段发生了错误，导致程序崩溃，就可以用断点调试确定出错的位置，然后加以修正[13]。

习题

1.阅读文献[1]和[14]，你能看懂其中的内容吗？

2.*阅读SLAM的综述文献，例如[9,15,16,17,18]等。这些文献关于SLAM的看法与本书有何异同？

3.g++命令有哪些参数？怎么填写参数可以更改生成的程序文件名？

4.使用build文件夹来编译你的cmake工程，然后在Kdevelop中试试。

5.刻意在代码中添加一些语法错误，看看编译会生成什么样的信

息。你能看懂g++的错误信息吗？

6.如果忘了把库链接到可执行程序上，编译会报错吗？报什么样的错？

7.*阅读《cmake实践》，了解cmake的其他语法。

8.*完善hello　SLAM小程序，把它做成一个小程序库，安装到本地硬盘中。然后，新建一个工程，使用 fi nd_package找这个库并调用。

9.*寻找其他cmake教学材料，深入了解cmake，例如https://github.com/TheErk/CMake-tutorial。

10.找到Kdevelop的官方网站，看看它还有哪些特性。你都用上了吗？

11.如果在上一讲学习了Vim，请试试Kdevelop的Vim编辑功能。

---

[1]不过现实中我们都会有一个大概的范围，例如室内和室外的区分。

[2]画成单目会比较吓人。

[3]数学上的原因将会在视觉里程计一讲中解释。

[4]你可以用手机录个小视频试试。

[5]更多时候称为后端（Back End）。由于主要使用的是优化方法，故称为后端优化。

[6]https://en.wikipedia.org/wiki/A*_search_algorithm

[7]在本书中，我们以“位姿”这个词表示“位置”加上“姿态”。

[8]我们以后称它为位姿（Pose），以与位置进行区别。我们说的位姿，包含了旋转（Rotation）和平移（Translation）。

[9]感谢TUNA同学们的维护！

[10]大多数人第一次看到Ubuntu都觉得很漂亮。

[11]MakeFile是一个自动化编译的脚本，读者现在可以将它理解成一系统自动生成的编译指令，而无须理会其内容。

[12]你多半猜错了，它并不叫作动态库。

[13]而不是直接给我们发邮件询问怎么处理遇到的问题。

书籍汇总下载 http://t.cn/RDIAj5D

第3讲　三维空间刚体运动

主要目标

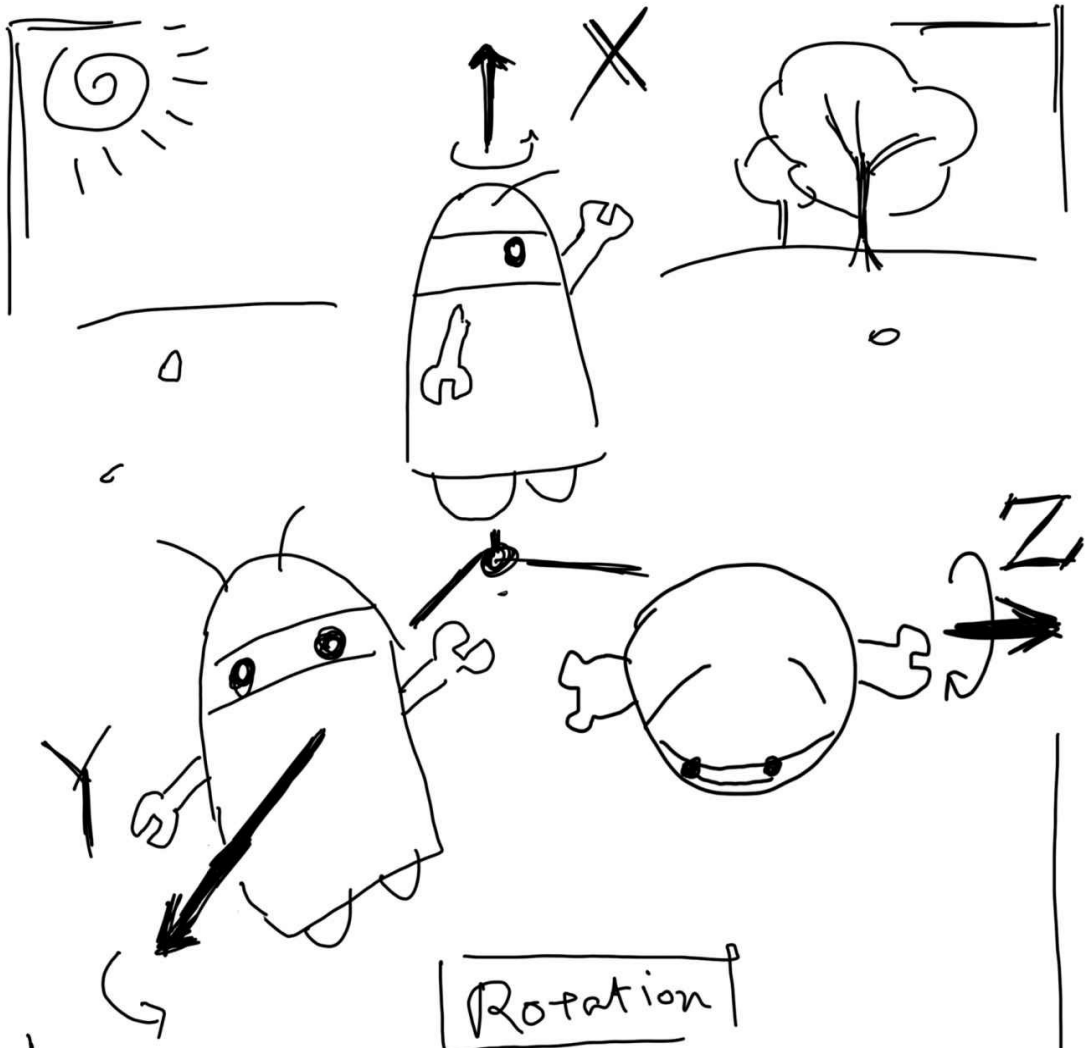
1.理解三维空间的刚体运动描述方式：旋转矩阵、变换矩阵、四元数和欧拉角。

2.掌握Eigen库的矩阵、几何模块使用方法。

在上一讲中，我们讲解了视觉SLAM的框架与内容。本讲将介绍视觉SLAM的基本问题之一：一个刚体在三维空间中的运动是如何描述的。我们当然知道这由一次旋转加一次平移组成。平移确实没有太大问题，但旋转的处理是件麻烦事。我们将介绍旋转矩阵、四元数、欧拉角的意义，以及它们是如何运算和转换的。在实践部分，我们将介绍线性代数库Eigen。它提供了C++中的矩阵运算，并且它的Geometry模块还提供了四元数等刚体运动的描述。Eigen的优化非常完善，但是它的使用方法有一些特殊的地方，我们会在程序中介绍。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D



Rotation

$$SO(3) = \{R \mid R^T R = I, \det(R) = 1\}$$

roll, pitch, yaw

$$q = q_0 + q_1 i + q_2 j + q_3 k$$

$$T = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \in SE(3)$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

3.1 旋转矩阵

3.1.1 点和向量，坐标系

我们日常生活的空间是三维的，因此我们生来就习惯于三维空间的运动。三维空间由3个轴组成，所以一个空间点的位置可以由3个坐标指定。不过，我们现在要考虑刚体，它不光有位置，还有自身的姿态。相机也可以看成三维空间的刚体，于是位置是指相机在空间中的哪个地方，而姿态则是指相机的朝向。结合起来，我们可以说，“相机正处于空间(0,0,0)点处，朝向正前方”这样的话。但是这种自然语言很烦琐，我们更喜欢用数学语言来描述它。

我们从最基本的内容讲起：点和向量。点的几何意义很容易理解。向量是什么呢？它是线性空间中的一个元素，可以把它想象成从原点指向某处的一个箭头。需要提醒读者的是，请不要把向量与它的坐标两个概念混淆。一个向量是空间当中的一样东西，比如说$\boldsymbol{a}$。这里$\boldsymbol{a}$并不是和若干个实数相关联的。只有当我们指定这个三维空间中的某个坐标系时，才可以谈论该向量在此坐标系下的坐标，也就是找到若干个实数对应这个向量。例如，三维空间中的某个向量的坐标可以用R^3当中的3个数来描述。某个点的坐标也可以用R^3来描述。怎么描述呢？如果我们确定了一个坐标系，也就是一个线性空间的基$(\boldsymbol{e}_1,\boldsymbol{e}_2,\boldsymbol{e}_3)$，那么就可以谈论向量$\boldsymbol{a}$在这组基下的坐标了：

$$\boldsymbol{a} = [\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3] \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = a_1\boldsymbol{e}_1 + a_2\boldsymbol{e}_2 + a_3\boldsymbol{e}_3. \tag{3.1}$$

所以坐标的具体取值，一是和向量本身有关，二是和坐标系的选取有关。坐标系通常由3个正交的坐标轴组成（尽管也可以有非正交的，但实际中很少见）。例如，给定x和y轴时，z轴就可以通过右手（或左

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

手）法则由$x \times y$定义出来。根据定义方式的不同，坐标系又分为左手系和右手系。左手系的第3个轴与右手系方向相反。就经验来讲，人们更习惯使用右手系，尽管也有一部分程序库仍使用左手系。

根据基本的线性代数知识，我们可以谈论向量与向量，以及向量与数之间的运算，例如数乘、加法、减法、内积、外积等。数乘和四则运算都是相当基本的内容，这里不再赘述。内外积对读者来说可能有些陌生，这里给出它们的运算方式。对于$a,b \in \mathbb{R}^3$，内积可以写成：

$$\boldsymbol{a} \cdot \boldsymbol{b} = \boldsymbol{a}^{\mathrm{T}} \boldsymbol{b} = \sum_{i=1}^{3} a_i b_i = |\boldsymbol{a}| \, |\boldsymbol{b}| \cos \langle \boldsymbol{a}, \boldsymbol{b} \rangle . \tag{3.2}$$

内积可以描述向量间的投影关系。而外积则是这个样子：

$$\boldsymbol{a} \times \boldsymbol{b} = \begin{bmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{bmatrix} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \boldsymbol{b} \triangleq \boldsymbol{a}^{\wedge} \boldsymbol{b}. \tag{3.3}$$

外积的方向垂直于这两个向量，大小为$|a||b|\sin\langle a,b\rangle$，是两个向量张成的四边形的有向面积。对于外积，我们引入了$^{\wedge}$符号，把a写成一个矩阵。事实上是一个反对称矩阵（Skew-symmetric），你可以将$^{\wedge}$记成一个反对称符号。这样就把外积$a \times b$写成了矩阵与向量的乘法$a^{\wedge} b$，把它变成了线性运算。这个符号将在后文经常用到，请记住它。外积只对三维向量存在定义，我们还能用外积表示向量的旋转。

为什么外积可以表示旋转呢？

考虑两个不平行的向量a,b，我们要描述从a到b之间是如何旋转的，如图3-1所示。我们可以用一个向量来描述三维空间中两个向量的旋转关系。在右手法则下，我们用右手的4个指头从a转向b，大拇指朝向就是旋转向量的方向，事实上也是$a \times b$的方向。它的大小则由a和b的夹角决定。通过这种方式，我们构造了从a到b的一个旋转向量。这个向量同样位于三维空间中，在此坐标系下，可以用3个实数来描述。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

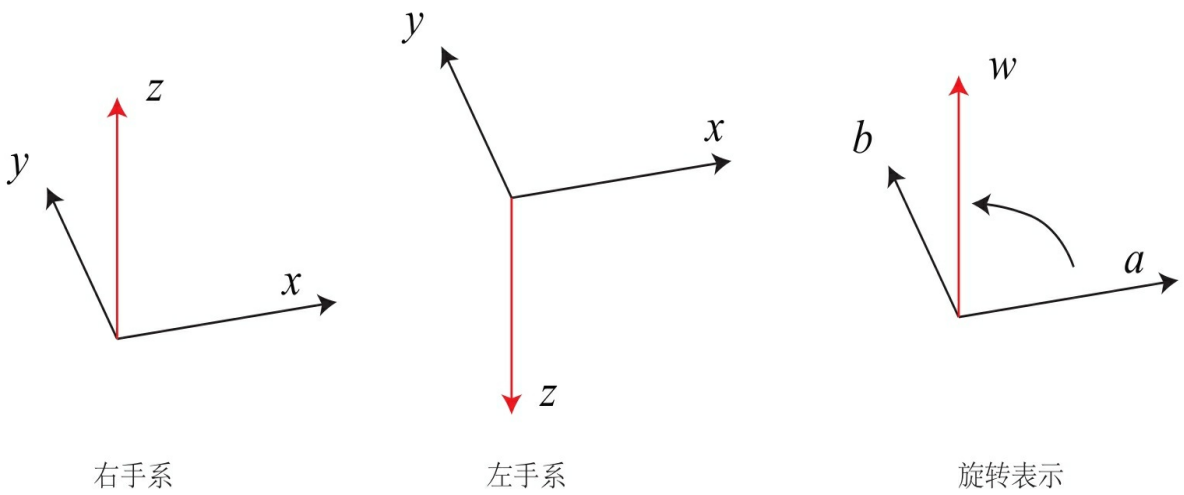


图3-1　左右手系的区别与向量间的旋转。a到b的旋转可以由向量w来描述。

3.1.2　坐标系间的欧氏变换

与向量间的旋转类似，同样可以描述两个坐标系之间的旋转关系，再加上平移，统称为坐标系之间的变换关系。在机器人的运动过程中，常见的做法是设定一个惯性坐标系（或者叫世界坐标系），可以认为它是固定不动的，例如图3-2中的x_W,y_W,z_W定义的坐标系。同时，相机或机器人则是一个移动坐标系，例如x_C,y_C,z_C定义的坐标系。我们可能会问：相机视野中某个向量p，它的坐标为p_c，而在世界坐标系下看，它的坐标p_w。这两个坐标之间是如何转换的呢？这时，就需要先得到该点针对机器人坐标系的坐标值，再根据机器人位姿转换到世界坐标系中，这个转换关系由一个矩阵T来描述，如图3-2所示。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

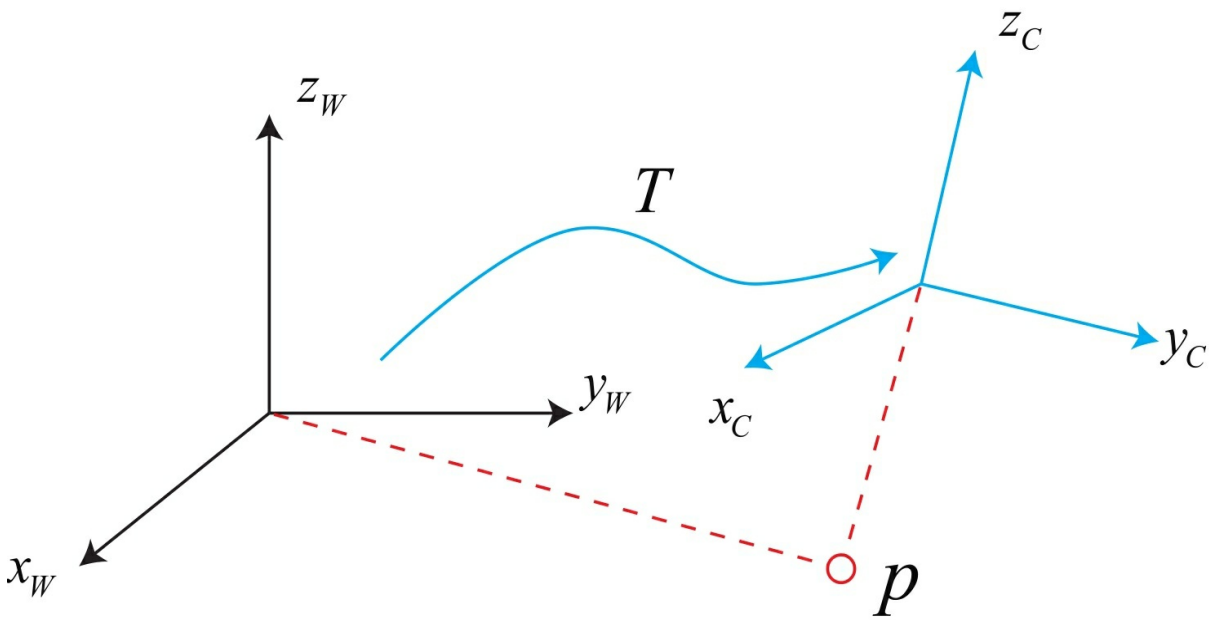


图3-2　坐标变换。对于同一个向量p，它在世界坐标系下的坐标p_w和在相机坐标系下的坐标p_c是不同的。这个变换关系由坐标系间的变换矩阵T来描述。

相机运动是一个刚体运动，它保证了同一个向量在各个坐标系下的长度和夹角都不会发生变化。这种变换称为欧氏变换。想象你把手机抛到空中，在它落地摔碎之前，只可能有空间位置和姿态的不同，而它自己的长度、各个面的角度等性质不会有任何变化。这样一个欧氏变换由一个旋转和一个平移两部分组成。首先来考虑旋转。我们设某个单位正交基$(\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3)$经过一次旋转变成了$(\boldsymbol{e}'_1, \boldsymbol{e}'_2, \boldsymbol{e}'_3)$。那么，对于同一个向量$\boldsymbol{a}$（注意该向量并没有随着坐标系的旋转而发生运动），它在两个坐标系下的坐标为$[a_1, a_2, a_3]^{\mathrm{T}}$和$[a'_1, a'_2, a'_3]^{\mathrm{T}}$。根据坐标的定义，有：

$$[\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3] \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = [\boldsymbol{e}'_1, \boldsymbol{e}'_2, \boldsymbol{e}'_3] \begin{bmatrix} a'_1 \\ a'_2 \\ a'_3 \end{bmatrix}. \tag{3.4}$$

为了描述两个坐标之间的关系，我们对上述等式的左右两边同时左

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

乘 $\begin{bmatrix} \boldsymbol{e}_1^{\mathrm{T}} \\ \boldsymbol{e}_2^{\mathrm{T}} \\ \boldsymbol{e}_3^{\mathrm{T}} \end{bmatrix}$，那么左边的系数就变成了单位矩阵，所以：

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{e}_1^{\mathrm{T}}\boldsymbol{e}_1' & \boldsymbol{e}_1^{\mathrm{T}}\boldsymbol{e}_2' & \boldsymbol{e}_1^{\mathrm{T}}\boldsymbol{e}_3' \\ \boldsymbol{e}_2^{\mathrm{T}}\boldsymbol{e}_1' & \boldsymbol{e}_2^{\mathrm{T}}\boldsymbol{e}_2' & \boldsymbol{e}_2^{\mathrm{T}}\boldsymbol{e}_3' \\ \boldsymbol{e}_3^{\mathrm{T}}\boldsymbol{e}_1' & \boldsymbol{e}_3^{\mathrm{T}}\boldsymbol{e}_2' & \boldsymbol{e}_3^{\mathrm{T}}\boldsymbol{e}_3' \end{bmatrix} \begin{bmatrix} a_1' \\ a_2' \\ a_3' \end{bmatrix} \triangleq \boldsymbol{R}\boldsymbol{a}'. \tag{3.5}$$

我们把中间的矩阵拿出来，定义成一个矩阵$\boldsymbol{R}$。这个矩阵由两组基之间的内积组成，刻画了旋转前后同一个向量的坐标变换关系。只要旋转是一样的，那么这个矩阵也是一样的。可以说，矩阵$\boldsymbol{R}$描述了旋转本身。因此又称为旋转矩阵。

旋转矩阵有一些特别的性质。事实上，它是一个行列式为1的正交矩阵[1]。反之，行列式为1的正交矩阵也是一个旋转矩阵。所以，可以把旋转矩阵的集合定义如下：

$$\mathrm{SO}(n) = \{\boldsymbol{R} \in \mathbb{R}^{n\times n} | \boldsymbol{R}\boldsymbol{R}^{\mathrm{T}} = \boldsymbol{I}, \det(\boldsymbol{R}) = 1\}. \tag{3.6}$$

SO(n)是特殊正交群（Special Orthogonal Group）的意思。我们把解释“群”的内容留到下一讲。这个集合由n维空间的旋转矩阵组成，特别地，SO(3)就是三维空间的旋转了。通过旋转矩阵，我们可以直接谈论两个坐标系之间的旋转变换，而不用再从基开始谈起。换句话说，旋转矩阵可以描述相机的旋转。

由于旋转矩阵为正交矩阵，它的逆（即转置）描述了一个相反的旋转。按照上面的定义方式，有：

$$\boldsymbol{a}' = \boldsymbol{R}^{-1}\boldsymbol{a} = \boldsymbol{R}^{\mathrm{T}}\boldsymbol{a}. \tag{3.7}$$

显然$\boldsymbol{R}^{\mathrm{T}}$刻画了一个相反的旋转。

在欧氏变换中，除了旋转之外还有平移。考虑世界坐标系中的向量$\boldsymbol{a}$，经过一次旋转（用$\boldsymbol{R}$描述）和一次平移$\boldsymbol{t}$后，得到了$\boldsymbol{a}'$，那么把旋转和

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

平移合到一起，有：

$$\boldsymbol{a}' = \boldsymbol{R}\boldsymbol{a} + \boldsymbol{t}. \tag{3.8}$$

其中，t称为平移向量。相比于旋转，平移部分只需把这个平移量加到旋转之后的坐标上，显得非常简洁。通过上式，我们用一个旋转矩阵R和一个平移向量t完整地描述了一个欧氏空间的坐标变换关系。

3.1.3 变换矩阵与齐次坐标

式（3.8）完整地表达了欧氏空间的旋转与平移，不过还存在一个小问题：这里的变换关系不是一个线性关系。假设我们进行了两次变换：R_1,t_1和R_2,t_2，满足：

$$\boldsymbol{b} = \boldsymbol{R}_1\boldsymbol{a} + \boldsymbol{t}_1, \quad \boldsymbol{c} = \boldsymbol{R}_2\boldsymbol{b} + \boldsymbol{t}_2.$$

但是从a到c的变换为

$$\boldsymbol{c} = \boldsymbol{R}_2\left(\boldsymbol{R}_1\boldsymbol{a} + \boldsymbol{t}_1\right) + \boldsymbol{t}_2.$$

这样的形式在变换多次之后会过于复杂。因此，我们要引入齐次坐标和变换矩阵重写式(3.8)：

$$\begin{bmatrix} \boldsymbol{a}' \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{a} \\ 1 \end{bmatrix} \triangleq \boldsymbol{T} \begin{bmatrix} \boldsymbol{a} \\ 1 \end{bmatrix}. \tag{3.9}$$

这是一个数学技巧：我们在一个三维向量的末尾添加1，将其变成了四维向量，称为齐次坐标。对于这个四维向量，我们可以把旋转和平移写在一个矩阵里面，使得整个关系变成线性关系。该式中，矩阵T称为变换矩阵（Transform Matrix）。我们暂时用$\tilde{\boldsymbol{a}}$表示a的齐次坐标。

稍微说一下齐次坐标。它是射影几何里的概念。通过添加最后一维，我们用4个实数描述了一个三维向量，这显然多了一个自由度，但允许我们把变换写成线性的形式。在齐次坐标中，某个点x的每个分量同乘一个非零常数k后，仍然表示同一个点。因此，一个点的具体坐标

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

值不是唯一的。如$[1,1,1,1]^{\mathrm{T}}$和$[2,2,2,2]^{\mathrm{T}}$是同一个点。但当最后一项不为零时，我们总可以把所有坐标除以最后一项，强制最后一项为1，从而得到一个点唯一的坐标表示（也就是转换成非齐次坐标）：

$$\tilde{\boldsymbol{x}} = [x, y, z, w]^{\mathrm{T}} = [x/w, y/w, z/w, 1]^{\mathrm{T}}. \tag{3.10}$$

这时，忽略掉最后一项，这个点的坐标和欧氏空间就是一样的。依靠齐次坐标和变换矩阵，两次变换的累加就可以有很好的形式：

$$\tilde{\boldsymbol{b}} = \boldsymbol{T}_1 \tilde{\boldsymbol{a}},\ \tilde{\boldsymbol{c}} = \boldsymbol{T}_2 \tilde{\boldsymbol{b}} \quad \Rightarrow \tilde{\boldsymbol{c}} = \boldsymbol{T}_2 \boldsymbol{T}_1 \tilde{\boldsymbol{a}}. \tag{3.11}$$

但是区分齐次和非齐次坐标的符号令我们感到厌烦。在不引起歧义的情况下，以后我们就直接把它写成*b=Ta*的样子，默认其中是齐次坐标了。

关于变换矩阵*T*，它具有比较特别的结构：左上角为旋转矩阵，右侧为平移向量，左下角为0向量，右下角为1。这种矩阵又称为特殊欧氏群（Special Euclidean Group）：

$$\mathrm{SE}(3) = \left\{ \boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} | \boldsymbol{R} \in \mathrm{SO}(3), \boldsymbol{t} \in \mathbb{R}^3 \right\}. \tag{3.12}$$

与SO(3)一样，求解该矩阵的逆表示一个反向的变换：

$$\boldsymbol{T}^{-1} = \begin{bmatrix} \boldsymbol{R}^{\mathrm{T}} & -\boldsymbol{R}^{\mathrm{T}} \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix}. \tag{3.13}$$

最后，为了保持符号的简洁，在不引起歧义的情况下，我们以后不区别齐次坐标与普通坐标的符号，默认使用的是符合运算法则的那一种。例如，当我们写*Ta*时，使用的是齐次坐标（不然没法计算）。而写*Ra*时，使用的是非齐次坐标。如果写在一个等式中，就假设齐次坐标到普通坐标的转换，是已经做好了的——因为齐次坐标和非齐次坐标之间的转换事实上非常容易。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

回顾一下：首先，介绍了向量及其坐标表示，并介绍了向量间的运算；然后，坐标系之间的运动由欧氏变换描述，它由平移和旋转组成。旋转可以由旋转矩阵SO(3)描述，而平移直接由一个R^3向量描述。最后，如果将平移和旋转放在一个矩阵中，就形成了变换矩阵SE(3)。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

3.2 实践：Eigen

本讲的实践部分有两节。第一部分中，将讲解如何使用Eigen来表示矩阵、向量，随后引申至旋转矩阵与变换矩阵的计算。本节的代码在slambook/ch3/useEigen中。

Eigen[2]是一个C++开源线性代数库。它提供了快速的有关矩阵的线性代数运算，还包括解方程等功能。许多上层的软件库也使用Eigen进行矩阵运算，包括g2o、Sophus等。照应本讲的理论部分，我们来学习一下Eigen的编程。

你的PC上可能还没有安装Eigen。请输入以下命令进行安装：

```
sudo apt-get install libeigen3-dev
```

大部分常用的库都已在Ubuntu软件源中提供。以后，若想要安装某个库，不妨先搜索一下Ubuntu的软件源中是否已提供。通过apt命令，我们能够方便地安装Eigen。回顾上一讲的知识，我们知道一个库由头文件和库文件组成。Eigen头文件的默认位置在“/usr/include/eigen3/”中。如果不确定，可以输入以下命令查找：

```
sudo updatedb
locate eigen3
```

相比于其他库，Eigen的特殊之处在于，它是一个纯用头文件搭建起来的库（这非常神奇！）。这意味着你只能找到它的头文件，而没有.so或.a那样的二进制文件。在使用时，只需引入Eigen的头文件即可，不需要链接库文件（因为它没有库文件）。下面写一段代码来实际练习一下Eigen的使用：

slambook/ch3/useEigen/eigenMatrix.cpp

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```cpp
#include <iostream>
#include <ctime>
using namespace std;

// Eigen 部分
#include <Eigen/Core>
// 稠密矩阵的代数运算（逆、特征值等）
#include <Eigen/Dense>

#define MATRIX_SIZE 50

/****************************
* 本程序演示了 Eigen 基本类型的使用
****************************/

int main( int argc, char** argv )
{
    // Eigen 以矩阵为基本数据单元。它是一个模板类。它的前三个参数为：数据类型，行，列
    // 声明一个 2×3 的 float 矩阵
    Eigen::Matrix<float, 2, 3> matrix_23;
    // 同时，Eigen 通过 typedef 提供了许多内置类型，不过底层仍是 Eigen::Matrix
    // 例如 Vector3d 实质上是 Eigen::Matrix<double, 3, 1>
    Eigen::Vector3d v_3d;
    // 还有 Matrix3d 实质上是 Eigen::Matrix<double, 3, 3>
    Eigen::Matrix3d matrix_33 = Eigen::Matrix3d::Zero(); //初始化为零
    // 如果不确定矩阵大小，可以使用动态大小的矩阵
    Eigen::Matrix< double, Eigen::Dynamic, Eigen::Dynamic > matrix_dynamic;
    // 更简单的
    Eigen::MatrixXd matrix_x;
    // 这种类型还有很多，我们不一一列举

    // 下面是对矩阵的操作
    // 输入数据
    matrix_23 << 1, 2, 3, 4, 5, 6;
    // 输出
    cout << matrix_23 << endl;

    // 用()访问矩阵中的元素
    for (int i=0; i<1; i++)
        for (int j=0; j<2; j++)
            cout<<matrix_23(i,j)<<endl;
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
42
43      v_3d << 3, 2, 1;
44      // 矩阵和向量相乘（实际上仍是矩阵和矩阵）
45      // 但是在这里你不能混合两种不同类型的矩阵，像这样是错的
46      // Eigen::Matrix<double, 2, 1> result_wrong_type = matrix_23 * v_3d;
47
48      // 应该显式转换
49      Eigen::Matrix<double, 2, 1> result = matrix_23.cast<double>() * v_3d;
50      cout << result << endl;
51
52      // 同样你不能搞错矩阵的维度
53      // 试着取消下面的注释，看看会报什么错
54      // Eigen::Matrix<double, 2, 3> result_wrong_dimension = matrix_23.cast<double
        >() * v_3d;
55
56      // 一些矩阵运算
57      // 四则运算就不演示了，直接用对应的运算符即可。
58      matrix_33 = Eigen::Matrix3d::Random();
59      cout << matrix_33 << endl << endl;
60
61      cout << matrix_33.transpose() << endl;      //转置
62      cout << matrix_33.sum() << endl;            //各元素和
63      cout << matrix_33.trace() << endl;          //迹
64      cout << 10*matrix_33 << endl;               //数乘
65      cout << matrix_33.inverse() << endl;        //逆
66      cout << matrix_33.determinant() << endl;    //行列式
67
68      // 特征值
69      // 实对称矩阵可以保证对角化成功
70      Eigen::SelfAdjointEigenSolver<Eigen::Matrix3d> eigen_solver ( matrix_33.
        transpose() * matrix_33 );
71      cout << "Eigen values = " << eigen_solver.eigenvalues() << endl;
72      cout << "Eigen vectors = " << eigen_solver.eigenvectors() << endl;
73
74      // 解方程
75      // 我们求解 matrix_NN * x = v_Nd 这个方程
76      // N 的大小在前边的宏里定义，矩阵由随机数生成
77      // 直接求逆自然是最直接的，但是求逆运算量大
78
79      Eigen::Matrix< double, MATRIX_SIZE, MATRIX_SIZE > matrix_NN;
80      matrix_NN = Eigen::MatrixXd::Random( MATRIX_SIZE, MATRIX_SIZE );
81      Eigen::Matrix< double, MATRIX_SIZE,  1> v_Nd;
82      v_Nd = Eigen::MatrixXd::Random( MATRIX_SIZE,1 );
83
84      clock_t time_stt = clock(); // 计时
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

```
85      // 直接求逆
86      Eigen::Matrix<double,MATRIX_SIZE,1> x = matrix_NN.inverse()*v_Nd;
87      cout <<"time use in normal invers is " << 1000* (clock() - time_stt)/(double)
        CLOCKS_PER_SEC << "ms"<< endl;
88
89      // 通常用矩阵分解来求，例如 QR 分解，速度会快很多
90      time_stt = clock();
91      x = matrix_NN.colPivHouseholderQr().solve(v_Nd);
92      cout <<"time use in Qr compsition is " <<1000*  (clock() - time_stt)/(double)
        CLOCKS_PER_SEC <<"ms" << endl;
93
94      return 0;
95  }
```

这个例程演示了Eigen矩阵的基本操作与运算。要编译它，需要在CMakeLists.txt里指定Eigen的头文件目录：

```
1  # 添加头文件
2  include_directories( "/usr/include/eigen3" )
```

重复一遍，因为Eigen库只有头文件，所以不需要再用target_link_libraries语句将程序链接到库上。不过，对于其他大部分库，多数时候需要用到链接命令。这里的做法并不见得是最好的，因为其他人可能把Eigen安装在了不同位置，那么就必须手动修改这里的头文件目录。在之后的工作中，我们会使用 fi nd_package命令去搜索库，不过在本讲中暂时保持这个样子。编译好这个程序后，运行它，可以看到各矩阵的输出结果。

书籍汇总下载 http://t.cn/RDIAj5D

```
% build/eigenMatrix
1 2 3
4 5 6
1
2
10
28
0.680375   0.59688 -0.329554
-0.211234  0.823295  0.536459
0.566198 -0.604897 -0.444451
……
```

由于在代码中给出了详细的注释，在此就不一一解释每行语句了。本书中，我们将仅给出几处重要地方的说明（后面的实践部分亦将保持这个风格）。

1.读者最好亲手输入一遍上面的代码（不包括注释）。至少要编译运行一遍上面的程序。

2.Kdevelop可能不会提示C++成员运算，这是它做得不够完善导致的。请照着上面的内容输入即可，不必理会它是否提示错误。

3.Eigen提供的矩阵和MATLAB很相似，几乎所有的数据都当作矩阵来处理。但是，为了实现更好的效率，在Eigen中需要指定矩阵的大小和类型。对于在编译时期就知道大小的矩阵，处理起来会比动态变化大小的矩阵更快一些。因此，像旋转矩阵、变换矩阵这样的数据，完全可在编译时期确定它们的大小和数据类型。

4.Eigen内部的矩阵实现比较复杂，这里不做介绍，我们希望你像使用 fl oat、double等内置数据类型那样使用Eigen的矩阵。这应该是符合其设计初衷的。

5.Eigen矩阵不支持自动类型提升，这和C++的内建数据类型有较大差异。在C++程序中，我们可以把一个 fl oat数据和double数据相加、相乘，编译器会自动把数据类型转换为最合适的那种。而在Eigen中，出于性能的考虑，必须显式地对矩阵类型进行转换。而如果忘了这样做，Eigen会（不太友好地）提示你一个“YOU MIXED DIFFERENT NUMERIC TYPES...”的编译错误。你可以尝试找一下这条信息出现在错

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

误提示的哪个部分。如果错误信息太长最好保存到一个文件里再找。

6.同理，在计算过程中也需要保证矩阵维数的正确性，否则会出现“YOU MIXED MATRICES OF DIFFERENT SIZES”错误。请不要抱怨这种错误提示方式，对于C++模板元编程，能够提示出可以阅读的信息已经是很幸运的了。以后，若发现Eigen出错，你可以直接寻找大写的部分，推测出了什么问题。

7.我们的例程只介绍了基本的矩阵运算。你可以阅读http://eigen.tuxfamily.org/dox-devel/modules.html学习更多的Eigen知识。这里只演示了最简单的部分，能看懂演示程序不等于你已经能够熟练操作Eigen。

最后一段代码中比较了求逆与求QR分解的运行效率，你可以看看自己机器上的时间差异，两种方法是否有明显的差异？

书籍汇总下载 http://t.cn/RDIAj5D

3.3 旋转向量和欧拉角

3.3.1 旋转向量

我们重新回到理论部分。有了旋转矩阵来描述旋转，有了变换矩阵描述一个6自由度的三维刚体运动，是不是已经足够了呢？矩阵表示方式至少有以下几个缺点：

1.SO(3)的旋转矩阵有9个量，但一次旋转只有3个自由度。因此这种表达方式是冗余的。同理，变换矩阵用16个量表达了6自由度的变换。那么，是否有更紧凑的表示呢？

2.旋转矩阵自身带有约束：它必须是个正交矩阵，且行列式为1。变换矩阵也是如此。当想要估计或优化一个旋转矩阵/变换矩阵时，这些约束会使得求解变得更困难。

因此，我们希望有一种方式能够紧凑地描述旋转和平移。例如，用一个三维向量表达旋转，用六维向量表达变换，可行吗？事实上，在前面介绍外积的那部分，我们提到过如何做这件事。我们介绍了如何用外积表达两个向量的旋转关系。对于坐标系的旋转，我们知道，任意旋转都可以用一个旋转轴和一个旋转角来刻画。于是，我们可以使用一个向量，其方向与旋转轴一致，而长度等于旋转角。这种向量称为旋转向量（或轴角，Axis-Angle）。这种表示法只需一个三维向量即可描述旋转。同样，对于变换矩阵，我们使用一个旋转向量和一个平移向量即可表达一次变换。这时的维数正好是六维。

事实上，旋转向量就是下一讲准备介绍的李代数。所以本讲中读者只需知道旋转可以这样表示即可。剩下的问题是，旋转向量和旋转矩阵之间是如何转换的呢？假设有一个旋转轴为$\boldsymbol{n}$，角度为θ的旋转，显然，它对应的旋转向量为$\theta \boldsymbol{n}$。从旋转向量到旋转矩阵的转换过程由罗德里格斯公式（Rodrigues's Formula）表明，由于推导过程比较复杂，这里不作描述，只给出转换的结果[3]：

$$\boldsymbol{R} = \cos\theta \boldsymbol{I} + (1 - \cos\theta)\boldsymbol{n}\boldsymbol{n}^{\mathrm{T}} + \sin\theta \boldsymbol{n}^{\wedge}. \tag{3.14}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

符号$\wedge$是向量到反对称的转换符，见式（3.3）。反之，我们也可以计算从一个旋转矩阵到旋转向量的转换。对于转角θ，有：

$$\begin{aligned} \mathrm{tr}(\boldsymbol{R}) &= \cos\theta \mathrm{tr}(\boldsymbol{I}) + (1-\cos\theta)\,\mathrm{tr}\left(\boldsymbol{n}\boldsymbol{n}^{\mathrm{T}}\right) + \sin\theta \mathrm{tr}(\boldsymbol{n}^{\wedge}) \\ &= 3\cos\theta + (1-\cos\theta) \\ &= 1 + 2\cos\theta. \end{aligned} \tag{3.15}$$

因此：

$$\theta = \arccos\left(\frac{\mathrm{tr}(\boldsymbol{R})-1}{2}\right). \tag{3.16}$$

关于转轴$\boldsymbol{n}$，由于旋转轴上的向量在旋转后不发生改变，说明

$$\boldsymbol{R}\boldsymbol{n} = \boldsymbol{n}.$$

因此，转轴$\boldsymbol{n}$是矩阵$\boldsymbol{R}$特征值1对应的特征向量。求解此方程，再归一化，就得到了旋转轴。读者也可以从“旋转轴经过旋转之后不变”的几何角度看待这个方程。顺便提一下，这里的两个转换公式在下一讲仍将出现，你会发现它们正是SO(3)上李群与李代数的对应关系。

3.3.2 欧拉角

下面来说说欧拉角。

无论是旋转矩阵、旋转向量，它们虽然能描述旋转，但对我们人类是非常不直观的。当我们看到一个旋转矩阵或旋转向量时，很难想象出这个旋转究竟是什么样的。当它们变换时，我们也不知道物体是向哪个方向在转动。而欧拉角则提供了一种非常直观的方式来描述旋转——它使用了3个分离的转角，把一个旋转分解成3次绕不同轴的旋转。当然，由于分解方式有许多种，所以欧拉角也存在着不同的定义方法。比如说，先绕X轴旋转，再绕Y轴，最后绕Z轴，就得到了一个$XY\quad Z$轴的旋转。同理，可以定义$ZY\ Z$、$ZY\ X$等旋转方式。如果讨论得更细一些，还需要区分每次是绕固定轴旋转的，还是绕旋转之后的轴旋转的，这也会给出不一样的定义方式。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

你或许在航空、航模中听说过“俯仰角”“偏航角”这些词。欧拉角当中比较常用的一种，便是用“偏航-俯仰-滚转”（yaw-pitch-roll）3个角度来描述一个旋转的。由于它等价于*ZY X*轴的旋转，因此就以*ZY X*为例。假设一个刚体的前方（朝向我们的方向）为*X*轴，右侧为*Y*轴，上方为*Z*轴，如图3-3所示。那么，*ZY X*转角相当于把任意旋转分解成以下3个轴上的转角：

1.绕物体的*Z*轴旋转，得到偏航角yaw；

2.绕旋转之后的*Y*轴旋转，得到俯仰角pitch；

3.绕旋转之后的*X*轴旋转，得到滚转角roll。

此时，可以使用$[r,p,y]^{\mathrm{T}}$这样一个三维的向量描述任意旋转。这个向量十分直观，我们可以从这个向量想象出旋转的过程。其他的欧拉角亦是通过这种方式，把旋转分解到3个轴上，得到一个三维的向量，只不过选用的轴及顺序不一样。这里介绍的rpy角是比较常用的一种，只有很少的欧拉角种类会有rpy这样脍炙人口的名字。不同的欧拉角是按照旋转轴的顺序来称呼的。例如，rpy角的旋转顺序是*ZY X*。同样，也有*XY Z*,*ZY Z*这样的欧拉角——但是它们就没有专门的名字了。值得一提的是，大部分领域在使用欧拉角时都有各自的坐标方向和顺序上的习惯，不一定和我们这里说的相同。

欧拉角的一个重大缺点是会碰到著名的万向锁问题（Gimbal Lock[4]）：在俯仰角为$\pm 90^{\circ}$时，第一次旋转与第三次旋转将使用同一个轴，使得系统丢失了一个自由度（由3次旋转变成了2次旋转）。这被称为奇异性问题，在其他形式的欧拉角中也同样存在。理论上可以证明，只要想用3个实数来表达三维旋转时，都会不可避免地碰到奇异性问题。由于这种原理，欧拉角不适于插值和迭代，往往只用于人机交互中。我们也很少在SLAM程序中直接使用欧拉角表达姿态，同样不会在滤波或优化中使用欧拉角表达旋转（因为它具有奇异性）。不过，若你想验证自己的算法是否有错，转换成欧拉角能够快速分辨结果是否正确。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

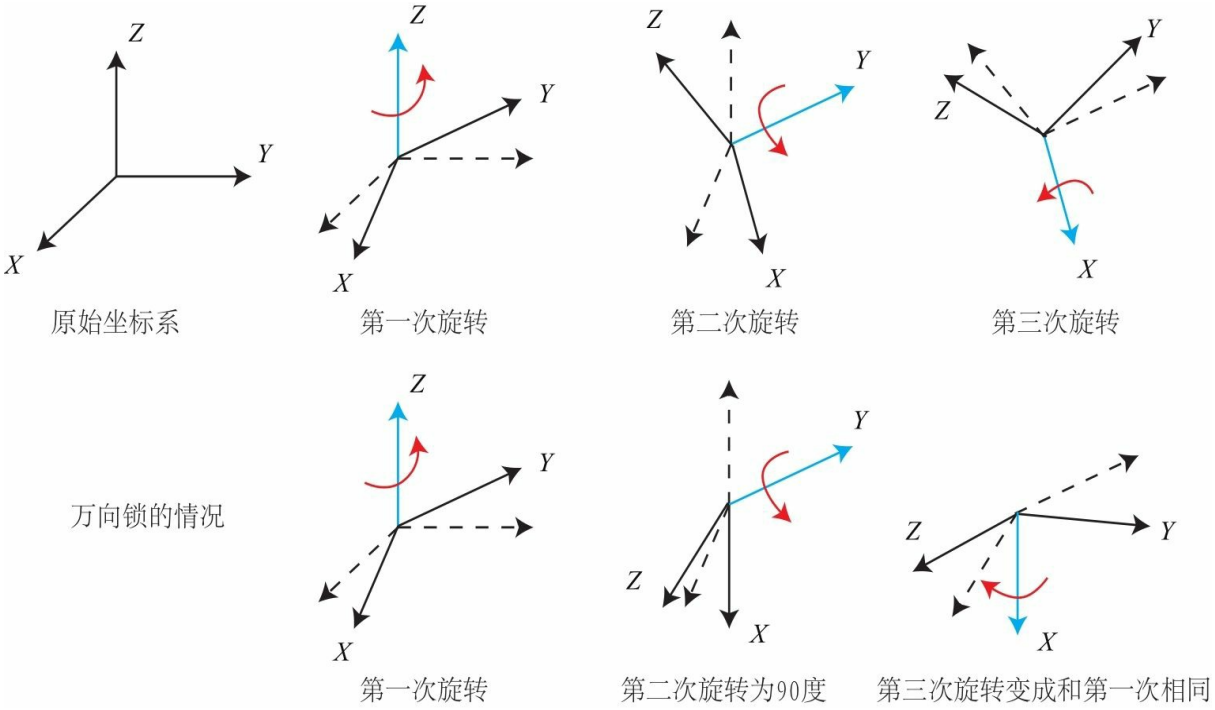


图3-3　欧拉角的旋转示意图。上方为ZYX角定义。下方为pitch=90°时，第三次旋转与第一次滚转角相同，使得系统丢失了一个自由度。如果你还没有理解万向锁，可以看看相关视频，理解起来会更方便。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

3.4 四元数

3.4.1 四元数的定义

旋转矩阵用9个量描述3自由度的旋转，具有冗余性；欧拉角和旋转向量是紧凑的，但具有奇异性。事实上，我们找不到不带奇异性的三维向量描述方式[19]。这有点类似于用两个坐标表示地球表面（如经度和纬度），将必定存在奇异性（纬度为±90°时经度无意义）。三维旋转是一个三维流形，想要无奇异性地表达它，用3个量是不够的。

回忆以前学习过的复数。我们用复数集C表示复平面上的向量，而复数的乘法则表示复平面上的旋转：例如，乘上复数i相当于逆时针把一个复向量旋转90°。类似地，在表达三维空间旋转时，也有一种类似于复数的代数：四元数（Quaternion）。四元数是Hamilton找到的一种扩展的复数。它既是紧凑的，也没有奇异性。如果说缺点，四元数不够直观，其运算稍复杂些。

一个四元数$\boldsymbol{q}$拥有一个实部和三个虚部。本书把实部写在前面（也有地方把实部写在后面），像下面这样：

$$\boldsymbol{q} = q_0 + q_1 i + q_2 j + q_3 k, \tag{3.17}$$

其中i,j,k为四元数的三个虚部。这三个虚部满足以下关系式：

$$\begin{cases} i^2 = j^2 = k^2 = -1 \\ ij = k, ji = -k \\ jk = i, kj = -i \\ ki = j, ik = -j \end{cases}. \tag{3.18}$$

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：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$\boldsymbol{q} = [s, \boldsymbol{v}], \quad s = q_0 \in \mathbb{R}, \quad \boldsymbol{v} = [q_1, q_2, q_3]^{\mathrm{T}} \in \mathbb{R}^3,$$

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

这和复数非常相似。考虑到三维空间需要3个轴，四元数也有3个虚部，那么，一个虚四元数能不能对应到一个空间点呢？事实上我们就是这样做的。同理，我们知道一个模长为1的复数可以表示复平面上的纯旋转（没有长度的缩放），那么，三维空间中的旋转是否能用单位四元数表达呢？答案也是肯定的。

我们能用单位四元数表示三维空间中任意一个旋转，不过这种表达方式和复数有着微妙的不同。在复数中，乘以*i*意味着旋转90°。这是否意味着四元数中，乘*i*就是绕*i*轴旋转90°？那么，*ij*=-*k*是否意味着，先绕*i*转90°，再绕*j*转90°，就等于绕*k*转-90°？读者可以找一部手机比划一下——然后你会发现情况并不是这样。正确的情形应该是，乘以*i*对应着旋转180°，这样才能保证*ij*=-*k*的性质。而i^2=-1，意味着绕*i*轴旋转360°后得到一个相反的东西。这个东西要旋转两周才会和它原先的样子相等。

这似乎有些玄妙了，完整的解释需要引入太多额外的东西，我们还是冷静一下回到眼前。至少，我们知道单位四元数能够表达三维空间的旋转。这种表达方式和旋转矩阵、旋转向量有什么关系呢？我们不妨先来看旋转向量。假设某个旋转是绕单位向量$\boldsymbol{n}=[n_x,n_y,n_z]^{\mathrm{T}}$进行了角度为$\theta$的旋转，那么这个旋转的四元数形式为

$$\boldsymbol{q} = \left[\cos\frac{\theta}{2}, n_x \sin\frac{\theta}{2}, n_y \sin\frac{\theta}{2}, n_z \sin\frac{\theta}{2}\right]^{\mathrm{T}}. \tag{3.19}$$

反之，亦可从单位四元数中计算出对应旋转轴与夹角：

$$\begin{cases} \theta = 2\arccos q_0 \\ [n_x, n_y, n_z]^{\mathrm{T}} = [q_1, q_2, q_3]^{\mathrm{T}} / \sin\frac{\theta}{2} \end{cases}. \tag{3.20}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

这个式子给了我们一种微妙的“转了一半”的感觉。同样，对式（3.19）的θ加上2π，我们得到一个相同的旋转，但此时对应的四元数变成了$-q$。因此，在四元数中，任意的旋转都可以由两个互为相反数的四元数表示。同理，取θ为0，则得到一个没有任何旋转的实四元数：

$$\boldsymbol{q}_0 = [\pm 1, 0, 0, 0]^{\mathrm{T}}. \tag{3.21}$$

3.4.2 四元数的运算

四元数和通常复数一样，可以进行一系列的运算。常见的有四则运算、数乘、求逆、共轭等。下面分别介绍。

现有两个四元数q_a,q_b，它们的向量表示为$[s_a,v_a]$,$[s_b,v_b]$，或者原始四元数表示为

$$\boldsymbol{q}_a = s_a + x_a i + y_a j + z_a k, \quad \boldsymbol{q}_b = s_b + x_b i + y_b j + z_b k.$$

那么，其运算可表示如下。

1.加法和减法

四元数q_a,q_b的加减运算为

$$\boldsymbol{q}_a \pm \boldsymbol{q}_b = [s_a \pm s_b, \boldsymbol{v}_a \pm \boldsymbol{v}_b]. \tag{3.22}$$

2.乘法

乘法是把q_a的每一项与q_b的每项相乘，最后相加，虚部要按照式（3.18）进行。整理可得：

$$\begin{aligned} \boldsymbol{q}_a \boldsymbol{q}_b = & s_a s_b - x_a x_b - y_a y_b - z_a z_b \\ & + (s_a x_b + x_a s_b + y_a z_b - z_a y_b) i \\ & + (s_a y_b - x_a z_b + y_a s_b + z_a x_b) j \\ & + (s_a z_b + x_a y_b - y_a x_b + z_a s_b) k. \end{aligned} \tag{3.23}$$

虽然稍为复杂，但形式上是整齐有序的。如果写成向量形式并利用内外积运算，该表达会更加简洁：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$\boldsymbol{q}_a\boldsymbol{q}_b = \left[s_a s_b - \boldsymbol{v}_a^{\mathrm{T}}\boldsymbol{v}_b, s_a\boldsymbol{v}_b + s_b\boldsymbol{v}_a + \boldsymbol{v}_a \times \boldsymbol{v}_b\right]. \tag{3.24}$$

在该乘法定义下，两个实的四元数乘积仍是实的，这与复数也是一致的。然而，注意到，由于最后一项外积的存在，四元数乘法通常是不可交换的，除非v_a和v_b在R^3中共线，此时外积项为零。

3.共轭

四元数的共轭是把虚部取成相反数：

$$\boldsymbol{q}_a^* = s_a - x_a i - y_a j - z_a k = [s_a, -\boldsymbol{v}_a]. \tag{3.25}$$

四元数共轭与其本身相乘，会得到一个实四元数，其实部为模长的平方：

$$\boldsymbol{q}^*\boldsymbol{q} = \boldsymbol{q}\boldsymbol{q}^* = [s_a^2 + \boldsymbol{v}^{\mathrm{T}}\boldsymbol{v}, \mathbf{0}]. \tag{3.26}$$

4.模长

四元数的模长定义为

$$\|\boldsymbol{q}_a\| = \sqrt{s_a^2 + x_a^2 + y_a^2 + z_a^2}. \tag{3.27}$$

可以验证，两个四元数乘积的模即为模的乘积。这保证了单位四元数相乘后仍是单位四元数。

$$\|\boldsymbol{q}_a\boldsymbol{q}_b\| = \|\boldsymbol{q}_a\|\|\boldsymbol{q}_b\|. \tag{3.28}$$

5.逆

一个四元数的逆为

$$\boldsymbol{q}^{-1} = \boldsymbol{q}^*/\|\boldsymbol{q}\|^2. \tag{3.29}$$

按此定义，四元数和自己的逆的乘积为实四元数1：

$$\boldsymbol{q}\boldsymbol{q}^{-1} = \boldsymbol{q}^{-1}\boldsymbol{q} = \mathbf{1}. \tag{3.30}$$

如果q为单位四元数，其逆和共轭就是同一个量。同时，乘积的逆

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

有和矩阵相似的性质：

$$(\boldsymbol{q}_a \boldsymbol{q}_b)^{-1} = \boldsymbol{q}_b^{-1} \boldsymbol{q}_a^{-1}. \tag{3.31}$$

6.数乘与点乘

和向量相似，四元数可以与数相乘：

$$k\boldsymbol{q} = [ks, k\boldsymbol{v}]. \tag{3.32}$$

点乘是指两个四元数每个位置上的数值分别相乘：

$$\boldsymbol{q}_a \cdot \boldsymbol{q}_b = s_a s_b + x_a x_b i + y_a y_b j + z_a z_b k. \tag{3.33}$$

3.4.3 用四元数表示旋转

我们可以用四元数表达对一个点的旋转。假设一个空间三维点$p=[x,y,z]\in\mathrm{R}^3$，以及一个由轴角n,θ指定的旋转。三维点p经过旋转之后变为p'。如果使用矩阵描述，那么有$p'=Rp$。而如果用四元数描述旋转，它们的关系又如何来表达呢？

首先，把三维空间点用一个虚四元数来描述：

$$\boldsymbol{p} = [0, x, y, z] = [0, \boldsymbol{v}].$$

这相当于把四元数的3个虚部与空间中的3个轴相对应。然后，参照式(3.19)，用四元数q表示这个旋转：

$$\boldsymbol{q} = [\cos\frac{\theta}{2}, \boldsymbol{n}\sin\frac{\theta}{2}].$$

那么，旋转后的点p'即可表示为这样的乘积：

$$\boldsymbol{p}' = \boldsymbol{q}\boldsymbol{p}\boldsymbol{q}^{-1}. \tag{3.34}$$

可以验证（留作习题），计算结果的实部为0，故为纯虚四元数。其虚部的3个分量表示旋转后3D点的坐标。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

3.4.4 四元数到旋转矩阵的转换

任意单位四元数描述了一个旋转，该旋转亦可用旋转矩阵或旋转向量描述。从旋转向量到四元数的转换方式已在式（3.20）中给出。因此，现在看来把四元数转换为矩阵的最直观方法，是先把四元数q转换为轴角θ和n，然后再根据罗德里格斯公式转换为矩阵。不过那样要计算一个arccos函数，代价较大。实际上这个计算是可以通过一定的技巧绕过的。这里省略推导过程，直接给出四元数到旋转矩阵的转换方式。

设四元数$q=q_0+q_1i+q_2j+q_3k$，对应的旋转矩阵R为

$$\boldsymbol{R} = \begin{bmatrix} 1-2q_2^2-2q_3^2 & 2q_1q_2+2q_0q_3 & 2q_1q_3-2q_0q_2 \\ 2q_1q_2-2q_0q_3 & 1-2q_1^2-2q_3^2 & 2q_2q_3+2q_0q_1 \\ 2q_1q_3+2q_0q_2 & 2q_2q_3-2q_0q_1 & 1-2q_1^2-2q_2^2. \end{bmatrix} \tag{3.35}$$

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

$$q_0 = \frac{\sqrt{\mathrm{tr}(R)+1}}{2}, q_1 = \frac{m_{23}-m_{32}}{4q_0}, q_2 = \frac{m_{31}-m_{13}}{4q_0}, q_3 = \frac{m_{12}-m_{21}}{4q_0}. \tag{3.36}$$

值得一提的是，由于q和$-q$表示同一个旋转，事实上一个R对应的四元数表示并不是唯一的。同时，除了上面给出的转换方式之外，还存在其他几种计算方法，而本书都省略了。实际编程中，当q_0接近0时，其余3个分量会非常大，导致解不稳定，此时我们再考虑使用其他的方式进行转换。

最后，无论是四元数、旋转矩阵还是轴角，它们都可以用来描述同一个旋转。我们应该在实际中选择最为方便的形式，而不必拘泥于某种特定的形式。在随后的实践和习题中，我们会演示各种表达方式之间的转换，以加深读者的印象。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

3.5 *相似、仿射、射影变换

3D空间中的变换，除了欧氏变换之外，还存在其他几种，只不过欧氏变换是最简单的。它们一部分和测量几何有关，因为在之后的讲解中可能会提到，所以先罗列出来。欧氏变换保持了向量的长度和夹角，相当于我们把一个刚体原封不动地进行了移动或旋转，不改变它自身的样子。而其他几种变换则会改变它的外形。它们都拥有类似的矩阵表示。

1.相似变换

相似变换比欧氏变换多了一个自由度，它允许物体进行均匀缩放，其矩阵表示为

$$\boldsymbol{T}_S = \begin{bmatrix} s\boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix}. \tag{3.37}$$

注意到旋转部分多了一个缩放因子s，表示我们在对向量旋转之后，可以在x,y,z三个坐标上进行均匀缩放。由于含有缩放，相似变换不再保持图形的面积不变。你可以想象一个边长为1的立方体通过相似变换后，变成边长为10的样子（但仍然是立方体）。

2.仿射变换

仿射变换的矩阵形式如下：

$$\boldsymbol{T}_A = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix}. \tag{3.38}$$

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。仿射变换也叫正交投影。经过仿射变换之后，立方体就不再是方的了，但是各个面仍然是平行四边形。

3.射影变换

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

射影变换是最一般的变换，它的矩阵形式为

$$\boldsymbol{T}_P = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ \boldsymbol{a}^{\mathrm{T}} & v \end{bmatrix}. \tag{3.39}$$

它的左上角为可逆矩阵A，右上角为平移t，左下角为缩放a^{T}。由于采用了齐次坐标，当v/=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；否则得到右下角为0的矩阵。因此，2D的射影变换一共有8个自由度，3D则共有15个自由度。射影变换是现在讲过的变换中，形式最为一般的。从真实世界到相机照片的变换可以看成一个射影变换。读者可以想象一个原本方形的地板砖，在照片当中是什么样子：首先，它不再是方形的。由于近大远小的关系，它甚至不是平行四边形，而是一个不规则的四边形。

表3-1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3-1常见变换性质比较

变换名称	矩阵形式	自由度	不变性质
欧氏变换	$\begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix}$	6	长度、夹角、体积
相似变换	$\begin{bmatrix} s\boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix}$	7	体积比
仿射变换	$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix}$	12	平行性、体积比
射影变换	$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ \boldsymbol{a}^{\mathrm{T}} & v \end{bmatrix}$	15	接触平面的相交和相切

我们之后会说到，从真实世界到相机照片的变换是一个射影变换。如果相机的焦距为无穷远，那么这个变换为仿射变换。不过，在详细讲述相机模型之前，我们只要对它们有个大致的印象即可。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

3.6 实践：Eigen几何模块

现在，我们来实际演练一下前面讲到的各种旋转表达方式。我们将在Eigen中使用四元数、欧拉角和旋转矩阵，演示它们之间的变换方式。我们还会给出一个可视化程序，帮助读者理解这几个变换的关系。

slambook/ch3/useGeometry/useGeometry.cpp

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```cpp
#include <iostream>
#include <cmath>
using namespace std;

#include <Eigen/Core>
// Eigen 几何模块
#include <Eigen/Geometry>

/*****************************
* 本程序演示了 Eigen 几何模块的使用方法
*****************************/

int main( int argc, char** argv )
{
    // Eigen/Geometry 模块提供了各种旋转和平移的表示
    // 3D 旋转矩阵直接使用 Matrix3d 或 Matrix3f
    Eigen::Matrix3d rotation_matrix = Eigen::Matrix3d::Identity();
    // 旋转向量使用 AngleAxis, 它底层不直接是 Matrix ，但运算可以当作矩阵（因为重载了运算符）
    Eigen::AngleAxisd rotation_vector ( M_PI/4, Eigen::Vector3d ( 0,0,1 ) );     // 沿 Z 轴旋转 45 度
    cout .precision(3);
    cout<<"rotation matrix =\n"<<rotation_vector.matrix() <<endl;                // 用 matrix() 转换成矩阵
    // 也可以直接赋值
    rotation_matrix = rotation_vector.toRotationMatrix();
    // 用 AngleAxis 可以进行坐标变换
    Eigen::Vector3d v ( 1,0,0 );
    Eigen::Vector3d v_rotated = rotation_vector * v;
    cout<<"(1,0,0) after rotation = "<<v_rotated.transpose()<<endl;
    // 或者用旋转矩阵
    v_rotated = rotation_matrix * v;
    cout<<"(1,0,0) after rotation = "<<v_rotated.transpose()<<endl;

    // 欧拉角: 可以将旋转矩阵直接转换成欧拉角
    Eigen::Vector3d euler_angles = rotation_matrix.eulerAngles ( 2,1,0 ); // ZYX 顺序，即 yaw pitch roll 顺序
    cout<<"yaw pitch roll = "<<euler_angles.transpose()<<endl;

    // 欧氏变换矩阵使用 Eigen::Isometry
    Eigen::Isometry3d T=Eigen::Isometry3d::Identity();      // 虽然称为 3d ，实质上是 4×4 的矩阵
    T.rotate ( rotation_vector );    // 按照 rotation_vector 进行旋转
    T.pretranslate ( Eigen::Vector3d ( 1,3,4 ) );       // 把平移向量设成 (1,3,4)
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

```
40      cout << "Transform matrix = \n" << T.matrix() <<endl;
41
42      // 用变换矩阵进行坐标变换
43      Eigen::Vector3d v_transformed = T*v;                  // 相当于 R*v+t
44      cout<<"v tranformed = "<<v_transformed.transpose()<<endl;
45
46      // 对于仿射和射影变换，使用 Eigen::Affine3d 和 Eigen::Projective3d 即可，略
47
48      // 四元数
49      // 可以直接把 AngleAxis 赋值给四元数，反之亦然
50      Eigen::Quaterniond q = Eigen::Quaterniond ( rotation_vector );
51      cout<<"quaternion = \n"<<q.coeffs() <<endl;   // 注意 coeffs 的顺序是 (x,y,z,w)
        ，w 为实部，前三者为虚部
52      // 也可以把旋转矩阵赋给它
53      q = Eigen::Quaterniond ( rotation_matrix );
54      cout<<"quaternion = \n"<<q.coeffs() <<endl;
55      // 使用四元数旋转一个向量，使用重载的乘法即可
56      v_rotated = q*v; // 注意数学上是 qvq^{-1}
57      cout<<"(1,0,0) after rotation = "<<v_rotated.transpose()<<endl;
58
59      return 0;
60  }
```

Eigen中对各种形式的表达方式总结如下。请注意每种类型都有单精度和双精度两种数据类型，而且和之前一样，不能由编译器自动转换。下面以双精度为例，你可以把最后的d改成f，即得到单精度的数据结构。

- 旋转矩阵（3×3）：Eigen::Matrix3d。
- 旋转向量（3×1）：Eigen::AngleAxisd。
- 欧拉角（3×1）：Eigen::Vector3d。
- 四元数（4×1）：Eigen::Quaterniond。
- 欧氏变换矩阵（4×4）：Eigen::Isometry3d。
- 仿射变换（4×4）：Eigen::A ffi ne3d。

书籍汇总下载 http://t.cn/RDIAj5D

•射影变换（4×4）：Eigen::Projective3d。

我们把如何编译此程序的问题交给读者。在这个程序中，演示了如何使用Eigen中的旋转矩阵、旋转向量（AngleAxis）、欧拉角和四元数。我们用这几种旋转方式去旋转一个向量*v*，发现结果是一样的（不一样就真是见鬼了）。同时，也演示了如何在程序中转换这几种表达方式。想进一步了解Eigen的几何模块的读者可以参考（http://eigen.tuxfamily.org/dox/group__TutorialGeometry.html）。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

3.7 可视化演示

最后，我们为读者准备了一个小程序，位于slambook/ch3/visualizeGeometry中。它以可视化的形式演示了各种表达方式的异同（见图3-4）。读者可以用鼠标操作一下，看看数据是如何变化的。

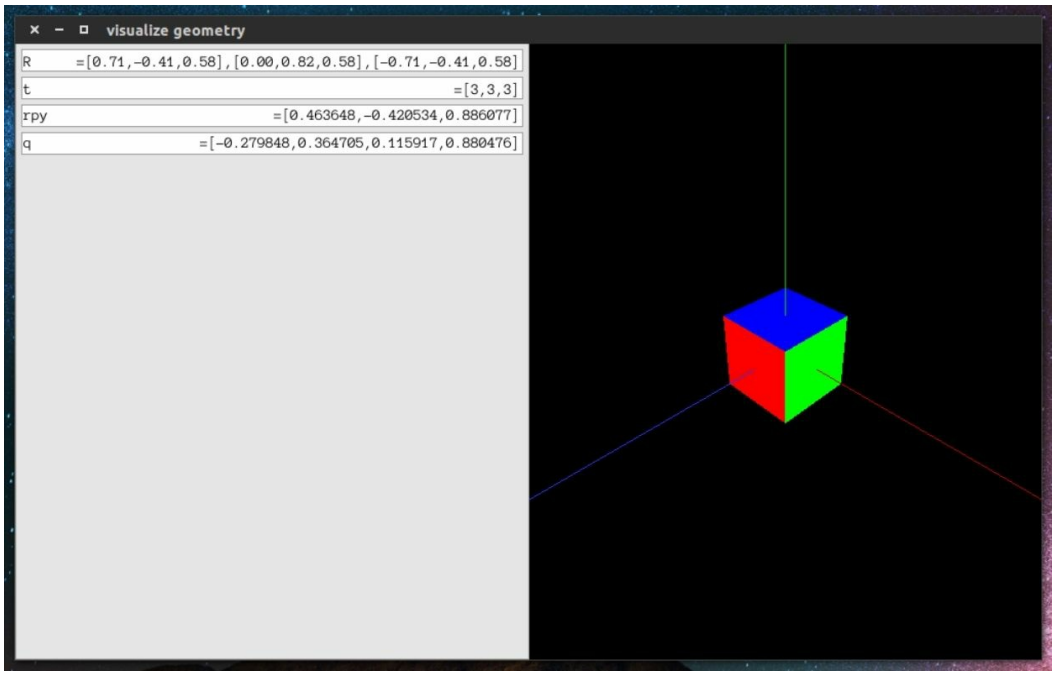


图3-4 旋转矩阵、欧拉角、四元数的可视化程序。

在这个小程序中，我们在坐标原点放置了一个彩色立方体。用鼠标可以平移/旋转相机。你可以实时地看到相机姿态的变化。我们显示了变换矩阵R,t、欧拉角和四元数的3种姿态，你可以实际体验一下这几个量是如何变化的。然而根据我们的经验，除了欧拉角之外，你应该看不出它们直观的含义。

关于该程序的源代码，这里就不做解释了，如果你感兴趣，可以自行查看。该程序的编译说明请参照其中的Readme.txt。

值得一提的是，实际中，我们会至少定义两个坐标系：世界坐标系和相机坐标系。在该定义下，设某个点在世界坐标系中的坐标为p_w，在

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

相机坐标系下为p_c，那么：

$$\boldsymbol{p}_c = \boldsymbol{T}_{\mathrm{cw}}\boldsymbol{p}_w, \tag{3.40}$$

这里T_{cw}表示世界坐标系到相机坐标系间的变换。或者可以用反过来的T_{wc}：

$$\boldsymbol{p}_w = \boldsymbol{T}_{\mathrm{wc}}\boldsymbol{p}_c = \boldsymbol{T}_{\mathrm{cw}}^{-1}\boldsymbol{p}_c. \tag{3.41}$$

原则上，T_{wc}和T_{cw}都可以用来表示相机的位姿，事实上它们也只差一个逆而已。实践当中使用T_{cw}更加常见，而T_{wc}更为直观。如果把上面两式的p_c取成零向量，也就是相机坐标系中的原点，那么，此时的p_w就是相机原点在世界坐标系下的坐标：

$$\boldsymbol{p}_w = \boldsymbol{T}_{\mathrm{wc}}\boldsymbol{0} = \boldsymbol{t}_{\mathrm{wc}}. \tag{3.42}$$

我们发现这正是T_{wc}的平移部分。因此，可以从T_{wc}中直接看到相机在何处，这也是我们说T_{wc}更为直观的原因。因此，在可视化程序里，我们显示了T_{wc}而不是T_{cw}。

习题

1.验证旋转矩阵是正交矩阵。

2.*寻找罗德里格斯公式的推导过程并加以理解。

3.验证四元数旋转某个点后，结果是一个虚四元数（实部为零），所以仍然对应到一个三维空间点（式3.34）。

4.画表总结旋转矩阵、轴角、欧拉角、四元数的转换关系。

5.假设有一个大的Eigen矩阵，想把它的左上角3×3的块取出来，然后赋值为$I_{3\times3}$。请编程实现。

6.*一般线性方程$Ax=b$有哪几种做法？你能在Eigen中实现吗？

7.设有小萝卜一号和小萝卜二号位于世界坐标系中。小萝卜一号的

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

位姿为q_1=[0.35,0.2,0.3,0.1],t_2=[0.3,0.1,0.1]^T（q的第一项为实部。请把q归一化后再进行计算）。这里的q和t表达的是T_{cw}，也就是世界坐标系到相机坐标系的变换关系。小萝卜二号的位姿为q_2=[-0.5,0.4,-0.1,0.2],t=[-0.1,0.5,0.3]^T。现在，小萝卜一号看到某个点在自身的坐标系下坐标为p=[0.5,0,0.2]^T，求该向量在小萝卜二号坐标系下的坐标。请编程实现。

---

[1]正交矩阵即逆为自身转置的矩阵。

[2]官方主页：http://eigen.tuxfamily.org/index.php?title=Main_Page。

[3]感兴趣的读者请参见https://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula。

[4]https://en.wikipedia.org/wiki/Gimbal_lock。

书籍汇总下载 http://t.cn/RDIAj5D

第4讲　李群与李代数

主要目标

1.理解李群与李代数的概念，掌握SO(3),SE(3)与对应李代数的表示方式。

2.理解BCH近似的意义。

3.学会在李代数上的扰动模型。

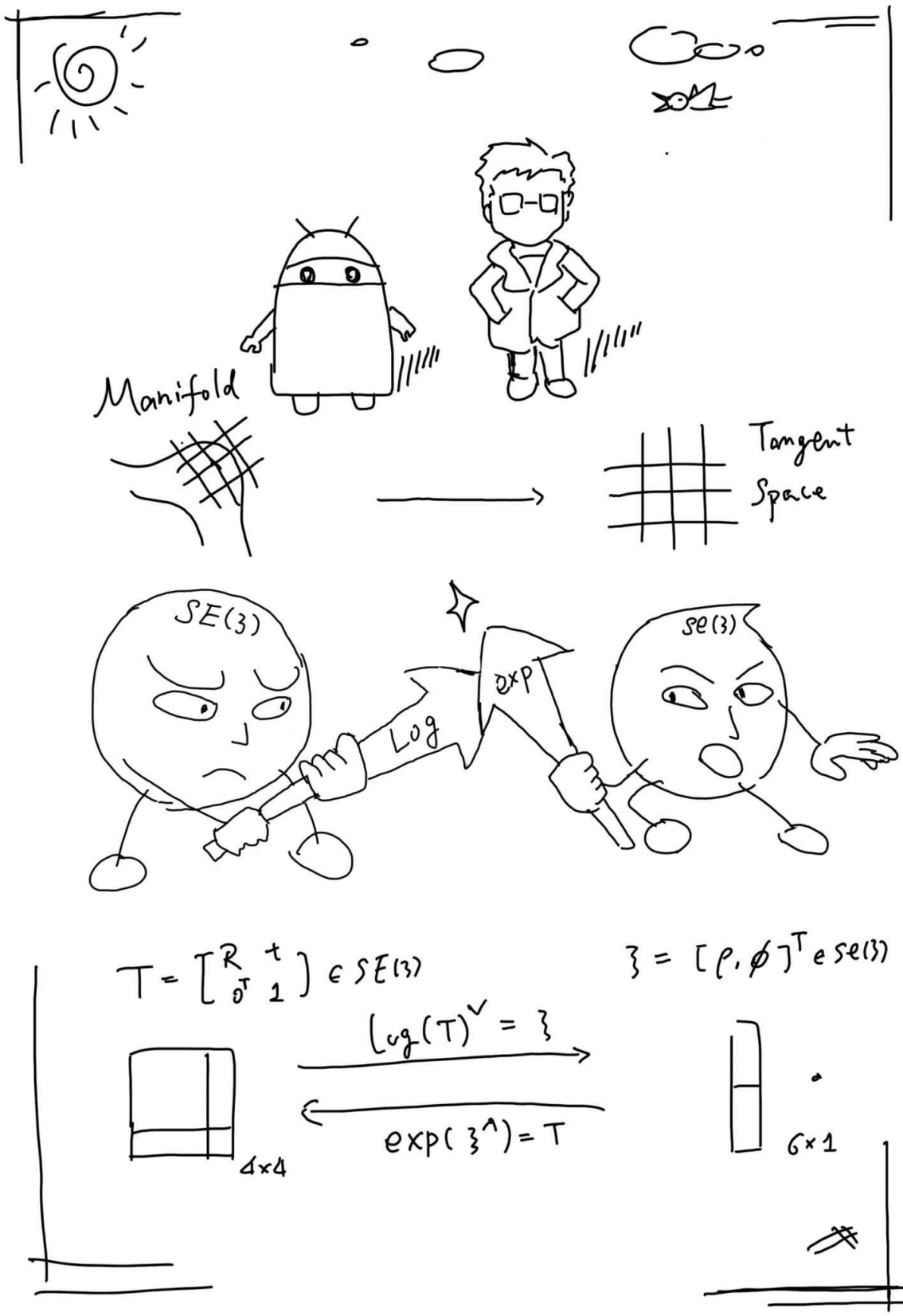
4.使用Sophus对李代数进行运算。

上一讲，我们介绍了三维世界中刚体运动的描述方式，包括旋转矩阵、旋转向量、欧拉角、四元数等若干种方式。我们重点介绍了旋转的表示，但是在SLAM中，除了表示之外，我们还要对它们进行估计和优化。因为在SLAM中位姿是未知的，而我们需要解决什么样的相机位姿最符合当前观测数据这样的问题。一种典型的方式是把它构建成一个优化问题，求解最优的R,t，使得误差最小化。

如前所言，旋转矩阵自身是带有约束的（正交且行列式为1）。它们作为优化变量时，会引入额外的约束，使优化变得困难。通过李群—李代数间的转换关系，我们希望把位姿估计变成无约束的优化问题，简化求解方式。考虑到读者可能还没有李群李代数的基本知识，我们将从最基本的知识开始讲起。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D



电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

4.1 李群与李代数基础

上一讲，我们介绍了旋转矩阵和变换矩阵的定义。当时，我们说三维旋转矩阵构成了特殊正交群SO(3)，而变换矩阵构成了特殊欧氏群SE(3)：

$$\mathrm{SO}(3) = \{\boldsymbol{R} \in \mathbb{R}^{3\times 3} | \boldsymbol{R}\boldsymbol{R}^{\mathrm{T}} = \boldsymbol{I}, det(\boldsymbol{R}) = 1\}. \tag{4.1}$$

$$\mathrm{SE}(3) = \left\{ \boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} | \boldsymbol{R} \in \mathrm{SO}(3), \boldsymbol{t} \in \mathbb{R}^3 \right\}. \tag{4.2}$$

不过，当时我们并未详细解释群的含义。细心的读者应该会注意到，旋转矩阵也好，变换矩阵也好，它们对加法是不封闭的。换句话说，对于任意两个旋转矩阵R_1,R_2，按照矩阵加法的定义，和不再是一个旋转矩阵：

$$\boldsymbol{R}_1 + \boldsymbol{R}_2 \notin \mathrm{SO}(3). \tag{4.3}$$

对于变换矩阵亦是如此。我们发现，这两种矩阵并没有良好定义的加法，相对地，它们只有一种较好的运算：乘法。SO(3)和SE(3)关于乘法是封闭的:

$$\boldsymbol{R}_1\boldsymbol{R}_2 \in \mathrm{SO}(3), \quad \boldsymbol{T}_1\boldsymbol{T}_2 \in \mathrm{SE}(3). \tag{4.4}$$

我们知道，乘法对应着旋转或变换的复合，两个旋转矩阵相乘表示做了两次旋转。对于这种只有一个运算的集合，我们称之为群。

4.1.1 群

群（Group）是一种集合加上一种运算的代数结构。我们把集合记作A，运算记作$\cdot$，那么群可以记作$G=(A,\cdot)$。群要求这个运算满足以下几个条件：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

1.封闭性：$\forall a_1, a_2 \in A, a_1 \cdot a_2 \in A$.

2.结合律：$\forall a_1, a_2, a_3 \in A, (a_1 \cdot a_2) \cdot a_3 = a_1 \cdot (a_2 \cdot a_3)$.

3.幺元：$\exists a_0 \in A, s.t. \forall a \in A, a_0 \cdot a = a \cdot a_0 = a$.

4.逆：$\forall a \in A, \exists a^{-1} \in A, s.t. a \cdot a^{-1} = a_0$.

读者可以记作“封结幺逆”[1]。我们可以验证，旋转矩阵集合和矩阵乘法构成群，同样变换矩阵和矩阵乘法也构成群（因此才能称它们为旋转矩阵群和变换矩阵群）。其他常见的群包括整数的加法(Z,+)，去掉0后的有理数的乘法（幺元为1）(Q\0,·)，等等。矩阵中常见的群有：

•一般线性群GL(n)指$n \times n$的可逆矩阵，它们对矩阵乘法成群。

•特殊正交群SO(n)也就是所谓的旋转矩阵群，其中SO(2)和SO(3)最为常见。

•特殊欧氏群SE(n)也就是前面提到的n维欧氏变换，如SE(2)和SE(3)。

群结构保证了在群上的运算具有良好的性质，而群论则是研究群的各种结构和性质的理论，但这里不多加介绍。感兴趣的读者可以参考任意一本近世代数教材。

李群是指具有连续（光滑）性质的群。像整数群Z那样离散的群没有连续性质，所以不是李群。而SO(n)和SE(n)在实数空间上是连续的。我们能够直观地想象一个刚体能够连续地在空间中运动，所以它们都是李群。由于SO(3)和SE(3)对于相机姿态估计尤其重要，所以我们主要讨论这两个李群。如果读者对李群的理论性质感兴趣，请参考文献[20]。

下面，我们先从较简单的SO(3)开始讨论，我们将发现每个李群都有对应的李代数。我们首先引出SO(3)上面的李代数so(3)。

4.1.2 李代数的引出

考虑任意旋转矩阵R，我们知道它满足：

$$\boldsymbol{R}\boldsymbol{R}^{\mathrm{T}} = \boldsymbol{I}. \tag{4.5}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

现在，我们说，R是某个相机的旋转，它会随时间连续地变化，即为时间的函数：$R(t)$。由于它仍是旋转矩阵，有

$$\boldsymbol{R}(t)\boldsymbol{R}(t)^T = \boldsymbol{I}.$$

在等式两边对时间求导，得到：

$$\dot{\boldsymbol{R}}(t)\boldsymbol{R}(t)^{\mathrm{T}} + \boldsymbol{R}(t)\dot{\boldsymbol{R}}(t)^{\mathrm{T}} = 0.$$

整理得：

$$\dot{\boldsymbol{R}}(t)\boldsymbol{R}(t)^{\mathrm{T}} = -\left(\dot{\boldsymbol{R}}(t)\boldsymbol{R}(t)^{\mathrm{T}}\right)^{\mathrm{T}}. \tag{4.6}$$

可以看出$\dot{R}(t)R(t)^{\mathrm{T}}$是一个反对称矩阵。回忆一下，我们在式（3.3）介绍叉积时，引入了$\wedge$符号，将一个向量变成了反对称矩阵。同理，对于任意反对称矩阵，我们亦能找到一个与之对应的向量。把这个运算用符号$\vee$表示：

$$\boldsymbol{a}^{\wedge} = \boldsymbol{A} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}, \quad \boldsymbol{A}^{\vee} = \boldsymbol{a}. \tag{4.7}$$

于是，由于$\dot{R}(t)R(t)^{\mathrm{T}}$是一个反对称矩阵，我们可以找到一个三维向量$\phi(t) \in \mathrm{R}^3$与之对应。于是有：

$$\dot{\boldsymbol{R}}(t)\boldsymbol{R}(t)^{\mathrm{T}} = \boldsymbol{\phi}(t)^{\wedge}.$$

等式两边右乘$R(t)$，由于R为正交阵，有：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$\dot{\boldsymbol{R}}(t) = \boldsymbol{\phi}(t)^{\wedge}\boldsymbol{R}(t) = \begin{bmatrix} 0 & -\phi_3 & \phi_2 \\ \phi_3 & 0 & -\phi_1 \\ -\phi_2 & \phi_1 & 0 \end{bmatrix} \boldsymbol{R}(t). \tag{4.8}$$

可以看到，每对旋转矩阵求一次导数，只需左乘一个$\boldsymbol{\phi}^{\wedge}(t)$矩阵即可。为方便讨论，我们设$t_0$=0，并设此时旋转矩阵为$\boldsymbol{R}(0)=\boldsymbol{I}$。按照导数定义，可以把$\boldsymbol{R}(t)$在0附近进行一阶泰勒展开：

$$\begin{aligned} \boldsymbol{R}(t) &\approx \boldsymbol{R}(t_0) + \dot{\boldsymbol{R}}(t_0)(t - t_0) \\ &= \boldsymbol{I} + \boldsymbol{\phi}(t_0)^{\wedge}(t). \end{aligned} \tag{4.9}$$

我们看到$\boldsymbol{\phi}$反映了$\boldsymbol{R}$的导数性质，故称它在SO(3)原点附近的正切空间(Tangent Space)上。同时在t_0附近，设$\boldsymbol{\phi}$保持为常数$\boldsymbol{\phi}(t_0)=\boldsymbol{\phi}_0$。那么根据式（4.8），有：

$$\dot{\boldsymbol{R}}(t) = \boldsymbol{\phi}(t_0)^{\wedge}\boldsymbol{R}(t) = \boldsymbol{\phi}_0^{\wedge}\boldsymbol{R}(t).$$

上式是一个关于$\boldsymbol{R}$的微分方程，而且知道初始值$\boldsymbol{R}(0)=\boldsymbol{I}$，解之，得：

$$\boldsymbol{R}(t) = \exp(\boldsymbol{\phi}_0^{\wedge} t). \tag{4.10}$$

读者可以验证上式对微分方程和初始值均成立。不过，由于做了一定的假设，所以它只在t=0附近有效。我们看到，旋转矩阵$\boldsymbol{R}$与另一个反对称矩阵$\boldsymbol{\phi}_0$通过指数关系发生了联系。也就是说，当我们知道某个时刻的$\boldsymbol{R}$时，存在一个向量$\boldsymbol{\phi}$，它们满足这个矩阵指数关系。但是矩阵的指数是什么呢？这里我们有两个问题需要澄清：

1.如果上式成立，那么给定某时刻的$\boldsymbol{R}$，我们就能求得一个$\boldsymbol{\phi}$，它描述了$\boldsymbol{R}$在局部的导数关系。与$\boldsymbol{R}$对应的$\boldsymbol{\phi}$有什么含义呢？后面会看到，$\boldsymbol{\phi}$正是对应到SO(3)上的李代数so(3)；

2.其次，矩阵指数$\exp(\boldsymbol{\phi}^{\wedge})$如何计算？事实上，这正是李群与李代数间的指数/对数映射。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

下面一一加以介绍。

4.1.3 李代数的定义

每个李群都有与之对应的李代数。李代数描述了李群的局部性质。通用的李代数的定义如下：

李代数由一个集合V，一个数域F和一个二元运算[,]组成。如果它们满足以下几条性质，则称(V,F,[,])为一个李代数，记作g。

1.封闭性$\forall X,Y\in \mathrm{V},[X,Y]\in \mathrm{V}$.

2.双线性$\forall X,Y,Z\in \mathrm{V},a,b\in \mathrm{F}$,有：

$$[a\boldsymbol{X}+b\boldsymbol{Y},\boldsymbol{Z}]=a[\boldsymbol{X},\boldsymbol{Z}]+b[\boldsymbol{Y},\boldsymbol{Z}],\quad [\boldsymbol{Z},a\boldsymbol{X}+b\boldsymbol{Y}]=a[\boldsymbol{Z},\boldsymbol{X}]+b[\boldsymbol{Z},\boldsymbol{Y}].$$

3.自反性[2]$\forall X\in \mathrm{V},[X,X]=0$.

4.雅可比等价$\forall X,Y,Z\in \mathrm{V},[X,[Y,Z]]+[Z,[X,Y]]+[Y,[Z,X]]=0$.

其中二元运算被称为李括号。从表面上来看，李代数所需要的性质还是挺多的。相比于群中的较为简单的二元运算，李括号表达了两个元素的差异。它不要求结合律，而要求元素和自己做李括号之后为零的性质。作为例子，三维向量R^3上定义的叉积×是一种李括号，因此g=$(\mathrm{R}^3,\mathrm{R},\times)$构成了一个李代数。读者可以尝试将叉积的性质代入到上面四条性质中。

4.1.4 李代数so(3)

之前提到的ϕ，事实上是一种李代数。SO(3)对应的李代数是定义在R^3上的向量，我们记作ϕ。根据前面的推导，每个ϕ都可以生成一个反对称矩阵：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$\boldsymbol{\Phi} = \boldsymbol{\phi}^\wedge = \begin{bmatrix} 0 & -\phi_3 & \phi_2 \\ \phi_3 & 0 & -\phi_1 \\ -\phi_2 & \phi_1 & 0 \end{bmatrix} \in \mathbb{R}^{3\times 3}. \tag{4.11}$$

在此定义下，两个向量$\boldsymbol{\phi}_1$,$\boldsymbol{\phi}_2$的李括号为

$$[\boldsymbol{\phi}_1, \boldsymbol{\phi}_2] = (\boldsymbol{\Phi}_1\boldsymbol{\Phi}_2 - \boldsymbol{\Phi}_2\boldsymbol{\Phi}_1)^\vee. \tag{4.12}$$

读者可以验证该定义下的李括号满足上面的几条性质。由于$\boldsymbol{\phi}$与反对称矩阵关系很紧密，在不引起歧义的情况下，就说so(3)的元素是三维向量或者三维反对称矩阵，不加区别：

$$\mathfrak{so}(3) = \left\{\boldsymbol{\phi} \in \mathbb{R}^3, \boldsymbol{\Phi} = \boldsymbol{\phi}^\wedge \in \mathbb{R}^{3\times 3}\right\}. \tag{4.13}$$

至此，我们已清楚了so(3)的内容。它们是一个由三维向量组成的集合，每个向量对应到一个反对称矩阵，可以表达旋转矩阵的导数。它与SO(3)的关系由指数映射给定：

$$\boldsymbol{R} = \exp(\boldsymbol{\phi}^\wedge). \tag{4.14}$$

指数映射会在稍后介绍。由于已经介绍了so(3)，我们顺带先来看SE(3)上对应的李代数。

4.1.5 李代数se(3)

对于SE(3)，它也有对应的李代数se(3)。为节省篇幅，这里就不介绍如何引出se(3)了。与so(3)相似，se(3)位于R^6空间中：

$$\mathfrak{se}(3) = \left\{\boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{\rho} \\ \boldsymbol{\phi} \end{bmatrix} \in \mathbb{R}^6, \boldsymbol{\rho} \in \mathbb{R}^3, \boldsymbol{\phi} \in \mathfrak{so}(3), \boldsymbol{\xi}^\wedge = \begin{bmatrix} \boldsymbol{\phi}^\wedge & \boldsymbol{\rho} \\ \mathbf{0}^{\mathrm{T}} & 0 \end{bmatrix} \in \mathbb{R}^{4\times 4}\right\}. \tag{4.15}$$

我们把每个se(3)元素记作$\boldsymbol{\xi}$，它是一个六维向量。前三维为平移

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

（但含义与变换矩阵中的平移不同，分析见后），记作ρ；后三维为旋转，记作ϕ，实质上是so(3)元素[3]。同时，我们拓展了$\wedge$符号的含义。在se(3)中，同样使用$\wedge$符号，将一个六维向量转换成四维矩阵，但这里不再表示反对称：

$$\boldsymbol{\xi}^{\wedge} = \begin{bmatrix} \boldsymbol{\phi}^{\wedge} & \boldsymbol{\rho} \\ \mathbf{0}^{\mathrm{T}} & 0 \end{bmatrix} \in \mathbb{R}^{4\times 4}. \tag{4.16}$$

我们仍使用$\wedge$和$\vee$符号来指代“从向量到矩阵”和“从矩阵到向量”的关系，以保持和so(3)上的一致性。读者可以简单地把se(3)理解成“由一个平移加上一个so(3)元素构成的向量”（尽管这里的ρ还不直接是平移）。同样，李代数se(3)亦有类似于so(3)的李括号：

$$[\boldsymbol{\xi}_1, \boldsymbol{\xi}_2] = \left(\boldsymbol{\xi}_1^{\wedge}\boldsymbol{\xi}_2^{\wedge} - \boldsymbol{\xi}_2^{\wedge}\boldsymbol{\xi}_1^{\wedge}\right)^{\vee}. \tag{4.17}$$

读者可以验证它是否满足李代数的定义（留作习题）。至此我们已经见过两种重要的李代数so(3)和se(3)了。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

4.2 指数与对数映射

4.2.1 SO(3)上的指数映射

现在来考虑第二个问题：exp($\phi^\wedge$)是如何计算的？它是一个矩阵的指数，在李群和李代数中，称为指数映射（Exponential Map）。同样，我们会先讨论so(3)的指数映射，再讨论se(3)的情形。

任意矩阵的指数映射可以写成一个泰勒展开，但是只有在收敛的情况下才会有结果，其结果仍是一个矩阵。

$$\exp(\boldsymbol{A}) = \sum_{n=0}^{\infty} \frac{1}{n!} \boldsymbol{A}^n. \tag{4.18}$$

同样地，对so(3)中任意元素ϕ，我们亦可按此方式定义它的指数映射：

$$\exp(\boldsymbol{\phi}^\wedge) = \sum_{n=0}^{\infty} \frac{1}{n!} (\boldsymbol{\phi}^\wedge)^n. \tag{4.19}$$

我们来仔细推导一下这个定义。由于ϕ是三维向量，我们可以定义它的模长和它的方向，分别记作θ和$\boldsymbol{a}$，于是有$\boldsymbol{\phi}=\theta\boldsymbol{a}$。这里$\boldsymbol{a}$是一个长度为1的方向向量。首先，对于$\boldsymbol{a}^\wedge$，有以下两条性质：

$$\boldsymbol{a}^\wedge \boldsymbol{a}^\wedge = \boldsymbol{a}\boldsymbol{a}^{\mathrm{T}} - \boldsymbol{I}, \tag{4.20}$$

以及

$$\boldsymbol{a}^\wedge \boldsymbol{a}^\wedge \boldsymbol{a}^\wedge = -\boldsymbol{a}^\wedge. \tag{4.21}$$

读者可以自行验证上述性质。它们提供了处理$\boldsymbol{a}^\wedge$高阶项的方法。利用这两个性质，我们可以把指数映射写成：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$
\begin{aligned}
\exp\left(\boldsymbol{\phi}^{\wedge}\right) = \exp\left(\theta \boldsymbol{a}^{\wedge}\right) &= \sum_{n=0}^{\infty} \frac{1}{n!} \left(\theta \boldsymbol{a}^{\wedge}\right)^n \\
&= \boldsymbol{I} + \theta \boldsymbol{a}^{\wedge} + \frac{1}{2!}\theta^2 \boldsymbol{a}^{\wedge}\boldsymbol{a}^{\wedge} + \frac{1}{3!}\theta^3 \boldsymbol{a}^{\wedge}\boldsymbol{a}^{\wedge}\boldsymbol{a}^{\wedge} + \frac{1}{4!}\theta^4 \left(\boldsymbol{a}^{\wedge}\right)^4 + \ldots \\
&= \boldsymbol{a}\boldsymbol{a}^{\mathrm{T}} - \boldsymbol{a}^{\wedge}\boldsymbol{a}^{\wedge} + \theta \boldsymbol{a}^{\wedge} + \frac{1}{2!}\theta^2 \boldsymbol{a}^{\wedge}\boldsymbol{a}^{\wedge} - \frac{1}{3!}\theta^3 \boldsymbol{a}^{\wedge} - \frac{1}{4!}\theta^4 \left(\boldsymbol{a}^{\wedge}\right)^2 + \ldots \\
&= \boldsymbol{a}\boldsymbol{a}^{\mathrm{T}} + \left(\theta - \frac{1}{3!}\theta^3 + \frac{1}{5!}\theta^5 - \ldots\right) \boldsymbol{a}^{\wedge} - \left(1 - \frac{1}{2!}\theta^2 + \frac{1}{4!}\theta^4 - \ldots\right) \boldsymbol{a}^{\wedge}\boldsymbol{a}^{\wedge} \\
&= \boldsymbol{a}^{\wedge}\boldsymbol{a}^{\wedge} + \boldsymbol{I} + \sin\theta \boldsymbol{a}^{\wedge} - \cos\theta \boldsymbol{a}^{\wedge}\boldsymbol{a}^{\wedge} \\
&= (1 - \cos\theta)\boldsymbol{a}^{\wedge}\boldsymbol{a}^{\wedge} + \boldsymbol{I} + \sin\theta \boldsymbol{a}^{\wedge} \\
&= \cos\theta \boldsymbol{I} + (1 - \cos\theta)\boldsymbol{a}\boldsymbol{a}^{\mathrm{T}} + \sin\theta \boldsymbol{a}^{\wedge}.
\end{aligned}
$$

最后得到一个似曾相识的式子：

$$
\exp(\theta \boldsymbol{a}^{\wedge}) = \cos\theta \boldsymbol{I} + (1 - \cos\theta)\boldsymbol{a}\boldsymbol{a}^{\mathrm{T}} + \sin\theta \boldsymbol{a}^{\wedge}. \tag{4.22}
$$

回想前一讲内容，它和罗德里格斯公式，即式（3.14）如出一辙。这表明，so(3)实际上就是由所谓的旋转向量组成的空间，而指数映射即罗德里格斯公式。通过它们，我们把so(3)中任意一个向量对应到了一个位于SO(3)中的旋转矩阵。反之，如果定义对数映射，也能把SO(3)中的元素对应到so(3)中：

$$
\boldsymbol{\phi} = \ln(\boldsymbol{R})^{\vee} = \left(\sum_{n=0}^{\infty} \frac{(-1)^n}{n+1} (\boldsymbol{R} - \boldsymbol{I})^{n+1} \right)^{\vee}. \tag{4.23}
$$

不过我们通常不按照泰勒展开去计算对数映射。在第3讲中，我们已经介绍过如何根据旋转矩阵计算对应的李代数，即使用式（3.16），利用迹的性质分别求解转角和转轴，采用那种方式更加省事一些。

现在，我们介绍了指数映射的计算方法。读者可能会问，指数映射性质如何呢？是否对于任意的$\boldsymbol{R}$都能找到一个唯一的$\boldsymbol{\phi}$？很遗憾，指数映射只是一个满射。这意味着每个SO(3)中的元素，都可以找到一个so(3)元素与之对应；但是可能存在多个so(3)中的元素，对应到同一个SO(3)。至少对于旋转角θ，我们知道多转360°和没有转是一样的——它具有周期性。但是，如果我们把旋转角度固定在$\pm\pi$之间，那么李群和李

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

代数元素是一一对应的。

SO(3)与so(3)的结论似乎在我们的意料之中。它和我们前面讲的旋转向量与旋转矩阵很相似，而指数映射即是罗德里格斯公式。旋转矩阵的导数可以由旋转向量指定，指导着如何在旋转矩阵中进行微积分运算。

4.2.2 SE(3)上的指数映射

下面介绍se(3)上的指数映射。为了节省篇幅，我们不再像so(3)那样详细推导指数映射。se(3)上的指数映射形式如下：

$$\exp\left(\boldsymbol{\xi}^{\wedge}\right) = \begin{bmatrix} \sum_{n=0}^{\infty} \frac{1}{n!}(\boldsymbol{\phi}^{\wedge})^n & \sum_{n=0}^{\infty} \frac{1}{(n+1)!}(\boldsymbol{\phi}^{\wedge})^n \boldsymbol{\rho} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix} \tag{4.24}$$

$$\triangleq \begin{bmatrix} \boldsymbol{R} & \boldsymbol{J}\boldsymbol{\rho} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix} = \boldsymbol{T}. \tag{4.25}$$

如果你有耐心，可以照着so(3)上的做法推导，把exp进行泰勒展开推导此式。从结果上看，ξ的指数映射左上角的R是我们熟知的SO(3)中的元素，与se(3)当中的旋转部分ϕ对应。而右上角的J则可整理为（设$\phi=\theta a$）：

$$\boldsymbol{J} = \frac{\sin\theta}{\theta}\boldsymbol{I} + \left(1 - \frac{\sin\theta}{\theta}\right)\boldsymbol{a}\boldsymbol{a}^{T} + \frac{1-\cos\theta}{\theta}\boldsymbol{a}^{\wedge}. \tag{4.26}$$

该式与罗德里格斯公式有些相似，但不完全一样。我们看到，平移部分经过指数映射之后，发生了一次以J为系数矩阵的线性变换。请读者重视这里的J，因为后面还要用到。

同样地，虽然我们也可以类比推得对数映射，不过根据变换矩阵T求so(3)上的对应向量也有更省事的方式：从左上角的R计算旋转向量，而右上角的t满足：

书籍汇总下载 http://t.cn/RDIAj5D

$$\boldsymbol{t} = \boldsymbol{J}\boldsymbol{\rho}. \tag{4.27}$$

由于J可以由ϕ得到，所以这里的ρ亦可由此线性方程解得。现在，我们已经弄清了李群、李代数的定义与相互的转换关系，总结如图4-1所示。如果读者有哪里不明白，可以翻回去几页看看公式推导。

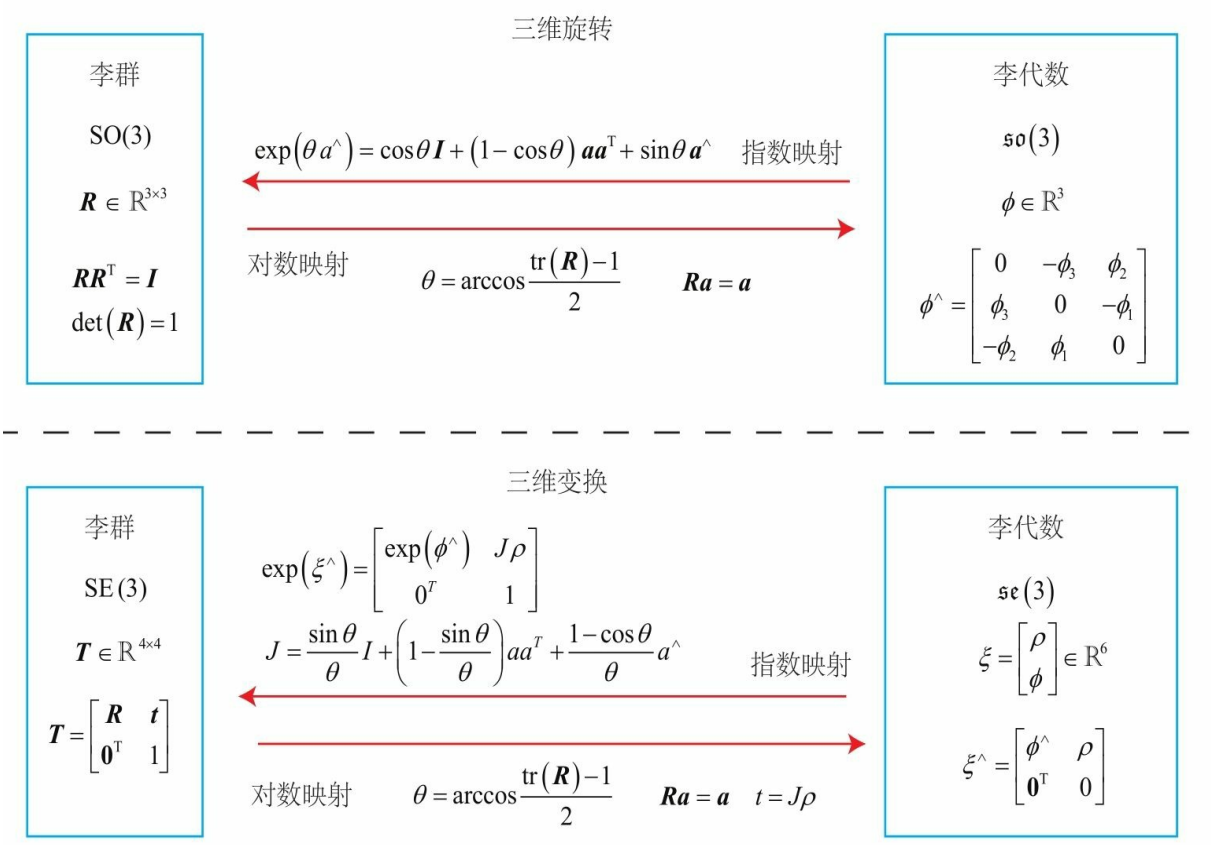


图4-1　SO(3),SE(3),so(3),se(3)的对应关系。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

4.3 李代数求导与扰动模型

4.3.1 BCH公式与近似形式

使用李代数的一大动机是进行优化，而在优化过程中导数是非常必要的信息（我们会在第6讲详细介绍）。下面来考虑一个问题。虽然我们已经清楚了SO(3)和SE(3)上的李群与李代数关系，但是，当在SO(3)中完成两个矩阵乘法时，李代数中so(3)上发生了什么改变呢？反过来说，当so(3)上做两个李代数的加法时，SO(3)上是否对应着两个矩阵的乘积？如果成立，相当于：

$$\exp\left(\boldsymbol{\phi}_1^\wedge\right)\exp\left(\boldsymbol{\phi}_2^\wedge\right)=\exp\left(\left(\boldsymbol{\phi}_1+\boldsymbol{\phi}_2\right)^\wedge\right).$$

如果$\boldsymbol{\phi}_1$,$\boldsymbol{\phi}_2$为标量，那显然该式成立；但此处我们计算的是矩阵的指数函数，而非标量的指数。换言之，我们在研究下式是否成立：

$$\ln\left(\exp\left(\boldsymbol{A}\right)\exp\left(\boldsymbol{B}\right)\right)=\boldsymbol{A}+\boldsymbol{B}\ ?$$

很遗憾，该式在矩阵时并不成立。

两个李代数指数映射乘积的完整形式，由Baker-Campbell-Hausdorff公式（BCH公式）[4]给出。由于其完整形式较复杂，我们只给出其展开式的前几项：

$$\ln\left(\exp\left(\boldsymbol{A}\right)\exp\left(\boldsymbol{B}\right)\right)=\boldsymbol{A}+\boldsymbol{B}+\frac{1}{2}\left[\boldsymbol{A},\boldsymbol{B}\right]+\frac{1}{12}\left[\boldsymbol{A},\left[\boldsymbol{A},\boldsymbol{B}\right]\right]-\frac{1}{12}\left[\boldsymbol{B},\left[\boldsymbol{A},\boldsymbol{B}\right]\right]+\cdots \tag{4.28}$$

其中[]为李括号。BCH公式告诉我们，当处理两个矩阵指数之积时，它们会产生一些由李括号组成的余项。特别地，考虑SO(3)上的李代数$\ln\left(\exp\left(\boldsymbol{\phi}_1^\wedge\right)\exp\left(\boldsymbol{\phi}_2^\wedge\right)\right)^\vee$，当$\boldsymbol{\phi}_1$或$\boldsymbol{\phi}_2$为小量时，小量二次以上的项都可以被忽略掉。此时，BCH拥有线性近似表达：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$\ln\left(\exp\left(\boldsymbol{\phi}_1^{\wedge}\right)\exp\left(\boldsymbol{\phi}_2^{\wedge}\right)\right)^{\vee} \approx \begin{cases} \boldsymbol{J}_l(\boldsymbol{\phi}_2)^{-1}\boldsymbol{\phi}_1 + \boldsymbol{\phi}_2 & \text{当}\boldsymbol{\phi}_1\text{为小量,} \\ \boldsymbol{J}_r(\boldsymbol{\phi}_1)^{-1}\boldsymbol{\phi}_2 + \boldsymbol{\phi}_1 & \text{当}\boldsymbol{\phi}_2\text{为小量.} \end{cases} \tag{4.29}$$

以第一个近似为例。该式告诉我们，当对一个旋转矩阵R_2（李代数为$\boldsymbol{\phi}_2$）左乘一个微小旋转矩阵R_1（李代数为$\boldsymbol{\phi}_1$）时，可以近似地看作，在原有的李代数$\boldsymbol{\phi}_2$上加上了一项$\boldsymbol{J}_l(\boldsymbol{\phi}_2)^{-1}\boldsymbol{\phi}_1$。同理，第二个近似描述了右乘一个微小位移的情况。于是，李代数在BCH近似下，分成了左乘近似和右乘近似两种，在使用时我们须注意使用的是左乘模型还是右乘模型。

本书以左乘为例。左乘BCH近似雅可比$\boldsymbol{J}_l$事实上就是式（4.26）的内容：

$$\boldsymbol{J}_l = \boldsymbol{J} = \frac{\sin\theta}{\theta}\boldsymbol{I} + \left(1 - \frac{\sin\theta}{\theta}\right)\boldsymbol{a}\boldsymbol{a}^{\mathrm{T}} + \frac{1-\cos\theta}{\theta}\boldsymbol{a}^{\wedge}. \tag{4.30}$$

它的逆为

$$\boldsymbol{J}_l^{-1} = \frac{\theta}{2}\cot\frac{\theta}{2}\boldsymbol{I} + \left(1 - \frac{\theta}{2}\cot\frac{\theta}{2}\right)\boldsymbol{a}\boldsymbol{a}^{\mathrm{T}} - \frac{\theta}{2}\boldsymbol{a}^{\wedge}. \tag{4.31}$$

而右乘雅可比仅需要对自变量取负号即可：

$$\boldsymbol{J}_r(\boldsymbol{\phi}) = \boldsymbol{J}_l(-\boldsymbol{\phi}). \tag{4.32}$$

这样，我们就可以谈论李群乘法与李代数加法的关系了。为了方便读者理解，我们重新叙述一下BCH近似的意义。

假定对某个旋转R，对应的李代数为$\boldsymbol{\phi}$。我们给它左乘一个微小旋转，记作ΔR，对应的李代数为$\Delta\boldsymbol{\phi}$。那么，在李群上，得到的结果就是$\Delta R \cdot R$，而在李代数上，根据BCH近似，为$\boldsymbol{J}_l^{-1}(\boldsymbol{\phi})\Delta\boldsymbol{\phi} + \boldsymbol{\phi}$。合并起来，可以简单地写成：

$$\exp\left(\Delta\boldsymbol{\phi}^{\wedge}\right)\exp\left(\boldsymbol{\phi}^{\wedge}\right) = \exp\left(\left(\boldsymbol{\phi} + \boldsymbol{J}_l^{-1}\left(\boldsymbol{\phi}\right)\Delta\boldsymbol{\phi}\right)^{\wedge}\right). \tag{4.33}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

反之，如果我们在李代数上进行加法，让一个ϕ加上$\Delta\phi$，那么可以近似为李群上带左右雅可比的乘法：

$$\exp\left((\phi+\Delta\phi)^\wedge\right)=\exp\left((\boldsymbol{J}_l\Delta\phi)^\wedge\right)\exp\left(\phi^\wedge\right)=\exp\left(\phi^\wedge\right)\exp\left((\boldsymbol{J}_r\Delta\phi)^\wedge\right). \tag{4.34}$$

这就为之后李代数上做微积分提供了理论基础。同样地，对于SE(3)，亦有类似的BCH近似公式：

$$\exp\left(\Delta\boldsymbol{\xi}^\wedge\right)\exp\left(\boldsymbol{\xi}^\wedge\right)\approx\exp\left(\left(\mathcal{J}_l^{-1}\Delta\boldsymbol{\xi}+\boldsymbol{\xi}\right)^\wedge\right), \tag{4.35}$$

$$\exp\left(\boldsymbol{\xi}^\wedge\right)\exp\left(\Delta\boldsymbol{\xi}^\wedge\right)\approx\exp\left(\left(\mathcal{J}_r^{-1}\Delta\boldsymbol{\xi}+\boldsymbol{\xi}\right)^\wedge\right). \tag{4.36}$$

这里J_l形式比较复杂，它是一个6×6的矩阵，读者可以参考文献[6]中式(7.82)和(7.83)的内容。由于我们在计算中不用到该雅可比，故这里略去它的实际形式。

4.3.2 SO(3)李代数上的求导

下面来讨论一个带有李代数的函数，如何关于该李代数求导的问题。该问题有很强的实际背景。在SLAM中，我们要估计一个相机的位置和姿态，该位姿是由SO(3)上的旋转矩阵或SE(3)上的变换矩阵描述的。不妨设某个时刻小萝卜的位姿为T。它观察到了一个世界坐标位于p的点，产生了一个观测数据z。那么，由坐标变换关系知：

$$\boldsymbol{z}=\boldsymbol{T}\boldsymbol{p}+\boldsymbol{w}. \tag{4.37}$$

然而，由于观测噪声w的存在，z往往不可能精确地满足$z=Tp$的关系。所以，我们通常会计算理想的观测与实际数据的误差：

$$\boldsymbol{e}=\boldsymbol{z}-\boldsymbol{T}\boldsymbol{p}. \tag{4.38}$$

假设一共有N个这样的路标点和观测，于是就有N个上式。那么，对小萝卜的位姿估计，相当于是寻找一个最优的T，使得整体误差最小化：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$\min_{\boldsymbol{T}} J(\boldsymbol{T}) = \sum_{i=1}^{N} \|\boldsymbol{z}_i - \boldsymbol{T}\boldsymbol{p}_i\|_2^2 . \tag{4.39}$$

求解此问题，需要计算目标函数J关于变换矩阵T的导数。我们把具体的算法留到后面再讲。这里重点要说的是，我们经常会构建与位姿有关的函数，然后讨论该函数关于位姿的导数，以调整当前的估计值。然而，SO(3),SE(3)上并没有良好定义的加法，它们只是群。如果我们把T当成一个普通矩阵来处理优化，那就必须对它加以约束。而从李代数角度来说，由于李代数由向量组成，具有良好的加法运算。因此，使用李代数解决求导问题的思路分为两种：

1.用李代数表示姿态，然后根据李代数加法来对李代数求导。

2.对李群左乘或右乘微小扰动，然后对该扰动求导，称为左扰动和右扰动模型。

第一种方式对应到李代数的求导模型，而第二种则对应到扰动模型。下面来讨论这两种思路的异同。

4.3.3 李代数求导

首先，考虑SO(3)上的情况。假设我们对一个空间点p进行了旋转，得到了Rp。现在，要计算旋转之后点的坐标相对于旋转的导数，我们不严谨地记为[5]：

$$\frac{\partial (\boldsymbol{R}\boldsymbol{p})}{\partial \boldsymbol{R}}.$$

由于SO(3)没有加法，所以该导数无法按照导数的定义进行计算。设R对应的李代数为ϕ，我们转而计算：

$$\frac{\partial (\exp(\boldsymbol{\phi}^{\wedge})\boldsymbol{p})}{\partial \boldsymbol{\phi}}.$$

按照导数的定义，有：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$
\begin{aligned}
\frac{\partial\left(\exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p}\right)}{\partial\boldsymbol{\phi}} &= \lim_{\delta\boldsymbol{\phi}\to 0}\frac{\exp\left(\left(\boldsymbol{\phi}+\delta\boldsymbol{\phi}\right)^{\wedge}\right)\boldsymbol{p}-\exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p}}{\delta\boldsymbol{\phi}} \\
&= \lim_{\delta\boldsymbol{\phi}\to 0}\frac{\exp\left(\left(\boldsymbol{J}_l\delta\boldsymbol{\phi}\right)^{\wedge}\right)\exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p}-\exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p}}{\delta\boldsymbol{\phi}} \\
&\approx \lim_{\delta\boldsymbol{\phi}\to 0}\frac{\left(\boldsymbol{I}+\left(\boldsymbol{J}_l\delta\boldsymbol{\phi}\right)^{\wedge}\right)\exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p}-\exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p}}{\delta\boldsymbol{\phi}} \\
&= \lim_{\delta\boldsymbol{\phi}\to 0}\frac{\left(\boldsymbol{J}_l\delta\boldsymbol{\phi}\right)^{\wedge}\exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p}}{\delta\boldsymbol{\phi}} \\
&= \lim_{\delta\boldsymbol{\phi}\to 0}\frac{-\left(\exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p}\right)^{\wedge}\boldsymbol{J}_l\delta\boldsymbol{\phi}}{\delta\boldsymbol{\phi}} = -\left(\boldsymbol{R}\boldsymbol{p}\right)^{\wedge}\boldsymbol{J}_l.
\end{aligned}
$$

第2行的近似为BCH线性近似，第3行为泰勒展开舍去高阶项后的近似，第4行至第5行将反对称符号看作叉积，交换之后变号。于是，我们推导出了旋转后的点相对于李代数的导数：

$$
\frac{\partial\left(\boldsymbol{R}\boldsymbol{p}\right)}{\partial\boldsymbol{\phi}} = \left(-\boldsymbol{R}\boldsymbol{p}\right)^{\wedge}\boldsymbol{J}_l. \tag{4.40}
$$

不过，由于这里仍然含有形式比较复杂的J_l，我们不太希望计算它。而下面要讲的扰动模型则提供了更简单的导数计算方式。

4.3.4 扰动模型（左乘）

另一种求导方式，是对R进行一次扰动ΔR。这个扰动可以乘在左边也可以乘在右边，最后结果会有一点儿微小的差异，我们以左扰动为例。设左扰动ΔR对应的李代数为φ。然后，对φ求导，即：

$$
\frac{\partial\left(\boldsymbol{R}\boldsymbol{p}\right)}{\partial\boldsymbol{\varphi}} = \lim_{\boldsymbol{\varphi}\to 0}\frac{\exp\left(\boldsymbol{\varphi}^{\wedge}\right)\exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p}-\exp\left(\boldsymbol{\phi}^{\wedge}\right)\boldsymbol{p}}{\boldsymbol{\varphi}}. \tag{4.41}
$$

该式的求导比上面更为简单：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$\begin{aligned}\frac{\partial(\boldsymbol{Rp})}{\partial\boldsymbol{\varphi}} &= \lim_{\boldsymbol{\varphi}\to 0}\frac{\exp(\boldsymbol{\varphi}^\wedge)\exp(\boldsymbol{\phi}^\wedge)\boldsymbol{p}-\exp(\boldsymbol{\phi}^\wedge)\boldsymbol{p}}{\boldsymbol{\varphi}}\\ &\approx \lim_{\boldsymbol{\varphi}\to 0}\frac{(1+\boldsymbol{\varphi}^\wedge)\exp(\boldsymbol{\phi}^\wedge)\boldsymbol{p}-\exp(\boldsymbol{\phi}^\wedge)\boldsymbol{p}}{\boldsymbol{\varphi}}\\ &= \lim_{\boldsymbol{\varphi}\to 0}\frac{\boldsymbol{\varphi}^\wedge\boldsymbol{Rp}}{\boldsymbol{\varphi}} = \lim_{\boldsymbol{\varphi}\to 0}\frac{-(\boldsymbol{Rp})^\wedge\boldsymbol{\varphi}}{\boldsymbol{\varphi}} = -(\boldsymbol{Rp})^\wedge.\end{aligned}$$

可见，相比于直接对李代数求导，省去了一个雅可比J_l的计算。这使得扰动模型更为实用。请读者务必理解这里的求导运算，这在位姿估计当中具有重要的意义。

4.3.5 SE(3)上的李代数求导

最后，我们给出SE(3)上的扰动模型，而直接李代数上的求导就不再介绍了。假设某空间点p经过一次变换T（对应李代数为ξ），得到Tp[6]。现在，给T左乘一个扰动$\Delta T=\exp(\delta\xi^\wedge)$，我们设扰动项的李代数为$\delta\xi=[\delta\rho,\delta\phi]^{\mathrm{T}}$，那么：

$$\begin{aligned}\frac{\partial(\boldsymbol{Tp})}{\partial\delta\boldsymbol{\xi}} &= \lim_{\delta\boldsymbol{\xi}\to\mathbf{0}}\frac{\exp(\delta\boldsymbol{\xi}^\wedge)\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{p}-\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{p}}{\delta\boldsymbol{\xi}}\\ &\approx \lim_{\delta\boldsymbol{\xi}\to\mathbf{0}}\frac{(\boldsymbol{I}+\delta\boldsymbol{\xi}^\wedge)\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{p}-\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{p}}{\delta\boldsymbol{\xi}}\\ &= \lim_{\delta\boldsymbol{\xi}\to\mathbf{0}}\frac{\delta\boldsymbol{\xi}^\wedge\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{p}}{\delta\boldsymbol{\xi}}\\ &= \lim_{\delta\boldsymbol{\xi}\to\mathbf{0}}\frac{\begin{bmatrix}\delta\boldsymbol{\phi}^\wedge & \delta\boldsymbol{\rho}\\ \mathbf{0}^T & 0\end{bmatrix}\begin{bmatrix}\boldsymbol{Rp}+\boldsymbol{t}\\ 1\end{bmatrix}}{\delta\boldsymbol{\xi}}\\ &= \lim_{\delta\boldsymbol{\xi}\to\mathbf{0}}\frac{\begin{bmatrix}\delta\boldsymbol{\phi}^\wedge(\boldsymbol{Rp}+\boldsymbol{t})+\delta\boldsymbol{\rho}\\ 0\end{bmatrix}}{\delta\boldsymbol{\xi}} = \begin{bmatrix}\boldsymbol{I} & -(\boldsymbol{Rp}+\boldsymbol{t})^\wedge\\ \mathbf{0}^T & \mathbf{0}^T\end{bmatrix}\triangleq(\boldsymbol{Tp})^\odot.\end{aligned}$$

我们把最后的结果定义成一个算符$\odot$[7]，它把一个齐次坐标的空间

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

点变换成一个4×6的矩阵。

至此，我们已经介绍了李群李代数上的微分运算。之后的章节中，我们将应用这些知识去解决实际问题。关于李群李代数的某些重要数学性质，我们作为习题留给读者。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

4.4 实践：Sophus

我们已经介绍了李代数的入门知识，现在是通过实践演练巩固一下所学知识的机会了。我们来讨论如何在程序中操作李代数。在第3讲中，我们看到Eigen提供了几何模块，但没有提供李代数的支持。一个较好的李代数库是Strasdat维护的Sophus库[8]。Sophus库支持本章主要讨论的SO(3)和SE(3)，此外还含有二维运动SO(2),SE(2)以及相似变换Sim(3)的内容。它是直接在Eigen基础上开发的，我们不需要安装额外的依赖库。读者可以直接从GitHub上获取Sophus[9]，或者，在本书的代码目录slambook/3rdparty下也提供了Sophus源代码。由于历史原因，Sophus早期版本只提供了双精度的李群/李代数类。后续版本改写成了模板类。模板类的Sophus中可以使用不同精度的李群/李代数，但同时增加了使用难度。本书使用非模板类的Sophus库。如果读者准备使用GitHub上的Sophus，请确保使用的是非模板类的版本。你可以输入以下命令获得非模板类的Sophus：

```
git clone https://github.com/strasdat/Sophus.git
cd Sophus
git checkout a621ff
```

本书的3rdparty中提供的Sophus也是非模板版本。Sophus本身亦是一个cmake工程。想必你已经了解如何编译cmake工程了，这里不再赘述。Sophus库只需编译即可，无须安装。

下面来演示一下Sophus库中的SO(3)和SE(3)运算：

slambook/ch4/useSophus/useSophus.cpp

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```cpp
#include <iostream>
#include <cmath>
using namespace std;

#include <Eigen/Core>
#include <Eigen/Geometry>

#include "sophus/so3.h"
#include "sophus/se3.h"

int main( int argc, char** argv )
{
    // 沿 Z 轴转 90 度的旋转矩阵
    Eigen::Matrix3d R = Eigen::AngleAxisd(M_PI/2, Eigen::Vector3d(0,0,1)).
    toRotationMatrix();

    Sophus::SO3 SO3_R(R);               // Sophus::SO(3)可以直接从旋转矩阵构造
    Sophus::SO3 SO3_v( 0, 0, M_PI/2 );  // 亦可从旋转向量构造
    Eigen::Quaterniond q(R);            // 或者四元数
    Sophus::SO3 SO3_q( q );
    // 上述表达方式都是等价的
    // 输出 SO(3) 时，以 so(3) 形式输出
    cout<<"SO(3) from matrix: "<<SO3_R<<endl;
    cout<<"SO(3) from vector: "<<SO3_v<<endl;
    cout<<"SO(3) from quaternion :"<<SO3_q<<endl;

    // 使用对数映射获得它的李代数
    Eigen::Vector3d so3 = SO3_R.log();
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
    cout<<"so3 = "<<so3.transpose()<<endl;
    // hat 为向量到反对称矩阵
    cout<<"so3 hat="<<Sophus::SO3::hat(so3)<<endl;
    // 相对地， vee 为反对称到向量
    cout<<"so3 hat vee= "<<Sophus::SO3::vee( Sophus::SO3::hat(so3) ).transpose()<<
    endl;

    // 增量扰动模型的更新
    Eigen::Vector3d update_so3(1e-4, 0, 0); // 假设更新量有这么多
    Sophus::SO3 SO3_updated = Sophus::SO3::exp(update_so3)*SO3_R; // 左乘更新
    cout<<"SO3 updated = "<<SO3_updated<<endl;

    /******************** 萌萌的分割线 ****************************/
    cout<<"************ 我是分割线 *************"<<endl;
    // 对 SE(3) 操作大同小异
    Eigen::Vector3d t(1,0,0);           // 沿 X 轴平移1
    Sophus::SE3 SE3_Rt(R, t);           // 从 R,t 构造SE(3)
    Sophus::SE3 SE3_qt(q,t);            // 从 q,t 构造SE(3)
    cout<<"SE3 from R,t= "<<endl<<SE3_Rt<<endl;
    cout<<"SE3 from q,t= "<<endl<<SE3_qt<<endl;
    // 李代数 se(3) 是一个六维向量，方便起见先 typedef 一下
    typedef Eigen::Matrix<double,6,1> Vector6d;
    Vector6d se3 = SE3_Rt.log();
    cout<<"se3 = "<<se3.transpose()<<endl;
    // 观察输出，会发现在 Sophus 中， se(3) 平移在前，旋转在后。与我们的书中是一致
    的。
    // 同样地，有 hat 和 vee 两个运算符
    cout<<"se3 hat = "<<endl<<Sophus::SE3::hat(se3)<<endl;
    cout<<"se3 hat vee = "<<Sophus::SE3::vee( Sophus::SE3::hat(se3) ).transpose()<<
    endl;

    // 最后，演示一下更新
    Vector6d update_se3; //更新量
    update_se3.setZero();
    update_se3(0,0) = 1e-4d;
    Sophus::SE3 SE3_updated = Sophus::SE3::exp(update_se3)*SE3_Rt;
    cout<<"SE3 updated = "<<endl<<SE3_updated.matrix()<<endl;

    return 0;
}
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

该演示程序分为两部分。前半部分介绍SO(3)上的操作，后半部分则为SE(3)。我们演示了如何构造SO(3),SE(3)对象，对它们进行指数、对数映射，以及当知道更新量后，如何对李群元素进行更新。如果读者切实理解了本讲内容，那么这个程序对你来说应该没有什么难度。为了编译它，请在CMakeLists.txt里添加以下几行：

slambook/ch4/useSophus/CMakeLists.txt

```
# 为使用 sophus，需要使用 find_package 命令找到它
find_package( Sophus REQUIRED )
include_directories( ${Sophus_INCLUDE_DIRS} )

add_executable( useSophus useSophus.cpp )
target_link_libraries( useSophus ${Sophus_LIBRARIES} )
```

fi nd_package命令是cmake提供的寻找某个库的头文件与库文件的指令。如果cmake能够找到它，就会提供头文件和库文件所在的目录的变量。在Sophus这个例子中，就是Sophus_INCLUDE_DIRS和Sophus_LIBRARIES这两个变量。根据它们，我们就能将Sophus库引入自己的cmake工程了。请读者自行查看此程序的输出信息，它与我们之前的推导是一致的。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

4.5 *相似变换群与李代数

最后，我们要提一下在单目视觉中使用的相似变换群Sim(3)，以及对应的李代数sim(3)。如果你只对双目SLAM或RGB-D SLAM感兴趣，可以跳过本节。

我们已经介绍过单目的尺度不确定性。如果在单目SLAM中使用SE(3)表示位姿，那么由于尺度不确定性与尺度漂移，整个SLAM过程中的尺度会发生变化，这在SE(3)中未能体现出来。因此，在单目情况下我们一般会显式地把尺度因子表达出来。用数学语言来说，对于位于空间的点p，在相机坐标系下要经过一个相似变换，而非欧氏变换：

$$\boldsymbol{p}' = \begin{bmatrix} s\boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix} \boldsymbol{p} = s\boldsymbol{R}\boldsymbol{p} + \boldsymbol{t}. \tag{4.42}$$

在相似变换中，我们把尺度s表达了出来。它同时作用在p的3个坐标之上，对p进行了一次缩放。与SO(3)、SE(3)相似，相似变换亦对矩阵乘法构成群，称为相似变换群Sim(3)：

$$\mathrm{Sim}(3) = \left\{ \boldsymbol{S} = \begin{bmatrix} s\boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^{\mathrm{T}} & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} \right\}. \tag{4.43}$$

同样地，Sim(3)也有对应的李代数、指数映射、对数映射等。李代数sim(3)元素是一个7维向量ζ。它的前6维与se(3)相同，最后多了一项σ。

$$\mathfrak{sim}(3) = \left\{ \boldsymbol{\zeta} | \boldsymbol{\zeta} = \begin{bmatrix} \boldsymbol{\rho} \\ \boldsymbol{\phi} \\ \sigma \end{bmatrix} \in \mathbb{R}^7, \boldsymbol{\zeta}^\wedge = \begin{bmatrix} \sigma \boldsymbol{I} + \boldsymbol{\phi}^\wedge & \boldsymbol{\rho} \\ \boldsymbol{0}^T & 0 \end{bmatrix} \in \mathbb{R}^{4\times 4} \right\}. \tag{4.44}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

它比se(3)多了一项σ。关联Sim(3)和sim(3)的仍是指数映射和对数映射。指数映射为

$$\exp\left(\boldsymbol{\zeta}^{\wedge}\right)=\begin{bmatrix} e^{\sigma}\exp\left(\boldsymbol{\phi}^{\wedge}\right) & \boldsymbol{J}_{s}\boldsymbol{\rho} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix}. \tag{4.45}$$

其中J_s形式为

$$\begin{aligned}\boldsymbol{J}_{s}=&\frac{e^{\sigma}-1}{\sigma}\boldsymbol{I}+\frac{\sigma e^{\sigma}\sin\theta+\left(1-e^{\sigma}\cos\theta\right)\theta}{\sigma^{2}+\theta^{2}}\boldsymbol{a}^{\wedge}\\&+\left(\frac{e^{\sigma}-1}{\sigma}-\frac{\left(e^{\sigma}\cos\theta-1\right)\sigma+\left(e^{\sigma}\sin\theta\right)\theta}{\sigma^{2}+\theta^{2}}\right)\boldsymbol{a}^{\wedge}\boldsymbol{a}^{\wedge}.\end{aligned}$$

通过指数映射，我们能够找到李代数与李群的关系。对于李代数元素ζ，它与李群的对应关系为

$$s=e^{\sigma},\ \boldsymbol{R}=\exp(\boldsymbol{\phi}^{\wedge}),\ \boldsymbol{t}=\boldsymbol{J}_{s}\boldsymbol{\rho}. \tag{4.46}$$

旋转部分和SO(3)是一致的。平移部分，在se(3)中需要乘一个雅可比J，而相似变换的雅可比更复杂一些。对于尺度因子，可以看到李群中的s即为李代数中σ的指数函数。

Sim(3)的BCH近似与SE(3)是类似的。我们可以讨论一个点p经过相似变换Sp后，相对于S的导数。同样地，存在微分模型和扰动模型两种方式，而扰动模型较为简单。我们省略推导过程，直接给出扰动模型的结果。设给予Sp左侧一个小扰动$\exp(\zeta^{\wedge})$，并求Sp对于扰动的导数。因为Sp是4维的齐次坐标，ζ是7维向量，该导数应该是4×7的雅可比。为了方便起见，记Sp的前3维组成向量q，那么：

$$\frac{\partial \boldsymbol{S}\boldsymbol{p}}{\partial\boldsymbol{\zeta}}=\begin{bmatrix} \boldsymbol{I} & -\boldsymbol{q}^{\wedge} & \boldsymbol{q} \\ \mathbf{0}^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} & 0 \end{bmatrix}. \tag{4.47}$$

关于Sim(3)，我们就介绍到这里。更详细的关于Sim(3)的资料，建议读者参见文献[21]。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

4.6 小结

本讲引入了李群SO(3)和SE(3)，以及它们对应的李代数so(3)和se(3)。我们介绍了位姿在它们上面的表达和转换，然后通过BCH的线性近似，我们可以对位姿进行求导和扰动了。这给之后讲解位姿的优化打下了理论基础，因为我们需要经常地对某一个位姿的估计值进行调整，使它对应的误差减小。只有在弄清楚如何对位姿进行调整和更新之后，我们才能继续下一步的内容。

可能本讲的内容比较偏理论化，毕竟它不像计算机视觉那样经常有好看的图片可以展示。相比于讲解李群李代数的数学教科书，由于我们只关心实用的内容，所以讲的内容非常精简，速度相对快了一些。请读者务必理解本章内容，它是解决后续许多问题的基础，特别是位姿估计部分。

习题

1.验证SO(3)、SE(3)和Sim(3)关于乘法成群。

2.验证($\mathbb{R}^3$,R,×)构成李代数。

3.验证so(3)和se(3)满足李代数要求的性质。

4.验证性质（4.20）和（4.21）。

5.证明：

$$\boldsymbol{R}\boldsymbol{p}^{\wedge}\boldsymbol{R}^{\mathrm{T}} = (\boldsymbol{R}\boldsymbol{p})^{\wedge}.$$

6.证明：

$$\boldsymbol{R}\exp(\boldsymbol{p}^{\wedge})\boldsymbol{R}^{\mathrm{T}} = \exp((\boldsymbol{R}\boldsymbol{p})^{\wedge}).$$

该式称为SO(3)上的伴随性质。同样地，在SE(3)上亦有伴随性质：

$$\boldsymbol{T}\exp(\boldsymbol{\xi}^{\wedge})\boldsymbol{T}^{-1} = \exp\left((\mathrm{Ad}(\boldsymbol{T})\boldsymbol{\xi})^{\wedge}\right), \tag{4.48}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

其中：

$$\mathrm{Ad}(\boldsymbol{T}) = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t}^{\wedge}\boldsymbol{R} \\ \boldsymbol{0} & \boldsymbol{R} \end{bmatrix}. \tag{4.49}$$

7.仿照左扰动的推导，推导SO(3)和SE(3)在右扰动下的导数。

8.搜索cmake的 fi nd_package指令是如何运作的。它有哪些可选的参数？为了让cmake找到某个库，需要哪些先决条件？

---

[1]谐音“凤姐咬你”。

[2]自反性是指自己与自己的运算为零。

[3]请注意有些地方把旋转放前面，平移放后面，也是可行的。

[4]参见https://en.wikipedia.org/wiki/Baker-Campbell-Hausdorff_formula。

[5]请注意这里并不能按照矩阵微分来定义导数，这只是一个记号。同时，在后文极限运算中，严谨的写法需要对分母进行转置，才能得到现在的结果。这里为方便书写故省去。

[6]请注意为了使乘法成立，p必须使用齐次坐标。

[7]我会读作“咚”，像一个石子掉在井里的声音。

[8]最早提出李代数的是Sophus Lie，这个库就以他的名字命名了。

[9]https://github.com/strasdat/Sophus。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

第5讲　相机与图像

主要目标

1.理解针孔相机的模型、内参与径向畸变参数。

2.理解一个空间点是如何投影到相机成像平面的。

3.掌握OpenCV的图像存储与表达方式。

4.学会基本的摄像头标定方法。

前面两讲中，我们介绍了“机器人如何表示自身位姿”的问题，部分地解释了SLAM经典模型中变量的含义和运动方程部分。本讲将讨论“机器人如何观测外部世界”，也就是观测方程部分。而在以相机为主的视觉SLAM中，观测主要是指相机成像的过程。

我们在现实生活中能看到大量的照片。在计算机中，一张照片由很多个像素组成，每个像素记录了色彩或亮度的信息。三维世界中的一个物体反射或发出的光线，穿过相机光心后，投影在相机的成像平面上。相机的感光器件接收到光线后，产生测量值，就得到了像素，形成了我们见到的照片。这个过程能否用数学原理来描述呢？本讲将首先讨论相机模型，说明投影关系具体如何描述，相机的内参是什么。同时，简单介绍双目成像与RGB-D相机的原理。然后，介绍二维照片像素的基本操作。最后，根据内外参数的含义，演示一个点云拼接的实验。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D



电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

5.1 相机模型

相机将三维世界中的坐标点（单位为米）映射到二维图像平面（单位为像素）的过程能够用一个几何模型进行描述。这个模型有很多种，其中最简单的称为针孔模型。针孔模型是很常用而且有效的模型，它描述了一束光线通过针孔之后，在针孔背面投影成像的关系。在本书中我们用一个简单的针孔相机模型来对这种映射关系进行建模。同时，由于相机镜头上的透镜的存在，使得光线投影到成像平面的过程中会产生畸变。因此，我们使用针孔和畸变两个模型来描述整个投影过程。

在本节先给出相机的针孔模型，再对透镜的畸变模型进行讲解。这两个模型能够把外部的三维点投影到相机内部成像平面，构成相机的内参数。

5.1.1 针孔相机模型

在初中物理课堂上，我们可能都见过一个蜡烛投影实验：在一个暗箱的前方放着一支点燃的蜡烛，蜡烛的光透过暗箱上的一个小孔投影在暗箱的后方平面上，并在这个平面上形成一个倒立的蜡烛图像。在这个过程中，小孔模型能够把三维世界中的蜡烛投影到一个二维成像平面。同理，我们可以用这个简单的模型来解释相机的成像过程，如图5-1所示。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

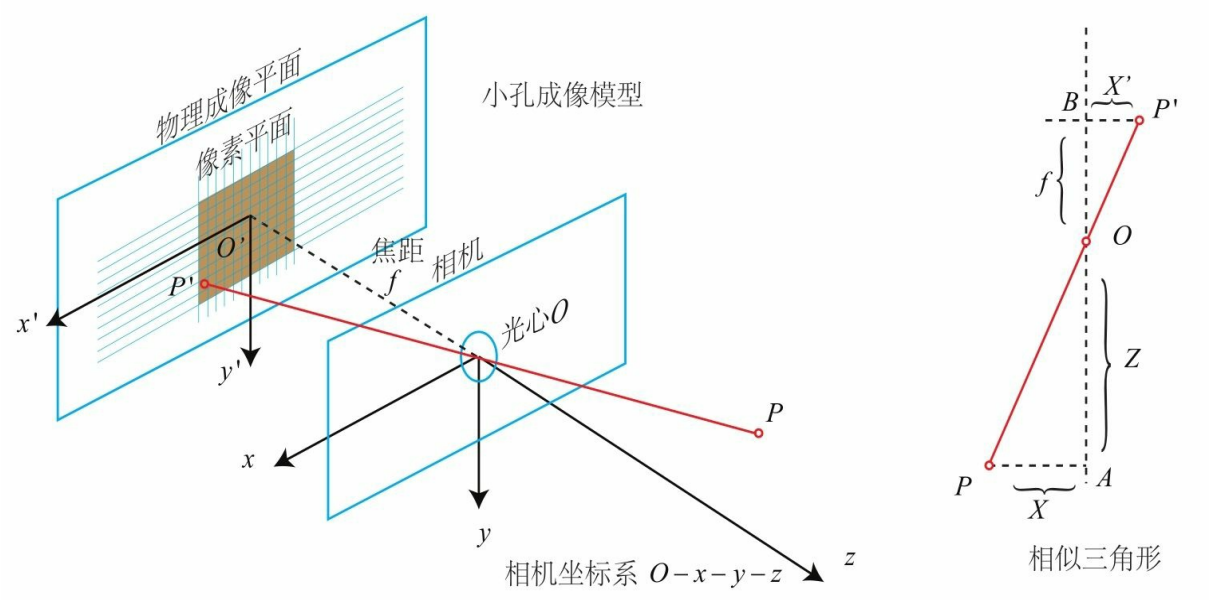


图5-1　针孔相机模型

现在来对这个简单的针孔模型进行几何建模。设O-x-y-z为相机坐标系，习惯上我们让z轴指向相机前方，x向右，y向下。O为摄像机的光心，也是针孔模型中的针孔。现实世界的空间点P，经过小孔O投影之后，落在物理成像平面O'-x'-y'上，成像点为P'。设P的坐标为$[X,Y,Z]^{\mathrm{T}}$，P'为$[X',Y',Z']^{\mathrm{T}}$，并且设物理成像平面到小孔的距离为f（焦距）。那么，根据三角形相似关系，有：

$$\frac{Z}{f}=-\frac{X}{X'}=-\frac{Y}{Y'}. \tag{5.1}$$

其中负号表示成的像是倒立的。为了简化模型，我们可以把成像平面对称到相机前方，和三维空间点一起放在摄像机坐标系的同一侧，如图5-2的中图所示。这样做可以把公式中的负号去掉，使式子更加简洁：

$$\frac{Z}{f}=\frac{X}{X'}=\frac{Y}{Y'}. \tag{5.2}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

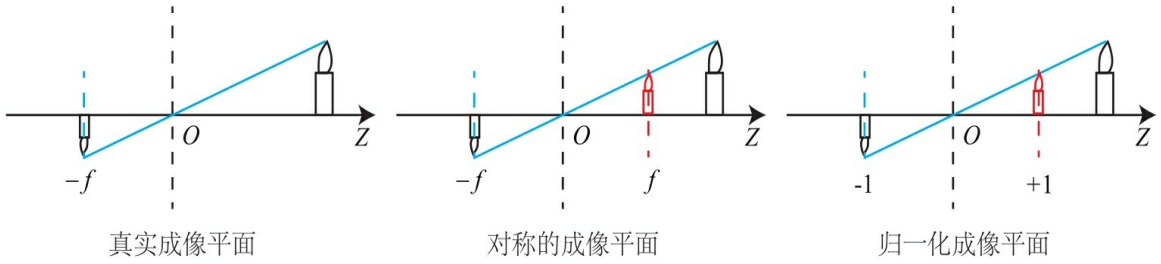


图5-2　真实成像平面，对称成像平面，归一化成像平面的图示。

整理得：

$$\begin{aligned} X' &= f\frac{X}{Z} \\ Y' &= f\frac{Y}{Z} \end{aligned} . \tag{5.3}$$

读者可能要问，为什么我们可以看似随意地把成像平面挪到前方呢？这只是我们处理真实世界与相机投影的数学手段，并且，大多数相机输出的图像并不是倒像——相机自身的软件会帮你翻转这张图像，所以你看到的一般是正着的像，也就是对称的成像平面上的像。所以，尽管从物理原理来说，小孔成像应该是倒像，但由于我们对图像作了预处理，所以理解成在对称平面上的像并不会带来什么坏处。于是，在不引起歧义的情况下，我们也不加限制地称后一种情况为针孔模型。

式（5.3）描述了点P和它的像之间的空间关系。不过，在相机中，我们最终获得的是一个个的像素，这需要在成像平面上对像进行采样和量化。为了描述传感器将感受到的光线转换成图像像素的过程，我们设在物理成像平面上固定着一个像素平面o-u-v。我们在像素平面得到了P'的像素坐标：$[u,v]^{\mathrm{T}}$。

像素坐标系[1]通常的定义方式是：原点o'位于图像的左上角，u轴向右与x轴平行，v轴向下与y轴平行。像素坐标系与成像平面之间，相差了一个缩放和一个原点的平移。我们设像素坐标在u轴上缩放了α倍，在v上缩放了β倍。同时，原点平移了$[c_x,c_y]^{\mathrm{T}}$。

那么，P'的坐标与像素坐标$[u,v]^{\mathrm{T}}$的关系为

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$\begin{cases} u = \alpha X' + c_x \\ v = \beta Y' + c_y \end{cases}. \tag{5.4}$$

代入式（5.3）并把αf合并成f_x，把βf合并成f_y，得：

$$\begin{cases} u = f_x \frac{X}{Z} + c_x \\ v = f_y \frac{Y}{Z} + c_y \end{cases}. \tag{5.5}$$

其中，f的单位为米，α,β的单位为像素/米，所以f_x,f_y的单位为像素。把该式写成矩阵形式会更加简洁，不过左侧需要用到齐次坐标：

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \frac{1}{Z} \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \triangleq \frac{1}{Z} \boldsymbol{K}\boldsymbol{P}. \tag{5.6}$$

我们按照传统的习惯把Z挪到左侧：

$$Z \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \triangleq \boldsymbol{K}\boldsymbol{P}. \tag{5.7}$$

该式中，我们把中间的量组成的矩阵称为相机的内参数矩阵（CameraIntrinsics）K。通常认为，相机的内参在出厂之后是固定的，不会在使用过程中发生变化。有的相机生产厂商会告诉你相机的内参，而有时需要你自己确定相机的内参，也就是所谓的标定。鉴于标定算法业已成熟，且网络上能找到大量的标定教学，这里就不介绍了。

有内参，自然也有相对的外参。考虑到在式（5.6）中我们使用的是P在相机坐标系下的坐标。由于相机在运动，所以P的相机坐标应该是它的世界坐标（记为P_w），根据相机的当前位姿变换到相机坐标系下的结果。相机的位姿由它的旋转矩阵R和平移向量t来描述。那么有：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$Z\boldsymbol{P}_{uv} = Z\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{K}(\boldsymbol{R}\boldsymbol{P}_w + \boldsymbol{t}) = \boldsymbol{K}\boldsymbol{T}\boldsymbol{P}_w. \tag{5.8}$$

注意后一个式子隐含了一次齐次坐标到非齐次坐标的转换（你能看出来吗？）。它描述了P的世界坐标到像素坐标的投影关系。其中，相机的位姿R,t又称为相机的外参数（Camera　Extrinsics）。相比于不变的内参，外参会随着相机运动发生改变，同时也是SLAM中待估计的目标，代表着机器人的轨迹。

上式两侧都是齐次坐标。因为齐次坐标乘上非零常数后表达同样的含义，所以可以简单地把Z去掉：

$$\boldsymbol{P}_{uv} = \boldsymbol{K}\boldsymbol{T}\boldsymbol{P}_w. \tag{5.9}$$

但这样等号意义就变了，成为在齐次坐标下相等的概念，相差了一个非零常数。为了避免麻烦，我们还是从传统意义上来定义书写等号。

这里还是提一下隐含着的齐次到非齐次的变换。可以看到，右侧的TP_w表示把一个世界坐标系下的齐次坐标变换到相机坐标系下。为了使它与K相乘，需要取它的前三维组成向量——因为TP_w最后一维为1。此时，对于这个三维向量，我们还可以按照齐次坐标的方式，把最后一维进行归一化处理，得到P在相机归一化平面上的投影：

$$\tilde{\boldsymbol{P}}_c = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = (\boldsymbol{T}\boldsymbol{P}_w)_{(1:3)}, \quad \boldsymbol{P}_c = \begin{bmatrix} X/Z \\ Y/Z \\ 1 \end{bmatrix}. \tag{5.10}$$

这时P_c可以看成一个二维的齐次坐标，称为归一化坐标。它位于相机前方z=1处的平面上。该平面称为归一化平面。由于P_c经过内参之后就得到了像素坐标，所以我们可以把像素坐标$[u,v]^{\mathrm{T}}$看成对归一化平面上的点进行量化测量的结果。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

至此，针孔相机的成像模型我们就讲清楚了。

5.1.2 畸变

为了获得好的成像效果，我们在相机的前方加了透镜。透镜的加入对成像过程中光线的传播会产生新的影响：一是透镜自身的形状对光线传播的影响，二是在机械组装过程中，透镜和成像平面不可能完全平行，这也会使得光线穿过透镜投影到成像面时的位置发生变化。

由透镜形状引起的畸变称为径向畸变。在针孔模型中，一条直线投影到像素平面上还是一条直线。可是，在实际拍摄的照片中，摄像机的透镜往往使得真实环境中的一条直线在图片中变成了曲线[2]。越靠近图像的边缘，这种现象越明显。由于实际加工制作的透镜往往是中心对称的，这使得不规则的畸变通常径向对称。它们主要分为两大类：桶形畸变和枕形畸变，如图5-3所示。

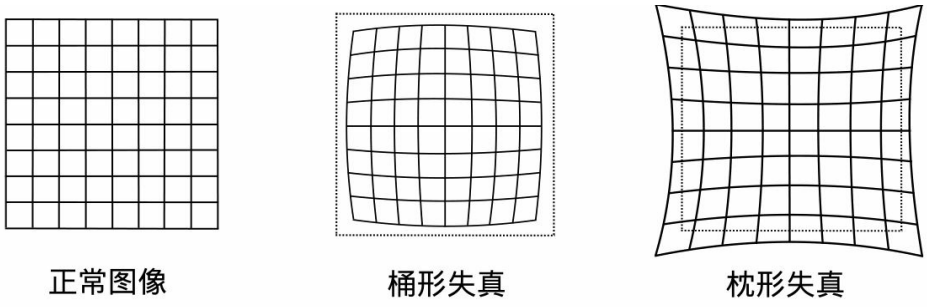


图5-3　径向畸变的两种类型。

桶形畸变是由于图像放大率随着与光轴之间的距离增加而减小，而枕形畸变则恰好相反。在这两种畸变中，穿过图像中心和光轴有交点的直线还能保持形状不变。

除了透镜的形状会引入径向畸变外，在相机的组装过程中由于不能使透镜和成像面严格平行也会引入切向畸变，如图5-4所示。

书籍汇总下载 http://t.cn/RDIAj5D

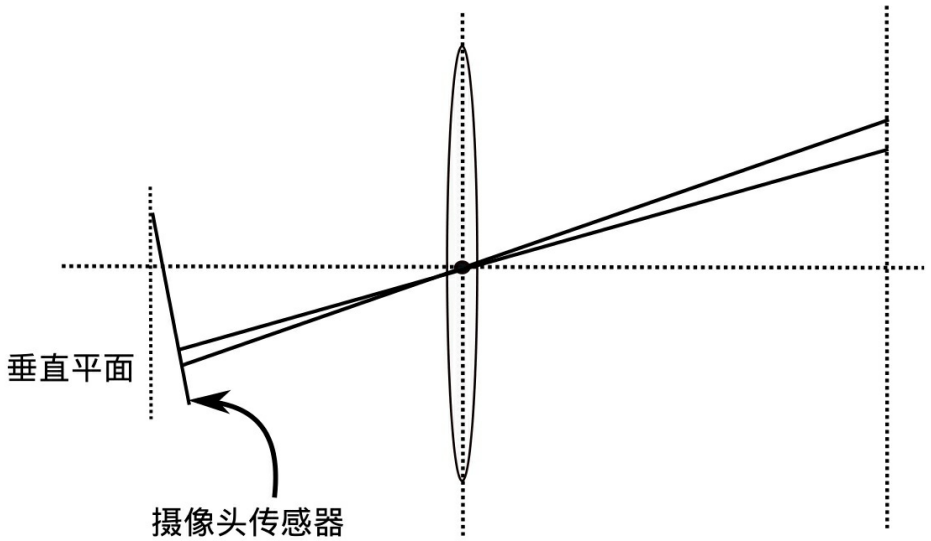


图5-4　切向畸变来源示意图。

为更好地理解径向畸变和切向畸变，我们用更严格的数学形式对两者进行描述。我们知道，平面上的任意一点p可以用笛卡儿坐标表示为$[x,y]^{\mathrm{T}}$，也可以把它写成极坐标的形式$[r,\theta]^{\mathrm{T}}$，其中r表示点p与坐标系原点之间的距离，θ表示与水平轴的夹角。径向畸变可看成坐标点沿着长度方向发生了变化δr，也就是其距离原点的长度发生了变化。切向畸变可以看成坐标点沿着切线方向发生了变化，也就是水平夹角发生了变化$\delta\theta$。

对于径向畸变，无论是桶形畸变还是枕形畸变，由于它们都是随着与中心之间的距离增加而增加，因此可以用一个多项式函数来描述畸变前后的坐标变化：这类畸变可以用与距中心的距离有关的二次及高次多项式函数进行纠正。其中$[x,y]^{\mathrm{T}}$是未纠正的点的坐标，$[x_{\text{corrected}},y_{\text{corrected}}]^{\mathrm{T}}$是纠正后的点的坐标。注意，它们都是归一化平面上的点，而不是像素平面上的点。

$$\begin{aligned} x_{\text{corrected}} &= x(1+k_1r^2+k_2r^4+k_3r^6) \\ y_{\text{corrected}} &= y(1+k_1r^2+k_2r^4+k_3r^6) \end{aligned}. \tag{5.11}$$

在式(5.11)描述的纠正模型中，对于畸变较小的图像中心区域，畸变纠正主要是k_1起作用；而对于畸变较大的边缘区域，主要是k_2起作用。普通摄像头用这两个系数就能很好地纠正径向畸变。对畸变很大的

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

摄像头，比如鱼眼镜头，可以加入k_3畸变项对畸变进行纠正。

另一方面，对于切向畸变，可以使用另外的两个参数p_1,p_2来进行纠正：

$$\begin{aligned} x_{\text{corrected}} &= x + 2p_1xy + p_2(r^2 + 2x^2) \\ y_{\text{corrected}} &= y + p_1(r^2 + 2y^2) + 2p_2xy \end{aligned}. \tag{5.12}$$

因此，联合式(5.11)和式(5.12)，对于相机坐标系中的一点$P(X,Y,Z)$，我们能够通过5个畸变系数找到这个点在像素平面上的正确位置：

1.将三维空间点投影到归一化图像平面。设它的归一化坐标为$[x,y]^{\mathrm{T}}$。

2.对归一化平面上的点进行径向畸变和切向畸变纠正。

$$\begin{cases} x_{\text{corrected}} = x(1 + k_1r^2 + k_2r^4 + k_3r^6) + 2p_1xy + p_2(r^2 + 2x^2) \\ y_{\text{corrected}} = y(1 + k_1r^2 + k_2r^4 + k_3r^6) + p_1(r^2 + 2y^2) + 2p_2xy \end{cases}. \tag{5.13}$$

3.将纠正后的点通过内参数矩阵投影到像素平面，得到该点在图像上的正确位置。

$$\begin{cases} u = f_x x_{\text{corrected}} + c_x \\ v = f_y y_{\text{corrected}} + c_y \end{cases}. \tag{5.14}$$

在上面的纠正畸变的过程中，我们使用了5个畸变项。实际应用中，可以灵活选择纠正模型，比如只选择k_1,p_1,p_2这3项等。

在这一节中，我们对相机的成像过程使用针孔模型进行了建模，也对透镜引起的径向畸变和切向畸变进行了描述。实际的图像系统中，学者们提出了很多其他的模型，比如相机的仿射模型和透视模型等，同时也存在很多其他类型的畸变。考虑到视觉SLAM中一般都使用普通的摄像头，有针孔模型及径向畸变和切向畸变模型已经足够，因此，我们不再对其他模型进行描述。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

值得一提的是，存在两种去畸变处理（Undistort，或称畸变校正）做法。我们可以选择先对整张图像进行去畸变，得到去畸变后的图像，然后讨论此图像上的点的空间位置。或者，也可以先考虑图像中的某个点，然后按照去畸变方程，讨论其去畸变后的空间位置。二者都是可行的，不过前者在视觉SLAM中似乎更加常见一些。所以，当一个图像去畸变之后，我们就可以直接用针孔模型建立投影关系，而不用考虑畸变了。因此，在后文的讨论中，我们可以直接假设图像已经进行了去畸变处理。

最后，我们小结一下单目相机的成像过程：

1.首先，世界坐标系下有一个固定的点P，世界坐标为P_w。

2.由于相机在运动，它的运动由R,t或变换矩阵$T\in SE(3)$描述。P的相机坐标为$\tilde{\boldsymbol{P}}_c = \boldsymbol{R}\boldsymbol{P}_w + \boldsymbol{t}$。

3.这时的$\tilde{\boldsymbol{P}}_c$仍有X,Y,Z三个量，把它们投影到归一化平面$Z=1$上，得到P的归一化相机坐标：$P_c=[X/Z,Y/Z,1]^T$[3]。

4.最后，P的归一化坐标经过内参后，对应到它的像素坐标：$P_{uv}=KP_c$。

综上所述，我们一共谈到了四种坐标：世界坐标、相机坐标、归一化相机坐标和像素坐标。请读者厘清它们的关系，它反映了整个成像的过程。

5.1.3 双目相机模型

针孔相机模型描述了单个相机的成像模型。然而，仅根据一个像素，我们是无法确定这个空间点的具体位置的。这是因为，从相机光心到归一化平面连线上的所有点，都可以投影至该像素上。只有当P的深度确定时（比如通过双目或RGB-D相机），我们才能确切地知道它的空间位置。如图5-5所示。

书籍汇总下载 http://t.cn/RDIAj5D

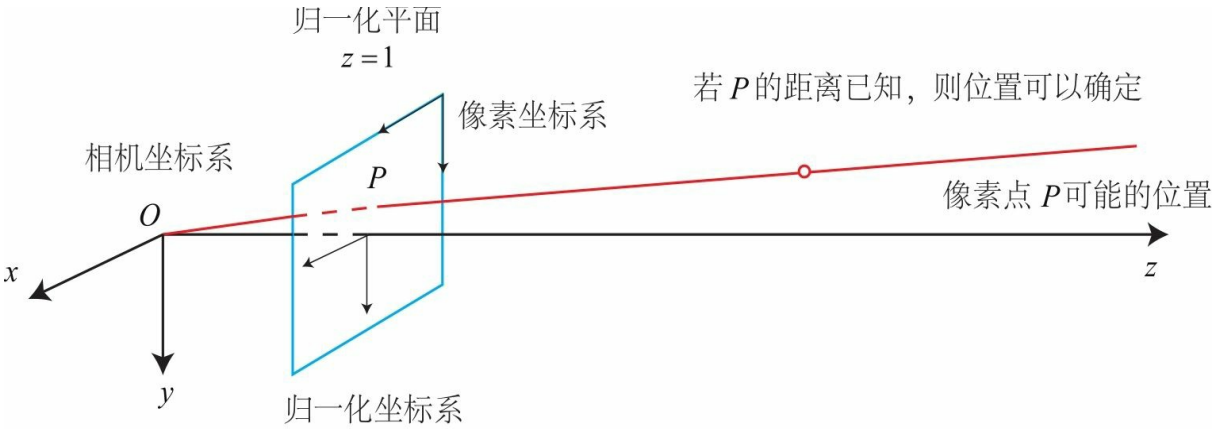


图5-5　像素点可能存在的位置。

测量像素距离（或深度）的方式有很多种，像人眼就可以根据左右眼看到的景物差异（或称视差）来判断物体与我们之间的距离。双目相机的原理亦是如此：通过同步采集左右相机的图像，计算图像间视差，来估计每一个像素的深度。下面简单讲讲双目相机的成像原理（如图5-6所示）。

双目相机一般由左眼相机和右眼相机两个水平放置的相机组成。当然也可以做成上下两个目[4]，不过我们见到的主流双目都是做成左右形式的。在左右双目相机中，我们可以把两个相机都看作针孔相机。它们是水平放置的，意味着两个相机的光圈中心都位于x轴上。两者之间的距离称为双目相机的基线（Baseline，记作b），是双目相机的重要参数。

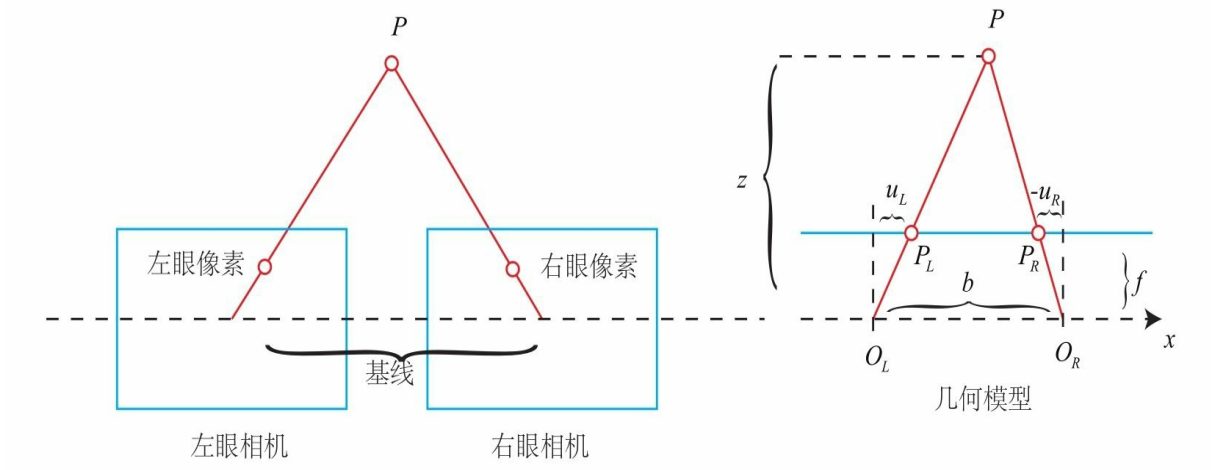


图5-6　双目相机的成像模型。O_L,O_R为左右光圈中心，方框为成像平面，f为焦距。u_L和u_R为成像平面的坐标。请注意，按照图中坐标定义，u_R应该是负数，

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

所以图中标出的距离为$-u_R$。

现在，考虑一个空间点P，它在左眼相机和右眼相机各成一像，记作P_L,P_R。由于相机基线的存在，这两个成像位置是不同的。理想情况下，由于左右相机只在x轴上有位移，因此P的像也只在x轴（对应图像的u轴）上有差异。记它的左侧坐标为u_L，右侧坐标为u_R。那么，其几何关系如图5-6右侧所示。根据$\triangle PP_LP_R$和$\triangle PO_LO_R$的相似关系，有：

$$\frac{z-f}{z}=\frac{b-u_L+u_R}{b}. \tag{5.15}$$

稍加整理，得：

$$z=\frac{fb}{d},\quad d=u_L-u_R. \tag{5.16}$$

这里，d为左右图的横坐标之差，称为视差（Disparity）。根据视差，我们可以估计一个像素与相机之间的距离。视差与距离成反比：视差越大，距离越近[5]。同时，由于视差最小为一个像素，于是双目的深度存在一个理论上的最大值，由fb确定。我们看到，当基线越长时，双目能测到的最大距离就会越远；反之，小型双目器件则只能测量很近的距离。

虽然由视差计算深度的公式很简洁，但视差d本身的计算却比较困难。我们需要确切地知道左眼图像某个像素出现在右眼图像的哪一个位置（即对应关系），这件事亦属于“人类觉得容易而计算机觉得困难”的任务。当我们想计算每个像素的深度时，其计算量与精度都将成为问题，而且只有在图像纹理变化丰富的地方才能计算视差。由于计算量的原因，双目深度估计仍需要使用GPU或FPGA来计算。这将在第13讲中提到。

5.1.4 RGB-D相机模型

相比于双目相机通过视差计算深度的方式，RGB-D相机的做法更为“主动”一些，它能够主动测量每个像素的深度。目前的RGB-D相机按

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

原理可分为两大类（见图5-7）：

1.通过红外结构光（Structured　Light）来测量像素距离的。例子有Kinect 1代、Project Tango 1代、Intel RealSense等。

2.通过飞行时间法（Time-of- fl ight，ToF）原理测量像素距离的。例子有Kinect 2代和一些现有的ToF传感器等。

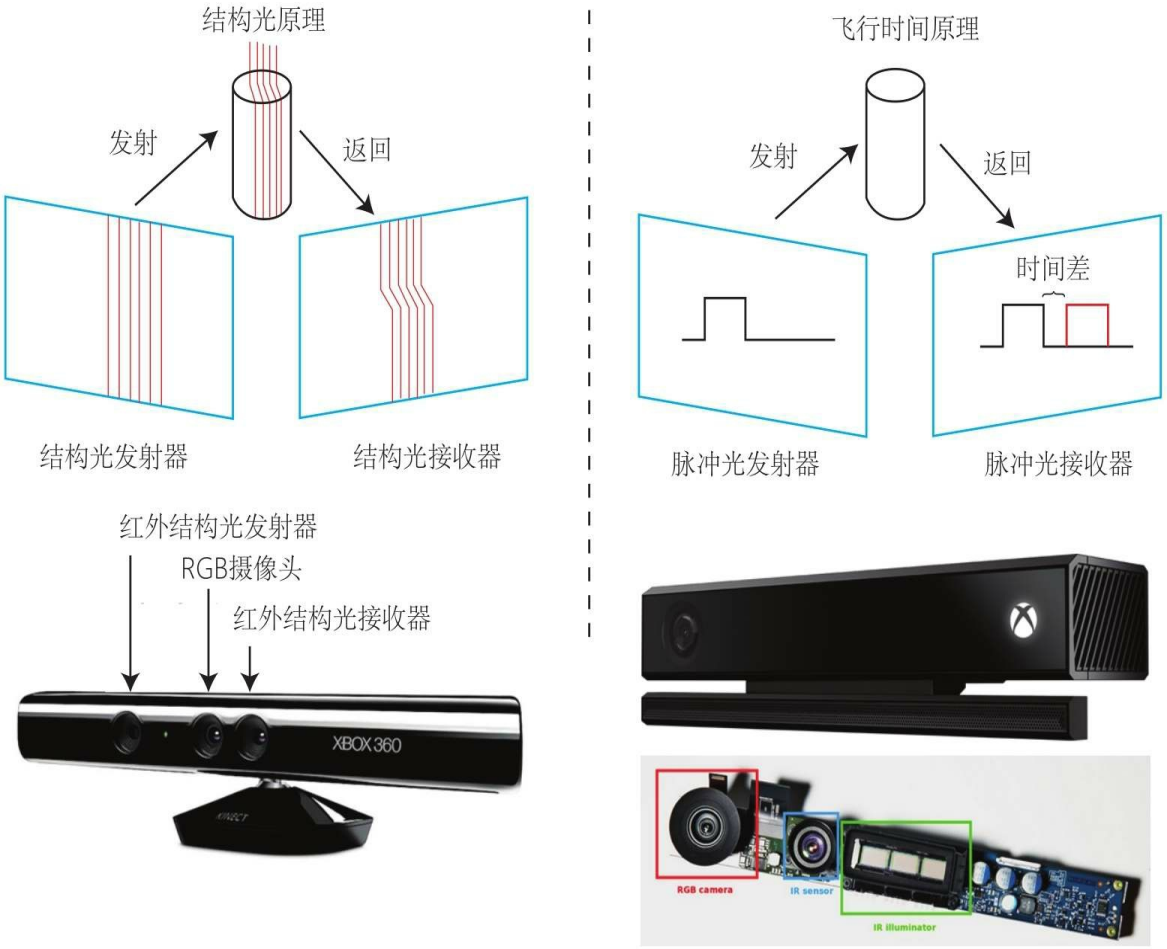


图5-7　RGB-D相机原理示意图

无论是哪种类型，RGB-D相机都需要向探测目标发射一束光线（通常是红外光）。在结构光原理中，相机根据返回的结构光图案，计算物体与自身之间的距离。而在ToF原理中，相机向目标发射脉冲光，然后根据发送到返回之间的光束飞行时间，确定物体与自身之间的距离。ToF原理和激光传感器十分相似，只不过激光是通过逐点扫描来获取距离，而ToF相机则可以获得整个图像的像素深度，这也正是RGB-D相机

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

的特点。所以，如果你把一个RGB-D相机拆开，通常会发现除了普通的摄像头之外，至少会有一个发射器和一个接收器。

在测量深度之后，RGB-D相机通常按照生产时的各相机摆放位置，自己完成深度与彩色图像素之间的配对，输出一一对应的彩色图和深度图。我们可以在同一个图像位置，读取到色彩信息和距离信息，计算像素的3D相机坐标，生成点云（Point Cloud）。对RGB-D数据，既可以在图像层面进行处理，亦可在点云层面处理。本讲的第二个实验将演示RGB-D相机的点云构建过程。

RGB-D相机能够实时地测量每个像素点的距离。但是，由于这种发射-接收的测量方式，其使用范围比较受限。用红外光进行深度值测量的RGB-D相机，容易受到日光或其他传感器发射的红外光干扰，因此不能在室外使用，同时使用多个时也会相互干扰。对于透射材质的物体，因为接收不到反射光，所以无法测量这些点的位置。此外，RGB-D相机在成本、功耗方面，都有一些劣势。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

5.2 图像

相机加上镜头，把三维世界中的信息转换成了一张由像素组成的照片，随后存储在计算机中，作为后续处理的数据来源。在数学中，图像可以用一个矩阵来描述；而在计算机中，它们占据一段连续的磁盘或内存空间，可以用二维数组来表示。这样一来，程序就不必区别它们处理的是一个数值矩阵，还是有实际意义的图像了。

本节，我们将介绍计算机图像处理的一些基本操作。特别地，通过OpenCV中图像数据的处理，理解计算机中处理图像的常见步骤，为后续章节打下基础。

计算机中图像的表示

我们从最简单的图像——灰度图说起。在一张灰度图中，每个像素位置(x,y)对应一个灰度值I，所以，一张宽度为w、高度为h的图像，数学上可以记为一个矩阵：

$$\boldsymbol{I}(x,y) \in \mathbb{R}^{w \times h}.$$

然而，计算机并不能表达整个实数空间，所以我们只能在某个范围内对图像进行量化。例如，常见的灰度图中，用0～255的整数（即一个unsigned char，1个字节）来表达图像的灰度大小。那么，一张宽度为640像素、高度为480像素分辨率的灰度图就可以表示为

```
unsigned char image[480][640];
```

为什么这里的二维数组是480×640呢？因为在程序中，图像以二维数组形式存储。它的第一个下标是指数组的行，而第二个下标则是列。在图像中，数组的行数对应图像的高度，而列数对应图像的宽度。

下面考察这幅图像的内容。图像自然是由像素组成的。当访问某一个像素时，需要指明它所处的坐标，如图5-8所示。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

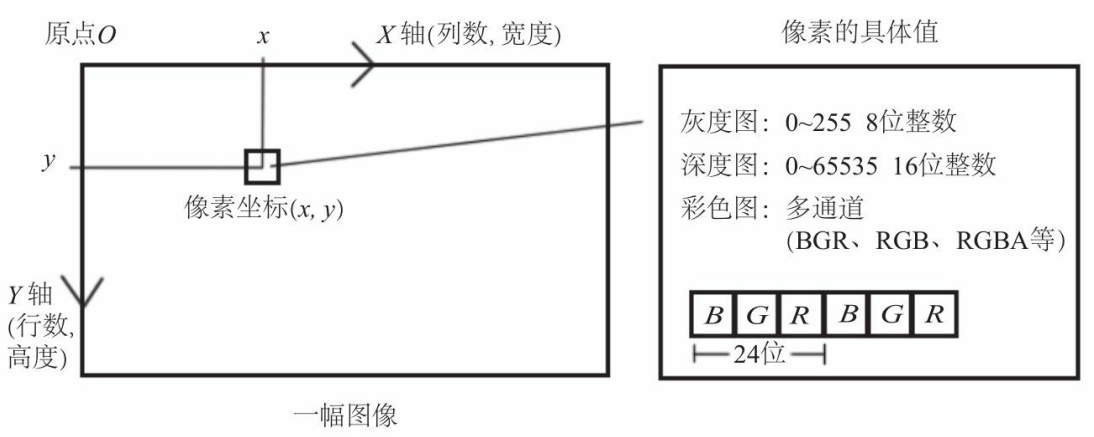


图5-8　图像坐标示意图。

图5-8左边显示了传统像素坐标系的定义方式。像素坐标系原点位于图像的左上角，*X*轴向右，*Y*轴向下（也就是前面所说的*u*,*v*坐标）。如果它还有第三个轴——*Z*轴，那么根据右手法则，*Z*轴应该是向前的。这种定义方式是与相机坐标系一致的。我们平时说的图像的宽度或列数，对应着*X*轴；而图像的行数或高度，则对应着它的*Y*轴。

根据这种定义方式，如果我们讨论一个位于*x*,*y*处的像素，那么它在程序中的访问方式应该是

```
unsigned char pixel = image[y][x];
```

它对应着灰度值*I*(*x*,*y*)的读数。请注意这里的*x*和*y*的顺序。虽然我们不厌其烦地讨论坐标系的问题，但是像这种下标顺序的错误，仍会是新手在调试过程中经常碰到的，且具有一定隐蔽性的错误之一。如果你在写程序时不慎调换了*x*,*y*的坐标，编译器无法提供任何信息，而你所能看到的只是程序运行中的一个越界错误而已。

一个像素的灰度可以用8位整数记录，也就是一个0～255的值。当我们要记录的信息更多时，一个字节恐怕就不够了。例如，在RGB-D相机的深度图中，记录了各个像素与相机之间的距离。这个距离通常是以毫米为单位，而RGB-D相机的量程通常在十几米左右，超过了255。这时，人们会采用16位整数（C++中的unsigned short）来记录深度图的信息，也就是位于0～65536的值。换算成米的话，最大可以表示65米，足够RGB-D相机使用了。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

彩色图像的表示则需要通道（channel）的概念。在计算机中，我们用红色、绿色和蓝色这三种颜色的组合来表达任意一种色彩。于是对于每一个像素，就要记录其R、G、B三个数值，每一个数值就称为一个通道。例如，最常见的彩色图像有三个通道，每个通道都由8位整数表示。在这种规定下，一个像素占据24位空间。

通道的数量、顺序都是可以自由定义的。在OpenCV的彩色图像中，通道的默认顺序是B、G、R。也就是说，当我们得到一个24位的像素时，前8位表示蓝色数值，中间8位为绿色，最后8位为红色。同理，亦可使用R、G、B的顺序表示一个彩色图。如果还想表达图像的透明度，就使用R、G、B、A四个通道。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

5.3 实践：图像的存取与访问

下面通过一个演示程序来理解，在OpenCV中图像是如何存取，而我们又是如何访问其中的像素的。

5.3.1 安装OpenCV

OpenCV[6]提供了大量的开源图像算法，是计算机视觉中使用极广的图像处理算法库。本书也使用OpenCV做基本的图像处理。在使用之前，建议读者从源代码安装它。在Ubuntu下，有从源代码安装和只安装库文件两种方式可以选择：

1.从源代码安装，是指从OpenCV网站下载所有的OpenCV源代码，并在机器上编译安装，以便使用。好处是可以选择的版本比较丰富，而且能看到源代码，不过需要花费一些编译时间。

2.只安装库文件，是指通过Ubuntu来安装由Ubuntu社区人员已经编译好的库文件，这样就无须重新编译一遍。

由于我们使用较新版本的OpenCV，所以必须从源代码来安装它。一来，可以调整一些编译选项，匹配编程环境（例如，需不需要GPU加速等）；再者，源代码安装可以使用一些额外的功能。OpenCV目前维护了两个主要版本，分为OpenCV 2.4系列和OpenCV 3系列。本书使用OpenCV 3系列。

由于OpenCV工程比较大，就不放在本书的3rdparty下了。请读者从http://opencv.org/downloads.html下载，选择OpenCV for Linux版本即可。你会获得一个像opencv-3.1.0.zip这样的压缩包。将它解压到任意目录下，我们发现OpenCV亦是一个cmake工程。

在编译之前，先来安装OpenCV的依赖项：

```
sudo apt-get install build-essential libgtk2.0-dev libvtk5-dev libjpeg-dev libtiff4
-dev libjasper-dev libopenexr-dev libtbb-dev
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

事实上，OpenCV的依赖项很多，缺少某些编译项会影响它的部分功能（不过我们也不会用到所有功能）。OpenCV会在cmake阶段检查依赖项是否会安装，并调整自己的功能。如果你的电脑上有GPU并且安装了相关依赖项，OpenCV也会把GPU加速打开。不过对于本书，上面那些依赖项就足够了。

随后的编译安装和普通的cmake工程一样，请在make之后，调用sudo make install将OpenCV安装到你的机器上（而不是仅仅编译它）。视机器配置，这个编译过程大概需要二十分钟到一个小时不等。如果你的CPU比较强，可以使用“make-j4”这样的命令，调用多个线程进行编译（-j后面的参数就是使用的线程数量）。安装后，OpenCV默认存储在/usr/local目录下。你可以去寻找OpenCV头文件与库文件的安装位置，看看它们都在哪里。另外，如果之前已经安装了OpenCV 2系列，那么建议你把OpenCV 3安装到别的地方（想想这应该如何操作）。

5.3.2 操作OpenCV图像

接下来通过一个例程熟悉一下OpenCV对图像的操作。

slambook/ch5/imageBasics.cpp

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
#include <iostream>
#include <chrono>
using namespace std;

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>

int main ( int argc, char** argv )
{
    // 读取 argv[1] 指定的图像
    cv::Mat image;
    image = cv::imread ( argv[1] ); // cv::imread 函数读取指定路径下的图像
    // 判断图像文件是否正确读取
    if ( image.data == nullptr ) // 数据不存在，可能是文件不存在
    {
        cerr<<"文件"<<argv[1]<<"不存在."<<endl;
        return 0;
    }

    // 文件顺利读取，首先输出一些基本信息
    cout<<"图像宽为"<<image.cols<<",高为"<<image.rows<<",通道数为"<<image.channels
    ()<<endl;
    cv::imshow ( "image", image );      // 用 cv::imshow 显示图像
    cv::waitKey ( 0 );     // 暂停程序，等待一个按键输入
    // 判断 image 的类型
    if ( image.type() != CV_8UC1 && image.type() != CV_8UC3 )
    {
        // 图像类型不符合要求
        cout<<"请输入一张彩色图或灰度图."<<endl;
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
29          return 0;
30      }
31
32      // 遍历图像，请注意以下遍历方式亦可使用于随机访问
33      // 使用 std::chrono 来给算法计时
34      chrono::steady_clock::time_point t1 = chrono::steady_clock::now();
35      for ( size_t y=0; y<image.rows; y++ )
36      {
37          for ( size_t x=0; x<image.cols; x++ )
38          {
39              // 访问位于 x,y 处的像素
40              // 用 cv::Mat::ptr 获得图像的行指针
41              unsigned char* row_ptr = image.ptr<unsigned char> ( y );  // row_ptr 是
                第 y 行的头指针
42              unsigned char* data_ptr = &row_ptr[ x*image.channels() ]; // data_ptr
                指向待访问的像素数据
43              // 输出该像素的每个通道，如果是灰度图就只有一个通道
44              for ( int c = 0; c != image.channels(); c++ )
45              {
46                  unsigned char data = data_ptr[c]; // data 为 I(x,y) 第 c 个通道的值
47              }
48          }
49      }
50      chrono::steady_clock::time_point t2 = chrono::steady_clock::now();
51      chrono::duration<double> time_used = chrono::duration_cast<chrono::duration<
        double>>( t2-t1 );
52      cout<<"遍历图像用时：  "<<time_used.count()<<" 秒。"<<endl;
53
54      // 关于 cv::Mat 的拷贝
55      // 直接赋值并不会复制数据
56      cv::Mat image_another = image;
57      // 修改 image_another 会导致 image 发生变化
58      image_another ( cv::Rect ( 0,0,100,100 ) ).setTo ( 0 ); // 将左上角 100×100 的
        块置零
59      cv::imshow ( "image", image );
60      cv::waitKey ( 0 );
61
62      // 使用 clone 函数来复制数据
63      cv::Mat image_clone = image.clone();
64      image_clone ( cv::Rect ( 0,0,100,100 ) ).setTo ( 255 );
65      cv::imshow ( "image", image );
66      cv::imshow ( "image_clone", image_clone );
67      cv::waitKey ( 0 );
68
69      // 其他图像操作请参见 OpenCV 官方文档，查询每个函数的调用方法。
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

```
70      cv::destroyAllWindows();
71      return 0;
72  }
```

在该例程中，我们演示了如下几个操作：图像读取、显示、像素遍历、复制、赋值等。大部分的注解已写在代码里面。编译该程序时，你需要在CMakeLists.txt中添加OpenCV的头文件，然后把程序链接到库文件上。同时，由于使用了C++11标准（如nullptr和chrono），还需要设置一下编译器：

```
1  # 添加 C++ 11 标准支持
2  set( CMAKE_CXX_FLAGS "-std=c++11" )
3
4  # 寻找 OpenCV 库
5  find_package( OpenCV REQUIRED )
6  # 添加头文件
7  include_directories( ${OpenCV_INCLUDE_DIRS} )
8
9  add_executable( imageBasics imageBasics.cpp )
10 # 链接 OpenCV 库
11 target_link_libraries( imageBasics ${OpenCV_LIBS} )
```

关于代码，我们给出几点说明：

1.程序从argv[1]，也就是命令行的第一个参数中读取图像位置。我们为读者准备了一张图像（ubuntu.png，一张Ubuntu的壁纸，希望你喜欢）供测试使用。因此，编译之后，使用如下命令调用此程序：如果在Kdevelop中调用此程序，请务必确保把参数同时给它。这可以在启动项中配置。

```
1  build/imageBasics ubuntu.png
```

2.程序的10～17行，使用cv::imread函数读取图像，并把图像和基本信息显示出来。

书籍汇总下载 http://t.cn/RDIAj5D

3.在32～52行，遍历了图像中的所有像素，并计算了整个循环所用的时间。请注意像素的遍历方式并不是唯一的，而且例程给出的方式也不是最高效的。OpenCV提供了迭代器，你可以通过迭代器遍历图像的像素。或者，cv::Mat::data提供了指向图像数据开头的指针，你可以直接通过该指针自行计算偏移量，然后得到像素的实际内存位置。例程所用的方式是为了便于读者理解图像的结构。在笔者的机器上（虚拟机），遍历这张图像用时大约12.74ms。你可以对比一下自己机器上的速度。不过，我们使用的是cmake默认的debug模式，如果使用release模式会快很多。

4.OpenCV提供了许多对图像进行操作的函数，我们在此不一一列举，否则本书就会变成OpenCV操作手册了。例程给出了较为常见的读取、显示操作，以及复制图像中可能陷入的深拷贝误区。在编程过程中，读者还会碰到图像的旋转、插值等操作，这时你应该自行查阅函数对应的文档，以了解它们的原理与使用方式。

应该指出，OpenCV并不是唯一的图像库，它只是许多图像库里使用范围较广泛的一个。不过，多数图像库对图像的表达是大同小异的。我们希望读者了解了OpenCV对图像的表示后，能够理解其他库中图像的表达，从而在需要数据格式时能够自己处理。

另外，由于cv::Mat亦是矩阵类，除了表示图像之外，我们也可以用它来存储位姿等矩阵数据。只是一般认为，Eigen对于固定大小的矩阵使用起来效率更高一些。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

5.4 实践：拼接点云

最后，我们来练习一下相机内外参的使用方法。本节程序提供了5张RGB-D图像，并且知道每个图像的内参和外参。根据RGB-D图像和相机内参，我们可以计算任何一个像素在相机坐标系下的位置。同时，根据相机位姿，又能计算这些像素在世界坐标系下的位置。如果把所有像素的空间坐标都求出来，相当于构建一张类似于地图的东西。现在我们就来练习一下。

我们准备了5对图像，位于slambook/ch5/joinMap中。在color/下有1.png到5.png共5张RGB图，而在depth/下有5张对应的深度图。同时，pose.txt文件给出了5张图像的相机位姿（以T_{wc}形式）。位姿记录的形式是平移向量加旋转四元数：

$$[x, y, z, q_x, q_y, q_z, q_w],$$

其中q_w是四元数的实部。例如，第一对图的外参为：

$$[-0.228993, 0.00645704, 0.0287837, -0.0004327, -0.113131, -0.0326832, 0.993042].$$

下面我们写一段程序，完成两件事：(1).根据内参计算一对RGB-D图像对应的点云；(2).根据各张图的相机位姿（也就是外参），把点云加起来，组成地图。

本书的点云库使用PCL（Point Cloud Library）[7]。PCL的安装比较容易，执行以下命令即可[8]：安装完成后，PCL的头文件将安装在/usr/include/pcl-1.7下，库文件位于/usr/lib/下。

```
sudo add-apt-repository ppa:v-launchpad-jochen-sprickerhof-de/pcl
sudo apt-get update
sudo apt-get install libpcl-all
```

现在编写拼接部分的程序：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

slambook/ch5/joinMap/joinMap.cpp

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```cpp
#include <iostream>
#include <fstream>
using namespace std;
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <Eigen/Geometry>
#include <boost/format.hpp>  // for formating strings
#include <pcl/point_types.h>
#include <pcl/io/pcd_io.h>
#include <pcl/visualization/pcl_visualizer.h>

int main( int argc, char** argv )
{
    vector<cv::Mat> colorImgs, depthImgs;    // 彩色图和深度图
    vector<Eigen::Isometry3d> poses;         // 相机位姿

    ifstream fin("./pose.txt");
    if (!fin)
    {
        cerr<<"请在有 pose.txt 的目录下运行此程序"<<endl;
        return 1;
    }

    for ( int i=0; i<5; i++ )
    {
        boost::format fmt( "./%s/%d.%s" ); // 图像文件格式
        colorImgs.push_back( cv::imread( (fmt%"color"%(i+1)%"png").str() ));
        depthImgs.push_back( cv::imread( (fmt%"depth"%(i+1)%"pgm").str(), -1 )); // 使用 -1 读取原始图像

        double data[7] = {0};
        for ( auto& d:data )
            fin>>d;
        Eigen::Quaterniond q( data[6], data[3], data[4], data[5] );
        Eigen::Isometry3d T(q);
        T.pretranslate( Eigen::Vector3d( data[0], data[1], data[2] ));
        poses.push_back( T );
    }

    // 计算点云并拼接
    // 相机内参
    double cx = 325.5;
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
    double cy = 253.5;
    double fx = 518.0;
    double fy = 519.0;
    double depthScale = 1000.0;

    cout<<"正在将图像转换为点云..."<<endl;
    // 定义点云使用的格式：这里用的是 XYZRGB
    typedef pcl::PointXYZRGB PointT;
    typedef pcl::PointCloud<PointT> PointCloud;

    // 新建一个点云
    PointCloud::Ptr pointCloud( new PointCloud );
    for ( int i=0; i<5; i++ )
    {
        cout<<"转换图像中： "<<i+1<<endl;
        cv::Mat color = colorImgs[i];
        cv::Mat depth = depthImgs[i];
        Eigen::Isometry3d T = poses[i];
        for ( int v=0; v<color.rows; v++ )
            for ( int u=0; u<color.cols; u++ )
            {
                unsigned int d = depth.ptr<unsigned short> ( v )[u]; // 深度值
                if ( d==0 ) continue; // 为 0 表示没有测量到
                Eigen::Vector3d point;
                point[2] = double(d)/depthScale;
                point[0] = (u-cx)*point[2]/fx;
                point[1] = (v-cy)*point[2]/fy;
                Eigen::Vector3d pointWorld = T*point;

                PointT p ;
                p.x = pointWorld[0];
                p.y = pointWorld[1];
                p.z = pointWorld[2];
                p.b = color.data[ v*color.step+u*color.channels() ];
                p.g = color.data[ v*color.step+u*color.channels()+1 ];
                p.r = color.data[ v*color.step+u*color.channels()+2 ];
                pointCloud->points.push_back( p );
            }
    }

    pointCloud->is_dense = false;
    cout<<"点云共有"<<pointCloud->size()<<"个点."<<endl;
    pcl::io::savePCDFileBinary("map.pcd", *pointCloud );
    return 0;
}
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

一点说明：

1.14～37行，读取彩色和深度图像对及位姿信息，并把位姿从四元数与平移向量转换为变换矩阵。注意程序里使用了boost::format进行字符串的格式化。

2.65～78行：计算位于(u,v)、深度为d的像素在相机坐标系下的位置，并根据外参把它们变换到世界坐标。我们知道，相机坐标p_c到像素坐标(u,v,d)的关系为

$$d\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{K}\boldsymbol{p}_c. \tag{5.17}$$

反推p_c的形式亦非常简单。设p_c=[x,y,z]，那么：

$$\begin{aligned} z &= d \\ x &= \frac{u - c_x}{f_x} z \\ y &= \frac{v - c_y}{f_y} z. \end{aligned}$$

3.为了编译此程序，我们需3个库：Eigen、OpenCV和PCL。因此主程序的CMake-Lists.txt应该如下：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
# opencv
find_package( OpenCV REQUIRED )
include_directories( ${OpenCV_INCLUDE_DIRS} )

# eigen
include_directories( "/usr/include/eigen3/" )

# pcl
find_package( PCL REQUIRED COMPONENT common io )
include_directories( ${PCL_INCLUDE_DIRS} )
add_definitions( ${PCL_DEFINITIONS} )

add_executable( joinMap joinMap.cpp )
target_link_libraries( joinMap ${OpenCV_LIBS} ${PCL_LIBRARIES} )
```

最后，我们把生成的点云以PCD格式存储在map.pcd中。用PCL提供的可视化程序打开这个文件：

```
pcl_viewer map.pcd
```

随后就可以看到拼合的点云地图了（见图5-9）。你可以拖动鼠标查看。

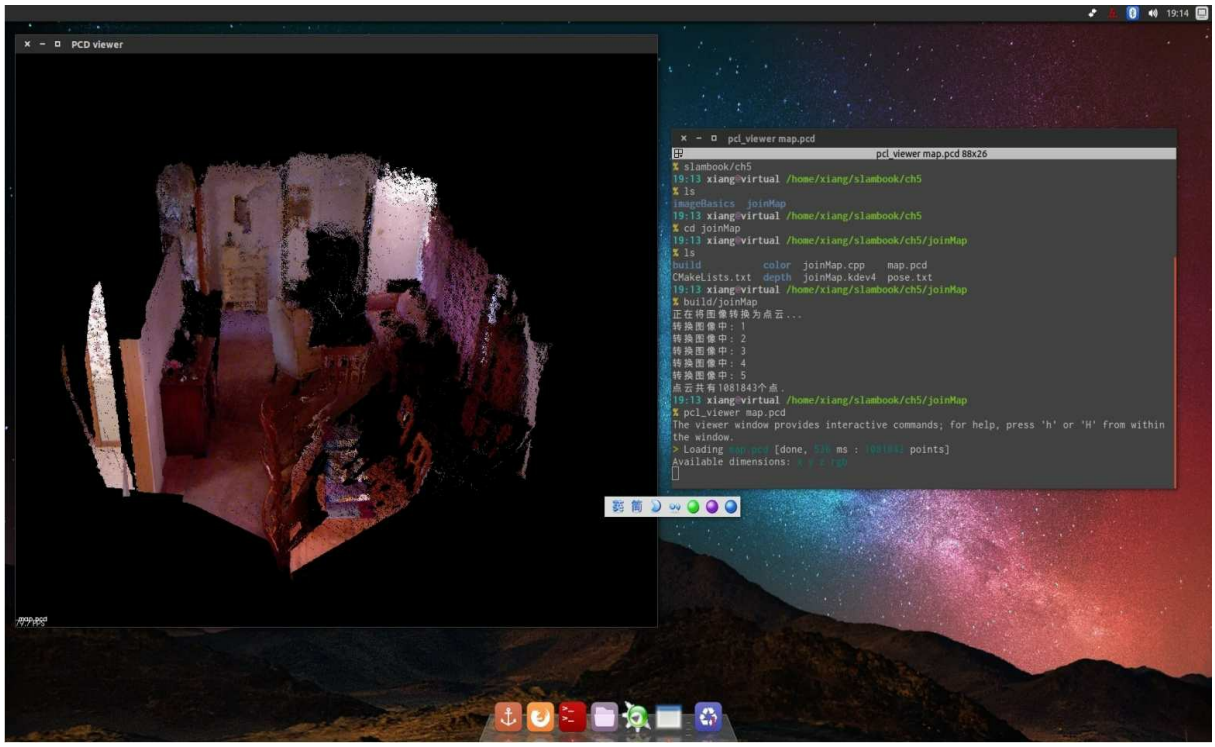


图5-9　查看拼合的点云地图。

在这个例程中，我们使用相机内参和外参来计算像素在世界坐标系中的位置，并把它们合并成一个点云。这是一个综合性的示例，请读者仔细体会并掌握其内容。

习题

1.*寻找一部相机（你的手机或笔记本的摄像头即可），标定它的内参。你可能会用到标定板，或者自己打印一张标定用的棋盘格。

2.叙述相机内参的物理意义。如果一部相机的分辨率变为原来的两倍而其他地方不变，它的内参如何变化？

3.搜索特殊相机（鱼眼或全景相机）的标定方法。它们与普通的针孔模型有何不同

4.调研全局快门相机（global shutter）和卷帘快门相机（rolling shutter）的异同。它们在SLAM中有何优缺点？

5.RGB-D相机是如何标定的？以Kinect为例，需要标定哪些参数？（参照https://github.com/code-iai/iai_kinect2。）

6.除了示例程序演示的遍历图像的方式，你还能举出哪些遍历图像

书籍汇总下载 http://t.cn/RDIAj5D

的方法？

7.*阅读OpenCV官方教程，学习它的基本用法。

---

[1]或图像坐标系，见本讲第2节。

[2]是的，它不再直了，而是变成弯的。如果往里弯，称为桶形失真；往外弯则是枕形失真。

[3]注意到Z可能小于1，说明该点位于归一化平面后面，它可能不会在相机平面上成像，实践当中要检查一次。

[4]那样的话外观会有些奇特。

[5]读者可以自己用眼睛模拟一下。

[6]官方主页：http://opencv.org。

[7]官网：http://pointclouds.org/。

[8]在Ubuntu 16.04下请通过apt-get安装，想想这该怎么做。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

第6讲　非线性优化

主要目标

1.理解最小二乘法的含义和处理方式。

2.理解高斯牛顿法（Gauss-Newton）、列文伯格—马夸尔特方法（Levenburg-Marquadt）等下降策略。

3.学习Ceres库和g2o库的基本使用方法。

在前面几讲，我们介绍了经典SLAM模型的运动方程和观测方程。现在我们已经知道，方程中的位姿可以由变换矩阵来描述，然后用李代数进行优化。观测方程由相机成像模型给出，其中内参是随相机固定的，而外参则是相机的位姿。于是，我们已经弄清了经典SLAM模型在视觉情况下的具体表达。

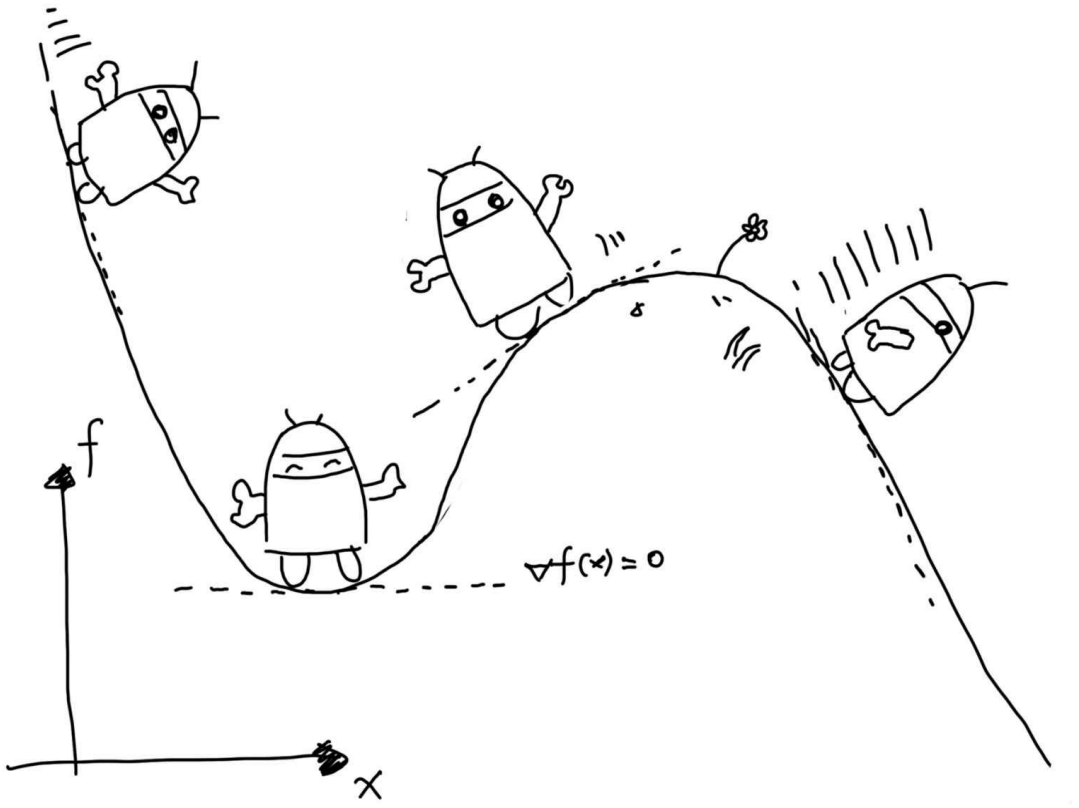
然而，由于噪声的存在，运动方程和观测方程的等式必定不是精确成立的。尽管相机可以非常好地符合针孔模型，但遗憾的是，我们得到的数据通常是受各种未知噪声影响的。即使我们有高精度的相机，运动方程和观测方程也只能近似成立。所以，与其假设数据必须符合方程，不如来讨论如何在有噪声的数据中进行准确的状态估计。

大多数现代视觉SLAM算法都不需要那么高成本的传感器，甚至也不需要那么昂贵的处理器来计算这些数据，这全是算法的功劳。由于在SLAM问题中，同一个点往往会被一部相机在不同的时间内多次观测，同一部相机在每个时刻观测到的点也不止一个。这些因素交织在一起，使我们拥有了更多的约束，最终能够较好地从噪声数据中恢复出我们需要的东西。本节就将介绍如何通过优化处理噪声数据，并且由这些表层逐渐深入到图优化本质，给出图优化的解决算法初步介绍并且提供训练实例。

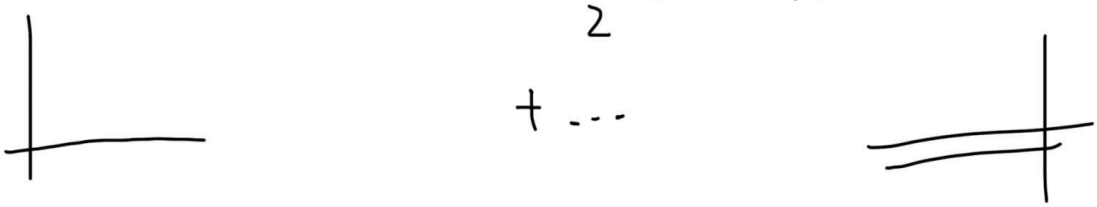
电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D





$$f(x+\Delta x) \approx f(x) + \nabla f(x)\Delta x + \frac{1}{2}\Delta x^T H(x)\Delta x + \cdots$$



电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

6.1 状态估计问题

6.1.1 最大后验与最大似然

接着前面几讲的内容，我们回顾一下第2讲讨论的经典SLAM模型。它由一个状态方程和一个运动方程构成，如式(2.5)所示：

$$\begin{cases} \boldsymbol{x}_k = f\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k\right) + \boldsymbol{w}_k \\ \boldsymbol{z}_{k,j} = h\left(\boldsymbol{y}_j, \boldsymbol{x}_k\right) + \boldsymbol{v}_{k,j} \end{cases}. \tag{6.1}$$

通过第4讲的知识，我们了解到这里的x_k乃是相机的位姿。我们可以使用变换矩阵或李代数表示它。至于观测方程，第5讲已经说明，即针孔相机模型。为了让读者对它们有更深的印象，我们不妨讨论一下其具体参数化形式。首先，位姿变量x_k可以由T_k或$\exp(\xi_k^\wedge)$表达，二者是等价的。由于运动方程在视觉SLAM中没有特殊性，我们暂且不讨论，而主要讨论观测方程。假设在x_k处对路标y_j进行了一次观测，对应到图像上的像素位置$z_{k,j}$，那么，观测方程可以表示成

$$s\boldsymbol{z}_{k,j} = \boldsymbol{K}\exp\left(\boldsymbol{\xi}^\wedge\right)\boldsymbol{y}_j. \tag{6.2}$$

根据上一讲的内容，读者应该知道，这里K为相机内参，s为像素点的距离。同时，这里的$z_{k,j}$和y_j都必须以齐次坐标来描述，且中间有一次齐次到非齐次的转换。如果你还不熟悉这个过程，请回到上一讲再仔细阅读。

现在，考虑数据受噪声影响后会发生什么改变。在运动和观测方程中，我们通常假设两个噪声项w_k,$v_{k,j}$满足零均值的高斯分布：

$$\boldsymbol{w}_k \sim N\left(0, \boldsymbol{R}_k\right), \boldsymbol{v}_k \sim N\left(0, \boldsymbol{Q}_{k,j}\right). \tag{6.3}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

在这些噪声的影响下，我们希望通过带噪声的数据z和u推断位姿x和地图y（以及它们的概率分布），这构成了一个状态估计问题。由于在SLAM过程中，这些数据是随时间逐渐到来的，所以在历史上很长一段时间内，研究者们使用滤波器，尤其是扩展卡尔曼滤波器（EKF）求解它。卡尔曼滤波器关心当前时刻的状态估计x_k，而对之前的状态则不多考虑；相对地，近年来普遍使用的非线性优化方法，使用所有时刻采集到的数据进行状态估计，被认为优于传统的滤波器[13]，而成为当前视觉SLAM的主流方法。因此，本书重点介绍以非线性优化为主的优化方法，对卡尔曼滤波器则留到第10讲再进行讨论。本讲将介绍非线性优化的基本知识，然后在第10讲、第11讲中对它们进行更深入的分析。

首先，从概率学角度来看一下我们正在讨论什么问题。在非线性优化中，把所有待估计的变量放在一个“状态变量”中：

$$\boldsymbol{x} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_M\}.$$

现在，我们说，对机器人状态的估计，就是已知输入数据u和观测数据z的条件下，求计算状态x的条件概率分布：

$$P(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{u}). \tag{6.4}$$

类似于x，这里u和z也是对所有数据的统称。特别地，当没有测量运动的传感器，而只有一张张的图像时，即只考虑观测方程带来的数据时，相当于估计$P(x|z)$的条件概率分布。如果忽略图像在时间上的联系，把它们看作一堆彼此没有关系的图片，该问题也称为Structure from Motion（SfM），即如何从许多图像中重建三维空间结构[22]。在这种情况下，SLAM可以看作图像具有时间先后顺序，需要实时求解一个SfM问题。为了估计状态变量的条件分布，利用贝叶斯法则，有：

$$P(\boldsymbol{x}|\boldsymbol{z}) = \frac{P(\boldsymbol{z}|\boldsymbol{x})\,P(\boldsymbol{x})}{P(\boldsymbol{z})} \propto P(\boldsymbol{z}|\boldsymbol{x})\,P(\boldsymbol{x}). \tag{6.5}$$

贝叶斯法则左侧通常称为后验概率，右侧的$P(z|x)$称为似然，另一部分$P(x)$称为先验。直接求后验分布是困难的，但是求一个状态最优估计，使得在该状态下后验概率最大化（Maximize a Posterior，MAP），

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

则是可行的：

$$\boldsymbol{x}^*_{\mathrm{MAP}} = \arg\max P(\boldsymbol{x}|\boldsymbol{z}) = \arg\max P(\boldsymbol{z}|\boldsymbol{x})P(\boldsymbol{x}). \tag{6.6}$$

请注意贝叶斯法则的分母部分与待估计的状态x无关，因而可以忽略。贝叶斯法则告诉我们，求解最大后验概率相当于最大化似然和先验的乘积。进一步，我们当然也可以说，对不起，我不知道机器人位姿大概在什么地方，此时就没有了先验。那么，可以求解x的最大似然估计（Maximize Likelihood Estimation，MLE）：

$$\boldsymbol{x}^*_{\mathrm{MLE}} = \arg\max P(\boldsymbol{z}|\boldsymbol{x}). \tag{6.7}$$

直观讲，似然是指“在现在的位姿下，可能产生怎样的观测数据”。由于我们知道观测数据，所以最大似然估计可以理解成：“在什么样的状态下，最可能产生现在观测到的数据”。这就是最大似然估计的直观意义。

6.1.2 最小二乘的引出

那么如何求最大似然估计呢？我们说，在高斯分布的假设下，最大似然能够有较简单的形式。回顾观测模型，对于某一次观测：

$$\boldsymbol{z}_{k,j} = h(\boldsymbol{y}_j, \boldsymbol{x}_k) + \boldsymbol{v}_{k,j},$$

由于我们假设了噪声项$v_k \sim N(0, Q_{k,j})$，所以观测数据的条件概率为

$$P(\boldsymbol{z}_{j,k}|\boldsymbol{x}_k, \boldsymbol{y}_j) = N\left(h(\boldsymbol{y}_j, \boldsymbol{x}_k), \boldsymbol{Q}_{k,j}\right).$$

它依然是一个高斯分布。为了计算使它最大化的x_k, y_j，我们往往使用最小化负对数的方式来求一个高斯分布的最大似然。

高斯分布在负对数下有较好的数学形式。考虑任意高维高斯分布$x \sim N(\mu, \Sigma)$，它的概率密度函数展开形式为

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$P(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^N \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right). \tag{6.8}$$

对其取负对数，则变为

$$-\ln(P(\boldsymbol{x})) = \frac{1}{2}\ln\left((2\pi)^N \det(\boldsymbol{\Sigma})\right) + \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}). \tag{6.9}$$

对原分布求最大化相当于对负对数求最小化。在最小化上式的x时，第一项与x无关，可以略去。于是，只要最小化右侧的二次型项，就得到了对状态的最大似然估计。代入SLAM的观测模型，相当于在求：

$$\boldsymbol{x}^* = \arg\min\left((\boldsymbol{z}_{k,j} - h(\boldsymbol{x}_k, \boldsymbol{y}_j))^{\mathrm{T}}\boldsymbol{Q}_{k,j}^{-1}(\boldsymbol{z}_{k,j} - h(\boldsymbol{x}_k, \boldsymbol{y}_j))\right). \tag{6.10}$$

我们发现，该式等价于最小化噪声项（即误差）的平方（Σ范数意义下）。因此，对于所有的运动和任意的观测，我们定义数据与估计值之间的误差：

$$\begin{aligned} \boldsymbol{e}_{v,k} &= \boldsymbol{x}_k - f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \\ \boldsymbol{e}_{y,j,k} &= \boldsymbol{z}_{k,j} - h(\boldsymbol{x}_k, \boldsymbol{y}_j), \end{aligned} \tag{6.11}$$

$$J(\boldsymbol{x}) = \sum_k \boldsymbol{e}_{v,k}^{\mathrm{T}}\boldsymbol{R}_k^{-1}\boldsymbol{e}_{v,k} + \sum_k \sum_j \boldsymbol{e}_{y,k,j}^{\mathrm{T}}\boldsymbol{Q}_{k,j}^{-1}\boldsymbol{e}_{y,k,j}. \tag{6.12}$$

并求该误差的平方和：这样就得到了一个总体意义下的最小二乘问题（Least Square Problem）。我们明白它的最优解等价于状态的最大似然估计。直观讲，由于噪声的存在，当我们把估计的轨迹与地图代入SLAM的运动、观测方程中时，它们并不会完美地成立。这时怎么办呢？我们对状态的估计值进行微调，使得整体的误差下降一些。当然这个下降也有限度，它一般会到达一个极小值。这就是一个典型非线性优

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

化的过程。

仔细观察式(6.12)，我们发现SLAM中的最小二乘问题具有一些特定的结构：

•首先，整个问题的目标函数由许多个误差的（加权的）平方和组成。虽然总体的状态变量维数很高，但每个误差项都是简单的，仅与一两个状态变量有关。例如，运动误差只与x_{k-1},x_k有关，观测误差只与x_k,y_j有关。每个误差项都是一个小规模的约束，我们之后会谈论如何对它们进行线性近似，最后再把这个误差项的小雅可比矩阵块放到整体的雅可比矩阵中。由于这种做法，我们称每个误差项对应的优化变量为参数块（Parameter Block）。

•整体误差由很多小型误差项之和组成的问题，其增量方程的求解会具有一定的稀疏性（会在第10讲详细讲解），使得它们在大规模时亦可求解。

•其次，如果使用李代数表示，则该问题是无约束的最小二乘问题。但如果用旋转矩阵（变换矩阵）描述位姿，则会引入旋转矩阵自身的约束（旋转矩阵必须是正交矩阵且行列式为1）。额外的约束会使优化变得更困难。这体现了李代数的优势。

•最后，我们使用了平方形式（二范数）度量误差，它是直观的，相当于欧氏空间中距离的平方。但也存在着一些问题，并且不是唯一的度量方式。我们亦可使用其他的范数构建优化问题。

现在，我们介绍如何求解这个最小二乘问题。本讲将介绍非线性优化的基本知识，特别地，针对这样一个通用的无约束非线性最小二乘问题，探讨它是如何求解的。在后续几讲，我们会大量使用本讲的结果，详细讨论它在SLAM前端、后端中的应用。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

6.2 非线性最小二乘

我们先来考虑一个简单的最小二乘问题：

$$\min_{x} \frac{1}{2}\|f(\boldsymbol{x})\|_2^2. \tag{6.13}$$

这里自变量$x \in \mathbf{R}^n$，f是任意非线性函数，我们设它有m维：$f(x) \in \mathbf{R}^m$。下面讨论如何求解这样一个优化问题。

如果f是个数学形式上很简单的函数，那么问题也许可以用解析形式来求。令目标函数的导数为零，然后求解x的最优值，就和求二元函数的极值一样：

$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \mathbf{0}. \tag{6.14}$$

解此方程，就得到了导数为零处的极值。它们可能是极大、极小或鞍点处的值，只要逐个比较它们的函数值大小即可。但是，这个方程是否容易求解呢？这取决于f导函数的形式。在SLAM中，我们使用李代数来表示机器人的旋转和位移。尽管我们在李代数一讲讨论了它的导数形式，但这不代表我们就能够顺利求解上式这样一个复杂的非线性方程。

对于不方便直接求解的最小二乘问题，我们可以用迭代的方式，从一个初始值出发，不断地更新当前的优化变量，使目标函数下降。具体步骤可列写如下：

1.给定某个初始值x_0。

2.对于第k次迭代，寻找一个增量Δx_k，使得$\|f(x_k+\Delta x_k)\|^2_2$达到极小值。

3.若Δx_k足够小，则停止。

4.否则，令$x_{k+1}=x_k+\Delta x_k$，返回第2步。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

这让求解导函数为零的问题变成了一个不断寻找梯度并下降的过程。直到某个时刻增量非常小，无法再使函数下降。此时算法收敛，目标达到了一个极小，我们也就完成了寻找极小值的过程。在这个过程中，我们只要找到迭代点的梯度方向即可，而无须寻找全局导函数为零的情况。

接下来的问题是，增量Δx_k如何确定？实际上，研究者们已经花费了大量精力探索增量的求解方式。我们将介绍两类办法，用不同的手段来寻找这个增量。目前这两种方法在视觉SLAM的优化问题上被广泛采用，大多数优化库都可以使用它们。

6.2.1 一阶和二阶梯度法

求解增量最直观的方式是将目标函数在x附近进行泰勒展开：

$$\|f(\boldsymbol{x}+\Delta\boldsymbol{x})\|_2^2 \approx \|f(\boldsymbol{x})\|_2^2 + \boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x} + \frac{1}{2}\Delta\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\Delta\boldsymbol{x}. \tag{6.15}$$

这里J是$\|f(x)\|^2$关于x的导数（雅可比矩阵），而H则是二阶导数（海塞〔Hessian〕矩阵）。我们可以选择保留泰勒展开的一阶或二阶项，对应的求解方法则为一阶梯度或二阶梯度法。如果保留一阶梯度，那么增量的解就为

$$\Delta\boldsymbol{x}^* = -\boldsymbol{J}^{\mathrm{T}}(\boldsymbol{x}). \tag{6.16}$$

它的直观意义非常简单，只要我们沿着反向梯度方向前进即可。通常我们还会计算该方向上的一个步长λ，求得最快的下降方式。这种方法被称为最速下降法。

另一方面，如果保留二阶梯度信息，那么增量方程为

$$\Delta\boldsymbol{x}^* = \arg\min \|f(\boldsymbol{x})\|_2^2 + \boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x} + \frac{1}{2}\Delta\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\Delta\boldsymbol{x}. \tag{6.17}$$

求右侧等式关于Δx的导数并令它为零，就得到了增量的解：

$$\boldsymbol{H}\Delta\boldsymbol{x} = -\boldsymbol{J}^{\mathrm{T}}. \tag{6.18}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

该方法又称为牛顿法。我们看到，一阶和二阶梯度法都十分直观，只要把函数在迭代点附近进行泰勒展开，并针对更新量做最小化即可。由于泰勒展开之后函数变成了多项式，所以求解增量时只需解线性方程即可，避免了直接求导函数为零这样的非线性方程的困难。不过，这两种方法也存在它们自身的问题。最速下降法过于贪心，容易走出锯齿路线，反而增加了迭代次数。而牛顿法则需要计算目标函数的H矩阵，这在问题规模较大时非常困难，我们通常倾向于避免H的计算。所以，接下来我们详细地介绍两类更加实用的方法：高斯牛顿法和列文伯格—马夸尔特方法。

6.2.2 高斯牛顿法

高斯牛顿法是最优化算法中最简单的方法之一。它的思想是将$f(x)$进行一阶的泰勒展开（请注意不是目标函数$f(x)^2$）：

$$f(\boldsymbol{x}+\Delta\boldsymbol{x}) \approx f(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x}. \tag{6.19}$$

这里$J(x)$为$f(x)$关于x的导数，实际上是一个$m\times n$矩阵，也是一个雅可比矩阵。根据前面的框架，当前的目标是寻找下降矢量Δx，使得$\|f(x+\Delta x)\|^2$达到最小。为了求Δx，我们需要解一个线性的最小二乘问题：

$$\Delta\boldsymbol{x}^* = \arg\min_{\Delta\boldsymbol{x}} \frac{1}{2}\|f(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x}\|^2. \tag{6.20}$$

这个方程与之前有什么不一样呢？根据极值条件，将上述目标函数对Δx求导，并令导数为零。由于这里考虑的是Δx的导数（而不是x），我们最后将得到一个线性的方程。为此，先展开目标函数的平方项：

$$\begin{aligned}\frac{1}{2}\|f(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x}\|^2 &= \frac{1}{2}(f(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x})^{\mathrm{T}}(f(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x}) \\ &= \frac{1}{2}\left(\|f(\boldsymbol{x})\|_2^2 + 2f(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x} + \Delta\boldsymbol{x}^{\mathrm{T}}\boldsymbol{J}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x}\right).\end{aligned}$$

求上式关于Δx的导数，并令其为零：

$$2\boldsymbol{J}(\boldsymbol{x})^{\mathrm{T}}f(\boldsymbol{x}) + 2\boldsymbol{J}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x} = \boldsymbol{0}.$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

可以得到如下方程组：

$$\boldsymbol{J}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x} = -\boldsymbol{J}(\boldsymbol{x})^{\mathrm{T}}f(\boldsymbol{x}). \tag{6.21}$$

注意，我们要求解的变量是Δx，因此这是一个线性方程组，我们称它为增量方程，也可以称为高斯牛顿方程（Gauss Newton equation）或者正规方程（Normal equation）。我们把左边的系数定义为H，右边定义为g，那么上式变为

$$\boldsymbol{H}\Delta\boldsymbol{x} = \boldsymbol{g}. \tag{6.22}$$

这里把左侧记作H是有意义的。对比牛顿法可见，高斯牛顿法用$J^{\mathrm{T}}J$作为牛顿法中二阶Hessian矩阵的近似，从而省略了计算H的过程。求解增量方程是整个优化问题的核心所在。如果我们能够顺利解出该方程，那么高斯牛顿法的算法步骤可以写成：

1.给定初始值x_0。

2.对于第k次迭代，求出当前的雅可比矩阵$J(x_k)$和误差$f(x_k)$。

3.求解增量方程：$H\Delta x_k = g$。

4.若Δx_k足够小，则停止。否则，令$x_{k+1} = x_k + \Delta x_k$，返回第2步。

从算法步骤中可以看到，增量方程的求解占据着主要地位。原则上，它要求我们所用的近似H矩阵是可逆的（而且是正定的），但实际数据中计算得到的$J^{\mathrm{T}}J$却只有半正定性。也就是说，在使用高斯牛顿法时，可能出现$J^{\mathrm{T}}J$为奇异矩阵或者病态（illcondition）的情况，此时增量的稳定性较差，导致算法不收敛。更严重的是，就算我们假设H非奇异也非病态，如果我们求出来的步长Δx太大，也会导致我们采用的局部近似(6.19)不够准确，这样一来我们甚至无法保证它的迭代收敛，哪怕是让目标函数变得更大都是有可能的。

尽管高斯牛顿法有这些缺点，但它依然值得我们去学习，因为在非线性优化里，相当多的算法都可以归结为高斯牛顿法的变种。这些算法都借助了高斯牛顿法的思想并且通过自己的改进修正其缺点。例如一些线搜索方法(line search method)，这类改进就是加入了一个标量α，在确

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

定了Δx后进一步找到α使得$\|f(x+\alpha\Delta x)\|^2$达到最小，而不是像高斯牛顿法那样简单地令$\alpha=1$。

列文伯格—马夸尔特方法在一定程度上修正了这些问题，一般认为它比高斯牛顿法更为健壮。尽管它的收敛速度可能会比高斯牛顿法更慢，被称为阻尼牛顿法（Damped Newton Method），但是在SLAM里面却被大量应用。

6.2.3 列文伯格—马夸尔特方法

由于高斯牛顿法中采用的近似二阶泰勒展开只能在展开点附近有较好的近似效果，所以我们很自然地想到应该给Δx添加一个信赖区域（Trust Region），不能让它太大而使得近似不准确。非线性优化中有一系列这类方法，这类方法也被称为信赖区域方法（Trust Region Method）。在信赖区域里边，我们认为近似是有效的；出了这个区域，近似可能会出问题。

那么如何确定这个信赖区域的范围呢？一个比较好的方法是根据我们的近似模型跟实际函数之间的差异来确定：如果差异小，我们就让范围尽可能大；如果差异大，我们就缩小这个近似范围。因此，考虑使用

$$\rho = \frac{f(\boldsymbol{x}+\Delta\boldsymbol{x}) - f(\boldsymbol{x})}{\boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x}}. \tag{6.23}$$

来判断泰勒近似是否够好。ρ的分子是实际函数下降的值，分母是近似模型下降的值。如果ρ接近于1，则近似是好的。如果ρ太小，说明实际减小的值远少于近似减小的值，则认为近似比较差，需要缩小近似范围。反之，如果ρ比较大，则说明实际下降的比预计的更大，我们可以放大近似范围。

于是，我们构建一个改良版的非线性优化框架，该框架会比高斯牛顿法有更好的效果：

1.给定初始值x_0，以及初始优化半径μ。

2.对于第k次迭代，求解：

书籍汇总下载 http://t.cn/RDIAj5D

$$\min_{\Delta \boldsymbol{x}_k} \frac{1}{2}\|f(\boldsymbol{x}_k)+\boldsymbol{J}(\boldsymbol{x}_k)\Delta \boldsymbol{x}_k\|^2, \quad s.t.\|\boldsymbol{D}\Delta \boldsymbol{x}_k\|^2 \leqslant \mu, \tag{6.24}$$

这里μ是信赖区域的半径，D将在后文说明。

3.计算ρ。

4.若$\rho > \frac{3}{4}$，则$\mu=2\mu$。

5.若$\rho < \frac{1}{4}$，则$\mu=0.5\mu$。

6.如果ρ大于某阈值，则认为近似可行。令$x_{k+1}=x_k+\Delta x_k$。

7.判断算法是否收敛。如不收敛则返回第2步，否则结束。

这里近似范围扩大的倍数和阈值都是经验值，可以替换成别的数值。在式(6.24)中，我们把增量限定于一个半径为μ的球中，认为只在这个球内才是有效的。带上D之后，这个球可以看成一个椭球。在列文伯格提出的优化方法中，把D取成单位阵I，相当于直接把Δx约束在一个球中。随后，马夸尔特提出将D取成非负数对角阵——实际中通常用J^TJ的对角元素平方根，使得在梯度小的维度上约束范围更大一些。

不论如何，在列文伯格—马夸尔特优化中，我们都需要解式(6.24)那样一个子问题来获得梯度。这个子问题是带不等式约束的优化问题，我们用拉格朗日乘子将它转化为一个无约束优化问题：

$$\min_{\Delta \boldsymbol{x}_k} \frac{1}{2}\|f(\boldsymbol{x}_k)+\boldsymbol{J}(\boldsymbol{x}_k)\Delta \boldsymbol{x}_k\|^2 + \frac{\lambda}{2}\|\boldsymbol{D}\Delta \boldsymbol{x}\|^2. \tag{6.25}$$

这里λ为拉格朗日乘子。类似于高斯牛顿法中的做法，把它展开后，我们发现该问题的核心仍是计算增量的线性方程：

$$\left(\boldsymbol{H}+\lambda \boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}\right)\Delta \boldsymbol{x}=\boldsymbol{g}. \tag{6.26}$$

可以看到，增量方程相比于高斯牛顿法，多了一项λD^TD。如果考虑它的简化形式，即$D=I$，那么相当于求解：

$$(\boldsymbol{H}+\lambda \boldsymbol{I})\Delta \boldsymbol{x}=\boldsymbol{g}.$$

我们看到，当参数λ比较小时，H占主要地位，这说明二次近似模型

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

在该范围内是比较好的，列文伯格—马夸尔特方法更接近于高斯牛顿法。另一方面，当λ比较大时，λI占据主要地位，列文伯格—马夸尔特方法更接近于一阶梯度下降法（即最速下降），这说明附近的二次近似不够好。列文伯格—马夸尔特方法的求解方式，可在一定程度上避免线性方程组的系数矩阵的非奇异和病态问题，提供更稳定、更准确的增量Δx。

在实际中，还存在许多其他的方式来求解函数的增量，例如Dog-Leg等方法。我们在这里所介绍的，只是最常见而且最基本的方式，也是视觉SLAM中用得最多的方式。总而言之，非线性优化问题的框架，分为Line Search和Trust Region两类。Line Search先固定搜索方向，然后在该方向寻找步长，以最速下降法和高斯牛顿法为代表。而Trust Region则先固定搜索区域，再考虑找该区域内的最优点。此类方法以列文伯格—马夸尔特方法为代表。实际问题中，我们通常选择高斯牛顿法或列文伯格—马夸尔特方法作为梯度下降策略。

6.2.4 小结

由于不希望这本书变成一本让人觉得头疼的数学教科书，所以这里只罗列了最常见的两种非线性优化方案——高斯牛顿法和列文伯格—马夸尔特方法。我们避开了许多数学性质上的讨论。如果读者对优化感兴趣，可以进一步阅读专门介绍数值优化的书籍（这是一个很大的课题），例如[23]。以高斯牛顿法和列文伯格—马夸尔特方法为代表的优化方法，在很多开源的优化库中都已经实现并提供给用户，我们会在下文进行实验。最优化是处理许多实际问题的基本数学工具，不光在视觉SLAM中起着核心作用，在类似于深度学习等其他领域，它也是求解问题的核心方法之一。我们希望读者能够根据自身能力，去了解更多的最优化算法。

也许你发现了，无论是高斯牛顿法还是列文伯格—马夸尔特方法，在做最优化计算时，都需要提供变量的初始值。你也许会问到，这个初始值能否随意设置？当然不是。实际上非线性优化的所有迭代求解方案，都需要用户来提供一个良好的初始值。由于目标函数太复杂，导致在求解空间上的变化难以琢磨，对问题提供不同的初始值往往会导致不同的计算结果。这种情况是非线性优化的通病：大多数算法都容易陷入

书籍汇总下载 http://t.cn/RDIAj5D

局部极小值。因此，无论是哪类科学问题，我们提供初始值都应该有科学依据，例如视觉SLAM问题中，我们会用ICP、PnP之类的算法提供优化初始值。总之，一个良好的初始值对最优化问题非常重要！

也许读者还会对上面提到的最优化产生疑问：如何求解线性增量方程组呢？我们只讲到了增量方程是一个线性方程，但是直接对系数矩阵进行求逆岂不是要进行大量的计算？当然不是。在视觉SLAM算法里，经常遇到Δx的维度大到好几百或者上千，如果你是要做大规模的视觉三维重建，就会经常发现这个维度可以轻易达到几十万甚至更高的级别。要对那么大个矩阵进行求逆是大多数处理器无法负担的，因此存在着许多针对线性方程组的数值求解方法。在不同的领域有不同的求解方式，但几乎没有一种方式是直接求系数矩阵的逆，我们会采用矩阵分解的方法来解线性方程，例如QR、Cholesky等分解方法。这些方法通常在矩阵论等教科书中可以找到，我们不多加介绍。

幸运的是，视觉SLAM里这个矩阵往往有特定的稀疏形式，这为实时求解优化问题提供了可能性。我们将在第10讲中详细介绍它的原理。利用稀疏形式的消元、分解，最后再进行求解增量，会让求解的效率大大提高。在很多开源的优化库上，维度为一万多的变量在一般的PC上就可以在几秒甚至更短的时间内被求解出来，其原因也是用了更加高级的数学工具。视觉SLAM算法现在能够实时地实现，也多亏了系数矩阵是稀疏的，如果矩阵是稠密的，恐怕优化这类视觉SLAM算法就不会被学界广泛采纳了[24,25,26]。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

6.3 实践：Ceres

我们前面说了很多理论，现在来实践一下前面提到的优化算法。在本讲的实践部分中，我们主要向大家介绍两个C++的优化库：来自谷歌的Ceres库[27]以及基于图优化的g2o库[28]。由于g2o的使用还需要介绍一点图优化的相关知识，所以我们先来介绍Ceres，然后介绍一些图优化理论，最后来讲g2o。由于优化算法在之后的“视觉里程计”和“后端”中都会出现，所以请读者务必掌握优化算法的意义，理解程序的内容。

6.3.1 Ceres简介

Ceres库面向通用的最小二乘问题的求解，作为用户，我们需要做的就是定义优化问题，然后设置一些选项，输入进Ceres求解即可。Ceres求解的最小二乘问题最一般的形式如下（带边界的核函数最小二乘）：

$$\begin{aligned} \min_{x} \quad & \frac{1}{2}\sum_{i}\rho_i\left(\left\|f_i\left(x_{i_1},\cdots x_{i_n}\right)\right\|^2\right) \\ s.t. \quad & l_j \leqslant x_j \leqslant u_j. \end{aligned} \tag{6.27}$$

可以看到，目标函数由许多平方项经过一个核函数$\rho(\cdot)$之后求和组成[1]。在最简单的情况下，取ρ为恒等函数，则目标函数即为许多项的平方和。在这个问题中，优化变量为$x_1,\cdots,x_n$，f_i称为代价函数（Cost function），在SLAM中亦可理解为误差项。l_j和u_j为第j个优化变量的上限和下限。在最简单的情况下，取l_j=-∞,u_j=∞（不限制优化变量的边界），并且取ρ为恒等函数时，就得到了无约束的最小二乘问题，和我们先前说的是一致的。

在Ceres中，我们将定义优化变量x和每个代价函数f_i，再调用Ceres进行求解。我们可以选择使用高斯牛顿法或者列文伯格—马夸尔特方法进行梯度下降，并设定梯度下降的条件，Ceres会在优化之后将最优估

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

计值返回。下面，我们通过一个曲线拟合的实验来实际操作一下Ceres，理解优化的过程。

6.3.2 安装Ceres

为了使用Ceres，首先需要进行编译安装！建议去GitHub上下载Ceres：https://github.com/ceres-solver/ceres-solver。本书资源的3rdparty下也附带了Ceres库。

与之前碰到的库一样，Ceres是一个cmake工程。先来安装它的依赖项，在Ubuntu中可以用apt-get安装，主要是谷歌自己使用的一些日志和测试工具：

```
sudo apt-get install liblapack-dev libsuitesparse-dev libcxsparse3.1.2 libgflags-dev libgoogle-glog-dev libgtest-dev
```

然后，进入Ceres库目录下，使用cmake编译并安装它。这个过程我们已经做过很多遍了，此处不再赘述。安装完成后，在/usr/local/include/ceres下找到Ceres的头文件，并在/usr/local/lib/下找到名为libceres.a的库文件。有了这些文件，就可以使用Ceres进行优化计算了。

6.3.3 使用Ceres拟合曲线

我们的演示实验包括使用Ceres和接下来的g2o进行曲线拟合。假设有一条满足以下方程的曲线：

$$y = \exp(ax^2 + bx + c) + w,$$

其中a,b,c为曲线的参数，w为高斯噪声。我们故意选择了这样一个非线性模型，以使问题不至于太简单。现在，假设我们有N个关于x,y的观测数据点，想根据这些数据点求出曲线的参数。那么，可以求解下面的最小二乘问题以估计曲线参数：

书籍汇总下载 http://t.cn/RDIAj5D

$$\min_{a,b,c} \frac{1}{2} \sum_{i=1}^{N} \left\| y_i - \exp\left(ax_i^2 + bx_i + c\right) \right\|^2. \tag{6.28}$$

请注意，在这个问题中，待估计的变量是*a,b,c*，而不是*x*。我们写一个程序，先根据模型生成*x,y*的真值，然后在真值中添加高斯分布的噪声。随后，使用Ceres从带噪声的数据拟合参数模型。

slambook/ch6/ceres_curve_ fi tting/main.cpp

```cpp
#include <iostream>
#include <opencv2/core/core.hpp>
#include <ceres/ceres.h>
#include <chrono>

using namespace std;

// 代价函数的计算模型
struct CURVE_FITTING_COST
{
    CURVE_FITTING_COST ( double x, double y ) : _x ( x ), _y ( y ) {}
    // 残差的计算
    template <typename T>
    bool operator() (
        const T* const abc,     // 模型参数，有 3 维
        T* residual ) const     // 残差
    {
        // y-exp(ax^2+bx+c)
        residual[0] = T ( _y ) - ceres::exp ( abc[0]*T ( _x ) *T ( _x ) + abc[1]*T
        ( _x ) + abc[2] );
        return true;
    }
    const double _x, _y;    // x,y 数据
};

```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
25 int main ( int argc, char** argv )
26 {
27     double a=1.0, b=2.0, c=1.0;         // 真实参数值
28     int N=100;                          // 数据点
29     double w_sigma=1.0;                 // 噪声 Sigma 值
30     cv::RNG rng;                        // OpenCV 随机数产生器
31     double abc[3] = {0,0,0};            // abc 参数的估计值
32
33     vector<double> x_data, y_data;      // 数据
34
35     cout<<"generating data: "<<endl;
36     for ( int i=0; i<N; i++ )
37     {
38         double x = i/100.0;
39         x_data.push_back ( x );
40         y_data.push_back (
41             exp ( a*x*x + b*x + c ) + rng.gaussian ( w_sigma )
42         );
43         cout<<x_data[i]<<" "<<y_data[i]<<endl;
44     }
45
46     // 构建最小二乘问题
47     ceres::Problem problem;
48     for ( int i=0; i<N; i++ )
49     {
50         problem.AddResidualBlock (     // 向问题中添加误差项
51             // 使用自动求导，模板参数：误差类型，输出维度，输入维度，数值参照前面
                struct 中写法
52             new ceres::AutoDiffCostFunction<CURVE_FITTING_COST, 1, 3> (
53                 new CURVE_FITTING_COST ( x_data[i], y_data[i] )
54             ),
55             nullptr,            // 核函数，这里不使用，为空
56             abc                 // 待估计参数
57         );
58     }
59
60     // 配置求解器
61     ceres::Solver::Options options;     // 这里有很多配置项可以填
62     options.linear_solver_type = ceres::DENSE_QR;  // 增量方程如何求解
63     options.minimizer_progress_to_stdout = true;   // 输出到cout
64
65     ceres::Solver::Summary summary;                // 优化信息
66     chrono::steady_clock::time_point t1 = chrono::steady_clock::now();
67     ceres::Solve ( options, &problem, &summary );  // 开始优化
68     chrono::steady_clock::time_point t2 = chrono::steady_clock::now();
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

```
    chrono::duration<double> time_used = chrono::duration_cast<chrono::duration<
    double>>( t2-t1 );
    cout<<"solve time cost = "<<time_used.count()<<" seconds. "<<endl;

    // 输出结果
    cout<<summary.BriefReport() <<endl;
    cout<<"estimated a,b,c = ";
    for ( auto a:abc ) cout<<a<<" ";
    cout<<endl;

    return 0;
}
```

程序中需要说明的地方均已加注释。可以看到，我们利用OpenCV的噪声生成器生成了100个带高斯噪声的数据，随后利用Ceres进行拟合。Ceres的用法如下：

1.定义Cost Function模型。方法是书写一个类，并在类中定义带模板参数的()运算符，这样该类就成为了一个拟函数（Functor，C++术语）。这种定义方式使得Ceres可以像调用函数一样，对该类的某个对象（比如a）调用a<double>()方法——这使对象具有像函数那样的行为。

2.调用AddResidualBlock将误差项添加到目标函数中。由于优化需要梯度，我们有若干种选择：（1）使用Ceres的自动求导（Auto Diff）；（2）使用数值求导（Numeric Diff）；（3）自行推导解析的导数形式，提供给Ceres。其中自动求导在编码上是最方便的，于是我们使用自动求导。

3.自动求导需要指定误差项和优化变量的维度。这里的误差是标量，维度为1；优化的是a,b,c三个量，维度为3。于是，在自动求导类的模板参数中设定变量维度为1、3。

4.设定好问题后，调用solve函数进行求解。你可以在options里配置（非常详细的）优化选项。例如，可以选择使用Line Search还是Trust Region、迭代次数、步长，等等。读者可以查看Options的定义，看看有哪些优化方法可选，当然默认的配置已经可用于很广泛的问题了。

最后，我们来看看实验结果。调用build/curve_ fi tting查看优化结果：

```
% build/curve_fitting
generating data:
0 2.71828
0.01 2.93161
0.02 2.12942
0.03 2.46037
......
iter         cost cost_change |gradient|    |step|   tr_ratio tr_radius ls_iter
iter_time total_time

0    1.824887e+04    0.00e+00   1.38e+03  0.00e+00   0.00e+00  1.00e+04       0
4.09e-05   1.48e-04
1    2.748700e+39   -2.75e+39   0.00e+00  7.67e+01  -1.52e+35  5.00e+03       1
1.09e-04   3.13e-04
2    2.429783e+39   -2.43e+39   0.00e+00  7.62e+01  -1.35e+35  1.25e+03       1
3.57e-05   3.75e-04
......
18   5.310764e+01    3.42e+00   8.50e+00  2.81e-01   9.89e-01  2.53e+03       1
3.09e-05   1.15e-03
19   5.125939e+01    1.85e+00   2.84e+00  2.98e-01   9.90e-01  7.60e+03       1
2.85e-05   1.19e-03
20   5.097693e+01    2.82e-01   4.34e-01  1.48e-01   9.95e-01  2.28e+04       1
2.82e-05   1.23e-03
21   5.096854e+01    8.39e-03   3.24e-02  2.87e-02   9.96e-01  6.84e+04       1
3.04e-05   1.27e-03
solve time cost = 0.00133349 seconds.
Ceres Solver Report: Iterations: 22, Initial cost: 1.824887e+04, Final cost:
5.096854e+01, Termination: CONVERGENCE
estimated a,b,c = 0.891943 2.17039 0.944142
```

从Ceres给出的优化过程可以看到，整体误差大约从18248下降到了50.9，并且梯度也是越来越小。在迭代22次后算法收敛，最后的估计值为

$$a = 0.891943, \quad b = 2.17039, \quad c = 0.944142.$$

书籍汇总下载 http://t.cn/RDIAj5D

而我们设定的真值为

$$a = 1, \quad b = 2, \quad c = 1.$$

它们相差不多。

为了更直观地显示数据，可以把它画出来，如图6-1所示。其中显示了带噪声的数据、真实模型和估计模型，可以看到估计模型和真实模型非常接近，几乎重合。我们同时记录了Ceres的运行时间，对这样一个100个点的优化问题，计算时间约1.3ms（虚拟机上）。

希望读者通过这个简单的例子对Ceres的使用方法有一个大致了解。它的优点是提供了自动求导工具，使得不必去计算很麻烦的雅可比矩阵。Ceres的自动求导是通过模板元实现的，在编译时期就可以完成自动求导工作，不过仍然是数值导数。本书大部分时候仍然会介绍雅可比矩阵的计算，因为那样对理解问题更有帮助，而且在优化中更少出现问题。此外，Ceres的优化过程配置也很丰富，使其适合很广泛的最小二乘优化问题，包括SLAM中的各种问题。

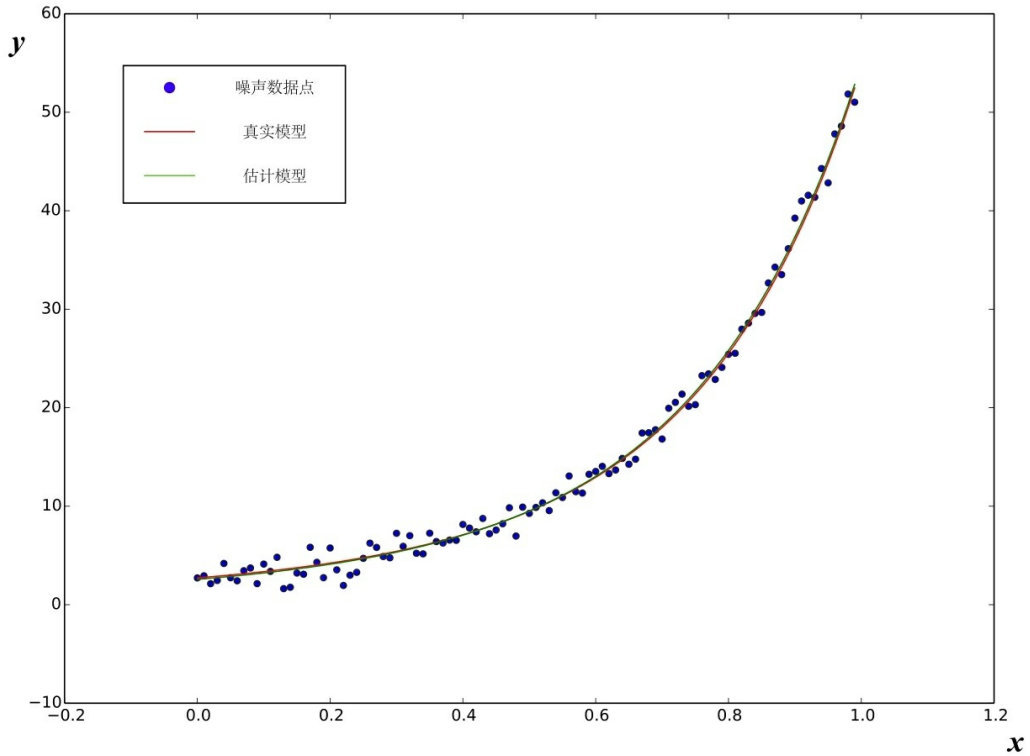


图6-1　使用Ceres进行曲线拟合。真实模型和估计模型非常接近。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

6.4 实践：g2o

本讲的第2个实践部分将介绍另一个（主要在SLAM领域）广为使用的优化库：g2o（General Graphic Optimization，G^2O）。它是一个基于图优化的库。图优化是一种将非线性优化与图论结合起来的理论，因此在使用它之前，我们花一点篇幅介绍一下图优化理论。

6.4.1 图优化理论简介

我们已经介绍了非线性最小二乘的求解方式。它们是由很多个误差项之和组成的。然而，仅有一组优化变量和许多个误差项，我们并不清楚它们之间的关联。比如，某个优化变量x_j存在于多少个误差项中呢？我们能保证对它的优化是有意义的吗？进一步，我们希望能够直观地看到该优化问题长什么样。于是，就牵涉到了图优化。

图优化，是把优化问题表现成图（Graph）的一种方式。这里的图是图论意义上的图。一个图由若干个顶点（Vertex），以及连接着这些顶点的边（Edge）组成。进而，用顶点表示优化变量，用边表示误差项。于是，对任意一个上述形式的非线性最小二乘问题，我们可以构建与之对应的一个图。

图6-2是一个简单的图优化例子。我们用三角形表示相机位姿节点，用圆形表示路标点，它们构成了图优化的顶点；同时，实线表示相机的运动模型，虚线表示观测模型，它们构成了图优化的边。此时，虽然整个问题的数学形式仍是式(6.12)那样，但现在我们可以直观地看到问题的结构了。如果希望，也可以做去掉孤立顶点或优先优化边数较多（或按图论的术语，度数较大）的顶点这样的改进。但是最基本的图优化是用图模型来表达一个非线性最小二乘的优化问题。而我们可以利用图模型的某些性质做更好的优化。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

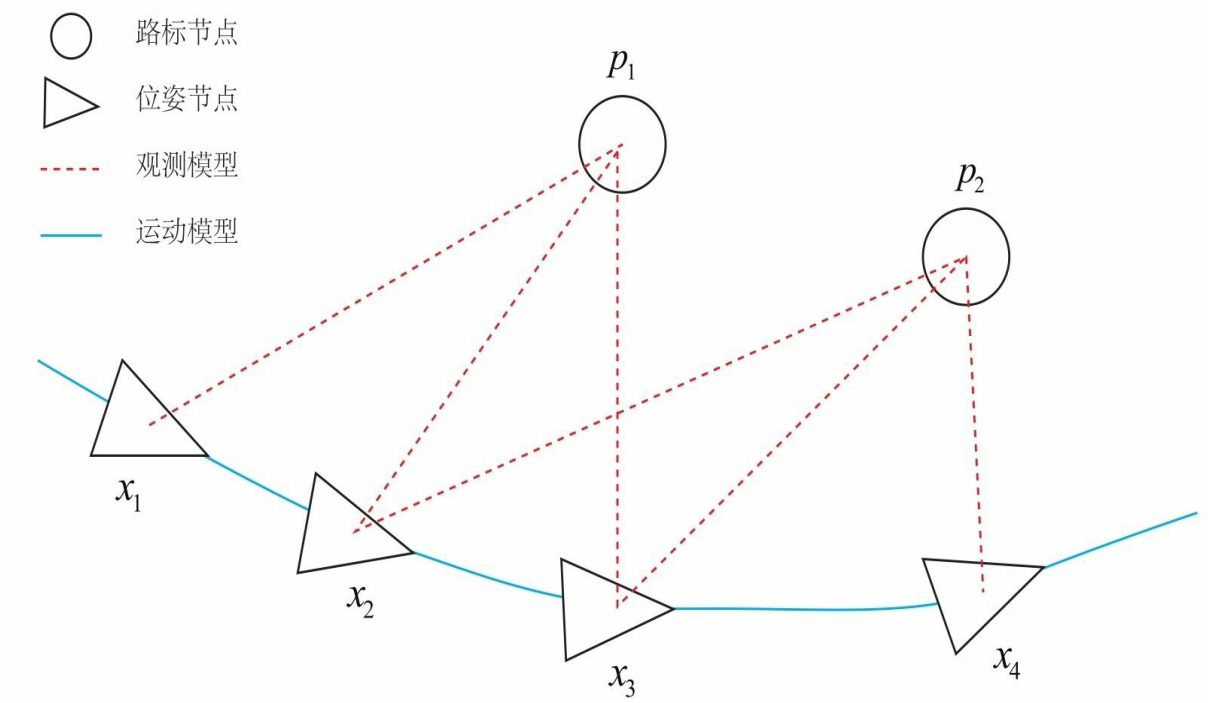


图6-2　图优化的例子。

g2o为SLAM提供了图优化所需的内容。下面演示一下g2o的使用方法。

6.4.2　g2o的编译与安装

在使用一个库之前，我们需要对它进行编译和安装。读者应该已经体验过很多次这种过程了，它们基本大同小异。关于g2o，读者可以从GitHub下载它：https://github.com/RainerKuemmerle/g2o，或从本书提供的第三方代码库中获得。

解压代码包后，你会看到g2o库的所有源码，它也是一个cmake工程。我们先来安装它的依赖项（部分依赖项与Ceres重合）：

```
sudo apt-get install libqt4-dev qt4-qmake libqglviewer-dev libsuitesparse-dev
libcxsparse3.1.2 libcholmod-dev
```

然后，按照cmake的方式对g2o进行编译安装即可，这里略去对该过程的说明。安装完成后，g2o的头文件将位于/usr/local/g2o下，库文件位

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

于/usr/local/lib/下。现在，我们重新考虑Ceres例程中的曲线拟合实验，在g2o中实验一遍。

6.4.3 使用g2o拟合曲线

为了使用g2o，首先要将曲线拟合问题抽象成图优化。这个过程中，只要记住节点为优化变量，边为误差项即可。曲线拟合的图优化问题可以画成图6-3的形式。

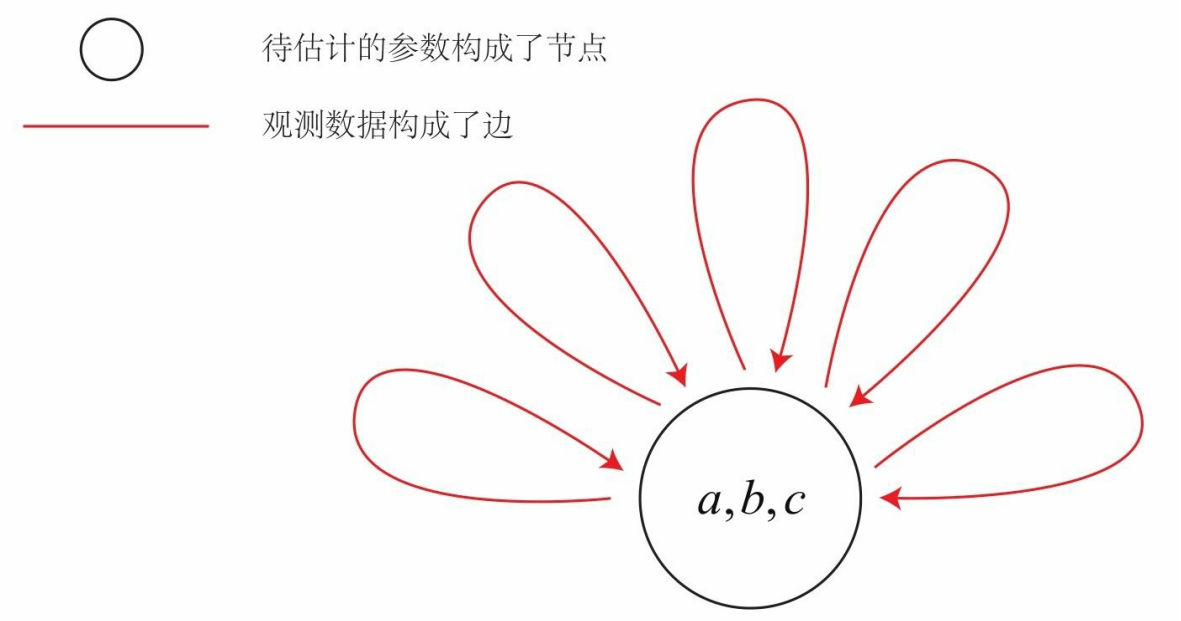


图6-3　曲线拟合对应的图优化模型。（莫明其妙地有些像华为的标志）

在曲线拟合问题中，整个问题只有一个顶点：曲线模型的参数a,b,c；而各个带噪声的数据点，构成了一个个误差项，也就是图优化的边。但这里的边与我们平时想的边不太一样，它们是一元边（Unary Edge），即只连接一个顶点——因为整个图只有一个顶点。所以在图6-3中，我们只能把它画成自己连到自己的样子。事实上，图优化中一条边可以连接一个、两个或多个顶点，这主要反映每个误差与多少个优化变量有关。在稍有些玄妙的说法中，我们把它叫作超边（Hyper Edge），整个图叫作超图（Hyper Graph）[2]。

弄清了这个图模型之后，接下来就是在g2o中建立该模型进行优化了。作为g2o的用户，我们要做的事主要包含以下步骤：

书籍汇总下载 http://t.cn/RDIAj5D

1.定义顶点和边的类型。

2.构建图。

3.选择优化算法。

4.调用g2o进行优化，返回结果。

下面演示一下程序。

slambook/ch6/g2o_curve_ fi tting/main.cpp

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```cpp
#include <iostream>
#include <g2o/core/base_vertex.h>
#include <g2o/core/base_unary_edge.h>
#include <g2o/core/block_solver.h>
#include <g2o/core/optimization_algorithm_levenberg.h>
#include <g2o/core/optimization_algorithm_gauss_newton.h>
#include <g2o/core/optimization_algorithm_dogleg.h>
#include <g2o/solvers/dense/linear_solver_dense.h>
#include <Eigen/Core>
#include <opencv2/core/core.hpp>
#include <cmath>
#include <chrono>
using namespace std;

// 曲线模型的顶点，模板参数：优化变量维度和数据类型
class CurveFittingVertex: public g2o::BaseVertex<3, Eigen::Vector3d>
{
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
    virtual void setToOriginImpl() // 重置
    {
        _estimate << 0,0,0;
    }

    virtual void oplusImpl( const double* update ) // 更新
    {
        _estimate += Eigen::Vector3d(update);
    }
    // 存盘和读盘：留空
    virtual bool read( istream& in ) {}
    virtual bool write( ostream& out ) const {}
};

// 误差模型 模板参数：观测值维度，类型，连接顶点类型
class CurveFittingEdge: public g2o::BaseUnaryEdge<1,double,CurveFittingVertex>
{
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
    CurveFittingEdge( double x ): BaseUnaryEdge(), _x(x) {}
    // 计算曲线模型误差
    void computeError()
    {
        const CurveFittingVertex* v = static_cast<const CurveFittingVertex*> (
        _vertices[0]);
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
44          const Eigen::Vector3d abc = v->estimate();
45          _error(0,0) = _measurement - std::exp( abc(0,0)*_x*_x + abc(1,0)*_x + abc
            (2,0) ) ;
46      }
47      virtual bool read( istream& in ) {}
48      virtual bool write( ostream& out ) const {}
49  public:
50      double _x;  // x 值， y 值为 _measurement
51  };
52
53  int main( int argc, char** argv )
54  {
55      double a=1.0, b=2.0, c=1.0;         // 真实参数值
56      int N=100;                          // 数据点
57      double w_sigma=1.0;                 // 噪声Sigma值
58      cv::RNG rng;                        // OpenCV随机数产生器
59      double abc[3] = {0,0,0};            // abc参数的估计值
60
61      vector<double> x_data, y_data;      // 数据
62
63      cout<<"generating data: "<<endl;
64      for ( int i=0; i<N; i++ )
65      {
66          double x = i/100.0;
67          x_data.push_back ( x );
68          y_data.push_back (
69              exp ( a*x*x + b*x + c ) + rng.gaussian ( w_sigma )
70          );
71          cout<<x_data[i]<<" "<<y_data[i]<<endl;
72      }
73
74      // 构建图优化，先设定g2o
75      // 矩阵块：每个误差项优化变量维度为 3 ，误差值维度为 1
76      typedef g2o::BlockSolver< g2o::BlockSolverTraits<3,1> > Block;
77      // 线性方程求解器：稠密的增量方程
78      Block::LinearSolverType* linearSolver = new g2o::LinearSolverDense<Block::
        PoseMatrixType>();
79      Block* solver_ptr = new Block( linearSolver );      // 矩阵块求解器
80      // 梯度下降方法，从GN, LM, DogLeg 中选
81      g2o::OptimizationAlgorithmLevenberg* solver = new g2o::
        OptimizationAlgorithmLevenberg( solver_ptr );
82      // 取消下面的注释以使用 GN 或 DogLeg
83      // g2o::OptimizationAlgorithmGaussNewton* solver = new g2o::
        OptimizationAlgorithmGaussNewton( solver_ptr );
84      // g2o::OptimizationAlgorithmDogleg* solver = new g2o::
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
    OptimizationAlgorithmDogleg( solver_ptr );
    g2o::SparseOptimizer optimizer;     // 图模型
    optimizer.setAlgorithm( solver );   // 设置求解器
    optimizer.setVerbose( true );       // 打开调试输出

    // 向图中增加顶点
    CurveFittingVertex* v = new CurveFittingVertex();
    v->setEstimate( Eigen::Vector3d(0,0,0) );
    v->setId(0);
    optimizer.addVertex( v );

    // 向图中增加边
    for ( int i=0; i<N; i++ )
    {
        CurveFittingEdge* edge = new CurveFittingEdge( x_data[i] );
        edge->setId(i);
        edge->setVertex( 0, v );                // 设置连接的顶点
        edge->setMeasurement( y_data[i] );      // 观测数值
        // 信息矩阵：协方差矩阵之逆
        edge->setInformation( Eigen::Matrix<double,1,1>::Identity()*1/(w_sigma*
        w_sigma) );
        optimizer.addEdge( edge );
    }

    // 执行优化
    cout<<"start optimization"<<endl;
    chrono::steady_clock::time_point t1 = chrono::steady_clock::now();
    optimizer.initializeOptimization();
    optimizer.optimize(100);
    chrono::steady_clock::time_point t2 = chrono::steady_clock::now();
    chrono::duration<double> time_used = chrono::duration_cast<chrono::duration<
    double>>( t2-t1 );
    cout<<"solve time cost = "<<time_used.count()<<" seconds. "<<endl;

    // 输出优化值
    Eigen::Vector3d abc_estimate = v->estimate();
    cout<<"estimated model: "<<abc_estimate.transpose()<<endl;

    return 0;
}
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

在这个程序中，我们从g2o派生出了用于曲线拟合的图优化顶点和边：CurveFittingVertex和CurveFittingEdge，这实质上扩展了g2o的使用方式。在这两个派生类中，我们重写了重要的虚函数：

1.顶点的更新函数：oplusImpl。我们知道优化过程最重要的是增量Δx的计算，而该函数处理的是$x_{k+1}=x_k+\Delta x$的过程。读者也许觉得这并不是什么值得一提的事情，因为仅仅是个简单的加法而已，为什么g2o不帮我们完成呢？在曲线拟合过程中，由于优化变量（曲线参数）本身位于向量空间中，这个更新计算确实就是简单的加法。但是，当优化变量不在向量空间中时，比如说x是相机位姿，它本身不一定有加法运算。这时，就需要重新定义增量如何加到现有的估计上的行为了。按照第4讲的解释，我们可能使用左乘更新或右乘更新，而不是直接的加法。

2.顶点的重置函数：setToOriginImpl。这是平凡的，我们把估计值置零即可。

3.边的误差计算函数：computeError。该函数需要取出边所连接的顶点的当前估计值，根据曲线模型，与它的观测值进行比较。这和最小二乘问题中的误差模型是一致的。

4.存盘和读盘函数：read、write。由于我们并不想进行读/写操作，所以留空。

定义了顶点和边之后，我们在main函数里声明了一个图模型，然后按照生成的噪声数据，往图模型中添加顶点和边，最后调用优化函数进行优化。g2o会给出优化的结果：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
% build/curve_fitting
generating data:
0 2.71828
0.01 2.93161
0.02 2.12942
......
iteration= 13 chi2= 101.937020 time= 4.06e-05    cumTime= 0.00048135  edges= 100
schur= 0 lambda= 3678.088107 levenbergIter= 6
iteration= 14 chi2= 101.937020 time= 3.2215e-05  cumTime= 0.000513565 edges= 100
schur= 0 lambda= 19616.469906 levenbergIter= 3
iteration= 15 chi2= 101.937020 time= 0.000108524 cumTime= 0.000622089 edges= 100
schur= 0 lambda= 836969.382664 levenbergIter= 4
iteration= 16 chi2= 101.937020 time= 0.000159817 cumTime= 0.000781906 edges= 100
schur= 0 lambda= 224672257893341.656250 levenbergIter= 7
solve time cost = 0.00173976 seconds.
estimated model: 0.890911   2.1719 0.943629
```

我们使用列文伯格—马夸尔特方法进行梯度下降，在迭代了16次后，最后优化结果与Ceres实验中相差无几。我们也在程序中提供了使用高斯牛顿法和DogLeg下降方式，请读者去掉它们前面的注释符号，自行对比各种梯度下降方法的差异。

书籍汇总下载 http://t.cn/RDIAj5D

6.5 小结

本节介绍了SLAM中经常碰到的一种非线性优化问题：由许多个误差项平方和组成的最小二乘问题。我们介绍了它的定义和求解，并且讨论了两种主要的梯度下降方式：高斯牛顿法和列文伯格—马夸尔特方法。在实践部分中，分别使用了Ceres和g2o两种优化库求解同一个曲线拟合问题，发现它们给出了相似的结果。

由于还没有详细谈Bundle Adjustment，所以实践部分选择了曲线拟合这样一个简单但有代表性的例子，以演示一般的非线性最小二乘求解方式。特别地，如果用g2o来拟合曲线，必须先把问题转换为图优化，定义新的顶点和边，这种做法是有一些迂回的——g2o的主要目的并不在此。相比之下，Ceres定义误差项求曲线拟合问题则自然了很多，因为它本身即是一个优化库。然而，在SLAM中更多的问题是，一个带有许多个相机位姿和许多个空间点的优化问题如何求解。特别地，当相机位姿以李代数表示时，误差项关于相机位姿的导数如何计算，将是一件值得详细讨论的事。我们将在后续内容发现，g2o提供了大量的顶点和边的类型，非常便于相机位姿估计问题。而在Ceres中，我们不得不自己实现每一个Cost Function，有一些不便。

在实践部分的两个程序中，我们没有去计算曲线模型关于三个参数的导数，而是利用了优化库的数值求导，这使得理论和代码都会简洁一些。Ceres库提供了基于模板元的自动求导和运行时的数值求导，而g2o只提供了运行时数值求导这一种方式。但是，对于大多数问题，如果能够推导出雅可比矩阵的解析形式并告诉优化库，就可以避免数值求导中的诸多问题。

最后，希望读者能够适应Ceres和g2o这些大量使用模板编程的方式。也许一开始会看上去比较吓人（特别是Ceres设置Problem和g2o初始化部分的代码），但是熟悉之后，就会觉得这样的方式是自然的，而且容易扩展。我们将在SLAM后端一讲中继续讨论稀疏性、核函数、位姿图（Pose Graph）等问题。

习题

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

1.证明线性方程$Ax=b$当系数矩阵A超定时，最小二乘解为$x=(A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}b$。

2.调研最速下降法、牛顿法、高斯牛顿法和列文伯格—马夸尔特方法各有什么优缺点。除了我们举的Ceres库和g2o库，还有哪些常用的优化库？（你可能会找到一些MATLAB上的库。）

3.为什么高斯牛顿法的增量方程系数矩阵可能不正定？不正定有什么几何含义？为什么在这种情况下解就不稳定了？

4.DogLeg是什么？它与高斯牛顿法和列文伯格—马夸尔特方法有何异同？请搜索相关的材料[3]。

5.阅读Ceres的教学材料（http://ceres-solver.org/tutorial.html）以更好地掌握其用法。

6.阅读g2o自带的文档，你能看懂它吗？如果还不能完全看懂，请在第10讲和第11讲之后回来再看。

7.*请更改曲线拟合实验中的曲线模型，并用Ceres和g2o进行优化实验。例如，可以使用更多的参数和更复杂的模型。

---

[1]核函数的详细讨论见第10讲。

[2]虽然笔者个人并不太喜欢有些故弄玄虚的说法。

[3]例如，
http://www.numerical.rl.ac.uk/people/nimg/course/lectures/raphael/lectures/lec7slides.pdf。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

第7讲　视觉里程计1

主要目标

1.理解图像特征点的意义,并掌握在单幅图像中提取出特征点及多幅图像中匹配特征点的方法。

2.理解对极几何的原理，利用对极几何的约束，恢复出图像之间的摄像机的三维运动。

3.理解PNP问题，以及利用已知三维结构与图像的对应关系求解摄像机的三维运动。

4.理解ICP问题，以及利用点云的匹配关系求解摄像机的三维运动。

5.理解如何通过三角化获得二维图像上对应点的三维结构。

本书前面介绍了运动方程和观测方程的具体形式，并讲解了以非线性优化为主的求解方法。从本讲开始，我们结束基础知识的铺垫而步入正题：按照第2讲的内容，分别介绍视觉里程计、优化后端、回环检测和地图构建4个模块。本讲和下一讲主要介绍作为视觉里程计的主要理论，然后在第9讲中进行一次实践。本讲关注基于特征点方式的视觉里程计算法。我们将介绍什么是特征点、如何提取和匹配特征点，以及如何根据配对的特征点估计相机运动。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D



电子书寻找看手相qq37391319 钉钉或微信 pythontesting
书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

7.1 特征点法

回顾第2讲的内容，我们说过视觉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，给后端提供较好的初始值。VO的实现方法，按是否需要提取特征，分为特征点法的前端及不提特征的直接法前端。基于特征点法的前端，长久以来（直到现在）被认为是视觉里程计的主流方法。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。在本讲中，我们将从特征点法入手，学习如何提取、匹配图像特征点，然后估计两帧之间的相机运动和场景结构，从而实现一个基本的两帧间视觉里程计。

7.1.1 特征点

VO的主要问题是如何根据图像来估计相机运动。然而，图像本身是一个由亮度和色彩组成的矩阵，如果直接从矩阵层面考虑运动估计，将会非常困难。所以，我们习惯于采用这样一种做法：首先，从图像中选取比较有代表性的点。这些点在相机视角发生少量变化后会保持不变，所以我们会在各个图像中找到相同的点。然后，在这些点的基础上，讨论相机位姿估计问题，以及这些点的定位问题。在经典SLAM模型中，把它们称为路标。而在视觉SLAM中，路标则是指图像特征（Feature）。

根据维基百科的定义，图像特征是一组与计算任务相关的信息，计算任务取决于具体的应用[29]。简而言之，特征是图像信息的另一种数字表达形式。一组好的特征对于在指定任务上的最终表现至关重要，所以多年来研究者们花费了大量的精力对特征进行研究。数字图像在计算机中以灰度值矩阵的方式存储，所以最简单的，单个图像像素也是一种“特征”。但是，在视觉里程计中，我们希望特征点在相机运动之后保持稳定，而灰度值受光照、形变、物体材质的影响严重，在不同图像间变化非常大，不够稳定。理想的情况是，当场景和相机视角发生少量改变时，还能从图像中判断哪些地方是同一个点，因此仅凭灰度值是不够

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

的，我们需要对图像提取特征点。

特征点是图像里一些特别的地方。以图7-1为例。我们可以把图像中的角点、边缘和区块都当成图像中有代表性的地方。不过，我们更容易精确地指出，某两幅图像中出现了同一个角点；同一个边缘则稍微困难一些，因为沿着该边缘前进，图像局部是相似的；同一个区块则是最困难的。我们发现，图像中的角点、边缘相比于像素区块而言更加“特别”，在不同图像之间的辨识度更强。所以，一种直观的提取特征的方式就是在不同图像间辨认角点，确定它们的对应关系。在这种做法中，角点就是所谓的特征。

然而，在大多数应用中，单纯的角点依然不能满足我们的很多需求。例如，从远处看上去是角点的地方，当相机走近之后，可能就不显示为角点了。或者，当旋转相机时，角点的外观会发生变化，我们也就不容易辨认出那是同一个角点。为此，计算机视觉领域的研究者们在长年的研究中设计了许多更加稳定的局部图像特征，如著名的SIFT[30]、SURF[31]、ORB[32]，等等。相比于朴素的角点，这些人工设计的特征点能够拥有如下的性质：

1.可重复性（Repeatability）：相同的“区域”可以在不同的图像中找到。

2.可区别性（Distinctiveness）：不同的“区域”有不同的表达。

3.高效率（E ffi ciency）：同一图像中，特征点的数量应远小于像素的数量。

4.本地性（Locality）：特征仅与一小片图像区域相关。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

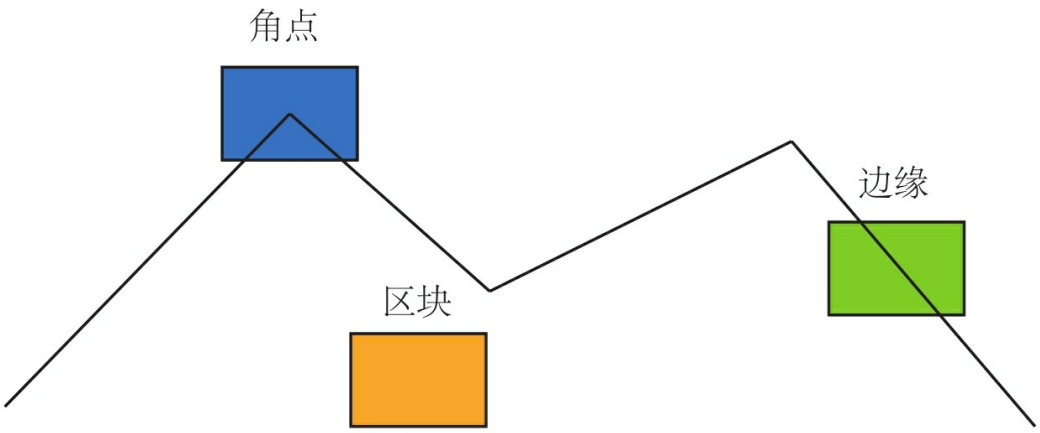


图7-1　可以作为图像特征的部分：角点、边缘、区块

特征点由关键点（Key-point）和描述子（Descriptor）两部分组成。比如，当谈论SIFT特征时，是指“提取SIFT关键点，并计算SIFT描述子”两件事情。关键点是指该特征点在图像里的位置，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，描述了该关键点周围像素的信息。描述子是按照“外观相似的特征应该有相似的描述子”的原则设计的。因此，只要两个特征点的描述子在向量空间上的距离相近，就可以认为它们是同样的特征点。

历史上，研究者们提出过许多图像特征。它们有些很精确，在相机的运动和光照变化下仍具有相似的表达，但相应地需要较大的计算量。其中，SIFT（尺度不变特征变换，Scale-Invariant Feature Transform）当属最为经典的一种。它充分考虑了在图像变换过程中出现的光照、尺度、旋转等变化，但随之而来的是极大的计算量。由于整个SLAM过程中图像特征的提取与匹配仅仅是诸多环节中的一个，到目前（2016年）为止，普通PC的CPU还无法实时地计算SIFT特征，进行定位与建图。所以在SLAM中我们甚少使用这种“奢侈”的图像特征。

另一些特征，则考虑适当降低精度和健壮性，以提升计算的速度。例如，FAST关键点属于计算特别快的一种特征点（注意这里“关键点”的表述，说明它没有描述子）。而ORB（Oriented FAST and Rotated BRIEF）特征则是目前看来非常具有代表性的实时图像特征。它改进了FAST检测子[33]不具有方向性的问题，并采用速度极快的二进制描述子BRIEF[34]，使整个图像特征提取的环节大大加速。根据作者在论文中所述测试，在同一幅图像中同时提取约1000个特征点的情况下，ORB约要

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

花费15.3ms，SURF约花费217.3ms，SIFT约花费5228.7ms。由此可以看出，ORB在保持了特征子具有旋转、尺度不变性的同时，速度方面提升明显，对于实时性要求很高的SLAM来说是一个很好的选择。

大部分特征提取都具有较好的并行性，可以通过GPU等设备来加速计算。经过GPU加速后的SIFT，就可以满足实时计算要求。但是，引入GPU将带来整个SLAM成本的提升。由此带来的性能提升是否足以抵去付出的计算成本，需要系统的设计人员仔细考量。在目前的SLAM方案中，ORB是质量与性能之间较好的折中，因此，我们以ORB为代表介绍提取特征的整个过程。

7.1.2 ORB特征

ORB特征亦由关键点和描述子两部分组成。它的关键点称为“Oriented FAST”，是一种改进的FAST角点，关于什么是FAST角点我们将在下文介绍。它的描述子称为BRIEF（Binary Robust Independent Elementary Feature）。因此，提取ORB特征分为如下两个步骤：

1.FAST角点提取：找出图像中的“角点”。相较于原版的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特性。

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。

下面分别介绍FAST和BRIEF。

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：如果一个像素与邻域的像素差别较大（过亮或过暗），那么它更可能是角点。相比于其他角点检测算法，FAST只需比较像素亮度的大小，十分快捷。它的检测过程如下（见图7-2）：

1.在图像中选取像素p，假设它的亮度为I_p。

2.设置一个阈值T（比如，I_p的20%）。

3.以像素p为中心，选取半径为3的圆上的16个像素点。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

4.假如选取的圆上有连续的N个点的亮度大于I_p+T或小于I_p-T，那么像素p可以被认为是特征点（N通常取12，即为FAST-12。其他常用的N取值为9和11，它们分别被称为FAST-9和FAST-11）。

5.循环以上四步，对每一个像素执行相同的操作。

在FAST-12算法中，为了更高效，可以添加一项预测试操作，以快速地排除绝大多数不是角点的像素。具体操作为，对于每个像素，直接检测邻域圆上的第1,5,9,13个像素的亮度。只有当这4个像素中有3个同时大于I_p+T或小于I_p-T时，当前像素才有可能是一个角点，否则应该直接排除。这样的预测试操作大大加速了角点检测。此外，原始的FAST角点经常出现“扎堆”的现象。所以在第一遍检测之后，还需要用非极大值抑制（Non-maximal suppression），在一定区域内仅保留响应极大值的角点，避免角点集中的问题。

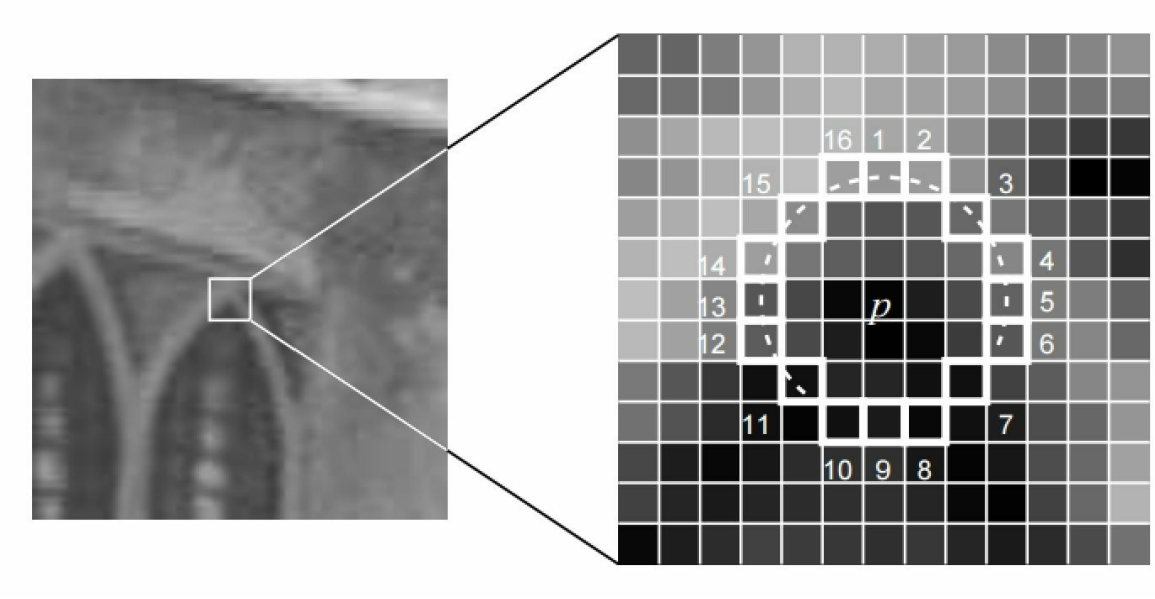


图7-2　FAST特征点[33]。

FAST特征点的计算仅仅是比较像素间亮度的差异，速度非常快，但它也有一些问题。首先，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算法进行了改进。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响应值，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。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

其次，FAST角点不具有方向信息。而且，由于它固定取半径为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弱点，ORB添加了尺度和旋转的描述。尺度不变性由构建图像金字塔[1]，并在金字塔的每一层上检测角点来实现。而特征的旋转是由灰度质心法（Intensity Centroid）实现的。我们稍微介绍一下。

所谓质心是指以图像块灰度值作为权重的中心。其具体操作步骤如下[35]：

1.在一个小的图像块B中，定义图像块的矩为

$$m_{pq} = \sum_{x,y \in B} x^p y^q I(x,y), \quad p,q = \{0,1\}.$$

2.通过矩可以找到图像块的质心：

$$C = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}).$$

3.连接图像块的几何中心O与质心C，得到一个方向向量$\overrightarrow{OC}$，于是特征点的方向可以定义为

$$\theta = \arctan(m_{01}/m_{10}).$$

通过以上方法，FAST角点便具有了尺度与旋转的描述，从而大大提升了其在不同图像之间表述的健壮性。所以在ORB中，把这种改进后的FAST称为Oriented FAST。

BRIEF描述子

在提取Oriented FAST关键点后，我们对每个点计算其描述子。ORB使用改进的BRIEF特征描述。我们先来介绍一下BRIEF是什么。

BRIEF是一种二进制描述子，其描述向量由许多个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。如果我们取了128个这样的p,q，最后就得到128维由0、1组成的向量。那么，p和q如何选取呢？在作者原始的论文

书籍汇总下载 http://t.cn/RDIAj5D

中给出了若干种挑选方法，大体上都是按照某种概率分布，随机地挑选p和q的位置，读者可以阅读BRIEF论文或OpenCV源码以查看其具体实现[34]。BRIEF使用了随机选点的比较，速度非常快，而且由于使用了二进制表达，存储起来也十分方便，适用于实时的图像匹配。原始的BRIEF描述子不具有旋转不变性，因此在图像发生旋转时容易丢失。而ORB在FAST特征点提取阶段计算了关键点的方向，所以可以利用方向信息，计算了旋转之后的“Steer BRIEF”特征使ORB的描述子具有较好的旋转不变性。

由于考虑到了旋转和缩放，使得ORB在平移、旋转和缩放的变换下仍有良好的表现。同时，FAST和BREIF的组合也非常高效，使得ORB特征在实时SLAM中非常受欢迎。我们在图7-3中展示了一张图像提取ORB之后的结果，下面来介绍如何在不同的图像之间进行特征匹配。

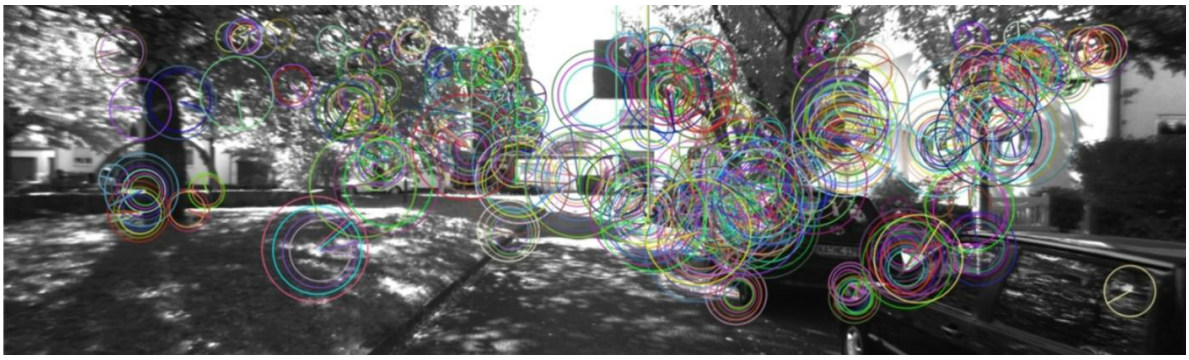


图7-3　OpenCV提供的ORB特征点检测结果。

7.1.3　特征匹配

特征匹配（如图7-4所示）是视觉SLAM中极为关键的一步，宽泛地说，特征匹配解决了SLAM中的数据关联问题（dataassociation），即确定当前看到的路标与之前看到的路标之间的对应关系。通过对图像与图像或者图像与地图之间的描述子进行准确匹配，我们可以为后续的姿态估计、优化等操作减轻大量负担。然而，由于图像特征的局部特性，误匹配的情况广泛存在，而且长期以来一直没有得到有效解决，目前已经成为视觉SLAM中制约性能提升的一大瓶颈。部分原因是场景中经常存在大量的重复纹理，使得特征描述非常相似。在这种情况下，仅利用局部特征解决误匹配是非常困难的。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

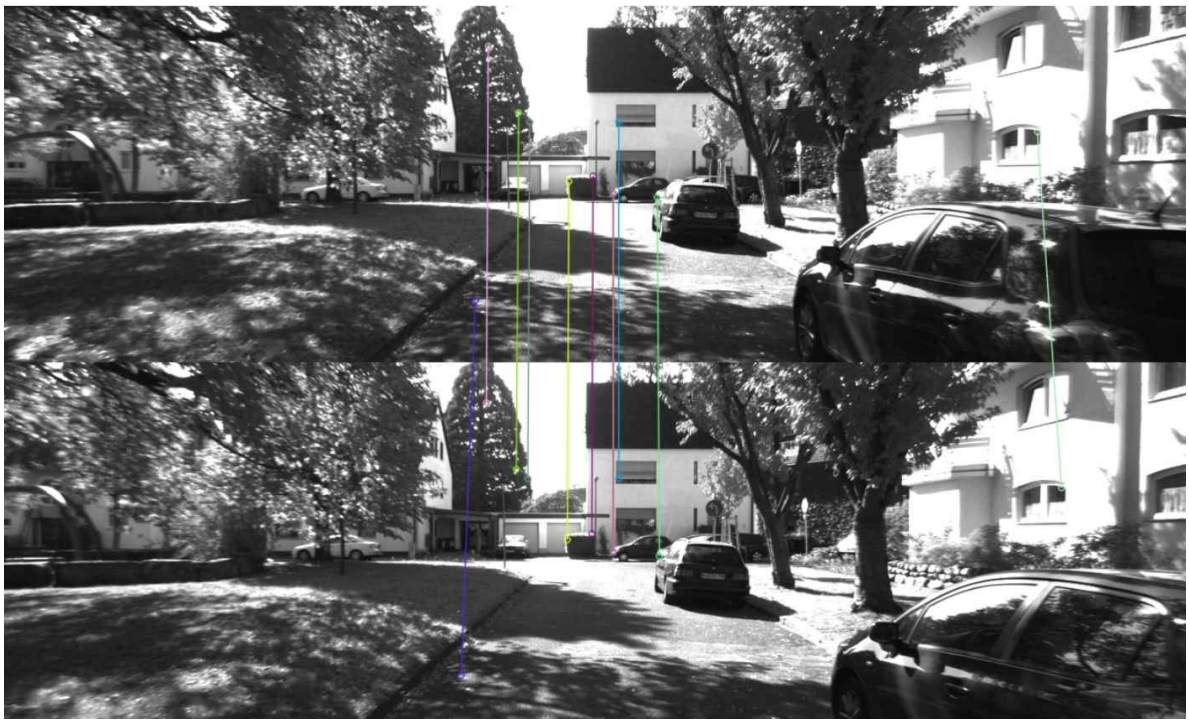


图7-4　两帧图像间的特征匹配。

不过，让我们先来看正确匹配的情况，等做完实验再回头去讨论误匹配问题。考虑两个时刻的图像。如果在图像I_t中提取到特征点$x_t^m, m = 1,2,...,M$，在图像I_{t+1}中提取到特征点$x_{t+1}^n, n = 1,2,...,N$，如何寻找这两个集合元素的对应关系呢？最简单的特征匹配方法就是暴力匹配（Brute-Force Matcher）。即对每一个特征点x_t^m与所有的x_{t+1}^n测量描述子的距离，然后排序，取最近的一个作为匹配点。描述子距离表示了两个特征之间的相似程度，不过在实际运用中还可以取不同的距离度量范数。对于浮点类型的描述子，使用欧氏距离进行度量即可。而对于二进制的描述子（比如BRIEF这样的），我们往往使用汉明距离（Hamming distance）作为度量——两个二进制串之间的汉明距离，指的是其不同位数的个数。

然而，当特征点数量很大时，暴力匹配法的运算量将变得很大，特别是当想要匹配某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（FLANN）算法更加适合于匹配点数量极多的情况。由于这些匹配算法理论已经成熟，而且实现上也已集成到OpenCV，所以这里就不再描述它的技术细节了。感兴趣的读者可以参考阅读文献[36]。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

7.2 实践：特征提取和匹配



图7-5 实验使用的两帧图像。

目前主流的几种图像特征在OpenCV开源图像库中都已经集成，我们可以很方便地进行调用。下面就来实际练习一下OpenCV的图像特征提取、计算和匹配的过程。我们为此实验准备了两张图像，位于slambook/ch7/下的1.png和2.png，如图7-5所示。它们是来自公开数据集[37]中的两张图像，我们看到相机发生了微小的运动。本节程序演示如何提取ORB特征并进行匹配。下一个程序将演示如何估计相机运动。

特征提取与匹配代码：

slambook/ch7/feature_extraction.cpp

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/features2d/features2d.hpp>
#include <opencv2/highgui/highgui.hpp>

using namespace std;
using namespace cv;

int main ( int argc, char** argv )
{
    if ( argc != 3 )
    {
        cout<<"usage: feature_extraction img1 img2"<<endl;
        return 1;
    }
    //-- 读取图像
```

书籍汇总下载 http://t.cn/RDIAj5D

```
17      Mat img_1 = imread ( argv[1], CV_LOAD_IMAGE_COLOR );
18      Mat img_2 = imread ( argv[2], CV_LOAD_IMAGE_COLOR );
19
20      //-- 初始化
21      std::vector<KeyPoint> keypoints_1, keypoints_2;
22      Mat descriptors_1, descriptors_2;
23      Ptr<ORB> orb = ORB::create ( 500, 1.2f, 8, 31, 0, 2, ORB::HARRIS_SCORE,31,20 );
24
25      //-- 第1步：检测 Oriented FAST 角点位置
26      orb->detect ( img_1,keypoints_1 );
27      orb->detect ( img_2,keypoints_2 );
28
29      //-- 第2步：根据角点位置计算 BRIEF 描述子
30      orb->compute ( img_1, keypoints_1, descriptors_1 );
31      orb->compute ( img_2, keypoints_2, descriptors_2 );
32
33      Mat outimg1;
34      drawKeypoints( img_1, keypoints_1, outimg1, Scalar::all(-1), DrawMatchesFlags::
        DEFAULT );
35      imshow("ORB特征点",outimg1);
36
37      //-- 第3步：对两幅图像中的BRIEF描述子进行匹配，使用 Hamming 距离
38      vector<DMatch> matches;
39      BFMatcher matcher ( NORM_HAMMING );
40      matcher.match ( descriptors_1, descriptors_2, matches );
41
42      //-- 第4步：匹配点对筛选
43      double min_dist=10000, max_dist=0;
44
45      // 找出所有匹配之间的最小距离和最大距离，即最相似的和最不相似的两组点之间的距离
46      for ( int i = 0; i < descriptors_1.rows; i++ )
47      {
48          double dist = matches[i].distance;
49          if ( dist < min_dist ) min_dist = dist;
50          if ( dist > max_dist ) max_dist = dist;
51      }
52
53      printf ( "-- Max dist : %f \n", max_dist );
54      printf ( "-- Min dist : %f \n", min_dist );
55
56      // 当描述子之间的距离大于两倍的最小距离时，即认为匹配有误。
57      // 但有时候最小距离会非常小，设置一个经验值作为下限。
58      std::vector< DMatch > good_matches;
59      for ( int i = 0; i < descriptors_1.rows; i++ )
60      {
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
61          if ( matches[i].distance <= max ( 2*min_dist, 30.0 ) )
62          {
63              good_matches.push_back ( matches[i] );
64          }
65      }
66
67      //-- 第5步：绘制匹配结果
68      Mat img_match;
69      Mat img_goodmatch;
70      drawMatches ( img_1, keypoints_1, img_2, keypoints_2, matches, img_match );
71      drawMatches ( img_1, keypoints_1, img_2, keypoints_2, good_matches,
        img_goodmatch );
72      imshow ( "所有匹配点对", img_match );
73      imshow ( "优化后匹配点对", img_goodmatch );
74      waitKey(0);
75
76      return 0;
77  }
```

运行此程序（需要输入两个图像位置），将输出运行结果：

```
1  % build/feature_extraction 1.png 2.png
2  -- Max dist : 95.000000
3  -- Min dist : 4.000000
```

图7-6显示了例程的运行结果。我们看到未筛选的匹配中带有大量的误匹配。经过一次筛选之后，匹配数量减少了许多，但大多数匹配都是正确的。这里，筛选的依据是汉明距离小于最小距离的两倍，这是一种工程上的经验方法，不一定有理论依据。不过，尽管在示例图像中能够筛选出正确的匹配，但我们仍然不能保证在所有其他图像中得到的匹配都是正确的。因此，在后面的运动估计中，还需要使用去除误匹配的算法。

接下来，我们希望根据匹配的点对估计相机的运动。这里由于相机的原理不同，情况发生了变化：

书籍汇总下载 http://t.cn/RDIAj5D

1.当相机为单目时，我们只知道2D的像素坐标，因而问题是根据两组2D点估计运动。该问题用对极几何来解决。

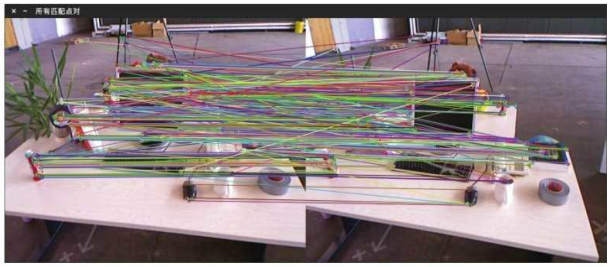
2.当相机为双目、RGB-D时，或者通过某种方法得到了距离信息，那么问题就是根据两组3D点估计运动。该问题通常用ICP来解决。

3.如果有3D点及其在相机的投影位置，也能估计相机的运动。该问题通过PnP求解。

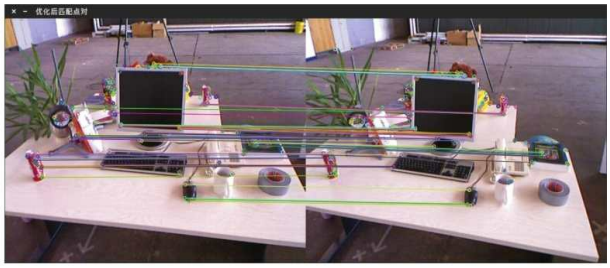
因此，下面几节就来介绍这三种情形下的相机运动估计。我们将从最基本的2D-2D情形出发，看看它如何求解，求解过程又具有哪些麻烦的问题。



ORB关键点



未筛选的匹配



筛选后的匹配

图7-6　特征提取与匹配结果。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

7.3 2D-2D：对极几何

7.3.1 对极约束

现在，假设我们从两张图像中得到了一对配对好的特征点，如图7-7所示。如果有若干对这样的匹配点，就可以通过这些二维图像点的对应关系，恢复出在两帧之间摄像机的运动。这里“若干对”具体是多少对呢？我们会在下文介绍。下面先来看看两个图像当中的匹配点有什么几何关系。

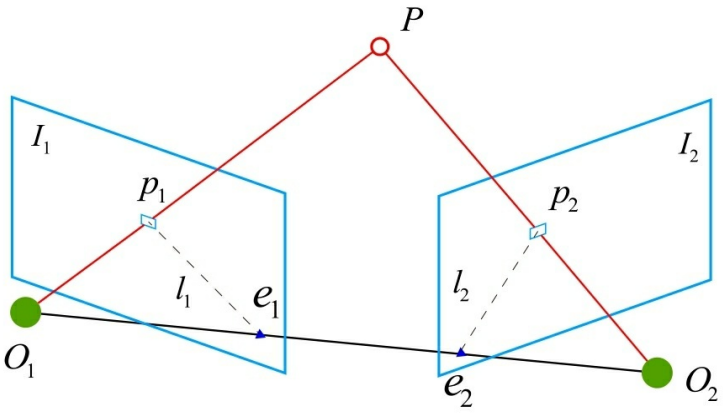


图7-7 对极几何约束。

以图7-7为例，我们希望求取两帧图像I_1,I_2之间的运动，设第一帧到第二帧的运动为R,t。两个相机中心分别为O_1,O_2。现在，考虑I_1中有一个特征点p_1，它在I_2中对应着特征点p_2。我们知道两者是通过特征匹配得到的。如果匹配正确，说明它们确实是同一个空间点在两个成像平面上的投影。这里需要一些术语来描述它们之间的几何关系。首先，连线$\overrightarrow{O_1p_1}$和连线$\overrightarrow{O_2p_2}$在三维空间中会相交于点P。这时候点O_1,O_2,P三个点可以确定一个平面，称为极平面（Epipolar plane）。O_1O_2连线与像平面I_1,I_2的交点分别为e_1,e_2。e_1,e_2称为极点（Epipoles），O_1O_2被称为基线（Baseline）。我们称极平面与两个像平面I_1,I_2之间的相交线l_1,l_2为极线

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

（Epipolar line）。

直观讲，从第一帧的角度看，射线$\overrightarrow{O_1p_1}$是某个像素可能出现的空间位置——因为该射线上的所有点都会投影到同一个像素点。同时，如果不知道P的位置，那么当我们在第二幅图像上看时，连线$\overrightarrow{e_2p_2}$（也就是第二幅图像中的极线）就是P可能出现的投影的位置，也就是射线$\overrightarrow{O_1p_1}$在第二个相机中的投影。现在，由于我们通过特征点匹配确定了p_2的像素位置，所以能够推断P的空间位置，以及相机的运动。要提醒读者的是，这完全多亏了正确的特征匹配。如果没有特征匹配，我们就没法确定p_2到底在极线的哪个位置了。那时，就必须在极线上搜索以获得正确的匹配，这将在第13讲中提到。

现在，我们从代数角度来看一下这里出现的几何关系。在第一帧的坐标系下，设P的空间位置为

$$\boldsymbol{P} = [X, Y, Z]^{\mathrm{T}}.$$

根据第5讲介绍的针孔相机模型，我们知道两个像素点p_1,p_2的像素位置为

$$s_1\boldsymbol{p}_1 = \boldsymbol{K}\boldsymbol{P}, \quad s_2\boldsymbol{p}_2 = \boldsymbol{K}(\boldsymbol{R}\boldsymbol{P} + \boldsymbol{t}). \tag{7.1}$$

这里K为相机内参矩阵，R,t为两个坐标系的相机运动（如果我们愿意，也可以写成李代数形式）。如果使用齐次坐标，也可以把上式写成在乘以非零常数下成立的（up to a scale）等式[2]：

$$\boldsymbol{p}_1 = \boldsymbol{K}\boldsymbol{P}, \quad \boldsymbol{p}_2 = \boldsymbol{K}(\boldsymbol{R}\boldsymbol{P} + \boldsymbol{t}). \tag{7.2}$$

现在，取：

$$\boldsymbol{x}_1 = \boldsymbol{K}^{-1}\boldsymbol{p}_1, \quad \boldsymbol{x}_2 = \boldsymbol{K}^{-1}\boldsymbol{p}_2. \tag{7.3}$$

这里的x_1,x_2是两个像素点的归一化平面上的坐标。代入上式，得：

$$\boldsymbol{x}_2 = \boldsymbol{R}\boldsymbol{x}_1 + \boldsymbol{t}. \tag{7.4}$$

两边同时左乘$t^\wedge$。回忆$^\wedge$的定义，这相当于两侧同时与t做外积：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$\boldsymbol{t}^{\wedge}\boldsymbol{x}_2 = \boldsymbol{t}^{\wedge}\boldsymbol{R}\boldsymbol{x}_1. \tag{7.5}$$

然后，两侧同时左乘$\boldsymbol{x}_2^{\mathrm{T}}$：

$$\boldsymbol{x}_2^{\mathrm{T}}\boldsymbol{t}^{\wedge}\boldsymbol{x}_2 = \boldsymbol{x}_2^{\mathrm{T}}\boldsymbol{t}^{\wedge}\boldsymbol{R}\boldsymbol{x}_1. \tag{7.6}$$

观察等式左侧，$t^{\wedge}x_2$是一个与t和x_2都垂直的向量。把它再和x_2做内积时，将得到0。因此，我们就得到了一个简洁的式子：

$$\boldsymbol{x}_2^{\mathrm{T}}\boldsymbol{t}^{\wedge}\boldsymbol{R}\boldsymbol{x}_1 = 0. \tag{7.7}$$

重新代入p_1,p_2，有：

$$\boldsymbol{p}_2^{\mathrm{T}}\boldsymbol{K}^{-\mathrm{T}}\boldsymbol{t}^{\wedge}\boldsymbol{R}\boldsymbol{K}^{-1}\boldsymbol{p}_1 = 0. \tag{7.8}$$

这两个式子都称为对极约束，它以形式简洁著名。它的几何意义是O_1,P,O_2三者共面。对极约束中同时包含了平移和旋转。我们把中间部分记作两个矩阵：基础矩阵（Fundamental Matrix）F和本质矩阵（Essential Matrix）E，于是可以进一步简化对极约束：

$$\boldsymbol{E} = \boldsymbol{t}^{\wedge}\boldsymbol{R}, \quad \boldsymbol{F} = \boldsymbol{K}^{-\mathrm{T}}\boldsymbol{E}\boldsymbol{K}^{-1}, \quad \boldsymbol{x}_2^{\mathrm{T}}\boldsymbol{E}\boldsymbol{x}_1 = \boldsymbol{p}_2^{\mathrm{T}}\boldsymbol{F}\boldsymbol{p}_1 = 0. \tag{7.9}$$

对极约束简洁地给出了两个匹配点的空间位置关系。于是，相机位姿估计问题变为以下两步：

1.根据配对点的像素位置求出E或者F。

2.根据E或者F求出R,t。

由于E和F只相差了相机内参，而内参在SLAM中通常是已知的[3]，所以实践当中往往使用形式更简单的E。我们以E为例，介绍上面两个问题如何求解。

7.3.2 本质矩阵

根据定义，本质矩阵$E=t^{\wedge}R$。它是一个3×3的矩阵，内有9个未知

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

数。那么，是不是任意一个3×3的矩阵都可以被当成本质矩阵呢？从E的构造方式上看，有以下值得注意的地方：

•本质矩阵是由对极约束定义的。由于对极约束是等式为零的约束，所以对E乘以任意非零常数后，对极约束依然满足。我们把这件事情称为E在不同尺度下是等价的。

•根据$E=t^\wedge R$，可以证明[3]，本质矩阵E的奇异值必定是$[\sigma,\sigma,0]^\mathrm{T}$的形式。这称为本质矩阵的内在性质。

•另一方面，由于平移和旋转各有3个自由度，故$t^\wedge R$共有6个自由度。但由于尺度等价性，故E实际上有5个自由度。

E具有5个自由度的事实，表明我们最少可以用5对点来求解E。但是，E的内在性质是一种非线性性质，在求解线性方程时会带来麻烦，因此，也可以只考虑它的尺度等价性，使用8对点来估计E——这就是经典的八点法（Eight-point-algorithm）[38,39]。八点法只利用了E的线性性质，因此可以在线性代数框架下求解。下面我们来看八点法是如何工作的。

考虑一对匹配点，它们的归一化坐标为$x_1=[u_1,v_1,1]^\mathrm{T}$,$x_2=[u_2,v_2,1]^\mathrm{T}$。根据对极约束，有：

$$\begin{pmatrix} u_1, v_1, 1 \end{pmatrix} \begin{pmatrix} e_1 & e_2 & e_3 \\ e_4 & e_5 & e_6 \\ e_7 & e_8 & e_9 \end{pmatrix} \begin{pmatrix} u_2 \\ v_2 \\ 1 \end{pmatrix} = 0. \tag{7.10}$$

我们把矩阵E展开，写成向量的形式：

$$\boldsymbol{e} = [e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9]^\mathrm{T},$$

那么对极约束可以写成与$\boldsymbol{e}$有关的线性形式：

$$[u_1u_2, u_1v_2, u_1, v_1u_2, v_1v_2, v_1, u_2, v_2, 1] \cdot \boldsymbol{e} = 0. \tag{7.11}$$

同理，对于其他点对也有相同的表示。我们把所有点都放到一个方

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

程中，变成线性方程组（u^i,v^i表示第i个特征点，以此类推）：

$$\begin{pmatrix} u_1^1u_2^1 & u_1^1v_2^1 & u_1^1 & v_1^1u_2^1 & v_1^1v_2^1 & v_1^1 & u_2^1 & v_2^1 & 1 \\ u_1^2u_2^2 & u_1^2v_2^2 & u_1^2 & v_1^2u_2^2 & v_1^2v_2^2 & v_1^2 & u_2^2 & v_2^2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ u_1^8u_2^8 & u_1^8v_2^8 & u_1^8 & v_1^8u_2^8 & v_1^8v_2^8 & v_1^8 & u_2^8 & v_2^8 & 1 \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \\ e_7 \\ e_8 \\ e_9 \end{pmatrix} = 0. \tag{7.12}$$

这8个方程构成了一个线性方程组。它的系数矩阵由特征点位置构成，大小为8×9。e位于该矩阵的零空间中。如果系数矩阵是满秩的（即秩为8），那么它的零空间维数为1，也就是e构成一条线。这与e的尺度等价性是一致的。如果8对匹配点组成的矩阵满足秩为8的条件，那么E的各元素就可由上述方程解得。

接下来的问题是如何根据已经估得的本质矩阵E，恢复出相机的运动R,t。这个过程是由奇异值分解（SVD）得到的。设E的SVD分解为

$$\boldsymbol{E} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}, \tag{7.13}$$

其中U,V为正交阵，Σ为奇异值矩阵。根据E的内在性质，我们知道$\Sigma=\mathrm{diag}(\sigma,\sigma,0)$。在SVD分解中，对于任意一个$E$，存在两个可能的$t$,$R$与它对应：

$$\begin{aligned} \boldsymbol{t}_1^\wedge &= \boldsymbol{U}\boldsymbol{R}_Z\left(\tfrac{\pi}{2}\right)\boldsymbol{\Sigma}\boldsymbol{U}^{\mathrm{T}}, \quad \boldsymbol{R}_1 = \boldsymbol{U}\boldsymbol{R}_Z^{\mathrm{T}}\left(\tfrac{\pi}{2}\right)\boldsymbol{V}^{\mathrm{T}} \\ \boldsymbol{t}_2^\wedge &= \boldsymbol{U}\boldsymbol{R}_Z\left(-\tfrac{\pi}{2}\right)\boldsymbol{\Sigma}\boldsymbol{U}^{\mathrm{T}}, \quad \boldsymbol{R}_2 = \boldsymbol{U}\boldsymbol{R}_Z^{\mathrm{T}}\left(-\tfrac{\pi}{2}\right)\boldsymbol{V}^{\mathrm{T}}. \end{aligned} \tag{7.14}$$

其中$\boldsymbol{R}_Z\left(\frac{\pi}{2}\right)$表示沿$Z$轴旋转90°得到的旋转矩阵。同时，由于-$E$和$E$等

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

价，所以对任意一个t取负号，也会得到同样的结果。因此，从E分解到t,R时，一共存在4个可能的解。

图7-8形象地展示了分解本质矩阵得到的4个解。我们已知空间点在相机（蓝色线）上的投影（红色点），想要求解相机的运动。在保持红色点不变的情况下，可以画出4种可能的情况。不过幸运的是，只有第一种解中P在两个相机中都具有正的深度。因此，只要把任意一点代入4种解中，检测该点在两个相机下的深度，就可以确定哪个解是正确的了。

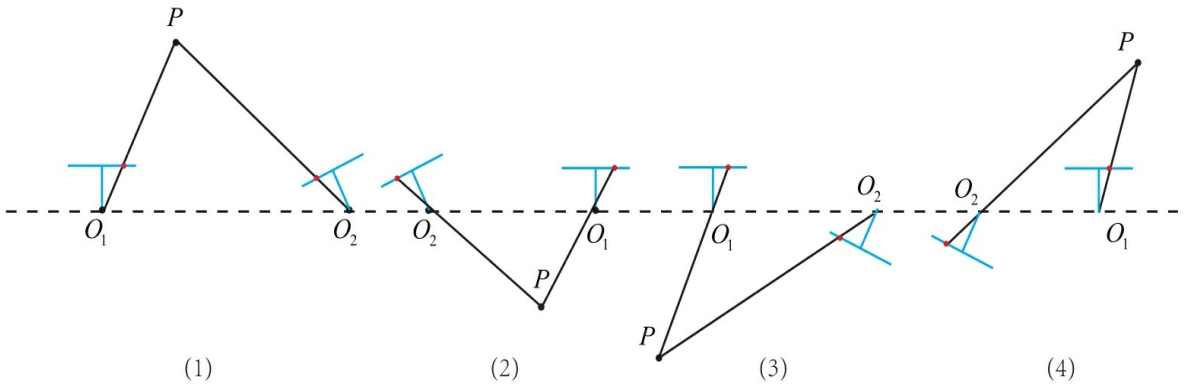


图7-8　分解本质矩阵得到的4个解。在保持投影点（红色点）不变的情况下，两个相机及空间点一共有4种可能的情况。

如果利用E的内在性质，那么它只有5个自由度。所以最小可以通过5对点来求解相机运动[40,41]。然而这种做法形式复杂，从工程实现角度考虑，由于平时通常会有几十对乃至上百对的匹配点，从8对减至5对意义并不明显。为保持简单，我们这里就只介绍基本的八点法。

剩下的问题还有一个：根据线性方程解出的E，可能不满足E的内在性质——它的奇异值不一定为$\sigma,\sigma,0$的形式。这时，在做SVD时我们会刻意地把Σ矩阵调整成上面的样子。通常的做法是，对八点法求得的E进行SVD分解后，会得到奇异值矩阵$\Sigma=\mathrm{diag}(\sigma_1,\sigma_2,\sigma_3)$，不妨设$\sigma_1\geq\sigma_2\geq\sigma_3$。取：

$$\boldsymbol{E}=\boldsymbol{U}\mathrm{diag}(\frac{\sigma_1+\sigma_2}{2},\frac{\sigma_1+\sigma_2}{2},0)\boldsymbol{V}^{\mathrm{T}}. \tag{7.15}$$

这相当于是把求出来的矩阵投影到了E所在的流形上。当然，更简

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

单的做法是将奇异值矩阵取成diag(1,1,0)，因为E具有尺度等价性，所以这样做也是合理的。

7.3.3 单应矩阵

除了基本矩阵和本质矩阵，还有单应矩阵（Homography）H，它描述了两个平面之间的映射关系。若场景中的特征点都落在同一平面上（比如墙、地面等），则可以通过单应性来进行运动估计。这种情况在无人机携带的俯视相机或扫地机携带的顶视相机中比较常见。由于之前没有提到过单应，因此这里稍微介绍一下。

单应矩阵通常描述处于共同平面上的一些点在两张图像之间的变换关系。考虑在图像I_1和I_2有一对匹配好的特征点p_1和p_2。这些特征点落在平面P上，设这个平面满足方程：

$$\boldsymbol{n}^{\mathrm{T}}\boldsymbol{P} + d = 0. \tag{7.16}$$

稍加整理，得：

$$-\frac{\boldsymbol{n}^{\mathrm{T}}\boldsymbol{P}}{d} = 1. \tag{7.17}$$

然后，回顾本节开头的式(7.1)，得：

$$\begin{aligned}
\boldsymbol{p}_2 &= \boldsymbol{K}(\boldsymbol{R}\boldsymbol{P} + \boldsymbol{t}) \\
&= \boldsymbol{K}\left(\boldsymbol{R}\boldsymbol{P} + \boldsymbol{t}\cdot(-\frac{\boldsymbol{n}^{\mathrm{T}}\boldsymbol{P}}{d})\right) \\
&= \boldsymbol{K}\left(\boldsymbol{R} - \frac{\boldsymbol{t}\boldsymbol{n}^{\mathrm{T}}}{d}\right)\boldsymbol{P} \\
&= \boldsymbol{K}\left(\boldsymbol{R} - \frac{\boldsymbol{t}\boldsymbol{n}^{\mathrm{T}}}{d}\right)\boldsymbol{K}^{-1}\boldsymbol{p}_1.
\end{aligned}$$

于是，我们得到了一个直接描述图像坐标p_1和p_2之间的变换，把中间这部分记为H，于是：

$$\boldsymbol{p}_2 = \boldsymbol{H}\boldsymbol{p}_1. \tag{7.18}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

它的定义与旋转、平移及平面的参数有关。与基础矩阵*F*类似，单应矩阵*H*也是一个3×3的矩阵，求解时的思路也和*F*类似，同样可以先根据匹配点计算*H*，然后将它分解以计算旋转和平移。把上式展开，得：

$$\begin{pmatrix} u_2 \\ v_2 \\ 1 \end{pmatrix} = \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{pmatrix} \begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix}. \tag{7.19}$$

请注意，这里的等号是在非零因子下成立的。我们在实际处理中通常乘以一个非零因子使得$h_9=1$（在它取非零值时）。然后根据第3行，去掉这个非零因子，于是有：

$$\begin{aligned} u_2 &= \frac{h_1 u_1 + h_2 v_1 + h_3}{h_7 u_1 + h_8 v_1 + h_9} \\ v_2 &= \frac{h_4 u_1 + h_5 v_1 + h_6}{h_7 u_1 + h_8 v_1 + h_9}. \end{aligned}$$

整理得：

$$\begin{aligned} h_1 u_1 + h_2 v_1 + h_3 - h_7 u_1 u_2 - h_8 v_1 u_2 = u_2 \\ h_4 u_1 + h_5 v_1 + h_6 - h_7 u_1 v_2 - h_8 v_1 v_2 = v_2. \end{aligned}$$

这样一组匹配点对就可以构造出两项约束（事实上有三个约束，但是因为线性相关，只取前两个），于是自由度为8的单应矩阵可以通过4对匹配特征点算出（注意，这些特征点不能有三点共线的情况），即求解以下的线性方程组（当$h_9=0$时，右侧为零）：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$\begin{pmatrix} u_1^1 & v_1^1 & 1 & 0 & 0 & 0 & -u_1^1u_2^1 & -v_1^1u_2^1 \\ 0 & 0 & 0 & u_1^1 & v_1^1 & 1 & -u_1^1v_2^1 & -v_1^1v_2^1 \\ u_1^2 & v_1^2 & 1 & 0 & 0 & 0 & -u_1^2u_2^2 & -v_1^2u_2^2 \\ 0 & 0 & 0 & u_1^2 & v_1^2 & 1 & -u_1^2v_2^2 & -v_1^2v_2^2 \\ u_1^3 & v_1^3 & 1 & 0 & 0 & 0 & -u_1^3u_2^3 & -v_1^3u_2^3 \\ 0 & 0 & 0 & u_1^3 & v_1^3 & 1 & -u_1^3v_2^3 & -v_1^3v_2^3 \\ u_1^4 & v_1^4 & 1 & 0 & 0 & 0 & -u_1^4u_2^4 & -v_1^4u_2^4 \\ 0 & 0 & 0 & u_1^4 & v_1^4 & 1 & -u_1^4v_2^4 & -v_1^4v_2^4 \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \end{pmatrix} = \begin{pmatrix} u_2^1 \\ v_2^1 \\ u_2^2 \\ v_2^2 \\ u_2^3 \\ v_2^3 \\ u_2^4 \\ v_2^4 \end{pmatrix}. \tag{7.20}$$

这种做法把H矩阵看成了向量，通过解该向量的线性方程来恢复H，又称直接线性变换法（Direct Linear Transform）。与本质矩阵相似，求出单应矩阵以后需要对其进行分解，才可以得到相应的旋转矩阵R和平移向量t。分解的方法包括数值法[42,43]与解析法[44]。与本质矩阵的分解类似，单应矩阵的分解同样会返回4组旋转矩阵与平移向量，并且同时可以计算出它们分别对应的场景点所在平面的法向量。如果已知成像的地图点的深度全为正值（即在相机前方），则又可以排除两组解。最后仅剩两组解，这时需要通过更多的先验信息进行判断。通常我们可以通过假设已知场景平面的法向量来解决，如场景平面与相机平面平行，那么法向量n的理论值为1^{T}。

单应性在SLAM中具有重要意义。当特征点共面或者相机发生纯旋转时，基础矩阵的自由度下降，这就出现了所谓的退化（degenerate）。现实中的数据总包含一些噪声，这时候如果继续使用八点法求解基础矩阵，基础矩阵多余出来的自由度将会主要由噪声决定。为了能够避免退化现象造成的影响，通常我们会同时估计基础矩阵F和单应矩阵H，选择重投影误差比较小的那个作为最终的运动估计矩阵。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

7.4 实践：对极约束求解相机运动

下面，我们来练习一下如何通过本质矩阵求解相机运动。上一节实践部分的程序提供了特征匹配，而这次我们就使用匹配好的特征点来计算E,F和H，进而分解E得到R,t。整个程序使用OpenCV提供的算法进行求解。我们把上一节的特征提取封装成函数，以供后面使用。本节只展示位姿估计部分的代码。

slambook/ch7/pose_estimation_2d2d.cpp（片段）

```
void pose_estimation_2d2d (
    std::vector<KeyPoint> keypoints_1,
    std::vector<KeyPoint> keypoints_2,
    std::vector< DMatch > matches,
    Mat& R, Mat& t )
{
    // 相机内参, TUM Freiburg2
    Mat K = ( Mat_<double> ( 3,3 ) << 520.9, 0, 325.1, 0, 521.0, 249.7, 0, 0, 1 );

    //-- 把匹配点转换为 vector<Point2f> 的形式
    vector<Point2f> points1;
    vector<Point2f> points2;

    for ( int i = 0; i < ( int ) matches.size(); i++ )
    {
        points1.push_back( keypoints_1[matches[i].queryIdx].pt );
        points2.push_back( keypoints_2[matches[i].trainIdx].pt );
    }

    //-- 计算基础矩阵
    Mat fundamental_matrix;
    fundamental_matrix = findFundamentalMat ( points1, points2, CV_FM_8POINT );
    cout<<"fundamental_matrix is "<<endl<< fundamental_matrix<<endl;
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
24
25      //-- 计算本质矩阵
26      Point2d principal_point ( 325.1, 249.7 );   // 光心, TUM dataset 标定值
27      int focal_length = 521;                     // 焦距, TUM dataset 标定值
28      Mat essential_matrix;
29      essential_matrix = findEssentialMat ( points1, points2, focal_length,
        principal_point, RANSAC );
30      cout<<"essential_matrix is "<<endl<< essential_matrix<<endl;
31
32      //-- 计算单应矩阵
33      Mat homography_matrix;
34      homography_matrix = findHomography ( points1, points2, RANSAC, 3, noArray(),
        2000, 0.99 );
35      cout<<"homography_matrix is "<<endl<<homography_matrix<<endl;
36
37      //-- 从本质矩阵中恢复旋转和平移信息
38      recoverPose ( essential_matrix, points1, points2, R, t, focal_length,
        principal_point );
39      cout<<"R is "<<endl<<R<<endl;
40      cout<<"t is "<<endl<<t<<endl;
41  }
```

该函数提供了从特征点求解相机运动的部分，然后，我们在主函数中调用它，就能得到相机的运动：

slambook/ch7/pose_estimation_2d2d.cpp（片段）

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
int main( int argc, char** argv )
{
    if ( argc != 3 )
    {
        cout<<"usage: feature_extraction img1 img2"<<endl;
        return 1;
    }
    //-- 读取图像
    Mat img_1 = imread ( argv[1], CV_LOAD_IMAGE_COLOR );
    Mat img_2 = imread ( argv[2], CV_LOAD_IMAGE_COLOR );

    vector<KeyPoint> keypoints_1, keypoints_2;
    vector<DMatch> matches;
    find_feature_matches( img_1, img_2, keypoints_1, keypoints_2, matches );
    cout<<"一共找到了"<<matches.size()<<"组匹配点"<<endl;

    //-- 估计两张图像间的运动
    Mat R,t;
    pose_estimation_2d2d( keypoints_1, keypoints_2, matches, R, t );
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
20
21      //-- 验证 E=t^R*scale
22      Mat t_x = (Mat_<double>(3,3) <<
23          0,                      -t.at<double>(2,0),     t.at<double>(1,0),
24          t.at<double>(2,0),      0,                      -t.at<double>(0,0),
25          -t.at<double>(1.0),     t.at<double>(0,0),      0);
26
27      cout<<"t^R="<<endl<<t_x*R<<endl;
28      //-- 验证对极约束
29      Mat K = ( Mat_<double> ( 3,3 ) << 520.9, 0, 325.1, 0, 521.0, 249.7, 0, 0, 1 );
30      for ( DMatch m: matches )
31      {
32          Point2d pt1 = pixel2cam( keypoints_1[ m.queryIdx ].pt, K );
33          Mat y1 = (Mat_<double>(3,1) << pt1.x, pt1.y, 1);
34          Point2d pt2 = pixel2cam( keypoints_2[ m.trainIdx ].pt, K );
35          Mat y2 = (Mat_<double>(3,1) << pt2.x, pt2.y, 1);
36          Mat d = y2.t() * t_x * R * y1;
37          cout << "epipolar constraint = " << d << endl;
38      }
39      return 0;
40  }
```

我们在函数中输出了E,F和H的数值，然后验证了对极约束是否成立，以及$t^\wedge R$和E在非零数乘下等价的事实。现在，调用此程序即可看到输出结果：

```
% build/pose_estimation_2d2d 1.png 2.png
-- Max dist : 95.000000
-- Min dist : 4.000000
一共找到了 79 组匹配点
fundamental_matrix is
[4.844484382466111e-06, 0.0001222601840188731, -0.01786737827487386;
-0.0001174326832719333, 2.122888800459598e-05, -0.01775877156212593;
0.01799658210895528, 0.008143605989020664, 1]
essential_matrix is
[-0.0203618550523477, -0.4007110038118445, -0.03324074249824097;
0.3939270778216369, -0.03506401846698079, 0.5857110303721015;
-0.006788487241438284, -0.5815434272915686, -0.01438258684486258]
homography_matrix is
[0.9497129583105288, -0.143556453147626, 31.20121878625771;
0.04154536627445031, 0.9715568969832015, 5.306887618807696;
-2.81813676978796e-05, 4.353702039810921e-05, 1]
R is
[0.9985961798781875, -0.05169917220143662, 0.01152671359827873;
0.05139607508976055, 0.9983603445075083, 0.02520051547522442;
-0.01281065954813571, -0.02457271064688495, 0.9996159607036126]
```

```
t is
[-0.8220841067933337;
-0.03269742706405412;
0.5684264241053522]

t^R=
[0.02879601157010516, 0.5666909361828478, 0.04700950886436416;
-0.5570970160413605, 0.0495880104673049, -0.8283204827837456;
0.009600370724838804, 0.8224266019846683, 0.02034004937801349]
epipolar constraint = [0.002528128704106625]
epipolar constraint = [-0.001663727901710724]
epipolar constraint = [-0.0008009088410884102]
......
```

在程序的输出结果可以看出，对极约束的满足精度约在10^{-3}量级。根据前面的讨论，分解得到的R,t一共有4种可能性。不过，OpenCV会替我们使用三角化检测角点的深度是否为正，从而选出正确的解。

书籍汇总下载 http://t.cn/RDIAj5D

需要注意的地方是，我们要弄清程序求解出来的R,t是什么意义。按照例程的定义，我们的对极约束是从

$$\boldsymbol{x}_2 = \boldsymbol{R}\boldsymbol{x}_1 + \boldsymbol{t}$$

得到的。这里的R,t组成的变换矩阵，是第一个图到第二个图的坐标变换矩阵：

$$\boldsymbol{x}_2 = \boldsymbol{T}_{21}\boldsymbol{x}_1. \tag{7.21}$$

请读者在实践中务必清楚这里使用的变换顺序（因为有时我们会用T_{12}），它们非常容易搞反。

讨论

从演示程序中可以看到，输出的E和F之间相差了相机内参矩阵。虽然它们在数值上并不直观，但可以验证它们的数学关系。从E,F和H都可以分解出运动，不过H需要假设特征点位于平面上。对于本实验的数据，这个假设是不好的，所以我们这里主要用E来分解运动。

值得一提的是，由于E本身具有尺度等价性，它分解得到的t,R也有一个尺度等价性。而$R \in \mathrm{SO}(3)$自身具有约束，所以我们认为t具有一个尺度。换言之，在分解过程中，对t乘以任意非零常数，分解都是成立的。因此，我们通常把t进行归一化，让它的长度等于1。

尺度不确定性

对t长度的归一化，直接导致了单目视觉的尺度不确定性（Scale Ambiguity）。例如，程序中输出的t第一维约为0.822。这个0.822究竟是指0.822米还是0.822厘米，我们是没法确定的。因为对t乘以任意比例常数后，对极约束依然是成立的。换言之，在单目SLAM中，对轨迹和地图同时缩放任意倍数，我们得到的图像依然是一样的。这在第2讲中就已经向读者介绍过了。

在单目视觉中，我们对两张图像的t归一化相当于固定了尺度。虽然我们不知道它的实际长度是多少，但我们以这时的t为单位1，计算相机运动和特征点的3D位置。这被称为单目SLAM的初始化。在初始化之后，就可以用3D-2D来计算相机运动了。初始化之后的轨迹和地图的单位，就是初始化时固定的尺度。因此，单目SLAM有一步不可避免的初

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

始化。初始化的两张图像必须有一定程度的平移，而后的轨迹和地图都将以此步的平移为单位。

除了对t进行归一化之外，另一种方法是令初始化时所有的特征点平均深度为1，也可以固定一个尺度。相比于令t长度为1的做法，把特征点深度归一化可以控制场景的规模大小，使计算在数值上更稳定些。不过这并没有理论上的差别。

初始化的纯旋转问题

从E分解到R,t的过程中，如果相机发生的是纯旋转，导致t为零，那么，得到的E也将为零，这将导致我们无从求解R。不过，此时我们可以依靠H求取旋转，但仅有旋转时，我们无法用三角测量估计特征点的空间位置（这将在下文提到），于是，另一个结论是，单目初始化不能只有纯旋转，必须要有一定程度的平移。如果没有平移，单目将无法初始化。在实践当中，如果初始化时平移太小，会使得位姿求解与三角化结果不稳定，从而导致失败。相对地，如果把相机左右移动而不是原地旋转，就容易让单目SLAM初始化。因而，有经验的SLAM研究人员，在单目SLAM情况下经常选择让相机进行左右平移以顺利地进行初始化。

多于8对点的情况

当给定的点数多于8对时（比如，例程找到了79对匹配），我们可以计算一个最小二乘解。回忆式(7.12)中线性化后的对极约束，我们把左侧的系数矩阵记为A：

$$\boldsymbol{A}\boldsymbol{e} = \boldsymbol{0}. \tag{7.22}$$

对于八点法，A的大小为8×9。如果给定的匹配点多于8，该方程构成一个超定方程，即不一定存在e使得上式成立。因此，可以通过最小化一个二次型来求：

$$\min_{\boldsymbol{e}} \|\boldsymbol{A}\boldsymbol{e}\|_2^2 = \min_{\boldsymbol{e}} \boldsymbol{e}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{e}. \tag{7.23}$$

于是就求出了在最小二乘意义下的E矩阵。不过，当可能存在误匹配的情况时，我们会更倾向于使用随机采样一致性（Random Sample Concensus，RANSAC）来求，而不是最小二乘。RANSAC是一种通用

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

的做法，适用于很多带错误数据的情况，可以处理带有错误匹配的数据。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

7.5 三角测量

之前两节我们使用对极几何约束估计了相机运动，也讨论了这种方法的局限性。在得到运动之后，下一步我们需要用相机的运动估计特征点的空间位置。在单目SLAM中，仅通过单张图像无法获得像素的深度信息，我们需要通过三角测量（Triangulation）（或三角化）的方法来估计地图点的深度，如图7-9所示。

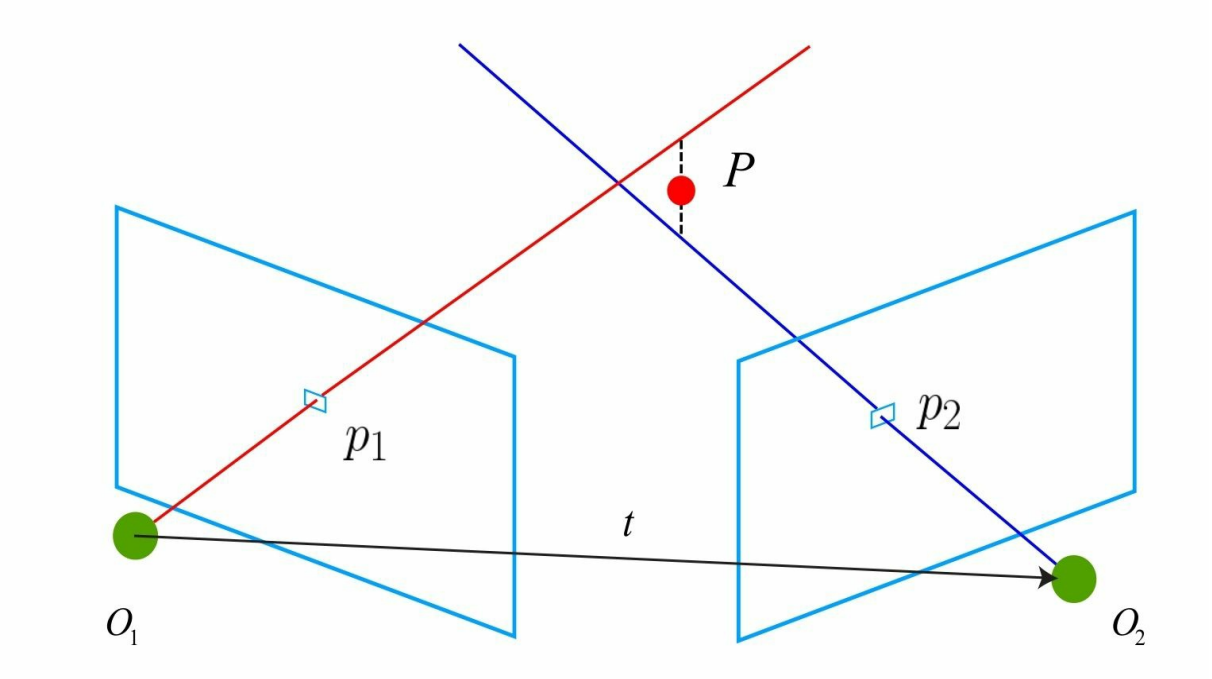


图7-9　三角化获得地图点深度

三角测量是指，通过在两处观察同一个点的夹角，从而确定该点的距离。三角测量最早由高斯提出并应用于测量学中，它在天文学、地理学的测量中都有应用。例如，我们可以通过不同季节观察到的星星的角度，估计它离我们的距离。在SLAM中，我们主要用三角化来估计像素点的距离。

和上一节类似，考虑图像I_1和I_2，以左图为参考，右图的变换矩阵为T。相机光心为O_1和O_2。在I_1中有特征点p_1，对应I_2中有特征点p_2。理

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

论上直线O_1p_1与O_2p_2在场景中会相交于一点P，该点即两个特征点所对应的地图点在三维场景中的位置。然而由于噪声的影响，这两条直线往往无法相交。因此，可以通过最二小乘法求解。

按照对极几何中的定义，设x_1,x_2为两个特征点的归一化坐标，那么它们满足：

$$s_1\boldsymbol{x}_1 = s_2\boldsymbol{R}\boldsymbol{x}_2 + \boldsymbol{t}. \tag{7.24}$$

现在我们已经知道了R,t，想要求解的是两个特征点的深度s_1,s_2。当然这两个深度是可以分开求的，比如，先来看s_2。如果要算s_2，那么先对上式两侧左乘一个$x^\wedge_1$，得：

$$s_1\boldsymbol{x}_1^\wedge\boldsymbol{x}_1 = 0 = s_2\boldsymbol{x}_1^\wedge\boldsymbol{R}\boldsymbol{x}_2 + \boldsymbol{x}_1^\wedge\boldsymbol{t}. \tag{7.25}$$

该式左侧为零，右侧可看成s_2的一个方程，可以根据它直接求得s_2。有了s_2，s_1也非常容易求出。于是，我们就得到了两帧下的点的深度，确定了它们的空间坐标。当然，由于噪声的存在，我们估得的R,t不一定精确使式(7.24)为零，所以更常见的做法是求最小二乘解而不是零解。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

7.6 实践：三角测量

7.6.1 三角测量代码

下面，我们演示如何根据之前利用对极几何求解的相机位姿，通过三角化求出上一节特征点的空间位置。我们调用OpenCV提供的triangulation函数进行三角化。

slambook/ch7/triangulation.cpp（片断）

```
void triangulation (
    const vector<KeyPoint>& keypoint_1,
    const vector<KeyPoint>& keypoint_2,
    const std::vector< DMatch >& matches,
    const Mat& R, const Mat& t,
    vector<Point3d>& points
);

void triangulation (
    const vector< KeyPoint >& keypoint_1,
    const vector< KeyPoint >& keypoint_2,
    const std::vector< DMatch >& matches,
    const Mat& R, const Mat& t,
    vector< Point3d >& points )
{
    Mat T1 = (Mat_<double> (3,4) <<
        1,0,0,0,
        0,1,0,0,
        0,0,1,0);
    Mat T2 = (Mat_<double> (3,4) <<
        R.at<double>(0,0), R.at<double>(0,1), R.at<double>(0,2), t.at<double>(0,0),
        R.at<double>(1,0), R.at<double>(1,1), R.at<double>(1,2), t.at<double>(1,0),
        R.at<double>(2,0), R.at<double>(2,1), R.at<double>(2,2), t.at<double>(2,0)
    );
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
25
26      Mat K = ( Mat_<double> ( 3,3 ) << 520.9, 0, 325.1, 0, 521.0, 249.7, 0, 0, 1 );
27      vector<Point2d> pts_1, pts_2;
28      for ( DMatch m:matches )
29      {
30          // 将像素坐标转换至相机坐标
31          pts_1.push_back ( pixel2cam( keypoint_1[m.queryIdx].pt, K) );
32          pts_2.push_back ( pixel2cam( keypoint_2[m.trainIdx].pt, K) );
33      }
34
35      Mat pts_4d;
36      cv::triangulatePoints( T1, T2, pts_1, pts_2, pts_4d );
37
38      // 转换成非齐次坐标
39      for ( int i=0; i<pts_4d.cols; i++ )
40      {
41          Mat x = pts_4d.col(i);
42          x /= x.at<float>(3,0); // 归一化
43          Point3d p (
44              x.at<float>(0,0),
45              x.at<float>(1,0),
46              x.at<float>(2,0)
47          );
48          points.push_back( p );
49      }
50  }
```

同时，在main函数中增加三角测量部分，并验证重投影关系：

书籍汇总下载 http://t.cn/RDIAj5D

```
int main (int argc, char** argv)
{
    // .....
    //-- 三角化
    vector<Point3d> points;
    triangulation( keypoints_1, keypoints_2, matches, R, t, points );

    //-- 验证三角化点与特征点的重投影关系
    Mat K = ( Mat_<double> ( 3,3 ) << 520.9, 0, 325.1, 0, 521.0, 249.7, 0, 0, 1 );
    for ( int i=0; i<matches.size(); i++ )
    {
        Point2d pt1_cam = pixel2cam( keypoints_1[ matches[i].queryIdx ].pt, K );
        Point2d pt1_cam_3d (
            points[i].x/points[i].z,
            points[i].y/points[i].z
        );

        cout<<"point in the first camera frame: "<<pt1_cam<<endl;
        cout<<"point projected from 3D "<<pt1_cam_3d<<", d="<<points[i].z<<endl;

        // 第2幅图
        Point2f pt2_cam = pixel2cam( keypoints_2[ matches[i].trainIdx ].pt, K );
        Mat pt2_trans = R*( Mat_<double>(3,1) << points[i].x, points[i].y, points[i
        ].z ) + t;
        pt2_trans /= pt2_trans.at<double>(2,0);
        cout<<"point in the second camera frame: "<<pt2_cam<<endl;
        cout<<"point reprojected from second frame: "<<pt2_trans.t()<<endl;
        cout<<endl;
    }
    // ...
}
```

我们打印了每个空间点在两个相机坐标系下的投影坐标与像素坐标——相当于P的投影位置与看到的特征点位置。由于误差的存在，它们会有一些微小的差异。以下是某一特征点的信息：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
point in the first camera frame: [0.0844072, -0.0734976]
point projected from 3D [0.0843702, -0.0743606], d=14.9895
point in the second camera frame: [0.0431343, -0.0459876]
point reprojected from second frame: [0.04312769812378599, -0.04515455276163744, 1]
```

可以看到，误差的量级大约在小数点后第3位。可以看到，三角化特征点的距离大约为15。但由于尺度不确定性，我们并不知道这里的15究竟是多少米。

7.6.2 讨论

关于三角测量，还有一个必须注意的地方。

三角测量是由平移得到的，有平移才会有对极几何中的三角形，才谈得上三角测量。因此，纯旋转是无法使用三角测量的，因为对极约束将永远满足。在平移存在的情况下，我们还要关心三角测量的不确定性，这会引出一个三角测量的矛盾。

如图7-10所示，当平移很小时，像素上的不确定性将导致较大的深度不确定性。也就是说，如果特征点运动一个像素δx，使得视线角变化了一个角度$\delta\theta$，那么将测量到深度值有δd的变化。从几何关系可以看到，当t较大时，δd将明显变小，这说明平移较大时，在同样的相机分辨率下，三角化测量将更精确。对该过程的定量分析可以使用正弦定理得到，不过这里先考虑定性分析。

因此，要提高三角化的精度，其一是提高特征点的提取精度，也就是提高图像分辨率——但这会导致图像变大，增加计算成本。另一方式是使平移量增大。但是，这会导致图像的外观发生明显的变化，比如箱子原先被挡住的侧面显示出来，又比如反射光发生变化，等等。外观变化会使得特征提取与匹配变得困难。总而言之，再增大平移，会导致匹配失效；而平移太小，则三角化精度不够——这就是三角化的矛盾。

书籍汇总下载 http://t.cn/RDIAj5D

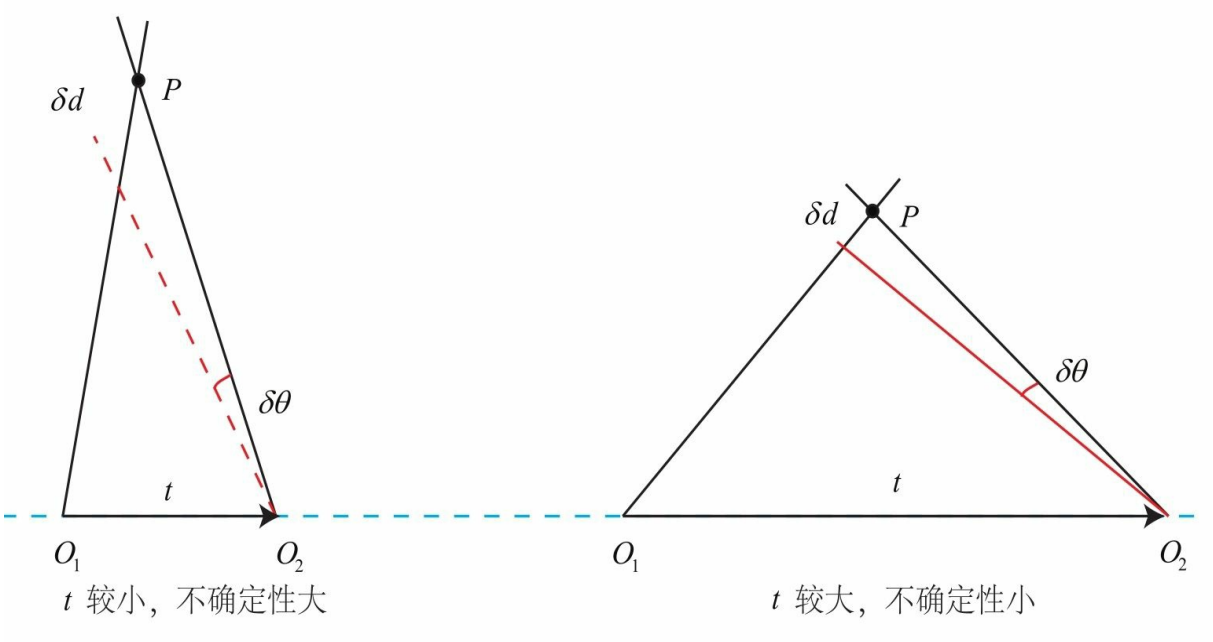


图7-10　三角测量的矛盾。

虽然本节只介绍了三角化的深度估计，但只要我们愿意，也能够定量地计算每个特征点的位置及不确定性。所以，如果假设特征点服从高斯分布，并且不断地对它进行观测，在信息正确的情况下，我们就能够期望它的方差会不断减小乃至收敛。这就得到了一个滤波器，称为深度滤波器（Depth Filter）。不过，由于它的原理较复杂，我们将留到第13讲再详细讨论它。下面，我们来讨论从3D-2D的匹配点来估计相机运动，以及3D-3D的估计方法。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

7.7 3D-2D：PnP

PnP（Perspective-n-Point）是求解3D到2D点对运动的方法。它描述了当知道n个3D空间点及其投影位置时，如何估计相机的位姿。前面说到，2D-2D的对极几何方法需要8个或8个以上的点对（以八点法为例），且存在着初始化、纯旋转和尺度的问题。然而，如果两张图像中其中一张特征点的3D位置已知，那么最少只需3个点对（需要至少一个额外点验证结果）就可以估计相机运动。特征点的3D位置可以由三角化或者RGB-D相机的深度图确定。因此，在双目或RGB-D的视觉里程计中，我们可以直接使用PnP估计相机运动。而在单目视觉里程计中，必须先进行初始化，然后才能使用PnP。3D-2D方法不需要使用对极约束，又可以在很少的匹配点中获得较好的运动估计，是最重要的一种姿态估计方法。

PnP问题有很多种求解方法，例如，用3对点估计位姿的P3P[45]、直接线性变换（DLT）、EPnP（E ffi cientPnP）[46]、UPnP[47]，等等。此外，还能用非线性优化的方式，构建最小二乘问题并迭代求解，也就是万金油式的Bundle Adjustment。我们先来看DLT，然后再讲解Bundle Adjustment。

7.7.1 直接线性变换

考虑某个空间点P，它的齐次坐标为$P=(X,Y,Z,1)^{\mathrm{T}}$。在图像I_1中，投影到特征点$x_1=(u_1,v_1,1)^{\mathrm{T}}$（以归一化平面齐次坐标表示）。此时相机的位姿$R,t$是未知的。与单应矩阵的求解类似，我们定义增广矩阵$[R|t]$为一个3×4的矩阵，包含了旋转与平移信息[4]。我们将其展开形式列写如下：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$s\begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix} = \begin{pmatrix} t_1 & t_2 & t_3 & t_4 \\ t_5 & t_6 & t_7 & t_8 \\ t_9 & t_{10} & t_{11} & t_{12} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}. \tag{7.26}$$

用最后一行把s消去，得到两个约束：为了简化表示，定义$\boldsymbol{T}$的行向量：

$$u_1 = \frac{t_1 X + t_2 Y + t_3 Z + t_4}{t_9 X + t_{10} Y + t_{11} Z + t_{12}}, \quad v_1 = \frac{t_5 X + t_6 Y + t_7 Z + t_8}{t_9 X + t_{10} Y + t_{11} Z + t_{12}}.$$

$$\boldsymbol{t}_1 = (t_1, t_2, t_3, t_4)^{\mathrm{T}}, \boldsymbol{t}_2 = (t_5, t_6, t_7, t_8)^{\mathrm{T}}, \boldsymbol{t}_3 = (t_9, t_{10}, t_{11}, t_{12})^{\mathrm{T}},$$

于是有：

$$\boldsymbol{t}_1^{\mathrm{T}} \boldsymbol{P} - \boldsymbol{t}_3^{\mathrm{T}} \boldsymbol{P} u_1 = 0,$$

和

$$\boldsymbol{t}_2^{\mathrm{T}} \boldsymbol{P} - \boldsymbol{t}_3^{\mathrm{T}} \boldsymbol{P} v_1 = 0.$$

请注意，t是待求的变量，可以看到，每个特征点提供了两个关于t的线性约束。假设一共有N个特征点，则可以列出如下线性方程组：

$$\begin{pmatrix} \boldsymbol{P}_1^{\mathrm{T}} & 0 & -u_1 \boldsymbol{P}_1^{\mathrm{T}} \\ 0 & \boldsymbol{P}_1^{\mathrm{T}} & -v_1 \boldsymbol{P}_1^{\mathrm{T}} \\ \vdots & \vdots & \vdots \\ \boldsymbol{P}_N^{\mathrm{T}} & 0 & -u_N \boldsymbol{P}_N^{\mathrm{T}} \\ 0 & \boldsymbol{P}_N^{\mathrm{T}} & -v_N \boldsymbol{P}_N^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} \boldsymbol{t}_1 \\ \boldsymbol{t}_2 \\ \boldsymbol{t}_3 \end{pmatrix} = 0. \tag{7.27}$$

由于t一共有12维，因此，最少通过6对匹配点即可实现矩阵$\boldsymbol{T}$的线性求解，这种方法称为直接线性变换（Direct Linear Transform，DLT）。当匹配点大于6对时，也可以使用SVD等方法对超定方程求最

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

小二乘解。

在DLT求解中，我们直接将T矩阵看成了12个未知数，忽略了它们之间的联系。因为旋转矩阵$R \in SO(3)$，用DLT求出的解不一定满足该约束，它是一个一般矩阵。平移向量比较好办，它属于向量空间。对于旋转矩阵R，我们必须针对DLT估计的T左边3×3的矩阵块，寻找一个最好的旋转矩阵对它进行近似。这可以由QR分解完成[3,48]，相当于把结果从矩阵空间重新投影到SE(3)流形上，转换成旋转和平移两部分。

需要解释的是，我们这里的x_1使用了归一化平面坐标，去掉了内参矩阵K的影响——这是因为内参K在SLAM中通常假设为已知。即使内参未知，也能用PnP去估计K,R,t三个量。然而由于未知量增多，效果会差一些。

7.7.2 P3P

下面讲的P3P是另一种解PnP的方法。它仅使用3对匹配点，对数据要求较少，因此这里也简单介绍一下（这部分推导借鉴了文献[49]）。

P3P需要利用给定的3个点的几何关系。它的输入数据为3对3D-2D匹配点。记3D点为A,B,C，2D点为a,b,c，其中小写字母代表的点为对应大写字母代表的点在相机成像平面上的投影，如图7-11所示。此外，P3P还需要使用一对验证点，以从可能的解中选出正确的那一个（类似于对极几何情形）。记验证点对为D-d，相机光心为O。请注意，我们知道的是A,B,C在世界坐标系中的坐标，而不是在相机坐标系中的坐标。一旦3D点在相机坐标系下的坐标能够算出，我们就得到了3D-3D的对应点，把PnP问题转换为了ICP问题。

首先，显然三角形之间存在对应关系：

$$\Delta Oab - \Delta OAB, \quad \Delta Obc - \Delta OBC, \quad \Delta Oac - \Delta OAC. \tag{7.28}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

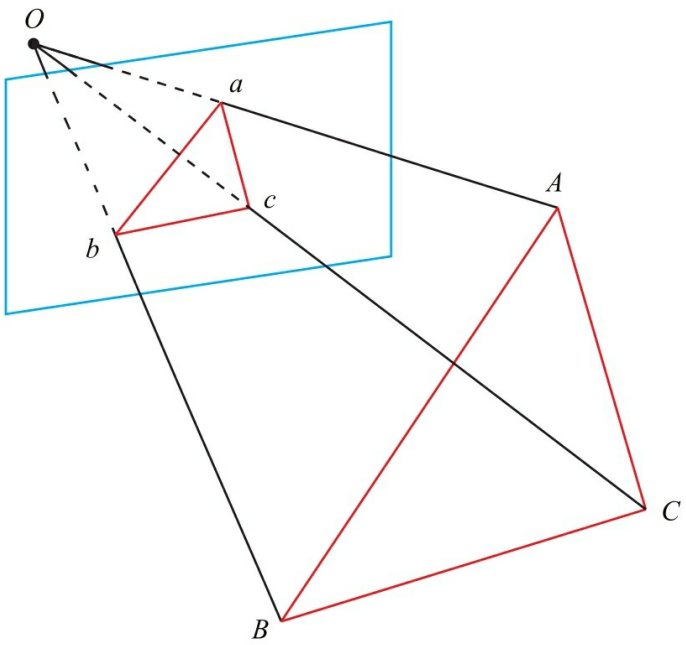


图7-11　P3P问题示意图。

来考虑*Oab*和*OAB*的关系。利用余弦定理，有：

$$OA^2 + OB^2 - 2OA \cdot OB \cdot \cos\langle a, b\rangle = AB^2. \tag{7.29}$$

对于其他两个三角形亦有类似性质，于是有：

$$\begin{aligned} &OA^2 + OB^2 - 2OA \cdot OB \cdot \cos\langle a, b\rangle = AB^2 \\ &OB^2 + OC^2 - 2OB \cdot OC \cdot \cos\langle b, c\rangle = BC^2 \\ &OA^2 + OC^2 - 2OA \cdot OC \cdot \cos\langle a, c\rangle = AC^2. \end{aligned} \tag{7.30}$$

对以上三式全体除以OC^2，并且记$x=OA/OC$,$y=OB/OC$，得：

$$\begin{aligned} &x^2 + y^2 - 2xy \cos\langle a, b\rangle = AB^2/OC^2 \\ &y^2 + 1^2 - 2y \cos\langle b, c\rangle = BC^2/OC^2 \\ &x^2 + 1^2 - 2x \cos\langle a, c\rangle = AC^2/OC^2. \end{aligned} \tag{7.31}$$

记$v=AB^2/OC^2$,$uv=BC^2/OC^2$,$wv=AC^2/OC^2$，有：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$
\begin{aligned}
&x^2 + y^2 - 2xy\cos\langle a,b\rangle - v = 0 \\
&y^2 + 1^2 - 2y\cos\langle b,c\rangle - uv = 0 \\
&x^2 + 1^2 - 2x\cos\langle a,c\rangle - wv = 0.
\end{aligned} \tag{7.32}
$$

我们可以把第一个式子中的v放到等式一边，并代入其后两式，得：

$$
\begin{aligned}
&(1-u)\,y^2 - ux^2 - \cos\langle b,c\rangle\, y + 2uxy\cos\langle a,b\rangle + 1 = 0 \\
&(1-w)\,x^2 - wy^2 - \cos\langle a,c\rangle\, x + 2wxy\cos\langle a,b\rangle + 1 = 0.
\end{aligned} \tag{7.33}
$$

注意这些方程中的已知量和未知量。由于我们知道2D点的图像位置，3个余弦角cos〈*a,b*〉,cos〈*b,c*〉,cos〈*a,c*〉是已知的。同时，$u=BC^2/AB^2$,$w=AC^2/AB^2$可以通过*A,B,C*在世界坐标系下的坐标算出，变换到相机坐标系下之后，这个比值并不改变。该式中的*x,y*是未知的，随着相机移动会发生变化。因此，该方程组是关于*x,y*的一个二元二次方程（多项式方程）。解析地求解该方程组是一个复杂的过程，需要用吴消元法。这里不展开对该方程解法的介绍，感兴趣的读者请参阅文献[45]。类似于分解*E*的情况，该方程最多可能得到4个解，但我们可以用验证点来计算最可能的解，得到*A,B,C*在相机坐标系下的3D坐标。然后，根据3D-3D的点对，计算相机的运动*R,t*。这部分将在7.9节介绍。

从P3P的原理可以看出，为了求解PnP，我们利用了三角形相似性质，求解投影点*a,b,c*在相机坐标系下的3D坐标，最后把问题转换成一个3D到3D的位姿估计问题。在后文将看到，带有匹配信息的3D-3D位姿求解非常容易，所以这种思路是非常有效的。其他的一些方法，例如EPnP，亦采用了这种思路。然而，P3P也存在着一些问题：

1.P3P只利用3个点的信息。当给定的配对点多于3组时，难以利用更多的信息。

2.如果3D点或2D点受噪声影响，或者存在误匹配，则算法失效。

所以后续人们还提出了许多别的方法，如EPnP、UPnP等。它们利用更多的信息，而且用迭代的方式对相机位姿进行优化，以尽可能地消除噪声的影响。不过，相对于P3P来说，原理会更加复杂一些，所以我们建议读者阅读原始的论文，或通过实践来理解PnP过程。在SLAM当

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

中，通常的做法是先使用P3P/EPnP等方法估计相机位姿，然后构建最小二乘优化问题对估计值进行调整（Bundle Adjustment）。接下来我们从非线性优化角度来看一下PnP问题。

7.7.3 Bundle Adjustment

除了使用线性方法之外，我们还可以把PnP问题构建成一个定义于李代数上的非线性最小二乘问题。这将用到本书第4讲和第5讲的知识。前面说的线性方法，往往是先求相机位姿，再求空间点位置，而非线性优化则是把它们都看成优化变量，放在一起优化。这是一种非常通用的求解方式，我们可以用它对PnP或ICP给出的结果进行优化。在PnP中，这个Bundle Adjustment问题，是一个最小化重投影误差（Reprojection error）的问题。我们将在本节给出此问题在两个视图下的基本形式，然后在第10讲讨论较大规模的BA问题。

考虑n个三维空间点P及其投影p，我们希望计算相机的位姿R,t，它的李代数表示为ξ。假设某空间点坐标为$P_i=[X_i,Y_i,Z_i]^{\mathrm{T}}$，其投影的像素坐标为$u_i=[u_i,v_i]^{\mathrm{T}}$。根据第5讲的内容，像素位置与空间点位置的关系如下：

$$s_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \boldsymbol{K} \exp\left(\boldsymbol{\xi}^{\wedge}\right) \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}. \tag{7.34}$$

除了用ξ表示相机姿态之外，别的都和前面的定义保持一致。写成矩阵形式就是：

$$s_i \boldsymbol{u}_i = \boldsymbol{K} \exp\left(\boldsymbol{\xi}^{\wedge}\right) \boldsymbol{P}_i.$$

请读者脑补中间隐含着的齐次坐标到非齐次的转换，否则按矩阵的乘法来说，维度是不对的[5]。现在，由于相机位姿未知及观测点的噪

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

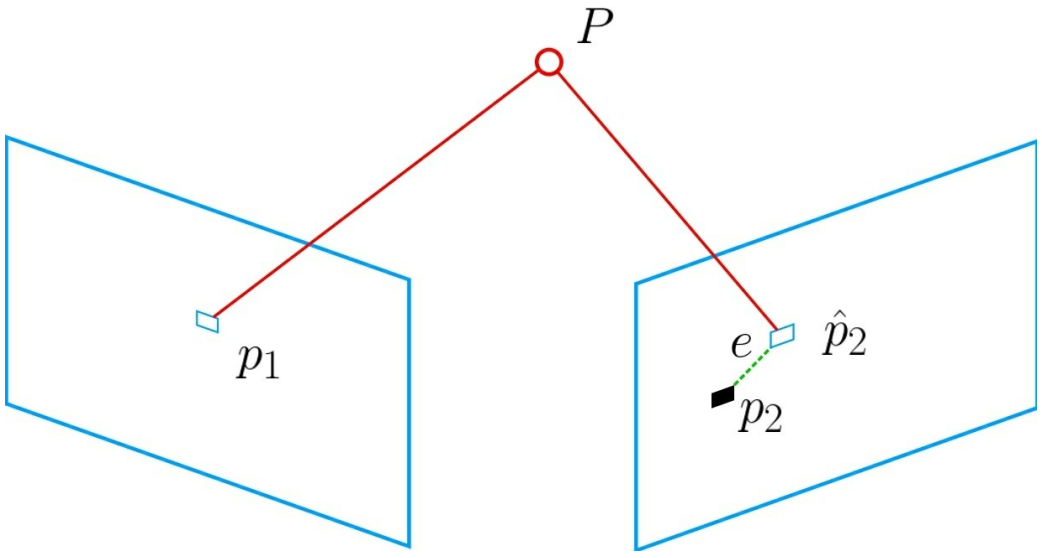
声，该等式存在一个误差。因此，我们把误差求和，构建最小二乘问题，然后寻找最好的相机位姿，使它最小化：

$$\boldsymbol{\xi}^* = \arg\min_{\boldsymbol{\xi}} \frac{1}{2} \sum_{i=1}^{n} \left\| \boldsymbol{u}_i - \frac{1}{s_i} \boldsymbol{K} \exp\left(\boldsymbol{\xi}^\wedge\right) \boldsymbol{P}_i \right\|_2^2. \tag{7.35}$$

该问题的误差项，是将像素坐标（观测到的投影位置）与3D点按照当前估计的位姿进行投影得到的位置相比较得到的误差，所以称为重投影误差。使用齐次坐标时，这个误差有3维。不过，由于u最后一维为1，该维度的误差一直为零，因而我们更多时候使用非齐次坐标，于是误差就只有2维了。如图7-12所示，我们通过特征匹配知道了p_1和p_2是同一个空间点P的投影，但是不知道相机的位姿。在初始值中，P的投影$\hat{p}_2$与实际的p_2之间有一定的距离。于是我们调整相机的位姿，使得这个距离变小。不过，由于这个调整需要考虑很多个点，所以最后每个点的误差通常都不会精确为零。

最小二乘优化问题已经在第6讲介绍过了。使用李代数，可以构建无约束的优化问题，很方便地通过高斯牛顿法、列文伯格—马夸尔特方法等优化算法进行求解。不过，在使用高斯牛顿法和列文伯格—马夸尔特方法之前，我们需要知道每个误差项关于优化变量的导数，也就是线性化：

$$\boldsymbol{e}(\boldsymbol{x} + \Delta\boldsymbol{x}) \approx \boldsymbol{e}(\boldsymbol{x}) + \boldsymbol{J}\Delta\boldsymbol{x}. \tag{7.36}$$



电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

图7-12　重投影误差示意图。

这里的J的形式是值得讨论的，甚至可以说是关键所在。我们固然可以使用数值导数，但如果能够推导出解析形式，则我们会优先考虑解析导数。现在，当e为像素坐标误差（2维），x为相机位姿（6维）时，J将是一个2×6的矩阵。我们来推导J的形式。

回忆李代数的内容，我们介绍了如何使用扰动模型来求李代数的导数。首先，记变换到相机坐标系下的空间点坐标为P'，并且将其前3维取出来：

$$\boldsymbol{P}' = (\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{P})_{1:3} = [X', Y', Z']^{\mathrm{T}}. \tag{7.37}$$

那么，相机投影模型相对于P'为

$$s\boldsymbol{u} = \boldsymbol{K}\boldsymbol{P}'. \tag{7.38}$$

展开：

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix}. \tag{7.39}$$

利用第3行消去s（实际上就是P'的距离），得：

$$u = f_x \frac{X'}{Z'} + c_x, \quad v = f_y \frac{Y'}{Z'} + c_y. \tag{7.40}$$

这与之前讲的相机模型是一致的。当我们求误差时，可以把这里的u,v与实际的测量值比较，求差。在定义了中间变量后，我们对$\xi^\wedge$左乘扰动量$\delta\xi$，然后考虑e的变化关于扰动量的导数。利用链式法则，可以列写如下：

$$\frac{\partial \boldsymbol{e}}{\partial \delta\boldsymbol{\xi}} = \lim_{\delta\boldsymbol{\xi}\to 0} \frac{e(\delta\boldsymbol{\xi} \oplus \boldsymbol{\xi})}{\delta\boldsymbol{\xi}} = \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{P}'} \frac{\partial \boldsymbol{P}'}{\partial \delta\boldsymbol{\xi}}. \tag{7.41}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

这里的$\oplus$指李代数上的左乘扰动。第一项是误差关于投影点的导数，在式(7.40)已经列出了变量之间的关系，易得：

$$\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{P}'} = -\begin{bmatrix} \frac{\partial u}{\partial X'} & \frac{\partial u}{\partial Y'} & \frac{\partial u}{\partial Z'} \\ \frac{\partial v}{\partial X'} & \frac{\partial v}{\partial Y'} & \frac{\partial v}{\partial Z'} \end{bmatrix} = -\begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} \end{bmatrix}. \tag{7.42}$$

而第二项为变换后的点关于李代数的导数，根据4.3.5节中的推导，得：

$$\frac{\partial (\boldsymbol{TP})}{\partial \delta \boldsymbol{\xi}} = (\boldsymbol{TP})^{\odot} = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{P}'^{\wedge} \\ \boldsymbol{0}^{\mathrm{T}} & \boldsymbol{0}^{\mathrm{T}} \end{bmatrix}. \tag{7.43}$$

而在$\boldsymbol{P}'$的定义中，我们取出了前3维，于是得：

$$\frac{\partial \boldsymbol{P}'}{\partial \delta \boldsymbol{\xi}} = [\boldsymbol{I}, -\boldsymbol{P}'^{\wedge}]. \tag{7.44}$$

将这两项相乘，就得到了2×6的雅可比矩阵：

$$\frac{\partial \boldsymbol{e}}{\partial \delta \boldsymbol{\xi}} = -\begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} & -\frac{f_x X' Y'}{Z'^2} & f_x + \frac{f_x X'^2}{Z'^2} & -\frac{f_x Y'}{Z'} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} & -f_y - \frac{f_y Y'^2}{Z'^2} & \frac{f_y X' Y'}{Z'^2} & \frac{f_y X'}{Z'} \end{bmatrix}. \tag{7.45}$$

这个雅可比矩阵描述了重投影误差关于相机位姿李代数的一阶变化关系。我们保留了前面的负号，这是因为误差是由观测值减预测值定义的。它当然也可反过来，定义成“预测值减观测值”的形式。在那种情况下，只要去掉前面的负号即可。此外，如果se(3)的定义方式是旋转在前，平移在后，只要把这个矩阵的前3列与后3列对调即可。

另一方面，除了优化位姿，我们还希望优化特征点的空间位置。因此，需要讨论$\boldsymbol{e}$关于空间点$\boldsymbol{P}$的导数。所幸这个导数矩阵相对来说容易一些。仍利用链式法则，有：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{P}} = \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{P}'} \frac{\partial \boldsymbol{P}'}{\partial \boldsymbol{P}}. \tag{7.46}$$

第一项在前面已推导，关于第二项，按照定义

$$\boldsymbol{P}' = \exp(\boldsymbol{\xi}^\wedge)\boldsymbol{P} = \boldsymbol{R}\boldsymbol{P} + \boldsymbol{t},$$

我们发现$\boldsymbol{P}'$对$\boldsymbol{P}$求导后将只剩下$\boldsymbol{R}$。于是：

$$\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{P}} = -\begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} \end{bmatrix} \boldsymbol{R}. \tag{7.47}$$

于是，我们推导出了观测相机方程关于相机位姿与特征点的两个导数矩阵。它们十分重要，能够在优化过程中提供重要的梯度方向，指导优化的迭代。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

7.8 实践：求解PnP

7.8.1 使用EPnP求解位姿

下面，我们通过实验理解一下PnP的过程。首先，我们用OpenCV提供的EPnP求解PnP问题，然后通过g2o对结果进行优化。由于PnP需要使用3D点，为了避免初始化带来的麻烦，我们使用了RGB-D相机中的深度图（1_depth.png）作为特征点的3D位置。首先来看OpenCV提供的PnP函数：

slambook/ch7/pose_estimation_3d2d.cpp（片段）

```
int main( int argc, char** argv )
{
    ......
    // 建立 3D 点
    Mat d1 = imread( argv[3], CV_LOAD_IMAGE_UNCHANGED );       // 深度图为 16 位无符号数，单通道图像
    Mat K = ( Mat_<double> ( 3,3 ) << 520.9, 0, 325.1, 0, 521.0, 249.7, 0, 0, 1 );
    vector<Point3f> pts_3d;
    vector<Point2f> pts_2d;
    for ( DMatch m:matches )
    {
        ushort d = d1.ptr<unsigned short> ( int(keypoints_1[m.queryIdx].pt.y) )[ int(keypoints_1[m.queryIdx].pt.x) ];
        if ( d == 0 )   // bad depth
            continue;
        float dd = d/1000.0;
        Point2d p1 = pixel2cam( keypoints_1[m.queryIdx].pt, K );
        pts_3d.push_back( Point3f(p1.x*dd, p1.y*dd, dd) );
        pts_2d.push_back ( keypoints_2[m.trainIdx].pt );
    }

    cout<<"3d-2d pairs: "<<pts_3d.size()<<endl;

```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
    Mat r, t;
    // 调用 OpenCV 的 PnP 求解，可选择 EPNP、DLS 等方法
    solvePnP( pts_3d, pts_2d, K, Mat(), r, t, false, cv::SOLVEPNP_EPNP );
    Mat R;
    cv::Rodrigues(r, R); // r 为旋转向量形式，用 Rodrigues 公式转换为矩阵

    cout<<"R="<<endl<<R<<endl;
    cout<<"t="<<endl<<t<<endl;
}
```

在例程中，得到配对特征点后，我们在第一个图的深度图中寻找它们的深度，并求出空间位置。以此空间位置为3D点，再以第二个图像的像素位置为2D点，调用EPnP求解PnP问题。程序输出如下：

```
% build/pose_estimation_3d2d 1.png 2.png d1.png d2.png
-- Max dist : 95.000000
-- Min dist : 4.000000
一共找到了79组匹配点
3d-2d pairs: 78
R=
[0.9977970937403702, -0.05195299069131867, 0.04125344205637558;
0.05073872610592159, 0.9982626103770279, 0.02995567385972873;
-0.04273805559942161, -0.02779653722084675, 0.9986995599889442]
t=
[-0.6455324432075111;
-0.05776758294184359;
0.2844565219506077]
```

读者可以对比先前2D-2D情况下求解的R,t看看有什么不同。可以看到，在有3D信息时，估计的R几乎是相同的，而t相差得较多。这是由于引入了新的深度信息所致。不过，由于Kinect采集的深度图本身会有一些误差，所以这里的3D点也不是准确的。我们会希望把位姿ξ和所有三维特征点P同时优化。

7.8.2 使用BA优化

下面演示如何进行BundleAdjustment。我们将使用前一步的估计值作为初始值。优化可以使用前面讲的Ceres或g2o库实现，这里采用g2o作为例子。

g2o的基本知识在第6讲中已经介绍过了。在使用g2o之前，我们要把问题建模成一个最小二乘的图优化问题，如图7-13所示。

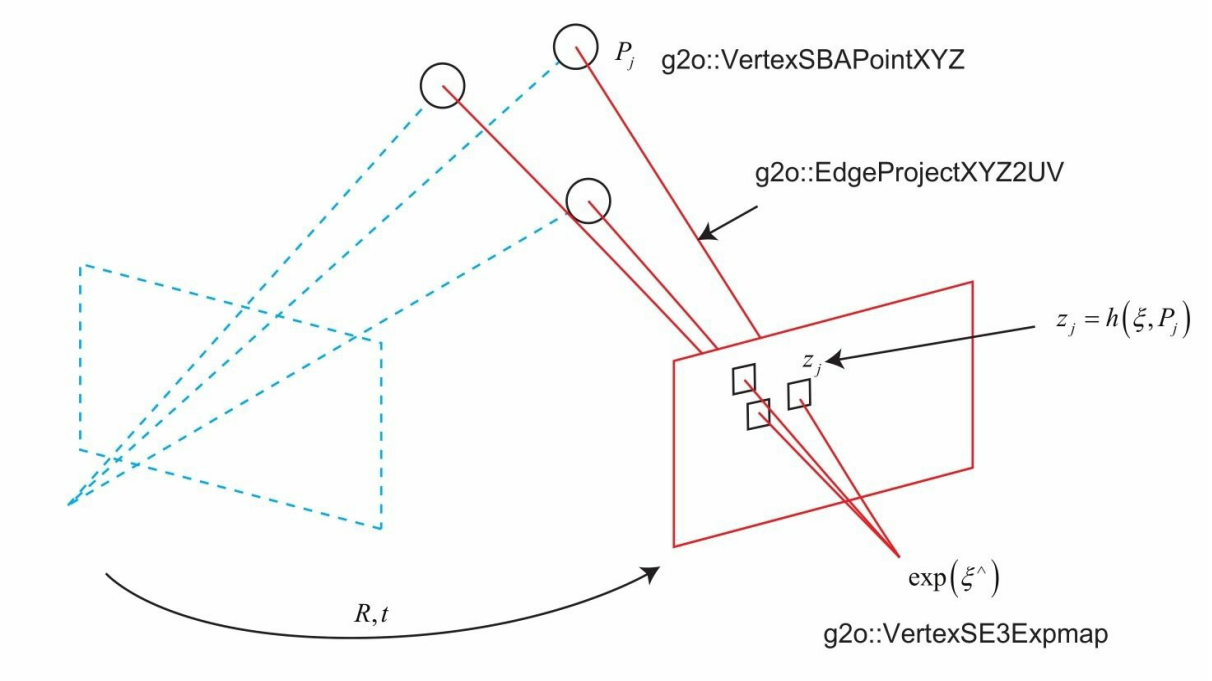


图7-13　PnP的Bundle Adjustment的图优化表示。

在这个图优化中，节点和边的选择如下：

1.节点：第二个相机的位姿节点$\xi \in se(3)$，所有特征点的空间位置$P \in \mathrm{R}^3$。

2.边：每个3D点在第二个相机中的投影，以观测方程来描述：

$$\boldsymbol{z}_j = h(\boldsymbol{\xi}, \boldsymbol{P}_j).$$

由于第一个相机位姿固定为零，我们没有把它写到优化变量里，但在习题中，希望你能够把第一个相机的位姿与观测也考虑进来。现在我们根据一组3D点和第二个图像中的2D投影，估计第二个相机的位姿。所以我们把第一个相机画成虚线，表明不希望考虑它。

g2o提供了许多关于BA的节点和边，我们不必自己从头实现所有的

计算。在g2o/types/sba/types_six_dof_expmap.h中则提供了李代数表达的节点和边。请读者打开这个文件，找到VertexSE3Expmap（李代数位姿）、VertexSBAPointXYZ（空间点位置）和EdgeProjectXYZ2UV（投影方程边）这三个类。我们来简单看一下它们的类定义，例如VertexSE3Expmap：

```
class G2O_TYPES_SBA_API VertexSE3Expmap : public BaseVertex<6, SE3Quat>{
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW

    VertexSE3Expmap();

    bool read(std::istream& is);

    bool write(std::ostream& os) const;


    virtual void setToOriginImpl() {
    _estimate = SE3Quat();
    }

    virtual void oplusImpl(const double* update_)  {
        Eigen::Map<const Vector6d> update(update_);
        setEstimate( SE3Quat::exp(update)*estimate());
    }
};
```

请注意它的模板参数。第一个参数6表示它内部存储的优化变量维度，可以看到这是一个6维的李代数。第二个参数是优化变量的类型，这里使用了g2o定义的相机位姿：SE3Quat。这个类内部使用了四元数加位移向量来存储位姿，但同时也支持李代数上的运算，例如对数映射（log函数）和李代数上增量（update函数）等操作。我们可以对照它的实现代码，看看g2o对李代数是如何操作的：

书籍汇总下载 http://t.cn/RDIAj5D

```
class G2O_TYPES_SBA_API VertexSBAPointXYZ : public BaseVertex<3, Vector3D>
{
 ......
};

class G2O_TYPES_SBA_API EdgeProjectXYZ2UV : public  BaseBinaryEdge<2, Vector2D,
VertexSBAPointXYZ, VertexSE3Expmap>
{
    ......
    void computeError()  {
        const VertexSE3Expmap* v1 = static_cast<const VertexSE3Expmap*>(_vertices
        [1]);
        const VertexSBAPointXYZ* v2 = static_cast<const VertexSBAPointXYZ*>(
        _vertices[0]);
        const CameraParameters * cam
            = static_cast<const CameraParameters *>(parameter(0));
        Vector2D obs(_measurement);
        _error = obs-cam->cam_map(v1->estimate().map(v2->estimate()));
    }
};
```

这里就不把整个类定义都搬过来了。从模板参数可以看到，空间点位置类的维度为3，类型是Eigen的Vector3D。另一方面，边EdgeProjectXYZ2UV连接了前面说的两个顶点，它的观测值为2维，由Vector2D表示，实际上就是空间点的像素坐标。它的误差计算函数表达了投影方程的误差计算方法，也就是我们前面提到的$z-h(\xi,P)$的方式。

现在，进一步观察EdgeProjectXYZ2UV的linearizeOplus函数的实现。这里用到了我们前面推导的雅可比矩阵：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
void EdgeProjectXYZ2UV::linearizeOplus() {
    VertexSE3Expmap * vj = static_cast<VertexSE3Expmap *>(_vertices[1]);
    SE3Quat T(vj->estimate());
    VertexSBAPointXYZ* vi = static_cast<VertexSBAPointXYZ*>(_vertices[0]);
    Vector3D xyz = vi->estimate();
    Vector3D xyz_trans = T.map(xyz);

    double x = xyz_trans[0];
    double y = xyz_trans[1];
    double z = xyz_trans[2];
    double z_2 = z*z;

    const CameraParameters * cam = static_cast<const CameraParameters *>(parameter
    (0));

    Matrix<double,2,3,Eigen::ColMajor> tmp;
    tmp(0,0) = cam->focal_length;
    tmp(0,1) = 0;
    tmp(0,2) = -x/z*cam->focal_length;

    tmp(1,0) = 0;
    tmp(1,1) = cam->focal_length;
    tmp(1,2) = -y/z*cam->focal_length;

    _jacobianOplusXi =  -1./z * tmp * T.rotation().toRotationMatrix();

    _jacobianOplusXj(0,0) =  x*y/z_2 *cam->focal_length;
    _jacobianOplusXj(0,1) = -(1+(x*x/z_2)) *cam->focal_length;
    _jacobianOplusXj(0,2) = y/z *cam->focal_length;
    _jacobianOplusXj(0,3) = -1./z *cam->focal_length;
    _jacobianOplusXj(0,4) = 0;
    _jacobianOplusXj(0,5) = x/z_2 *cam->focal_length;

    _jacobianOplusXj(1,0) = (1+y*y/z_2) *cam->focal_length;
    _jacobianOplusXj(1,1) = -x*y/z_2 *cam->focal_length;
    _jacobianOplusXj(1,2) = -x/z *cam->focal_length;
    _jacobianOplusXj(1,3) = 0;
    _jacobianOplusXj(1,4) = -1./z *cam->focal_length;
    _jacobianOplusXj(1,5) = y/z_2 *cam->focal_length;
}
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

仔细研究此段代码，我们会发现它与式(7.45)和(7.47)是一致的。成员变量“_jacobianOplusXi”是误差到空间点的导数，“_jacobianOplusXj”是误差到相机位姿的导数，以李代数的左乘扰动表达。稍有差别的是，g2o的相机里用f统一描述f_x,f_y，并且李代数定义顺序不同（g2o是旋转在前，平移在后；我们是平移在前，旋转在后），所以矩阵前3列和后3列与我们的定义是相反的，此外都一致。

值得一提的是，我们亦可自己实现相机位姿节点，并使用Sophus::SE3来表达位姿，提供类似的求导过程。然而，既然g2o已经提供了这样的类，在没有额外要求的情况下，自己重新实现就没有必要了。现在，我们在上一个PnP例程的基础上，加上g2o提供的Bundle Adjustment。

slambook/ch7/pose_estimation_3d2d.cpp（片段）

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
void bundleAdjustment (
    const vector< Point3f > points_3d,
    const vector< Point2f > points_2d,
    const Mat& K,
    Mat& R, Mat& t )
{
    // 初始化g2o
    typedef g2o::BlockSolver< g2o::BlockSolverTraits<6,3> > Block;  // pose 维度为
    6, landmark 维度为 3
    Block::LinearSolverType* linearSolver = new g2o::LinearSolverCSparse<Block::
    PoseMatrixType>();
    Block* solver_ptr = new Block( linearSolver );
    g2o::OptimizationAlgorithmLevenberg* solver = new g2o::
    OptimizationAlgorithmLevenberg( solver_ptr );
    g2o::SparseOptimizer optimizer;
    optimizer.setAlgorithm( solver );

    // vertex
    g2o::VertexSE3Expmap* pose = new g2o::VertexSE3Expmap(); // camera pose
    Eigen::Matrix3d R_mat;
    R_mat <<
        R.at<double>(0,0), R.at<double>(0,1), R.at<double>(0,2),
        R.at<double>(1,0), R.at<double>(1,1), R.at<double>(1,2),
        R.at<double>(2,0), R.at<double>(2,1), R.at<double>(2,2);
    pose->setId(0);
    pose->setEstimate( g2o::SE3Quat(
        R_mat,
        Eigen::Vector3d( t.at<double>(0,0), t.at<double>(1,0), t.at<double>(2,0))
    ) );
    optimizer.addVertex( pose );

    int index = 1;
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
    for ( const Point3f p:points_3d )   // landmarks
    {
        g2o::VertexSBAPointXYZ* point = new g2o::VertexSBAPointXYZ();
        point->setId( index++ );
        point->setEstimate( Eigen::Vector3d(p.x, p.y, p.z) );
        point->setMarginalized( true );
        optimizer.addVertex( point );
    }

    // parameter: camera intrinsics
    g2o::CameraParameters* camera = new g2o::CameraParameters(
    K.at<double>(0,0), Eigen::Vector2d(K.at<double>(0,2), K.at<double>(1,2)), 0
    );
    camera->setId(0);
    optimizer.addParameter( camera );

    // edges
    index = 1;
    for ( const Point2f p:points_2d )
    {
        g2o::EdgeProjectXYZ2UV* edge = new g2o::EdgeProjectXYZ2UV();
        edge->setId( index );
        edge->setVertex( 0, dynamic_cast<g2o::VertexSBAPointXYZ*> (optimizer.vertex
        (index)) );
        edge->setVertex( 1, pose );
        edge->setMeasurement( Eigen::Vector2d( p.x, p.y ) );
        edge->setParameterId(0,0);
        edge->setInformation( Eigen::Matrix2d::Identity() );
        optimizer.addEdge(edge);
        index++;
    }

    chrono::steady_clock::time_point t1 = chrono::steady_clock::now();
    optimizer.setVerbose( true );
    optimizer.initializeOptimization();
    optimizer.optimize(100);
    chrono::steady_clock::time_point t2 = chrono::steady_clock::now();
    chrono::duration<double> time_used = chrono::duration_cast<chrono::duration<
    double>>(t2-t1);
    cout<<"optimization costs time: "<<time_used.count()<<" seconds."<<endl;

    cout<<endl<<"after optimization:"<<endl;
    cout<<"T="<<endl<<Eigen::Isometry3d( pose->estimate() ).matrix()<<endl;
}
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

程序大体上和第6讲的g2o类似。我们首先声明了g2o图优化，并配置优化求解器和梯度下降方法。然后根据估计到的特征点，将位姿和空间点放到图中。最后调用优化函数进行求解。读者可以看到优化的结果如下：

```
calling bundle adjustment
iteration= 0  chi2= 1.083180 time= 0.000107183 cumTime= 0.000107183 edges= 76 schur = 1 lambda= 78.907222 levenbergIter= 1
iteration= 1  chi2= 0.000798 time= 5.8615e-05  cumTime= 0.000165798 edges= 76 schur = 1 lambda= 26.302407 levenbergIter= 1
iteration= 2  chi2= 0.000000 time= 3.0203e-05  cumTime= 0.000196001 edges= 76 schur = 1 lambda= 17.534938 levenbergIter= 1
...... 中间过程略
iteration= 11 chi2= 0.000000 time= 2.8394e-05  cumTime= 0.000525203 edges= 76 schur = 1 lambda= 11209.703029 levenbergIter= 1
optimization costs time: 0.00132938 seconds.

after optimization:
T=
0.997776 -0.0519476  0.0417755  -0.649778
0.050735   0.998274  0.0295806 -0.0545231
-0.0432401 -0.0273953   0.998689   0.295564
0          0          0          1
```

迭代11轮后，列文伯格—马夸尔特方法发现优化目标函数接近不变，于是停止优化。我们输出了最后得到的位姿变换矩阵T，对比之前直接做PnP的结果，大约在小数点后第3位发生了一些变化。这主要是由于我们同时优化了特征点和相机位姿导致的。

Bundle Adjustment是一种通用的做法。它可以不限于两幅图像。我们完全可以放入多幅图像匹配到的位姿和空间点进行迭代优化，甚至可以把整个SLAM过程放进来。那种做法规模较大，主要在后端使用，我们会在第10讲再次遇到这个问题。在前端，我们通常考虑局部相机位姿和特征点的小型Bundle Adjustment问题，希望对它进行实时求解和优化。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

7.9 3D-3D：ICP

最后，我们来介绍3D-3D的位姿估计问题。假设我们有一组配对好的3D点（比如我们对两幅RGB-D图像进行了匹配）：

$$\boldsymbol{P} = \{\boldsymbol{p}_1, \cdots, \boldsymbol{p}_n\}, \quad \boldsymbol{P}' = \{\boldsymbol{p}'_1, \cdots, \boldsymbol{p}'_n\},$$

现在，想要找一个欧氏变换$\boldsymbol{R}$,$\boldsymbol{t}$，使得：

$$\forall i, \boldsymbol{p}_i = \boldsymbol{R}\boldsymbol{p}'_i + \boldsymbol{t}.$$

这个问题可以用迭代最近点（Iterative Closest Point，ICP）求解。读者应该注意到了，3D-3D位姿估计问题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，和相机并没有关系。因此，在激光SLAM中也会碰到ICP，不过由于激光数据特征不够丰富，我们无从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，所以这个方法称为迭代最近点。而在视觉中，特征点为我们提供了较好的匹配关系，所以整个问题就变得更简单了。在RGB-D SLAM中，可以用这种方式估计相机位姿。下文我们用ICP指代匹配好的两组点间的运动估计问题。

和PnP类似，ICP的求解也分为两种方式：利用线性代数的求解（主要是SVD），以及利用非线性优化方式的求解（类似于Bundle Adjustment）。下面分别进行介绍。

7.9.1 SVD方法

首先来看以SVD为代表的代数方法。根据前面描述的ICP问题，我们先定义第i对点的误差项：

$$\boldsymbol{e}_i = \boldsymbol{p}_i - (\boldsymbol{R}\boldsymbol{p}'_i + \boldsymbol{t}). \tag{7.48}$$

然后，构建最小二乘问题，求使误差平方和达到极小的$\boldsymbol{R}$,$\boldsymbol{t}$：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$\min_{\boldsymbol{R},\boldsymbol{t}} J = \frac{1}{2}\sum_{i=1}^{n} \|(\boldsymbol{p}_i - (\boldsymbol{R}\boldsymbol{p}_i' + \boldsymbol{t}))\|_2^2. \tag{7.49}$$

下面来推导它的求解方法。首先，定义两组点的质心：

$$\boldsymbol{p} = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{p}_i), \quad \boldsymbol{p}' = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{p}_i'). \tag{7.50}$$

请注意，质心是没有下标的。随后，在误差函数中做如下的处理：

$$\begin{aligned}
\frac{1}{2}\sum_{i=1}^{n} \|\boldsymbol{p}_i - (\boldsymbol{R}\boldsymbol{p}_i' + \boldsymbol{t})\|^2 &= \frac{1}{2}\sum_{i=1}^{n} \|\boldsymbol{p}_i - \boldsymbol{R}\boldsymbol{p}_i' - \boldsymbol{t} - \boldsymbol{p} + \boldsymbol{R}\boldsymbol{p}' + \boldsymbol{p} - \boldsymbol{R}\boldsymbol{p}'\|^2 \\
&= \frac{1}{2}\sum_{i=1}^{n} \|(\boldsymbol{p}_i - \boldsymbol{p} - \boldsymbol{R}(\boldsymbol{p}_i' - \boldsymbol{p}')) + (\boldsymbol{p} - \boldsymbol{R}\boldsymbol{p}' - \boldsymbol{t})\|^2 \\
&= \frac{1}{2}\sum_{i=1}^{n} (\|\boldsymbol{p}_i - \boldsymbol{p} - \boldsymbol{R}(\boldsymbol{p}_i' - \boldsymbol{p}')\|^2 + \|\boldsymbol{p} - \boldsymbol{R}\boldsymbol{p}' - \boldsymbol{t}\|^2 + \\
&\quad 2(\boldsymbol{p}_i - \boldsymbol{p} - \boldsymbol{R}(\boldsymbol{p}_i' - \boldsymbol{p}'))^T (\boldsymbol{p} - \boldsymbol{R}\boldsymbol{p}' - \boldsymbol{t})).
\end{aligned}$$

注意到交叉项部分中$(\boldsymbol{p}_i - \boldsymbol{p} - \boldsymbol{R}(\boldsymbol{p}_i' - \boldsymbol{p}'))$在求和之后为零，因此优化目标函数可以简化为

$$\min_{\boldsymbol{R},\boldsymbol{t}} J = \frac{1}{2}\sum_{i=1}^{n} \|\boldsymbol{p}_i - \boldsymbol{p} - \boldsymbol{R}(\boldsymbol{p}_i' - \boldsymbol{p}')\|^2 + \|\boldsymbol{p} - \boldsymbol{R}\boldsymbol{p}' - \boldsymbol{t}\|^2. \tag{7.51}$$

仔细观察左右两项，我们发现左边只和旋转矩阵$\boldsymbol{R}$相关，而右边既有$\boldsymbol{R}$也有$\boldsymbol{t}$，但只和质心相关。只要我们获得了$\boldsymbol{R}$，令第二项为零就能得到$\boldsymbol{t}$。于是，ICP可以分为以下三个步骤求解：

1.计算两组点的质心位置$\boldsymbol{p}, \boldsymbol{p}'$，然后计算每个点的去质心坐标：

$$\boldsymbol{q}_i = \boldsymbol{p}_i - \boldsymbol{p}, \quad \boldsymbol{q}_i' = \boldsymbol{p}_i' - \boldsymbol{p}'.$$

2.根据以下优化问题计算旋转矩阵：

$$\boldsymbol{R}^* = \arg\min_{\boldsymbol{R}} \frac{1}{2}\sum_{i=1}^{n} \|\boldsymbol{q}_i - \boldsymbol{R}\boldsymbol{q}_i'\|^2. \tag{7.52}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

3.根据第2步的R计算t：

$$\boldsymbol{t}^* = \boldsymbol{p} - \boldsymbol{R}\boldsymbol{p}'. \tag{7.53}$$

我们看到，只要求出了两组点之间的旋转，平移量是非常容易得到的。所以我们重点关注R的计算。展开关于R的误差项，得：

$$\frac{1}{2}\sum_{i=1}^{n}\left\|\boldsymbol{q}_i - \boldsymbol{R}\boldsymbol{q}_i'\right\|^2 = \frac{1}{2}\sum_{i=1}^{n}\boldsymbol{q}_i^{\mathrm{T}}\boldsymbol{q}_i + \boldsymbol{q}_i'^{\mathrm{T}}\boldsymbol{R}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{q}_i' - 2\boldsymbol{q}_i^{\mathrm{T}}\boldsymbol{R}\boldsymbol{q}_i'. \tag{7.54}$$

注意到第一项和R无关，第二项由于$R^{\mathrm{T}}R=I$，亦与R无关。因此，实际上优化目标函数变为

$$\sum_{i=1}^{n} -\boldsymbol{q}_i^{\mathrm{T}}\boldsymbol{R}\boldsymbol{q}_i' = \sum_{i=1}^{n} -\mathrm{tr}\left(\boldsymbol{R}\boldsymbol{q}_i'\boldsymbol{q}_i^{\mathrm{T}}\right) = -\mathrm{tr}\left(\boldsymbol{R}\sum_{i=1}^{n}\boldsymbol{q}_i'\boldsymbol{q}_i^{\mathrm{T}}\right). \tag{7.55}$$

接下来，我们介绍怎样通过SVD解出上述问题中最优的R。关于最优性的证明较为复杂，感兴趣的读者请参考文献[50,51]。为了解R，先定义矩阵：

$$\boldsymbol{W} = \sum_{i=1}^{n}\boldsymbol{q}_i\boldsymbol{q}_i'^{\mathrm{T}}. \tag{7.56}$$

W是一个3×3的矩阵，对W进行SVD分解，得：

$$\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}. \tag{7.57}$$

其中，Σ为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

$$\boldsymbol{R} = \boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}. \tag{7.58}$$

解得R后，按式(7.53)求解t即可。

7.9.2 非线性优化方法

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

求解ICP的另一种方式是使用非线性优化，以迭代的方式去找最优值。该方法和我们前面讲述的PnP非常相似。以李代数表达位姿时，目标函数可以写成

$$\min_{\boldsymbol{\xi}} = \frac{1}{2}\sum_{i=1}^{n} \| (\boldsymbol{p}_i - \exp(\boldsymbol{\xi}^\wedge)\boldsymbol{p}_i') \|_2^2. \tag{7.59}$$

单个误差项关于位姿的导数在前面已推导，使用李代数扰动模型即可：

$$\frac{\partial \boldsymbol{e}}{\partial \delta \boldsymbol{\xi}} = -(\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{p}_i')^\odot. \tag{7.60}$$

于是，在非线性优化中只需不断迭代，就能找到极小值。而且，可以证明[6]，ICP问题存在唯一解或无穷多解的情况。在唯一解的情况下，只要能找到极小值解，那么这个极小值就是全局最优值——因此不会遇到局部极小而非全局最小的情况。这也意味着ICP求解可以任意选定初始值。这是已匹配点时求解ICP的一大好处。

需要说明的是，我们这里讲的ICP是指已由图像特征给定了匹配的情况下进行位姿估计的问题。在匹配已知的情况下，这个最小二乘问题实际上具有解析解[52,53,54]，所以并没有必要进行迭代优化。ICP的研究者们往往更加关心匹配未知的情况。不过，在RGB-D SLAM中，由于一个像素的深度数据可能测量不到，所以我们可以混合着使用PnP和ICP优化：对于深度已知的特征点，建模它们的3D-3D误差；对于深度未知的特征点，则建模3D-2D的重投影误差。于是，可以将所有的误差放在同一个问题中考虑，使得求解更加方便。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

7.10 实践：求解ICP

7.10.1 SVD方法

下面演示一下如何使用SVD及非线性优化来求解ICP。本节我们使用两幅RGB-D图像，通过特征匹配获取两组3D点，最后用ICP计算它们的位姿变换。由于OpenCV目前还没有计算两组带匹配点的ICP的方法，而且它的原理也并不复杂，所以我们自己来实现一个ICP。

slambook/ch7/pose_estimation_3d3d.cpp（片段）

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
void pose_estimation_3d3d(
    const vector<Point3f>& pts1,
    const vector<Point3f>& pts2,
    Mat& R, Mat& t
)
{
    Point3f p1, p2;     // center of mass
    int N = pts1.size();
    for ( int i=0; i<N; i++ )
    {
        p1 += pts1[i];
        p2 += pts2[i];
    }
    p1 /= N;    p2 /= N;
    vector<Point3f>     q1(N), q2(N); // remove the center
    for ( int i=0; i<N; i++ )
    {
        q1[i] = pts1[i] - p1;
        q2[i] = pts2[i] - p2;
    }

    // compute q1*q2^T
    Eigen::Matrix3d W = Eigen::Matrix3d::Zero();
    for ( int i=0; i<N; i++ )
    {
        W += Eigen::Vector3d( q1[i].x, q1[i].y, q1[i].z ) * Eigen::Vector3d( q2[i].
        x, q2[i].y, q2[i].z ).transpose();
    }
    cout<<"W="<<W<<endl;

    // SVD on W
    Eigen::JacobiSVD<Eigen::Matrix3d> svd(W, Eigen::ComputeFullU|Eigen::
    ComputeFullV);
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
32      Eigen::Matrix3d U = svd.matrixU();
33      Eigen::Matrix3d V = svd.matrixV();
34      cout<<"U="<<U<<endl;
35      cout<<"V="<<V<<endl;
36
37      Eigen::Matrix3d R_ = U*(V.transpose());
38      Eigen::Vector3d t_ = Eigen::Vector3d( p1.x, p1.y, p1.z ) - R_ * Eigen::Vector3d
        ( p2.x, p2.y, p2.z );
39
40      // convert to cv::Mat
41      R = ( Mat_<double>(3,3) <<
42          R_(0,0), R_(0,1), R_(0,2),
43          R_(1,0), R_(1,1), R_(1,2),
44          R_(2,0), R_(2,1), R_(2,2)
45      );
46      t = ( Mat_<double>(3,1) << t_(0,0), t_(1,0), t_(2,0) );
47  }
```

ICP的实现方式和前文讲述的是一致的。我们调用Eigen进行SVD，然后计算R,t矩阵。我们输出了匹配后的结果，不过请注意，由于前面的推导是按照$p_i=Rp'_i+t$进行的，这里的R,t是第二帧到第一帧的变换，与前面PnP部分是相反的。所以在输出结果中，我们同时打印了逆变换：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
% build/pose_estimation_3d3d 1.png 2.png 1_depth.png 2_depth.png
-- Max dist : 95.000000
-- Min dist : 4.000000
一共找到了 79 组匹配点
3d-3d pairs: 74
W=  298.51 -14.1815  41.0456
-44.8208  107.825 -164.404
78.1978 -163.954  271.439
U=  0.474143  -0.880373 -0.0114952
-0.460275  -0.258979   0.849163
0.750556   0.397334   0.528006
V=  0.535211  -0.844064 -0.0332488
-0.434767  -0.309001    0.84587
0.724242   0.438263   0.532352
ICP via SVD results:
R = [0.9972395976914055, 0.05617039049497474, -0.04855998381307948;
-0.05598344580804095, 0.9984181433274515, 0.005202390798842771;
0.04877538920134394, -0.002469474885032297, 0.998806719591959]
t = [0.7086246277241892;
-0.2775515782948791;
-0.1559573762377209]
R_inv = [0.9972395976914055, -0.05598344580804095, 0.04877538920134394;
0.05617039049497474, 0.9984181433274515, -0.002469474885032297;
-0.04855998381307948, 0.005202390798842771, 0.998806719591959]
t_inv = [-0.7145999506834847;
0.2369236766013986;
0.1916260075851286]
```

读者可以比较一下ICP与PnP、对极几何的运动估计结果之间的差异。可以认为，在这个过程中我们使用了越来越多的信息（没有深度—有一个图的深度—有两个图的深度），因此，在深度准确的情况下，得到的估计也将越来越准确。但是，由于Kinect的深度图存在噪声，而且有可能存在数据丢失的情况，使得我们不得不丢弃一些没有深度数据的特征点。这可能导致ICP的估计不够准确，并且，如果特征点丢弃得太多，可能引起由于特征点太少，无法进行运动估计的情况。

书籍汇总下载 http://t.cn/RDIAj5D

7.10.2 非线性优化方法

下面考虑用非线性优化来计算ICP。我们依然使用李代数来表达相机位姿。与SVD思路不同的地方在于，在优化中我们不仅考虑相机的位姿，同时会优化3D点的空间位置。对我们来说，RGB-D相机每次可以观测到路标点的三维位置，从而产生一个3D观测数据。不过，由于g2o/sba中没有提供3D到3D的边，而我们又想使用g2o/sba中李代数实现的位姿节点，所以最好的方式是自定义一种这样的边，并向g2o提供解析求导方式。

slambook/ch7/pose_estimation_3d3d.cpp

```
class EdgeProjectXYZRGBDPoseOnly : public g2o::BaseUnaryEdge<3, Eigen::Vector3d,
g2o::VertexSE3Expmap>
{
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW;
    EdgeProjectXYZRGBDPoseOnly( const Eigen::Vector3d& point ) :
        _point(point) {}

    virtual void computeError()
    {
        const g2o::VertexSE3Expmap* pose = static_cast<const g2o::VertexSE3Expmap*>
         ( _vertices[0] );
        // measurement is p, point is p'
        _error = _measurement - pose->estimate().map( _point );
    }

    virtual void linearizeOplus()
    {
        g2o::VertexSE3Expmap* pose = static_cast<g2o::VertexSE3Expmap *>(_vertices
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
        [0]);
        g2o::SE3Quat T(pose->estimate());
        Eigen::Vector3d xyz_trans = T.map(_point);
        double x = xyz_trans[0];
        double y = xyz_trans[1];
        double z = xyz_trans[2];

        _jacobianOplusXi(0,0) = 0;
        _jacobianOplusXi(0,1) = -z;
        _jacobianOplusXi(0,2) = y;
        _jacobianOplusXi(0,3) = -1;
        _jacobianOplusXi(0,4) = 0;
        _jacobianOplusXi(0,5) = 0;

        _jacobianOplusXi(1,0) = z;
        _jacobianOplusXi(1,1) = 0;
        _jacobianOplusXi(1,2) = -x;
        _jacobianOplusXi(1,3) = 0;
        _jacobianOplusXi(1,4) = -1;
        _jacobianOplusXi(1,5) = 0;

        _jacobianOplusXi(2,0) = -y;
        _jacobianOplusXi(2,1) = x;
        _jacobianOplusXi(2,2) = 0;
        _jacobianOplusXi(2,3) = 0;
        _jacobianOplusXi(2,4) = 0;
        _jacobianOplusXi(2,5) = -1;
    }

    bool read ( istream& in ) {}
    bool write ( ostream& out ) const {}
protected:
    Eigen::Vector3d _point;
};
```

这是一个一元边，写法类似于前面提到的g2o::EdgeSE3ProjectXYZ，不过观测量从2维变成了3维，内部没有相机模型，并且只关联到一个节点。请读者注意这里雅可比矩阵的书写，它

必须与我们前面的推导一致。雅可比矩阵给出了关于相机位姿的导数，是一个3×6的矩阵。

调用g2o进行优化的代码是相似的，我们设定好图优化的节点和边即可。这部分代码请读者查看源文件，这里不再列出。现在，来看看优化的结果：

```
calling bundle adjustment
iteration= 0 chi2= 452884.696837 time= 3.8443e-05 cumTime= 3.8443e-05  edges= 74
schur= 0
iteration= 1 chi2= 452762.638918 time= 1.436e-05  cumTime= 5.2803e-05  edges= 74
schur= 0
iteration= 2 chi2= 452762.618632 time= 1.1943e-05
...... 中间略
iteration= 9 chi2= 452762.618615 time= 1.0772e-05 cumTime= 0.000140108 edges= 74
schur= 0
optimization costs time: 0.000528066 seconds.

after optimization:
T=
0.99724   0.0561704    -0.04856    0.708625
-0.0559834    0.998418  0.00520239   -0.277551
0.0487754 -0.00246948    0.998807   -0.155957
0           0           0           1
```

我们发现，只迭代一次后总体误差就已经稳定不变，说明仅在一次迭代之后算法即已收敛。从位姿求解的结果可以看出，它和前面SVD给出的位姿结果几乎一模一样，这说明SVD已经给出了优化问题的解析解。所以，本实验中可以认为SVD给出的结果是相机位姿的最优值。

需要说明的是，在本例的ICP中，我们使用了在两个图都有深度读数的特征点。然而事实上，只要其中一个图深度确定，我们就能用类似于PnP的误差方式，把它们也加到优化中来。同时，除了相机位姿之外，将空间点也作为优化变量考虑，亦是一种解决问题的方式。我们应当清楚，实际的求解是非常灵活的，不必拘泥于某种固定的形式。如果同时考虑点和相机，整个问题就变得更自由了，你可能会得到其他的

书籍汇总下载 http://t.cn/RDIAj5D

解。比如，可以让相机少转一些角度，而把点多移动一些。这从另一侧面反映出，在Bundle Adjustment里面，我们会希望有尽可能多的约束，因为多次观测会带来更多的信息，使我们能够更准确地估计每个变量。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

7.11 小结

本讲介绍了基于特征点的视觉里程计中的几个重要的问题。包括：

1.特征点是如何提取并匹配的。

2.如何通过2D-2D的特征点估计相机运动。

3.如何从2D-2D的匹配估计一个点的空间位置。

4.3D-2D的PnP问题，其线性解法和Bundle Adjustment解法。

5.3D-3D的ICP问题，其线性解法和Bundle Adjustment解法。

本讲内容较为丰富，且结合应用了前几讲的基本知识。读者若觉得理解有困难，可以对前面知识稍加回顾。最好亲自做一遍实验，以理解整个运动估计的内容。

需要解释的是，为保证行文流畅，我们省略了大量关于某些特殊情况的讨论。例如，如果在对极几何求解过程中给定的特征点共面，会发生什么情况（这在单应矩阵H中提到了）？共线又会发生什么情况？在PnP和ICP中若给定这样的解，又会导致什么情况？求解算法能否识别这些特殊的情况，并报告所得的解可能不可靠？尽管它们都是值得研究和探索的，然而对它们的讨论势必让本书变得特别烦琐。而且在工程实现中，这些情况甚少出现，所以本书介绍的方法，是指在实际工程中能够有效运行的方法，我们假定了那些少见的情况并不发生。如果你关心这些少见的情况，可以阅读[3]等论文，在文献中我们经常会研究一些特殊情况下的解决方案。

习题

1.除了本书介绍的ORB特征点外，你还能找到哪些特征点？请说说SIFT或SURF的原理，并对比它们与ORB之间的优劣。

2.设计程序调用OpenCV中的其他种类特征点。统计在提取1000个特征点时在你的机器上所用的时间。

3.*我们发现，OpenCV提供的ORB特征点在图像当中分布不够均

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

匀。你是否能够找到或提出让特征点分布更加均匀的方法？

4.研究FLANN为何能够快速处理匹配问题。除了FLANN之外，还有哪些可以加速匹配的手段？

5.把演示程序使用的EPnP改成其他PnP方法，并研究它们的工作原理。

6.在PnP优化中，将第一个相机的观测也考虑进来，程序应如何书写？最后结果会有何变化？

7.在ICP程序中，将空间点也作为优化变量考虑进来，程序应如何书写？最后结果会有何变化？

8.*在特征点匹配过程中，不可避免地会遇到误匹配的情况。如果我们把错误匹配输入到PnP或ICP中，会发生怎样的情况？你能想到哪些避免误匹配的方法？

9.*使用Sophus的SE3类，自己设计g2o的节点与边，实现PnP和ICP的优化。

10.*在Ceres中实现PnP和ICP的优化。

---

[1]金字塔是指对图像进行不同层次的降采样，以获得不同分辨率的图像。

[2]也就是说，在等式一侧乘以任意非零常数时，我们认为等式仍是成立的。

[3]在SfM研究中则有可能是未知而有待估计的。

[4]请注意，这和SE(3)中的变换矩阵T是不同的。

[5]$\exp(\xi^\wedge)P_i$结果是4×1的，而其左侧的K是3×3的，所以必须把$\exp(\xi^\wedge)P_i$的前三维取出来，变成三维的非齐次坐标。这在前面讲过。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

第8讲　视觉里程计2

主要目标

1.理解光流法跟踪特征点的原理。

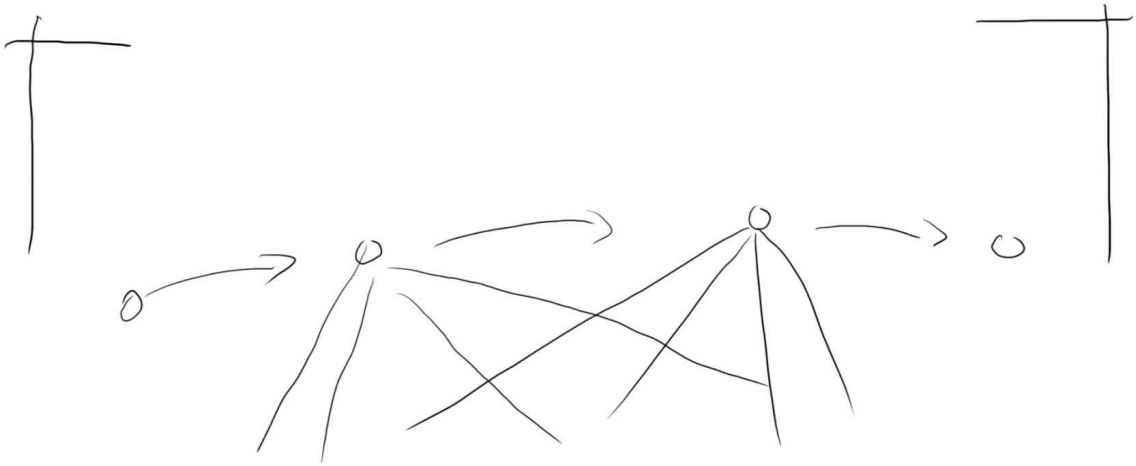
2.理解直接法是如何估计相机位姿的。

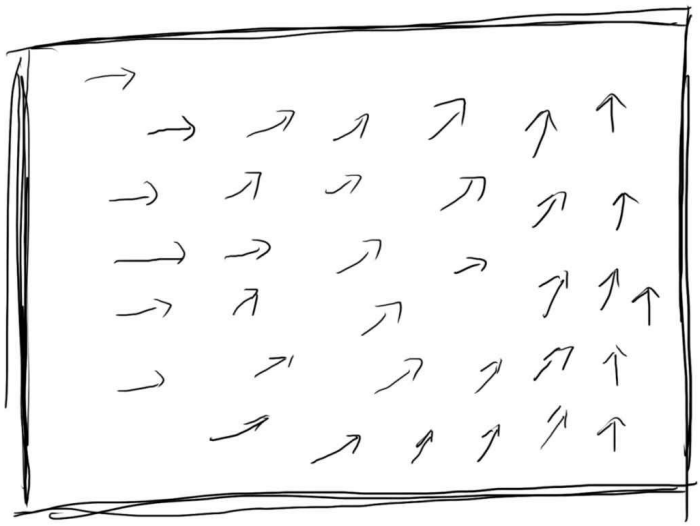
3.使用g2o进行直接法的计算。

直接法是视觉里程计另一主要分支，它与特征点法有很大不同。虽然它还没有成为现在VO中的主流，但经过近几年的发展，直接法在一定程度上已经能和特征点法平分秋色。本讲我们将介绍直接法的原理，并利用g2o实现直接法中的一些核心算法。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D





$$\min \| I_1(p) - I_2(K(Rp+t)) \|_2$$

$$J = \frac{\partial I}{\partial u} \frac{\partial u}{\partial \xi} \frac{\partial \xi}{\partial \xi}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

8.1 直接法的引出

上一讲我们介绍了使用特征点估计相机运动的方法。尽管特征点法在视觉里程计中占据主流地位，但研究者们还是认识到它至少有以下几个缺点：

1.关键点的提取与描述子的计算非常耗时。实践当中，SIFT目前在CPU上是无法实时计算的，而ORB也需要近20ms的计算。如果整个SLAM以30毫秒/帧的速度运行，那么一大半时间都将花在计算特征点上。

2.使用特征点时，忽略了除特征点以外的所有信息。一幅图像有几十万个像素，而特征点只有几百个。只使用特征点丢弃了大部分可能有用的图像信息。

3.相机有时会运动到特征缺失的地方，这些地方往往没有明显的纹理信息。例如，有时我们会面对一堵白墙，或者一个空荡荡的走廊。这些场景下特征点数量会明显减少，我们可能找不到足够的匹配点来计算相机运动。

我们看到使用特征点确实存在一些问题。有没有什么办法能够克服这些缺点呢？我们有以下几种思路：

•保留特征点，但只计算关键点，不计算描述子。同时，使用光流法（OpticalFlow）来跟踪特征点的运动。这样可以回避计算和匹配描述子带来的时间，但光流本身的计算需要一定时间。

•只计算关键点，不计算描述子。同时，使用直接法（Direct Method）来计算特征点在下一时刻图像中的位置。这同样可以跳过描述子的计算过程，而且直接法的计算更加简单。

•既不计算关键点，也不计算描述子，而是根据像素灰度的差异，直接计算相机运动。

第一种方法仍然使用特征点，只是把匹配描述子替换成了光流跟踪，估计相机运动时仍使用对极几何、PnP或ICP算法。而在后两种方法中，我们会根据图像的像素灰度信息来计算相机运动，它们都称为直接

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

法。

使用特征点法估计相机运动时，我们把特征点看作固定在三维空间的不动点。根据它们在相机中的投影位置，通过最小化重投影误差（Reprojection error）来优化相机运动。在这个过程中，我们需要精确地知道空间点在两个相机中投影后的像素位置——这也就是我们要对特征进行匹配或跟踪的原因。同时，我们也知道，计算、匹配特征需要付出大量的计算量。相对地，在直接法中，我们并不需要知道点与点之间的对应关系，而是通过最小化光度误差（Photometric error）来求得它们。

直接法是本讲介绍的重点。它是为了克服特征点法的上述缺点而存在的。直接法根据像素的亮度信息估计相机的运动，可以完全不用计算关键点和描述子，于是，既避免了特征的计算时间，也避免了特征缺失的情况。只要场景中存在明暗变化（可以是渐变，不形成局部的图像梯度），直接法就能工作。根据使用像素的数量，直接法分为稀疏、稠密和半稠密三种。相比于特征点法只能重构稀疏特征点（稀疏地图），直接法还具有恢复稠密或半稠密结构的能力。

历史上，早期也有对直接法的使用[55]。随着一些使用直接法的开源项目的出现（如SVO[56]、LSD-SLAM[57]等），它们逐渐地走上主流舞台，成为视觉里程计算法中重要的一部分。

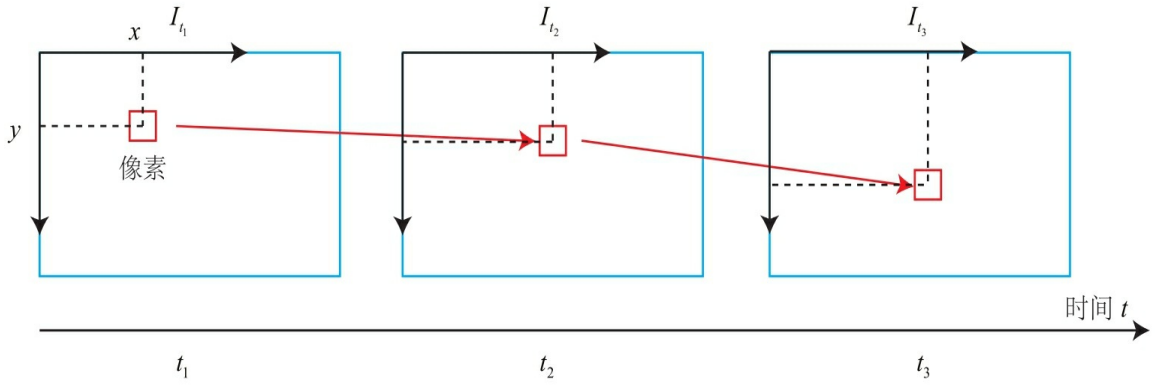
电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

8.2 光流（OpticalFlow）

直接法是从光流演变而来的。它们非常相似，具有相同的假设条件。光流描述了像素在图像中的运动，而直接法则附带着一个相机运动模型。为了说明直接法，我们先来介绍一下光流。

光流是一种描述像素随时间在图像之间运动的方法，如图8-1所示。随着时间的流逝，同一个像素会在图像中运动，而我们希望追踪它的运动过程。其中，计算部分像素运动的称为稀疏光流，计算所有像素的称为稠密光流。稀疏光流以Lucas-Kanade光流为代表，并可以在SLAM中用于跟踪特征点位置。因此，本节主要介绍Lucas-Kanade光流，亦称LK光流。



灰度不变假设：$I(x_1,y_1,t_1)=I(x_2,y_2,t_2)=I(x_3,y_3,t_3)$

图8-1　LK光流法示意图。

Lucas-Kanade光流

在LK光流中，我们认为来自相机的图像是随时间变化的。图像可以看作时间的函数：$I(t)$。那么，一个在t时刻，位于(x,y)处的像素，它的灰度可以写成

$$\boldsymbol{I}(x,y,t).$$

这种方式把图像看成了关于位置与时间的函数，它的值域就是图像中像素的灰度。现在考虑某个固定的空间点，它在t时刻的像素坐标

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

为x,y。由于相机的运动，它的图像坐标将发生变化。我们希望估计这个空间点在其他时刻图像中位置。怎么估计呢？这里要引入光流法的基本假设。

灰度不变假设：同一个空间点的像素灰度值，在各个图像中是固定不变的。

对于t时刻位于(x,y)处的像素，我们设t+dt时刻它运动到(x+dx,y+dy)处。由于灰度不变，我们有：

$$\boldsymbol{I}(x+\mathrm{d}x, y+\mathrm{d}y, t+\mathrm{d}t) = \boldsymbol{I}(x, y, t). \tag{8.1}$$

灰度不变假设是一个很强的假设，实际当中很可能不成立。事实上，由于物体的材质不同，像素会出现高光和阴影部分；有时，相机会自动调整曝光参数，使得图像整体变亮或变暗。这些时候灰度不变假设都是不成立的，因此光流的结果也不一定可靠。然而，从另一方面来说，所有算法都是在一定假设下工作的。如果我们什么假设都不做，就没法设计实用的算法。所以，让我们暂且认为该假设成立，看看如何计算像素的运动。

对左边进行泰勒展开，保留一阶项，得：

$$\boldsymbol{I}(x+\mathrm{d}x, y+\mathrm{d}y, t+\mathrm{d}t) \approx \boldsymbol{I}(x, y, t) + \frac{\partial \boldsymbol{I}}{\partial x}\mathrm{d}x + \frac{\partial \boldsymbol{I}}{\partial y}\mathrm{d}y + \frac{\partial \boldsymbol{I}}{\partial t}\mathrm{d}t. \tag{8.2}$$

因为我们假设了灰度不变，于是下一个时刻的灰度等于之前的灰度，从而：

$$\frac{\partial \boldsymbol{I}}{\partial x}\mathrm{d}x + \frac{\partial \boldsymbol{I}}{\partial y}\mathrm{d}y + \frac{\partial \boldsymbol{I}}{\partial t}\mathrm{d}t = 0. \tag{8.3}$$

两边除以dt，得：

$$\frac{\partial \boldsymbol{I}}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial \boldsymbol{I}}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t} = -\frac{\partial \boldsymbol{I}}{\partial t}. \tag{8.4}$$

其中dx/dt为像素在x轴上的运动速度，而dy/dt为y轴上的速度，把它

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

们记为u,v。同时$\partial I/\partial x$为图像在该点处x方向的梯度，另一项则是在y方向的梯度，记为I_x,I_y。把图像灰度对时间的变化量记为I_t，写成矩阵形式，有：

$$\begin{bmatrix} \boldsymbol{I}_x & \boldsymbol{I}_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\boldsymbol{I}_t. \tag{8.5}$$

我们想计算的是像素的运动u,v，但是该式是带有两个变量的一次方程，仅凭它无法计算出u,v。因此，必须引入额外的约束来计算u,v。在LK光流中，我们假设某一个窗口内的像素具有相同的运动。

考虑一个大小为$w\times w$的窗口，它含有w^2数量的像素。由于该窗口内像素具有同样的运动，因此我们共有w^2个方程：

$$\begin{bmatrix} \boldsymbol{I}_x & \boldsymbol{I}_y \end{bmatrix}_k \begin{bmatrix} u \\ v \end{bmatrix} = -\boldsymbol{I}_{tk}, \quad k = 1, \ldots, w^2. \tag{8.6}$$

记：

$$\boldsymbol{A} = \begin{bmatrix} [\boldsymbol{I}_x, \boldsymbol{I}_y]_1 \\ \vdots \\ [\boldsymbol{I}_x, \boldsymbol{I}_y]_k \end{bmatrix}, \boldsymbol{b} = \begin{bmatrix} \boldsymbol{I}_{t1} \\ \vdots \\ \boldsymbol{I}_{tk} \end{bmatrix}. \tag{8.7}$$

于是整个方程为

$$\boldsymbol{A} \begin{bmatrix} u \\ v \end{bmatrix} = -\boldsymbol{b}. \tag{8.8}$$

这是一个关于u,v的超定线性方程，传统解法是求最小二乘解。最小

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

二乘在很多时候都用到过：

$$\begin{bmatrix} u \\ v \end{bmatrix}^* = -\left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\right)^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}. \tag{8.9}$$

这样就得到了像素在图像间的运动速度u,v。当t取离散的时刻而不是连续时间时，我们可以估计某块像素在若干个图像中出现的位置。由于像素梯度仅在局部有效，所以如果一次迭代不够好，我们会多迭代几次这个方程。在SLAM中，LK光流常被用来跟踪角点的运动，我们不妨通过程序体会一下。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

8.3 实践：LK光流

8.3.1 使用TUM公开数据集

下面演示如何用OpenCV提供的光流法来跟踪特征点。与上一节一样，我们准备了若干幅数据集图像，存放在程序目录中的data/文件夹下。它们来自于慕尼黑工业大学（TUM）提供的公开RGB-D数据集[1]。以后我们就称之为TUM数据集。它含有许多个RGB-D视频，可以作为RGB-D或单目SLAM的实验数据。它还提供了用运动捕捉系统测量的精确轨迹，可以作为标准轨迹以校准SLAM系统。由于该数据集比较大，我们没有放到GitHub上（否则下载代码需要很长时间），请读者自行去数据集主页查找对应的数据。本程序中使用了一部分“freburg1_desk”数据集中的图像。读者可以在TUM数据集主页找到它的下载链接。或者，也可以直接使用本书在GitHub上提供的部分。

我们的数据位于本章目录的data/下，以压缩包形式提供（data.tar.gz）。由于TUM数据集是从实际环境中采集的，需要解释一下它的数据格式（数据集一般都有自己定义的格式）。在解压后，你将看到以下这些文件：

1.rgb.txt和depth.txt记录了各文件的采集时间和对应的文件名。

2.rgb/和depth/目录存放着采集到的PNG格式图像文件。彩色图像为8位3通道，深度图为16位单通道图像。文件名即采集时间。

3.groundtruth.txt为外部运动捕捉系统采集到的相机位姿，格式为

$$(\text{time}, t_x, t_y, t_z, q_x, q_y, q_z, q_w),$$

我们可以把它看成标准轨迹。

请注意，彩色图、深度图和标准轨迹的采集都是独立的，轨迹的采集频率比图像高很多。在使用数据之前，需要根据采集时间对数据进行一次时间上的对齐，以便对彩色图和深度图进行配对。原则上，我们可以把采集时间相近于一个阈值的数据，看成是一对图像。并把相近时间

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

的位姿，看作是该图像的真实采集位置。TUM提供了一个Python脚本“associate.py”（或使用slambook/tools/associate.py）帮我们完成这件事。请把此文件放到数据集目录下，运行：

```
python associate.py rgb.txt depth.txt > associate.txt
```

这段脚本会根据输入的两个文件中的采集时间进行配对，最后输出到文件associate.txt。输出文件含有配对后的两幅图像的时间、文件名信息，可以作为后续处理的来源。此外，TUM数据集还提供了比较估计轨迹与标准轨迹的工具，我们将在用到的地方再进行介绍。

8.3.2 使用LK光流

下面来编写程序使用OpenCV中的LK光流。使用LK的目的是跟踪特征点。我们对第一幅图像提取FAST角点，然后用LK光流跟踪它们，并画在图中。

slambook/ch8/useLK/useLK.cpp

```
#include <iostream>
#include <fstream>
#include <list>
#include <vector>
#include <chrono>
using namespace std;

```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/features2d/features2d.hpp>
#include <opencv2/video/tracking.hpp>

int main( int argc, char** argv )
{
    if ( argc != 2 )
    {
        cout<<"usage: useLK path_to_dataset"<<endl;
        return 1;
    }
    string path_to_dataset = argv[1];
    string associate_file = path_to_dataset + "/associate.txt";
    ifstream fin( associate_file );
    string rgb_file, depth_file, time_rgb, time_depth;
    list< cv::Point2f > keypoints;      // 因为要删除跟踪失败的点，使用list
    cv::Mat color, depth, last_color;
    for ( int index=0; index<100; index++ )
    {
        fin>>time_rgb>>rgb_file>>time_depth>>depth_file;
        color = cv::imread( path_to_dataset+"/"+rgb_file );
        depth = cv::imread( path_to_dataset+"/"+depth_file, -1 );
        if (index ==0 )
        {
            // 对第一帧提取 FAST 特征点
            vector<cv::KeyPoint> kps;
            cv::Ptr<cv::FastFeatureDetector> detector = cv::FastFeatureDetector::
            create();
            detector->detect( color, kps );
            for ( auto kp:kps )
                keypoints.push_back( kp.pt );
            last_color = color;
            continue;
        }
        if ( color.data==nullptr || depth.data==nullptr )
            continue;
        // 对其他帧用 LK 跟踪特征点
        vector<cv::Point2f> next_keypoints;
        vector<cv::Point2f> prev_keypoints;
        for ( auto kp:keypoints )
            prev_keypoints.push_back(kp);
        vector<unsigned char> status;
        vector<float> error;
        chrono::steady_clock::time_point t1 = chrono::steady_clock::now();
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
        cv::calcOpticalFlowPyrLK( last_color, color, prev_keypoints, next_keypoints
        , status, error );
        chrono::steady_clock::time_point t2 = chrono::steady_clock::now();
        chrono::duration<double> time_used = chrono::duration_cast<chrono::duration
        <double>>( t2-t1 );
        cout<<"LK Flow use time: "<<time_used.count()<<" seconds."<<endl;
        // 把跟丢的点删掉
        int i=0;
        for ( auto iter=keypoints.begin(); iter!=keypoints.end(); i++)
        {
            if ( status[i] == 0 )
            {
                iter = keypoints.erase(iter);
                continue;
            }
            *iter = next_keypoints[i];
            iter++;
        }
        cout<<"tracked keypoints: "<<keypoints.size()<<endl;
        if (keypoints.size() == 0)
        {
            cout<<"all keypoints are lost."<<endl;
            break;
        }
        // 画出 keypoints
        cv::Mat img_show = color.clone();
        for ( auto kp:keypoints )
            cv::circle(img_show, kp, 10, cv::Scalar(0, 240, 0), 1);
        cv::imshow("corners", img_show);
        cv::waitKey(0);
        last_color = color;
    }
    return 0;
}
```

读者应当已经熟悉了OpenCV的使用方式，这里就不提供CMakeLists.txt的写法了。该程序的运行参数里需要指定数据集所在的目

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

录，例如：

```
./build/useLK ../data
```

我们会在每次循环后暂停程序，按任意键可以继续运行。你会看到图像中大部分特征点都能够顺利跟踪到，但也有特征点会丢失。丢失的特征点或是被移出了视野外，或是被其他物体挡住了。如果我们不提取新的特征点，那么光流的跟踪会越来越少：

```
% build/useLK ../data
LK Flow use time: 0.0329535 seconds.
tracked keypoints: 1749
LK Flow use time: 0.0247758 seconds.
tracked keypoints: 1742
LK Flow use time: 0.0226143 seconds.
tracked keypoints: 1703
LK Flow use time: 0.0238692 seconds.
tracked keypoints: 1676
LK Flow use time: 0.0210466 seconds.
tracked keypoints: 1664
LK Flow use time: 0.0226533 seconds.
tracked keypoints: 1656
LK Flow use time: 0.0266527 seconds.
tracked keypoints: 1641
LK Flow use time: 0.0214207 seconds.
tracked keypoints: 1634
```

图8-2显示了程序运行过程中若干帧的情况（这里使用了完整的数据集，但本书的git上只给出了10张图）。最初我们大约有1700个特征点。跟踪过程中一部分特征点会丢失，直到100帧时我们还有约178个特征点，相机视角相对于最初的图像也发生了较大改变。

书籍汇总下载 http://t.cn/RDIAj5D

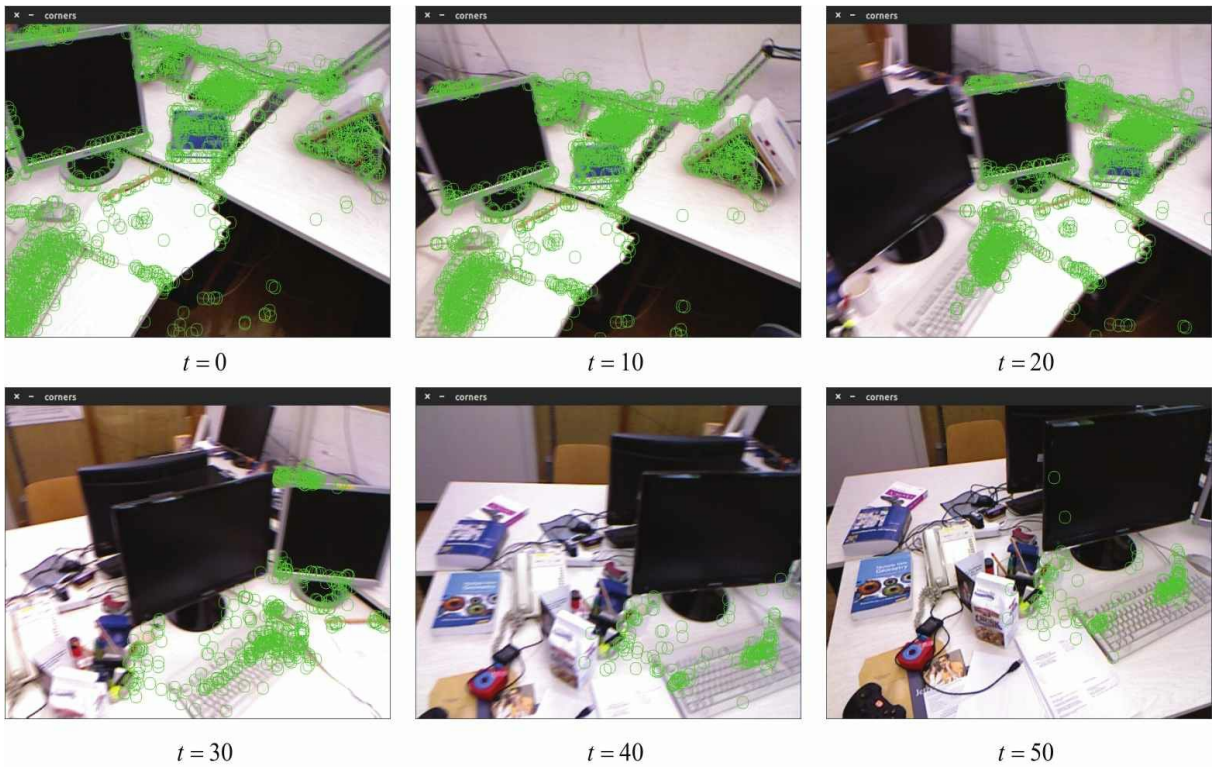


图8-2　LK光流法实验。

仔细观察特征点的跟踪过程，我们会发现位于物体角点处的特征更加稳定。边缘处的特征会沿着边缘“滑动”，这主要是由于沿着边缘移动时特征块的内容基本不变，因此程序容易认为是同一个地方。而既不在角点也不在边缘的特征点则会频繁跳动，位置非常不稳定。这个现象很像围棋中的“金角银边草肚皮”：角点具有更好的辨识度，边缘次之，区块最少。

另一方面，读者可以看到光流法的运行时间。在跟踪1500个特征点时，LK光流法大约需要20ms。如果减小特征点的数量，则会明显减少计算时间。我们看到，LK光流跟踪法避免了描述子的计算与匹配，但本身也需要一定的计算量。在我们的计算平台上，使用LK光流能够节省一定的计算量，但在具体SLAM系统中，使用光流还是匹配描述子，最好是亲自做实验测试一下。

另外，LK光流跟踪能够直接得到特征点的对应关系。这个对应关系就像是描述子的匹配，但实际上我们大多数时候只会碰到特征点跟丢的情况，而不太会遇到误匹配，这应该是光流相对于描述子的一点优势。但是，匹配描述子的方法在相机运动较大时仍能成功，而光流必须

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

要求相机运动是微小的。从这方面来说，光流的健壮性比描述子差一些。

最后，我们可以通过光流跟踪的特征点，用PnP、ICP或对极几何来估计相机运动，这些方法在上一讲中介绍过，这里不再讨论。总而言之，光流法可以加速基于特征点的视觉里程计算法，避免计算和匹配描述子的过程，但要求相机运动较慢（或采集频率较高）。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

8.4 直接法（DirectMethod）

接下来，我们来讨论与光流有一定相似性的直接法。与前面内容相似，我们先介绍直接法的原理，然后使用g2o实现直接法。

8.4.1 直接法的推导

如图8-3所示，考虑某个空间点P和两个时刻的相机。P的世界坐标为$[X,Y,Z]$，它在两个相机上成像，记非齐次像素坐标为p_1,p_2。

我们的目标是求第一个相机到第二个相机的相对位姿变换。我们以第一个相机为参照系，设第二个相机的旋转和平移为R,t（对应李代数为ξ）。同时，两相机的内参相同，记为K。为清楚起见，我们列写完整的投影方程：

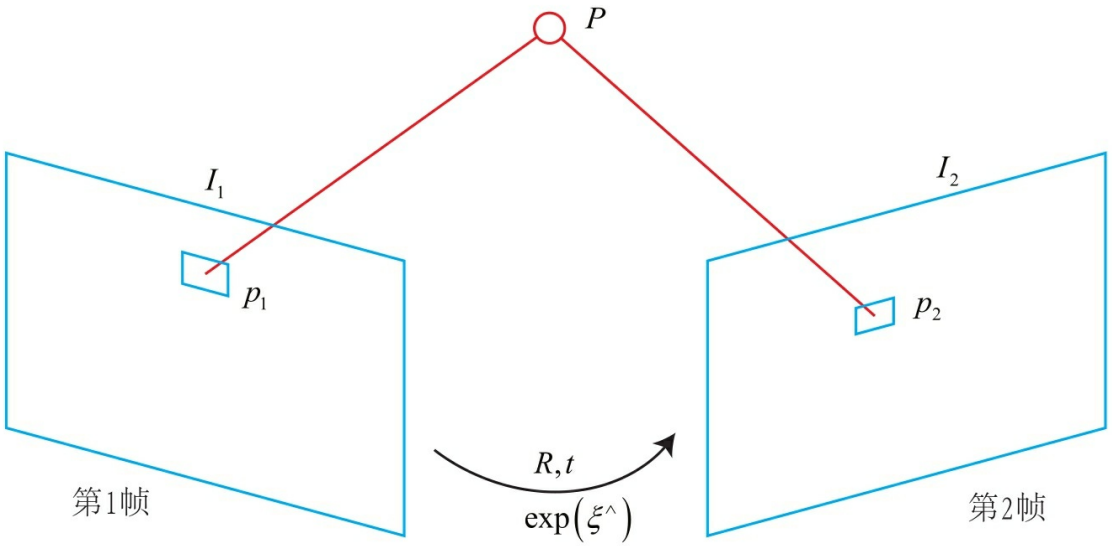


图8-3　直接法示意图。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$\boldsymbol{p}_1 = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_1 = \frac{1}{Z_1}\boldsymbol{K}\boldsymbol{P},$$

$$\boldsymbol{p}_2 = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_2 = \frac{1}{Z_2}\boldsymbol{K}\left(\boldsymbol{R}\boldsymbol{P}+\boldsymbol{t}\right) = \frac{1}{Z_2}\boldsymbol{K}\left(\exp\left(\boldsymbol{\xi}^{\wedge}\right)\boldsymbol{P}\right)_{1:3}.$$

其中Z_1是P的深度，Z_2是P在第二个相机坐标系下的深度，也就是$RP+t$的第3个坐标值。由于$\exp(\xi^{\wedge})$只能和齐次坐标相乘，所以我们乘完之后要取出前3个元素。这和上一讲及相机模型部分的内容是一致的。

回忆特征点法中，由于我们通过匹配描述子知道了p_1,p_2的像素位置，所以可以计算重投影的位置。但在直接法中，由于没有特征匹配，我们无从知道哪一个p_2与p_1对应着同一个点。直接法的思路是根据当前相机的位姿估计值来寻找p_2的位置。但若相机位姿不够好，p_2的外观和p_1会有明显差别。于是，为了减小这个差别，我们优化相机的位姿，来寻找与p_1更相似的p_2。这同样可以通过解一个优化问题完成，但此时最小化的不是重投影误差，而是光度误差（Photometric Error），也就是P的两个像素的亮度误差：

$$e = \boldsymbol{I}_1\left(\boldsymbol{p}_1\right) - \boldsymbol{I}_2\left(\boldsymbol{p}_2\right). \tag{8.10}$$

注意这里e是一个标量。同样地，优化目标为该误差的二范数，暂时取不加权的形式，为

$$\min_{\boldsymbol{\xi}} J\left(\boldsymbol{\xi}\right) = \|e\|^2. \tag{8.11}$$

能够做这种优化的理由，仍是基于灰度不变假设。在直接法中，我们假设一个空间点在各个视角下成像的灰度是不变的。我们有许多个

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

（比如N个）空间点P_i，那么，整个相机位姿估计问题变为

$$\min_{\boldsymbol{\xi}} J(\boldsymbol{\xi}) = \sum_{i=1}^{N} e_i^{\mathrm{T}} e_i, \quad e_i = \boldsymbol{I}_1(\boldsymbol{p}_{1,i}) - \boldsymbol{I}_2(\boldsymbol{p}_{2,i}). \tag{8.12}$$

注意这里的优化变量是相机位姿ξ。为了求解这个优化问题，我们关心误差e是如何随着相机位姿ξ变化的，需要分析它们的导数关系。因此，使用李代数上的扰动模型。我们给$\exp(\xi)$左乘一个小扰动$\exp(\delta\xi)$，得：[2]

$$\begin{aligned} e(\boldsymbol{\xi} \oplus \delta\boldsymbol{\xi}) &= \boldsymbol{I}_1\left(\frac{1}{Z_1}\boldsymbol{K}\boldsymbol{P}\right) - \boldsymbol{I}_2\left(\frac{1}{Z_2}\boldsymbol{K}\exp(\delta\boldsymbol{\xi}^\wedge)\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{P}\right) \\ &\approx \boldsymbol{I}_1\left(\frac{1}{Z_1}\boldsymbol{K}\boldsymbol{P}\right) - \boldsymbol{I}_2\left(\frac{1}{Z_2}\boldsymbol{K}(1+\delta\boldsymbol{\xi}^\wedge)\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{P}\right) \\ &= \boldsymbol{I}_1\left(\frac{1}{Z_1}\boldsymbol{K}\boldsymbol{P}\right) - \boldsymbol{I}_2\left(\frac{1}{Z_2}\boldsymbol{K}\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{P} + \frac{1}{Z_2}\boldsymbol{K}\delta\boldsymbol{\xi}^\wedge\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{P}\right). \end{aligned}$$

类似于上一讲，记：

$$\begin{aligned} \boldsymbol{q} &= \delta\boldsymbol{\xi}^\wedge \exp(\boldsymbol{\xi}^\wedge)\boldsymbol{P}, \\ \boldsymbol{u} &= \frac{1}{Z_2}\boldsymbol{K}\boldsymbol{q}. \end{aligned}$$

这里的q为扰动分量在第二个相机坐标系下的坐标，而u为它的像素坐标。利用一阶泰勒展开，有：

$$\begin{aligned} e(\boldsymbol{\xi} \oplus \delta\boldsymbol{\xi}) &= \boldsymbol{I}_1\left(\frac{1}{Z_1}\boldsymbol{K}\boldsymbol{P}\right) - \boldsymbol{I}_2\left(\frac{1}{Z_2}\boldsymbol{K}\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{P} + \boldsymbol{u}\right) \\ &\approx \boldsymbol{I}_1\left(\frac{1}{Z_1}\boldsymbol{K}\boldsymbol{P}\right) - \boldsymbol{I}_2\left(\frac{1}{Z_2}\boldsymbol{K}\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{P}\right) - \frac{\partial \boldsymbol{I}_2}{\partial \boldsymbol{u}}\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{q}}\frac{\partial \boldsymbol{q}}{\partial \delta\boldsymbol{\xi}}\delta\boldsymbol{\xi} \\ &= e(\boldsymbol{\xi}) - \frac{\partial \boldsymbol{I}_2}{\partial \boldsymbol{u}}\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{q}}\frac{\partial \boldsymbol{q}}{\partial \delta\boldsymbol{\xi}}\delta\boldsymbol{\xi}. \end{aligned}$$

我们看到，一阶导数由于链式法则分成了3项，而这3项都是容易计算的：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

1.$\partial I_2/\partial u$为u处的像素梯度。

2.$\partial u/\partial q$为投影方程关于相机坐标系下的三维点的导数。记$q=[X,Y,Z]^{\mathrm{T}}$，根据上一节的推导，导数为

$$\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{q}} = \begin{bmatrix} \frac{\partial u}{\partial X} & \frac{\partial u}{\partial Y} & \frac{\partial u}{\partial Z} \\ \frac{\partial v}{\partial X} & \frac{\partial v}{\partial Y} & \frac{\partial v}{\partial Z} \end{bmatrix} = \begin{bmatrix} \frac{f_x}{Z} & 0 & -\frac{f_x X}{Z^2} \\ 0 & \frac{f_y}{Z} & -\frac{f_y Y}{Z^2} \end{bmatrix}. \tag{8.13}$$

3.$\partial q/\partial \delta\xi$为变换后的三维点对变换的导数，这在李代数一讲介绍过了：

$$\frac{\partial \boldsymbol{q}}{\partial \delta \boldsymbol{\xi}} = [\boldsymbol{I}, -\boldsymbol{q}^{\wedge}]. \tag{8.14}$$

在实践中，由于后两项只与三维点q有关，而与图像无关，我们经常把它合并在一起：

$$\frac{\partial \boldsymbol{u}}{\partial \delta \boldsymbol{\xi}} = \begin{bmatrix} \frac{f_x}{Z} & 0 & -\frac{f_x X}{Z^2} & -\frac{f_x XY}{Z^2} & f_x + \frac{f_x X^2}{Z^2} & -\frac{f_x Y}{Z} \\ 0 & \frac{f_y}{Z} & -\frac{f_y Y}{Z^2} & -f_y - \frac{f_y Y^2}{Z^2} & \frac{f_y XY}{Z^2} & \frac{f_y X}{Z} \end{bmatrix}. \tag{8.15}$$

这个2×6的矩阵在上一讲中也出现过。于是，我们推导出误差相对于李代数的雅可比矩阵：

$$\boldsymbol{J} = -\frac{\partial \boldsymbol{I}_2}{\partial \boldsymbol{u}} \frac{\partial \boldsymbol{u}}{\partial \delta \boldsymbol{\xi}}. \tag{8.16}$$

对于N个点的问题，我们可以用这种方法计算优化问题的雅可比矩阵，然后使用高斯牛顿法或列文伯格—马夸尔特方法计算增量，迭代求解。至此，我们推导了直接法估计相机位姿的整个流程，下面通过程序来演示一下直接法是如何使用的。

8.4.2 直接法的讨论

在上面的推导中，P是一个已知位置的空间点，它是怎么来的呢？

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

在RGB-D相机下，我们可以把任意像素反投影到三维空间，然后投影到下一幅图像中。如果在单目相机中，这件事情要更为困难，因为我们还须考虑由P的深度带来的不确定性。详细的深度估计放到第13讲中讨论。现在我们先来考虑简单的情况，即P深度已知的情况。

根据P的来源，我们可以把直接法进行分类：

1.P来自于稀疏关键点，我们称之为稀疏直接法。通常我们使用数百个至上千个关键点，并且像L-K光流那样，假设它周围像素也是不变的。这种稀疏直接法不必计算描述子，并且只使用数百个像素，因此速度最快，但只能计算稀疏的重构。

2.P来自部分像素。我们看到式(8.16)中，如果像素梯度为零，整项雅可比矩阵就为零，不会对计算运动增量有任何贡献。因此，可以考虑只使用带有梯度的像素点，舍弃像素梯度不明显的地方。这称为半稠密（Semi-Dense）的直接法，可以重构一个半稠密结构。

3.P为所有像素，称为稠密直接法。稠密重构需要计算所有像素（一般几十万至几百万个），因此多数不能在现有的CPU上实时计算，需要GPU的加速。但是，如前面所讨论的，像素梯度不明显的点，在运动估计中不会有太大贡献，在重构时也会难以估计位置。

可以看到，从稀疏到稠密重构，都可以用直接法来计算。它们的计算量是逐渐增长的。稀疏方法可以快速地求解相机位姿，而稠密方法可以建立完整地图。具体使用哪种方法，需要视机器人的应用环境而定。特别地，在低端的计算平台上，稀疏直接法可以做到非常快速的效果，适用于实时性较高且计算资源有限的场合[58]。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

8.5 实践：RGB-D的直接法

8.5.1 稀疏直接法

现在，我们来演示如何使用稀疏的直接法。由于本书不涉及GPU编程，稠密的直接法就省略掉了。同时，为了保持程序简单，我们使用RGB-D数据而非单目数据，这样可以省略掉单目的深度恢复部分。基于特征点的深度恢复已经在上一讲介绍过，而基于块匹配的深度恢复将在后面介绍。所以本节我们来考虑RGB-D上的稀疏直接法VO。

由于求解直接法最后等价于求解一个优化问题，因此可以使用g2o或Ceres这些优化库来帮助求解。本节以g2o为例设计实验，而Ceres部分则留作习题。在使用g2o之前，需要把直接法抽象成一个图优化问题。显然，直接法是由以下顶点和边组成的：

1.优化变量为一个相机位姿，因此需要一个位姿顶点。由于我们在推导中使用了李代数，故程序中使用李代数表达的SE(3)位姿顶点。与上一讲一样，我们将使用“VertexSE3Expmap”作为相机位姿。

2.误差项为单个像素的光度误差。由于整个优化过程中$I_1(p_1)$保持不变，我们可以把它当成一个固定的预设值，然后调整相机位姿，使$I_2(p_2)$接近这个值。于是，这种边只连接一个顶点，为一元边。由于g2o中本身没有计算光度误差的边，我们需要自己定义一种新的边。

在上述的建模中，直接法图优化问题是由一个相机位姿顶点与许多条一元边组成的。如果使用稀疏的直接法，那我们大约会有几百至几千条这样的边；稠密直接法则会有几十万条边。优化问题对应的线性方程是计算李代数增量，本身规模不大（6×6），所以主要的计算时间会花费在每条边的误差与雅可比矩阵的计算上。下面的实验中，我们先来定义一种用于直接法位姿估计的边，然后，使用该边构建图优化问题并求解。

8.5.2 定义直接法的边

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

首先定义计算光度误差的边。按照前面的推导，还需要给出它的雅可比矩阵：

slambook/ch8/directMethod/direct_sparse.cpp（片段）

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
// project a 3d point into an image plane, the error is photometric error
// an unary edge with one vertex SE3Expmap (the pose of camera)
class EdgeSE3ProjectDirect: public BaseUnaryEdge< 1, double, VertexSE3Expmap>
{
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW

    EdgeSE3ProjectDirect() {}

    EdgeSE3ProjectDirect ( Eigen::Vector3d point, float fx, float fy, float cx,
    float cy, cv::Mat* image ) : x_world_ ( point ), fx_ ( fx ), fy_ ( fy ), cx_ (
    cx ), cy_ ( cy ), image_ ( image )
    {}

    virtual void computeError()
    {
        const VertexSE3Expmap* v  =static_cast<const VertexSE3Expmap*> ( _vertices
        [0] );
        Eigen::Vector3d x_local = v->estimate().map ( x_world_ );
        float x = x_local[0]*fx_/x_local[2] + cx_;
        float y = x_local[1]*fy_/x_local[2] + cy_;
        // check x,y is in the image
        if ( x-4<0 || ( x+4 ) >image_->cols || ( y-4 ) <0 || ( y+4 ) >image_->rows
        )
        {
            _error ( 0,0 ) = 0.0;
            this->setLevel ( 1 );
        }
        else
        {
            _error ( 0,0 ) = getPixelValue ( x,y ) - _measurement;
        }
    }

    // plus in manifold
    virtual void linearizeOplus( )
    {
         if ( level() == 1 )
         {
             _jacobianOplusXi = Eigen::Matrix<double, 1, 6>::Zero();
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
            return;
        }
        VertexSE3Expmap* vtx = static_cast<VertexSE3Expmap*> ( _vertices[0] );
        Eigen::Vector3d xyz_trans = vtx->estimate().map ( x_world_ );   // q in
        book

        double x = xyz_trans[0];
        double y = xyz_trans[1];
        double invz = 1.0/xyz_trans[2];
        double invz_2 = invz*invz;

        float u = x*fx_*invz + cx_;
        float v = y*fy_*invz + cy_;

        // jacobian from se3 to u,v
        // NOTE that in g2o the Lie algebra is (\omega, \epsilon), where \omega is
        so(3) and \epsilon the translation
        Eigen::Matrix<double, 2, 6> jacobian_uv_ksai;

        jacobian_uv_ksai ( 0,0 ) = - x*y*invz_2 *fx_;
        jacobian_uv_ksai ( 0,1 ) = ( 1+ ( x*x*invz_2 ) ) *fx_;
        jacobian_uv_ksai ( 0,2 ) = - y*invz *fx_;
        jacobian_uv_ksai ( 0,3 ) = invz *fx_;
        jacobian_uv_ksai ( 0,4 ) = 0;
        jacobian_uv_ksai ( 0,5 ) = -x*invz_2 *fx_;

        jacobian_uv_ksai ( 1,0 ) = - ( 1+y*y*invz_2 ) *fy_;
        jacobian_uv_ksai ( 1,1 ) = x*y*invz_2 *fy_;
        jacobian_uv_ksai ( 1,2 ) = x*invz *fy_;
        jacobian_uv_ksai ( 1,3 ) = 0;
        jacobian_uv_ksai ( 1,4 ) = invz *fy_;
        jacobian_uv_ksai ( 1,5 ) = -y*invz_2 *fy_;

        Eigen::Matrix<double, 1, 2> jacobian_pixel_uv;

        jacobian_pixel_uv ( 0,0 ) = ( getPixelValue ( u+1,v )-getPixelValue ( u-1,v
        ) ) /2;
        jacobian_pixel_uv ( 0,1 ) = ( getPixelValue ( u,v+1 )-getPixelValue ( u,v-1
        ) ) /2;

        _jacobianOplusXi = jacobian_pixel_uv*jacobian_uv_ksai;
    }

    // dummy read and write functions because we don't care...
    virtual bool read ( std::istream& in ) {}
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
78      virtual bool write ( std::ostream& out ) const {}
79
80  protected:
81      // get a gray scale value from reference image (bilinear interpolated)
82      inline float getPixelValue ( float x, float y )
83      {
84          uchar* data = & image_->data[ int ( y ) * image_->step + int ( x ) ];
85          float xx = x - floor ( x );
86          float yy = y - floor ( y );
87          return float (
88              ( 1-xx ) * ( 1-yy ) * data[0] +
89              xx* ( 1-yy ) * data[1] +
90              ( 1-xx ) *yy*data[ image_->step ] +
91              xx*yy*data[image_->step+1]
92          );
93      }
94  public:
95      Eigen::Vector3d x_world_;   // 3D point in world frame
96      float cx_=0, cy_=0, fx_=0, fy_=0; // Camera intrinsics
97      cv::Mat* image_=nullptr;    // reference image
98  };
```

我们的边继承自g2o::BaseUnaryEdge。在继承时，需要在模板参数里填入测量值的维度、类型，以及连接此边的顶点，同时，我们把空间点P、相机内参和图像存储在该边的成员变量中。为了让g2o优化该边对应的误差，我们需要覆写两个虚函数：用computeError()计算误差值，用linearizeOplus()计算雅可比矩阵。可以看到，这里的雅可比矩阵计算与式(8.16)是一致的。注意我们在程序中的误差计算里使用了$I_2(p_2)$-$I_1(p_1)$的形式，因此前面的负号可以省去，只需把像素梯度乘以像素到李代数的梯度即可。

在程序中，相机位姿是用浮点数表示的，投影到像素坐标也是浮点形式。为了更精细地计算像素亮度，我们要对图像进行插值。我们这里采用了简单的双线性插值，也可以使用更复杂的插值方式，但计算代价可能会变高一些。

书籍汇总下载 http://t.cn/RDIAj5D

8.5.3 使用直接法估计相机运动

定义了g2o边后，我们将节点和边组合成图，就可以调用g2o进行优化了。实现代码位于slambook/ch8/directMethod/direct_sparse.cpp中，请读者阅读该部分代码并编译。

在这个实验中，我们读取数据集的RGB-D图像序列。以第一幅图像为参考帧，然后用直接法求解后续图像的位姿。在参考帧中，对第一幅图像提取FAST关键点（不需要描述子），并使用直接法估计这些关键点在第二幅图像中的位置，以及第二幅图像的相机位姿。这就构成了一种简单的稀疏直接法。最后，我们画出这些关键点在第二幅图像中的投影。

执行如下命令：

```
build/direct_sparse ~/dataset/rgbd_dataset_freiburg1_desk
```

程序会在作图之后暂停，你可以看到特征点的位置关系，终端也会输出迭代误差的下降过程。

如图8-4所示，我们看到在两幅图像相差不多时，直接法会调整相机的位姿，使得大部分像素都能够正确跟踪。但是，在稍长一点的时间内，比如说0～9帧之间的对比，我们发现由于相机位姿估计不准确，特征点出现了明显的偏移现象。我们会在本讲末尾对其进行分析。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

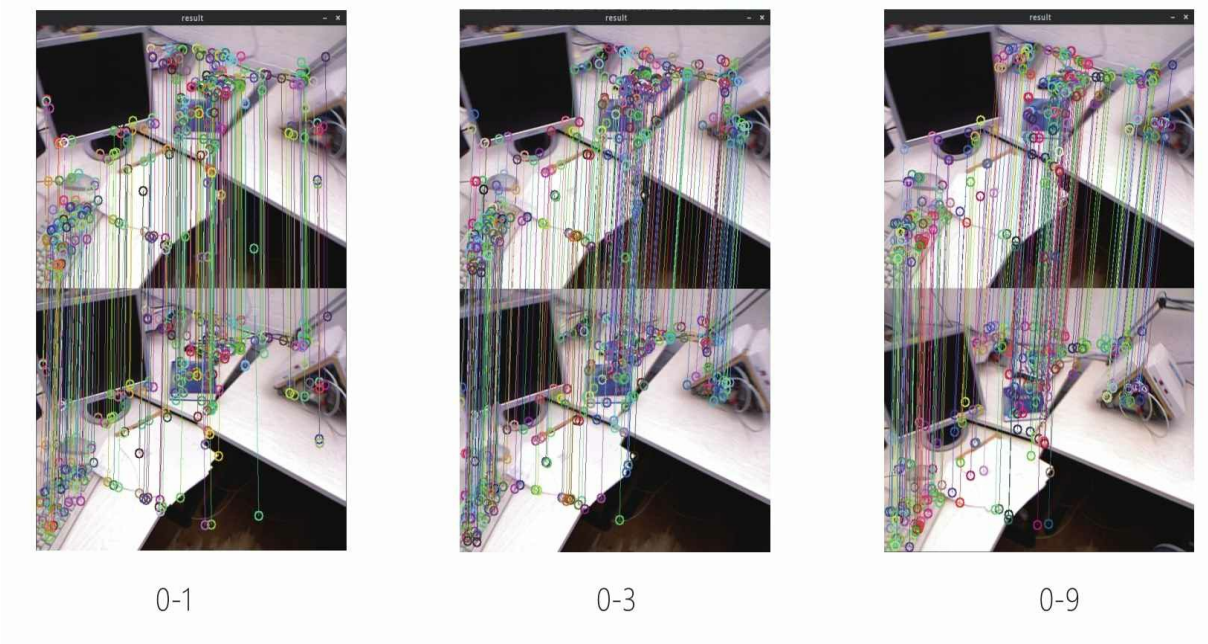


图8-4　稀疏直接法的实验。左：误差随着迭代下降。右：参考帧与后1～9帧对比（选取部分关键点）。

8.5.4　半稠密直接法

我们很容易就能把程序拓展成半稠密的直接法形式。对参考帧，先提取梯度较明显的像素，然后用直接法，以这些像素为图优化边来估计相机运动。对先前的程序做如下修改：

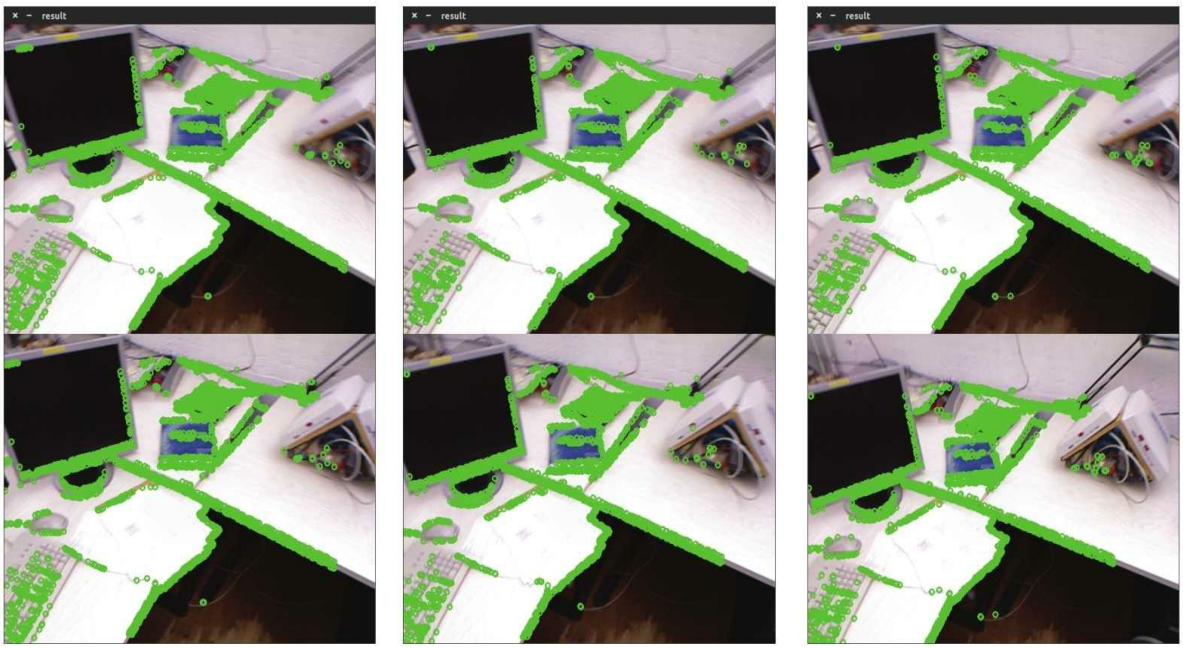
slambook/ch8/direct_semidense.cpp

```
// select the pixels with high gradiants
for ( int x=10; x<gray.cols-10; x++ )
    for ( int y=10; y<gray.rows-10; y++ )
    {
        Eigen::Vector2d delta (
            gray.ptr<uchar>(y)[x+1] - gray.ptr<uchar>(y)[x-1],
            gray.ptr<uchar>(y+1)[x] - gray.ptr<uchar>(y-1)[x]
        );
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

```
        if ( delta.norm() < 50 )
            continue;
        ushort d = depth.ptr<ushort> (y)[x];
        if ( d==0 )
            continue;
        Eigen::Vector3d p3d = project2Dto3D ( x, y, d, fx, fy, cx, cy, depth_scale
        );
        float grayscale = float ( gray.ptr<uchar> (y) [x] );
        measurements.push_back ( Measurement ( p3d, grayscale ) );
    }
```

这只是一个很简单的改动。我们把先前的稀疏特征点改成了带有明显梯度的像素。于是在图优化中会增加许多的边。这些边都会参与估计相机位姿的优化问题，利用大量的像素而不单单是稀疏的特征点。由于我们并没有使用所有的像素，所以这种方式又称为半稠密方法（Semi-dense）。我们把参与估计的像素取出来并在图像中显示出来，如图8-5所示。



$t_0 - t_2$　　$t_0 - t_5$　　$t_0 - t_8$

图8-5　半稠密直接法的实验。参考帧与第2,5,8帧的对比，绿色部分标出了参与优化的像素。

书籍汇总下载 http://t.cn/RDIAj5D

如果读者亲自做了实验，就可以看到参与估计的像素像是固定在空间中一样。当相机旋转时，它们的位置似乎没有发生变化。这代表了我们估计的相机运动是正确的。同时，你可以检查使用的像素数量与优化时间的关系。显然，当像素增多时，优化会更加费时，所以为了实时性，需要考虑使用较好的像素点，或者降低图像的分辨率。不过对于演示实验来说，笔者认为这样已经能够理解直接法的意义了。

8.5.5 直接法的讨论

相比于特征点法，直接法完全依靠优化来求解相机位姿。从式(8.16)中可以看到，像素梯度引导着优化的方向。如果想要得到正确的优化结果，就必须保证大部分像素梯度能够把优化引导到正确的方向。

这是什么意思呢？我们不妨设身处地地扮演一下优化算法。假设对于参考图像，我们测量到一个灰度值为229的像素。并且，由于我们知道它的深度，可以推断出空间点P的位置（图8-6所示在I_1中测量到的灰度）。

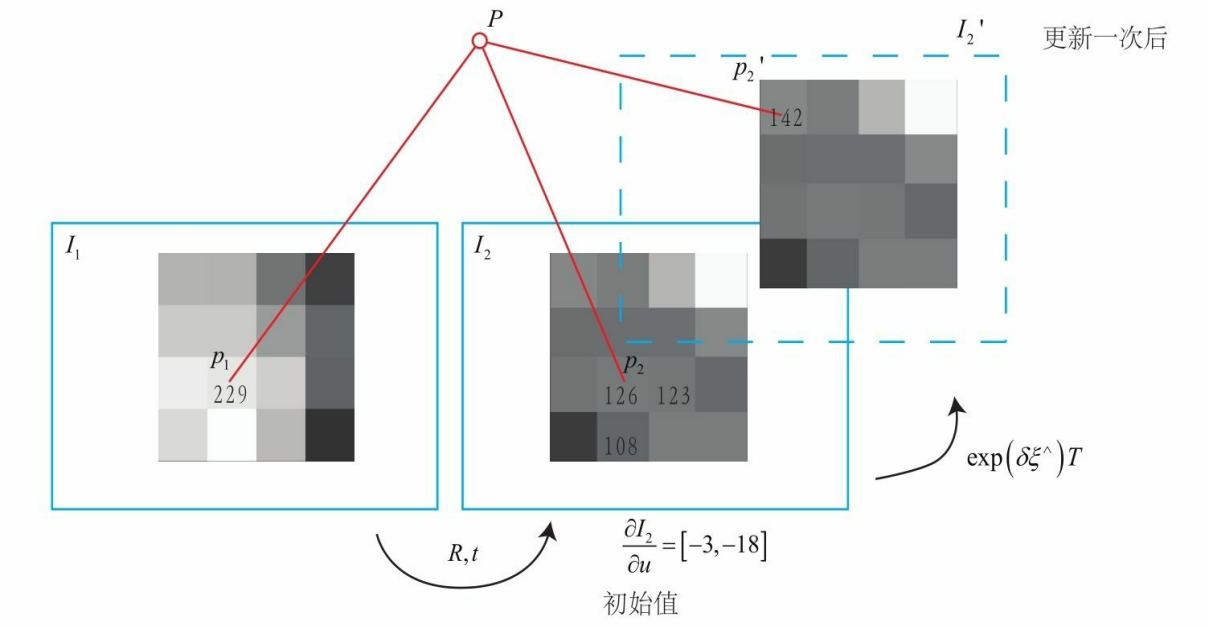


图8-6　一次迭代的图形化显示。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

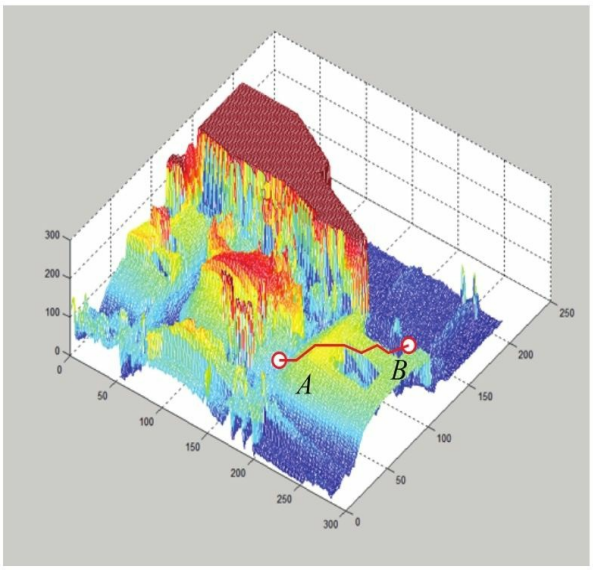
此时我们又得到了一幅新的图像，需要估计它的相机位姿。这个位姿是由一个初值不断地优化迭代得到的。假设我们的初值比较差，在这个初值下，空间点P投影后的像素灰度值是126。于是，这个像素的误差为229-126=103。为了减小这个误差，我们希望微调相机的位姿，使像素更亮一些。

怎么知道往哪里微调像素会更亮呢？这就需要用到局部的像素梯度。我们在图像中发现，沿u轴往前走一步，该处的灰度值变成了123，即减去了3。同样地，沿v轴往前走一步，灰度值减了18，变成108。在这个像素周围，我们看到梯度是[-3,-18]，为了提高亮度，我们会建议优化算法微调相机，使P的像往左上方移动。在这个过程中，我们用像素的局部梯度近似了它附近的灰度分布，不过请注意，真实图像并不是光滑的，所以这个梯度在远处就不成立了。

但是，优化算法不能只听这个像素的一面之词，还需要听取其他像素的建议[3]。综合听取了许多像素的意见之后，优化算法选择了一个和我们建议的方向偏离不远的地方，计算出一个更新量$\exp(\xi^\wedge)$。加上更新量后，图像从I_2移动到了I_2'，像素的投影位置也变到了一个更亮的地方。我们看到，通过这次更新，误差变小了。在理想情况下，我们期望误差会不断下降，最后收敛。

但是实际是不是这样呢？我们是否真的只要沿着梯度方向走，就能走到一个最优值？注意到，直接法的梯度是直接由图像梯度确定的，因此我们必须保证沿着图像梯度走时，灰度误差会不断下降。然而，图像通常是一个很强烈的非凸函数，如图8-7所示。实际当中，如果我们沿着图像梯度前进，很容易由于图像本身的非凸性（或噪声）落进一个局部极小值中，无法继续优化。只有当相机运动很小，图像中的梯度不会有很强的非凸性时，直接法才能成立。

书籍汇总下载 http://t.cn/RDIAj5D



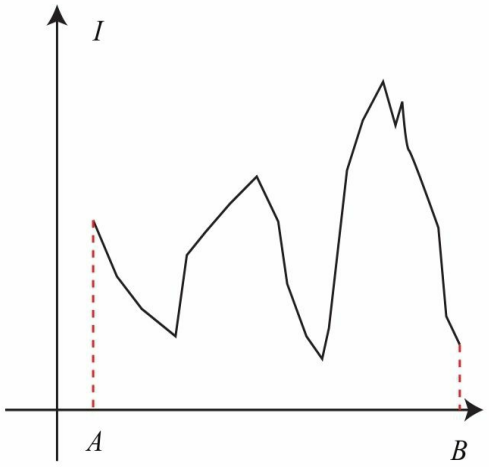


图8-7　一张图像的三维化显示。从图像中的一个点运动到另一个点的路径不见得是“笔直的下坡路”，而需要经常“翻山越岭”。这体现了图像本身的非凸性。

在例程中，我们只计算了单个像素的差异，并且这个差异是由灰度直接相减得到的。然而，单个像素没有什么区分性，周围很可能有好多像素和它的亮度差不多。所以，我们有时会使用小的图像块（patch），并且使用更复杂的差异度量方式，例如归一化相关性（Normalized Cross Correlation，NCC）等（见第13讲）。而例程为了简单起见，使用了误差的平方和，以保持与推导的一致性。

8.5.6　直接法优缺点总结

最后，我们总结一下直接法的优缺点。大体上，它的优点如下：

•可以省去计算特征点、描述子的时间。

•只要求有像素梯度即可，不需要特征点。因此，直接法可以在特征缺失的场合下使用。比较极端的例子是只有渐变的一幅图像。它可能无法提取角点类特征，但可以用直接法估计它的运动。

•可以构建半稠密乃至稠密的地图，这是特征点法无法做到的。

另一方面，它的缺点也很明显：

•非凸性。直接法完全依靠梯度搜索，降低目标函数来计算相机位

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

姿。其目标函数中需要取像素点的灰度值，而图像是强烈非凸的函数。这使得优化算法容易进入极小，只在运动很小时直接法才能成功。

•单个像素没有区分度。和它像的实在太多了！于是我们要么计算图像块，要么计算复杂的相关性。由于每个像素对改变相机运动的“意见”不一致，只能少数服从多数，以数量代替质量。

•灰度值不变是很强的假设。如果相机是自动曝光的，当它调整曝光参数时，会使得图像整体变亮或变暗。光照变化时亦会出现这种情况。特征点法对光照具有一定的容忍性，而直接法由于计算灰度间的差异，整体灰度变化会破坏灰度不变假设，使算法失败。针对这一点，目前的直接法开始使用更细致的光度模型标定相机，以便在曝光时间变化时也能工作。

习题

1.除了LK光流之外，还有哪些光流方法？它们各有什么特点？

2.在本节程序的求图像梯度过程中，我们简单地求了u+1和u-1的灰度之差除以2，作为u方向上的梯度值。这种做法有什么缺点？提示：对于距离较近的特征，变化应该较快；而距离较远的特征在图像中变化较慢，求梯度时能否利用此信息？

3.在稀疏直接法中，假设单个像素周围小块的光度也不变，是否可以提高算法健壮性？请编程实现。

4.*使用Ceres实现RGB-D上的稀疏直接法和半稠密直接法。

5.相比于RGB-D的直接法，单目直接法往往更加复杂。除了匹配未知之外，像素的距离也是待估计的，我们需要在优化时把像素深度也作为优化变量。阅读文献[59,57]，你能理解它的原理吗？如果不能，请在第13讲后再回来阅读。

6.由于图像的非凸性，直接法目前还只能用于短距离、非自动曝光的相机。你能否提出增强直接法健壮性的方案？阅读文献[58,60]可能会给你一些灵感。

---

[1]见http://vision.in.tum.de/data/datasets/rgbd-dataset/download。

[2]为了避免齐次/非齐次坐标转换而导致的公式形式复杂化，我们假设中间隐式地做

了所需的变化。它不会影响公式的推导。

[3]这可能是一种不严谨的拟人化说法，不过有助于理解。

书籍汇总下载 http://t.cn/RDIAj5D

第9讲　实践：设计前端

主要目标

1.实际设计一个视觉里程计前端。

2.理解SLAM软件框架是如何搭建的。

3.理解在前端设计中容易出现的问题，以及修补的方式。

本讲是全书比较少见的完全由实践部分组成的一讲。我们将使用前两讲学到的知识，实际书写一个视觉里程计程序。你会管理局部的机器人轨迹与路标点，并体验一下一个软件框架是如何组成的。在操作过程中，我们会遇到许多问题：相机运动过快、图像模糊、误匹配……都会使算法失效。要让程序稳定运行，我们需要处理以上的种种情况，这将带来许多工程实现方面的、有益的讨论。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

9.1 搭建VO框架

知晓砖头和水泥的原理，并不代表能够建造伟大的宫殿。

在笔者深爱的《我的世界》游戏中，玩家拥有的只是一些色彩、纹理不同的方块。其性质极其简单，而玩家所要做的只是把这些方块放在空地上而已。理解一个方块至为简单，但实际拿起它们时，初学者往往只能搭建简单的火柴盒房屋，而有经验、有创造力的玩家则可用这些简单的方块建造民居、园林、楼台亭榭，乃至城市（图9-1）[1]。



图9-1 从简单的事物出发，逐渐搭建越来越复杂但越来越优秀的作品。

在SLAM中，我们认为工程实现和理解算法原理应该至少是同等重要的，甚至更应强调如何书写实际可用的程序。算法的原理，就像一个个方块一样，我们可以清楚明确地讨论它们的原理和性质，但仅仅理解了一个个方块并不能使你建造真正的建筑：它们需要大量的尝试、时间和经验，我们鼓励读者朝更为实际的方向努力——当然这往往是十分复杂的。就像在《我的世界》里那样，你需要掌握各种立柱、墙面、屋顶的结构，墙面的雕花，几何形体角度的计算，这些远远不像讨论每个方块的性质那样简单。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

SLAM的具体实现亦是如此，一个实用的程序会有很多的工程设计和技巧（Trick），还需要讨论每一步出现问题之后该如何处理。原则上讲，每个人实现的SLAM都会有所不同，多数时候我们并不能说哪种实现方式就一定是最好的。但是，我们通常会遇到一些共同的问题：“怎么管理地图点”“如何处理误匹配”“如何选择关键帧”，等等。我们希望读者能对这些可能出现的问题产生一些直观的感觉——我们认为这种感觉是非常重要的。

所以，出于对实践的重视，本章我们将带领读者领略一下搭建SLAM框架的过程。就像建筑那样，我们要讨论柱间距、门面宽高比等琐碎但重要的问题。SLAM工程是复杂的。即使我们只保留核心的部分，也会占用大量的篇幅，使本书变得过于繁冗。不过，请注意，尽管完成之后的工程是复杂的，但是中间的“由简到繁”的过程，是值得详细讨论、有学习价值的。所以，我们要从简单的数据结构出发，先来做一个简单的视觉里程计，再慢慢地把一些额外的功能加进来。换言之，我们要把从简单到复杂的过程展现给读者看，这样你才会明白一个库是如何像雪人那样慢慢堆起来的。

本讲的代码放在slambook/project中。由于随着开发过程不断前进，我们会对工程做一些删改，因此它的内容也会发生变化。所以我们会把中间的代码也保留在目录中，以版本号命名，以便读者随时查看、模仿。

9.1.1 确定程序框架

根据前两讲的内容，我们知道视觉里程计分单目、双目、RGB-D三大类。单目视觉相对复杂，而RGB-D最为简单，没有初始化，也没有尺度问题。本着由简入繁的指导思想，我们先从RGB-D做起。为了方便读者做实验，我们将使用数据集而非实际的RGB-D相机（因为不能保证读者人手一台RGB-D相机）。

首先，我们来了解一下Linux程序的组织方式。在编写一个小规模的库时，我们通常会建立一些文件夹，把源代码、头文件、文档、测试数据、配置文件、日志等分类存放，这样会显得很有条理。如果一个库内容很多，我们还会把代码分解成各个独立的小模块，以便测试。读者可以参照OpenCV或g2o的组织方式，看看一个大中型库是如何组织的。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

例如，OpenCV有core、imgproc、features2d等模块，每个模块分别负责不同的任务。g2o则有core、solvers、types等若干模块。不过在小型程序里，我们也可以把所有的东西糅在一起，称为SLAM库。

现在我们要写的SLAM库是一个小型库，目标是帮读者将本书用到的各种算法融会贯通，书写自己的SLAM程序。挑选一个工程目录，在其下面建立如下文件夹来组织代码文件：

1.bin用来存放可执行的二进制文件。

2.include/myslam存放SLAM模块的头文件，主要是.h文件。这种做法的理由是，当把包含目录设到include，引用自己的头文件时，需要写include"myslam/xxx.h"，这样不容易和别的库混淆。

3.src存放源代码文件，主要是.cpp文件。

4.test存放测试用的文件，也是.cpp文件。

5.lib存放编译好的库文件。

6.con fi g存放配置文件。

7.cmake_modules第三方库的cmake文件，在使用g2o之类的库时会用到它。

以上就是我们的目录结构，如图9-2所示。相比于之前每一讲内零零散散地放着的main.cpp，这种做法显得更有条理。接下来，我们会在这些目录里不断地添加新文件，逐渐形成一个完整的程序。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

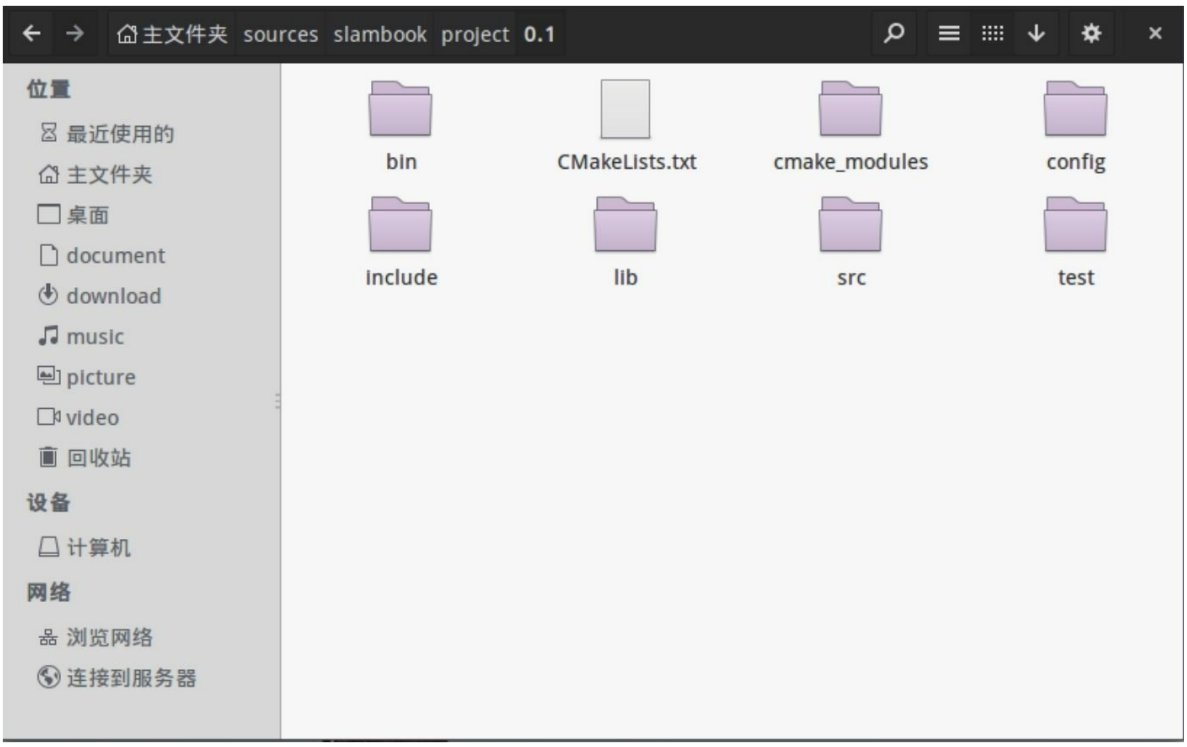


图9-2　工程项目的目录。

9.1.2　确定基本数据结构

为了让程序跑起来，我们要设计好数据单元，以及程序处理的流程。这好比构成房屋的一个个的柱子和砖块。那么，在一个SLAM程序中，有哪些结构是最基本的呢？我们抽象出以下基本概念：

1.帧：帧是相机采集到的图像单位。它主要包含一个图像（RGB-D情形下是一对图像）。此外，还有特征点、位姿、内参等信息。在视觉SLAM中我们会谈论关键帧（Key-frame）。由于相机采集的数据很多，存储所有的数据显然是不现实的。否则，如果相机放在桌上不动，程序的内存占用也会越来越高直至无法接受。通常的做法是把某些我们认为更重要的帧保存起来，并认为相机轨迹可以用这些关键帧来描述。关键帧如何选择是一个很大的问题，而且基于工程经验，很少有理论上的指导。在本书中我们会使用一个关键帧选择方法，但读者亦可考虑自己提出新的方式。

2.路标：路标点即图像中的特征点。在相机运动后，我们还能估计它们的3D位置。通常，会把路标点放在一个地图当中，并将新来的帧

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

与地图中的路标点进行匹配，估计相机位姿。

帧的位姿与路标的位置估计相当于一个局部的SLAM问题。除此之外，我们还需要一些工具，让程序写起来更流畅。例如：

1.配置文件：在写程序过程中你会经常遇到各种各样的参数，比如，相机的内参、特征点的数量、匹配时选择的比例，等等。你可以把这些数写在程序中，但那不是一个好习惯。你会经常修改这些参数，但每次修改后都要重新编译一遍程序。当其数量越来越多时，修改就变得越来越困难。所以，更好的方式是在外部定义一个配置文件，程序运行时读取该配置文件中的参数值。这样，每次只要修改配置文件内容就行了，不必对程序本身做任何修改。

2.坐标变换：你会经常需要在坐标系间进行坐标变换，例如，世界坐标到相机坐标、相机坐标到归一化相机坐标、归一化相机坐标到像素坐标，等等。定义一个类把这些操作都放在一起将更方便。

下面我们就来定义帧、路标这几个概念，在C++中都以类来表示。我们尽量保证一个类有单独的头文件和源文件，避免把许多个类放在同一个文件中。然后，把函数声明放在头文件，实现放在源文件中（除非函数很短，也可以写在头文件中）。我们参照Google的命名规范，同时考虑尽量以初学者也能看懂的方式来写程序。由于我们的程序是偏向算法而非软件工程的，所以不讨论复杂的类继承关系、接口、模板等，而更关注算法的正确实现，以及是否便于扩展。我们会把数据成员设置为公有，尽管这在C++软件设计中是应该避免的，如果读者愿意，也可以把它们改成private或protected接口，并添加设置和获取接口。在过程较为复杂的算法中，我们会把它分解成若干步骤，例如特征提取和匹配应该分别在不同的函数中实现，这样，当我们想修改算法流程时，就不需要修改整个运行流程，只需调整局部的处理方式即可。

现在，让我们开始写VO。我们把这个版本定为0.1版，表示这是刚开始的阶段。我们一共写5个类：Frame为帧，Camera为相机模型，MapPoint为特征点/路标点，Map管理特征点，Con fi g提供配置参数。它们的关系如图9-3所示。我们现在只写它们的数据成员和常用方法，而在后面用到更多内容时再行添加。

书籍汇总下载 http://t.cn/RDIAj5D

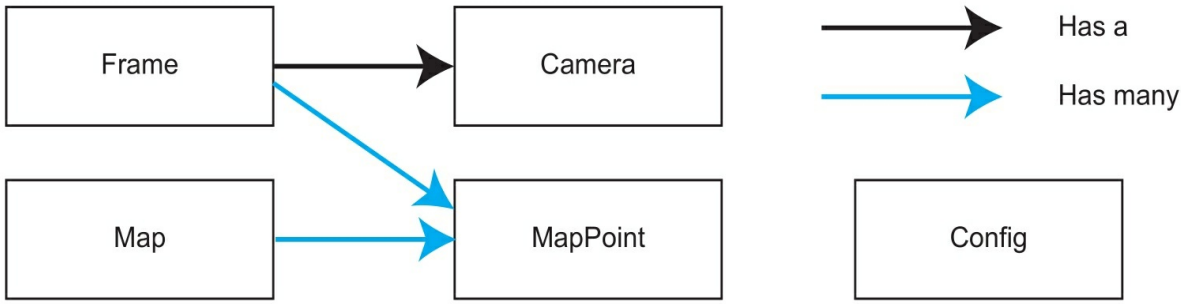


图9-3　基本类的关系示意图。

Camera类最简单，我们先来实现它。

9.1.3　Camera类

Camera类存储相机的内参和外参，并完成相机坐标系、像素坐标系和世界坐标系之间的坐标变换。当然，在世界坐标系中你需要一个相机的（变动的）外参，我们以参数的形式传入。

slambook/project/0.1/include/myslam/camera.h

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
#ifndef CAMERA_H
#define CAMERA_H
#include "myslam/common_include.h"
namespace myslam
{
// Pinhole RGB-D camera model
class Camera
{
public:
    typedef std::shared_ptr<Camera> Ptr;
    float   fx_, fy_, cx_, cy_, depth_scale_;  // Camera intrinsics

    Camera();
    Camera ( float fx, float fy, float cx, float cy, float depth_scale=0 ) :
    fx_ ( fx ), fy_ ( fy ), cx_ ( cx ), cy_ ( cy ), depth_scale_ ( depth_scale )
    {}

    // coordinate transform: world, camera, pixel
    Vector3d world2camera( const Vector3d& p_w, const SE3& T_c_w );
    Vector3d camera2world( const Vector3d& p_c, const SE3& T_c_w );
    Vector2d camera2pixel( const Vector3d& p_c );
    Vector3d pixel2camera( const Vector2d& p_p, double depth=1 );
    Vector3d pixel2world ( const Vector2d& p_p, const SE3& T_c_w, double depth=1 );
    Vector2d world2pixel ( const Vector3d& p_w, const SE3& T_c_w );
};
}
#endif // CAMERA_H
```

说明如下（由上往下）：

1.在这个简单的例子中，我们给出了防止头文件重复引用的ifndef宏定义。如果没有这个宏，在两处引用此头文件时将出现类的重复定义。所以，在每个程序头文件里都会定义这样一个宏。

2.我们用命名空间namespace　myslam将类定义包裹起来（因为是我们自己写的SLAM，所以命名空间就叫myslam了）。命名空间可以防止我们不小心定义出别的库里同名的函数，也是一种比较安全和规范的做法。由于宏定义和命名空间在每个文件中都会写一遍，所以我们只在这

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

里稍加介绍，后面将略去。

3.我们把一些常用的头文件放在common_include.h文件中，这样就可以避免每次书写很长的一串include。

4.我们把智能指针定义成Camera的指针类型，因此以后在传递参数时使用Camera::Ptr类型即可。

5.我们用Sophus::SE3来表达相机的位姿。Sophus库在李代数一讲介绍过。

在源文件中，给出Camera方法的实现：

slambook/project/0.1/src/camera.cpp

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```cpp
#include "myslam/camera.h"
namespace myslam
{

Camera::Camera()
{
}

Vector3d Camera::world2camera ( const Vector3d& p_w, const SE3& T_c_w )
{
    return T_c_w*p_w;
}

Vector3d Camera::camera2world ( const Vector3d& p_c, const SE3& T_c_w )
{
    return T_c_w.inverse() *p_c;
}

Vector2d Camera::camera2pixel ( const Vector3d& p_c )
{
    return Vector2d (
        fx_ * p_c ( 0,0 ) / p_c ( 2,0 ) + cx_,
        fy_ * p_c ( 1,0 ) / p_c ( 2,0 ) + cy_
    );
}

Vector3d Camera::pixel2camera ( const Vector2d& p_p, double depth )
{
    return Vector3d (
        ( p_p ( 0,0 )-cx_ ) *depth/fx_,
        ( p_p ( 1,0 )-cy_ ) *depth/fy_,
        depth
    );
}

Vector2d Camera::world2pixel ( const Vector3d& p_w, const SE3& T_c_w )
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
37 {
38     return camera2pixel ( world2camera ( p_w, T_c_w ) );
39 }
40
41 Vector3d Camera::pixel2world ( const Vector2d& p_p, const SE3& T_c_w, double depth
   )
42 {
43     return camera2world ( pixel2camera ( p_p, depth ), T_c_w );
44 }
45 }
```

读者可以对照一下这些方法是否和第5讲的内容一致。它们完成了像素坐标系、相机坐标系和世界坐标系间的坐标变换。

9.1.4 Frame类

下面来考虑Frame类。Frame类是基本数据单元，在许多地方会用到，但现在是初期设计阶段，我们还不清楚以后可能新加的内容。所以这里的Frame类只提供基本的数据存储和接口。如果之后有新增的内容，再继续往里添加。

slambook/project/0.1/include/myslam/frame.h

书籍汇总下载 http://t.cn/RDIAj5D

```
class Frame
{
public:
    typedef std::shared_ptr<Frame> Ptr;
    unsigned long  id_; // id of this frame
    double time_stamp_; // when it is recorded
    SE3 T_c_w_; // transform from world to camera
    Camera::Ptr camera_; // Pinhole RGB-D Camera model
    Mat color_, depth_; // color and depth image

public: // data members
    Frame();
    Frame( long id, double time_stamp=0, SE3 T_c_w=SE3(), Camera::Ptr camera=
    nullptr, Mat color=Mat(), Mat depth=Mat() );
    ~Frame();

    // factory function
    static Frame::Ptr createFrame();

    // find the depth in depth map
    double findDepth( const cv::KeyPoint& kp );

    // Get Camera Center
    Vector3d getCamCenter() const;

    // check if a point is in this frame
    bool isInFrame( const Vector3d& pt_world );
};
```

在Frame中，我们定义了ID、时间戳、位姿、相机、图像这几个量，这应该是一个帧当中含有的最重要的信息。在方法中，我们提取了几个重要的方法：创建Frame、寻找给定点对应的深度、获取相机光心、判断某个点是否在视野内，等等。它们的实现是比较平凡的，请读者参考frame.cpp了解这些函数的具体实现。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

9.1.5 MapPoint类

MapPoint表示路标点。我们将估计它的世界坐标，并且会拿当前帧提取到的特征点与地图中的路标点匹配，来估计相机的运动，因此还需要存储它对应的描述子。此外，我们会记录一个点被观测到的次数和被匹配的次数，作为评价其好坏程度的指标。

slambook/project/0.1/include/myslam/frame.h

```
class MapPoint
{
public:
    typedef shared_ptr<MapPoint> Ptr;
    unsigned long      id_; // ID
    Vector3d    pos_;       // Position in world
    Vector3d    norm_;      // Normal of viewing direction
    Mat         descriptor_; // Descriptor for matching
    int         observed_times_;    // being observed by feature matching algo.
    int         correct_times_;     // being an inliner in pose estimation

    MapPoint();
    MapPoint( long id, Vector3d position, Vector3d norm );

    // factory function
    static MapPoint::Ptr createMapPoint();
};
```

同样，读者可以浏览src/map.cpp查看其实现。目前为止我们只考虑这些数据成员的初始化问题。

9.1.6 Map类

Map类管理着所有的路标点，并负责添加新路标、删除不好的路标等工作。VO的匹配过程只要和Map打交道即可。当然Map也会有很多操作，但现阶段我们只定义主要的数据结构。

slambook/project/0.1/include/myslam/map.h

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
class Map
{
public:
    typedef shared_ptr<Map> Ptr;
    unordered_map<unsigned long, MapPoint::Ptr >  map_points_;        // all
    landmarks
    unordered_map<unsigned long, Frame::Ptr >     keyframes_;         // all key-
    frames

    Map() {}

    void insertKeyFrame( Frame::Ptr frame );
    void insertMapPoint( MapPoint::Ptr map_point );
};
```

Map类中实际存储了各个关键帧和路标点，既需要随机访问，又需要随时插入和删除，因此我们使用散列（Hash）来进行存储。

9.1.7 Con fi g类

Con fi g类负责参数文件的读取，并在程序的任意地方都可随时提供参数的值。所以我们把Con fi g写成单件模式（Singleton）。它只有一个全局对象，当我们设置参数文件时，创建该对象并读取参数文件，随后就可以在任意地方访问参数值，最后在程序结束时自动销毁。

slambook/project/0.1/include/myslam/con fi g.h

```
class Config
{
private:
    static std::shared_ptr<Config> config_;
    cv::FileStorage file_;

    Config () {} // private constructor makes a singleton
public:
    ~Config();  // close the file when deconstructing

    // set a new config file
    static void setParameterFile( const std::string& filename );

    // access the parameter values
    template< typename T >
    static T get( const std::string& key )
    {
        return T( Config::config_->file_[key] );
    }
};
```

说明如下：

1.我们把构造函数声明为私有，防止这个类的对象在别处建立，它只能在setParameterFile时构造。实际构造的对象是Con fi g的智能指针：static shared_ptr<Con fi g>con fi g_。用智能指针的原因是可以自动析构，省得我们再调一个别的函数来做析构。

2.在文件读取方面，我们使用OpenCV提供的FileStorage类。它可以读取一个YAML文件，且可以访问其中任意一个字段。由于参数实质值可能为整数、浮点数或字符串，所以我们通过一个模板函数get来获得任意类型的参数值。

下面是Con fi g的实现。注意，我们把单例模式的全局指针定义在此源文件中了：

slambook/project/0.1/src/con fi g.cpp

书籍汇总下载 http://t.cn/RDIAj5D

```
void Config::setParameterFile( const std::string& filename )
{
    if ( config_ == nullptr )
        config_ = shared_ptr<Config>(new Config);
    config_->file_ = cv::FileStorage( filename.c_str(), cv::FileStorage::READ );
    if ( config_->file_.isOpened() == false )
    {
        std::cerr<<"parameter file "<<filename<<" does not exist."<<std::endl;
        config_->file_.release();
        return ;
    }
}
Config::~Config()
{
    if ( file_.isOpened() )
        file_.release();
}
shared_ptr<Config> Config::config_ = nullptr;
```

在实现中，我们只要判断一下参数文件是否存在即可。定义了这个Con fi g类后，我们可以在任何地方获取参数文件里的参数。例如，当想要定义相机的焦距f_x时，按照如下步骤操作即可：

1.在参数文件中加入：“Camera.fx:500”。

2.在代码中使用：

```
myslam::Config::setParameterFile("parameter.yaml");
double fx = myslam::Config::get<double> ("Camera.fx");
```

就能获得f_x的值了。

当然，参数文件的实现方法绝对不止这一种。我们主要从程序开发上的便利角度来考虑这个实现，读者当然也可以用更简单的方式来实现参数的配置。

至此，我们定义了SLAM程序的基本数据结构，书写了若干个基本

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

类。这好比是造房子的砖头和水泥。你可以调用cmake编译这个0.1版，尽管它还没有实质性的功能。接下来我们来考虑把前面讲过的VO算法加到工程中，并做一些测试来调整各算法的性能。注意，笔者会刻意地暴露某些设计的问题，所以你看到的实现不见得就是最好的（或者足够好的）。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

9.2 基本的VO：特征提取和匹配

下面我们来实现VO，先来考虑特征点法。它的任务是，根据输入的图像，计算相机运动和特征点位置。前面我们讨论的都是在两两帧间的位姿估计，然而我们将发现，仅凭两帧的估计是不够的。我们会把特征点缓存成一个小地图，计算当前帧与地图之间的位置关系。但那样程序会复杂一些，所以，让我们先订个小目标，暂时从两两帧间的运动估计出发。

9.2.1 两两帧的视觉里程计

如果像前面两讲一样，只关心两个帧之间的运动估计，并且不优化特征点的位置。然后把估得的位姿“串”起来，也能得到一条运动轨迹。这种方式可以看成两两帧间的（Pairwise）无结构（Structureless）的VO，实现起来最为简单，但是效果不佳。为什么不佳呢？我们一起来体验一下。记该工程为0.2版本。

两两帧之间的VO工作示意图如图9-4所示。在这种VO里，我们定义了参考帧（Reference）和当前帧（Current）这两个概念。以参考帧为坐标系，我们把当前帧与它进行特征匹配，并估计运动关系。假设参考帧相对世界坐标的变换矩阵为T_{rw}，当前帧与世界坐标系间为T_{cw}，则待估计的运动与这两个帧的变换矩阵构成左乘关系：

$$\boldsymbol{T}_{cr}, \quad \text{s.t.} \quad \boldsymbol{T}_{cw} = \boldsymbol{T}_{cr}\boldsymbol{T}_{rw}.$$

在t-1到t时刻，我们以t-1为参考，求取t时刻的运动。这可以通过特征点匹配、光流或直接法得到，但这里我们只关心运动，不关心结构。换句话说，只要通过特征点成功求出了运动，我们就不再需要这一帧的特征点了。这种做法当然会有缺陷，但是忽略掉数量庞大的特征点可以节省许多的计算量。然后，在t到t+1时刻，我们又以t时刻为参考帧，考虑t到t+1间的运动关系。如此往复，就得到了一条运动轨迹。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

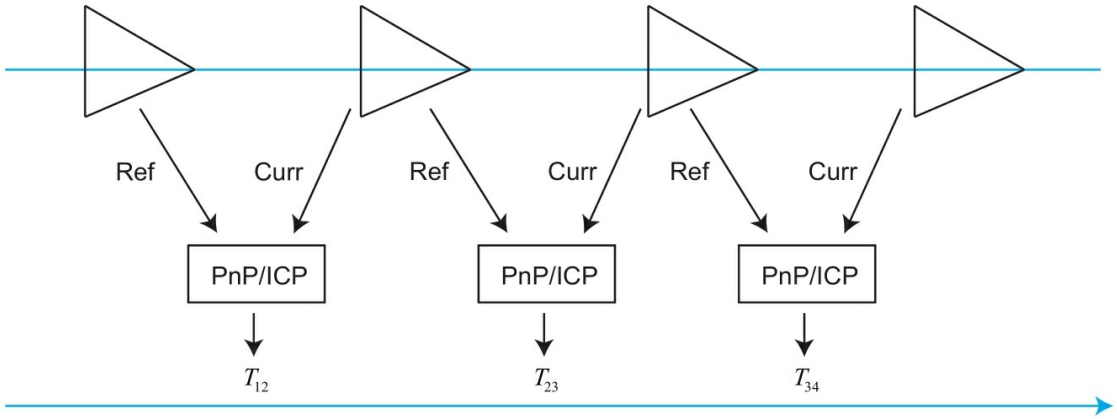


图9-4　两两帧的VO示意图。

这种VO的工作方式是简单的，不过实现也可以有若干种。我们以传统的匹配特征点——求PnP的方法为例实现一遍。希望读者能够结合之前几讲的知识，自己实现一下光流/直接法或ICP求运动的VO。在匹配特征点的方式中，最重要的是参考帧与当前帧之间的特征匹配关系，它的流程可归纳如下：

1.对新来的当前帧，提取关键点和描述子。

2.如果系统未初始化，以该帧为参考帧，根据深度图计算关键点的3D位置，返回第1步。

3.估计参考帧与当前帧间的运动。

4.判断上述估计是否成功。

5.若成功，把当前帧作为新的参考帧，返回第1步。

6.若失败，计录连续丢失帧数。当连续丢失超过一定帧数时，置VO状态为丢失，算法结束。若未超过，返回第1步。

VisualOdometry类给出了上述算法的实现。

slambook/project/0.2/include/myslam/visual_odometry.h

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
class VisualOdometry
{
public:
    typedef shared_ptr<VisualOdometry> Ptr;
    enum VOState {
        INITIALIZING=-1,
        OK=0,
        LOST
    };

```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
    VOState     state_; // current VO status
    Map::Ptr    map_; // map with all frames and map points
    Frame::Ptr  ref_; // reference frame
    Frame::Ptr  curr_; // current frame

    cv::Ptr<cv::ORB> orb_; // orb detector and computer
    vector<cv::Point3f> pts_3d_ref_; // 3d points in reference frame
    vector<cv::KeyPoint> keypoints_curr_; // keypoints in current frame
    Mat descriptors_curr_; // descriptor in current frame
    Mat descriptors_ref_; // descriptor in reference frame
    vector<cv::DMatch> feature_matches_;

    SE3 T_c_r_estimated_; // the estimated pose of current frame
    int num_inliers_; // number of inlier features in icp
    int num_lost_; // number of lost times

    // parameters
    int num_of_features_; // number of features
    double scale_factor_; // scale in image pyramid
    int level_pyramid_; // number of pyramid levels
    float match_ratio_; // ratio for selecting  good matches
    int max_num_lost_; // max number of continuous lost times
    int min_inliers_; // minimum inliers

    double key_frame_min_rot; // minimal rotation of two key-frames
    double key_frame_min_trans; // minimal translation of two key-frames

public: // functions
    VisualOdometry();
    ~VisualOdometry();

    bool addFrame( Frame::Ptr frame );      // add a new frame

protected:
    // inner operation
    void extractKeyPoints();
    void computeDescriptors();
    void featureMatching();
    void poseEstimationPnP();
    void setRef3DPoints();

    void addKeyFrame();
    bool checkEstimatedPose();
    bool checkKeyFrame();
};
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

关于这个VisualOdometry类，有几点需要解释：

1.VO本身有若干种状态：设定第一帧、顺利跟踪或丢失，你可以把它看成一个有限状态机（Finite State Machine，FSM）。当然状态也可以有更多种，例如，单目VO至少还有一个初始化状态。在我们的实现中，考虑最简单的三个状态：初始化、正常、丢失。

2.我们把一些中间变量定义在类中，这样可省去复杂的参数传递。因为它们都是定义在类内部的，所以各个函数都可以访问它们。

3.特征提取和匹配当中的参数从参数文件中读取。例如：

```
1 VisualOdometry::VisualOdometry() :
2     state_ ( INITIALIZING ), ref_ ( nullptr ), curr_ ( nullptr ), map_ (
      new Map ), num_lost_ ( 0 ), num_inliers_ ( 0 )
3 {
4     num_of_features_    = Config::get<int> ( "number_of_features" );
5     scale_factor_       = Config::get<double> ( "scale_factor" );
6     level_pyramid_      = Config::get<int> ( "level_pyramid" );
7     match_ratio_        = Config::get<float> ( "match_ratio" );
8     ...
9 }
```

4.addFrame函数是外部调用的接口。使用VO时，将图像数据装入Frame类后，调用addFrame估计其位姿。该函数根据VO所处的状态实现不同的操作：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
bool VisualOdometry::addFrame ( Frame::Ptr frame )
{
    switch ( state_ )
    {
    case INITIALIZING:
    {
        state_ = OK;
        curr_ = ref_ = frame;
        map_->insertKeyFrame ( frame );
        // extract features from first frame
        extractKeyPoints();
        computeDescriptors();
        // compute the 3d position of features in ref frame
        setRef3DPoints();
        break;
    }
    case OK:
    {
        curr_ = frame;
        extractKeyPoints();
        computeDescriptors();
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
22          featureMatching();
23          poseEstimationPnP();
24          if ( checkEstimatedPose() == true ) // a good estimation
25          {
26              curr_->T_c_w_ = T_c_r_estimated_ * ref_->T_c_w_;  // T_c_w =
                T_c_r*T_r_w
27              ref_ = curr_;
28              setRef3DPoints();
29              num_lost_ = 0;
30              if ( checkKeyFrame() == true ) // is a key-frame
31              {
32                  addKeyFrame();
33              }
34          }
35          else // bad estimation due to various reasons
36          {
37              num_lost_++;
38              if ( num_lost_ > max_num_lost_ )
39              {
40                  state_ = LOST;
41              }
42              return false;
43          }
44          break;
45      }
46      case LOST:
47      {
48          cout<<"vo has lost."<<endl;
49          break;
50      }
51      }
52      return true;
53  }
```

值得一提的是，由于各种原因，我们设计的上述VO算法，每一步都有可能失败。例如，图片中不易提特征、特征点缺少深度值、误匹配、运动估计出错，等等。因此，要设计一个健壮的VO，必须（最好是显式地）考虑到上述所有可能出错的地方——那自然会使程序变得非

书籍汇总下载 http://t.cn/RDIAj5D

常复杂。我们在checkEstimatedPose中根据内点（inlier）的数量及运动的大小做一个简单的检测：认为内点不可太少，而运动不可能过大。当然，读者也可以思考其他检测问题的手段，尝试一下效果。

我们略去VisualOdometry类其余的实现，读者可在GitHub上找到所有的源代码。最后，我们在test中加入该VO的测试程序，使用数据集观察估计的运动效果：

slambook/project/0.2/test/run_vo.cpp

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
int main ( int argc, char** argv )
{
    if ( argc != 2 )
    {
        cout<<"usage: run_vo parameter_file"<<endl;
        return 1;
    }

    myslam::Config::setParameterFile ( argv[1] );
    myslam::VisualOdometry::Ptr vo ( new myslam::VisualOdometry );

    string dataset_dir = myslam::Config::get<string> ( "dataset_dir" );
    cout<<"dataset: "<<dataset_dir<<endl;
    ifstream fin ( dataset_dir+"/associate.txt" );
    if ( !fin )
    {
        cout<<"please generate the associate file called associate.txt!"<<endl;
        return 1;
    }

    vector<string> rgb_files, depth_files;
    vector<double> rgb_times, depth_times;
    while ( !fin.eof() )
    {
        string rgb_time, rgb_file, depth_time, depth_file;
        fin>>rgb_time>>rgb_file>>depth_time>>depth_file;
        rgb_times.push_back ( atof ( rgb_time.c_str() ) );
        depth_times.push_back ( atof ( depth_time.c_str() ) );
        rgb_files.push_back ( dataset_dir+"/"+rgb_file );
        depth_files.push_back ( dataset_dir+"/"+depth_file );

        if ( fin.good() == false )
            break;
    }

    myslam::Camera::Ptr camera ( new myslam::Camera );

    // visualization
    cv::viz::Viz3d vis("Visual Odometry");
    cv::viz::WCoordinateSystem world_coor(1.0), camera_coor(0.5);
    cv::Point3d cam_pos( 0, -1.0, -1.0 ), cam_focal_point(0,0,0), cam_y_dir(0,1,0);
    cv::Affine3d cam_pose = cv::viz::makeCameraPose( cam_pos, cam_focal_point,
    cam_y_dir );
    vis.setViewerPose( cam_pose );
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
44
45      world_coor.setRenderingProperty(cv::viz::LINE_WIDTH, 2.0);
46      camera_coor.setRenderingProperty(cv::viz::LINE_WIDTH, 1.0);
47      vis.showWidget( "World", world_coor );
48      vis.showWidget( "Camera", camera_coor );
49
50      cout<<"read total "<<rgb_files.size() <<" entries"<<endl;
51      for ( int i=0; i<rgb_files.size(); i++ )
52      {
53          Mat color = cv::imread ( rgb_files[i] );
54          Mat depth = cv::imread ( depth_files[i], -1 );
55          if ( color.data==nullptr || depth.data==nullptr )
56          break;
57          myslam::Frame::Ptr pFrame = myslam::Frame::createFrame();
58          pFrame->camera_ = camera;
59          pFrame->color_ = color;
60          pFrame->depth_ = depth;
61          pFrame->time_stamp_ = rgb_times[i];
62
63          boost::timer timer;
64          vo->addFrame ( pFrame );
65          cout<<"VO costs time: "<<timer.elapsed()<<endl;
66
67          if ( vo->state_ == myslam::VisualOdometry::LOST )
68              break;
69          SE3 Tcw = pFrame->T_c_w_.inverse();
70
71          // show the map and the camera pose
72          cv::Affine3d M(
73              cv::Affine3d::Mat3(
74                  Tcw.rotation_matrix()(0,0), Tcw.rotation_matrix()(0,1), Tcw.
                    rotation_matrix()(0,2),
75                  Tcw.rotation_matrix()(1,0), Tcw.rotation_matrix()(1,1), Tcw.
                    rotation_matrix()(1,2),
76                  Tcw.rotation_matrix()(2,0), Tcw.rotation_matrix()(2,1), Tcw.
                    rotation_matrix()(2,2)
77              ),
78              cv::Affine3d::Vec3(
79                  Tcw.translation()(0,0), Tcw.translation()(1,0), Tcw.translation()
                    (2,0)
80              )
81          );
82          cv::imshow("image", color );
83          cv::waitKey(1);
84          vis.setWidgetPose( "Camera", M);
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
        vis.spinOnce(1, false);
    }
    return 0;
}
```

为了运行这个程序，需要做几件事：

1.因为我们用OpenCV　3的viz模块显示估计位姿，请确保你安装的是OpenCV 3，并且viz模块也已编译安装。

2.准备TUM数据集中的其中一个。简单起见，笔者推荐fr1_xyz那一个。请使用associate.py生成一个配对文件associate.txt。关于TUM数据集格式，在8.3节中已介绍。

3.在con fi g/default.yaml中填写你的数据集所在路径，参照笔者的写法即可。然后，用

```
bin/run_vo config/default.yaml
```

执行程序，就可以看到实时的演示了，如图9-5所示。

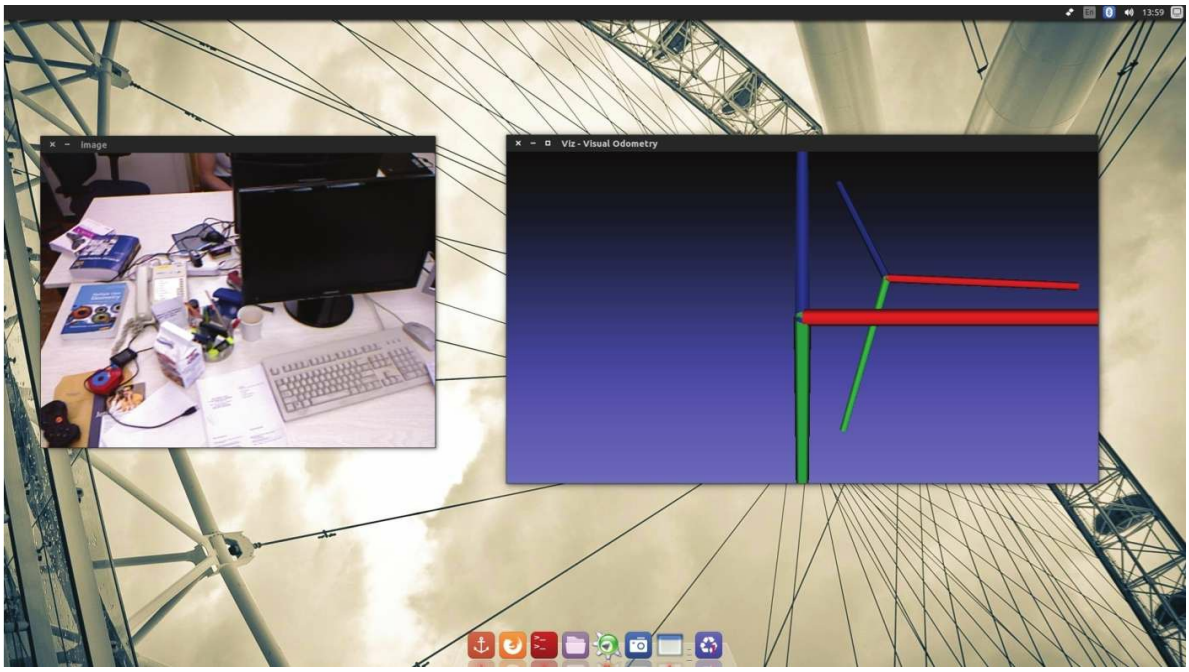


图9-5　0.2版本的VO演示。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

在演示程序中，你可以看到当前帧的图像与它的估计位置。我们画出了世界坐标系的坐标轴（大）与当前帧的坐标轴（小），颜色与轴的对应关系为：蓝色—Z，红色—X，绿色—Y。你可以直观地感受到相机的运动，它与我们人类的感觉是大致相符的，尽管效果距离预想还有一定的差距。程序还输出了VO单次计算的用时，在笔者的机器上，能够以每次花费30ms左右的速度运行。减少特征点数量可以提高运算速度。读者可以修改运行参数和数据集，看看它在各种情况下的表现。

9.2.2 讨论

本节，我们实现了一个简单的两两帧间的视觉里程计，然而不管从速度还是精度上来说，它的效果都不理想。似乎这种看似简单的思路并不能得到很好的结果。我们来考虑一下有哪些可能的原因：

1.在位姿估计时，我们使用了RANSAC求出的PnP解。由于RANSAC只采用少数几个随机点来计算PnP，虽然能够确定inlier，但该方法容易受噪声影响。在3D-2D点存在噪声的情形下，我们要用RANSAC的解作为初值，再用非线性优化求一个最优值。下一节将说明这种做法是优于现在的做法的。

2.由于现在的VO是无结构的，特征点的3D位置被当作真值来估计运动。但实际上，RGB-D的深度图必定带有一定的误差，特别是那些深度过近或过远的地方。并且，由于特征点往往位于物体的边缘处，那些地方的深度测量值通常是不准确的。所以现在的做法不够精确，我们需要把特征点也放在一起优化。

3.只考虑参考帧/当前帧的方式，一方面使得位姿估计过于依赖参考帧。如果参考帧质量太差，比如，出现严重遮挡、光照变化的情况下，跟踪容易丢失。并且，当参考帧位姿估计不准时，还会出现明显的漂移。另一方面，仅使用两帧数据显然没有充分地利用所有的信息。更自然的方式是比较当前帧和地图，而不是比较当前帧和参考帧。于是，我们要关心如何把当前帧与地图进行匹配，以及如何优化地图点的问题。

4.由于输出了各步骤的运行时间，我们可以对计算量有一个大概的了解（表9-1）。

书籍汇总下载 http://t.cn/RDIAj5D

表9-1某一次循环的各步骤用时

项目	特征提取	描述子计算	特征匹配	PnP 求解	其他	总计
时间	0.0102	0.0087	0.0118	0.0011	0.0001	0.0319

可以看到，特征点的提取和匹配占用了绝大多数的计算时间，而看似复杂的PnP优化，计算量与之相比基本可以忽略。因此，如何提高特征提取、匹配算法的速度，将是特征点方法的一个重要的主题。一种可预见的方式是使用直接法/光流，可有效地避开繁重的特征计算工作。本书之前已讨论过直接法和光流法，读者不妨自行尝试一下。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

9.3 改进：优化PnP的结果

接下来，我们沿着之前的内容，尝试一些改进VO的方法。本节中，我们来尝试RANSAC PnP加上迭代优化的方式估计相机位姿，看看是否对前一节的效果有所改进。

非线性优化问题的求解，已经在第6讲和第7讲介绍过了。由于本节的目标是估计位姿而非结构，我们以相机位姿ξ为优化变量，通过最小化重投影误差来构建优化问题。与之前一样，我们自定义一个g2o中的优化边。它只优化一个位姿，因此是一个一元边。

slambook/project/0.3/include/myslam/g2o_types.h

```
class EdgeProjectXYZ2UVPoseOnly: public g2o::BaseUnaryEdge<2, Eigen::Vector2d, g2o
::VertexSE3Expmap >
{
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW

    virtual void computeError();
    virtual void linearizeOplus();

    virtual bool read( std::istream& in ){}
    virtual bool write(std::ostream& os) const {};

    Vector3d point_;
    Camera* camera_;
};
```

把三维点和相机模型放入它的成员变量中，方便计算重投影误差和雅可比矩阵：

slambook/project/0.3/src/g2o_types.cpp

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
void EdgeProjectXYZ2UVPoseOnly::computeError()
{
const g2o::VertexSE3Expmap* pose = static_cast<const g2o::VertexSE3Expmap*> ( _vertices[0] );
    _error = _measurement - camera_->camera2pixel (
        pose->estimate().map(point_)
    );
}

void EdgeProjectXYZ2UVPoseOnly::linearizeOplus()
{
    g2o::VertexSE3Expmap* pose = static_cast<g2o::VertexSE3Expmap*> ( _vertices[0] );
    g2o::SE3Quat T ( pose->estimate() );
    Vector3d xyz_trans = T.map ( point_ );
    double x = xyz_trans[0];
    double y = xyz_trans[1];
    double z = xyz_trans[2];
    double z_2 = z*z;

    _jacobianOplusXi ( 0,0 ) =  x*y/z_2 *camera_->fx_;
    _jacobianOplusXi ( 0,1 ) = - ( 1+ ( x*x/z_2 ) ) *camera_->fx_;
    _jacobianOplusXi ( 0,2 ) = y/z * camera_->fx_;
    _jacobianOplusXi ( 0,3 ) = -1./z * camera_->fx_;
    _jacobianOplusXi ( 0,4 ) = 0;
    _jacobianOplusXi ( 0,5 ) = x/z_2 * camera_->fx_;

    _jacobianOplusXi ( 1,0 ) = ( 1+y*y/z_2 ) *camera_->fy_;
    _jacobianOplusXi ( 1,1 ) = -x*y/z_2 *camera_->fy_;
    _jacobianOplusXi ( 1,2 ) = -x/z *camera_->fy_;
    _jacobianOplusXi ( 1,3 ) = 0;
    _jacobianOplusXi ( 1,4 ) = -1./z *camera_->fy_;
    _jacobianOplusXi ( 1,5 ) = y/z_2 *camera_->fy_;
}
```

然后，在之前的PoseEstimationPnP函数中，修改成以RANSACPnP结果为初值，再调用g2o进行优化的形式：

书籍汇总下载 http://t.cn/RDIAj5D

slambook/project/0.3/src/visual_odometry.cpp

```
void VisualOdometry::poseEstimationPnP()
{
    ...
    // using bundle adjustment to optimize the pose
    typedef g2o::BlockSolver<g2o::BlockSolverTraits<6,2>> Block;
    Block::LinearSolverType* linearSolver = new g2o::LinearSolverDense<Block::
    PoseMatrixType>();
    Block* solver_ptr = new Block( linearSolver );
    g2o::OptimizationAlgorithmLevenberg* solver = new g2o::
    OptimizationAlgorithmLevenberg ( solver_ptr );
    g2o::SparseOptimizer optimizer;
    optimizer.setAlgorithm ( solver );

    g2o::VertexSE3Expmap* pose = new g2o::VertexSE3Expmap();
    pose->setId ( 0 );
    pose->setEstimate ( g2o::SE3Quat (
        T_c_r_estimated_.rotation_matrix(), T_c_r_estimated_.translation()
    ) );
    optimizer.addVertex ( pose );

    // edges
    for ( int i=0; i<inliers.rows; i++ )
    {
        int index = inliers.at<int>(i,0);
        // 3D -> 2D projection
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
        EdgeProjectXYZ2UVPoseOnly* edge = new EdgeProjectXYZ2UVPoseOnly();
        edge->setId(i);
        edge->setVertex(0, pose);
        edge->camera_ = curr_->camera_.get();
        edge->point_ = Vector3d( pts3d[index].x, pts3d[index].y, pts3d[index].z );
        edge->setMeasurement( Vector2d(pts2d[index].x, pts2d[index].y) );
        edge->setInformation( Eigen::Matrix2d::Identity() );
        optimizer.addEdge( edge );
    }

    optimizer.initializeOptimization();
    optimizer.optimize(10);

    T_c_r_estimated_ = SE3 (
        pose->estimate().rotation(),
        pose->estimate().translation()
    );
```

请读者运行此程序，对比之前的结果。你将发现估计的运动明显稳定了很多。同时，由于新增的优化仍是无结构的，规模很小，对计算时间的影响基本可以忽略不计。整体的视觉里程计计算时间仍在30ms左右。

讨论

我们发现，引入迭代优化方法之后，估计结果的质量比纯粹RANSAC PnP有明显的提高。尽管我们依然仅使用两两帧间的信息，但得到的运动却更加准确、平稳。从这次改进中我们看到了优化的重要性。不过，0.3版本的VO仍受两两帧间匹配的局限性影响。一旦视频序列当中某个帧丢失，就会导致后续的帧也无法和上一帧匹配。下面，我们把地图引入到VO中来。

书籍汇总下载 http://t.cn/RDIAj5D

9.4 改进：局部地图

本节我们将VO匹配到的特征点放到地图中，并将当前帧与地图点进行匹配，计算位姿。这种做法与之前的差异如图9-6所示。

在两两帧间比较时，我们只计算参考帧与当前帧之间的特征匹配和运动关系，在计算之后把当前帧设为新的参考帧。而在使用地图的VO中，每个帧为地图贡献一些信息，比如，添加新的特征点或更新旧特征点的位置估计。地图中的特征点位置往往是使用世界坐标的。因此，当前帧到来时，我们求它和地图之间的特征匹配与运动关系，即直接计算T_{cw}。

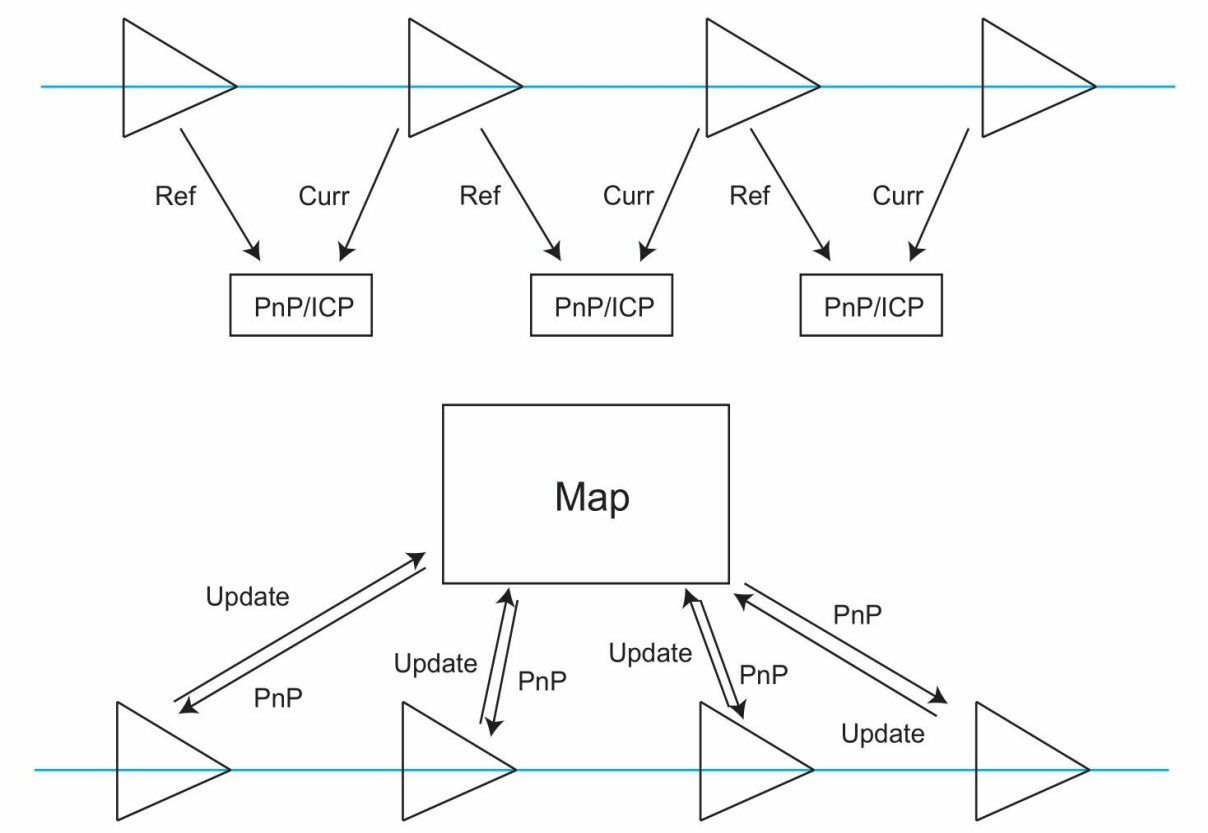


图9-6　两两帧VO与地图VO工作原理的差异。

这样做的好处是，我们能够维护一个不断更新的地图。只要地图是正确的，即使中间某帧出了差错，仍有希望求出之后那些帧的正确位

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

置。请注意，我们现在还没有详细地讨论SLAM的建图问题，所以这里的地图仅是一个临时性的概念，指的是把各帧特征点缓存到一个地方构成的特征点的集合。

地图又可以分为局部（Local）地图和全局（Global）地图两种，由于用途不同，往往分开讨论。顾名思义，局部地图描述了附近的特征点信息——我们只保留离相机当前位置较近的特征点，而把远的或视野外的特征点丢掉。这些特征点是用来和当前帧匹配来求相机位置的，所以我们希望它能够做得比较快。另一方面，全局地图则记录了从SLAM运行以来的所有特征点。显然它的规模要大一些，主要用来表达整个环境，但是直接在全局地图上定位，对计算机的负担就太大了。它主要用于回环检测和地图表达。

在视觉里程计中，我们更关心可以直接用于定位的局部地图（如果决心要用地图的话）。所以本讲我们来维护一个局部地图。随着相机运动，我们往地图里添加新的特征点。我们仍然要提醒读者：是否使用地图取决于你对精度—效率这个矛盾的把握。我们完全可以出于效率的考量，使用两两无结构式的VO；也可以为了更好的精度，构建局部地图乃至考虑地图的优化。

局部地图的一件麻烦事是维护它的规模。为了保证实时性，我们需要保证地图规模不至于太大（否则匹配会消耗大量的时间）。此外，单个帧与地图的特征匹配存在着一些加速手段，但由于技术上比较复杂，我们的例程中就不给出了。

现在，来实现地图点类吧。我们稍加完善之前没有用到的MapPoint类，主要是它的构造函数和生成函数。

slambook/project/0.4/include/myslam/mappoint.h

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
class MapPoint
{
    public:
    typedef shared_ptr<MapPoint> Ptr;
    unsigned long      id_; // ID
    static unsigned long factory_id_; // factory id
    bool  good_; // wheter a good point
    Vector3d pos_; // Position in world
    Vector3d norm_; // Normal of viewing direction
    Mat descriptor_; // Descriptor for matching

    list<Frame*> observed_frames_;   // key-frames that can observe this point

    int matched_times_; // being an inliner in pose estimation
    int visible_times_;  // being visible in current frame

    MapPoint();
    MapPoint (
        unsigned long id,
        const Vector3d& position,
        const Vector3d& norm,
        Frame* frame=nullptr,
        const Mat& descriptor=Mat()
    );

    inline cv::Point3f getPositionCV() const {
        return cv::Point3f( pos_(0,0), pos_(1,0), pos_(2,0) );
    }

    static MapPoint::Ptr createMapPoint();
    static MapPoint::Ptr createMapPoint (
        const Vector3d& pos_world,
        const Vector3d& norm_,
        const Mat& descriptor,
        Frame* frame
    );
};
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

主要的修改在VisualOdometry类上。由于工作流程的改变，我们修改了它的几个主要函数，例如，每次循环中要对地图进行增删、统计每个地图点被观测到的次数，等等[2]。这些事情是比较琐碎的，所以我们还是建议读者仔细看看GitHub上提供的源代码。重点观察以下几项：

1.在提取第一帧的特征点之后，将第一帧的所有特征点全部放入地图中：

```
void VisualOdometry::addKeyFrame()
{
    if ( map_->keyframes_.empty() )
    {
        // first key-frame, add all 3d points into map
        for ( size_t i=0; i<keypoints_curr_.size(); i++ )
        {
            double d = curr_->findDepth ( keypoints_curr_[i] );
            if ( d < 0 )
                continue;
            Vector3d p_world = ref_->camera_->pixel2world (
                Vector2d ( keypoints_curr_[i].pt.x, keypoints_curr_[i].pt.y
                 ), curr_->T_c_w_, d
            );
            Vector3d n = p_world - ref_->getCamCenter();
            n.normalize();
            MapPoint::Ptr map_point = MapPoint::createMapPoint(
                p_world, n, descriptors_curr_.row(i).clone(), curr_.get()
            );
            map_->insertMapPoint( map_point );
        }
    }
    map_->insertKeyFrame ( curr_ );
    ref_ = curr_;
}
```

2.后续的帧中，使用OptimizeMap函数对地图进行优化。包括删除不在视野内的点，在匹配数量减少时添加新点，等等。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
void VisualOdometry::optimizeMap()
{
    // remove the hardly seen and no visible points
    for ( auto iter = map_->map_points_.begin(); iter != map_->map_points_.
    end(); )
    {
        if ( !curr_->isInFrame(iter->second->pos_) )
        {
            iter = map_->map_points_.erase(iter);
            continue;
```

```
        }
        float match_ratio = float(iter->second->matched_times_)/iter->
        second->visible_times_;
        if ( match_ratio < map_point_erase_ratio_ )
        {
            iter = map_->map_points_.erase(iter);
            continue;
        }
        double angle = getViewAngle( curr_, iter->second );
        if ( angle > M_PI/6. )
        {
            iter = map_->map_points_.erase(iter);
            continue;
        }
        if ( iter->second->good_ == false )
        {
            // TODO try triangulate this map point
        }
        iter++;
    }

    if ( match_2dkp_index_.size()<100 )
        addMapPoints();
    if ( map_->map_points_.size() > 1000 )
    {
        // TODO map is too large, remove some one
        map_point_erase_ratio_ += 0.05;
    }
    else
        map_point_erase_ratio_ = 0.1;
    cout<<"map points: "<<map_->map_points_.size()<<endl;
}
```

我们刻意留空了一些地方，请感兴趣的读者自行完成。例如，你可以使用三角化来更新特征点的世界坐标，或者考虑更好地动态管理地图规模的策略。这些问题都是开放性的。

3.特征匹配代码。匹配之前，我们从地图中拿出一些候选点（出现在视野内的点），然后将它们与当前帧的特征描述子进行匹配。

书籍汇总下载 http://t.cn/RDIAj5D

```
void VisualOdometry::featureMatching()
{
    boost::timer timer;
    vector<cv::DMatch> matches;
    // select the candidates in map
    Mat desp_map;
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
    vector<MapPoint::Ptr> candidate;
    for ( auto& allpoints: map_->map_points_ )
    {
        MapPoint::Ptr& p = allpoints.second;
        // check if p in curr frame image
        if ( curr_->isInFrame(p->pos_) )
        {
            // add to candidate
            p->visible_times_++;
            candidate.push_back( p );
            desp_map.push_back( p->descriptor_ );
        }
    }

    matcher_flann_.match ( desp_map, descriptors_curr_, matches );
    // select the best matches
    float min_dis = std::min_element (
        matches.begin(), matches.end(),
        [] ( const cv::DMatch& m1, const cv::DMatch& m2 )
        {
            return m1.distance < m2.distance;
        } )->distance;

    match_3dpts_.clear();
    match_2dkp_index_.clear();
    for ( cv::DMatch& m : matches )
    {
        if ( m.distance < max<float> ( min_dis*match_ratio_, 30.0 ) )
        {
            match_3dpts_.push_back( candidate[m.queryIdx] );
            match_2dkp_index_.push_back( m.trainIdx );
        }
    }
    cout<<"good matches: "<<match_3dpts_.size() <<endl;
    cout<<"match cost time: "<<timer.elapsed() <<endl;
}
```

除了现有的地图之外，我们还引入了“关键帧”（Key-frame）的概

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

念。关键帧在许多视觉SLAM中都会用到，不过这个概念主要是给后端用的，所以我们在后面几讲再讨论对关键帧的详细处理。在实践中，我们肯定不希望对每个图像都做详细的优化和回环检测，那样毕竟太耗费资源。至少相机搁在原地不动时，我们不希望整个模型（地图也好、轨迹也好）变得越来越大。因此，后端优化的主要对象就是关键帧。

关键帧是相机运动过程当中某几个特殊的帧，这里“特殊”的意义是可以由我们自己指定的。常见的做法时，每当相机运动经过一定间隔，就取一个新的关键帧并保存起来[3]。这些关键帧的位姿将被仔细优化，而位于两个关键帧之间的那些东西，除了对地图贡献一些地图点外，就被理所当然地忽略掉了。

本节的实现也会提取一些关键帧，为后端的优化做一些数据上的准备。现在，读者可以编译这个工程，看看它的运行结果。本节的例程会把局部地图的点投影到图像平面并显示出来。如果位姿估计正确，它们看起来应该像是固定在空间中一样。反之，如果你感觉到某个特征点不自然地运动，那可能是相机位姿估计不够准确，或特征点的位置不够准确。

我们在0.4版没有提供对地图的优化，建议读者自行尝试一下。用到的原理主要是最小二乘和三角化，在前两讲已介绍过，不会太困难。

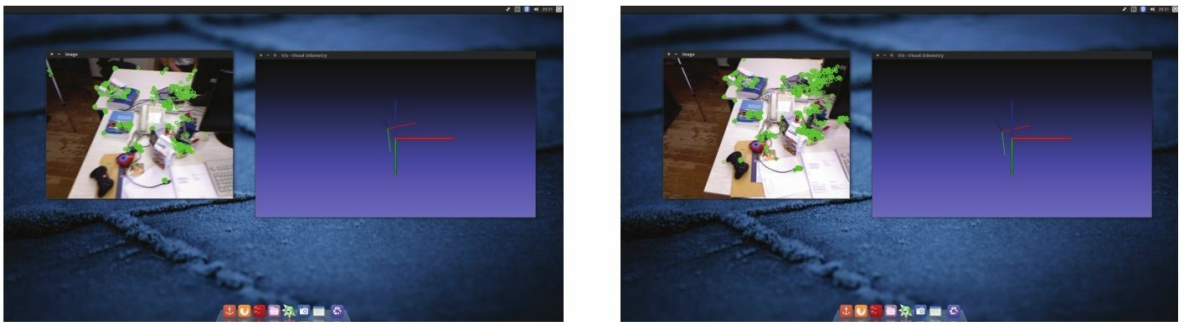


图9-7　0.4版VO的运行截图，在两个不同时刻标记了路标投影点。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

9.5 小结

作为实践，本讲带领读者从零开始实现了一个简单的视觉里程计，为的是让读者对前面两讲介绍的算法有一个经验上的认识。如果没有本讲，你很难亲身体会到例如“特征点的VO大约能够实时处理多少个ORB特征点”这样的问题[4]。我们看到，视觉里程计能够估算局部时间内的相机运动及特征点的位置，但是这种局部的方式有明显的缺点：

1.容易丢失。一旦丢失，我们要么“等待相机转回来”（保存参考帧并与新的帧比较），要么重置整个VO以跟踪新的图像数据。

2.轨迹漂移。主要原因是每次估计的误差会累积至下一次估计，导致长时间轨迹不准确。大一点的局部地图可以缓解这种现象，但它始终是存在的。

值得一提的是，如果只关心短时间内的运动，或者VO的精度已经满足应用需求，那么有时候你可能需要的仅仅就是一个视觉里程计，而不用完全的SLAM。比如，某些无人机控制或AR游戏的应用中，用户并不需要一个全局一致的地图，那么轻便的VO可能是更好的选择。不过，本书的目标是介绍整个SLAM，所以我们还要走得更远一些，看看后端和回环检测是如何工作的。

习题

1.本书使用的C++技巧你都看懂了吗？如果有不明白的地方，使用搜索引擎补习相关的知识，包括：基于范围的for循环、lambda表达式、智能指针、设计模式中的单例模式，等等。

2.在0.3版或0.4版的基础上，添加对地图进行优化的代码。或者，也可以根据PnP结果做一下三角化，消除RGB-D深度值的误差。

3.观察本讲代码是如何处理误匹配的。什么是RANSAC？阅读文献[61]或搜索相关资料进行了解。

---

[1]左下是笔者的练习作品。右下来自Epicwork团队作品：《圆明园》。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

[2]当然，从C++设计角度来说，保留之前的方式并使用继承会更有效地复用现有代码。

[3]这在李代数上很容易实现，请想想怎么实现。

[4]当然这取决于你的机器的性能。

书籍汇总下载 http://t.cn/RDIAj5D

第10讲　后端1

主要目标

1.理解后端的概念。

2.理解以EKF为代表的滤波器后端工作原理。

3.理解非线性优化的后端，明白稀疏性是如何利用的。

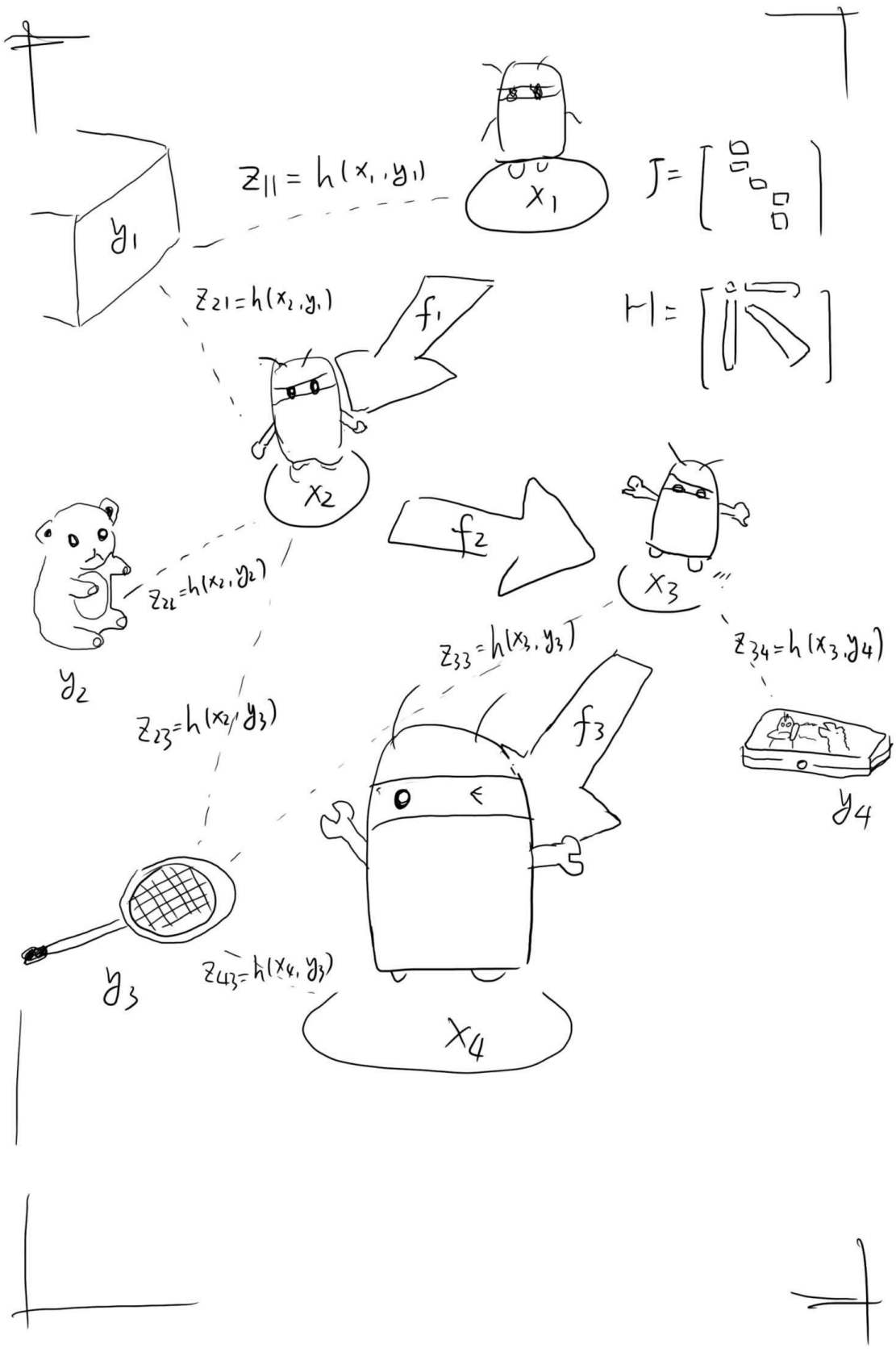
4.使用g2o和Ceres实际操作后端优化。

本讲开始，我们转入SLAM系统的另一个重要模块：后端优化。

我们看到，前端视觉里程计能给出一个短时间内的轨迹和地图，但由于不可避免的误差累积，这个地图在长时间内是不准确的。所以，在视觉里程计的基础上，我们还希望构建一个尺度、规模更大的优化问题，以考虑长时间内的最优轨迹和地图。不过，考虑到精度与性能的平衡，实际当中存在着许多不同的做法。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D



电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

10.1 概述

10.1.1 状态估计的概率解释

第2讲中提到，视觉里程计只有短暂的记忆，而我们希望整个运动轨迹在较长时间内都能保持最优的状态。我们可能会用最新的知识，更新较久远的状态——站在“久远的状态”的角度上看，仿佛是未来的信息告诉它“你应该在哪里”。所以，在后端优化中，我们通常考虑一段更长时间内（或所有时间内）的状态估计问题，而且不仅使用过去的信息更新自己的状态，也会用未来的信息来更新自己，这种处理方式不妨称为“批量的”（Batch）。否则，如果当前的状态只由过去的时刻决定，甚至只由前一个时刻决定，那不妨称为“渐进的”（Incremental）。

我们已经知道SLAM过程可以由运动方程和观测方程来描述。那么，假设在t=0到t=N的时间内，有位姿x_0到x_N，并且有路标y_1,⋯,y_M。按照之前的写法，运动和观测方程为

$$\begin{cases} \boldsymbol{x}_k = f\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k\right) + \boldsymbol{w}_k \\ \boldsymbol{z}_{k,j} = h\left(\boldsymbol{y}_j, \boldsymbol{x}_k\right) + \boldsymbol{v}_{k,j} \end{cases} \quad k = 1, \ldots, N,\ j = 1, \ldots, M. \tag{10.1}$$

注意以下几点：

1.观测方程中，只有当x_k看到了y_j时，才会产生观测数据，否则就没有。事实上，在一个位置通常只能看到一小部分路标。而且，由于视觉SLAM特征点数量众多，所以实际当中观测方程的数量会远远大于运动方程。

2.我们可能没有测量运动的装置，所以也可能没有运动方程。在这个情况下，有若干种处理方式：认为确实没有运动方程，或假设相机不动，或假设相机匀速运动。这几种方式都是可行的。在没有运动方程的情况下，整个优化问题就只由许多个观测方程组成。这就非常类似于SfM（Structure from Motion）问题，相当于我们通过一组图像来恢复运

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

动和结构。与SfM中不同的是，SLAM中的图像有时间上的先后顺序，而SfM中允许使用完全无关的图像。

我们知道每个方程都受噪声影响，所以要把这里的位姿x和路标y看成服从某种概率分布的随机变量，而不是单独的一个数。因此，我们关心的问题就变成：当我拥有某些运动数据u和观测数据z时，如何来确定状态量x,y的分布？进而，如果得到了新时刻的数据，那么它们的分布又将发生怎样的变化？在比较常见且合理的情况下，我们假设状态量和噪声项服从高斯分布——这意味着在程序中只需要储存它们的均值和协方差矩阵即可。均值可看作是对变量最优值的估计，而协方差矩阵则度量了它的不确定性。那么，问题就转变为：当存在一些运动数据和观测数据时，我们如何去估计状态量的高斯分布？

我们依然设身处地地扮演一下小萝卜。只有运动方程时，相当于我们蒙着眼睛在一个未知的地方走路。尽管我们知道自己每一步走了多远，但是随着时间增长，我们将对自己的位置越来越不确定——内心也就越不安。这说明在输入数据受噪声影响时，我们对位置方差的估计将越来越大。但是，当我们睁开眼睛时，由于能够不断地观测到外部场景，使得位置估计的不确定性变小，我们就会越来越自信。如果用椭圆或椭球直观地表达协方差阵，那么这个过程有点像是在手机地图软件中走路的感觉。以图10-1为例，读者可以想象，当没有观测数据时，这个圆会随着运动越来越大；而如果有正确的观测，圆就会缩小至一定的大小，保持稳定。

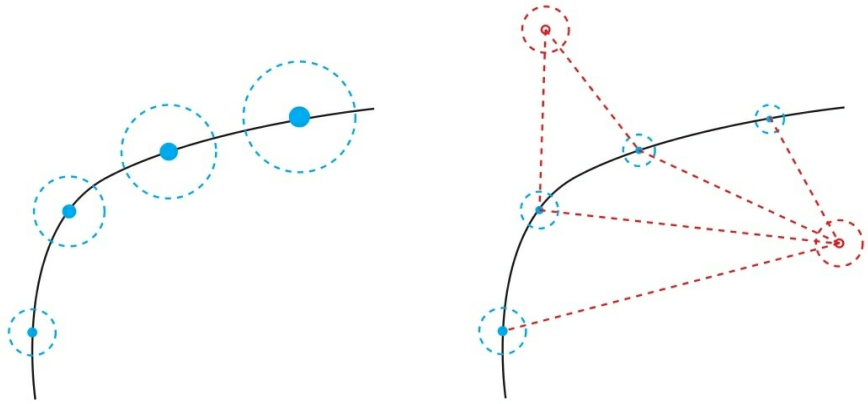


图10-1　不确定性的直观描述。左侧：只有运动方程时，由于下一时刻的位姿是在上一时刻基础上添加了噪声，所以不确定性越来越大。右侧：存在路标点时，不确定性会明显减小。不过请注意，这只是一个直观的示意图，并非实际

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

数据。

上面的过程以比喻的形式解释了状态估计中的问题，下面我们要以定量的方式来看待它。在第6讲中，我们介绍了最大似然估计，提到把状态估计转换为最小二乘的做法。在本讲我们要更仔细地讨论这个问题。首先，由于位姿和路标点都是待估计的变量，我们改变一下记号，令x_k为k时刻的所有未知量。它包含了当前时刻的相机位姿与m个路标点。在这种记号的意义下（虽然与之前稍有不同，但含义是清楚的），写成：

$$\boldsymbol{x}_k \triangleq \{\boldsymbol{x}_k, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_m\}. \tag{10.2}$$

同时，把k时刻的所有观测记作z_k。于是，运动方程与观测方程的形式可写得更加简洁。这里不会出现y，但我们心里要明白这时x中已经包含了之前的y：

$$\begin{cases} \boldsymbol{x}_k = f\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k\right) + \boldsymbol{w}_k \\ \boldsymbol{z}_k = h\left(\boldsymbol{x}_k\right) + \boldsymbol{v}_k \end{cases} \quad k = 1, \ldots, N. \tag{10.3}$$

现在考虑第k时刻的情况。我们希望用过去0到k中的数据来估计现在的状态分布：

$$P(\boldsymbol{x}_k | \boldsymbol{x}_0, \boldsymbol{u}_{1:k}, \boldsymbol{z}_{1:k}). \tag{10.4}$$

下标0:k表示从0时刻到k时刻的所有数据。请注意，z_k表示所有在k时刻的观测数据，它可能不止一个，只是这种记法更加方便。

下面我们来看如何对状态进行估计。按照贝叶斯法则，把z_k与x_k交换位置，有：

$$P\left(\boldsymbol{x}_k | \boldsymbol{x}_0, \boldsymbol{u}_{1:k}, \boldsymbol{z}_{1:k}\right) \propto P\left(\boldsymbol{z}_k | \boldsymbol{x}_k\right) P\left(\boldsymbol{x}_k | \boldsymbol{x}_0, \boldsymbol{u}_{1:k}, \boldsymbol{z}_{1:k-1}\right). \tag{10.5}$$

读者应该不会感到陌生。这里第一项称为似然，第二项称为先验。似然由观测方程给定，而先验部分，我们要明白当前状态x_k是基于过去

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

所有的状态估计得来的。至少，它会受x_{k-1}影响，于是按照x_{k-1}时刻为条件概率展开：

$$P(\boldsymbol{x}_k|\boldsymbol{x}_0,\boldsymbol{u}_{1:k},\boldsymbol{z}_{1:k-1}) = \int P(\boldsymbol{x}_k|\boldsymbol{x}_{k-1},\boldsymbol{x}_0,\boldsymbol{u}_{1:k},\boldsymbol{z}_{1:k-1})\,P(\boldsymbol{x}_{k-1}|\boldsymbol{x}_0,\boldsymbol{u}_{1:k},\boldsymbol{z}_{1:k-1})\,\mathrm{d}\boldsymbol{x}_{k-1}. \tag{10.6}$$

如果考虑更久之前的状态，也可以继续对此式进行展开，但现在我们只关心k时刻和k-1时刻的情况。至此，我们给出了贝叶斯估计，虽然上式还没有具体的概率分布形式，所以还没法实际地操作它。对这一步的后续处理，方法上产生了一些分歧。大体上讲，存在若干种选择：其一是假设马尔可夫性，简单的一阶马氏性认为，k时刻状态只与k-1时刻状态有关，而与再之前的无关。如果做出这样的假设，我们就会得到以扩展卡尔曼滤波（EKF）为代表的滤波器方法。在滤波方法中，我们会从某时刻的状态估计，推导到下一个时刻。另外一种方法是依然考虑k时刻状态与之前所有状态的关系，此时将得到非线性优化为主体的优化框架。非线性优化的基本知识已在前文介绍过。目前视觉SLAM的主流为非线性优化方法。不过，为了让本书更全面，我们要先介绍一下卡尔曼滤波器和EKF的原理。

10.1.2 线性系统和KF

我们首先来看滤波器模型。当我们假设了马尔可夫性，从数学角度会发生哪些变化呢？首先，当前时刻状态只和上一个时刻有关，式(10.6)中右侧第一部分可进一步简化：

$$P(\boldsymbol{x}_k|\boldsymbol{x}_{k-1},\boldsymbol{x}_0,\boldsymbol{u}_{1:k},\boldsymbol{z}_{1:k-1}) = P(\boldsymbol{x}_k|\boldsymbol{x}_{k-1},\boldsymbol{u}_k)\,. \tag{10.7}$$

这里，由于k时刻状态与k-1之前的无关，所以就简化成只与x_{k-1}和u_k有关的形式，与k时刻的运动方程对应。第二部分可简化为

$$P(\boldsymbol{x}_{k-1}|\boldsymbol{x}_0,\boldsymbol{u}_{1:k},\boldsymbol{z}_{1:k-1}) = P(\boldsymbol{x}_{k-1}|\boldsymbol{x}_0,\boldsymbol{u}_{1:k-1},\boldsymbol{z}_{1:k-1})\,. \tag{10.8}$$

这是考虑到k时刻的输入量u_k与k-1时刻的状态无关，所以我们把u_k拿掉。可以看到，这一项实际上是k-1时刻的状态分布。于是，这一系

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

列方程说明，我们实际在做的是“如何把k-1时刻的状态分布推导至k时刻”这样一件事。也就是说，在程序运行期间，我们只要维护一个状态量，对它不断地进行迭代和更新即可。进一步，如果假设状态量服从高斯分布，那么我们只需考虑维护状态量的均值和协方差即可。

我们从形式最简单的线性高斯系统开始，最后会得到卡尔曼滤波器。线性高斯系统是说，运动方程和观测方程可以由线性方程来描述：

$$\begin{cases} \boldsymbol{x}_k = \boldsymbol{A}_k \boldsymbol{x}_{k-1} + \boldsymbol{u}_k + \boldsymbol{w}_k \\ \boldsymbol{z}_k = \boldsymbol{C}_k \boldsymbol{x}_k + \boldsymbol{v}_k \end{cases} \quad k = 1, \dots, N. \tag{10.9}$$

并假设所有的状态和噪声均满足高斯分布。记这里的噪声服从零均值高斯分布：

$$\boldsymbol{w}_k \sim N(\boldsymbol{0}, \boldsymbol{R}). \quad \boldsymbol{v}_k \sim N(\boldsymbol{0}, \boldsymbol{Q}). \tag{10.10}$$

为了简洁，笔者省略了R和Q的下标。现在，利用马尔可夫性，假设我们知道了k-1时刻的后验（在k-1时刻看来）状态估计$\hat{\boldsymbol{x}}_{k-1}$及其协方差$\hat{\boldsymbol{P}}_{k-1}$，现在要根据$k$时刻的输入和观测数据，确定$x_k$的后验分布。为区分推导中的先验和后验，我们在记号上做一点区别：以尖帽子$\hat{\boldsymbol{x}}_k$表示后验，以横线$\bar{\boldsymbol{x}}$表示先验分布，请读者不要混淆。

卡尔曼滤波器的第一步，通过运动方程确定x_k的先验分布。根据高斯分布的性质，显然有：

$$P(\boldsymbol{x}_k | \boldsymbol{x}_0, \boldsymbol{u}_{1:k}, \boldsymbol{z}_{1:k-1}) = N\left(\boldsymbol{A}_k \hat{\boldsymbol{x}}_{k-1} + \boldsymbol{u}_k, \boldsymbol{A}_k \hat{\boldsymbol{P}}_k \boldsymbol{A}_k^{\mathrm{T}} + \boldsymbol{R}\right). \tag{10.11}$$

这一步称为预测，原理见附录A.3。它显示了如何从上一个时刻的状态，根据输入信息（但是有噪声）推断当前时刻的状态分布。这个分布也就是先验。记：

$$\bar{\boldsymbol{x}}_k = \boldsymbol{A}_k \hat{\boldsymbol{x}}_{k-1} + \boldsymbol{u}_k, \quad \bar{\boldsymbol{P}}_k = \boldsymbol{A}_k \hat{\boldsymbol{P}}_{k-1} \boldsymbol{A}_k^{\mathrm{T}} + \boldsymbol{R}. \tag{10.12}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

这非常自然。另一方面，由观测方程，我们可以计算在某个状态下应该产生怎样的观测数据：

$$P(\boldsymbol{z}_k|\boldsymbol{x}_k) = N(\boldsymbol{C}_k\boldsymbol{x}_k, \boldsymbol{R}). \tag{10.13}$$

为了得到后验概率，我们想要计算它们的乘积，也就是由式(10.5)给出的贝叶斯公式。然而，虽然我们知道最后会得到一个关于x_k的高斯分布，但计算上是有一点点麻烦的，我们先把结果设为$x_k \sim N(\hat{x}_k, \hat{P}_k)$，那么：

$$N(\hat{\boldsymbol{x}}_k, \hat{\boldsymbol{P}}_k) = N(\boldsymbol{C}_k\boldsymbol{x}_k, \boldsymbol{R}) \cdot N(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{P}}_k). \tag{10.14}$$

这里我们稍微用点讨巧的方法。既然我们已经知道等式两侧都是高斯分布，那就只需比较指数部分即可，而无须理会高斯分布前面的因子部分。指数部分很像是一个二次型的配方，我们来推导一下。首先把指数部分展开，有[1]：

$$(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k)^{\mathrm{T}} \hat{\boldsymbol{P}}_k^{-1} (\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k) = (\boldsymbol{z}_k - \boldsymbol{C}_k\boldsymbol{x}_k)^{\mathrm{T}} \boldsymbol{R}^{-1} (\boldsymbol{z}_k - \boldsymbol{C}_k\boldsymbol{x}_k) + (\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k)^{\mathrm{T}} \bar{\boldsymbol{P}}_k^{-1} (\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k). \tag{10.15}$$

为了求左侧的$\hat{\boldsymbol{x}}_k$和$\hat{\boldsymbol{P}}_k$，我们把两边展开，并比较x_k的二次和一次系数。对于二次系数，有：

$$\hat{\boldsymbol{P}}_k^{-1} = \boldsymbol{C}_k^{\mathrm{T}} \boldsymbol{R}^{-1} \boldsymbol{C}_k + \bar{\boldsymbol{P}}_k^{-1}. \tag{10.16}$$

该式给出了协方差的计算过程。为了便于后面列写式子，定义一个中间变量：

$$\boldsymbol{K} = \hat{\boldsymbol{P}}_k \boldsymbol{C}_k^{\mathrm{T}} \boldsymbol{R}^{-1}. \tag{10.17}$$

根据此定义，在式(10.16)的左右各乘$\hat{\boldsymbol{P}}_k$，有：

$$\boldsymbol{I} = \hat{\boldsymbol{P}}_k \boldsymbol{C}_k^{\mathrm{T}} \boldsymbol{R}^{-1} \boldsymbol{C}_k + \hat{\boldsymbol{P}}_k \bar{\boldsymbol{P}}_k^{-1} = \boldsymbol{K}\boldsymbol{C}_k + \hat{\boldsymbol{P}}_k \bar{\boldsymbol{P}}_k^{-1}. \tag{10.18}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

于是有[2]：

$$\hat{\boldsymbol{P}}_k = (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{C}_k)\bar{\boldsymbol{P}}_k. \tag{10.19}$$

然后再比较一次项的系数，有：

$$-2\hat{\boldsymbol{x}}_k^{\mathrm{T}}\hat{\boldsymbol{P}}_k^{-1}\boldsymbol{x}_k = -2\boldsymbol{z}_k^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{C}_k\boldsymbol{x}_k - 2\bar{\boldsymbol{x}}_k^{\mathrm{T}}\bar{\boldsymbol{P}}_k^{-1}\boldsymbol{x}_k. \tag{10.20}$$

整理（取系数并转置）得：

$$\hat{\boldsymbol{P}}_k^{-1}\hat{\boldsymbol{x}}_k = \boldsymbol{C}_k^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{z}_k + \bar{\boldsymbol{P}}_k^{-1}\bar{\boldsymbol{x}}_k. \tag{10.21}$$

两侧乘以$\hat{\boldsymbol{P}}_k$并代入式(10.17)，得：

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{P}}_k\boldsymbol{C}_k^{\mathrm{T}}\boldsymbol{R}^{-1}\boldsymbol{z}_k + \hat{\boldsymbol{P}}_k\bar{\boldsymbol{P}}_k^{-1}\bar{\boldsymbol{x}}_k \tag{10.22}$$

$$= \boldsymbol{K}\boldsymbol{z}_k + (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{C}_k)\,\bar{\boldsymbol{x}}_k = \bar{\boldsymbol{x}}_k + \boldsymbol{K}\left(\boldsymbol{z}_k - \boldsymbol{C}_k\bar{\boldsymbol{x}}_k\right). \tag{10.23}$$

于是我们又得到了后验均值的表达。总而言之，上面的两个步骤可以归纳为“预测”（Predict）和“更新”（Update）两个步骤：

1.预测：

$$\bar{\boldsymbol{x}}_k = \boldsymbol{A}_k\hat{\boldsymbol{x}}_{k-1} + \boldsymbol{u}_k, \quad \bar{\boldsymbol{P}}_k = \boldsymbol{A}_k\hat{\boldsymbol{P}}_{k-1}\boldsymbol{A}_k^{\mathrm{T}} + \boldsymbol{R}. \tag{10.24}$$

2.更新：先计算K，它又称为卡尔曼增益。

$$\boldsymbol{K} = \bar{\boldsymbol{P}}_k\boldsymbol{C}_k^{\mathrm{T}}\left(\boldsymbol{C}_k\bar{\boldsymbol{P}}_k\boldsymbol{C}_k^{\mathrm{T}} + \boldsymbol{R}_k\right)^{-1}. \tag{10.25}$$

然后计算后验概率的分布。

$$\begin{aligned}\hat{\boldsymbol{x}}_k &= \bar{\boldsymbol{x}}_k + \boldsymbol{K}\left(\boldsymbol{z}_k - \boldsymbol{C}_k\bar{\boldsymbol{x}}_k\right)\\ \hat{\boldsymbol{P}}_k &= \left(\boldsymbol{I} - \boldsymbol{K}\boldsymbol{C}_k\right)\bar{\boldsymbol{P}}_k.\end{aligned}. \tag{10.26}$$

至此，我们推导了经典的卡尔曼滤波器的整个过程。事实上，卡尔曼滤波器有若干种推导方式，而我们使用的是从概率角度出发的最大后验概率估计的形式。我们看到，在线性高斯系统中，卡尔曼滤波器构成了该系统中的最大后验概率估计。而且，由于高斯分布经过线性变换后

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

仍服从高斯分布，所以整个过程中我们没有进行任何的近似。可以说，卡尔曼滤波器构成了线性系统的最优无偏估计。

10.1.3 非线性系统和EKF

在理解了卡尔曼滤波之后，我们必须要澄清一点：SLAM中的运动方程和观测方程通常是非线性函数，尤其是视觉SLAM中的相机模型，需要使用相机内参模型及李代数表示的位姿，更不可能是一个线性系统。一个高斯分布，经过非线性变换后，往往不再是高斯分布，所以在非线性系统中，我们必须取一定的近似，将一个非高斯分布近似成高斯分布。

我们希望把卡尔曼滤波器的结果拓展到非线性系统中，称为扩展卡尔曼滤波器（Extended Kalman Filter，EKF）。通常的做法是，在某个点附近考虑运动方程及观测方程的一阶泰勒展开，只保留一阶项，即线性的部分，然后按照线性系统进行推导。令k-1时刻的均值与协方差矩阵为$\hat{\boldsymbol{x}}_{k-1}, \hat{\boldsymbol{P}}_{k-1}$。在$k$时刻，我们把运动方程和观测方程在$\hat{\boldsymbol{x}}_{k-1}, \hat{\boldsymbol{P}}_{k-1}$处进行线性化（相当于一阶泰勒展开），有：

$$\boldsymbol{x}_k \approx f\left(\hat{\boldsymbol{x}}_{k-1}, \boldsymbol{u}_k\right) + \left.\frac{\partial f}{\partial \boldsymbol{x}_{k-1}}\right|_{\hat{\boldsymbol{x}}_{k-1}} \left(\boldsymbol{x}_{k-1} - \hat{\boldsymbol{x}}_{k-1}\right) + \boldsymbol{w}_k. \tag{10.27}$$

记这里的偏导数为

$$\boldsymbol{F} = \left.\frac{\partial f}{\partial \boldsymbol{x}_{k-1}}\right|_{\hat{\boldsymbol{x}}_{k-1}}. \tag{10.28}$$

同样，对于观测方程，亦有：

$$\boldsymbol{z}_k \approx h\left(\bar{\boldsymbol{x}}_k\right) + \left.\frac{\partial h}{\partial \boldsymbol{x}_k}\right|_{\bar{\boldsymbol{x}}_k} \left(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k\right) + \boldsymbol{n}_k. \tag{10.29}$$

记这里的偏导数为

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$\boldsymbol{H} = \left.\frac{\partial h}{\partial \boldsymbol{x}_k}\right|_{\bar{\boldsymbol{x}}_k}. \tag{10.30}$$

那么，在预测步骤中，根据运动方程有：

$$P\left(\boldsymbol{x}_k|\boldsymbol{x}_0, \boldsymbol{u}_{1:k}, \boldsymbol{z}_{0:k-1}\right) = N(f\left(\hat{\boldsymbol{x}}_{k-1}, \boldsymbol{u}_k\right), \boldsymbol{F}\hat{\boldsymbol{P}}_{k-1}\boldsymbol{F}^{\mathrm{T}} + \boldsymbol{R}_k). \tag{10.31}$$

这些推导和卡尔曼滤波是十分相似的。为方便表述，记这里的先验和协方差的均值为

$$\bar{\boldsymbol{x}}_k = f\left(\hat{\boldsymbol{x}}_{k-1}, \boldsymbol{u}_k\right), \quad \bar{\boldsymbol{P}}_k = \boldsymbol{F}\hat{\boldsymbol{P}}_k\boldsymbol{F}^{\mathrm{T}} + \boldsymbol{R}_k. \tag{10.32}$$

然后，考虑在观测中我们有：

$$P\left(\boldsymbol{z}_k|\boldsymbol{x}_k\right) = N(h\left(\bar{\boldsymbol{x}}_k\right) + \boldsymbol{H}\left(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k\right), \boldsymbol{Q}_k). \tag{10.33}$$

最后，根据最开始的贝叶斯展开式，可以推导出x_k的后验概率形式。我们略去中间的推导过程，只介绍其结果。读者可以仿照卡尔曼滤波器的方式，推导EKF的预测与更新方程。简而言之，我们会先定义一个卡尔曼增益K_k：

$$\boldsymbol{K}_k = \bar{\boldsymbol{P}}_k\boldsymbol{H}^{\mathrm{T}}\left(\boldsymbol{H}\bar{\boldsymbol{P}}_k\boldsymbol{H}^{\mathrm{T}} + \boldsymbol{Q}_k\right)^{-1}. \tag{10.34}$$

在卡尔曼增益的基础上，后验概率的形式为

$$\hat{\boldsymbol{x}}_k = \bar{\boldsymbol{x}}_k + \boldsymbol{K}_k\left(\boldsymbol{z}_k - h\left(\bar{\boldsymbol{x}}_k\right)\right), \hat{\boldsymbol{P}}_k = \left(\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}\right)\bar{\boldsymbol{P}}_k. \tag{10.35}$$

卡尔曼滤波器给出了在线性化之后状态变量分布的变化过程。在线性系统和高斯噪声下，卡尔曼滤波器给出了无偏最优估计。而在SLAM这种非线性的情况下，它给出了单次线性近似下的最大后验估计（MAP）。

10.1.4 EKF的讨论

EKF以形式简洁、应用广泛著称。当想要在某段时间内估计某个不

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

确定量时，我们首先想到的就是EKF。在早期的SLAM中，EKF占据了很长一段时间的主导地位，研究者们讨论了各种各样的滤波器在SLAM中的应用，如IF（信息滤波器）[62]、IKF[63]（Iterated KF）、UKF[64]（Unscented KF）和粒子滤波器[65,66,67]、SWF（Sliding Window Filter）[68]，等等[17][3]，或者用分治法等思路改进EKF的效率[69,70]。直至今日，尽管我们认识到非线性优化比滤波器占有明显的优势，但是在计算资源受限，或待估计量比较简单的场合，EKF仍不失为一种有效的方式。

EKF有哪些局限呢？

1.首先，滤波器方法在一定程度上假设了马尔可夫性，也就是k时刻的状态只与

k-1时刻相关，而与k-1之前的状态和观测都无关（或者和前几个有限时刻的状态相关）。这有点像是在视觉里程计中只考虑相邻两帧关系一样。如果当前帧确实与很久之前的数据有关（例如回环），那么滤波器就会难以处理。

而非线性优化方法则倾向于使用所有的历史数据。它不光考虑邻近时刻的特征点与轨迹关系，更会把很久之前的状态也考虑进来，称为全体时间上的SLAM（Full-SLAM）。在这种意义下，非线性优化方法使用了更多信息，当然也需要更多的计算。

2.与第6讲介绍的优化方法相比，EKF滤波器仅在$\hat{x}_{k-1}$处做了一次线性化，然后就直接根据这次线性化结果，把后验概率给算了出来。这相当于在说，我们认为该点处的线性化近似在后验概率处仍然是有效的。而实际上，当我们离开工作点较远时，一阶泰勒展开并不一定能够近似整个函数，这取决于运动模型和观测模型的非线性情况。如果它们有强烈的非线性，那么线性近似就只在很小范围内成立，不能认为在很远的地方仍能用线性来近似。这就是EKF的非线性误差，也是它的主要问题所在。

在优化问题中，尽管我们也做一阶（最速下降）或二阶（高斯牛顿法或列文伯格—马夸尔特方法）的近似，但每迭代一次，状态估计发生改变之后，我们会重新对新的估计点做泰勒展开，而不像EKF那样只在固定点上做一次泰勒展开。这就使得优化方法适用范围更广，在状态变

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

化较大时亦能适用。

3.从程序实现上来说，EKF需要存储状态量的均值和方差，并对它们进行维护和更新。如果把路标也放进状态，由于视觉SLAM中路标数量很大，这个存储量是相当可观的，且与状态量呈平方增长（因为要存储协方差矩阵）。因此，EKF SLAM被普遍认为不适用于大型场景。

由于EKF存在这些明显的缺点，我们通常认为，在同等计算量的情况下，非线性优化能取得更好的效果[13]。下面我们来讨论以非线性优化为主的后端。我们将主要介绍图优化，并用g2o和Ceres体验一下后端的例子。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

10.2 BA与图优化

如果你做过视觉三维重建，那么应该对这个概念再熟悉不过了。所谓的Bundle Adjustment[4]，是指从视觉重建中提炼出最优的3D模型和相机参数（内参数和外参数）。从每一个特征点反射出来的几束光线（bundles of light rays），在我们把相机姿态和特征点空间位置做出最优的调整（adjustment）之后，最后收束到相机光心的这个过程[26]，简称为BA。

在图优化框架的视觉SLAM算法里，BA起到了核心作用。它类似于求解只有观测方程的SLAM问题。在最近几年视觉SLAM理论的研究中，BA算法不仅具有很高的精度，也开始具备良好的实时性，能够应用于在线计算的SLAM场景中。而在21世纪早期，虽然计算机视觉领域的研究者们已经开始利用BA进行重构，但SLAM的研究者们通常认为包含大量特征点和相机位姿的BA计算量过大，不适合实时计算。直到近十年，人们逐渐认识到SLAM问题中BA的稀疏特性，才使它能够在实时的场景中应用[9,24]。因此，掌握好Bundle Adjustment，深入研究其中的理论和实践细节，是做好视觉SLAM的关键。

本节结合第5讲和第6讲的内容，介绍Bundle Adjustment的模型，并且和读者逐步探讨求解过程，特别是探讨它的稀疏性，然后介绍一种通用的快速求解方法。

10.2.1 投影模型和BA代价函数

在第5讲里，我们曾介绍了投影模型和畸变，让我们复习一下整个投影的过程。从一个世界坐标系中的点p出发，把相机的内外参数和畸变都考虑进来，最后投影成像素坐标，需要如下步骤。

1.首先，把世界坐标转换到相机坐标，这里将用到相机外参数(R,t)：

$$\boldsymbol{P}' = \boldsymbol{R}\boldsymbol{p} + \boldsymbol{t} = [X', Y', Z']^{\mathrm{T}}. \tag{10.36}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

2.然后，将P'投至归一化平面，得到归一化坐标：

$$\boldsymbol{P}_c = [u_c, v_c, 1]^{\mathrm{T}} = [X'/Z', Y'/Z', 1]^{\mathrm{T}}. \tag{10.37}$$

3.考虑归一化坐标的畸变情况，得到去畸变前的原始像素坐标。这里暂时只考虑径向畸变：

$$\begin{cases} u'_c = u_c\left(1 + k_1 r_c^2 + k_2 r_c^4\right) \\ v'_c = v_c\left(1 + k_1 r_c^2 + k_2 r_c^4\right) \end{cases}. \tag{10.38}$$

4.最后，根据内参模型，计算像素坐标：

$$\begin{cases} u_s = f_x u'_c + c_x \\ v_s = f_y v'_c + c_y \end{cases}. \tag{10.39}$$

这一系列计算流程看似有些复杂。我们用流程图10-2形象化地表示整个过程，以帮助读者理解。读者应该能领会到，这个过程也就是前面讲的观测方程，之前我们把它抽象地记成：

$$\boldsymbol{z} = h(\boldsymbol{x}, \boldsymbol{y}). \tag{10.40}$$

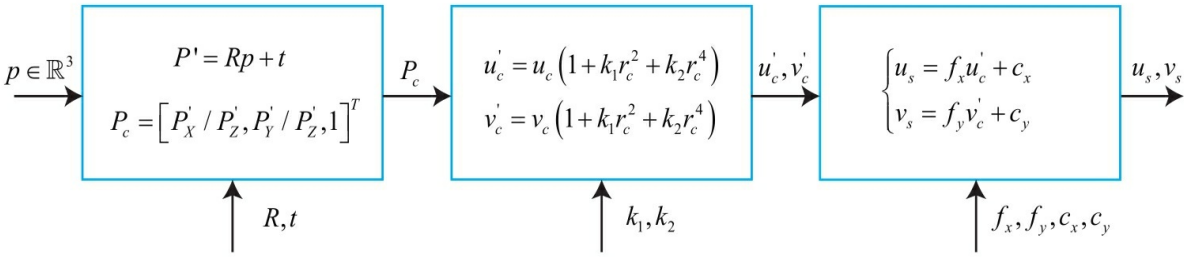


图10-2　计算流程示意图。左侧的p是全局坐标系下的三维坐标点，右侧的u_s,v_s是该点在图像平面上的最终像素坐标。中间畸变模块中的$r_c^2=u_c^2+v_c^2$。

现在，我们给出了它的详细参数化过程。具体地说，这里的x指代此时相机的位姿，即外参R,t，它对应的李代数为ξ。路标y即这里的三维点p，而观测数据则是像素坐标$z^{\Delta}=[u_s,v_s]^T$。以最小二乘的角度来考虑，那么可以列写关于此次观测的误差：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$e=z-h(\xi,p)$.(10.41)

然后，把其他时刻的观测量也考虑进来，我们可以给误差添加一个下标。设z_{ij}为在位姿ξ_i处观察路标p_j产生的数据，那么整体的代价函数（Cost Function）为

$$\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}\|\boldsymbol{e}_{ij}\|^2=\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}\|\boldsymbol{z}_{ij}-h(\boldsymbol{\xi}_i,\boldsymbol{p}_j)\|^2. \tag{10.42}$$

对这个最小二乘进行求解，相当于对位姿和路标同时做了调整，也就是所谓的BA。接下来，我们会根据该目标函数和第6讲介绍的非线性优化内容，逐步深入探讨该模型的求解。

10.2.2 BA的求解

观察在上一节中的观测模型$h(\xi,p)$，很容易判断该函数不是线性函数。所以我们希望使用6.2节介绍的一些非线性优化手段来优化它。根据非线性优化的思想，我们应该从某个初始值开始，不断地寻找下降方向Δx来找到目标函数的最优解，即不断地求解增量方程(6.22)中的增量Δx。尽管误差项都是针对单个位姿和路标点的，但在整体BA目标函数上，我们必须把自变量定义成所有待优化的变量：

$$\boldsymbol{x}=[\boldsymbol{\xi}_1,\ldots,\boldsymbol{\xi}_m,\boldsymbol{p}_1,\ldots,\boldsymbol{p}_n]^{\mathrm{T}}. \tag{10.43}$$

相应地，增量方程中的Δx则是对整体自变量的增量。在这个意义下，当我们给自变量一个增量时，目标函数变为

$$\frac{1}{2}\|f(\boldsymbol{x}+\Delta\boldsymbol{x})\|^2\approx\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}\|\boldsymbol{e}_{ij}+\boldsymbol{F}_{ij}\Delta\boldsymbol{\xi}_i+\boldsymbol{E}_{ij}\Delta\boldsymbol{p}_j\|^2. \tag{10.44}$$

其中F_{ij}表示整个代价函数在当前状态下对相机姿态的偏导数，而E_{ij}表示该函数对路标点位置的偏导。我们曾在7.7.3节中介绍了它们的具体形式，所以这里就不再展开推导了。现在，把相机位姿变量放在一起：

$$\boldsymbol{x}_c=[\boldsymbol{\xi}_1,\boldsymbol{\xi}_2,\ldots,\boldsymbol{\xi}_m]^{\mathrm{T}}\in\mathbb{R}^{6m}, \tag{10.45}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

并把空间点的变量也放在一起：

$$\boldsymbol{x}_p = [\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n]^{\mathrm{T}} \in \mathbb{R}^{3n}, \tag{10.46}$$

那么，式(10.44)可以简化表达如下：

$$\frac{1}{2}\left\|f(\boldsymbol{x}+\Delta\boldsymbol{x})\right\|^2 = \frac{1}{2}\left\|\boldsymbol{e}+\boldsymbol{F}\Delta\boldsymbol{x}_c+\boldsymbol{E}\Delta\boldsymbol{x}_p\right\|^2. \tag{10.47}$$

需要注意的是，该式从一个由很多个小型二次项之和，变成了一个更整体的样子。这里的雅可比矩阵$\boldsymbol{E}$和$\boldsymbol{F}$必须是整体目标函数对整体变量的导数，它将是一个很大块的矩阵，而里头每个小分块，需要由每个误差项的导数$\boldsymbol{F}_{ij}$和$\boldsymbol{E}_{ij}$“拼凑”起来。然后，无论我们使用高斯牛顿法还是列文伯格—马夸尔特方法，最后都将面对增量线性方程：

$$\boldsymbol{H}\Delta\boldsymbol{x} = \boldsymbol{g}. \tag{10.48}$$

根据第6讲的知识，我们知道高斯牛顿法和列文伯格—马夸尔特方法的主要差别在于，这里的$\boldsymbol{H}$是取$\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}$还是$\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}+\lambda\boldsymbol{I}$的形式。由于我们把变量归类成了位姿和空间点两种，所以雅可比矩阵可以分块为

$$\boldsymbol{J} = [\boldsymbol{F}\ \boldsymbol{E}]. \tag{10.49}$$

那么，以高斯牛顿法为例，则$\boldsymbol{H}$矩阵为

$$\boldsymbol{H} = \boldsymbol{J}^{\mathrm{T}}\boldsymbol{J} = \begin{bmatrix} \boldsymbol{F}^{\mathrm{T}}\boldsymbol{F} & \boldsymbol{F}^{\mathrm{T}}\boldsymbol{E} \\ \boldsymbol{E}^{\mathrm{T}}\boldsymbol{F} & \boldsymbol{E}^{\mathrm{T}}\boldsymbol{E} \end{bmatrix}. \tag{10.50}$$

当然在列文伯格—马夸尔特方法中我们也需要计算这个矩阵。不难发现，因为考虑了所有的优化变量，这个线性方程的维度将非常大，包含了所有的相机位姿和路标点。尤其是在视觉SLAM中，一幅图像就会提出数百个特征点，大大增加了这个线性方程的规模。如果直接对$\boldsymbol{H}$求逆来计算增量方程，由于矩阵求逆是复杂度为$O(n^3)$的操作[71]，这是非常消耗计算资源的。幸运的是，这里的$\boldsymbol{H}$矩阵是有一定的特殊结构的。利用这个特殊结构，我们可以加速求解过程。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

10.2.3 稀疏性和边缘化

21世纪视觉SLAM的一个重要进展是认识到了矩阵H的稀疏结构，并发现该结构可以自然、显式地用图优化来表示[28,72]。本节将详细讨论一下该矩阵稀疏结构。

H矩阵的稀疏性是由雅可比矩阵$J(x)$引起的。考虑这些代价函数当中的其中一个e_{ij}。注意到，这个误差项只描述了在ξ_i看到p_j这件事，只涉及第i个相机位姿和第j个路标点，对其余部分的变量的导数都为0。所以该误差项对应的雅可比矩阵有下面的形式：

$$\boldsymbol{J}_{ij}(\boldsymbol{x}) = \left(\boldsymbol{0}_{2\times 6}, \ldots \boldsymbol{0}_{2\times 6}, \frac{\partial \boldsymbol{e}_{ij}}{\partial \boldsymbol{\xi}_i}, \boldsymbol{0}_{2\times 6}, \ldots \boldsymbol{0}_{2\times 3}, \ldots \boldsymbol{0}_{2\times 3}, \frac{\partial \boldsymbol{e}_{ij}}{\partial \boldsymbol{p}_j}, \boldsymbol{0}_{2\times 3}, \ldots \boldsymbol{0}_{2\times 3}\right). \tag{10.51}$$

其中$0_{2\times 6}$表示维度为2×6的0矩阵，同理$0_{2\times 3}$也是一样。该误差项对相机姿态的偏导$\partial e_{ij}/\partial \xi_i$维度为2×6，对路标点的偏导$\partial e_{ij}/\partial p_j$维度是2×3。这个误差项的雅可比矩阵，除了这两处为非零块之外，其余地方都为零。这体现了该误差项与其他路标和轨迹无关的特性。那么，它对增量方程有何影响？H矩阵为什么会产生稀疏性呢？

以图10-3为例，我们设J_{ij}只在i,j处有非零块，那么它对H的贡献为$\boldsymbol{J}_{ij}^{\mathrm{T}}\boldsymbol{J}_{ij}$，具有图上所画的稀疏形式。这个$\boldsymbol{J}_{ij}^{\mathrm{T}}\boldsymbol{J}_{ij}$矩阵也仅有4个非零块，位于$(i,i),(i,j),(j,i),(j,j)$。

对于整体的H，由于：

$$\boldsymbol{H} = \sum_{i,j} \boldsymbol{J}_{ij}^{\mathrm{T}} \boldsymbol{J}_{ij}, \tag{10.52}$$

请注意i在所有相机位姿中取值，j在所有路标点中取值。我们把H进行分块：

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{H}_{11} & \boldsymbol{H}_{12} \\ \boldsymbol{H}_{21} & \boldsymbol{H}_{22} \end{bmatrix}. \tag{10.53}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

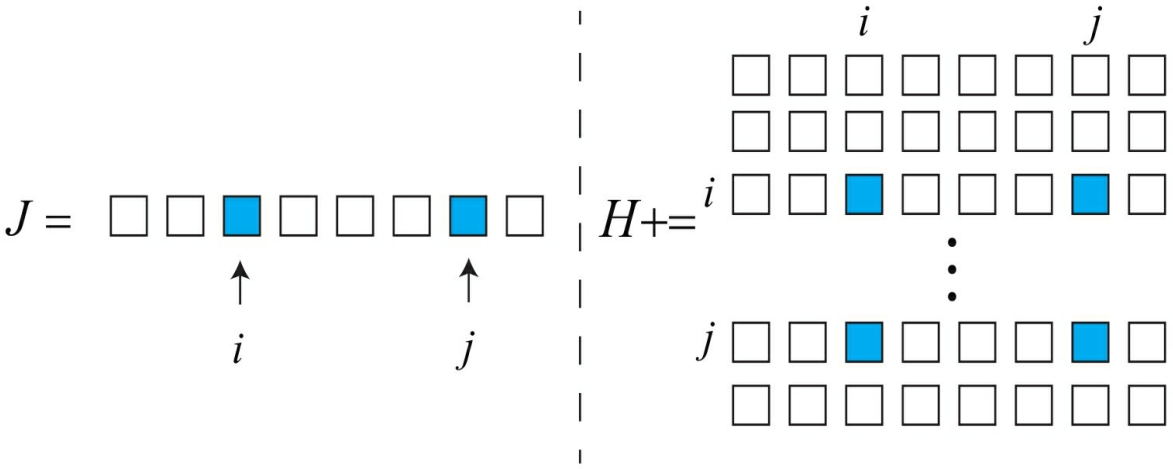


图10-3　当某个误差项J具有稀疏性时，它对H的贡献也具有稀疏形式。

这里H_{11}只和相机位姿有关，而H_{22}只和路标点有关。当我们遍历i,j时，以下事实总是成立的：

1.不管i,j怎么变，H_{11}都是对角阵，只在$H_{i,i}$处有非零块。

2.同理，H_{22}也是对角阵，只在$H_{j,j}$处有非零块。

3.对于H_{12}和H_{21}，它们可能是稀疏的，也可能是稠密的，视具体的观测数据而定。

这显示了H的稀疏结构。之后对线性方程的求解中，也正需要利用它的稀疏结构。也许读者还没有很好地领会这里的意思，我们举一个实例来直观说明它的情况。假设一个场景内有2个相机位姿（C_1,C_2）和6个路标（P_1,P_2,P_3,P_4,P_5,P_6）。这些相机和点云所对应的变量为ξ_i,i=1,2及p_j,j=1,⋯,6。相机C_1观测到路标P_1,P_2,P_3,P_4，相机C_2观测到路标P_3,P_4,P_5,P_6。我们把这个过程画成示意图10-4。相机和路标以圆形节点表示。如果i相机能够观测到j点云，我们就在它们对应的节点连上一条边。

可以推出，场景下的BA目标函数应该是

$$\frac{1}{2}\left(\|\boldsymbol{e}_{11}\|^2+\|\boldsymbol{e}_{12}\|^2+\|\boldsymbol{e}_{13}\|^2+\|\boldsymbol{e}_{14}\|^2+\|\boldsymbol{e}_{23}\|^2+\|\boldsymbol{e}_{24}\|^2+\|\boldsymbol{e}_{25}\|^2+\|\boldsymbol{e}_{26}\|^2\right). \tag{10.54}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

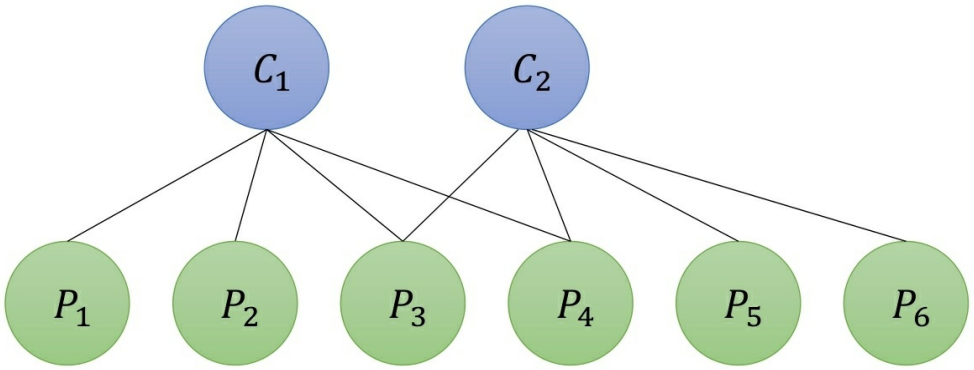


图10-4　点和边组成的示意图。该图显示相机C_1观测到了路标点P_1,P_2,P_3,P_4，相机C_2看到了P_3到P_6。

这里的e_{ij}使用之前定义过的代价函数，即式(10.42)。以e_{11}为例，它描述了在C_1看到了P_1这件事，与其他的相机位姿和路标无关。令J_{11}为e_{11}所对应的雅可比矩阵，不难看出e_{11}对相机变量ξ_2和路标点$p_2,\cdots,p_6$的偏导都为0。我们把所有变量以$x=(\xi_1,\xi_2,p_1,\cdots,p_2)^{\mathrm{T}}$的顺序摆放，则有：

$$\boldsymbol{J}_{11}=\frac{\partial \boldsymbol{e}_{11}}{\partial \boldsymbol{x}}=\left(\frac{\partial \boldsymbol{e}_{11}}{\partial \boldsymbol{\xi}_1},\mathbf{0}_{2\times 6},\frac{\partial \boldsymbol{e}_{11}}{\partial \boldsymbol{p}_1},\mathbf{0}_{2\times 3},\mathbf{0}_{2\times 3},\mathbf{0}_{2\times 3},\mathbf{0}_{2\times 3},\mathbf{0}_{2\times 3}\right). \tag{10.55}$$

为了方便表示稀疏性，我们用带有颜色的方块表示矩阵在该方块内有数值，其余没有颜色的区域表示矩阵在该处数值都为0。那么上面的J_{11}则可以表示成图10-5所示图案。同理，其他的雅可比矩阵也会有类似的稀疏图案。

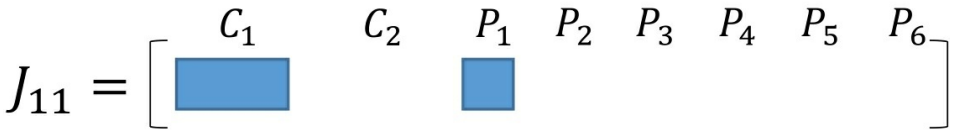


图10-5　J_{11}矩阵的非零块分布图。上方的标记表示矩阵该列所对应的变量。由于相机参数维数比点云参数维数要大，所以C_1对应的矩阵块要比P_1对应的矩阵块宽一些。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

为了得到该目标函数对应的雅可比矩阵，我们可以将这些J_{ij}按照一定顺序列为向量，那么整体雅可比矩阵及相应的H矩阵的稀疏情况就如图10-6所示。

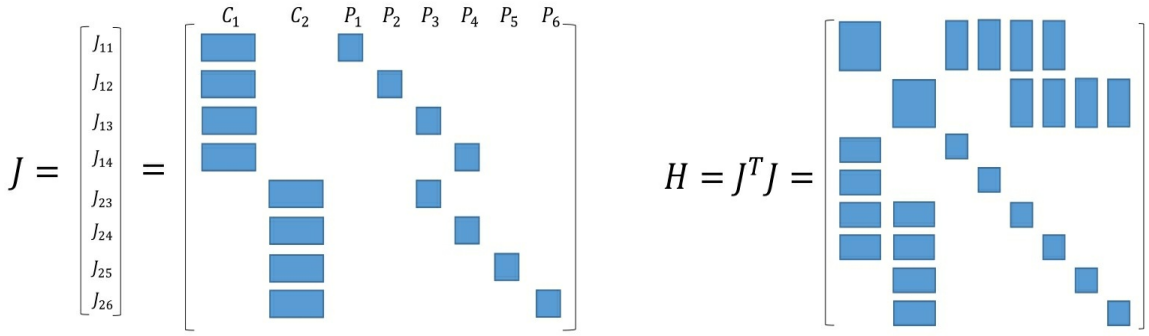


图10-6　雅可比矩阵的稀疏性（左）和H矩阵的稀疏性（右），填色的方块表示矩阵在对应的矩阵块处有数值，其余没有颜色的部分表示矩阵在该处的数值始终为0。

也许你已经注意到了，图10-4对应的邻接矩阵（Adjacency Matrix）[5]和上图中的H矩阵，除了对角元素以外的其余部分有着完全一致的结构。事实上的确如此。上面的H矩阵一共有8×8个矩阵块，对于H矩阵当中处于非对角线的矩阵块来说，如果该矩阵块非零，则其位置所对应的变量之间在图中会存在一条边，我们可以从图10-7中清晰地看到这一点。所以，H矩阵当中的非对角部分的非零矩阵块可以理解为其对应的两个变量之间存在联系，或者可以称之为约束。于是，我们发现图优化结构与增量方程的稀疏性存在着明显的联系。

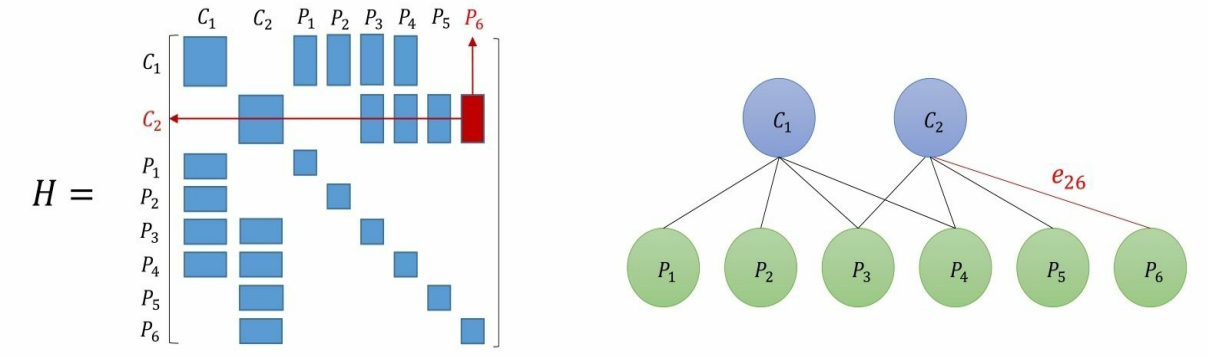


图10-7　H矩阵中非零矩阵块和图中边的对应关系。如左图H矩阵中右侧的红色矩阵块，表示在右图中其对应的变量C_2和P_6之间存在一条边e_{26}。

书籍汇总下载 http://t.cn/RDIAj5D

现在考虑更一般的情况，假如我们有m个相机位姿，n个路标点。由于通常路标的数量远远多于相机，于是有$n \gg m$。由上面的推理可知，实际的H矩阵会如图10-8所示。它的左上角块显得非常小，而右下角的对角块占据了大量地方。除此之外，非对角部分则分布着散乱的观测数据。由于它的形状很像箭头，又称为箭头形（Arrow-like）矩阵[6]。同时，它也很像一把镐子，所以笔者也称其为镐形矩阵。

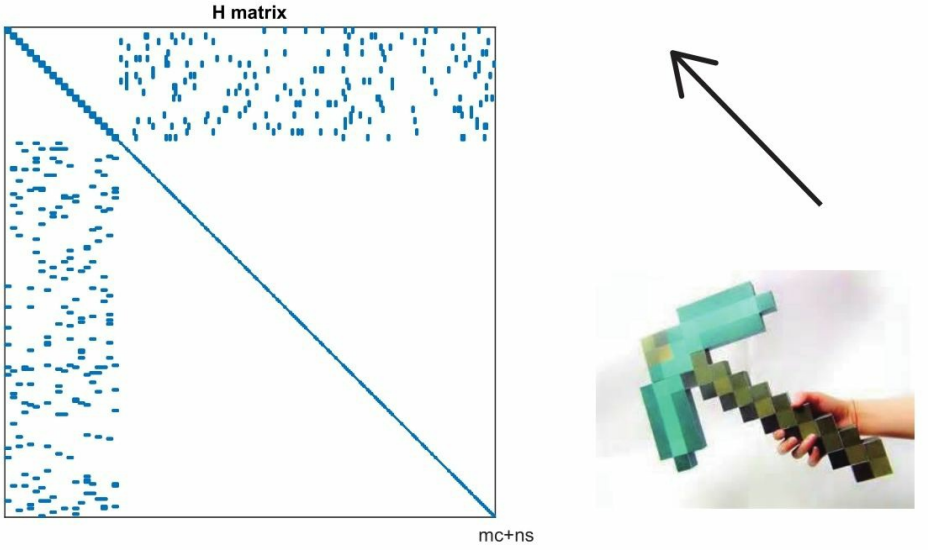


图10-8　一般情况下的H矩阵。

对于具有这种稀疏结构的H，线性方程$H\Delta x=g$的求解会有什么不同呢？现实当中存在着若干种利用H的稀疏性加速计算的方法，本节介绍视觉SLAM里一种最常用的手段：Schur消元。在SLAM研究中亦称为Marginalization（边缘化）。

仔细观察一下图10-8，我们不难发现这个矩阵可以分成4个块，和式(10.53)一致。左上角为对角块矩阵，每个对角块元素的维度与相机位姿的维度相同，且是一个对角块矩阵。右下角也是对角块矩阵，每个对角块的维度是路标的维度。非对角块的结构与具体观测数据相关。我们首先将这个矩阵按照图10-9所示的方式做区域划分，读者不难发现，这4个区域正好对应了公式(10.50)中的4个矩阵块。为了后续分析方便，我们记这4个块为B,E,E^{T},C。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

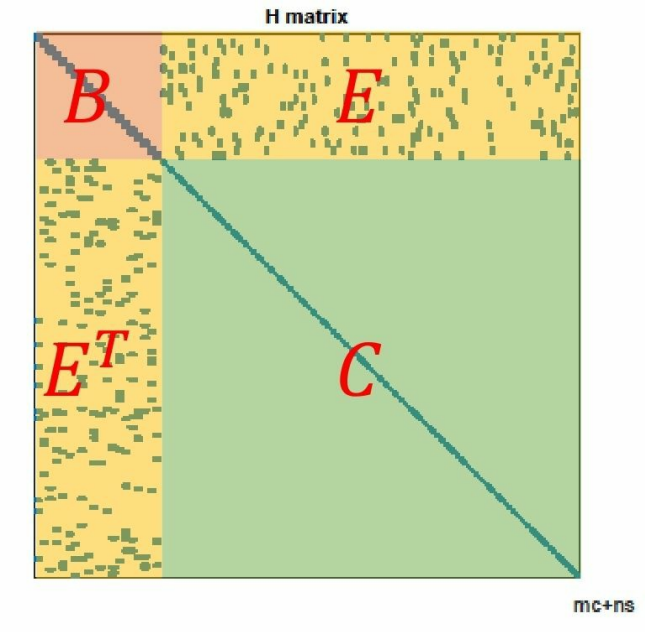


图10-9　H矩阵的区域划分。

于是，对应的线性方程组也可以由$H\Delta x=g$变为如下形式：

$$\begin{bmatrix} \boldsymbol{B} & \boldsymbol{E} \\ \boldsymbol{E}^{\mathrm{T}} & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x}_c \\ \Delta \boldsymbol{x}_p \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \end{bmatrix}. \tag{10.56}$$

其中B是对角块矩阵，每个对角块的维度和相机参数的维度相同，对角块的个数是相机变量的个数。由于路标数量会远远大于相机变量个数，所以C往往也远大于B。三维空间中每个路标点为三维，于是C矩阵为对角块矩阵，每个块为3×3矩阵。对角块矩阵求逆的难度远小于对一般矩阵的求逆难度，因为我们只需要对那些对角线矩阵块分别求逆即可。考虑到这个特性，我们对线性方程组进行高斯消元，目标是消去右上角的非对角部分E，得：

$$\begin{bmatrix} \boldsymbol{I} & -\boldsymbol{E}\boldsymbol{C}^{-1} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{B} & \boldsymbol{E} \\ \boldsymbol{E}^{\mathrm{T}} & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x}_c \\ \Delta \boldsymbol{x}_p \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{E}\boldsymbol{C}^{-1} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{w} \end{bmatrix}. \tag{10.57}$$

整理，得：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$\begin{bmatrix} \boldsymbol{B}-\boldsymbol{E}\boldsymbol{C}^{-1}\boldsymbol{E}^{\mathrm{T}} & \mathbf{0} \\ \boldsymbol{E}^{\mathrm{T}} & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{x}_c \\ \Delta\boldsymbol{x}_p \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}-\boldsymbol{E}\boldsymbol{C}^{-1}\boldsymbol{w} \\ \boldsymbol{w} \end{bmatrix}. \tag{10.58}$$

经过消元之后，方程组第一行变成和Δx_p无关的项。单独把它拿出来，得到关于位姿部分的增量方程：

$$\left[\boldsymbol{B}-\boldsymbol{E}\boldsymbol{C}^{-1}\boldsymbol{E}^{\mathrm{T}}\right]\Delta\boldsymbol{x}_c = \boldsymbol{v}-\boldsymbol{E}\boldsymbol{C}^{-1}\boldsymbol{w}. \tag{10.59}$$

这个线性方程的维度和B矩阵一样。我们的做法是先求解这个方程，然后把解得的Δx_c代入到原方程，然后求解Δx_p。这个过程称为Marginalization[68]，或者Schur消元（Schur Elimination）。相比于直接解线性方程的做法，它的优势在于：

1.在消元过程中，由于C为对角块，所以C^{-1}容易解出。

2.求解了Δx_c之后，路标部分的增量方程由$\Delta x_p=C^{-1}(w-E^{\mathrm{T}}\Delta x_c)$给出。这依然用到了$C^{-1}$易于求解的特性。

于是，边缘化的主要计算量在于求解式(10.59)。关于这个方程，我们能说的就不多了。它仅是一个普通的线性方程，没有特殊的结构可以利用。我们将此方程的系数记为S，它的稀疏性如何呢？图10-10显示了进行Schur消元之后的一个S实例，可以看到它的稀疏性是不规则的。

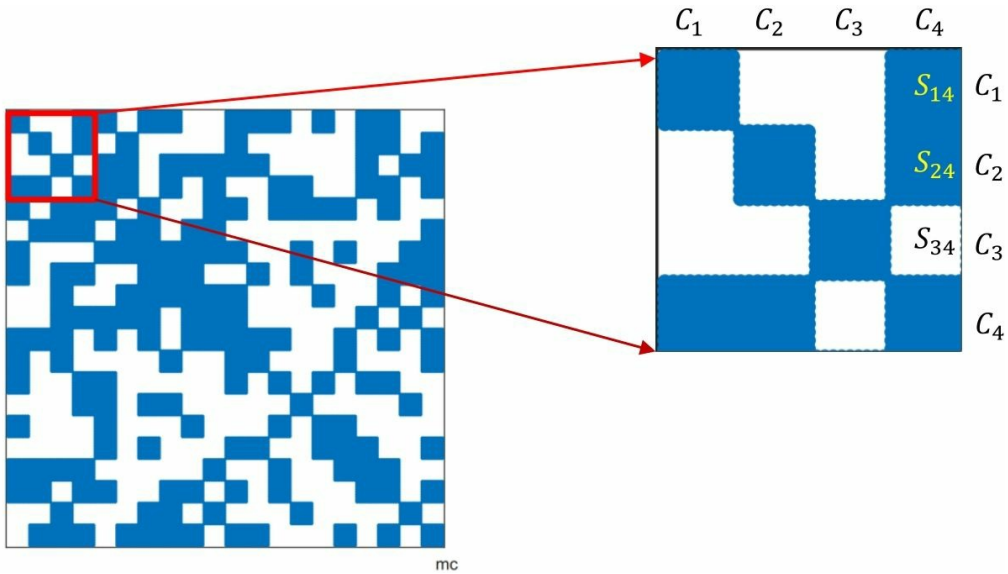
电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D



图10-10　对*H*矩阵进行Schur消元后*S*矩阵的稀疏状态。

前面说到，*H*矩阵的非对角块处的非零元素对应着相机和路标的关联。那么，进行了Schur消元后*S*的稀疏性是否具有物理意义呢？答案是肯定的。此处我们不加证明地说，*S*矩阵的非对角线上的非零矩阵块，表示了该处对应的两个相机变量之间存在着共同观测的路标点，有时候称为共视（Co-visibility）。反之，如果该块为零，则表示这两个相机没有共同观测。例如图10-11所示的稀疏矩阵，左上角前4×4个矩阵块可以表示对应的相机变量C_1,C_2,C_3,C_4之间有共同观测。



电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

图10-11　以S矩阵中前4×4个矩阵块为例，这个区域当中的矩阵块S_{14},S_{24}不为零，表示相机C_4和相机C_1、C_2之间有共同观测点；而S_{34}为零则表示C_3和C_4之间没有共同观测的路标。

于是，S矩阵的稀疏性结构当取决于实际观测的结果，我们无法提前预知。在实践当中，例如ORB-SLAM[73]中的Local Mapping环节，在做BA的时候刻意选择那些具有共同观测的帧作为关键帧，在这种情况下Schur消元后得到的S就是稠密矩阵。不过，由于这个模块并不是实时执行，所以这种做法也是可以接受的。但是在另一些方法里面，例如DSO[58]、OKVIS[74]等，它们采用了滑动窗口方法（Sliding Window）。这类方法对每一帧都要求做一次BA来防止误差的累积，因此它们也必须采用一些技巧来保持S矩阵的稀疏性。读者如果希望能够更加深入这一块，可以参考它们的论文。这里就不谈这些过于细节的事情了。

从概率角度来看，我们称这一步为边缘化，是因为我们实际上把求$(\Delta x_c, \Delta x_p)$的问题，转化成了先求Δx_c，再求Δx_p的过程。这一步相当于做了条件概率展开：

$$P(\boldsymbol{x}_c, \boldsymbol{x}_p) = P(\boldsymbol{x}_c) \cdot P(\boldsymbol{x}_p | \boldsymbol{x}_c). \tag{10.60}$$

结果是求出了关于x_c的边缘分布，故称边缘化。在前面讲的边缘化过程中，我们实际把所有的路标点都给边缘化了。根据实际情况，我们也能选择一部分进行边缘化。同时，Schur消元只是实现边缘化的其中一种方式，同样可以使用Cholesky分解进行边缘化。

读者可能会继续问，在进行了Schur消元后，我们还需要求解线性方程组(10.59)。对它的求解是否还有什么技巧呢？很遗憾，这部分就属于传统的矩阵数值求解了，通常是用分解来计算的。不管采用哪种求解办法，我们都建议利用H的稀疏性进行Schur消元。不光是因为这样可以提高速度，也因为消元后的S矩阵的条件数往往比之前的H矩阵要小。Schur消元也并不意味着将所有路标消元，将相机变量消元也是SLAM当中采用的手段。

10.2.4　鲁棒核函数

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

在前面的BA问题中，我们最小化误差项的二范数平方和作为目标函数。这种做法虽然很直观，但存在一个严重的问题：如果出于误匹配等原因，某个误差项给的数据是错误的，会发生什么呢？我们把一条原本不应该加到图中的边给加进去了，然而优化算法并不能辨别出这是个错误数据，它会把所有的数据都当作误差来处理。这时，算法会看到一条误差很大的边，它的梯度也很大，意味着调整与它相关的变量会使目标函数下降更多。所以，算法将试图调整这条边所连接的节点的估计值，使它们顺应这条边的无理要求。由于这条边的误差真的很大，往往会抹平了其他正确边的影响，使优化算法专注于调整一个错误的值。这显然不是我们希望看到的。

出现这种问题的原因是，当误差很大时，二范数增长得太快。于是就有了核函数的存在。核函数保证每条边的误差不会大得没边而掩盖掉其他的边。具体的方式是，把原先误差的二范数度量替换成一个增长没有那么快的函数，同时保证自己的光滑性质（不然无法求导！）。因为它们使得整个优化结果更为稳健，所以又叫它们鲁棒核函数（Robust Kernel）。

鲁棒核函数有许多种，例如最常用的Huber核：

$$H(e)=\begin{cases}\frac{1}{2}e^2 & \text{当}|e|\leqslant\delta,\\ \delta\left(|e|-\frac{1}{2}\delta\right) & \text{其他}\end{cases} \tag{10.61}$$

我们看到，当误差e大于某个阈值δ后，函数增长由二次形式变成了一次形式，相当于限制了梯度的最大值。同时，Huber核函数又是光滑的，可以很方便地求导。图10-12显示了Huber核函数与二次函数的对比，可见在误差较大时Huber核函数增长明显低于二次函数。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

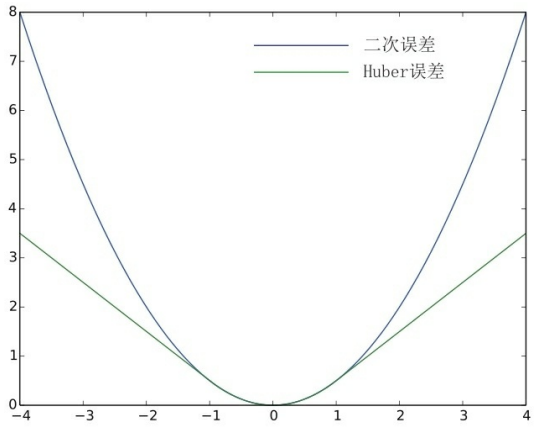


图10-12　Huber核函数。

除了Huber核之外，还有Cauchy核、Tukey核，等等，读者可以看看g2o和Ceres都提供了哪些核函数。

10.2.5　小结

本节我们重点介绍了BA中的稀疏性问题。不过，实践当中，有许多软件库为我们提供了细节操作，而我们需要做的主要是构造Bundle Adjustment问题，设置Schur消元，然后调用稠密或者稀疏矩阵求解器对变量进行优化即可。如果读者希望更深入地了解BA，可以在阅读完本节的基础上，进一步参考文献[26]学习。

下面的两节，我们将使用g2o和Ceres两个库来做Bundle Adjustment。为了体现出它们的区别，我们会让这个程序共用很多代码。共用的代码主要在common文件夹下，其他的文件则不同。我们将使用公开的数据库[75]当中的文件来进行编程练习。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

10.3 实践：g2o

10.3.1 BA数据集

在本讲代码目录下的common文件夹是两个实验共有的数据集部分。它的布局如图10-13所示。其中， fl ags文件夹下的两个文件定义了CommandArgs这个类，该类用来解析用户输入的参数，同时也对程序需要的参数提供默认值及文档说明。BundleParams这个类定义了Bundle Adjustment使用的所有参数，也调用了CommandArgs类型的变量。由于CommandArgs这个类的存在，我们可以直接在程序后面使用-help来查看程序所有参数的含义，使用方式可以参考该程序中BundleParams类型的写法。

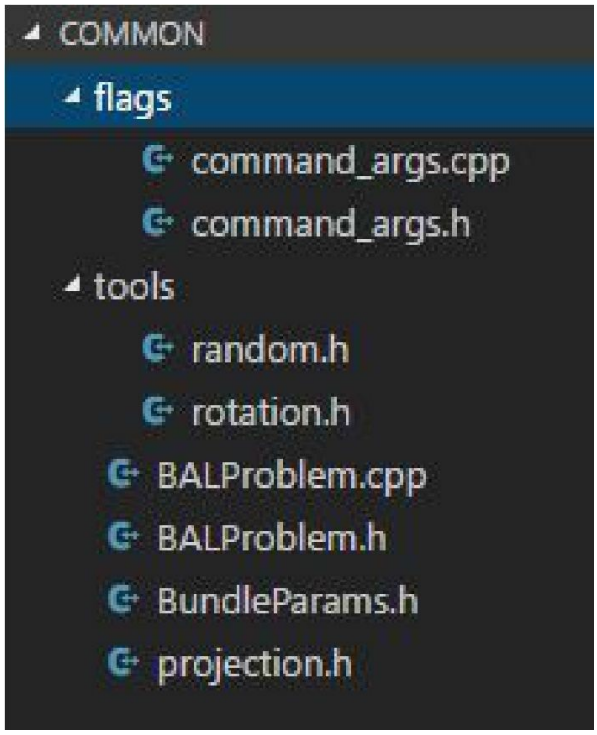


图10-13 common文件夹目录结构。

tools是一些数学工具函数，读者可自行查看。我们的相机和路标参数与前面提到的代价函数，即式(10.42)保持一致，不过在实验中我们要

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

把相机内参也当作优化变量考虑。我们用f,k_1,k_2来表示焦距和畸变，3个参数表示平移，3个参数表示旋转（于是相机一共有9个未知量），路标也使用三维坐标来表示。projection.h是前面提到的投影计算流程图10-2的代码实现。由于g2o和ceres都会用到这个投影模型，因此这个文件也将被两个工程所共享。

除此之外，BALProblem这个类将保存我们的BundleAdjustment所需要的所有数据，其中就包括了相机和路标之间的关联，以及相机变量和路标变量的初始值，这些数据的原始信息都保存在data文件夹下的TXT文件当中。我们可以打开data/problem-16-22106-pre.txt查看文件信息：

```
16 22106 83718
0 0     -3.859900e+02 3.871200e+02
1 0     -3.844000e+01 4.921200e+02
2 0     -6.679200e+02 1.231100e+02
7 0     -5.991800e+02 4.079300e+02
```

它的第一行记录了相机数量、路标数量和总的观测数量。从第二行起，分别记录第i个相机观测第j个路标所看到的像素坐标。这就构成了一个BA问题，亦可看成由纯观测方程组成的SLAM问题。除此之外，BALProblem也提供了将数据导出到PLY文件这样的功能，方便用户通过Meshlab软件查看点云的三维信息。

10.3.2 g2o求解BA

下面来考虑如何使用g2o求解这个BA问题。和以前一样，g2o使用图模型来描述问题的结构，所以我们要用节点来表示相机和路标，然后用边来表示它们之间的观测。我们仍然使用自定义的点和边，只需覆盖一些关键函数即可。针对相机和路标，我们可以定义如下继承，使用override关键字来表示对基类虚函数的覆盖：

slambook/ch10/g2o_custombundle/g2o_bal_class.h

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
#include "g2o/core/base_vertex.h"
#include "g2o/core/base_binary_edge.h"

#include <Eigen/Core>
#include "ceres/autodiff.h"

#include "tools/rotation.h"
#include "common/projection.h"

// 节点没什么好说的，直接把更新量看作向量求加法即可
class VertexCameraBAL : public g2o::BaseVertex<9,Eigen::VectorXd>
{
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW;
    VertexCameraBAL(){}
    virtual bool read(std::istream& is) {return false;}
    virtual bool write ( std::ostream& os ) const {return false;}
    virtual void setToOriginImpl(){}

    virtual void oplusImpl(const double* update) override {
        Eigen::VectorXd::ConstMapType v(update, VertexCameraBAL::Dimension);
        _estimate += v;
    }
```

书籍汇总下载 http://t.cn/RDIAj5D

```
};

class VertexPointBAL : public g2o::BaseVertex<3, Eigen::Vector3d>
{
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW;
    VertexPointBAL(){}
    virtual bool read( std::istream& is) {return false;}
    virtual bool write(std::ostream& os) const {return false;}
    virtual void setToOriginImpl() {}
    virtual void oplusImpl(const double* update) override {
        Eigen::Vector3d::ConstMapType v(update);
        _estimate += v;
    }
};
```

定义了节点以后，还需要定义边来表示节点之间的联系。每一条边都对应了一个代价函数(10.42)，同时为了避免复杂的求导运算，我们借助Ceres库当中的Autodi ff（即自动求导）功能。该功能需要类型将需要求导的公式实现在括号运算符operator()里，代码如下：

slambook/ch10/g2o_custombundle/g2o_bal_class.h

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
class EdgeObservationBAL : public g2o::BaseBinaryEdge<2, Eigen::Vector2d, VertexCameraBAL, VertexPointBAL>{
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW;
    EdgeObservationBAL(){}

    virtual bool read(std::istream& /*is*/){ return false;}

    virtual bool write(std::ostream& /*os*/)const { return false;}

    virtual void computeError() override   // 覆盖基类函数，使用operator()计算代价函数
    {
        const VertexCameraBAL* cam = static_cast<const VertexCameraBAL*>(vertex(0));
        const VertexPointBAL* point = static_cast<const VertexPointBAL*>(vertex(1));

        (*this)(cam->estimate().data(), point->estimate().data(), _error.data());
    }

    // 为了使用 Ceres 求导功能而定义的函数，让本类成为拟函数类
    template<typename T>
```

书籍汇总下载 http://t.cn/RDIAj5D

```
20      bool operator()( const T* camera, const T* point, T* residuals  )const
21      {
22          T predictions[2];
23          CamProjectionWithDistortion(camera, point, predictions);
24          residuals[0] = predictions[0] - T(measurement()(0));
25          residuals[1] = predictions[1] - T(measurement()(1));
26          return true;
27      }
28
29      virtual void linearizeOplus() override
30      {
31          const VertexCameraBAL* cam = static_cast<const VertexCameraBAL*>(vertex(0))
            ;
32          const VertexPointBAL* point = static_cast<const VertexPointBAL*>(vertex(1))
            ;
33          typedef ceres::internal::AutoDiff<EdgeObservationBAL, double,
            VertexCameraBAL::Dimension, VertexPointBAL::Dimension> BalAutoDiff;
34
35          Eigen::Matrix<double, Dimension, VertexCameraBAL::Dimension, Eigen::
            RowMajor> dError_dCamera;
36          Eigen::Matrix<double, Dimension, VertexPointBAL::Dimension, Eigen::RowMajor
            > dError_dPoint;
37          double *parameters[] = { const_cast<double*>(cam->estimate().data()),
            const_cast<double*>(point->estimate().data()) };
38          double *jacobians[] = { dError_dCamera.data(), dError_dPoint.data() };
39          double value[Dimension];
40          // Ceres 中的自动求导函数用法, 需要提供 operator() 函数成员
41          bool diffState = BalAutoDiff::Differentiate(*this, parameters, Dimension,
            value, jacobians);
42
43          // copy over the Jacobians (convert row-major -> column-major)
44          if (diffState) {
45              _jacobianOplusXi = dError_dCamera;
46              _jacobianOplusXj = dError_dPoint;
47          } else {
48              assert(0 && "Error while differentiating");
49              _jacobianOplusXi.setZero();
50              _jacobianOplusXi.setZero();
51          }
52      }
53  };
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

以上定义了本问题中使用的节点和边。下面需要根据BALProblem类当中的实际数据来生成一些节点和边，交给g2o进行优化。值得注意的是，为了充分利用BA中的稀疏性，需要在这里将路标中的setMarginalized属性设置为true。代码的主要片段如下：

slambook/ch10/g2o_custombundle/g2o_bundle.cpp（片段）

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
void BuildProblem(const BALProblem* bal_problem, g2o::SparseOptimizer* optimizer,
const BundleParams& params)
{
    const int num_points = bal_problem->num_points();
    const int num_cameras = bal_problem->num_cameras();
    const int camera_block_size = bal_problem->camera_block_size();
    const int point_block_size = bal_problem->point_block_size();

    // 添加相机和对应的初始值
    const double* raw_cameras = bal_problem->cameras();
    for(int i = 0; i < num_cameras; ++i)
    {
        ConstVectorRef temVecCamera(raw_cameras + camera_block_size * i,
        camera_block_size);
        VertexCameraBAL* pCamera = new VertexCameraBAL();
        pCamera->setEstimate(temVecCamera);   // 初始值
        pCamera->setId(i);
        optimizer->addVertex(pCamera);
    }

    // Set point vertex with initial value in the dataset.
    const double* raw_points = bal_problem->points();
    // const int point_block_size = bal_problem->point_block_size();
    for(int j = 0; j < num_points; ++j)
    {
        ConstVectorRef temVecPoint(raw_points + point_block_size * j,
        point_block_size);
        VertexPointBAL* pPoint = new VertexPointBAL();
        pPoint->setEstimate(temVecPoint);   // 点云的初始值
        pPoint->setId(j + num_cameras);     // 为了不和相机冲突，加上相机的 id

        // 设置该点在解方程时进行 Schur 消元
        pPoint->setMarginalized(true);
        optimizer->addVertex(pPoint);
    }

    // 为图添加边
    const int  num_observations = bal_problem->num_observations();
    const double* observations = bal_problem->observations();

    for(int i = 0; i < num_observations; ++i)
    {
        EdgeObservationBAL* bal_edge = new EdgeObservationBAL();
        const int camera_id = bal_problem->camera_index()[i]; // 使用前面的 id 获取
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

```
        相机
42      const int point_id = bal_problem->point_index()[i] + num_cameras; // 使用前
        面的 id 获得点
43      // Huber loss 函数（默认为不用设置）
44      if(params.robustify)
45      {
46          g2o::RobustKernelHuber* rk = new g2o::RobustKernelHuber;
47          rk->setDelta(1.0);
48          bal_edge->setRobustKernel(rk);
49      }
50      // 设置边所对应的两个节点
51      bal_edge->setVertex(0,dynamic_cast<VertexCameraBAL*>(optimizer->vertex(
        camera_id)));
52      bal_edge->setVertex(1,dynamic_cast<VertexPointBAL*>(optimizer->vertex(
        point_id)));
53      // 设置其协方差矩阵（在此处为单位矩阵）
54      bal_edge->setInformation(Eigen::Matrix2d::Identity());
55      // 设置边所对应的观测值
56      bal_edge->setMeasurement(Eigen::Vector2d(observations[2*i+0],observations
        [2*i + 1]));
57
58      optimizer->addEdge(bal_edge) ;
59    }
60  }
```

10.3.3 求解

在使用g2o时，求解的设置无非这几种：1.使用何种方法（列文伯格—马夸尔特方法、DogLeg等）来定义非线性优化的下降策略；2.使用哪类线性求解器。注意到这里需要使用到稀疏性，所以必须选用稀疏的求解器。

slambook/ch10/g2o_custombundle/g2o_bundle.cpp（片段）

```
typedef g2o::BlockSolver<g2o::BlockSolverTraits<9,3> > BalBlockSolver;

void SetSolverOptionsFromFlags(BALProblem* bal_problem, const BundleParams& params,
 g2o::SparseOptimizer* optimizer)
{
    BalBlockSolver* solver_ptr;

    g2o::LinearSolver<BalBlockSolver::PoseMatrixType>* linearSolver = 0;

    // 使用稠密计算方法
    if (params.linear_solver == "dense_schur" ) {
```

```
        linearSolver = new g2o::LinearSolverDense<BalBlockSolver::PoseMatrixType>()
        ;
    }
    // 使用稀疏计算方法
    else if (params.linear_solver == "sparse_schur") {
        linearSolver = new g2o::LinearSolverCholmod<BalBlockSolver::PoseMatrixType
        >();
        // 让 solver 对矩阵进行排序保持稀疏性
        dynamic_cast<g2o::LinearSolverCholmod<BalBlockSolver::PoseMatrixType>* >(
        linearSolver)->setBlockOrdering(true);
    }

    solver_ptr = new BalBlockSolver(linearSolver);

    g2o::OptimizationAlgorithmWithHessian* solver;
    if(params.trust_region_strategy == "levenberg_marquardt"){
        solver = new g2o::OptimizationAlgorithmLevenberg(solver_ptr);  // 使用列文
        伯格—马夸尔特下降法
    }
    else if(params.trust_region_strategy == "dogleg"){
        solver = new g2o::OptimizationAlgorithmDogleg(solver_ptr);  // 使用 DogLeg
        下降法
    }
    else
    {
        std::cout << "Please check your trust_region_strategy parameter again.."<<
        std::endl;
        exit(EXIT_FAILURE);
    }

    optimizer->setAlgorithm(solver);
}
```

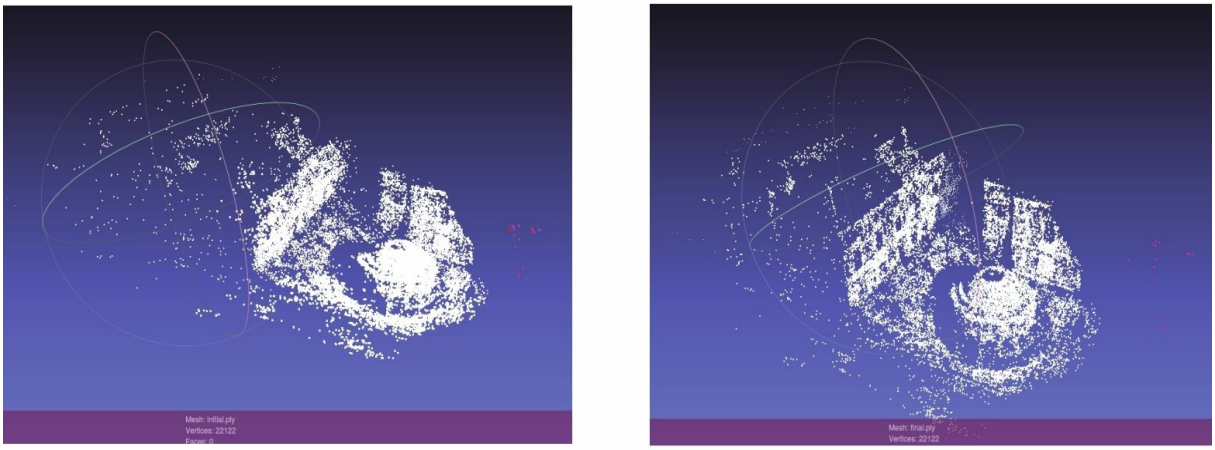
现在问题基本上搭建好了。我们通过BuildProblem函数完成了对目标函数的构造，也通过SetSolverOptionsFromFlags函数使用用户的输入参数来设置优化求解。剩下要做的就是思考该如何求解这个问题了。有了g2o这样的优化库，求解基本上可以一步完成，这点跟前面介绍的g2o用法几乎是一样的：

书籍汇总下载 http://t.cn/RDIAj5D

slambook/ch10/g2o_custombundle/g2o_bundle.cpp（片段）

```
optimizer.initializeOptimization();
optimizer.setVerbose(true);
optimizer.optimize(params.num_iterations);
```

使用Meshlab分别打开g2o执行文件夹下的initial.ply和 fi nal.ply两个文件，优化前后的效果如图10-14所示。



初始值　　优化值

图10-14　g2o优化结果。左侧为优化前初始值，右侧为优化后的。

由于g2o库本身公开API说明也少，我们通常只能通过网上的一些开源项目及其本身的源代码来了解它。这里需要提醒读者的是，g2o在做Bundle　Adjustment的优化时必须将其所有点云全部Schur掉，否则会出错！其原因在于我们使用了g2o::LinearSolver<BalBlockSolver::PoseMatrixType>这个类型来指定linearsolver，其中模板参数当中的Pose-MatrixType在程序中为相机姿态参数的维度，于是Bundle Adjustment当中Schur消元后解的线性方程组必须是只含有相机姿态变量[6]。

Ceres库则没有g2o这样的限制。Ceres给了开发者很大的空间去操作自己的优化策略，它在Schur消元操作时完全不需要把所有点云都消元掉，用户可以自己编写函数选择消元哪些点云。接下来我们也会给出Ceres的BA示例。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

10.4 实践：Ceres

现在我们再来看看换成Ceres实现同样的效果该如何做。如前面介绍的那样，Ceres是针对一般优化问题产生的库，因此它并没有像g2o那样提供一些Vertex或者Edge这类友好的接口。不过这也给Ceres带来了极大的灵活性。我们接下来可以看到，Ceres可以在原始数组上进行直接的优化操作，而g2o则通过复制的手段将数据通过Vertex或者Edge结构复制到optimizer里面。相比之下，Ceres对“优化”显得更贴近一些。

10.4.1 Ceres求解BA

g2o用Edge来保存每一个代价函数，而Ceres却是用Problem类型来构建最终的目标函数。与第6讲介绍的一致，我们使用AddResidualBlock来添加代价函数，最后组成整个目标函数。为了简化整个过程，我们首先定义代价函数类型，并且定义Create成员来使用Ceres当中的AutoDi ff特性：

slambook/ch10/ceres_custombundle/SnavelyReprojectionError.h

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
class SnavelyReprojectionError
{
public:
    SnavelyReprojectionError(double observation_x, double observation_y):observed_x
    (observation_x),observed_y(observation_y){}

    template<typename T>
    bool operator()(const T* const camera, const T* const point, T* residuals)const
    {
        // camera[0,1,2] are the angle-axis rotation
        T predictions[2];
        CamProjectionWithDistortion(camera, point, predictions);
        residuals[0] = predictions[0] - T(observed_x);
        residuals[1] = predictions[1] - T(observed_y);

        return true;
    }

    static ceres::CostFunction* Create(const double observed_x, const double
    observed_y){
        return (new ceres::AutoDiffCostFunction<SnavelyReprojectionError,2,9,3>(
            new SnavelyReprojectionError(observed_x,observed_y)));
    }

private:
    double observed_x;
    double observed_y;
};
```

定义好之后，我们就可以使用SnavelyReprojectionError::Create函数来轻松地构建这个目标函数：

slambook/ch10/ceres_custombundle/ceresBundle.cpp（片段）

```
void BuildProblem(BALProblem* bal_problem, Problem* problem, const BundleParams&
params)
{
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
    const int point_block_size = bal_problem->point_block_size();
    const int camera_block_size = bal_problem->camera_block_size();
    double* points = bal_problem->mutable_points();
    double* cameras = bal_problem->mutable_cameras();

    // Observations is 2 * num_observations long array observations
    // [u_1, u_2, ... u_n], where each u_i is two dimensional, the x
    // and y position of the observation.
    const double* observations = bal_problem->observations();

    for(int i = 0; i < bal_problem->num_observations(); ++i){
    CostFunction* cost_function;

    // Each Residual block takes a point and a camera as input
    // and outputs a 2 dimensional Residual

    cost_function = SnavelyReprojectionError::Create(observations[2*i + 0],
    observations[2*i + 1]);

    // If enabled use Huber's loss function.
    LossFunction* loss_function = params.robustify ? new HuberLoss(1.0) : NULL;

    // Each observatoin corresponds to a pair of a camera and a point
    // which are identified by camera_index()[i] and point_index()[i]
    // respectively.
    double* camera = cameras + camera_block_size * bal_problem->camera_index()[i];
    double* point = points + point_block_size * bal_problem->point_index()[i];

    problem->AddResidualBlock(cost_function, loss_function, camera, point);
}
```

值得一提的是，为了使用Schur消元，Ceres采取的策略和g2o有很大的不同。Ceres采用额外的类型ParameterBlockOrdering来管理Schur消元顺序，并且使用AddElement-ToGroup来对变量进行编号从而定义消元顺序。例如下面设置点云变量为0，相机变量为1就可以让点云变量先进行消元（优先消元编号最小的变量）：

slambook/ch10/ceres_custombundle/ceresBundle.cpp

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
void SetOrdering(BALProblem* bal_problem, ceres::Solver::Options* options, const
BundleParams& params)
{
    const int num_points = bal_problem->num_points();
    const int point_block_size = bal_problem->point_block_size();
    double* points = bal_problem->mutable_points();

    const int num_cameras = bal_problem->num_cameras();
    const int camera_block_size = bal_problem->camera_block_size();
    double* cameras = bal_problem->mutable_cameras();

    if (params.ordering == "automatic")
        return;

    ceres::ParameterBlockOrdering* ordering = new ceres::ParameterBlockOrdering;

    // The points come before the cameras
    for(int i = 0; i < num_points; ++i)
        ordering->AddElementToGroup(points + point_block_size * i, 0);

    for(int i = 0; i < num_cameras; ++i)
        ordering->AddElementToGroup(cameras + camera_block_size * i, 1);

    options->linear_solver_ordering.reset(ordering);
}
```

10.4.2 求解

Ceres除了能够在原始数组上操作以外，另一大优势在于它的优化设置非常便捷。g2o的设置全靠变量类型来选择不同的下降策略，以及选择稠密或者稀疏的线性方程组解法，而Ceres全靠给Solver::Options的类型成员变量进行赋值来调整，这种方式比g2o快捷便利很多。Options类型几乎集成了Ceres的所有求解方法设置和管理，包括我们上面提到的Schur消元顺序（注意上面函数当中的最后一行）。

slambook/ch10/ceres_custombundle/ceresBundle.cpp（片段）

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
void SetSolverOptionsFromFlags(BALProblem* bal_problem,
const BundleParams& params, Solver::Options* options){

    options->max_num_iterations = params.num_iterations;
    options->minimizer_progress_to_stdout = true;
    options->num_threads = params.num_threads; // 用于计算的线程数目，可以加速雅可比矩阵的计算

    // 下降策略的选取
    CHECK(StringToTrustRegionStrategyType(params.trust_region_strategy,
    &options->trust_region_strategy_type));

    //  linear solver 的选取
    CHECK(ceres::StringToLinearSolverType(params.linear_solver, &options->
    linear_solver_type));

    CHECK(ceres::StringToSparseLinearAlgebraLibraryType(params.
    sparse_linear_algebra_library, &options->sparse_linear_algebra_library_type));
    CHECK(ceres::StringToDenseLinearAlgebraLibraryType(params.
    dense_linear_algebra_library, &options->dense_linear_algebra_library_type));

    // 变量排序的设置
    SetOrdering(bal_problem, options,params);
}
```

Ceres的求解也很简单。和前面提到的曲线拟合的例子一样，只需要简单地设置一下关于梯度和相邻两次迭代之间目标函数之差的相关阈值就可以。Solve函数负责Ceres的求解功能，只需要传给它对应的选项、目标函数即可。Summary类型用来负责函数求解每一次迭代的结果统计。

slambook/ch10/ceres_custombundle/ceresBundle.cpp（片段）

```
   options.gradient_tolerance = 1e-16;
   options.function_tolerance = 1e-16;
   Solver::Summary summary;
   Solve(options, &problem, &summary);
```

和g2o示例一样，该程序在运行后也可以在执行文件目录下找到 final.ply文件和initial.ply，由于该程序和g2o程序采用一致的数据，因此initial.ply具有相同的图案。使用MeshLab软件打开 final.ply可以看到如图10-15所示的结果，通过对比也可以发现优化结果和g2o完全一样。

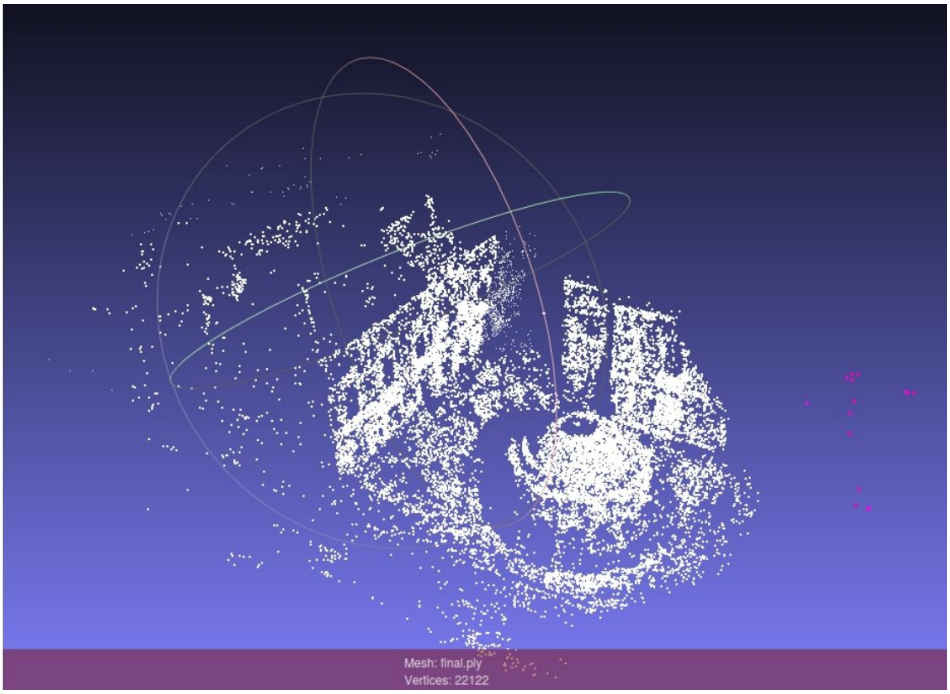


图10-15 Ceres的优化结果。

Ceres库公开的API说明详细，同时源代码可读性也高，推荐读者多多阅读Ceres源代码，并且自己尝试在Schur消元操作中只消去部分点云变量，或者夹杂着消去一些相机变量。这只需要操作ceres::ParameterBlockOrdering::AddElementToGroup函数，在对应变量地址上，用序号指定顺序即可。相比g2o这样公开文档太少的库，我们更加推荐读者更多地使用Ceres这样的库来做SLAM。

书籍汇总下载 http://t.cn/RDIAj5D

10.5 小结

本讲比较深入地探讨了状态估计问题与图优化的求解。我们看到在经典模型中SLAM可以看成状态估计问题。如果我们假设马尔可夫性，只考虑当前状态，则得到以EKF为代表的滤波器模型。如若不然，我们也可以选择考虑所有的运动和观测，它们构成一个最小二乘问题。在只有观测方程的情况下，这个问题称为BA，并可利用非线性优化方法求解。我们仔细讨论了求解过程中的稀疏性问题，指出了该问题与图优化之间的联系。最后，我们演示了如何使用g2o和Ceres库求解同一个优化问题，让读者对BA有一个直观的认识。

习题

1.证明式(10.25)成立。提示：你可能会用到SMW（Sherman-Morrison-Woodbury）公式，参考文献[76,6]。

2.对比g2o和Ceres的优化后目标函数的数值。指出为什么两者在Meshlab中效果一样但数值却不同。

3.对Ceres当中的部分点云进行Schur消元，看看结果会有什么区别。

4.证明S矩阵为半正定矩阵。

5.阅读文献[28]，看看g2o对核函数是如何处理的。与Ceres中的Loss function有何联系？

6.*在两个示例中，我们优化了相机位姿、以f,k_1,k_2为参数的相机内参及路标点。请考虑使用第5讲介绍的完整的相机模型进行优化，即，至少考虑f_x,f_y,p_1,p_2,k_1,k_2这些量。修改现在的Ceres和g2o程序以完成实验。

---

[1]这里的等号并不严格，实际允许相差与x_k无关的常数。

[2]这里看似有一点儿循环定义的意思。我们由$\hat{P}_k$定义了K，又把$\hat{P}_k$写成了K的表达式。然而，实际当中K可以不依靠$\hat{P}_k$算得，参见本讲的习题。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

[3]粒子滤波器的原理与卡尔曼滤波有较大不同。

[4]也译为光束法平差、捆集调整等，但笔者觉得没有Bundle Adjustment这个英文来得直观，所以这里保留英文名称。

[5]所谓邻接矩阵是这样一种矩阵，它的第i,j个元素描述了节点i和j是否存在一条边。如果存在此边，设这个元素为1，否则设为0。

[6]这种限制实在让人很懊恼。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

第11讲　后端2

主要目标

1.理解Pose Graph优化。

2.理解因子图优化。

3.理解增量式图优化的工作原理。

4.通过实验掌握g2o的Pose Graph优化与gtsam的因子图优化。

上一讲我们重点介绍了以BA为主的图优化。BA能精确地优化每个相机位姿与特征点位置。不过在更大的场景中，大量特征点的存在会严重降低计算效率，导致计算量越来越大以至于无法实时化。本讲介绍两种在更大场景下使用的后端优化方法：位姿图和因子图。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

11.1 位姿图（PoseGraph）

11.1.1 Pose Graph的意义

带有相机位姿和空间点的图优化称为BA，能够有效地求解大规模的定位与建图问题。但是，随着时间的流逝，机器人的运动轨迹将越来越长，地图规模也将不断增长。像BA这样的方法，计算效率会（令人担忧地）不断下降。根据前面的讨论，我们发现特征点在优化问题中占据了绝大部分。而实际上，经过若干次观测之后，那些收敛的特征点，空间位置估计会收敛至一个值保持不动，而发散的外点则通常看不到了。对收敛点再进行优化，似乎是有些费力不讨好的。因此，我们更倾向于在优化几次之后就把特征点固定住，只把它们看作位姿估计的约束，而不再实际地优化它们的位置估计。

沿着这个思路往下走，我们会想到：是否能够完全不管路标而只管轨迹呢？我们完全可以构建一个只有轨迹的图优化，而位姿节点之间的边，可以由两个关键帧之间通过特征匹配之后得到的运动估计来给定初始值。不同的是，一旦初始估计完成，我们就不再优化那些路标点的位置，而只关心所有的相机位姿之间的联系了。通过这种方式，我们省去了大量的特征点优化的计算，只保留了关键帧的轨迹，从而构建了所谓的位姿图（Pose Graph），如图11-1所示。

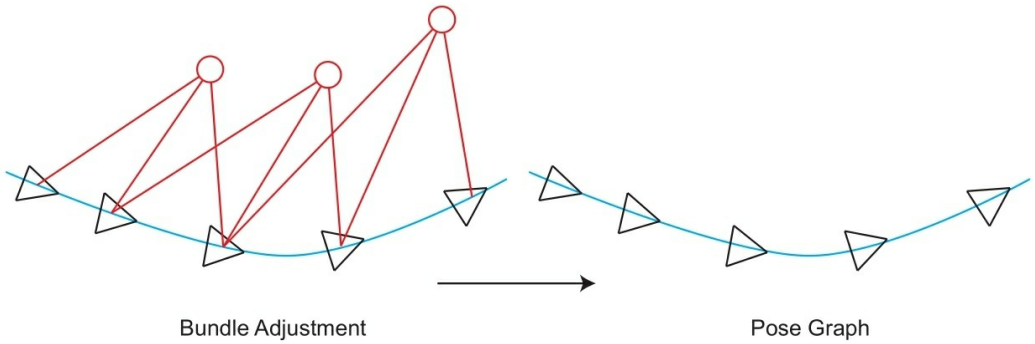


图11-1 Pose Graph示意图。当我们不再优化Bundle Adjustment中的路标点，仅把它们看成对姿态节点的约束时，就得到了一个计算规模减小很多的Pose Graph。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

我们知道，在BA中特征点数量远大于位姿节点。一个关键帧往往关联了数百个关键点，而实时BA的最大计算规模，即使利用稀疏性，在当前的主流CPU上一般也就是几万个点左右。这就限制了SLAM应用场景。所以，当机器人在更大范围的时间和空间中运动时，必须考虑一些解决方式：要么像滑动窗口法那样，丢弃一些历史数据[77]；要么像Pose Graph的做法那样，舍弃对路标点的优化，只保留Pose之间的边，使用Pose Graph[78,79,80]。

11.1.2 Pose Graph的优化

那么，Pose Graph图优化中的节点和边都是什么意思呢？这里的节点表示相机位姿，以$\xi_1,\cdots,\xi_n$来表达。而边，则是两个位姿节点之间相对运动的估计，该估计可能来自于特征点法或直接法，但不管如何，我们估计了，比如说ξ_i和ξ_j之间的一个运动$\Delta\xi_{ij}$。该运动可以有若干种表达方式，我们取比较自然的一种：

$$\Delta\boldsymbol{\xi}_{ij} = \boldsymbol{\xi}_i^{-1} \circ \boldsymbol{\xi}_j = \ln\left(\exp\left((-\boldsymbol{\xi}_i)^\wedge\right)\exp\left(\boldsymbol{\xi}_j^\wedge\right)\right)^\vee, \tag{11.1}$$

或按李群的写法：

$$\boldsymbol{\Delta T}_{ij} = \boldsymbol{T}_i^{-1}\boldsymbol{T}_j. \tag{11.2}$$

按照图优化的思路来看，实际当中该等式不会精确地成立，因此我们设立最小二乘误差，然后和以往一样，讨论误差关于优化变量的导数。这里，我们把上式的ΔT_{ij}移至等式右侧，构建误差e_{ij}：

$$\begin{aligned}\boldsymbol{e}_{ij} &= \ln\left(\Delta\boldsymbol{T}_{ij}^{-1}\boldsymbol{T}_i^{-1}\boldsymbol{T}_j\right)^\vee \\ &= \ln\left(\exp((-\boldsymbol{\xi}_{ij})^\wedge)\exp((-\boldsymbol{\xi}_i)^\wedge)\exp(\boldsymbol{\xi}_j^\wedge)\right)^\vee.\end{aligned} \tag{11.3}$$

注意优化变量有两个：ξ_i和ξ_j，因此我们求e_{ij}关于这两个变量的导数。按照李代数的求导方式，给ξ_i和ξ_j各一个左扰动：$\delta\xi_i$和$\delta\xi_j$。于是误差变为

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$\hat{\boldsymbol{e}}_{ij} = \ln\left(\boldsymbol{T}_{ij}^{-1}\boldsymbol{T}_i^{-1}\exp((-\boldsymbol{\delta\xi}_i)^\wedge)\exp(\delta\boldsymbol{\xi}_j^\wedge)\boldsymbol{T}_j\right)^\vee. \tag{11.4}$$

该式中，两个扰动项被夹在了中间。为了利用BCH近似，我们希望把扰动项移至式子左侧或右侧。回忆第4讲习题中的伴随性质，即式(4.49)。如果你没有做过这个习题，那就暂时想当然地认为其正确。

$$\exp\left((\mathrm{Ad}(\boldsymbol{T})\boldsymbol{\xi})^\wedge\right) = \boldsymbol{T}\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{T}^{-1}. \tag{11.5}$$

稍加改变，有：

$$\exp(\boldsymbol{\xi}^\wedge)\boldsymbol{T} = \boldsymbol{T}\exp\left(\left(\mathrm{Ad}(\boldsymbol{T}^{-1})\boldsymbol{\xi}\right)^\wedge\right). \tag{11.6}$$

该式表明，通过引入一个伴随项，我们能够“交换”扰动项左右侧的$\boldsymbol{T}$。利用它，可以将扰动挪到最右（当然最左亦可），导出右乘形式的雅可比矩阵（挪到左边时形成左乘）：

$$\begin{aligned}
\hat{\boldsymbol{e}}_{ij} &= \ln\left(\boldsymbol{T}_{ij}^{-1}\boldsymbol{T}_i^{-1}\exp((-\boldsymbol{\delta\xi}_i)^\wedge)\exp(\delta\boldsymbol{\xi}_j^\wedge)\boldsymbol{T}_j\right)^\vee \\
&= \ln\left(\boldsymbol{T}_{ij}^{-1}\boldsymbol{T}_i^{-1}\boldsymbol{T}_j\exp\left(\left(-\mathrm{Ad}(\boldsymbol{T}_j^{-1})\boldsymbol{\delta\xi}_i\right)^\wedge\right)\exp\left(\left(\mathrm{Ad}(\boldsymbol{T}_j^{-1})\boldsymbol{\delta\xi}_j\right)^\wedge\right)\right)^\vee \\
&\approx \ln\left(\boldsymbol{T}_{ij}^{-1}\boldsymbol{T}_i^{-1}\boldsymbol{T}_j\left[\boldsymbol{I} - (\mathrm{Ad}(\boldsymbol{T}_j^{-1})\boldsymbol{\delta\xi}_i)^\wedge + (\mathrm{Ad}(\boldsymbol{T}_j^{-1})\boldsymbol{\delta\xi}_j)^\wedge\right]\right)^\vee \\
&\approx \boldsymbol{e}_{ij} + \frac{\partial \boldsymbol{e}_{ij}}{\partial \boldsymbol{\delta\xi}_i}\boldsymbol{\delta\xi}_i + \frac{\partial \boldsymbol{e}_{ij}}{\partial \boldsymbol{\delta\xi}_j}\boldsymbol{\delta\xi}_j
\end{aligned}. \tag{11.7}$$

因此，按照李代数上的求导法则，我们求出了误差关于两个位姿的雅可比矩阵。关于$\boldsymbol{T}_i$的：

$$\frac{\partial \boldsymbol{e}_{ij}}{\partial \boldsymbol{\delta\xi}_i} = -\boldsymbol{\mathcal{J}}_r^{-1}(\boldsymbol{e}_{ij})\mathrm{Ad}(\boldsymbol{T}_j^{-1}). \tag{11.8}$$

以及关于$\boldsymbol{T}_j$的：

$$\frac{\partial \boldsymbol{e}_{ij}}{\partial \boldsymbol{\delta\xi}_j} = \boldsymbol{\mathcal{J}}_r^{-1}(\boldsymbol{e}_{ij})\mathrm{Ad}(\boldsymbol{T}_j^{-1}). \tag{11.9}$$

如果读者觉得这部分求导理解起来有困难，可以回到第4讲温习一

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

下李代数部分的内容。不过，前面也说过，由于se(3)上的左右雅可比J_r形式过于复杂，我们通常取它们的近似。如果误差接近于零，我们就可以设它们近似为I或

$$\boldsymbol{\mathcal{J}}_r^{-1}(\boldsymbol{e}_{ij}) \approx \boldsymbol{I} + \frac{1}{2}\begin{bmatrix} \boldsymbol{\phi}_{\boldsymbol{e}}^{\wedge} & \boldsymbol{\rho}_{\boldsymbol{e}}^{\wedge} \\ \boldsymbol{0} & \boldsymbol{\phi}_{\boldsymbol{e}}^{\wedge} \end{bmatrix}. \tag{11.10}$$

理论上讲，即使在优化之后，由于每条边给定的观测数据并不一致，误差通常也不见得近似于零，所以简单地把这里的J_r设置为I会有一定的损失。稍后我们将通过实践来看看理论上的区别是否明显。

了解雅可比求导后，剩下的部分就和普通的图优化一样了。简而言之，所有的位姿顶点和位姿——位姿边构成了一个图优化，本质上是一个最小二乘问题，优化变量为各个顶点的位姿，边来自于位姿观测约束。记ε为所有边的集合，那么总体目标函数为

$$\min_{\boldsymbol{\xi}} \frac{1}{2} \sum_{i,j \in \mathcal{E}} \boldsymbol{e}_{ij}^{\mathrm{T}} \boldsymbol{\Sigma}_{ij}^{-1} \boldsymbol{e}_{ij}. \tag{11.11}$$

我们依然可以用高斯牛顿法、列文伯格—马夸尔特方法等求解此问题，除了用李代数表示优化位姿以外，别的都是相似的。根据先前的经验，这自然可以用Ceres或g2o进行求解。我们不再讨论优化的详细过程，上一讲已经讲得够多了。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

11.2 实践：位姿图优化

11.2.1 g2o原生位姿图

下面演示使用g2o进行位姿图优化。首先，请读者用g2o_viewer打开我们预先生成的仿真位姿图，位于slambook/ch11/sphere.g2o中，如图11-2所示。

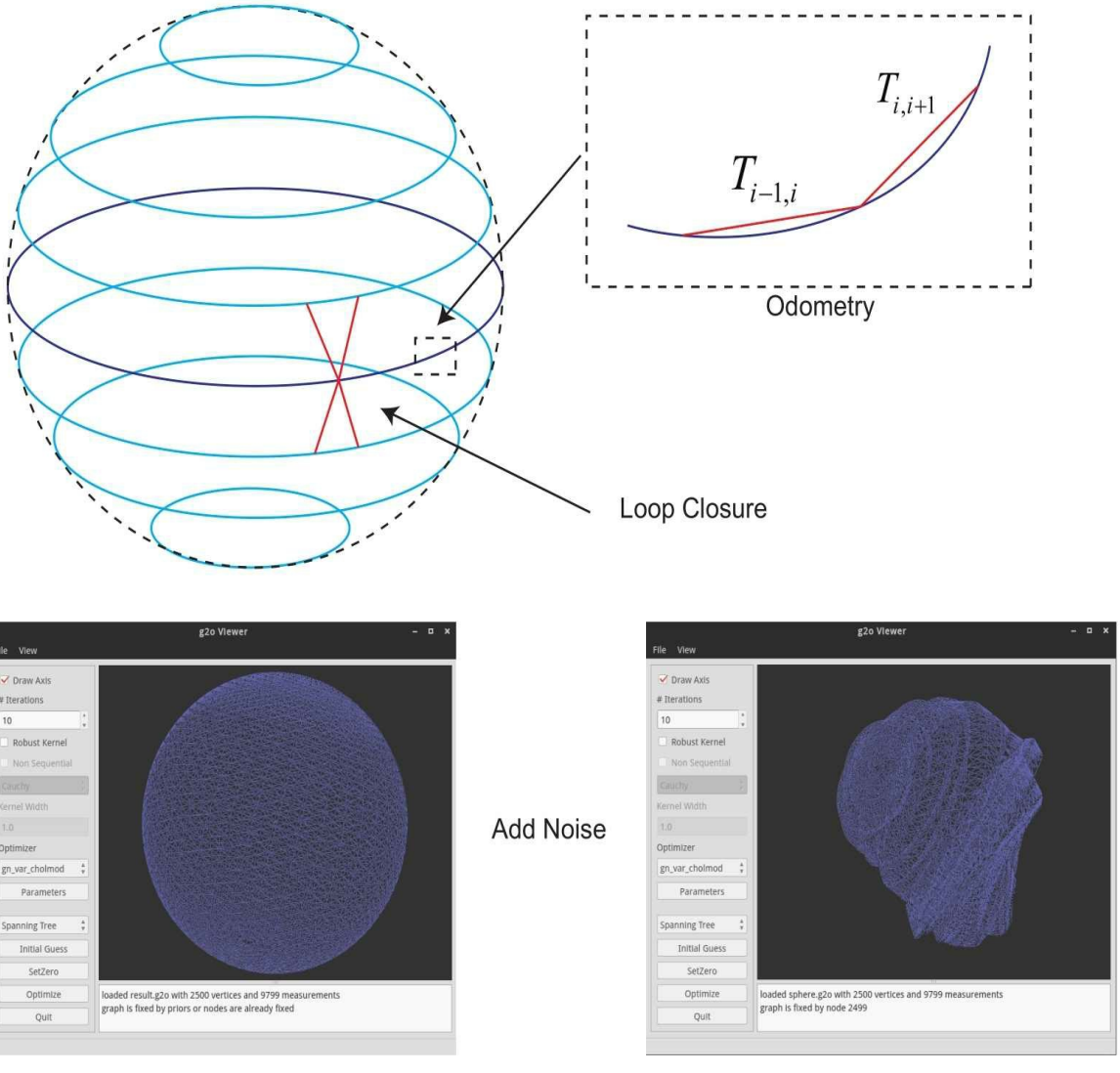


图11-2 g2o仿真产生的位姿图。真值是完整的球形，在真值上添加噪声后得到

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

带累计误差的仿真数据。

该位姿图是由g2o自带的create sphere程序仿真生成的。它的真实轨迹为一个球，由从下往上的多个层组成。每层为一个正圆形，很多个大小不一的圆形层组成了一个完整的球体，共包含2500个位姿节点（图11-2左上），可以看成一个转圈上升的过程。然后，仿真程序生成了t-1到t时刻的边，称为odometry边（里程计）。此外，又生成层与层之间的边，称为loop closure（回环，将在下一讲详细介绍）。随后，在每条边上添加观测噪声，并根据里程计边的噪声，重新设置节点的初始值。这样，就得到了带累积误差的位姿图数据（图11-2右下）。它局部看起来像球体的一部分，但整体形状与球体相差甚远。现在我们从这些带噪声的边和节点初始值出发，尝试优化整个位姿图，得到近似真值的数据。

当然，实际当中的机器人肯定不会出现这样正球形的运动轨迹，以及如此完整的里程计与回环观测数据。仿真成正球的好处是我们能够直观地看到优化结果是否正确（只要看它各个角度圆不圆就行了）。读者可以单击g2o_viewer中的optimize函数，看到每步的优化结果和收敛的过程。另一方面，sphere.g2o也是一个文本文件，可以用文本编辑器打开，查看它里面的内容。文件前半部分由节点组成，后半部分则是边：

```
1 VERTEX_SE3:QUAT 0 -0.125664 -1.53894e-17 99.9999 0.706662 4.32706e-17 0.707551
  -4.3325e-17
2 ......
3 EDGE_SE3:QUAT 1524 1574 -0.210399 -0.0101193 -6.28806 -0.00122939 0.0375067
  -2.85291e-05 0.999296 10000 0 0 0 0 0 10000 0 0 0 0 10000 0 0 0 40000 0 0 40000 0
  40000
```

可以看到，节点类型是VERTEX_SE3，表达一个相机位姿。g2o默认使用四元数和平移向量表达位姿，所以后面的字段意义为：ID，$t_x,t_y,t_z,q_x,q_y,q_z,q_w$。前3个为平移向量元素，后4个为表示旋转的单位四元数。同样，边的信息为两个节点的ID，$t_x,t_y,t_z,q_x,q_y,q_z,q_w$，信息矩阵的右上角（由于信息矩阵为对称阵，只需保存一半即可）。可以看到这里把信息矩阵设成了对角阵。

为了优化该位姿图，我们可以使用g2o默认的顶点和边，它们是由

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

四元数表示的。由于仿真数据也是g2o生成的，所以用g2o本身优化就无须我们多做什么工作了，只需配置一下优化参数即可。程序slambook/ch11/pose_graph_g2o_SE3.cpp演示了如何使用列文伯格—马夸尔特方法对该位姿图进行优化，并把结果存储至result.g2o文件中。

slambook/ch11/pose_graph_g2o_SE3.cpp

```
#include <iostream>
#include <fstream>
#include <string>

#include <g2o/types/slam3d/types_slam3d.h>
#include <g2o/core/block_solver.h>
#include <g2o/core/optimization_algorithm_levenberg.h>
#include <g2o/core/optimization_algorithm_gauss_newton.h>
#include <g2o/solvers/dense/linear_solver_dense.h>
#include <g2o/solvers/cholmod/linear_solver_cholmod.h>
using namespace std;

/************************************************
* 本程序演示如何用 g2o solver 进行位姿图优化
* sphere.g2o 是人工生成的一个 Pose graph ，我们来优化它。
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
* 尽管可以直接通过 load 函数读取整个图，但我们还是自己来实现读取代码，以期获得更深刻的理解。
* 这里使用 g2o/types/slam3d/ 中的 SE3 表示位姿，它实质上是四元数而非李代数。
* ********************************************/

int main( int argc, char** argv )
{
    if ( argc != 2 )
    {
        cout<<"Usage: pose_graph_g2o_SE3 sphere.g2o"<<endl;
        return 1;
    }
    ifstream fin( argv[1] );
    if ( !fin )
    {
        cout<<"file "<<argv[1]<<" does not exist."<<endl;
        return 1;
    }

    typedef g2o::BlockSolver<g2o::BlockSolverTraits<6,6>> Block;  // 6x6 BlockSolver
    Block::LinearSolverType* linearSolver = new g2o::LinearSolverCholmod<Block::PoseMatrixType>();
    Block* solver_ptr = new Block( linearSolver );      // 矩阵块求解器
    g2o::OptimizationAlgorithmLevenberg* solver=new g2o::OptimizationAlgorithmLevenberg(solver_ptr);
    g2o::SparseOptimizer optimizer;     // 图模型
    optimizer.setAlgorithm( solver );   // 设置求解器

    int vertexCnt = 0, edgeCnt = 0; // 顶点和边的数量
    while ( !fin.eof() )
    {
        string name;
        fin>>name;
        if ( name == "VERTEX_SE3:QUAT" )
        {
            // SE3 顶点
            g2o::VertexSE3* v = new g2o::VertexSE3();
            int index = 0;
            fin>>index;
            v->setId( index );
            v->read(fin);
            optimizer.addVertex(v);
            vertexCnt++;
            if ( index==0 )
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
57              v->setFixed(true);
58          }
59          else if ( name=="EDGE_SE3:QUAT" )
60          {
61              // SE3-SE3 边
62              g2o::EdgeSE3* e = new g2o::EdgeSE3();
63              int idx1, idx2;     // 关联的两个顶点
64              fin>>idx1>>idx2;
65              e->setId( edgeCnt++ );
66              e->setVertex( 0, optimizer.vertices()[idx1] );
67              e->setVertex( 1, optimizer.vertices()[idx2] );
68              e->read(fin);
69              optimizer.addEdge(e);
70          }
71          if ( !fin.good() ) break;
72      }
73
74      cout<<"read total "<<vertexCnt<<" vertices, "<<edgeCnt<<" edges."<<endl;
75
76      cout<<"prepare optimizing ..."<<endl;
77      optimizer.setVerbose(true);
78      optimizer.initializeOptimization();
79      cout<<"calling optimizing ..."<<endl;
80      optimizer.optimize(30);
81
82      cout<<"saving optimization results ..."<<endl;
83      optimizer.save("result.g2o");
84
85      return 0;
86  }
```

我们选择了6×6的块求解器，使用列文伯格—马夸尔特下降方式，迭代次数选择30次。调用此程序对位姿图进行优化：

```
$ build/pose_graph_g2o_SE3 sphere.g2o
read total 2500 vertices, 9799 edges.
prepare optimizing ...
calling optimizing ...
iteration= 0  chi2= 1023011093.967638 time= 0.851879 cumTime= 0.851879 edges= 9799
schur= 0 lambda= 805.622433 levenbergIter= 1
iteration= 1  chi2= 385118688.233188  time= 0.863567 cumTime= 1.71545  edges= 9799
schur= 0 lambda= 537.081622 levenbergIter= 1
iteration= 2  chi2= 166223726.693659  time= 0.861235 cumTime= 2.57668  edges= 9799
schur= 0 lambda= 358.054415 levenbergIter= 1
iteration= 3  chi2= 86610874.269316   time= 0.844105 cumTime= 3.42079  edges= 9799
schur= 0 lambda= 238.702943 levenbergIter= 1
iteration= 4  chi2= 40582782.710190   time= 0.862221 cumTime= 4.28301  edges= 9799
schur= 0 lambda= 159.135295 levenbergIter= 1
......
iteration= 28 chi2= 45095.174398 time= 0.869451 cumTime= 30.0809 edges= 9799 schur=
 0 lambda= 0.003127 levenbergIter= 1
iteration= 29 chi2= 44811.248504 time= 1.76326  cumTime= 31.8442 edges= 9799 schur=
 0 lambda= 0.003785 levenbergIter= 2
saving optimization results ...
```

然后，用g2o_viewer打开result.g2o查看结果，如图11-3所示。

书籍汇总下载 http://t.cn/RDIAj5D

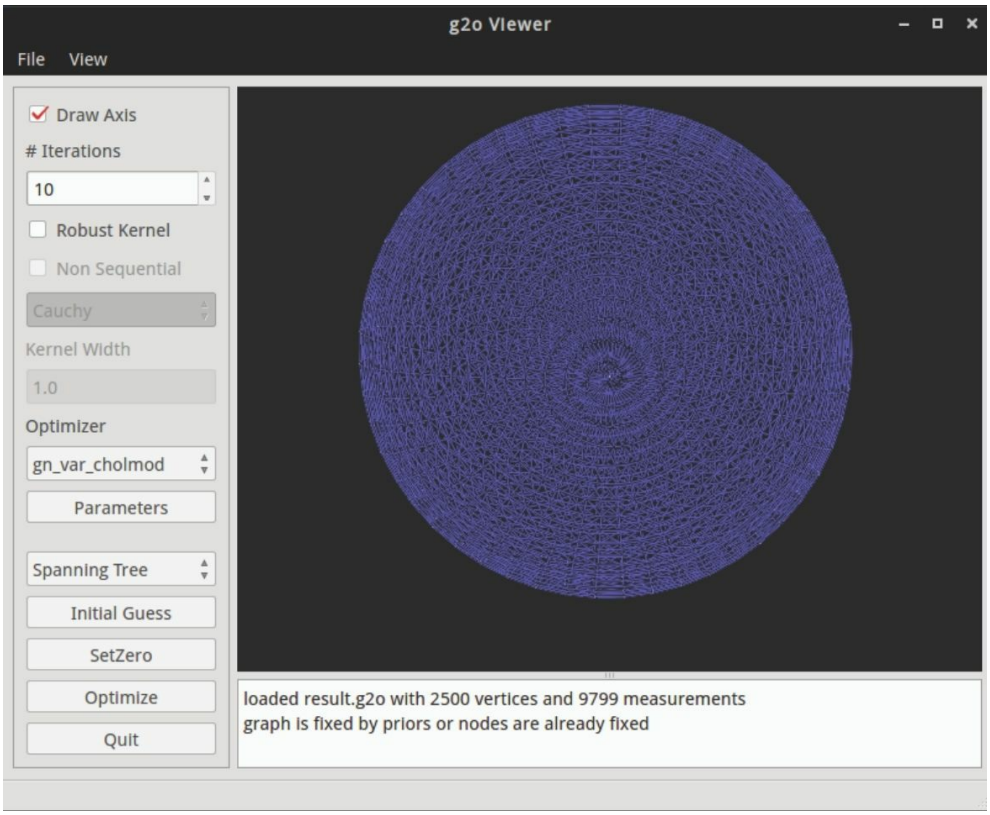


图11-3　使用g2o自带的顶点与边求解的结果。

结果从一个不规则的形状优化成了一个看起来完整的球。这个过程实质上和我们单击g2o_viewer上的Optimize按钮没有区别。下面，我们根据前面的李代数推导来实现一下李代数上的优化。

11.2.2　李代数上的位姿图优化

还记得我们用Sophus来表达李代数的事情吗？我们来试试把Sophus用到g2o中定义自己的顶点和边吧。

slambook/ch11/pose_graph_g2o_lie_algebra.cpp（片段）

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <Eigen/Core>

#include <g2o/core/base_vertex.h>
#include <g2o/core/base_binary_edge.h>
#include <g2o/core/block_solver.h>
#include <g2o/core/optimization_algorithm_levenberg.h>
#include <g2o/core/optimization_algorithm_gauss_newton.h>
#include <g2o/core/optimization_algorithm_dogleg.h>
#include <g2o/solvers/dense/linear_solver_dense.h>
#include <g2o/solvers/cholmod/linear_solver_cholmod.h>

#include <sophus/se3.h>
#include <sophus/so3.h>
using namespace std;
using Sophus::SE3;
using Sophus::SO3;

/**********************************************
* 本程序演示如何用 g2o solver 进行位姿图优化
* sphere.g2o 是人工生成的一个 Pose graph ，我们来优化它。
* 尽管可以直接通过 load 函数读取整个图，但我们还是自己来实现读取代码，以期获得更深刻的理解。
* 本节使用李代数表达位姿图，节点和边的方式为自定义。
* **********************************************/

typedef Eigen::Matrix<double,6,6> Matrix6d;

// 给定误差求 J_R^{-1} 的近似
Matrix6d JRInv( SE3 e )
{
    Matrix6d J;
    J.block(0,0,3,3) = SO3::hat(e.so3().log());
    J.block(0,3,3,3) = SO3::hat(e.translation());
    J.block(3,0,3,3) = Eigen::Matrix3d::Zero(3,3);
    J.block(3,3,3,3) = SO3::hat(e.so3().log());
    J = J*0.5 + Matrix6d::Identity();
    return J;
}
// 李代数顶点
typedef Eigen::Matrix<double, 6, 1> Vector6d;
class VertexSE3LieAlgebra: public g2o::BaseVertex<6, SE3>
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
{
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
    bool read ( istream& is )
    {
        double data[7];
        for ( int i=0; i<7; i++ )
        is>>data[i];
        setEstimate ( SE3 (
            Eigen::Quaterniond ( data[6],data[3], data[4], data[5] ),
            Eigen::Vector3d ( data[0], data[1], data[2] )
        ));
    }

    bool write ( ostream& os ) const
    {
        os<<id()<<" ";
        Eigen::Quaterniond q = _estimate.unit_quaternion();
        os<<_estimate.translation().transpose()<<" ";
        os<<q.coeffs()[0]<<" "<<q.coeffs()[1]<<" "<<q.coeffs()[2]<<" "<<q.coeffs()
        [3]<<endl;
        return true;
    }

    virtual void setToOriginImpl()
    {
        _estimate = Sophus::SE3();
    }
    // 左乘更新
    virtual void oplusImpl ( const double* update )
    {
        Sophus::SE3 up (
            Sophus::SO3 ( update[3], update[4], update[5] ),
            Eigen::Vector3d ( update[0], update[1], update[2] )
        );
        _estimate = up*_estimate;
    }
};

// 两个李代数节点之边
class EdgeSE3LieAlgebra: public g2o::BaseBinaryEdge<6, SE3, VertexSE3LieAlgebra,
VertexSE3LieAlgebra>
{
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
87      bool read ( istream& is )
88      {
89          double data[7];
90          for ( int i=0; i<7; i++ )
91          is>>data[i];
92          Eigen::Quaterniond q ( data[6], data[3], data[4], data[5] );
93          q.normalize();
94          setMeasurement (
95              Sophus::SE3 ( q, Eigen::Vector3d ( data[0], data[1], data[2] ) )
96          );
97          for ( int i=0; i<information().rows() && is.good(); i++ )
98              for ( int j=i; j<information().cols() && is.good(); j++ )
99              {
100                 is >> information() ( i,j );
101                 if ( i!=j )
102                     information() ( j,i ) =information() ( i,j );
103             }
104         return true;
105     }
106     bool write ( ostream& os ) const
107     {
108         VertexSE3LieAlgebra* v1 = static_cast<VertexSE3LieAlgebra*> (_vertices[0]);
109         VertexSE3LieAlgebra* v2 = static_cast<VertexSE3LieAlgebra*> (_vertices[1]);
110         os<<v1->id()<<" "<<v2->id()<<" ";
111         SE3 m = _measurement;
112         Eigen::Quaterniond q = m.unit_quaternion();
113         os<<m.translation().transpose()<<" ";
114         os<<q.coeffs()[0]<<" "<<q.coeffs()[1]<<" "<<q.coeffs()[2]<<" "<<q.coeffs()
            [3]<<" ";
115         // information matrix
116         for ( int i=0; i<information().rows(); i++ )
117             for ( int j=i; j<information().cols(); j++ )
118             {
119                 os << information() ( i,j ) << " ";
120             }
121         os<<endl;
122         return true;
123     }
124
125     // 误差计算与书中推导一致
126     virtual void computeError()
127     {
128         Sophus::SE3 v1 = (static_cast<VertexSE3LieAlgebra*> (_vertices[0]))->
            estimate();
129         Sophus::SE3 v2 = (static_cast<VertexSE3LieAlgebra*> (_vertices[1]))->
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
        estimate();
130     _error = (_measurement.inverse()*v1.inverse()*v2).log();
131 }
132
133 // 雅可比计算
134 virtual void linearizeOplus()
135 {
136     Sophus::SE3 v1 = (static_cast<VertexSE3LieAlgebra*> (_vertices[0]))->
        estimate();
137     Sophus::SE3 v2 = (static_cast<VertexSE3LieAlgebra*> (_vertices[1]))->
        estimate();
138     Matrix6d J = JRInv(SE3::exp(_error));
139     // 尝试把J近似为I?
140     _jacobianOplusXi = - J* v2.inverse().Adj();
141     _jacobianOplusXj = J*v2.inverse().Adj();
142 }
143 };
```

为了实现对g2o文件的存储和读取，本节例程实现了read和write函数，并且“伪装”成g2o内置的SE3顶点，使得g2o_viewer能够认识并渲染它。事实上，除了内部使用Sophus的李代数表示之外，从外部看起来没有什么区别。

值得注意的是这里雅可比的计算过程。我们有若干种选择：一是不提供雅可比计算函数，让g2o自动计算数值雅可比。二是提供完整或近似的雅可比计算过程。这里我们用JRInv()函数提供近似的$\mathcal{J}_r^{-1}$。读者可以尝试把它近似为I，或者干脆注释掉oplusImpl函数，看看结果会有什么区别。

之后调用g2o进行优化问题：

书籍汇总下载 http://t.cn/RDIAj5D

```
$ build/pose_graph_g2o_lie sphere.g2o
read total 2500 vertices, 9799 edges.
prepare optimizing ...
calling optimizing ...
iteration= 0  chi2= 781963143.389706 time= 1.9322  cumTime= 1.9322  edges= 9799
schur= 0 lambda= 6706.585223 levenbergIter= 1
iteration= 1  chi2= 236521032.458035 time= 1.91309 cumTime= 3.84529 edges= 9799
schur= 0 lambda= 2235.528408 levenbergIter= 1
iteration= 2  chi2= 142934798.398778 time= 1.9792  cumTime= 5.8245  edges= 9799
schur= 0 lambda= 745.176136  levenbergIter= 1
iteration= 3  chi2= 84490229.050137  time= 2.03394 cumTime= 7.85844 edges= 9799
schur= 0 lambda= 248.392045  levenbergIter= 1
iteration= 4  chi2= 42690811.624643  time= 2.05149 cumTime= 9.90993 edges= 9799
schur= 0 lambda= 82.797348   levenbergIter= 1
......
iteration= 21 chi2= 127607.121623    time= 1.9686  cumTime= 43.526  edges= 9799
schur= 0 lambda= 0.000712    levenbergIter= 1
```

```
iteration= 22 chi2= 127578.889888    time= 1.94773 cumTime= 45.4737 edges= 9799
schur= 0 lambda= 0.000237    levenbergIter= 1
iteration= 23 chi2= 127578.158794    time= 1.98009 cumTime= 47.4538 edges= 9799
schur= 0 lambda= 0.000079    levenbergIter= 1
iteration= 24 chi2= 127578.157859    time= 2.04546 cumTime= 49.4993 edges= 9799
schur= 0 lambda= 0.000053    levenbergIter= 1
iteration= 25 chi2= 127578.157859    time= 1.96722 cumTime= 51.4665 edges= 9799
schur= 0 lambda= 0.000035    levenbergIter= 1
iteration= 26 chi2= 127578.157859    time= 2.11235 cumTime= 53.5789 edges= 9799
schur= 0 lambda= 0.000023    levenbergIter= 1
iteration= 27 chi2= 127578.157859    time= 3.29151 cumTime= 56.8704 edges= 9799
schur= 0 lambda= 0.000031    levenbergIter= 2
iteration= 28 chi2= 127578.157859    time= 3.20302 cumTime= 60.0734 edges= 9799
schur= 0 lambda= 0.000042    levenbergIter= 2
iteration= 29 chi2= 127578.157859    time= 5.56337 cumTime= 65.6368 edges= 9799
schur= 0 lambda= 0.001779    levenbergIter= 4
saving optimization results ...
```

我们发现，迭代23次后，总体误差保持不变，事实上可以让优化算法停止了。而上一个实验中，用满了30次迭代后误差仍在下降[1]。在调

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

用优化后，查看result_lie.g2o观察它的结果，如图11-4所示。从肉眼上看不出任何区别。

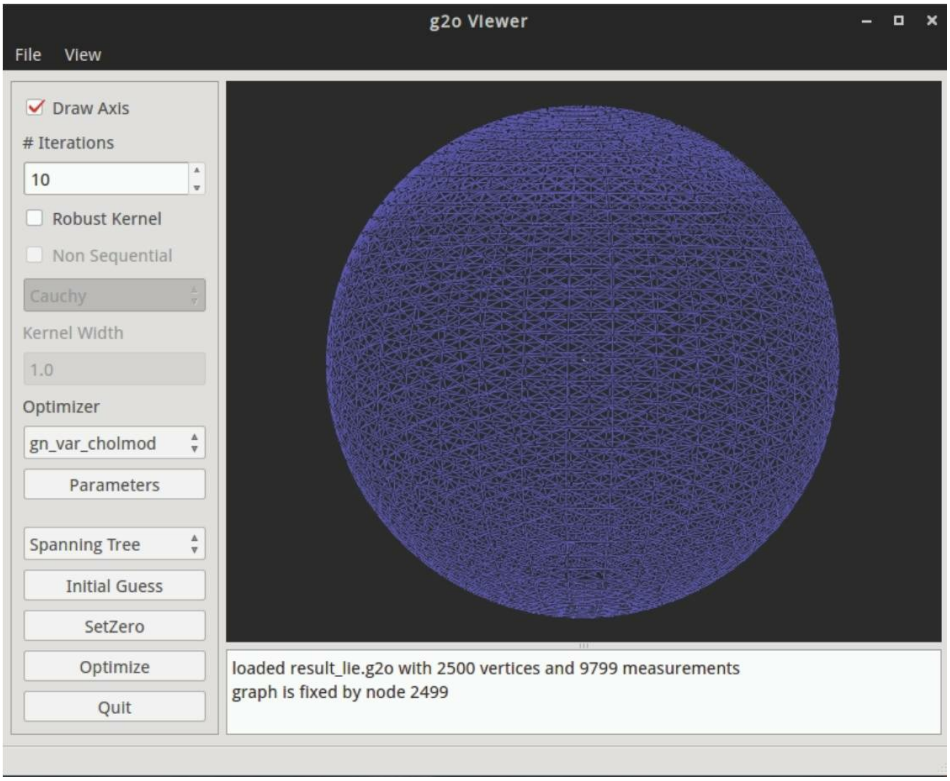


图11-4　使用李代数自定义节点与边优化后的结果。

如果你在这个g2o_viewer界面按下Optimize按钮，g2o将使用它自带的SE3顶点进行优化，你可以在下方文本框中看到：

```
loaded result_lie.g2o with 2500 vertices and 9799 measurements
graph is fixed by node 2499
# Using CHOLMOD poseDim -1 landMarkDim -1 blockordering 0
Preparing (no marginalization of Landmarks)
iteration= 0 chi2= 44360.509723 time= 0.567504 cumTime= 0.567504 edges= 9799 schur=
 0
iteration= 1 chi2= 44360.471110 time= 0.595993 cumTime= 1.1635   edges= 9799 schur=
 0
iteration= 2 chi2= 44360.471110 time= 0.582909 cumTime= 1.74641  edges= 9799 schur=
 0
```

整体误差在SE3边的度量下为44360，略小于之前30次迭代时的

44811。这说明使用李代数进行优化后，我们在更少的迭代次数下得到了更好的结果[2]。

11.2.3 小结

球的例子是一个比较有代表性的案例。它具有和实际中相似的里程计边（Odometry）和回环边（LoopClosure），这也正是实际SLAM中一个位姿图中可能有的东西。同时，“球”也具有一定的计算规模：它总共有2,500个位姿节点和近10,000条边，我们发现优化它费了不少时间（相对于实时性要求很强的前端来说）。另一方面，一般认为位姿图是结构最简单的图之一。在我们不假设机器人如何运动的前提下，很难再进一步讨论它的稀疏性了——因为机器人可能会直线往前运动，形成带状的位姿图，是稀疏的；也可能是“左手右手一个慢动作”，形成大量的小型回环需要优化（Loopy motion），从而变成像“球”那样比较稠密的位姿图。无论如何，在没有进一步的信息之前，我们似乎无法再利用位姿图的求解结构了。

自PTAM[81]提出以来，人们就意识到，后端的优化没必要实时地响应前端的图像数据。人们倾向于把前端和后端分开，运行于两个独立线程之中，历史上称为跟踪（Tracking）和建图（Mapping）——虽然如此叫，建图部分主要是指后端的优化内容。通俗地说，前端需要实时响应视频的速度，例如每秒30帧；而优化可以慢悠悠地运行，只要在优化完成时把结果返回给前端即可。所以我们通常不会对后端优化提出很高的速度要求。

书籍汇总下载 http://t.cn/RDIAj5D

11.3 *因子图优化初步

11.3.1 贝叶斯网络

下面，我们从另一个角度来看后端优化：所谓的因子图（FactorGraph）优化。由于这部分内容牵涉到概率图理论，超出了本书范围，所以我们只能大概地介绍如何从概率图角度来看待这个问题。如果读者对概率图模型感兴趣，建议阅读文献[82]。或者，如果对如何使用因子图实现SLAM后端优化的细节感兴趣，可以阅读文献[83,84,85,86]。

从贝叶斯网络（Bayes Network）的角度来看，SLAM可以自然地表达成一个动态贝叶斯网络（Dynamic Bayes Network，DBN）。与图优化类似，贝叶斯网络是一种概率图，由随机变量的节点和表达随机变量条件独立性的边组成，形成一个有向无环图（Directed Acyclic Graph）。在SLAM中，由于我们有运动方程和观测方程，它们恰好表示了状态变量之间的条件概率，如图11-5所示。

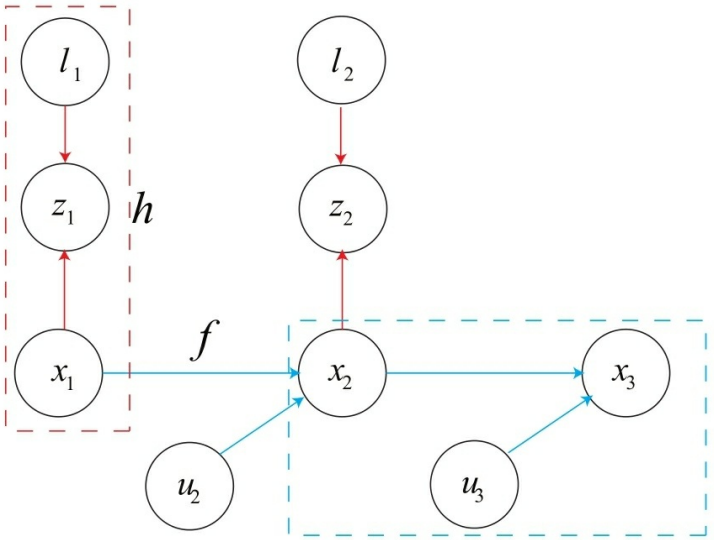


图11-5 以贝叶斯网络形式表达的SLAM过程示意图。竖线为运动方程，其他线为观测方程。虚线框表示一次观测，实线框表示一次运动。

图11-5中的圆圈表示了贝叶斯网络的节点，也就是与图相关的随机

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

变量，包括：

1.相机位姿形成的节点：x_1,x_2…。

2.输入量节点：u。

3.路标节点：l。

4.观测数据节点：z。

这组成了所有与SLAM过程相关的信息。另一方面，各线条表示了它们之间的关系，箭头表示依赖关系。例如，从x_1指向x_2的箭头，说明x_2依赖于x_1，此边表示的概率是$P(x_2|x_1)$——事实上，运动方程指出了从x_1到x_2的运动关系。我们观察实线框。这个框中，变量x_3依赖于x_2,u_3。回忆运动方程：

$$\boldsymbol{x}_3 = f(\boldsymbol{x}_2, \boldsymbol{u}_3) + \boldsymbol{w}_3. \tag{11.12}$$

它实际上给出了这几个变量的条件概率的度量：

$$P(\boldsymbol{x}_3|\boldsymbol{x}_2, \boldsymbol{u}_3). \tag{11.13}$$

同样，虚线框中的一次观测，亦说明观测方程给出了变量间的条件概率关系：

$$P(\boldsymbol{z}_1|\boldsymbol{x}_1, \boldsymbol{l}_1). \tag{11.14}$$

通过这种方式，我们构建了一个贝叶斯网络，它表达了所有变量，以及各个方程给出的变量之间的条件概率关系。请注意，这只是说贝叶斯网络表达了它们，还没有说到贝叶斯网络的求解。事实上，后端优化的目标，就是在这些既有的约束之上，通过调整贝叶斯网络中随机变量的取值，使整个后验概率达到最大：

$$\{\boldsymbol{x}, \boldsymbol{l}\}^* = \arg\max(\boldsymbol{x}_0) \prod P(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \prod P(\boldsymbol{z}_k|\boldsymbol{x}_i, \boldsymbol{l}_j). \tag{11.15}$$

直到现在为止，这和我们在图优化中谈论的东西并没有太大区别，我们只是把变量间的关系显式地用有向图给表达出来而已。这样一个由条件概率描述的贝叶斯网络，已经可以使用概率图模型中的算法进行求

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

解了。不过，进一步观察，我们发现最大后验概率由许多项因子乘积而成，因此该贝叶斯网络又可以转化成一个因子图（Factor Graph）。

11.3.2 因子图

因子图是一种无向图，由两种节点组成：表示优化变量的变量节点，以及表示因子的因子节点。如果我们把图11-5表示成因子图，就可画成如图11-6所示的样子。

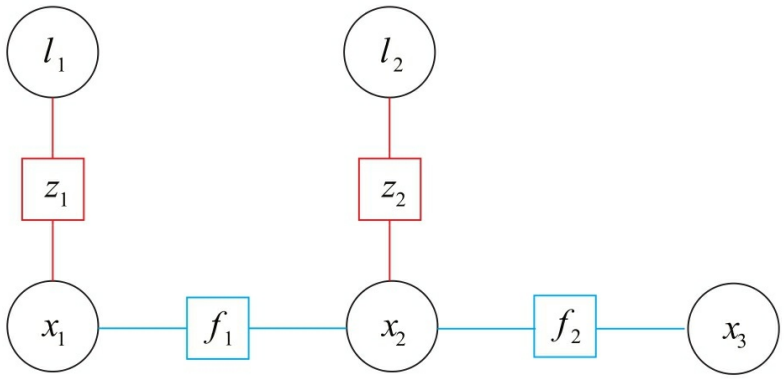


图11-6　以因子图形式表达的SLAM过程示意图。圆圈为变量节点，方块为因子节点。

在图11-6中，我们用圆圈表示变量节点。注意与贝叶斯网络不同，这里的变量是SLAM中待优化的部分，即相机位姿x和路标l，而没有观测z和输入u——因为这几个量是给定的而不是待优化的。与之相对，因子节点包含了待优化变量之间的关系，它们来自运动方程和观测方程。

对因子图的优化，就是调整各变量的值，使它们的因子之乘积最大化——它依然对应着一个优化问题。在通常的做法中，我们把各因子的条件概率取高斯分布的形式。考虑到运动方程为

$$\boldsymbol{x}_k = f\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k\right) + \boldsymbol{w}_k. \tag{11.16}$$

其中$w_k \sim N(0, R_k)$。如果x_{k-1}（实际上x_{k-1}的取值还受其他因素影响）已知，那么：

$$P\left(\boldsymbol{x}_k | \boldsymbol{x}_{k-1}\right) = N\left(f\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k\right), \boldsymbol{R}_k\right). \tag{11.17}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

同理，对于观测数据，有：

$$P(\boldsymbol{z}_{kj}|\boldsymbol{x}_k, \boldsymbol{l}_j) = N(h(\boldsymbol{x}_k, \boldsymbol{l}_j), \boldsymbol{Q}_{kj}), \tag{11.18}$$

其中Q_{kj}为观测方程噪声项的协方差。

通过假设高斯分布，我们显式地表达了因子图优化的目标函数。和图优化一样，由于取最大的后验概率相当于取负对数的最小化，所以因子图优化也对应着一个和第6讲介绍的相似的最小二乘问题，我们可以用高斯牛顿法或列文伯格—马夸尔特方法求解一个因子图优化问题。类似于图优化，我们还可以使用单元的、二元的或多元的因子。这主要看它和几个变量节点有关。

图11-7是一个更实际的因子图的例子。

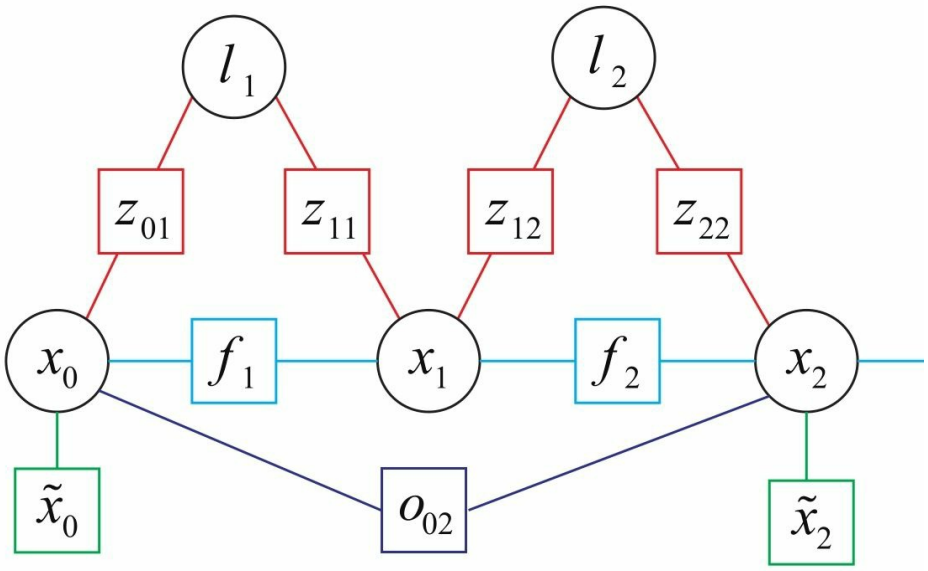


图11-7　更加实际的因子图。添加了回环因子（底部中央）和先验因子（底部两侧）。

运动方程和观测方程作为因子存在于图中。此外，我们可能对某些相机位姿具有先验信息——例如一辆在室外行驶的无人车，我们可能通过GPS信号确定了轨迹当中某些节点的位姿，那么就可以对这些位姿添加先验因子。因为可能表达成概率分布的信息有许多种，于是因子图也可以定义许多不同的因子，比如，轮式编码器的测量、IMU的测量，等等，使之成为一种非常通用的优化方式。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

11.3.3 增量特性

然而，到现在为止，从优化角度来看，优化一个因子图和普通的图优化并没有太大的区别——因为我们最后面对的都是一个最小二乘问题，不断地寻找梯度，使目标函数下降。因子图优化的稀疏性也与图优化类似，我们可以通过稀疏QR分解、Schur补或Cholesky分解，加速对因子图优化的求解。所以我们不禁疑惑：因子图优化是否就是普通图优化换了种说法呢？事情并不完全是这样。

Kaess等人提出的iSAM（incremental Smooth and Mapping）[83]中，对因子图进行了更加精细的处理，使得它可以增量式地处理后端优化。回忆第6讲的内容，我们知道，在普通的图优化中，最终要计算的是一个增量式方程。即如何调整目标函数

$$J(\boldsymbol{x}) = \sum_k \boldsymbol{e}_{v,k}^{\mathrm{T}} \boldsymbol{R}_k^{-1} \boldsymbol{e}_{v,k} + \sum_k \sum_j \boldsymbol{e}_{y,k,j}^{\mathrm{T}} \boldsymbol{Q}_{k,j}^{-1} \boldsymbol{e}_{y,k,j} \tag{11.19}$$

中的优化变量，使目标函数下降。无论采用哪种梯度下降策略，最后我们将碰到一个形如

$$\boldsymbol{H}\Delta\boldsymbol{x} = \boldsymbol{g}. \tag{11.20}$$

的线性方程需要求解。上一讲，我们介绍了利用该方程中的稀疏性加速它的求解过程。然而，考虑到图优化并不是固定的，当机器人运动时，新的节点和边将被加入图中，使得它的规模不断增长。那么问题来了：是否每次新加一个节点，我们就要重新计算一遍所有节点的更新量呢？（包括雅可比的求取〔或称线性化〕和更新方程的计算。）

显然，这样做是不经济的，我们以图11-8为例来解释增量更新的情况。这是一个位姿图。当我们按照里程计方式向其中添加节点时，受影响的节点可以近似地看成只有最后一个与之相连的节点，而早先节点的估计值，可以近似地看成没有发生变化。因此就没有必要对它们进行优化。为什么要说近似呢？因为实际上新增节点还是会对之前的估计产生影响的，只是对最近的数据影响最大，对较远的数据影响很小，可以忽略掉。如果认为增量特性可行，那么这件事情可以为我们省掉大量的

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

计算——至少我们不必在每次新增节点时，对整个图进行优化。不过，如果按照回环检测的方式添加节点，那么受影响的范围应该是回环开始到当前帧这一段中的所有节点，也就是整段轨迹都可能被重新调整。这虽然加大了计算量，然而我们依然无须优化整张图，因为回环之外的节点是不受影响的。综上所述，我们发现，在向图中添加节点时，通过分析受影响的区域，可以（直观上）减少一些没必要的计算，加速后端优化流程。

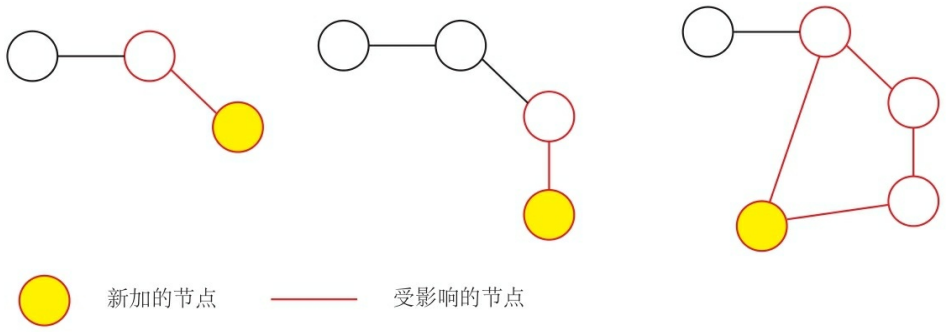


图11-8　增量更新示意图。

在此思想的基础上，Kaess等人提出的增量式因子图优化一定程度上解决了上面的问题。从技术层面上来看，我们希望在每次优化中保存一些中间结果；而当新的变量和因子加入时，首先分析它们因子图之间的连接和影响关系，考虑之前存储的信息有哪些可以继续利用，哪些必须重新计算，最后处理对增量的优化。话虽如此，由于具体的操作步骤需要介绍大量的技术细节，且牵涉到了概率图理论的知识，超出了本书的讨论范围。所以本节就作为可选阅读材料提供给读者。有兴趣的读者可以阅读iSAM、iSAM2等原始论文来了解它的细节处理。最后，尽管有增量分析，但我们必须清楚这里的受影响节点亦是近似的，所以在实际操作中，当图的规模发生一定程度的改变时，我们需要再做一次全图优化。

书籍汇总下载 http://t.cn/RDIAj5D

11.4 *实践：gtsam

11.4.1 安装gtsam 4.0

下面，我们演示一个用因子图进行位姿图优化的例子。我们仍使用前面“球”的数据，不同的是，这次用因子图来优化它，而不是g2o中的位姿图。我们将使用Gtsam[87]，它是一个基于因子图优化的SLAM后端库，理论来自文献[83,84]。我们将在它的基础上，对上节“球”的例子进行优化。

```
git clone https://bitbucket.org/gtborg/gtsam.git
```

gtsam最新版本为4.0，位于https://bitbucket.org/gtborg/gtsam。你可以输入以下命令下载它：由于gtsam比较大，我们没有在3rdparty文件夹中提供。与以往遇到的库一样，gtsam亦是一个cmake工程，我们按照cmake的方式来编译安装它。gtsam的依赖项比较少，主要是Eigen和tbb库。如果读者跟着本书一路走来，那么大多数库应该已经安装好了，只需安装tbb库即可：

```
sudo apt-get install libtbb-dev
```

然后，使用cmake命令对gtsam进行编译安装，这里不再赘述。gtsam库比较大，集成了许多与因子图相关的内容，甚至底层的矩阵、李代数运算，所以编译安装会比较费时，请读者耐心等待。安装完成后，可以在/usr/local/include中找到头文件，在/usr/local/lib下找到库文件。gtsam的库文件比较简单，仅由一个libgtsam.so组成。所以你应该知道如何在自己的工程中书写CMakeLists.txt来调用gtsam了吧？

11.4.2 位姿图优化

下面我们来演示“球”的例子。与前面的实验一样，我们从

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

sphere.g2o文件中读取节点和边的信息，转换为因子图交给gtsam处理，然后，把优化结果重新写到g2o文件，并“伪装”成g2o的节点和边以便显示。虽然这似乎没有体现出gtsam的“增量”特性，但至少我们可以通过这个例子体验一下它的用法。

slambook/ch11/pose_graph_gtsam.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <Eigen/Core>

#include <sophus/se3.h>
#include <sophus/so3.h>

#include <gtsam/slam/dataset.h>
#include <gtsam/slam/BetweenFactor.h>
#include <gtsam/slam/PriorFactor.h>
#include <gtsam/nonlinear/GaussNewtonOptimizer.h>
#include <gtsam/nonlinear/LevenbergMarquardtOptimizer.h>

using namespace std;
using Sophus::SE3;
using Sophus::SO3;

/************************************************
* 本程序演示如何用 gtsam 进行位姿图优化
* sphere.g2o 是人工生成的一个 Pose graph，我们来优化它。
* 与 g2o 相似，在 gtsam 中添加的是因子，相当于误差。
* **********************************************/

int main ( int argc, char** argv )
{
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
27      if ( argc != 2 )
28      {
29          cout<<"Usage: pose_graph_gtsam sphere.g2o"<<endl;
30          return 1;
31      }
32      ifstream fin ( argv[1] );
33      if ( !fin )
34      {
35          cout<<"file "<<argv[1]<<" does not exist."<<endl;
36          return 1;
37      }
38
39      gtsam::NonlinearFactorGraph::shared_ptr graph ( new gtsam::NonlinearFactorGraph
        );  // gtsam的因子图
40      gtsam::Values::shared_ptr initial ( new gtsam::Values ); // 初始值
41      // 从g2o文件中读取节点和边的信息
42      int cntVertex=0, cntEdge = 0;
43      cout<<"reading from g2o file"<<endl;
44
45      while ( !fin.eof() )
46      {
47          string tag;
48          fin>>tag;
49          if ( tag == "VERTEX_SE3:QUAT" )
50          {
51              // 顶点
52              gtsam::Key id;
53              fin>>id;
54              double data[7];
55              for ( int i=0; i<7; i++ ) fin>>data[i];
56              // 转换至gtsam的Pose3
57              gtsam::Rot3 R = gtsam::Rot3::Quaternion ( data[6], data[3], data[4],
                data[5] );
58              gtsam::Point3 t ( data[0], data[1], data[2] );
59              initial->insert ( id, gtsam::Pose3 ( R,t ) );       // 添加初始值
60              cntVertex++;
61          }
62          else if ( tag == "EDGE_SE3:QUAT" )
63          {
64              // 边，对应到因子图中的因子
65              gtsam::Matrix m = gtsam::I_6x6;     // 信息矩阵
66              gtsam::Key id1, id2;
67              fin>>id1>>id2;
68              double data[7];
69              for ( int i=0; i<7; i++ ) fin>>data[i];
```

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
70              gtsam::Rot3 R = gtsam::Rot3::Quaternion ( data[6], data[3], data[4],
                data[5] );
71              gtsam::Point3 t ( data[0], data[1], data[2] );
72              for ( int i=0; i<6; i++ )
73                  for ( int j=i; j<6; j++ )
74                  {
75                      double mij;
76                      fin>>mij;
77                      m ( i,j ) = mij;
78                      m ( j,i ) = mij;
79                  }
80
81              // g2o 的信息矩阵定义方式与 gtsam 不同，这里对它进行修改
82              gtsam::Matrix mgtsam = gtsam::I_6x6;
83              mgtsam.block<3,3> ( 0,0 ) = m.block<3,3> ( 3,3 ); // cov rotation
84              mgtsam.block<3,3> ( 3,3 ) = m.block<3,3> ( 0,0 ); // cov translation
85              mgtsam.block<3,3> ( 0,3 ) = m.block<3,3> ( 0,3 ); // off diagonal
86              mgtsam.block<3,3> ( 3,0 ) = m.block<3,3> ( 3,0 ); // off diagonal
87
88              // 高斯噪声模型
89              gtsam::SharedNoiseModel model = gtsam::noiseModel::Gaussian::
                Information ( mgtsam );
90              gtsam::NonlinearFactor::shared_ptr factor (
91                  new gtsam::BetweenFactor<gtsam::Pose3> ( id1, id2, gtsam::Pose3 ( R
                    ,t ), model ) // 添加一个因子
92              );
93              graph->push_back ( factor );
94              cntEdge++;
95          }
96          if ( !fin.good() )
97              break;
98      }
99
100     cout<<"read total "<<cntVertex<<" vertices, "<<cntEdge<<" edges."<<endl;
101     // 固定第一个顶点，在 gtsam 中相当于添加一个先验因子
102     gtsam::NonlinearFactorGraph graphWithPrior = *graph;
103     gtsam::noiseModel::Diagonal::shared_ptr priorModel =
104     gtsam::noiseModel::Diagonal::Variances (
105         ( gtsam::Vector ( 6 ) <<1e-6, 1e-6, 1e-6, 1e-6, 1e-6, 1e-6 ).finished()
106     );
107     gtsam::Key firstKey = 0;
108     for ( const gtsam::Values::ConstKeyValuePair& key_value: *initial )
109     {
110         cout<<"Adding prior to g2o file "<<endl;
111         graphWithPrior.add ( gtsam::PriorFactor<gtsam::Pose3> (
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
            key_value.key, key_value.value.cast<gtsam::Pose3>(), priorModel )
        );
        break;
    }

    // 开始因子图优化，配置优化选项
    cout<<"optimizing the factor graph"<<endl;
    // 我们使用列文伯格—马夸尔特方法优化
    gtsam::LevenbergMarquardtParams params_lm;
    params_lm.setVerbosity("ERROR");
    params_lm.setMaxIterations(20);
    params_lm.setLinearSolverType("MULTIFRONTAL_QR");
    gtsam::LevenbergMarquardtOptimizer optimizer_LM( graphWithPrior, *initial,
    params_lm );

    // 你可以尝试下高斯牛顿法
    // gtsam::GaussNewtonParams params_gn;
    // params_gn.setVerbosity("ERROR");
    // params_gn.setMaxIterations(20);
    // params_gn.setLinearSolverType("MULTIFRONTAL_QR");
    // gtsam::GaussNewtonOptimizer optimizer ( graphWithPrior, *initial, params_gn
    );

    gtsam::Values result = optimizer_LM.optimize();
    cout<<"Optimization complete"<<endl;
    cout<<"initial error: "<<graph->error ( *initial ) <<endl;
    cout<<"final error: "<<graph->error ( result ) <<endl;

    cout<<"done. write to g2o ... "<<endl;
    // 写入 g2o 文件，同样伪装成 g2o 中的顶点和边，以便用 g2o_viewer 查看。
    // 顶点
    ofstream fout ( "result_gtsam.g2o" );
    for ( const gtsam::Values::ConstKeyValuePair& key_value: result )
    {
        gtsam::Pose3 pose = key_value.value.cast<gtsam::Pose3>();
        gtsam::Point3 p = pose.translation();
        gtsam::Quaternion q = pose.rotation().toQuaternion();
        fout<<"VERTEX_SE3:QUAT "<<key_value.key<<" "
        <<p.x() <<" "<<p.y() <<" "<<p.z() <<" "
        <<q.x()<<" "<<q.y()<<" "<<q.z()<<" "<<q.w()<<" "<<endl;
    }
    // 边
    for ( gtsam::NonlinearFactor::shared_ptr factor: *graph )
    {
        gtsam::BetweenFactor<gtsam::Pose3>::shared_ptr f = dynamic_pointer_cast<
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
        gtsam::BetweenFactor<gtsam::Pose3>>( factor );
155     if ( f )
156     {
157         gtsam::SharedNoiseModel model = f->noiseModel();
158         gtsam::noiseModel::Gaussian::shared_ptr gaussianModel =
            dynamic_pointer_cast<gtsam::noiseModel::Gaussian>( model );
159         if ( gaussianModel )
160         {
161             // write the edge information
162             gtsam::Matrix info = gaussianModel->R().transpose() * gaussianModel
                ->R();
163             gtsam::Pose3 pose = f->measured();
164             gtsam::Point3 p = pose.translation();
165             gtsam::Quaternion q = pose.rotation().toQuaternion();
166             fout<<"EDGE_SE3:QUAT "<<f->key1()<<" "<<f->key2()<<" "
167                 <<p.x() <<" "<<p.y() <<" "<<p.z() <<" "
168                 <<q.x()<<" "<<q.y()<<" "<<q.z()<<" "<<q.w()<<" ";
169             gtsam::Matrix infoG2o = gtsam::I_6x6;
170             infoG2o.block(0,0,3,3) = info.block(3,3,3,3); // cov translation
171             infoG2o.block(3,3,3,3) = info.block(0,0,3,3); // cov rotation
172             infoG2o.block(0,3,3,3) = info.block(0,3,3,3); // off diagonal
173             infoG2o.block(3,0,3,3) = info.block(3,0,3,3); // off diagonal
174             for ( int i=0; i<6; i++ )
175             for ( int j=i; j<6; j++ )
176             {
177                 fout<<infoG2o(i,j)<<" ";
178             }
179             fout<<endl;
180         }
181     }
182 }
183 fout.close();
184 cout<<"done."<<endl;
185 }
```

在演示程序中，我们以文本方式读取一个g2o文件，并完成了向gtsam接口的转换。请留意代码中g2o与gtsam的异同点。比如说相同的

地方：

1.由于它们本质上都是同一个最小二乘优化问题，所以我们要设置的东西也是类似的。简而言之，即顶点（对应到因子图中的变量）的初值，以及边（对应到因子）的观测值，还有噪声大小。

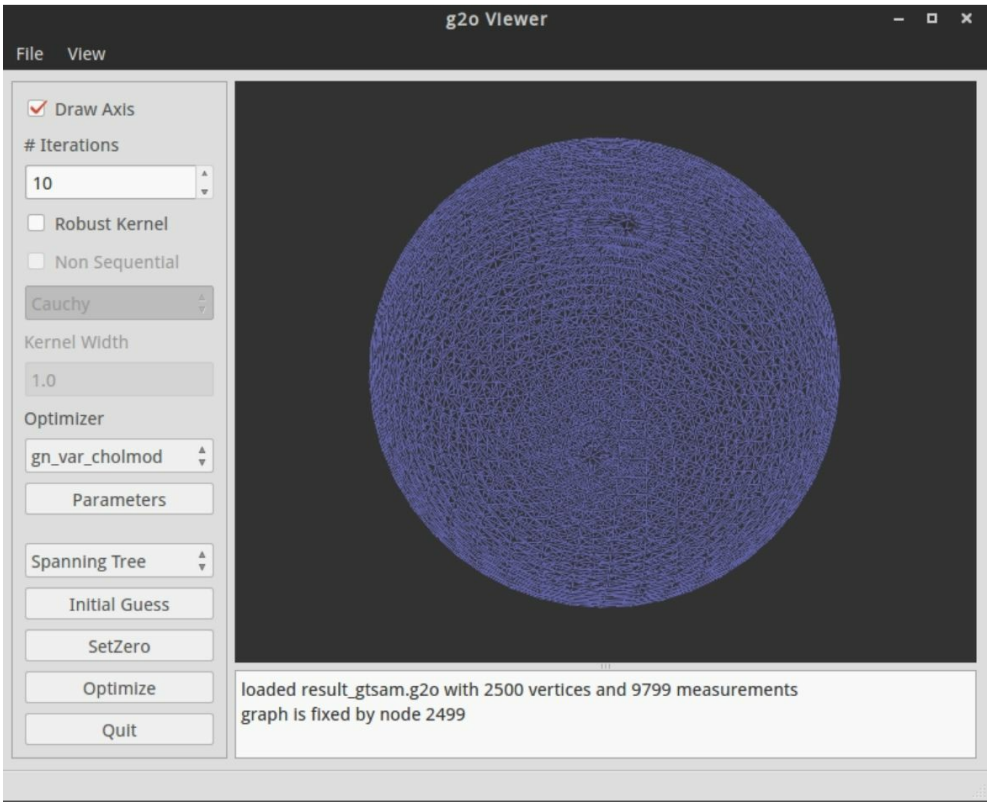
2.在优化设置方面，同样可以使用高斯牛顿法或列文伯格—马夸尔特方法，并配置详细的参数，只不过接口略有不同。g2o通过构建优化算法对象来实现，而gtsam则是传入优化参数。

不同的地方：

1.g2o的噪声模型和gtsam稍有不同，我们在代码中做了一下转换。

2.在稀疏化的处理方面，gtsam可选择使用QR分解或Cholesky分解，尽管我们没有详细解释这方面的具体过程。读者可以追踪参数的配置方式，看看gtsam提供哪些求解方法。由于单纯的Pose Graph没有太多稀疏性可以利用，所以这里区别不大。

最后，我们把gtsam的优化结果转换为g2o的输出文件，同样可以用g2o_viewer查看此结果，如图11-9所示。



书籍汇总下载 http://t.cn/RDIAj5D

图11-9　gtsam的优化结果。

下面是终端输出的优化信息：

```
$ build/pose_graph_gtsam sphere.g2o
reading from g2o file
read total 2500 vertices, 9799 edges.
Adding prior to g2o file
optimizing the factor graph
Initial error: 4.7724e+09
newError: 6.07118e+08
errorThreshold: 6.07118e+08 > 0
absoluteDecrease: 4165284178.46 >= 1e-05
relativeDecrease: 0.872785675288 >= 1e-05
// 中间略
newError: 63793.9289983
```

```
errorThreshold: 63793.9289983 > 0
absoluteDecrease: 0.279685092828 >= 1e-05
relativeDecrease: 4.38417684928e-06 < 1e-05
converged
errorThreshold: 63793.9289983 <? 0
absoluteDecrease: 0.279685092828 <? 1e-05
relativeDecrease: 4.38417684928e-06 <? 1e-05
iterations: 5 >? 20
Optimization complete
initial error: 4772402087.24
final error: 63793.9289982
done. write to g2o ...
done.
```

我们发现，虽然设置了最大迭代次数为20次，但gtsam只迭代了5次后算法就收敛了。同样，由于gtsam的误差定义方式与g2o不同，所以这里直接比较误差大小没有太大意义。如果用g2o_viewer打开result_gtsam.g2o文件并选择优化，就会发现在g2o度量下的误差大约为44360左右，和我们上一节使用李代数优化的结果一致，但gtsam的迭代次数更少一些。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

有点遗憾的是，本例没有体现出gtsam的增量特性。如果我们把g2o文件中的节点和边按照时间排序，然后一个一个地放入gtsam中，可能对它的增量特征会有更好的表述。我们把这个实验留作习题，交给读者来完成。

习题

1.如果将位姿图中的误差定义为$\Delta\boldsymbol{\xi}_{ij} = \boldsymbol{\xi}_i \circ \boldsymbol{\xi}_j^{-1}$，推导按照此定义的左乘扰动雅可比矩阵。

2.参照g2o的程序，在Ceres中实现对“球”位姿图的优化。

3.对“球”中的信息按照时间排序，分别喂给g2o和gtsam优化，比较它们的性能差异。

4.*阅读iSAM相关论文，理解它是如何实现增量式优化的。

---

[1]请注意，尽管数值上看此处的误差要大一些，但是由于我们自定义边时重新定义了误差的计算方式，所以此处数值的大小并不能直接用于比较。

[2]由于没有做更多的实验，所以该结论只在“球”这个例子上有效。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

第12讲　回环检测

主要目标

1.理解回环检测的必要性。

2.掌握基于词袋的外观式回环检测。

3.通过DBoW3的实验，学习词袋模型的实际用途。

本讲中，我们来介绍SLAM中另一个主要模块：回环检测。我们知道SLAM主体（前端、后端）主要的目的在于估计相机运动，而回环检测模块，无论是目标上还是方法上，都与前面讲的内容相差较大，所以通常被认为是一个独立的模块。我们将介绍主流视觉SLAM中检测回环的方式：词袋模型，并通过DBoW库上的程序实验，使读者得到更加直观的理解。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D



电子书寻找看手相qq37391319 钉钉或微信 pythontesting

书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

12.1 回环检测概述

12.1.1 回环检测的意义

我们已然介绍了前端和后端：前端提供特征点的提取和轨迹、地图的初值，而后端负责对所有这些数据进行优化。然而，如果像VO那样仅考虑相邻时间上的关联，那么，之前产生的误差将不可避免地累积到下一个时刻，使得整个SLAM会出现累积误差，长期估计的结果将不可靠，或者说，我们无法构建全局一致的轨迹和地图。

举例来说，假设我们在前端提取了特征，然后忽略掉特征点，在后端使用PoseGraph优化整个轨迹，如图12-1(a)所示。由于前端给出的只是局部的位姿间约束，比如，可能是x_1-x_2,x_2-x_3，等等。但是，由于x_1的估计存在误差，而x_2是根据x_1决定的，x_3又是由x_2决定的。以此类推，误差就会被累积起来，使得后端优化的结果如图12-1(b)所示，慢慢地趋向不准确。

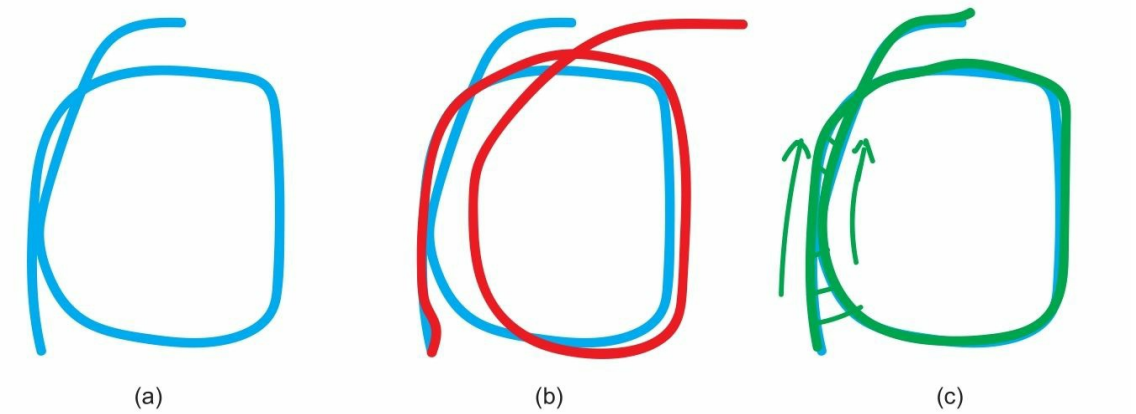


图12-1 漂移示意图。（a）真实轨迹；（b）由于前端只给出相邻帧间的估计，优化后的Pose Graph出现漂移；（c）添加回环检测后的Pose Graph可以消除累积误差。

虽然后端能够估计最大后验误差，但所谓“好模型架不住烂数据”，只有相邻关键帧数据时，我们能做的事情并不很多，也无从消除累积误差。但是，回环检测模块能够给出除了相邻帧之外的一些时隔更加久远

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

的约束：例如x_1-x_{100}之间的位姿变换。为什么它们之间会有约束呢？这是因为我们察觉到相机经过了同一个地方，采集到了相似的数据。而回环检测的关键，就是如何有效地检测出相机经过同一个地方这件事。如果我们能够成功地检测到这件事，就可以为后端的Pose Graph提供更多的有效数据，使之得到更好的估计，特别是得到一个全局一致（Global Consistent）的估计。由于Pose Graph可以看成一个质点——弹簧系统，所以回环检测相当于在图像中加入了额外的弹簧，提高了系统稳定性。读者亦可直观地想象成回环边把带有累积误差的边“拉”到了正确的位置——如果回环本身正确的话。

回环检测对于SLAM系统意义重大。它关系到我们估计的轨迹和地图在长时间下的正确性。另一方面，由于回环检测提供了当前数据与所有历史数据的关联，在跟踪算法丢失之后，我们还可以利用回环检测进行重定位。因此，回环检测对整个SLAM系统精度与稳健性的提升是非常明显的。甚至在某些时候，我们把仅有前端和局部后端的系统称为VO，而把带有回环检测和全局后端的系统称为SLAM。

12.1.2 方法

下面我们来考虑回环检测如何实现的问题。

最简单的方式就是对任意两幅图像都做一遍特征匹配，根据正确匹配的数量确定哪两幅图像存在关联——这确实是一种朴素且有效的思想。缺点在于，我们盲目地假设了“任意两幅图像都可能存在回环”，使得要检测的数量实在太大：对于N个可能的回环，我们要检测C^2_N那么多次，这是$O(N^2)$的复杂度，随着轨迹变长增长太快，在大多数实时系统当中是不实用的。另一种朴素的方式是，随机抽取历史数据并进行回环检测，比如说在n帧当中随机抽5帧与当前帧比较。这种做法能够维持常数时间的运算量，但是这种盲目试探方法在帧数N增长时，抽到回环的几率又大幅下降，使得检测效率不高。

上面说的朴素思路都过于粗糙。尽管随机检测在有些实现中确实有用[88]，但我们至少希望有一个“哪处可能出现回环”的预计，才好不那么盲目地去检测。这样的方式大体分为两种思路：基于里程计的几何关系

书籍汇总下载 http://t.cn/RDIAj5D

（Odometry based），或基于外观（Appearance based）。基于几何关系是说，当我们发现当前相机运动到了之前的某个位置附近时，检测它们有没有回环关系[89]——这自然是一种直观的想法，但是由于累积误差的存在，我们往往没法正确地发现“运动到了之前的某个位置附近”这件事实，回环检测也无从谈起。因此，这种做法在逻辑上存在一点问题，因为回环检测的目标在于发现“相机回到之前位置”的事实，从而消除累积误差。而基于几何关系的做法假设了“相机回到之前位置附近”，这样才能检测回环。这是有倒果为因的嫌疑的，因而也无法在累积误差较大时工作[12]。

另一种方法是基于外观的。它和前端、后端的估计都无关，仅根据两幅图像的相似性确定回环检测关系。这种做法摆脱了累积误差，使回环检测模块成为SLAM系统中一个相对独立的模块（当然前端可以为它提供特征点）。自21世纪初被提出以来，基于外观的回环检测方式能够有效地在不同场景下工作，成为了视觉SLAM中主流的做法，并被应用于实际的系统中去[90,80,73]。

在基于外观的回环检测算法中，核心问题是如何计算图像间的相似性。比如，对于图像A和图像B，我们要设计一种方法，计算它们之间的相似性评分：$s(A,B)$。当然这个评分会在某个区间内取值，当它大于一定量后我们认为出现了一个回环。读者可能会有疑问：计算两幅图像之间的相似性很困难吗？例如直观上看，图像能够表示成矩阵，那么直接让两幅图像相减，然后取某种范数行不行呢？

$$s(\boldsymbol{A}, \boldsymbol{B}) = \|\boldsymbol{A} - \boldsymbol{B}\|. \tag{12.1}$$

为什么我们不这样做？

1.首先，前面也说过，像素灰度是一种不稳定的测量值，它严重地受环境光照和相机曝光的影响。假设相机未动，我们打开了一支电灯，那么图像会整体变亮一些。这样，即使对于同样的数据，我们都会得到一个很大的差异值。

2.另一方面，当相机视角发生少量变化时，即使每个物体的光度不变，它们的像素也会在图像中发生位移，造成一个很大的差异值。

由于这两种情况的存在，实际当中，即使对于非常相似的图像，A-

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

*B*也会经常得到一个（不符合实际的）很大的值。所以我们说，这个函数不能很好地反映图像间的相似关系。这里牵涉到一个“好”和“不好”的定义问题。我们要问，怎样的函数能够更好地反映相似关系，而怎样的函数不够好呢？从这里可以引出感知偏差（Perceptual Aliasing）和感知变异（PerceptualVariability）两个概念。现在我们来更详细地讨论一下。

12.1.3 准确率和召回率

从人类的角度看，（至少我们自认为）我们能够以很高的精确度，感觉到“两幅图像是否相似”或“这两张照片是从同一个地方拍摄的”这一事实，但由于目前尚未掌握人脑的工作原理，我们无法清楚地描述自己是如何完成这件事的。从程序角度看，我们希望程序算法能够得出和人类，或者和事实一致的判断。当我们觉得，或者事实上就是，两幅图像从同一个地方拍摄，那么回环检测算法也应该给出“这是回环”的结果。反之，如果我们觉得，或事实上是，两幅图像是从不同地方拍摄的，那么程序也应该给出“这不是回环”的判断。[1]当然，程序的判断并不总是与我们人类的想法一致，所以可能出现表12-1中的4种情况：

表12-1回环检测的结果分类

算法 \ 事实	是回环	不是回环
是回环	真阳性（True Positive）	假阳性（False Positive）
不是回环	假阴性（False Negative）	真阴性（True Negative）

这里阴性/阳性的说法是借用了医学上的说法。假阳性（False Positive）又称为感知偏差，而假阴性（False Negative）称为感知变异（见图12-2）。为方便书写，用缩写TP代表True Positive，其余类推。由于我们希望算法和人类的判断一致，所以希望TP和TN要尽量高，而FP和FN要尽可能低。所以，对于某种特定算法，我们可以统计它在某个数据集上的TP、TN、FP、FN的出现次数，并计算两个统计量：准确率和召回率（Precision&Recall）。

$$\text{Precision} = \text{TP}/(\text{TP} + \text{FP}), \quad \text{Recall} = \text{TP}/(\text{TP} + \text{FN}). \tag{12.2}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

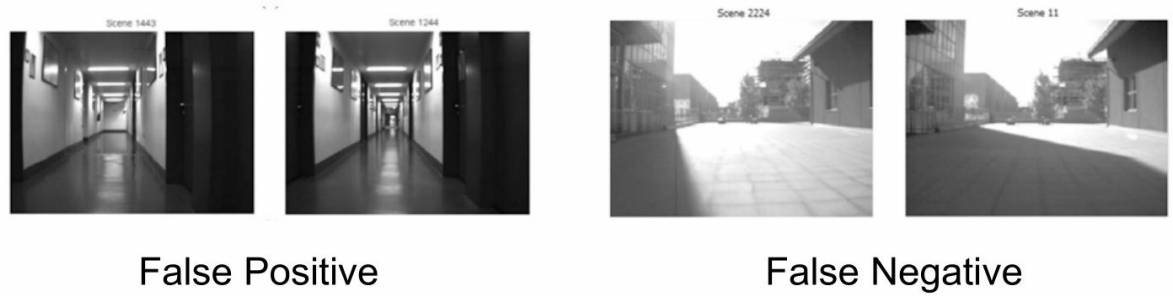


图12-2　假阳性与假阴性的例子。左侧为假阳性，两幅图像看起来很像，但并非同一走廊；右侧为假阴性，由于光照变化，同一地点不同时刻的照片看起来很不一样。

从公式字面意义上来看，准确率描述的是，算法提取的所有回环中确实是真实回环的概率。而召回率则是说，在所有真实回环中被正确检测出来的概率。为什么取这两个统计量呢？因为它们有一定的代表性，并且通常来说是一对矛盾。

一个算法往往有许多的设置参数。比如说，当提高某个阈值时，算法可能变得更加“严格”——它检出更少的回环，使准确率得以提高。但同时，由于检出的数量变少了，许多原本是回环的地方就可能被漏掉了，导致召回率下降。反之，如果我们选择更加宽松的配置，那么检出的回环数量将增加，得到更高的召回率，但其中可能混杂一些不是回环的情况，于是准确率下降。

为了评价算法的好坏，我们会测试它在各种配置下的P和R值，然后做出一条Precision-Recall曲线（见图12-3）。当用召回率为横轴，用准确率为纵轴时，我们会关心整条曲线偏向右上方的程度、100%准确率下的召回率或者50%召回率时的准确率，作为评价算法的指标。不过请注意，除去一些“天壤之别”的算法，我们通常不能一概而论地说算法A就是优于算法B的。我们可能会说A在准确率较高时还有很好的召回率，而B在70%召回率的情况下还能保证较好的准确率，诸如此类。

值得一提的是，在SLAM中，我们对准确率要求更高，而对召回率则相对宽容一些。由于假阳性的（检测结果是而实际不是的）回环将在后端的Pose Graph中添加根本错误的边，有些时候会导致优化算法给出完全错误的结果。想象一下，如果SLAM程序错误地将所有的办公桌当成了同一张，那建出来的图会怎么样呢？你可能会看到走廊不直了，墙壁被交错在一起了，最后整个地图都失效了。而相比之下，召回率低一些，则顶多有部分的回环没有被检测到，地图可能受一些累积误差的影

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

响——然而仅需一两次回环就可以完全消除它们了。所以说在选择回环检测算法时，我们更倾向于把参数设置得更严格一些，或者在检测之后再加上回环验证的步骤。

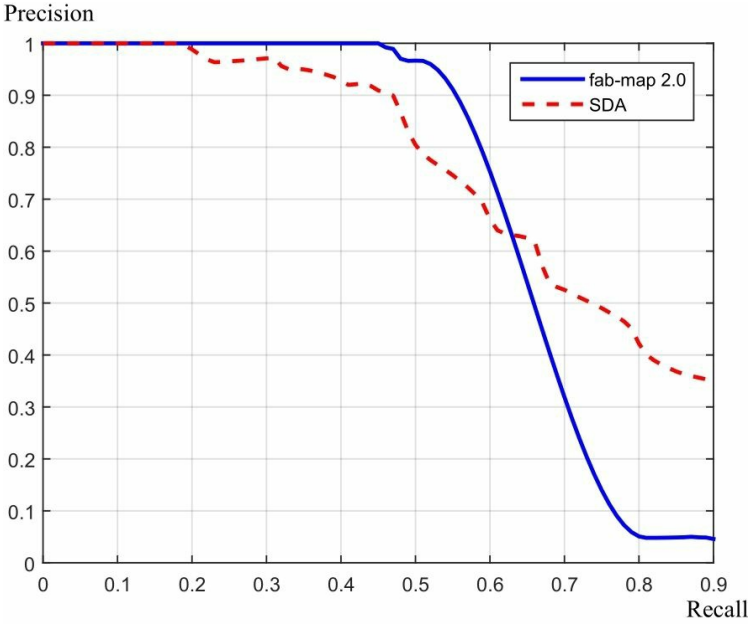


图12-3　准确率-召回率曲线的例子[91]。随着召回率的上升，检测条件变得宽松，准确率随之下降。好的算法在较高召回率情况下仍能保证较好的准确率。

那么，回到之前的问题，为什么不用*A-B*来计算相似性呢？我们会发现它的准确率和召回率都很差，可能出现大量的False Positive或False Negative的情况，所以说这样做“不好”。那么，什么方法更好一些呢？

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

12.2 词袋模型

既然直接用两张图像相减的方式不够好，那么我们就需要一种更加可靠的方式。结合前面几讲的内容，一种直观的思路是：为何不像VO那样使用特征点来做回环检测呢？和VO一样，我们对两幅图像的特征点进行匹配，只要匹配数量大于一定值，就认为出现了回环。进一步，根据特征点匹配，我们还能计算出这两幅图像之间的运动关系。当然这种做法会存在一些问题，例如，特征的匹配会比较费时、当光照变化时特征描述可能不稳定等，但离我们要介绍的词袋模型已经很相近了。下面我们先来讲词袋的做法，再来讨论数据结构之类的实现细节。

词袋，也就是Bag-of-Words（BoW），目的是用“图像上有哪几种特征”来描述一幅图像。例如，我们说某张照片中有一个人、一辆车；而另一张中有两个人、一只狗。根据这样的描述，就可以度量这两幅图像的相似性。再具体一些，我们要做以下几件事：

1.确定“人”“车”“狗”等概念——对应于BoW中的“单词”（Word），许多单词放在一起，组成了“字典”（Dictionary）。

2.确定一幅图像中出现了哪些在字典中定义的概念——我们用单词出现的情况（或直方图）描述整幅图像。这就把一幅图像转换成了一个向量的描述。

3.比较上一步中的描述的相似程度。

以上面举的例子来说，首先我们通过某种方式得到了一本“字典”。字典上记录了许多单词，每个单词都有一定意义，例如“人”“车”“狗”都是记录在字典中的单词，我们不妨记为w_1,w_2,w_3。然后，对于任意图像A，根据它们含有的单词，可记为

$$A = 1 \cdot w_1 + 1 \cdot w_2 + 0 \cdot w_3. \tag{12.3}$$

字典是固定的，所以只要用$[1,1,0]^{\mathrm{T}}$这个向量就可以表达A的意义。通过字典和单词，只需一个向量就可以描述整幅图像了。该向量描述的是“图像是否含有某类特征”的信息，比单纯的灰度值更加稳定。又因为

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

描述向量说的是“是否出现”，而不管它们“在哪儿出现”，所以与物体的空间位置和排列顺序无关，因此在相机发生少量运动时，只要物体仍在视野中出现，我们就仍然保证描述向量不发生变化。[2]基于这种特性，我们称它为Bag-of-Words而不是什么List-of-Words，强调的是Words的有无，而无关其顺序。因此，可以说字典类似于单词的一个集合。

回到上面的例子，同理，用$[2,0,1]^{\mathrm{T}}$可以描述图像B。如果只考虑“是否出现”而不考虑数量，也可以是$[1,0,1]^{\mathrm{T}}$，这时候这个向量就是二值的。于是，根据这两个向量，设计一定的计算方式，就能确定图像间的相似性了。当然，如果对两个向量求差仍然有一些不同的做法，比如对于$a,b\in\mathrm{R}^{W}$，可以计算：

$$s(\boldsymbol{a},\boldsymbol{b})=1-\frac{1}{W}\|\boldsymbol{a}-\boldsymbol{b}\|_1. \tag{12.4}$$

其中范数取L_1范数，即各元素绝对值之和。请注意在两个向量完全一样时，我们将得到1；完全相反时（a为0的地方b为1）得到0。这样就定义了两个描述向量的相似性，也就定义了图像之间的相似程度。

接下来的问题是什么呢？

1.我们虽然清楚了字典的定义方式，但它到底是怎么来的呢？

2.如果我们能够计算两幅图像间的相似程度评分，是否就足够判断回环了呢？

所以接下来，我们首先介绍字典的生成方式，然后介绍如何利用字典实际地计算两幅图像间的相似性。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

12.3 字典

12.3.1 字典的结构

按照前面的介绍，字典由很多单词组成，而每一个单词代表了一个概念。一个单词与一个单独的特征点不同，它不是从单幅图像上提取出来的，而是某一类特征的组合。所以，字典生成问题类似于一个聚类（Clustering）问题。

聚类问题在无监督机器学习（Unsupervised ML）中特别常见，用于让机器自行寻找数据中的规律。BoW的字典生成问题亦属于其中之一。首先，假设我们对大量的图像提取了特征点，比如说有N个。现在，我们想找一个有k个单词的字典，每个单词可以看作局部相邻特征点的集合，应该怎么做呢？这可以用经典的K-means（K均值）算法[92]解决。

K-means是一个非常简单有效的方法，因此在无监督学习中广为使用，下面对其原理稍做介绍。简单的说，当有N个数据，想要归成k个类，那么用K-means来做主要包括如下步骤：

1.随机选取k个中心点：$c_1,\cdots,c_k$。

2.对每一个样本，计算它与每个中心点之间的距离，取最小的作为它的归类。

3.重新计算每个类的中心点。

4.如果每个中心点都变化很小，则算法收敛，退出；否则返回第2步。

K-means的做法是朴素且简单有效的，不过也存在一些问题，例如，需要指定聚类数量、随机选取中心点使得每次聚类结果都不相同，以及一些效率上的问题。随后研究者们亦开发出了层次聚类法、K-means++[93]等算法以弥补它的不足，不过这都是后话，我们就不详细讨论了。总之，根据K-means，我们可以把已经提取的大量特征点聚类成

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

一个含有k个单词的字典了。现在的问题变成了如何根据图像中某个特征点查找字典中相应的单词。

仍然有朴素的思想：只要和每个单词进行比对，取最相似的那个就可以了——这当然是简单有效的做法。然而，考虑到字典的通用性[3]，我们通常会使用一个较大规模的字典，以保证当前使用环境中的图像特征都曾在字典里出现，或至少有相近的表达。如果你觉得对十个单词一一比较不是什么麻烦事，那么对于一万个呢？十万个呢？

也许读者学过数据结构，这种$O(n)$的查找算法显然不是我们想要的。如果字典排过序，那么二分查找显然可以提升查找效率，达到对数级别的复杂度。而实践当中，我们可能会用更复杂的数据结构，例如Fabmap[94,95,96]中的Chou-Liu tree[97]等。但我们不想把本书写成复杂细节的集合，所以介绍另一种较为简单实用的树结构[98]。

在文献[98]中，使用一种k叉树来表达字典。它的思路很简单（如图12-4所示），类似于层次聚类，是k-means的直接扩展。假定我们有N个特征点，希望构建一个深度为d、每次分叉为k的树，那么做法如下[4]：

1.在根节点，用k-means把所有样本聚成k类（实际中为保证聚类均匀性会使用k-means++）。这样得到了第一层。

2.对第一层的每个节点，把属于该节点的样本再聚成k类，得到下一层。

3.以此类推，最后得到叶子层。叶子层即为所谓的Words。

书籍汇总下载 http://t.cn/RDIAj5D

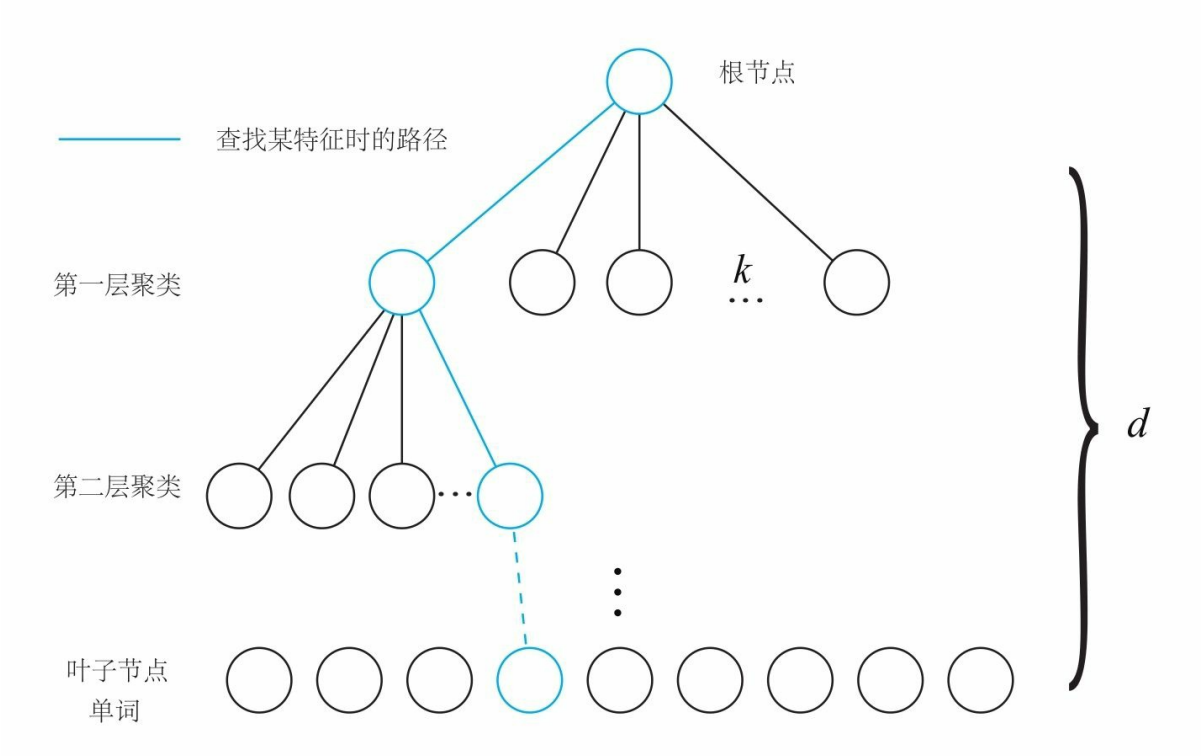


图12-4　K叉树字典示意图。训练字典时，逐层使用K-means聚类。根据已知特征查找单词时，亦可逐层比对，找到对应的单词。

实际上，最终我们仍在叶子层构建了单词，而树结构中的中间节点仅供快速查找时使用。这样一个k分支、深度为d的树，可以容纳k^d个单词。另一方面，在查找某个给定特征对应的单词时，只需将它与每个中间节点的聚类中心比较（一共d次），即可找到最后的单词，保证了对数级别的查找效率。

12.3.2　实践：创建字典

既然讲到了字典生成，我们就来实际演示一下。前面的VO部分大量使用了ORB特征描述，所以这里就来演示一下如何生成及使用ORB字典。

本实验中，我们选取TUM数据集中的10幅图像（位于slambook/ch9/data中，如图12-5所示），它们来自一组实际的相机运动轨迹。可以看出，第一幅图像与最后一幅图像明显采自同一个地方，现在我们要看程序能否检测到这件事情。根据词袋模型，我们先来生成这

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

十张图像对应的字典。



图12-5　演示实验中使用的十幅图像，采集自不同时刻的轨迹。

需要声明的是，实际BoW使用时字典往往是从更大的数据集生成的，而且最好是来自与目标环境类似的地方。我们通常使用较大规模的字典——越大代表字典单词量越丰富，越容易找到与当前图像对应的单词，但也不能大到超过我们的计算能力和内存。笔者不打算在GitHub上存放一个很大的字典文件，所以我们暂时从十幅图像训练一个小的字典。如果读者想进一步追求更好的效果，应该下载更多的数据，训练更大的字典，这样程序才会实用。也可以使用别人训练好的字典，但请注意字典使用的特征类型是否一致。

下面开始训练字典。首先，请安装本程序使用的BoW库。我们使用的是DBoW3[5]：https://github.com/rmsalinas/DBow3。读者也可从本书代码的3rdparty文件夹中找到它，它是一个cmake工程。

接下来考虑训练字典：

slambook/ch9/feature_training.cpp

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

```
int main( int argc, char** argv )
{
    // read the image
    cout<<"reading images... "<<endl;
    vector<Mat> images;
    for ( int i=0; i<10; i++ )
    {
        string path = "./data/"+to_string(i+1)+".png";
        images.push_back( imread(path) );
    }

    // detect ORB features
    cout<<"detecting ORB features ... "<<endl;
    Ptr< Feature2D > detector = ORB::create();
    vector<Mat> descriptors;
    for ( Mat& image:images )
    {
        vector<KeyPoint> keypoints;
        Mat descriptor;
        detector->detectAndCompute( image, Mat(), keypoints, descriptor );
        descriptors.push_back( descriptor );
    }

    // create vocabulary
    cout<<"creating vocabulary ... "<<endl;
    DBoW3::Vocabulary vocab;
    vocab.create( descriptors );
    cout<<"vocabulary info: "<<vocab<<endl;
    vocab.save( "vocabulary.yml.gz" );
    cout<<"done"<<endl;

    return 0;
}
```

DBoW3的使用非常容易。我们对十张目标图像提取ORB特征并存放至vector容器中，然后调用DBoW3的字典生成接口即可。在

书籍汇总下载 http://t.cn/RDIAj5D

DBoW3::Vocabulary对象的构造函数中，我们能够指定树的分叉数量及深度，不过这里使用了默认构造函数，也就是k=10,d=5。这是一个小规模的字典，最大能容纳10,000个单词。对于图像特征，我们亦使用默认参数，即每幅图像500个特征点。最后，我们把字典存储为一个压缩文件。

运行此程序，将看到如下字典信息输出：

```
$ build/feature_training
reading images...
detecting ORB features ...
creating vocabulary ...
vocabulary info: Vocabulary: k = 10, L = 5, Weighting = tf-idf, Scoring = L1-norm,
Number of words = 4983
done
```

我们看到：分支数量k为10，深度L为5[6]，单词数量为4983，没有达到最大容量。但是，剩下的Weighting和Scoring是什么呢？从字面上看，Weighting是权重，Scoring似乎指的是评分，但评分是如何计算的呢？

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

12.4 相似度计算

12.4.1 理论部分

下面我们来讨论相似度计算的问题。有了字典之后，给定任意特征f_i，只要在字典树中逐层查找，最后都能找到与之对应的单词w_j——当字典足够大时，我们可以认为f_i和w_j来自同一类物体（尽管没有理论上的保证，仅是在聚类意义下这样说）。那么，假设一幅图像中提取了N个特征，找到这N个特征对应的单词之后，我们就相当于拥有了该图像在单词列表中的分布，或者直方图。直观地讲（或理想情况下），相当于是说“这幅图里有一个人和一辆汽车”这样的意思了。根据Bag-of-Words的说法，不妨认为这是一个Bag。

注意到这种做法中我们对所有单词都是“一视同仁”的——有就是有，没有就是没有。这样做好不好呢？考虑到不同的单词在区分性上的重要性并不相同。例如，“的”“是”这样的字可能在许许多多的句子中出现，我们无法根据它们判别句子的类型；但如果有“文档”“足球”这样的单词，对判别句子的作用就大一些，可以说它们提供了更多信息。所以概括起来，我们希望对单词的区分性或重要性加以评估，给它们不同的权值以起到更好的效果。

在文本检索中，常用的一种做法称为TF-IDF（Term Frequency–Inverse Document Frequency）[100,101]，或译频率-逆文档频率[7]。TF部分的思想是，某单词在一幅图像中经常出现，它的区分度就高。另一方面，IDF的思想是，某单词在字典中出现的频率越低，则分类图像时区分度越高。

在词袋模型中，在建立字典时可以考虑IDF部分。我们统计某个叶子节点w_i中的特征数量相对于所有特征数量的比例作为IDF部分。假设所有特征数量为n，w_i数量为n_i，那么该单词的IDF为

$$\mathrm{IDF}_i = \log \frac{n}{n_i}. \tag{12.5}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

另一方面，TF部分则是指某个特征在单幅图像中出现的频率。假设图像A中单词w_i出现了n_i次，而一共出现的单词次数为n，那么TF为

$$\mathrm{TF}_i = \frac{n_i}{n}. \tag{12.6}$$

于是，w_i的权重等于TF乘IDF之积：

$$\eta_i = \mathrm{TF}_i \times \mathrm{IDF}_i. \tag{12.7}$$

考虑权重以后，对于某幅图像A，它的特征点可对应到许多个单词，组成它的Bagof-Words：

$$A = \{(w_1, \eta_1), (w_2, \eta_2), \ldots, (w_N, \eta_N)\} \triangleq \boldsymbol{v}_A. \tag{12.8}$$

由于相似的特征可能落到同一个类中，因此实际的v_A中会存在大量的零。无论如何，通过词袋我们用单个向量v_A描述了一幅图像A。这个向量v_A是一个稀疏的向量，它的非零部分指示了图像A中含有哪些单词，而这些部分的值为TF-IDF的值。

接下来的问题是：给定v_A和v_B，如何计算它们的差异呢？这个问题和范数定义的方式一样，存在若干种解决方式，比如文献[102]中提到的L_1范数形式：

$$s(\boldsymbol{v}_A - \boldsymbol{v}_B) = 2\sum_{i=1}^{N} |\boldsymbol{v}_{Ai}| + |\boldsymbol{v}_{Bi}| - |\boldsymbol{v}_{Ai} - \boldsymbol{v}_{Bi}|. \tag{12.9}$$

当然也有很多种别的方式等你探索，在这里我们仅举一例作为演示。至此，我们已说明了如何通过词袋模型来计算任意图像间的相似度。下面通过程序实际演练一下。

12.4.2 实践：相似度的计算

上一节的实践部分中，我们已对十幅图像生成了字典。这次我们使用此字典生成Bag-of-Words并比较它们的差异，看看与实际有什么不

同。

slambook/ch9/loop_closure.cpp

```
int main( int argc, char** argv )
{
    // read the images and database
    cout<<"reading database"<<endl;
    DBoW3::Vocabulary vocab("./vocabulary.yml.gz");
    if ( vocab.empty() )
    {
        cerr<<"Vocabulary does not exist."<<endl;
        return 1;
    }
    cout<<"reading images... "<<endl;
    vector<Mat> images;
    for ( int i=0; i<10; i++ )
    {
        string path = "./data/"+to_string(i+1)+".png";
```

书籍汇总下载 http://t.cn/RDIAj5D

```
16          images.push_back( imread(path) );
17      }
18
19      // NOTE: in this case we are comparing images with a vocabulary generated by
        themselves, this may leed to overfitting.
20      // detect ORB features
21      cout<<"detecting ORB features ... "<<endl;
22      Ptr< Feature2D > detector = ORB::create();
23      vector<Mat> descriptors;
24      for ( Mat& image:images )
25      {
26          vector<KeyPoint> keypoints;
27          Mat descriptor;
28          detector->detectAndCompute( image, Mat(), keypoints, descriptor );
29          descriptors.push_back( descriptor );
30      }
31
32      // we can compare the images directly or we can compare one image to a database
33      // images
34      cout<<"comparing images with images "<<endl;
35      for ( int i=0; i<images.size(); i++ )
36      {
37          DBoW3::BowVector v1;
38          vocab.transform( descriptors[i], v1 );
39          for ( int j=i; j<images.size(); j++ )
40          {
41              DBoW3::BowVector v2;
42              vocab.transform( descriptors[j], v2 );
43              double score = vocab.score(v1, v2);
44              cout<<"image "<<i<<" vs image "<<j<<" : "<<score<<endl;
45          }
46          cout<<endl;
47      }
48
49      // or with database
50      cout<<"comparing images with database "<<endl;
51      DBoW3::Database db( vocab, false, 0);
52      for ( int i=0; i<descriptors.size(); i++ )
53      db.add(descriptors[i]);
54      cout<<"database info: "<<db<<endl;
55      for ( int i=0; i<descriptors.size(); i++ )
56      {
57          DBoW3::QueryResults ret;
58          db.query( descriptors[i], ret, 4);      // max result=4
59          cout<<"searching for image "<<i<<" returns "<<ret<<endl<<endl;
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
60      }
61      cout<<"done."<<endl;
62  }
```

本程序演示了两种比对方式：图像之间的直接比较，以及图像与数据库之间的比较——尽管它们是大同小异的。此外，我们输出了每幅图像对应的Bag-of-Words描述向量，读者可以从输出数据中看到它们。

```
1  $ build/feature_training
2  reading database
3  reading images...
4  detecting ORB features ...
5  comparing images with images
6  desp 0 size: 500
7  transform image 0 into BoW vector: size = 455
8  key value pair = <1, 0.00155622>, <3, 0.00222645>, <12, 0.00222645>, <13,
   0.00222645>, <14, 0.00222645>, <22, 0.00222645>, <33, 0.00222645>, <37,
   0.00155622>, <38, 0.00222645>, <39, 0.00222645>, <43, 0.00222645>, <57, 0.00155622>
    ......
```

可以看到，BoW描述向量中含有每个单词的ID和权重，它们构成了整个稀疏的向量。当我们比较两个向量时，DBoW3会为我们计算一个分数，计算的方式由之前构造字典时定义：

```
1  image 0 vs image 0 : 1
2  image 0 vs image 1 : 0.0234552
3  image 0 vs image 2 : 0.0225237
4  image 0 vs image 3 : 0.0254611
5  image 0 vs image 4 : 0.0253451
6  image 0 vs image 5 : 0.0272257
7  image 0 vs image 6 : 0.0217745
8  image 0 vs image 7 : 0.0231948
9  image 0 vs image 8 : 0.0311284
10 image 0 vs image 9 : 0.0525447
```

书籍汇总下载 http://t.cn/RDIAj5D

在数据库查询时，DBoW对上面的分数进行排序，给出最相似的结果：

```
searching for image 0 returns 4 results:
<EntryId: 0, Score: 1>
<EntryId: 9, Score: 0.0525447>
<EntryId: 8, Score: 0.0311284>
<EntryId: 5, Score: 0.0272257>

searching for image 1 returns 4 results:
<EntryId: 1, Score: 1>
<EntryId: 2, Score: 0.0339641>
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
<EntryId: 8, Score: 0.0299387>
<EntryId: 3, Score: 0.0256668>

searching for image 2 returns 4 results:
<EntryId: 2, Score: 1>
<EntryId: 7, Score: 0.036092>
<EntryId: 9, Score: 0.0348702>
<EntryId: 1, Score: 0.0339641>

searching for image 3 returns 4 results:
<EntryId: 3, Score: 1>
<EntryId: 9, Score: 0.0357317>
<EntryId: 8, Score: 0.0278496>
<EntryId: 5, Score: 0.0270168>

searching for image 4 returns 4 results:
<EntryId: 4, Score: 1>
<EntryId: 5, Score: 0.0493492>
<EntryId: 0, Score: 0.0253451>
<EntryId: 6, Score: 0.0253017>

searching for image 5 returns 4 results:
<EntryId: 5, Score: 1>
<EntryId: 4, Score: 0.0493492>
<EntryId: 9, Score: 0.028996>
<EntryId: 6, Score: 0.0277584>

searching for image 6 returns 4 results:
<EntryId: 6, Score: 1>
<EntryId: 8, Score: 0.0306241>
<EntryId: 5, Score: 0.0277584>
<EntryId: 3, Score: 0.0267135>

searching for image 7 returns 4 results:
<EntryId: 7, Score: 1>
<EntryId: 2, Score: 0.036092>
<EntryId: 1, Score: 0.0239091>
<EntryId: 0, Score: 0.0231948>

searching for image 8 returns 4 results:
<EntryId: 8, Score: 1>
<EntryId: 9, Score: 0.0329149>
<EntryId: 0, Score: 0.0311284>
<EntryId: 6, Score: 0.0306241>

```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
searching for image 9 returns 4 results:
<EntryId: 9, Score: 1>
<EntryId: 0, Score: 0.0525447>
<EntryId: 3, Score: 0.0357317>
<EntryId: 2, Score: 0.0348702>
```

读者可以查看所有的输出，看看不同图像与相似图像评分有多大差异。我们看到，明显相似的图1和图10（在C++中下标为分别为0和9），其相似度评分约0.0525；而其他图像约为0.02。

在本节的演示实验中，我们看到相似图像1和10的评分明显高于其他图像对，然而就数值上看并没有我们想象的那么明显。按说如果自己和自己比较相似度为100%，那么我们（从人类角度）认为图1和图10至少也有百分之七八十的相似度，而其他图可能为百分之二三十。然而实验结果却是无关图像约2%，相似图像约5%，似乎没有我们想象的那么明显。这是否是我们想要看到的结果呢？

书籍汇总下载 http://t.cn/RDIAj5D

12.5 实验分析与评述

12.5.1 增加字典规模

在机器学习领域，如果代码没有出错而结果不满意，我们首先怀疑“网络结构是否够大，层数是否足够深，数据样本是否够多”等，这依然是出于“好模型敌不过烂数据”的大原则（一方面也是因为缺乏更深层次的理论分析）。尽管我们现在是在研究SLAM，但出现这种情况，我们首先会怀疑：是不是字典选得太小了？毕竟我们是从十幅图生成的字典，然后又根据这个字典来计算图像相似性。

slambook/ch9/vocab_larger.yml.gz是我们生成的一个稍微大一点儿的字典——事实上是对同一个数据序列的所有图像生成的，大约有2,900幅图像。字典的规模仍然取k=10,d=5，即最多一万个单词。读者可以使用同目录下的gen_vocab_large.cpp文件自行训练字典。请注意，若要训练大型字典，可能需要一台内存较大的机器，并且耐心等上一段时间。我们对上一节的程序稍加修改，使用更大的字典去检测图像相似性：

```
comparing images with database
database info: Database: Entries = 10, Using direct index = no. Vocabulary: k = 10,
 L = 5, Weighting = tf-idf, Scoring = L1-norm, Number of words = 99566
searching for image 0 returns 4 results:
<EntryId: 0, Score: 1>
<EntryId: 9, Score: 0.0320906>
<EntryId: 8, Score: 0.0103268>
<EntryId: 4, Score: 0.0066729>

searching for image 1 returns 4 results:
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
<EntryId: 1, Score: 1>
<EntryId: 2, Score: 0.0238409>
<EntryId: 8, Score: 0.00814409>
<EntryId: 3, Score: 0.00697527>

searching for image 2 returns 4 results:
<EntryId: 2, Score: 1>
<EntryId: 1, Score: 0.0238409>
<EntryId: 5, Score: 0.00897928>
<EntryId: 8, Score: 0.00893477>

searching for image 3 returns 4 results:
<EntryId: 3, Score: 1>
<EntryId: 5, Score: 0.0107005>
<EntryId: 8, Score: 0.00870392>
<EntryId: 6, Score: 0.00720695>

searching for image 4 returns 4 results:
<EntryId: 4, Score: 1>
<EntryId: 6, Score: 0.0069998>
<EntryId: 0, Score: 0.0066729>
<EntryId: 5, Score: 0.0062834>

searching for image 5 returns 4 results:
<EntryId: 5, Score: 1>
<EntryId: 3, Score: 0.0107005>
<EntryId: 2, Score: 0.00897928>
<EntryId: 4, Score: 0.0062834>

searching for image 6 returns 4 results:
<EntryId: 6, Score: 1>
<EntryId: 7, Score: 0.00915307>
<EntryId: 3, Score: 0.00720695>
<EntryId: 4, Score: 0.0069998>

searching for image 7 returns 4 results:
<EntryId: 7, Score: 1>
<EntryId: 6, Score: 0.00915307>
<EntryId: 8, Score: 0.00814517>
<EntryId: 1, Score: 0.00538609>

searching for image 8 returns 4 results:
<EntryId: 8, Score: 1>
<EntryId: 0, Score: 0.0103268>
<EntryId: 2, Score: 0.00893477>
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
<EntryId: 3, Score: 0.00870392>

searching for image 9 returns 4 results:
<EntryId: 9, Score: 1>
<EntryId: 0, Score: 0.0320906>
<EntryId: 8, Score: 0.00636511>
<EntryId: 1, Score: 0.00587605>
```

可以看到，当字典规模增加时，无关图像的相似性明显变小了。而相似的图像，例如图像1和10，虽然分值也略微下降，但相对于其他图像的评分，却变得更为显著了。这说明增加字典训练样本是有益的。同理，读者可以尝试使用更大规模的字典，看看结果会发生怎样的变化。

12.5.2 相似性评分的处理

对任意两幅图像，我们都能给出一个相似性评分，但是只利用这个分值的绝对大小并不一定有很好的帮助。比如说，有些环境的外观本来就很相似，像办公室往往有很多同款式的桌椅；另一些环境则各个地方都有很大的不同。考虑到这种情况，我们会取一个先验相似度$s(v_t,v_{t-\Delta t})$，它表示某时刻关键帧图像与上一时刻的关键帧的相似性。然后，其他的分值都参照这个值进行归一化：

$$s\left(\boldsymbol{v}_t,\boldsymbol{v}_{t_j}\right)'=s\left(\boldsymbol{v}_t,\boldsymbol{v}_{t_j}\right)/s\left(\boldsymbol{v}_t,\boldsymbol{v}_{t-\Delta t}\right). \tag{12.10}$$

站在这个角度上，我们说：如果当前帧与之前某关键帧的相似度超过当前帧与上一个关键帧相似度的3倍，就认为可能存在回环。这个步骤避免了引入绝对的相似性阈值，使得算法能够适应更多的环境。

12.5.3 关键帧的处理

在检测回环时，我们必须考虑到关键帧的选取。如果关键帧选得太近，那么将导致两个关键帧之间的相似性过高，相比之下不容易检测出

历史数据中的回环。比如，检测结果经常是第n帧和第n-2帧、第n-3帧最为相似，这种结果似乎太平凡了，意义不大。所以从实践上说，用于回环检测的帧最好是稀疏一些，彼此之间不太相同，又能涵盖整个环境。

另一方面，如果成功检测到了回环，比如说出现在第1帧和第n帧。那么很可能第n+1帧、第n+2帧都会和第1帧构成回环。但是，确认第1帧和第n帧之间存在回环对轨迹优化是有帮助的，而再接下去的第n+1帧、第n+2帧都会和第1帧构成回环产生的帮助就没那么大了，因为我们已经用之前的信息消除了累积误差，更多的回环并不会带来更多的信息。所以，我们会把“相近”的回环聚成一类，使算法不要反复地检测同一类的回环。

12.5.4 检测之后的验证

词袋的回环检测算法完全依赖于外观而没有利用任何的几何信息，这导致外观相似的图像容易被当成回环。并且，由于词袋不在乎单词顺序，只在意单词有无的表达方式，更容易引发感知偏差。所以，在回环检测之后，我们通常还会有一个验证步骤[80,103]。

验证的方法有很多。其一是设立回环的缓存机制，认为单次检测到的回环并不足以构成良好的约束，而在一段时间中一直检测到的回环，才认为是正确的回环。这可以看成时间上的一致性检测。另一方法是空间上的一致性检测，即对回环检测到的两个帧进行特征匹配，估计相机的运动。然后，再把运动放到之前的Pose Graph中，检查与之前的估计是否有很大的出入。总之，验证部分通常是必需的，但如何实现却是见仁见智的问题。

12.5.5 与机器学习的关系

从前面的论述中可以看出，回环检测与机器学习有着千丝万缕的关联。回环检测本身非常像是一个分类问题。与传统模式识别的区别在于，回环中的类别数量很大，而每类的样本很少——极端情况下，当机器人发生运动后，图像发生变化，就产生了新的类别，我们甚至可以把类别当成连续变量而非离散变量；而回环检测，相当于两幅图像落入同

一类，则是很少出现的。从另一个角度看，回环检测也相当于对“图像间相似性”概念的一个学习。既然人类能够掌握图像是否相似的判断，让机器学习到这样的概念也是非常有可能的。

从词袋模型来说，它本身是一个非监督的机器学习过程——构建词典相当于对特征描述子进行聚类，而树只是对所聚的类的一个快速查找的数据结构而已。既然是聚类，结合机器学习里的知识，我们至少可以问：

1.是否能对机器学习的图像特征进行聚类，而不是对SURF、ORB这样的人工设计特征进行聚类？

2.是否有更好的方式进行聚类，而不是用树结构加上K-means这些较朴素的方式？

结合目前机器学习的发展，二进制描述子的学习和无监督的聚类，都是非常有望在深度学习框架中得以解决的问题。我们也陆续看到了利用机器学习进行回环检测的工作。尽管目前词袋方法仍是主流，但笔者本人相信，未来深度学习方法很有希望打败这些人工设计特征的、“传统”的机器学习方法[104,105]。毕竟词袋方法在物体识别问题上已经明显不如神经网络了，而回环检测又是非常相似的一个问题。

习题

1.请书写计算PR曲线的小程序。用MATLAB或Python可能更加简便一些，因为它们擅长作图。

2.验证回环检测算法，需要有人工标记回环的数据集，例如[94]。然而人工标记回环是很不方便的，我们会考虑根据标准轨迹计算回环。即，如果轨迹中有两个帧的位姿非常相近，就认为它们是回环。请根据TUM数据集给出的标准轨迹，计算出一个数据集中的回环。这些回环的图像真的相似吗？

3.学习DBoW3或DBoW2库，自己寻找几张图片，看能否从中正确检测出回环。

4.调研相似性评分的常用度量方式，哪些比较常用？

5.Chow-Liu树是什么原理？它是如何被用于构建字典和回环检测的？

书籍汇总下载 http://t.cn/RDIAj5D

6.阅读文献[106]，除了词袋模型，还有哪些用于回环检测的方法？

[1]有机器学习背景的读者，应该能感受出这段话与机器学习是何等相似。你是不是已经在想如何训练网络了呢？

[2]虽然这种性质有时也会带来一些问题，例如，眼睛长在嘴巴下的脸仍是人脸吗？

[3]你会把一页只有十个单词的纸叫作字典吗？笔者相信大多数人心目中的字典都是相当厚重的。

[4]我们用了k和d表达树的分支和深度，这可能会令你想到k-d树[99]。笔者觉得虽然做法不尽相同，但它们表达的含义确实是一致的。

[5]选用它的主要原因是其对OpenCV3兼容性较好，且编译和使用都容易上手。

[6]这里的L即前文说的d。

[7]个人觉得TF-IDF称呼起来更顺口，所以后文就用英文缩写而非中文译文了。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

第13讲　建图

主要目标

1.理解单目SLAM中稠密深度估计的原理。

2.通过实验了解单目稠密重建的过程。

3.了解几种RGB-D重建中的地图形式。

本讲我们开始介绍建图部分的算法。在前端和后端中，我们重点关注同时估计相机运动轨迹与特征点空间位置的问题。然而，在实际使用SLAM时，除了对相机本体进行定位之外，还存在许多其他的需求。例如，考虑放在机器人上的SLAM，那么我们会希望地图能够用于定位、导航、避障和交互，特征点地图显然不能满足所有的这些需求。所以，本讲我们将更详细地讨论各种形式的地图，并指出目前视觉SLAM地图中存在着的缺陷。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

13.1 概述

建图（Mapping），本应该是SLAM的两大目标之一——因为SLAM被称为同时定位与建图。但是直到现在，我们讨论的都是定位问题，包括通过特征点的定位、直接法的定位，以及后端优化。那么，这是否暗示建图在SLAM里没有那么重要，所以我们直到本讲才开始讨论呢？

答案是否定的。事实上，在经典的SLAM模型中，我们所谓的地图，即所有路标点的集合。一旦确定了路标点的位置，那就可以说我们完成了建图。于是，前面说的视觉里程计也好，Bundle Adjustment也好，事实上都建模了路标点的位置，并对它们进行优化。从这个角度上说，我们已经探讨了建图问题。那么为何还要单独列一讲建图呢？

这是因为人们对建图的需求不同。SLAM作为一种底层技术，往往是用来为上层应用提供信息的。如果上层是机器人，那么应用层的开发者可能希望使用SLAM来做全局的定位，并且让机器人在地图中导航——例如扫地机需要完成扫地工作，希望计算一条能够覆盖整张地图的路径。或者，如果上层是一个增强现实设备，那么开发者可能希望将虚拟物体叠加在现实物体之中，特别地，还可能需要处理虚拟物体和真实物体的遮挡关系。

我们发现，应用层面对于“定位”的需求是相似的，它们希望SLAM提供相机或搭载相机的主体的空间位姿信息。而对于地图，则存在着许多不同的需求。在视觉SLAM看来，“建图”是服务于“定位”的；但是在应用层面看来，“建图”明显还带有许多其他的需求。关于地图的用处，我们大致归纳如下：

1.定位。定位是地图的一项基本功能。在前面的视觉里程计部分，我们讨论了如何利用局部地图来实现定位。在回环检测部分，我们也看到，只要有全局的描述子信息，我们也能通过回环检测确定机器人的位置。更进一步，我们还希望能够把地图保存下来，让机器人在下次开机后依然能在地图中定位，这样只需对地图进行一次建模，而不是每次启动机器人都重新做一次完整的SLAM。

2.导航。导航是指机器人能够在地图中进行路径规划，在任意两个

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

地图点间寻找路径，然后控制自己运动到目标点的过程。该过程中，我们至少需要知道地图中哪些地方不可通过，而哪些地方是可以通过的。这就超出了稀疏特征点地图的能力范围，我们必须有另外的地图形式。稍后我们会说，这至少得是一种稠密的地图。

3.避障。避障也是机器人经常碰到的一个问题。它与导航类似，但更注重局部的、动态的障碍物的处理。同样，仅有特征点，我们无法判断某个特征点是否为障碍物，所以需要稠密地图。

4.重建。有时候，我们希望利用SLAM获得周围环境的重建效果，并把它展示给其他人看。这种地图主要用于向人展示，所以我们希望它看上去比较舒服、美观。或者，我们也可以把该地图用于通信，使其他人能够远程地观看我们重建得到的三维物体或场景——例如三维的视频通话或者网上购物等。这种地图亦是稠密的，并且我们还对它的外观有一些要求。我们可能不满足于稠密点云重建，更希望能够构建带纹理的平面，就像电子游戏中的三维场景那样。

5.交互。交互主要指人与地图之间的互动。例如，在增强现实中，我们会在房间里放置虚拟的物体，并与这些虚拟物体之间有一些互动——比方说我会点击墙面上放着的虚拟网页浏览器来观看视频，或者向墙面投掷物体，希望它们有（虚拟的）物理碰撞。另一方面，机器人应用中也会有与人、与地图之间的交互。例如，机器人可能会收到命令“取桌子上的报纸”，那么，除了有环境地图之外，机器人还需要知道哪一块地图是“桌子”，什么叫作“之上”，什么又叫作“报纸”。这需要机器人对地图有更高层面的认知——亦称为语义地图。

图13-1形象地解释了上面讨论的各种地图类型与用途之间的关系。我们之前的讨论，基本集中于“稀疏路标地图”部分，还没有探讨稠密地图。所谓稠密地图是相对于稀疏地图而言的。稀疏地图只建模感兴趣的部分，也就是前面说了很久的特征点（路标点）。而稠密地图是指，建模所有看到过的部分。对于同一张桌子，稀疏地图可能只建模了桌子的四个角，而稠密地图则会建模整个桌面。虽然从定位角度看，只有四个角的地图也可以用于对相机进行定位，但由于我们无法从四个角推断这几个点之间的空间结构，所以无法仅用四个角来完成导航、避障等需要稠密地图才能完成的工作。

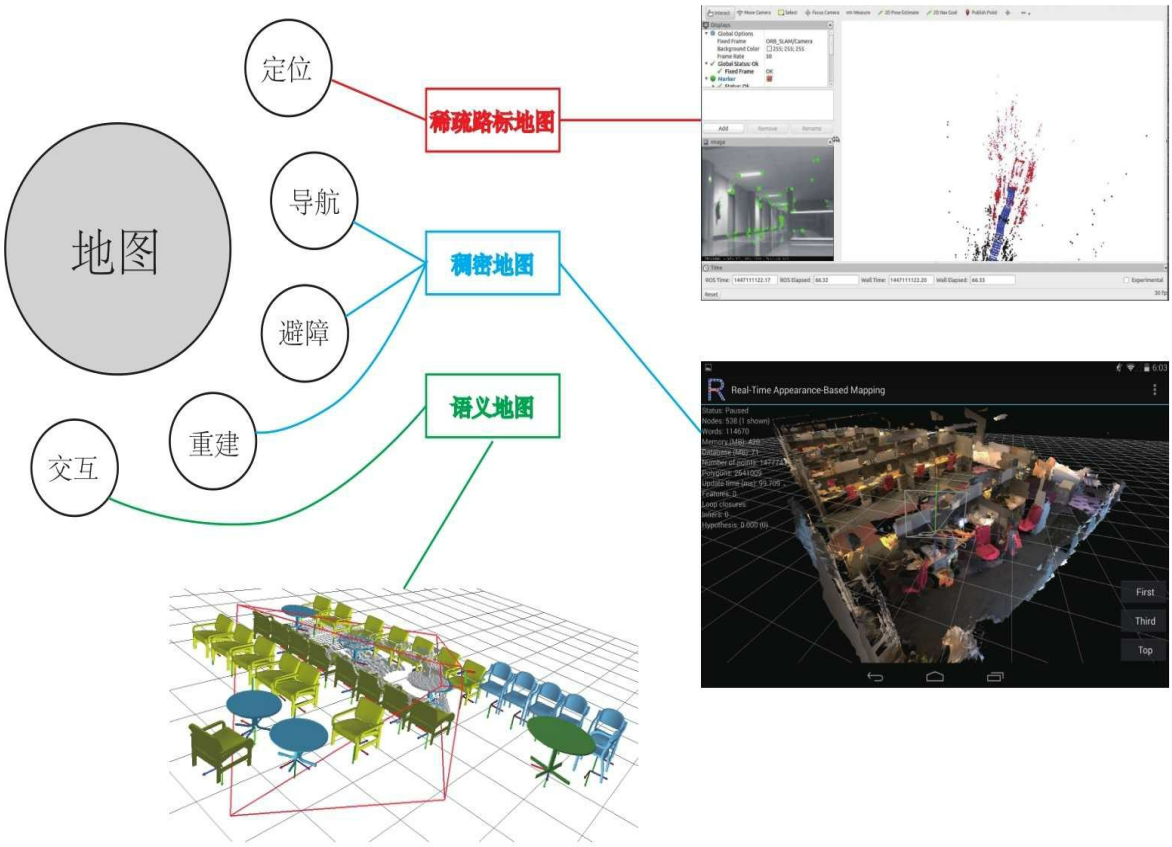


图13-1　各种地图的示意图。三种例子地图分别来自文献[73,107,108]。

从上面的讨论中可以看出，稠密地图占据着一个非常重要的位置。于是，剩下的问题是：通过视觉SLAM能建立稠密地图吗？如果能，怎么建呢？

书籍汇总下载 http://t.cn/RDIAj5D

13.2 单目稠密重建

13.2.1 立体视觉

视觉SLAM的稠密重建问题是本讲的第一个重要话题。相机，很久以来被认为是只有角度的传感器（Bearing only）。单幅图像中的像素，只能提供物体与相机成像平面的角度及物体采集到的亮度，而无法提供物体的距离（Range）。而在稠密重建中，我们需要知道每一个像素点（或大部分像素点）的距离，对此大致上有如下解决方案：

1.使用单目相机，通过移动相机之后进行三角化测量像素的距离。

2.使用双目相机，利用左右目的视差计算像素的距离（多目原理相同）。

3.使用RGB-D相机直接获得像素距离。

前两种方式称为立体视觉（Stereo Vision），其中移动单目的又称为移动视角的立体视觉（Moving View Stereo）。相比于RGB-D直接测量的深度，单目和双目对深度的获取往往是“费力不讨好”的——我们需要花费大量的计算，最后得到一些不怎么可靠的[1]深度估计。当然，RGB-D也有一些量程、应用范围和光照的限制，不过相比于单目和双目的结果，使用RGB-D进行稠密重建往往是更常见的选择。而单目、双目的好处是，在目前RGB-D还无法很好应用的室外、大场景场合中，仍能通过立体视觉估计深度信息。

话虽如此，本节我们将带领读者实现一遍单目的稠密估计，体验为何说它是费力不讨好的。我们从最简单的情况开始说起：在给定相机轨迹的基础上，如何根据一段时间的视频序列来估计某幅图像的深度。换言之，我们不考虑SLAM，先来考虑略为简单的建图问题。

假定有一段视频序列，我们通过某种魔法得到了每一帧对应的轨迹（当然也很可能是由视觉里程计前端估计所得）。现在我们以第一幅图像为参考帧，计算参考帧中每一个像素的深度（或者说距离）。首先，请回忆在特征点部分我们是如何完成该过程的：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

1.首先，我们对图像提取特征，并根据描述子计算了特征之间的匹配。换言之，通过特征，我们对某一个空间点进行了跟踪，知道了它在各个图像之间的位置。

2.然后，由于我们无法仅用一幅图像确定特征点的位置，所以必须通过不同视角下的观测估计它的深度，原理即前面讲过的三角测量。

在稠密深度图估计中，不同之处在于，我们无法把每个像素都当作特征点计算描述子。因此，稠密深度估计问题中，匹配就成为很重要的一环：如何确定第一幅图的某像素出现在其他图里的位置呢？这需要用到极线搜索和块匹配技术[109]。然后，当我们知道了某个像素在各个图中的位置，就能像特征点那样，利用三角测量确定它的深度。不过不同的是，在这里我们要使用很多次三角测量让深度估计收敛，而不仅是一次。我们希望深度估计能够随着测量的增加从一个非常不确定的量，逐渐收敛到一个稳定值。这就是深度滤波器技术。所以，下面的内容将主要围绕这个主题展开。

13.2.2 极线搜索与块匹配

我们先来探讨不同视角下观察同一个点产生的几何关系。这非常像在第7.3节讨论的对极几何关系。请看图13-2。左边的相机观测到了某个像素p_1。由于这是一个单目相机，我们无从知道它的深度，所以假设这个深度可能在某个区域之内，不妨说是某最小值到无穷远之间：$(d_{min},+\infty)$。因此，该像素对应的空间点就分布在某条线段（本例中是射线）上。在另一个视角（右侧相机）看来，这条线段的投影也形成图像平面上的一条线，我们知道这称为极线。当知道两部相机间的运动时，这条极线也是能够确定的[2]。那么问题就是：极线上的哪一个点是我们刚才看到的p_1点呢？

书籍汇总下载 http://t.cn/RDIAj5D

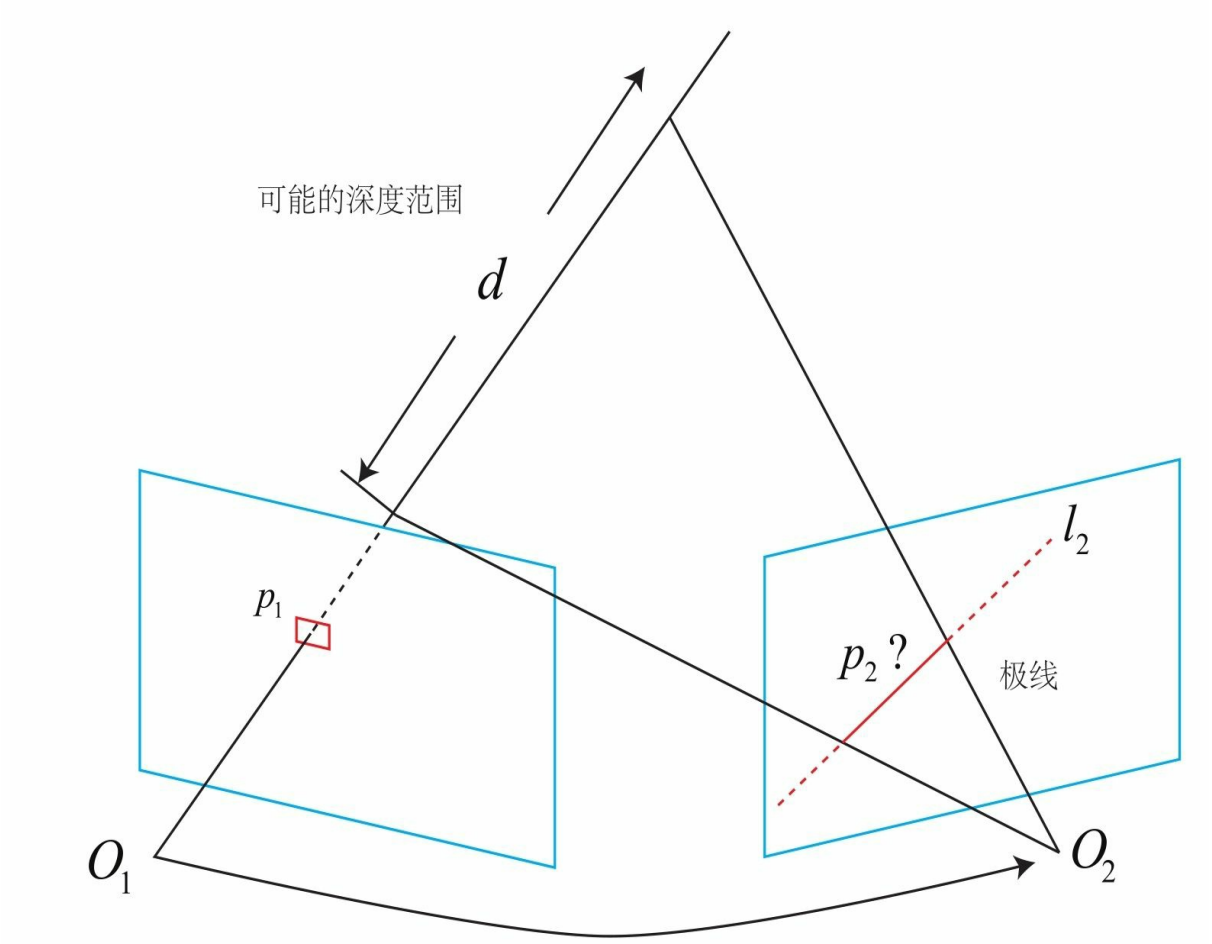


图13-2　极线搜索示意图。

重复一遍，在特征点方法中，我们通过特征匹配找到了p_2的位置。然而现在我们没有描述子，所以只能在极线上搜索和p_1长得比较相似的点。再具体地说，我们可能沿着第二幅图像中的极线的某一头走到另一头，逐个比较每个像素与p_1的相似程度。从直接比较像素的角度来看，这种做法倒是和直接法是异曲同工的。

在直接法的讨论中我们知道，比较单个像素的亮度值并不一定稳定可靠。一件很明显的事情就是：万一极线上有很多和p_1相似的点，我们怎么确定哪一个是真实的呢？这似乎回到了我们在回环检测当中说到的问题：如何确定两幅图像（或两个点）的相似性？回环检测是通过词袋来解决的，但这里由于没有特征，所以只好寻求另外的途径。

一种直观的想法是：既然单个像素的亮度没有区分性，那是否可以

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

比较像素块呢？我们在p_1周围取一个大小为$w \times w$的小块，然后在极线上也取很多同样大小的小块进行比较，就可以在一定程度上提高区分性。这就是所谓的块匹配。注意到在这个过程中，只有假设在不同图像间整个小块的灰度值不变，这种比较才有意义。所以算法的假设，从像素的灰度不变性，变成了图像块的灰度不变性——在一定程度上变得更强了。

好了，现在我们取了p_1周围的小块，并且在极线上也取了很多个小块。不妨把p_1周围的小块记成$A \in \mathrm{R}^{w \times w}$，把极线上的$n$个小块记成$B_i, i=1,\cdots,n$。那么，如何计算小块与小块间的差异呢？有若干种不同的计算方法：

1.SAD（SumofAbsoluteDi ff erence）。顾名思义，即取两个小块的差的绝对值之和：

$$S(\boldsymbol{A}, \boldsymbol{B})_{\mathrm{SAD}} = \sum_{i,j} |\boldsymbol{A}(i,j) - \boldsymbol{B}(i,j)|. \tag{13.1}$$

2.SSD。这里的SSD并不是指大家熟悉的固态硬盘，而是Sum of Squared Distance（平方和）的意思：

$$S(\boldsymbol{A}, \boldsymbol{B})_{\mathrm{SSD}} = \sum_{i,j} (\boldsymbol{A}(i,j) - \boldsymbol{B}(i,j))^2. \tag{13.2}$$

3.NCC（Normalized Cross Correlation，归一化互相关）。这种方式比前两种要复杂一些，它计算的是两个小块的相关性：

$$S(\boldsymbol{A}, \boldsymbol{B})_{\mathrm{NCC}} = \frac{\sum_{i,j} \boldsymbol{A}(i,j)\boldsymbol{B}(i,j)}{\sqrt{\sum_{i,j} \boldsymbol{A}(i,j)^2 \sum_{i,j} \boldsymbol{B}(i,j)^2}}. \tag{13.3}$$

请注意，由于这里用的是相关性，所以相关性接近0表示两幅图像不相似，而接近1才表示相似。前面两种距离则是反过来的，接近0表示相似，而大的数值表示不相似。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

和我们遇到过的许多情形一样，这些计算方式往往存在一个精度-效率之间的矛盾。精度好的方法往往需要复杂的计算，而简单的快速算法又往往效果不佳。这需要我们在实际工程中进行取舍。另外，除了这些简单版本之外，我们可以先把每个小块的均值去掉，称为去均值的SSD、去均值的NCC，等等。去掉均值之后，我们允许像“小块B比A整体上亮一些，但仍然很相似”这样的情况[3]，因此比之前的更加可靠一些。如果读者对更多的块匹配度量方法感兴趣，建议阅读文献[110,111]作为补充材料。

现在，我们在极线上计算了A与每一个B_i的相似性度量。为了方便叙述，假设我们用了NCC，那么，我们将得到一个沿着极线的NCC分布。这个分布的形状严重取决于图像本身的样子，如图13-3所示。在搜索距离较长的情况下，我们通常会得到一个非凸函数：这个分布存在着许多峰值，然而真实的对应点必定只有一个。在这种情况下，我们会倾向于使用概率分布来描述深度值，而非用某个单一的数值来描述深度。于是，我们的问题就转到了在不断对不同图像进行极线搜索时，我们估计的深度分布将发生怎样的变化——这就是所谓的深度滤波器。

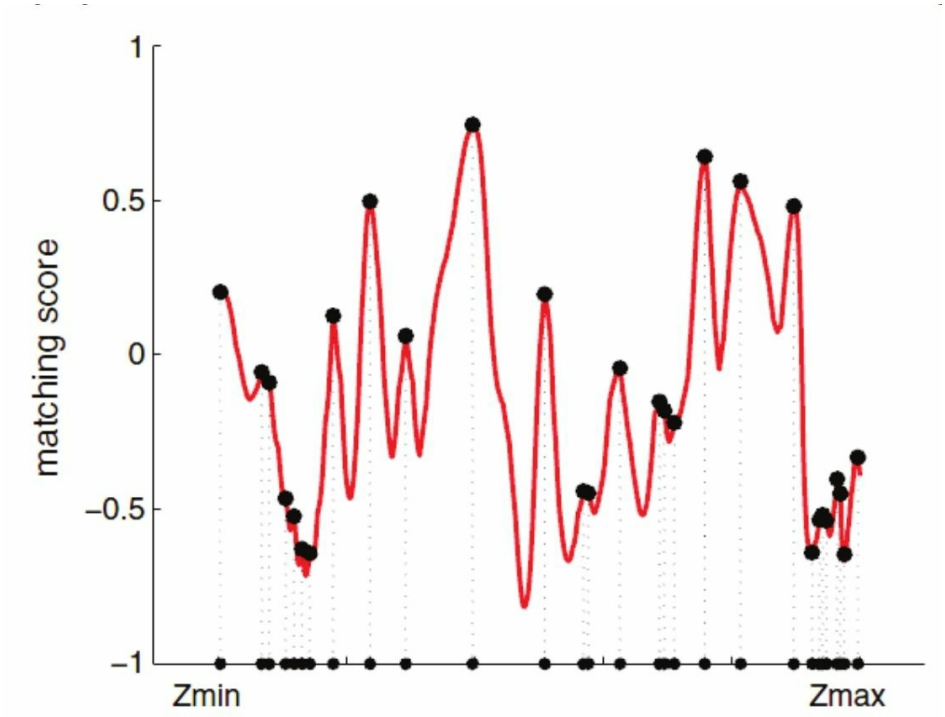


图13-3　匹配得分沿距离的分布，图像来自文献[112]。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

13.2.3 高斯分布的深度滤波器

对像素点深度的估计，本身亦可建模为一个状态估计问题，于是就自然存在滤波器与非线性优化两种求解思路。虽然非线性优化效果较好，但是在SLAM这种实时性要求较强的场合，考虑到前端已经占据了不少的计算量，建图方面则通常采用计算量较少的滤波器方式了。这也是本节讨论深度滤波器的目的。

对深度的分布假设存在着若干种不同的做法。首先，在比较简单的假设条件下，我们可以假设深度值服从高斯分布，得到一种类卡尔曼式的方法（我们稍后会看到）。而另一方面，在[112,56]等文献中，亦采用了均匀-高斯混合分布的假设，推导了另一种形式更为复杂的深度滤波器。本着简单易用的原则，我们先来介绍并演示高斯分布假设下的深度滤波器，然后把均匀-高斯混合分布的滤波器作为习题。

设某个像素点的深度d服从：

$$P(d) = N(\mu, \sigma^2). \tag{13.4}$$

而每当新的数据到来，我们都会观测到它的深度。同样，假设这次观测亦是一个高斯分布：

$$P(d_{\text{obs}}) = N(\mu_{\text{obs}}, \sigma_{\text{obs}}^2). \tag{13.5}$$

于是，我们的问题是，如何使用观测的信息更新原先d的分布。这正是一个信息融合问题。根据附录A，我们明白两个高斯分布的乘积依然是一个高斯分布。设融合后的d的分布为$N(\mu_{\text{fuse}}, \sigma_{\text{fuse}}^2)$，那么根据高斯分布的乘积，有：

$$\mu_{\text{fuse}} = \frac{\sigma_{\text{obs}}^2 \mu + \sigma^2 \mu_{\text{obs}}}{\sigma^2 + \sigma_{\text{obs}}^2}, \quad \sigma_{\text{fuse}}^2 = \frac{\sigma^2 \sigma_{\text{obs}}^2}{\sigma^2 + \sigma_{\text{obs}}^2}. \tag{13.6}$$

由于我们仅有观测方程而没有运动方程，所以这里深度仅用到了信息融合部分，而无须像完整的卡尔曼那样进行预测和更新。可以看到融合的方程确实比较浅显易懂，不过问题仍然存在：如何确定我们观测到深度的分布呢？即，如何计算μ_{obs},σ_{obs}呢？

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

关于$\mu_{\text{obs}},\sigma_{\text{obs}}$，亦存在一些不同的处理方式。例如，文献[59]考虑了几何不确定性和光度不确定性二者之和，而[112]则仅考虑了几何不确定性。我们暂时只考虑由几何关系带来的不确定性。现在，假设我们通过极线搜索和块匹配确定了参考帧某个像素在当前帧的投影位置。那么，这个位置对深度的不确定性有多大呢？

以图13-4为例。考虑某次极线搜索，我们找到了p_1对应的p_2点，从而观测到了p_1的深度值，认为p_1对应的三维点为P。从而，可记O_1P为p，O_1O_2为相机的平移t，O_2P记为a。并且，把这个三角形的下面两个角记作α,β。现在，考虑极线l_2上存在着一个像素大小的误差，使得β角变成了β'，而p也变成了p'，并记上面那个角为γ。我们要问的是，这一个像素的误差会导致p'与p产生多大的差距呢？

这是一个典型的几何问题。我们来列写这些量之间的几何关系。显然有：

$$\begin{aligned} \boldsymbol{a} &= \boldsymbol{p} - \boldsymbol{t} \\ \alpha &= \arccos\langle \boldsymbol{p}, \boldsymbol{t} \rangle \\ \beta &= \arccos\langle \boldsymbol{a}, -\boldsymbol{t} \rangle . \end{aligned} \tag{13.7}$$

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

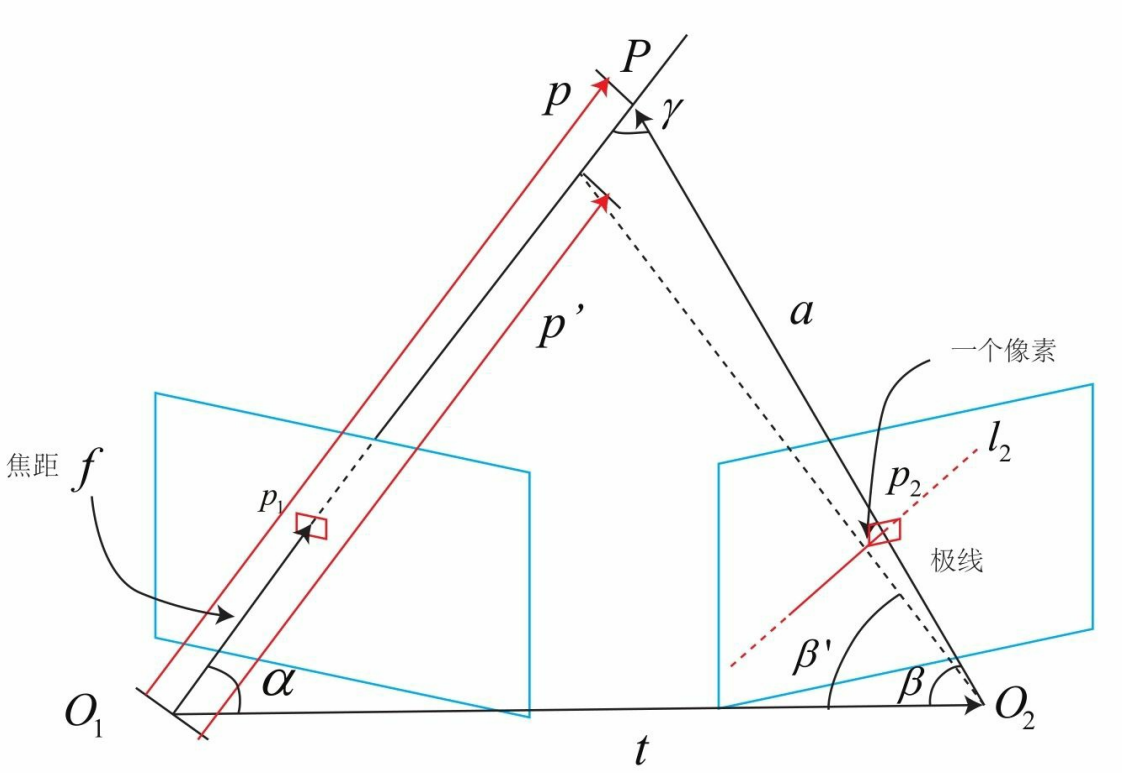


图13-4　不确定性分析。

对p_2扰动一个像素，将使得β产生一个变化量$\delta\beta$，由于相机焦距为f，于是：

$$\delta\beta = \arctan\frac{1}{f}. \tag{13.8}$$

所以：

$$\begin{aligned}\beta' &= \beta + \delta\beta \\ \gamma &= \pi - \alpha - \beta'.\end{aligned} \tag{13.9}$$

于是，由正弦定理，p'的大小可以求得：

$$\|\boldsymbol{p}'\| = \|\boldsymbol{t}\|\frac{\sin\beta'}{\sin\gamma}. \tag{13.10}$$

由此，我们确定了由单个像素的不确定引起的深度不确定性。如果认为极线搜索的块匹配仅有一个像素的误差，那么就可以设：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$\sigma_{\text{obs}} = \|\boldsymbol{p}\| - \|\boldsymbol{p}'\|. \tag{13.11}$$

当然，如果极线搜索的不确定性大于一个像素，我们亦可按照此推导来放大这个不确定性。接下来的深度数据融合，已经在前面介绍过了。在实际工程中，当不确定性小于一定阈值之后，就可以认为深度数据已经收敛了。

综上所述，我们给出了估计稠密深度的一个完整的过程：

1.假设所有像素的深度满足某个初始的高斯分布。

2.当新数据产生时，通过极线搜索和块匹配确定投影点位置。

3.根据几何关系计算三角化后的深度及不确定性。

4.将当前观测融合进上一次的估计中。若收敛则停止计算，否则返回第2步。

这些步骤组成了一套可行的深度估计方式。它的实际结果如何，我们将在实践部分进行演示。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

13.3 实践：单目稠密重建

本节的示例程序将使用REMODE[113,109]的测试数据集。它提供了一架无人机采集的单目俯视图像，共有200张，同时提供了每张图像的真实位姿。下面我们来考虑，在这些数据的基础上估算第一帧图像每个像素对应的深度值，即进行单目稠密重建。

首先，请读者从http://rpg.ifi.uzh.ch/datasets/remode_test_data.zip处下载示例程序所用的数据。你可以使用网页浏览器或下载工具进行下载。解压后，将在test_data/Images中发现从0至200的所有图像，并在test_data目录下看到一个文本文件，它记录了每幅图像对应的位姿：

```
scene_000.png 1.086410 4.766730 -1.449960 0.789455 0.051299 -0.000779 0.611661
scene_001.png 1.086390 4.766370 -1.449530 0.789180 0.051881 -0.001131 0.611966
scene_002.png 1.086120 4.765520 -1.449090 0.788982 0.052159 -0.000735 0.612198
......
```

图13-5展示了若干时刻的图像。可以看到场景主要由地面、桌子及桌子上的杂物组成。如果深度估计大致正确，那么我们至少可以看出桌子与地面的深度值不同之处。下面，我们按照之前的讲解书写稠密深度估计程序。为了方便理解，程序书写成了C语言风格，放在单个文件中。本程序稍微有点长，在书中重点讲解几个重要函数，其余内容请读者对照GitHub的源码进行阅读。

slambook/ch13/dense_monocular/dense_mapping.cpp（片段）

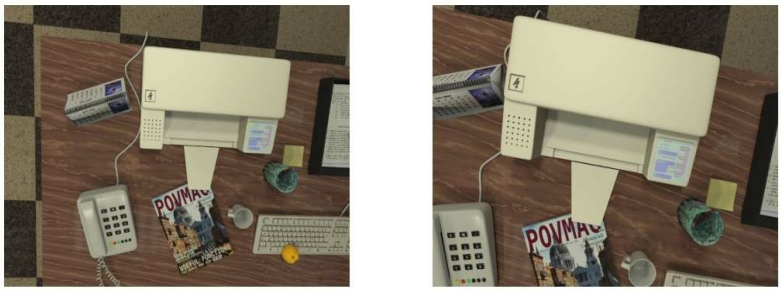
电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```cpp
#include <iostream>
#include <vector>
#include <fstream>
using namespace std;
#include <boost/timer.hpp>

// for sophus
#include <sophus/se3.h>
```



$t=0$ $t=50$ $t=100$



$t=150$ $t=200$

图13-5　数据集图示。

书籍汇总下载 http://t.cn/RDIAj5D

```
9  using Sophus::SE3;
10
11 // for eigen
12 #include <Eigen/Core>
13 #include <Eigen/Geometry>
14 using namespace Eigen;
15
16 #include <opencv2/core/core.hpp>
17 #include <opencv2/highgui/highgui.hpp>
18 #include <opencv2/imgproc/imgproc.hpp>
19
20 using namespace cv;
21
22 /*********************************************
23 * 本程序演示了单目相机在已知轨迹下的稠密深度估计
24 * 使用极线搜索 + NCC匹配的方式，与本书13.2节对应
25 * 请注意，本程序并不完美，你完全可以改进它。
26 *********************************************/
27
28 // ------------------------------------------------------------------
29 // parameters
30 const int boarder = 20;     // 边缘宽度
31 const int width = 640;      // 宽度
32 const int height = 480;     // 高度
33 const double fx = 481.2f;   // 相机内参
34 const double fy = -480.0f;
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
35 const double cx = 319.5f;
36 const double cy = 239.5f;
37 const int ncc_window_size = 2;  // NCC 取的窗口半宽度
38 const int ncc_area = (2*ncc_window_size+1)*(2*ncc_window_size+1); // NCC窗口面积
39 const double min_cov = 0.1; // 收敛判定：最小方差
40 const double max_cov = 10;  // 发散判定：最大方差
41
42 // ------------------------------------------------------------------
43 // 重要的函数
44 // 从 REMODE 数据集读取数据
45 bool readDatasetFiles(
46     const string& path,
47     vector<string>& color_image_files,
48     vector<SE3>& poses
49 );
50
51 // 根据新的图像更新深度估计
52 bool update(
53     const Mat& ref,
54     const Mat& curr,
55     const SE3& T_C_R,
56     Mat& depth,
57     Mat& depth_cov
58 );
59
60 // 极线搜索
61 bool epipolarSearch(
62     const Mat& ref,
63     const Mat& curr,
64     const SE3& T_C_R,
65     const Vector2d& pt_ref,
66     const double& depth_mu,
67     const double& depth_cov,
68     Vector2d& pt_curr
69 );
70
71 // 更新深度滤波器
72 bool updateDepthFilter(
73     const Vector2d& pt_ref,
74     const Vector2d& pt_curr,
75     const SE3& T_C_R,
76     Mat& depth,
77     Mat& depth_cov
78 );
79
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
// 计算 NCC 评分
double NCC( const Mat& ref, const Mat& curr, const Vector2d& pt_ref, const Vector2d
& pt_curr );

// 双线性灰度插值
inline double getBilinearInterpolatedValue( const Mat& img, const Vector2d& pt ) {
    uchar* d = & img.data[ int(pt(1,0))*img.step+int(pt(0,0)) ];
    double xx = pt(0,0) - floor(pt(0,0));
    double yy = pt(1,0) - floor(pt(1,0));
    return  (( 1-xx ) * ( 1-yy ) * double(d[0]) +
        xx* ( 1-yy ) * double(d[1]) +
        ( 1-xx ) *yy* double(d[img.step]) +
        xx*yy*double(d[img.step+1]))/255.0;
}

// ------------------------------------------------------------------
// 一些小工具
// 显示估计的深度图
bool plotDepth( const Mat& depth );

// 像素到相机坐标系
inline Vector3d px2cam ( const Vector2d px ) {
    return Vector3d (
        (px(0,0) - cx)/fx,
        (px(1,0) - cy)/fy,
        1
    );
}

// 相机坐标系到像素
inline Vector2d cam2px ( const Vector3d p_cam ) {
    return Vector2d (
        p_cam(0,0)*fx/p_cam(2,0) + cx,
        p_cam(1,0)*fy/p_cam(2,0) + cy
    );
}

// 检测一个点是否在图像边框内
inline bool inside( const Vector2d& pt ) {
    return pt(0,0) >= boarder && pt(1,0)>=boarder
        && pt(0,0)+boarder<width && pt(1,0)+boarder<=height;
}

int main( int argc, char** argv )
{
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
124     if ( argc != 2 )
125     {
126         cout<<"Usage: dense_mapping path_to_test_dataset"<<endl;
127         return -1;
128     }
129
130     // 从数据集读取数据
131     vector<string> color_image_files;
132     vector<SE3> poses_TWC;
133     bool ret = readDatasetFiles( argv[1], color_image_files, poses_TWC );
134     if ( ret==false )
135     {
136         cout<<"Reading image files failed!"<<endl;
137         return -1;
138     }
139     cout<<"read total "<<color_image_files.size()<<" files."<<endl;
140
141     // 第一张图
142     Mat ref = imread( color_image_files[0], 0 );      // gray-scale image
143     SE3 pose_ref_TWC = poses_TWC[0];
144     double init_depth   = 3.0;    // 深度初始值
145     double init_cov2    = 3.0;    // 方差初始值
146     Mat depth( height, width, CV_64F, init_depth );         // 深度图
147     Mat depth_cov( height, width, CV_64F, init_cov2 );      // 深度图方差
148
149     for ( int index=1; index<color_image_files.size(); index++ )
150     {
151         cout<<"*** loop "<<index<<" ***"<<endl;
152         Mat curr = imread( color_image_files[index], 0 );
153         if (curr.data == nullptr) continue;
154         SE3 pose_curr_TWC = poses_TWC[index];
155         SE3 pose_T_C_R = pose_curr_TWC.inverse() * pose_ref_TWC;
156             // 坐标转换关系： T_C_W * T_W_R = T_C_R
157         update( ref, curr, pose_T_C_R, depth, depth_cov );
158         plotDepth( depth );
159         imshow("image", curr);
160         waitKey(1);
161     }
162
163
164     return 0;
165 }
166
167 // 对整个深度图进行更新
168 bool update(const Mat& ref, const Mat& curr, const SE3& T_C_R, Mat& depth, Mat&
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
depth_cov )
{
#pragma omp parallel for
    for ( int x=boarder; x<width-boarder; x++ )
#pragma omp parallel for
        for ( int y=boarder; y<height-boarder; y++ )
        {
            // 遍历每个像素
            if ( depth_cov.ptr<double>(y)[x] < min_cov
                || depth_cov.ptr<double>(y)[x] > max_cov ) // 深度已收敛或发散
                continue;
            // 在极线上搜索 (x,y) 的匹配
            Vector2d pt_curr;
            bool ret = epipolarSearch (
                ref,
                curr,
                T_C_R,
                Vector2d(x,y),
                depth.ptr<double>(y)[x],
                sqrt(depth_cov.ptr<double>(y)[x]),
                pt_curr
            );

            if ( ret == false ) // 匹配失败
                continue;

            // 取消该注释以显示匹配
            // showEpipolarMatch( ref, curr, Vector2d(x,y), pt_curr );

            // 匹配成功，更新深度图
            updateDepthFilter( Vector2d(x,y), pt_curr, T_C_R, depth, depth_cov );
        }
}

// 极线搜索
// 方法见本书 13.2、13.3 两节
bool epipolarSearch(
    const Mat& ref, const Mat& curr,
    const SE3& T_C_R, const Vector2d& pt_ref,
    const double& depth_mu, const double& depth_cov,
    Vector2d& pt_curr
)
{
    Vector3d f_ref = px2cam( pt_ref );
    f_ref.normalize();
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
213     Vector3d P_ref = f_ref*depth_mu;    // 参考帧的 P 向量
214
215     Vector2d px_mean_curr = cam2px( T_C_R*P_ref ); // 按深度均值投影的像素
216     double d_min = depth_mu-3*depth_cov, d_max = depth_mu+3*depth_cov;
217     if ( d_min<0.1 ) d_min = 0.1;
218     Vector2d px_min_curr = cam2px( T_C_R*(f_ref*d_min) );   // 按最小深度投影的像素
219     Vector2d px_max_curr = cam2px( T_C_R*(f_ref*d_max) );   // 按最大深度投影的像素
220
221     Vector2d epipolar_line = px_max_curr - px_min_curr; // 极线（线段形式）
222     Vector2d epipolar_direction = epipolar_line;        // 极线方向
223     epipolar_direction.normalize();
224     double half_length = 0.5*epipolar_line.norm();  // 极线线段的半长度
225     if ( half_length>100 ) half_length = 100;   // 我们不希望搜索太多东西
226
227     // 取消此句注释以显示极线（线段）
228     // showEpipolarLine( ref, curr, pt_ref, px_min_curr, px_max_curr );
229
230     // 在极线上搜索，以深度均值点为中心，左右各取半长度
231     double best_ncc = -1.0;
232     Vector2d best_px_curr;
233     for ( double l=-half_length; l<=half_length; l+=0.7 )  // l+=sqrt(2)
234     {
235         Vector2d px_curr = px_mean_curr + l*epipolar_direction;  // 待匹配点
236         if ( !inside(px_curr) )
237         continue;
238         // 计算待匹配点与参考帧的 NCC
239         double ncc = NCC( ref, curr, pt_ref, px_curr );
240         if ( ncc>best_ncc )
241         {
242             best_ncc = ncc;
243             best_px_curr = px_curr;
244         }
245     }
246     if ( best_ncc < 0.85f )      // 只相信 NCC 很高的匹配
247         return false;
248     pt_curr = best_px_curr;
249     return true;
250 }
251
252 double NCC (
253     const Mat& ref, const Mat& curr,
254     const Vector2d& pt_ref, const Vector2d& pt_curr
255 )
256 {
257     // 零均值-归一化互相关
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
    // 先算均值
    double mean_ref = 0, mean_curr = 0;
    vector<double> values_ref, values_curr; // 参考帧和当前帧的均值
    for ( int x=-ncc_window_size; x<=ncc_window_size; x++ )
        for ( int y=-ncc_window_size; y<=ncc_window_size; y++ )
        {
            double value_ref = double(ref.ptr<uchar>( int(y+pt_ref(1,0)) )[ int(x+
            pt_ref(0,0)) ])/255.0;
            mean_ref += value_ref;

            double value_curr = getBilinearInterpolatedValue( curr, pt_curr+
            Vector2d(x,y) );
            mean_curr += value_curr;

            values_ref.push_back(value_ref);
            values_curr.push_back(value_curr);
        }

    mean_ref /= ncc_area;
    mean_curr /= ncc_area;

    // 计算 Zero mean NCC
    double numerator = 0, demoniator1 = 0, demoniator2 = 0;
    for ( int i=0; i<values_ref.size(); i++ )
    {
        double n = (values_ref[i]-mean_ref) * (values_curr[i]-mean_curr);
        numerator += n;
        demoniator1 += (values_ref[i]-mean_ref)*(values_ref[i]-mean_ref);
        demoniator2 += (values_curr[i]-mean_curr)*(values_curr[i]-mean_curr);
    }
    return numerator / sqrt( demoniator1*demoniator2+1e-10 );   // 防止分母出现零
}

bool updateDepthFilter(
    const Vector2d& pt_ref,
    const Vector2d& pt_curr,
    const SE3& T_C_R,
    Mat& depth,
    Mat& depth_cov
)
{
    // 用三角化计算深度
    SE3 T_R_C = T_C_R.inverse();
    Vector3d f_ref = px2cam( pt_ref );
    f_ref.normalize();
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
301     Vector3d f_curr = px2cam( pt_curr );
302     f_curr.normalize();
303
304     // 方程参照本书第 7 讲三角化一节
305     Vector3d t = T_R_C.translation();
306     Vector3d f2 = T_R_C.rotation_matrix() * f_curr;
307     Vector2d b = Vector2d ( t.dot ( f_ref ), t.dot ( f2 ) );
308     double A[4];
309     A[0] = f_ref.dot ( f_ref );
310     A[2] = f_ref.dot ( f2 );
311     A[1] = -A[2];
312     A[3] = - f2.dot ( f2 );
313     double d = A[0]*A[3]-A[1]*A[2];
314     Vector2d lambdavec =
315         Vector2d (  A[3] * b ( 0,0 ) - A[1] * b ( 1,0 ),
316             -A[2] * b ( 0,0 ) + A[0] * b ( 1,0 )) /d;
317     Vector3d xm = lambdavec ( 0,0 ) * f_ref;
318     Vector3d xn = t + lambdavec ( 1,0 ) * f2;
319     Vector3d d_esti = ( xm+xn ) / 2.0;  // 三角化算得的深度向量
320     double depth_estimation = d_esti.norm();   // 深度值
321
322     // 计算不确定性（以一个像素为误差）
323     Vector3d p = f_ref*depth_estimation;
324     Vector3d a = p - t;
325     double t_norm = t.norm();
326     double a_norm = a.norm();
327     double alpha = acos( f_ref.dot(t)/t_norm );
328     double beta = acos( -a.dot(t)/(a_norm*t_norm));
329     double beta_prime = beta + atan(1/fx);
330     double gamma = M_PI - alpha - beta_prime;
331     double p_prime = t_norm * sin(beta_prime) / sin(gamma);
332     double d_cov = p_prime - depth_estimation;
333     double d_cov2 = d_cov*d_cov;
334
335     // 高斯融合
336     double mu = depth.ptr<double>( int(pt_ref(1,0)) )[ int(pt_ref(0,0)) ];
337     double sigma2 = depth_cov.ptr<double>( int(pt_ref(1,0)) )[ int(pt_ref(0,0)) ];
338
339     double mu_fuse = (d_cov2*mu+sigma2*depth_estimation) / ( sigma2+d_cov2);
340     double sigma_fuse2 = ( sigma2 * d_cov2 ) / ( sigma2 + d_cov2 );
341
342     depth.ptr<double>( int(pt_ref(1,0)) )[ int(pt_ref(0,0)) ] = mu_fuse;
343     depth_cov.ptr<double>( int(pt_ref(1,0)) )[ int(pt_ref(0,0)) ] = sigma_fuse2;
344
345     return true;
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
346 }
347
348 // 其他次要的函数略
```

如果读者理解了上一节内容，相信读懂此处源代码亦不是难事。尽管如此，我们对几个关键函数稍做说明：

1.main函数非常简单。它只负责从数据集中读取图像，然后交给update函数，对深度图进行更新。

2.update函数中，我们遍历了参考帧的每个像素，先在当前帧中寻找极线匹配，若能匹配上，则利用极线匹配的结果更新深度图的估计。

3.极线搜索原理大致和上一节介绍的相同，但实现上添加了一些细节：因为假设深度值服从高斯分布，我们就以均值为中心，左右各取$\pm 3\sigma$作为半径，然后在当前帧中寻找极线的投影。然后，遍历此极线上的像素（步长取$\sqrt{2}/2$的近似值0.7），寻找NCC最高的点作为匹配点。如果最高的NCC也低于阈值（这里取0.85），则认为匹配失败。

4.NCC的计算使用了去均值化后的做法，即对于图像块A,B，取：

$$\mathrm{NCC}_z(\boldsymbol{A},\boldsymbol{B}) = \frac{\sum_{i,j}\left(\boldsymbol{A}(i,j)-\bar{\boldsymbol{A}}(i,j)\right)\left(\boldsymbol{B}(i,j)-\bar{\boldsymbol{B}}(i,j)\right)}{\sqrt{\sum_{i,j}\left(\boldsymbol{A}(i,j)-\bar{\boldsymbol{A}}(i,j)\right)^2\sum_{i,j}\left(\boldsymbol{B}(i,j)-\bar{\boldsymbol{B}}(i,j)\right)^2}}. \tag{13.12}$$

5.三角化的计算方式与7.5节一致，不确定性的计算与高斯融合方法和上一节一致。

虽然程序有些长，相信读者根据上面的提示能读懂。下面我们来看它的实际运行效果。

实验结果

编译此程序后，以数据集目录作为参数运行之[4]：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
$ build/dense_mapping ~/dataset/test_data
read total 202 files.
*** loop 1 ***
*** loop 2 ***
......
```

程序输出的信息比较简洁，仅显示了迭代次数、当前图像和深度图。关于深度图，我们显示的是深度值乘以0.4后的结果——也就是纯白点（数值为1.0）的深度约2.5米，颜色越深表示深度值越小，也就是物体离我们越近。如果实际运行了程序，应该会发现深度估计是一个动态的过程——从一个不怎么确定的初始值逐渐收敛到稳定值的过程。我们的初始值使用了均值和方差均为3.0的分布。当然你也可以修改初始分布，看看对结果会产生怎样的影响。

从图13-6可以发现，当迭代次数超过一定值之后，深度图趋于稳定，不再对新的数据产生改变。观察稳定之后的深度图，我们发现大致可以看出地板和桌子的区别，而桌上的物体深度则接近于桌子。整个估计大部分是正确的，但也存在着大量错误估计。它们表现为深度图中与周围数据不一致的地方，为过大或过小的估计。此外，位于边缘处的地方，由于运动过程中看到的次数较少，所以亦没有得到正确的估计。综上所述，我们认为这个深度图的大部分是正确的，但没有达到预想的效果。我们将在下一节分析这些情况的出现原因，并讨论有哪些可以改进的地方。

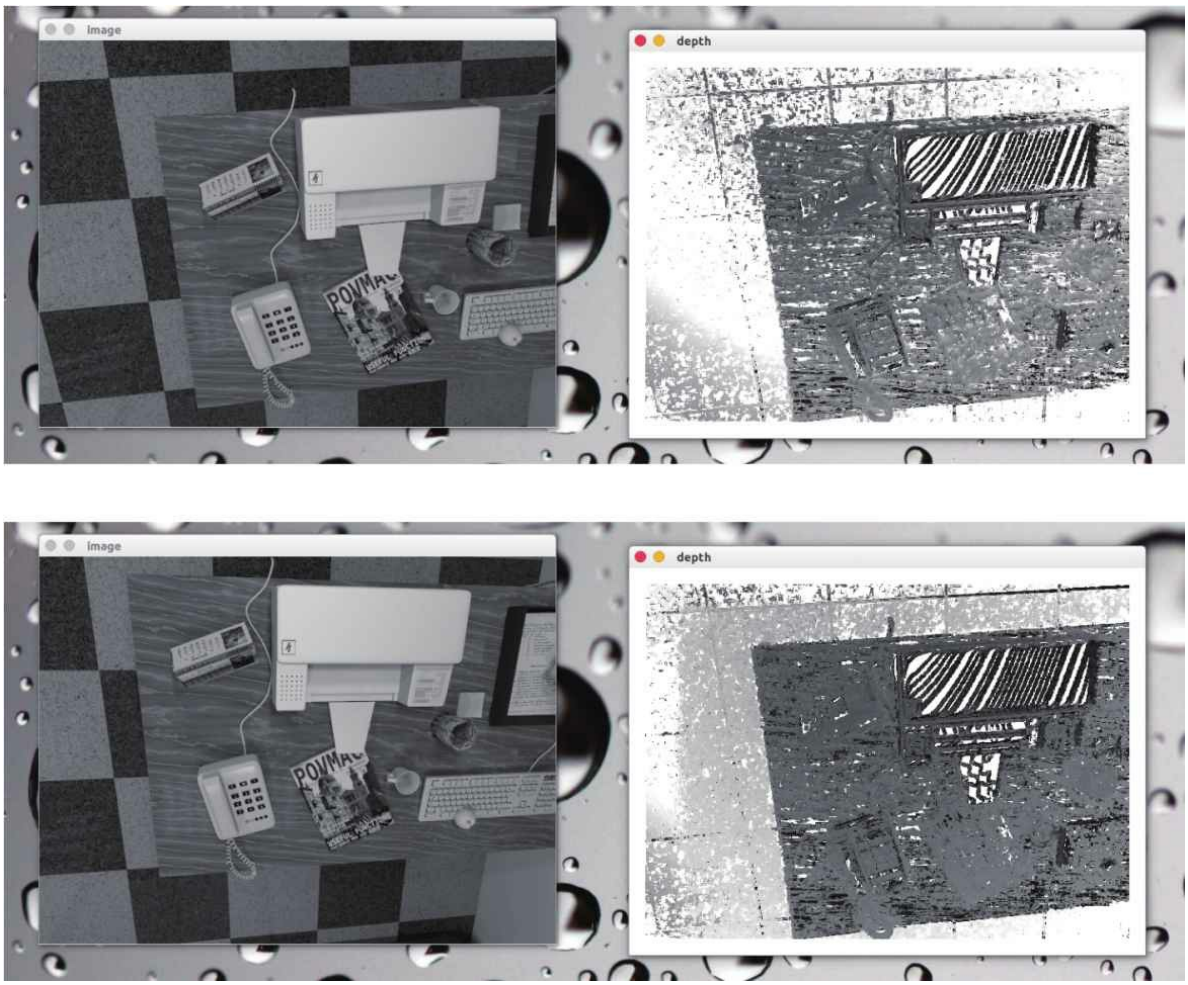


图13-6　演示程序运行时截图。两图分别是迭代10次和30次的结果。

书籍汇总下载 http://t.cn/RDIAj5D

13.4 实验分析与讨论

上一节我们演示了移动单目相机的稠密建图，估计了参考帧的每个像素深度。我们的代码是相对简单直接的，没有使用许多的技巧（trick），因此出现了实际工程中常见的情形——简单的往往并不是最有效的。

由于真实数据的复杂性，能够在实际环境下工作的程序，往往需要周密的考虑和大量的工程技巧，这使得每种实际可行的代码都极其复杂——它们很难向初学者解释清楚，所以我们只好使用不那么有效，但相对易读易写的实现方式。我们当然可以提出若干种对演示程序加以改进的意见，不过这里并不打算把已经改好的（非常复杂的）程序直接呈现给读者。

下面我们对上一节实验的结果进行初步分析。我们将从计算机视觉和滤波器两个角度来分析演示实验的结果。

13.4.1 像素梯度的问题

对深度图像进行观察，我们会发现一件明显的事实。块匹配的正确与否，依赖于图像块是否具有区分度。显然，如果图像块仅是一片黑或者一片白，缺少有效的信息，那么在NCC计算中我们就很可能错误地将它与周围的某块像素给匹配起来。请读者观察演示程序中的打印机表面。由于它是均匀的白色，非常容易引起误匹配，因此打印机表面的深度信息多半是不正确的——示例程序的空间表面出现了明显不该有的条纹状深度估计，而根据我们的直观想象，打印机表面肯定是光滑的。

这里牵涉到了一个问题，该问题在直接法中我们已经见过一次。在进行块匹配（和NCC的计算）时，我们必须假设小块不变，然后将该小块与其他小块进行对比。这时，有明显梯度的小块将具有良好的区分度，不易引起误匹配。对于梯度不明显的像素，由于在块匹配时没有区分性，所以我们将难以有效地估计其深度。反之，像素梯度比较明显的地方，我们得到的深度信息也相对准确，例如桌面上的杂志、电话等具

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

有明显纹理的物体。因此，演示程序反映了立体视觉中一个非常常见的问题：对物体纹理的依赖性。该问题在双目视觉中也极其常见，体现了立体视觉的重建质量十分依赖于环境纹理。

我们的演示程序刻意使用了纹理较好的环境，例如，像棋盘格一般的地板，带有木纹的桌面，等等，因此能得到一个看似不错的结果。然而在实际中，像墙面、光滑物体表面等亮度均匀的地方将经常出现，影响我们对它的深度估计。从某种角度来说，该问题是无法在现有的算法流程上加以改进并解决的——如果我们依然只关心某个像素周围的邻域（小块）的话。

进一步讨论像素梯度问题，我们还会发现像素梯度和极线之间的联系。文章[59]详细讨论过它们的关系，不过在我们的演示程序里也有直观的体现。

以图13-7为例，我们举两种比较极端的情况：像素梯度平行于极线方向，以及垂直于极线方向。先来看垂直的情况。在垂直的例子里，即使小块有明显梯度，但当我们沿着极线去做块匹配时，会发现匹配程度都是一样的，因此得不到有效的匹配。反之，在平行的例子里，我们能够精确地确定匹配度最高点出现在何处。而实际当中，梯度与极线的情况很可能介于二者之间：既不是完全垂直亦不是完全平行。这时，我们说，当像素梯度与极线夹角较大时，极线匹配的不确定性大；而当夹角较小时，匹配的不确定性变小。而在演示程序中，我们统一地把这些情况都当成一个像素的误差，实际是不够精细的。考虑到极线与像素梯度的关系后，应该使用更精确的不确定性模型。具体的调整和改进留作习题。

书籍汇总下载 http://t.cn/RDIAj5D

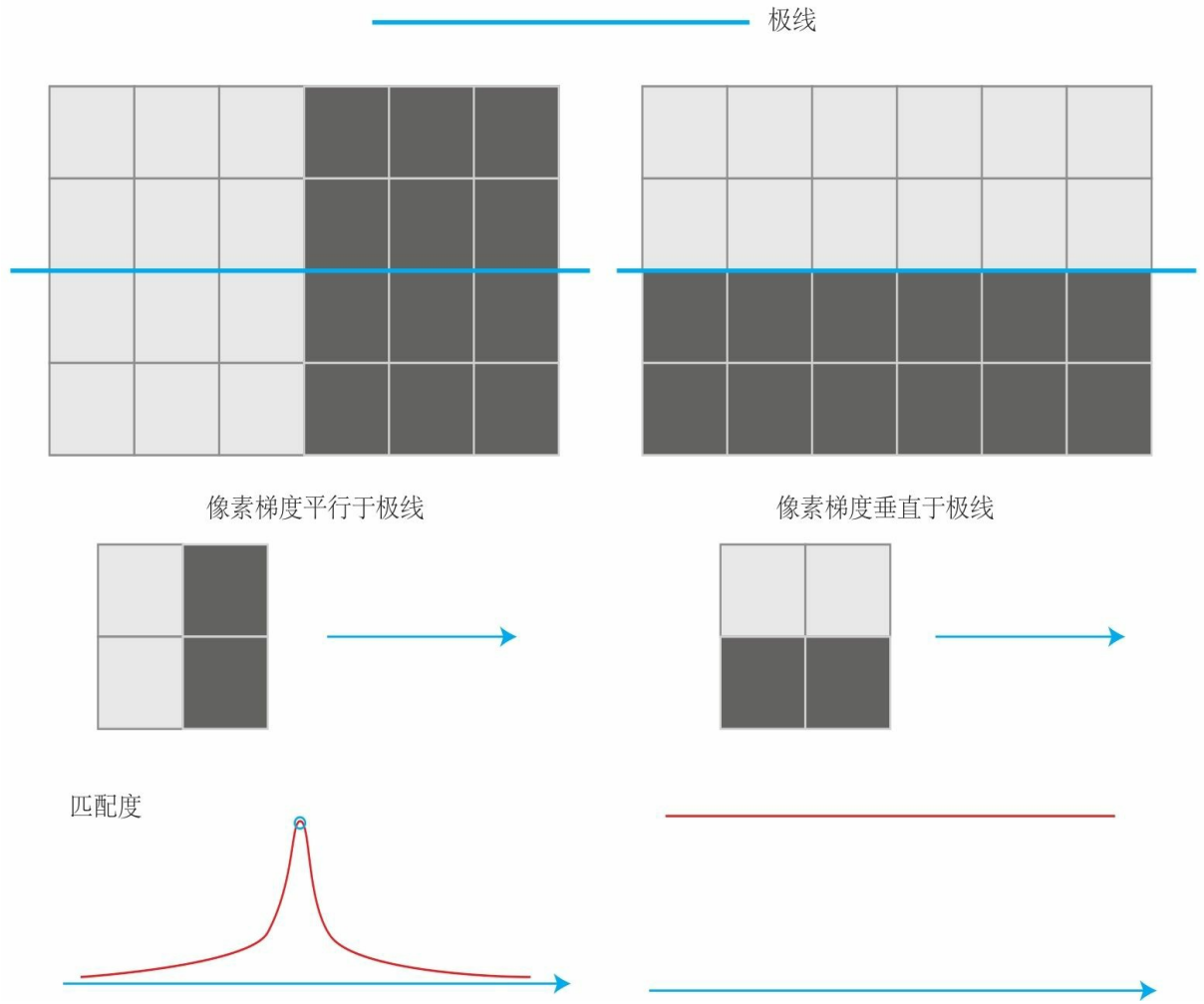


图13-7　像素梯度与极线之关系示意图。

13.4.2　逆深度

从另一个角度看，我们不妨问：把像素深度假设成高斯分布是否合适呢？这里关系到一个参数化的问题（Parameterization）。

在前面的内容中，我们经常用一个点的世界坐标x,y,z三个量来描述它，这是一种参数化形式。我们认为x,y,z三个量都是随机的，它们服从（三维的）高斯分布。然而，本讲使用了图像坐标u,v和深度值d来描述某个空间点（即稠密建图）。我们认为u,v不动，而d服从（一维的）高斯分布，这是另一种参数化形式。那么我们要问：这两种参数化形式有什么不同吗？我们是否也能假设u,v服从高斯分布，从而形成另一种参数化形式呢？

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

不同的参数化形式，实际都描述了同一个量，也就是某个三维空间点。考虑到当我们在相机看到某个点时，它的图像坐标u,v是比较确定的[5]，而深度值d则是非常不确定的。此时，若用世界坐标x,y,z描述这个点，那么根据相机当前的位姿，x,y,z三个量之间可能存在明显的相关性。反映在协方差矩阵中，表现为非对角元素不为零。而如果用u,v,d参数化一个点，那么它的u,v和d至少是近似独立的，甚至我们还能认为u,v也是独立的——从而它的协方差矩阵近似为对角阵，更为简洁。

逆深度（Inverse depth）是近年来SLAM研究中出现的一种广泛使用的参数化技巧[114,115]。在演示程序中，我们假设深度值满足高斯分布：$d \sim N(\mu,\sigma^2)$。然而这样做合不合理呢？深度真的近似于一个高斯分布吗？仔细想想，深度的正态分布确实存在一些问题：

1.我们实际想表达的是：这个场景深度大概是5～10米，可能有一些更远的点，但近处肯定不会小于相机焦距（或认为深度不会小于0）。这个分布并不是像高斯分布那样，形成一个对称的形状。它的尾部可能稍长，而负数区域则为零。

2.在一些室外应用中，可能存在距离非常远，乃至无穷远处的点。我们的初始值中难以涵盖这些点，并且用高斯分布描述它们会有一些数值计算上的困难。

于是，逆深度应运而生。人们在仿真中发现，假设深度的倒数，也就是逆深度，为高斯分布是比较有效的[115]。随后，在实际应用中，逆深度也具有更好的数值稳定性，从而逐渐成为一种通用的技巧，存在于现有SLAM方案中的标准做法中[56,57,73]。

把演示程序从正深度改成逆深度亦不复杂。只要在前面出现深度的推导中，将d改成逆深度d^{-1}即可。我们亦把这个改动留作习题，交给读者完成。

13.4.3 图像间的变换

在块匹配之前，做一次图像到图像间的变换亦是一种常见的预处理方式。这是因为，我们假设了图像小块在相机运动时保持不变，而这个假设在相机平移时（示例数据集基本都是这样的例子）能够保持成立，

书籍汇总下载 http://t.cn/RDIAj5D

但当相机发生明显的旋转时，就难以继续保持了。特别地，当相机绕光心旋转时，一块下黑上白的图像可能会变成一个上黑下白的图像块，导致相关性直接变成了负数（尽管仍然是同样一个块）。

为了防止这种情况的出现，我们通常需要在块匹配之前，把参考帧与当前帧之间的运动考虑进来。根据相机模型，参考帧上的一个像素P_R与真实的三维点世界坐标P_W有以下关系：

$$d_R \boldsymbol{P}_R = \boldsymbol{K}\left(\boldsymbol{R}_{\mathrm{RW}} \boldsymbol{P}_W + \boldsymbol{t}_{\mathrm{RW}}\right). \tag{13.13}$$

类似地，对于当前帧，亦有P_W在它上边的投影，记作P_C：

$$d_C \boldsymbol{P}_C = \boldsymbol{K}\left(\boldsymbol{R}_{\mathrm{CW}} \boldsymbol{P}_W + \boldsymbol{t}_{\mathrm{CW}}\right). \tag{13.14}$$

代入并消去P_W，即得两幅图像之间的像素关系：

$$d_C \boldsymbol{P}_C = d_R \boldsymbol{K} \boldsymbol{R}_{\mathrm{CW}} \boldsymbol{R}_{\mathrm{RW}}^{\mathrm{T}} \boldsymbol{K}^{-1} \boldsymbol{P}_R + \boldsymbol{K} \boldsymbol{t}_{\mathrm{CW}} - \boldsymbol{K} \boldsymbol{R}_{\mathrm{CW}} \boldsymbol{R}_{\mathrm{RW}}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{t}_{\mathrm{RW}}. \tag{13.15}$$

当知道d_R,P_R时，可以计算出P_C的投影位置。此时，再给P_R两个分量各一个增量du,dv，就可以求得P_C的增量du_c,dv_c。通过这种方式，算出在局部范围内参考帧和当前帧图像坐标变换的一个线性关系构成仿射变换：

$$\begin{bmatrix} \mathrm{d}\, u_c \\ \mathrm{d}\, v_c \end{bmatrix} = \begin{bmatrix} \frac{\mathrm{d}\, u_c}{\mathrm{d}\, u} & \frac{\mathrm{d}\, u_c}{\mathrm{d}\, v} \\ \frac{\mathrm{d}\, v_c}{\mathrm{d}\, u} & \frac{\mathrm{d}\, v_c}{\mathrm{d}\, v} \end{bmatrix} \begin{bmatrix} \mathrm{d}\, u \\ \mathrm{d}\, v \end{bmatrix} \tag{13.16}$$

根据仿射变换矩阵，我们可以把当前帧（或参考帧）的像素进行变换后再进行块匹配，以期获得对旋转更好的效果。

13.4.4 并行化：效率的问题

在实验当中我们也看到，稠密深度图的估计非常费时，这是因为我们要估计的点从原先的数百个特征点一下子变成了几十万个像素点，即

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

使现在主流的CPU也无法实时地计算那样庞大的数量。不过，该问题亦有另一个性质：这几十万个像素点的深度估计是彼此无关的！这使并行化有了用武之地。

在示例程序中，我们在一个二重循环里遍历了所有像素，并逐个对它们进行极线搜索。当我们使用CPU时，这个过程是串行进行的：必须是上一个像素计算完毕后，再计算下一个像素。然而实际上，下一个像素完全没有必要等待上一个像素的计算结束，因为它们之间并没有明显的联系，所以我们可以用多个线程，分别计算每个像素，然后将结果统一起来。理论上，如果我们有30万个线程，那么该问题的计算时间和计算一个像素是一样的。

GPU的并行计算架构非常适合这样的问题，因此，在单双和双目的稠密重建中，经常看到利用GPU进行并行加速的方式。当然，本书不准备涉及GPU编程，所以我们在这里仅指出利用GPU加速的可能性，具体实践留给读者作为验证。根据一些类似的工作，利用GPU的稠密深度估计是可以在主流GPU上实时化的。

13.4.5 其他的改进

事实上，我们还能提出许多对本例程进行改进的方案，例如：

1.现在各像素完全是独立计算的，可能存在这个像素深度很小，边上一个又很大的情况。我们可以假设深度图中相邻的深度变化不会太大，从而给深度估计加上了空间正则项。这种做法会使得到的深度图更加平滑。

2.我们没有显式地处理外点（Outlier）的情况。事实上，由于遮挡、光照、运动模糊等各种因素的影响，不可能对每个像素都能保持成功匹配。而演示程序的做法中，只要NCC大于一定值，就认为出现了成功的匹配，没有考虑到错误匹配的情况。

处理错误匹配亦有若干种方式。例如，文献[112]提出的均匀-高斯混合分布下的深度滤波器，显式地将内点与外点进行区别并进行概率建模，能够较好地处理外点数据。然而这种类型的滤波器理论较为复杂，本书不想过多涉及，读者可以阅读原始论文。

书籍汇总下载 http://t.cn/RDIAj5D

从上面的讨论可以看出，存在着许多可能的改进方案。如果我们细致地改进每一步的做法，最后是有希望得到一个良好的稠密建图的方案的。然而，正如我们所讨论的，有一些问题存在理论上的困难，例如对环境纹理的依赖，又如像素梯度与极线方向的关联（以及平行的情况）。这些问题很难通过调整代码实现来解决。所以，直到目前为止，尽管双目和移动单目能够建立稠密的地图，我们通常认为它们过于依赖于环境纹理和光照，不够可靠。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

13.5 RGB-D稠密建图

除了使用单目和双目进行稠密重建之外，在适用范围内，RGB-D相机是一种更好的选择。在上一讲中详细讨论的深度估计问题，在RGB-D相机中可以完全通过传感器中硬件测量得到，无须消耗大量的计算资源来估计。并且，RGB-D的结构光或飞时原理，保证了深度数据对纹理的无关性。即使面对纯色的物体，只要它能够反射光，我们就能测量到它的深度。这亦是RGB-D传感器的一大优势。

利用RGB-D进行稠密建图是相对容易的。不过，根据地图形式不同，也存在着若干种不同的主流建图方式。最直观、最简单的方法，就是根据估算的相机位姿，将RGB-D数据转化为点云（Point Cloud），然后进行拼接，最后得到一个由离散的点组成的点云地图（Point Cloud Map）。在此基础上，如果我们对外观有进一步的要求，希望估计物体的表面，可以使用三角网格（Mesh）、面片（Surfel）进行建图。另一方面，如果希望知道地图的障碍物信息并在地图上导航，亦可通过体素（Voxel）建立占据网格地图（Occupancy Map）。

我们似乎引入了很多新概念。请读者不要着急，我们将慢慢地逐一加以介绍。对于部分适合进行实验的，我们亦会像往常一样，提供若干个演示程序。由于RGB-D建图牵涉到的理论知识并不很多，所以下面几节就直接以实践部分来介绍了。GPU建图超出了本书的范围，我们就简单讲解其原理，不做演示。

13.5.1 实践：点云地图

首先，我们来讲解最简单的点云地图。所谓点云，就是由一组离散的点表示的地图。最基本的点包含x,y,z三维坐标，也可以带有r,g,b的彩色信息。由于RGB-D相机提供了彩色图和深度图，很容易根据相机内参来计算RGB-D点云。如果通过某种手段，得到了相机的位姿，那么只要直接把点云进行加和，就可以获得全局的点云。在本书的5.4节，曾给出了一个通过相机内外参拼接点云的例子。不过，那个例子主要是为了让读者理解相机的内外参，而在实际建图当中，我们还会对点云加一些

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

滤波处理，以获得更好的视觉效果。在本程序中，我们主要使用两种滤波器：外点去除滤波器，以及降采样滤波器。示例程序的代码如下（由于部分代码与之前的相同，我们主要看改变的部分）：

slambook/ch13/dense_RGBD/pointcloud_mapping.cpp（片段）

```
int main( int argc, char** argv )
{
    // 图像读取部分略
    // 定义点云使用的格式：这里用的是XYZRGB
    typedef pcl::PointXYZRGB PointT;
    typedef pcl::PointCloud<PointT> PointCloud;

    // 新建一个点云
    PointCloud::Ptr pointCloud( new PointCloud );
    for ( int i=0; i<5; i++ )
    {
        PointCloud::Ptr current( new PointCloud );
        cout<<"转换图像中：  "<<i+1<<endl;
        cv::Mat color = colorImgs[i];
        cv::Mat depth = depthImgs[i];
        Eigen::Isometry3d T = poses[i];
        for ( int v=0; v<color.rows; v++ )
        for ( int u=0; u<color.cols; u++ )
        {
            unsigned int d = depth.ptr<unsigned short> ( v )[u]; // 深度值
            if ( d==0 ) continue; // 为 0 表示没有测量到
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

```
            if ( d >= 7000 ) continue; // 深度太大时不稳定，去掉
            Eigen::Vector3d point;
            point[2] = double(d)/depthScale;
            point[0] = (u-cx)*point[2]/fx;
            point[1] = (v-cy)*point[2]/fy;
            Eigen::Vector3d pointWorld = T*point;

            PointT p ;
            p.x = pointWorld[0];
            p.y = pointWorld[1];
            p.z = pointWorld[2];
            p.b = color.data[ v*color.step+u*color.channels() ];
            p.g = color.data[ v*color.step+u*color.channels()+1 ];
            p.r = color.data[ v*color.step+u*color.channels()+2 ];
            current->points.push_back( p );
        }
        // depth filter and statistical removal
        PointCloud::Ptr tmp ( new PointCloud );
        pcl::StatisticalOutlierRemoval<PointT> statistical_filter;
        statistical_filter.setMeanK(50);
        statistical_filter.setStddevMulThresh(1.0);
        statistical_filter.setInputCloud(current);
        statistical_filter.filter( *tmp );
        (*pointCloud) += *tmp;
    }

    pointCloud->is_dense = false;
    cout<<"点云共有"<<pointCloud->size()<<"个点."<<endl;

    // voxel filter
    pcl::VoxelGrid<PointT> voxel_filter;
    voxel_filter.setLeafSize( 0.01, 0.01, 0.01 );       // resolution
    PointCloud::Ptr tmp ( new PointCloud );
    voxel_filter.setInputCloud( pointCloud );
    voxel_filter.filter( *tmp );
    tmp->swap( *pointCloud );

    cout<<"滤波之后，点云共有"<<pointCloud->size()<<"个点."<<endl;

    pcl::io::savePCDFileBinary("map.pcd", *pointCloud );
    return 0;
}
```

我们的思路没有太大变化，主要不同之处在于：

1.在生成每帧点云时，去掉深度值太大或无效的点。这主要是考虑到Kinect的有效量程，超过量程之后的深度值会有较大误差。

2.利用统计滤波器方法去除孤立点。该滤波器统计每个点与距离它最近的*N*个点的距离值的分布，去除距离均值过大的点。这样，我们保留了那些“粘在一起”的点，去掉了孤立的噪声点。

3.最后，利用体素滤波器（Voxel Filter）进行降采样。由于多个视角存在视野重叠，在重叠区域会存在大量的位置十分相近的点。这会白白占用许多内存空间。体素滤波保证了在某个一定大小的立方体（或称体素）内仅有一个点，相当于对三维空间进行了降采样，从而可节省很多存储空间。

图13-8显示了滤波前后的对比图。左侧是第5讲程序生成的点云地图，而右侧是经过滤波后的点云地图。观察白色框中部分，可以看到在滤波前存在着由噪声产生的许多孤立的点。经过统计外点去除之后，我们消去了这些噪声，使得整个地图变得更干净。另一方面，我们在体素滤波器中，把分辨率调至0.01，表示每立方厘米有一个点。这是一个比较高的分辨率，所以在截图中我们感觉不出地图的差异，然而从程序输出中可以看到点数明显减少了许多（从90万个点减到了44万个点，去除了一半左右）。



滤波前　　滤波后

图13-8　滤波前后的对比图（你可能需要放大才能看清，或者自己在电脑上尝试一下）。

点云地图为我们提供了比较基本的可视化地图，让我们能够大致了

解环境的样子。它以三维方式存储，使得我们能够快速地浏览场景的各个角落，乃至在场景中进行漫游。点云的一大优势是可以直接由RGB-D图像高效地生成，不需要额外处理。它的滤波操作也非常直观，且处理效率尚能接受。

不过，使用点云表达地图仍然是十分初级的，我们不妨按照之前提的对地图的需求，看看点云地图是否能满足。

1.定位需求：取决于前端视觉里程计的处理方式。如果是基于特征点的视觉里程计，由于点云中没有存储特征点信息，所以无法用于基于特征点的定位方法。如果前端是点云的ICP，那么可以考虑将局部点云对全局点云进行ICP以估计位姿。然而，这要求全局点云具有较好的精度。在我们这种处理点云的方式中，并没有对点云本身进行优化，所以是不够的。

2.导航与避障的需求：无法直接用于导航和避障。纯粹的点云无法表示“是否有障碍物”的信息，我们也无法在点云中做“任意空间点是否被占据”这样的查询，而这是导航和避障的基本需要。不过，可以在点云基础上进行加工，得到更适合导航与避障的地图形式。

3.可视化和交互：具有基本的可视化与交互能力。我们能够看到场景的外观，也能在场景里漫游。从可视化角度来说，由于点云只含有离散的点，而没有物体表面信息（例如法线），所以不太符合人们的可视化习惯。例如，点云地图的物体从正面看和背面看是一样的，而且还能透过物体看到它背后的东西：这些都不符合我们日常的经验，因为我们没有物体表面的信息。

综上所述，我们说点云地图是“基础的”或“初级的”，是指它更接近于传感器读取的原始数据。它具有一些基本的功能，但通常用于调试和基本的显示，不便直接用于应用程序。如果我们希望地图有更高级的功能，点云地图是一个不错的出发点。例如，针对导航功能，我们可以从点云出发，构建占据网格地图（Occupancy Grid），以供导航算法查询某点是否可以通过。再如，SfM中常用的泊松重建[116]方法，就能通过基本的点云重建物体网格地图，得到物体的表面信息。除泊松重建外，Surfel亦是一种表达物体表面的方式，以面元作为地图的基本单位，能够建立漂亮的可视化地图[117]。

图13-9显示了泊松重建和Surfel的一个样例，可以看到它们的视觉效果明显优于纯粹点云建图，而它们都可以通过点云进行构建。大部分由点云转换得到的地图形式都在PCL库中提供，感兴趣的读者可以进一步探索PCL库内容。而本书作为入门材料，就不详尽地介绍每一种地图形式了。

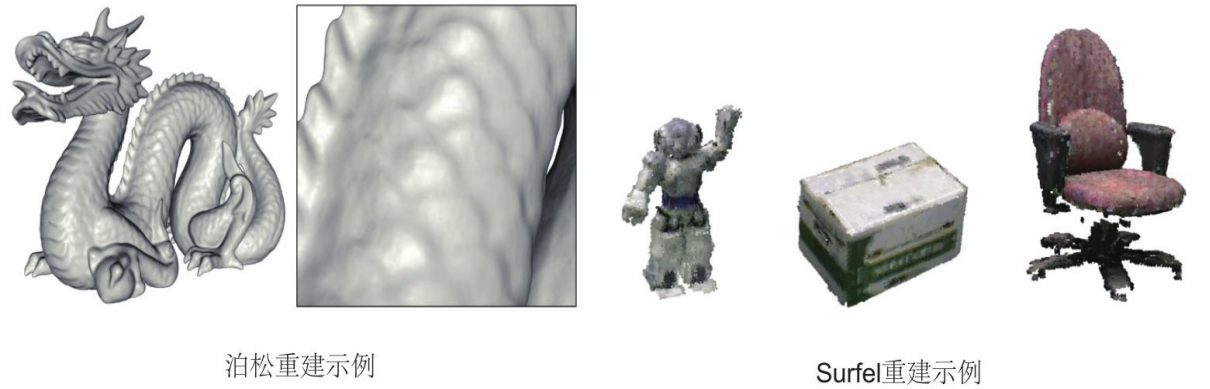


图13-9　泊松重建与Surfel的示意图

13.5.2　八叉树地图

下面介绍一种在导航中比较常用的，本身有较好的压缩性能的地图形式：八叉树地图。

在点云地图中，我们虽然有了三维结构，也进行了体素滤波以调整分辨率，但是点云有几个明显的缺陷：

•点云地图通常规模很大，所以pcd文件也会很大。一幅640像素×480像素的图像，会产生30万个空间点，需要大量的存储空间。即使经过一些滤波后，pcd文件也是很大的。而且讨厌之处在于，它的“大”并不是必需的。点云地图提供了很多不必要的细节。对于地毯上的褶皱、阴暗处的影子，我们并不特别关心这些东西。把它们放在地图里是在浪费空间。由于这些空间的占用，除非我们降低分辨率，否则在有限的内存中，无法建模较大的环境。然而降低分辨率会导致地图质量下降。有没有什么方式对地图进行压缩存储，舍弃一些重复的信息呢？

•点云地图无法处理运动物体。因为我们的做法里只有“添加点”，而没有“当点消失时把它移除”的做法。而在实际环境中，运动物体的普遍存在，使得点云地图变得不够实用。

而我们接来下要介绍的，就是一种灵活的、压缩的、又能随时更新的地图形式：八叉树（Octo-map）[118]。

我们知道，把三维空间建模为许多个小方块（或体素）是一种常见的做法。如果我们把一个小方块的每个面平均切成两片，那么这个小方块就会变成同样大小的八个小方块。这个步骤可以不断地重复，直到最后的方块大小达到建模的最高精度。在这个过程中，把“将一个小方块分成同样大小的八个”这件事，看成“从一个节点展开成八个子节点”，那么，整个从最大空间细分到最小空间的过程，就是一棵八叉树（Octo-tree）。

如图13-10所示，左侧显示了一个大立方体不断地均匀分成八块，直到变成最小的方块为止。于是，整个大方块可以看成是根节点，而最小的块可以看作是“叶子节点”。于是，在八叉树中，当我们由下一层节点往上走一层时，地图的体积就能扩大为原来的八倍。我们不妨做一点简单的计算：如果叶子节点的方块大小为1 cm^3，那么当我们限制八叉树为10层时，总共能建模的体积大约为$8^{10}cm^3=1,073m^3$，这足够建模一间屋子。由于体积与深度呈指数关系，所以当我们用更大的深度时，建模的体积会增长得非常快。

读者可能会疑惑，在点云的体素滤波器中，我们不是也限制了一个体素中只有一个点吗？为何我们说点云占空间，而八叉树比较节省空间呢？这是因为，在八叉树中，我们在节点中存储它是否被占据的信息。然而，不同之处在于，当某个方块的所有子节点都被占据或都不被占据时，就没必要展开这个节点。例如，一开始地图为空白时，我们就只需一个根节点，而不需要完整的树。当向地图中添加信息时，由于实际的物体经常连在一起，空白的地方也会常常连在一起，所以大多数八叉树节点都无须展开到叶子层面。所以说，八叉树比点云节省大量的存储空间。

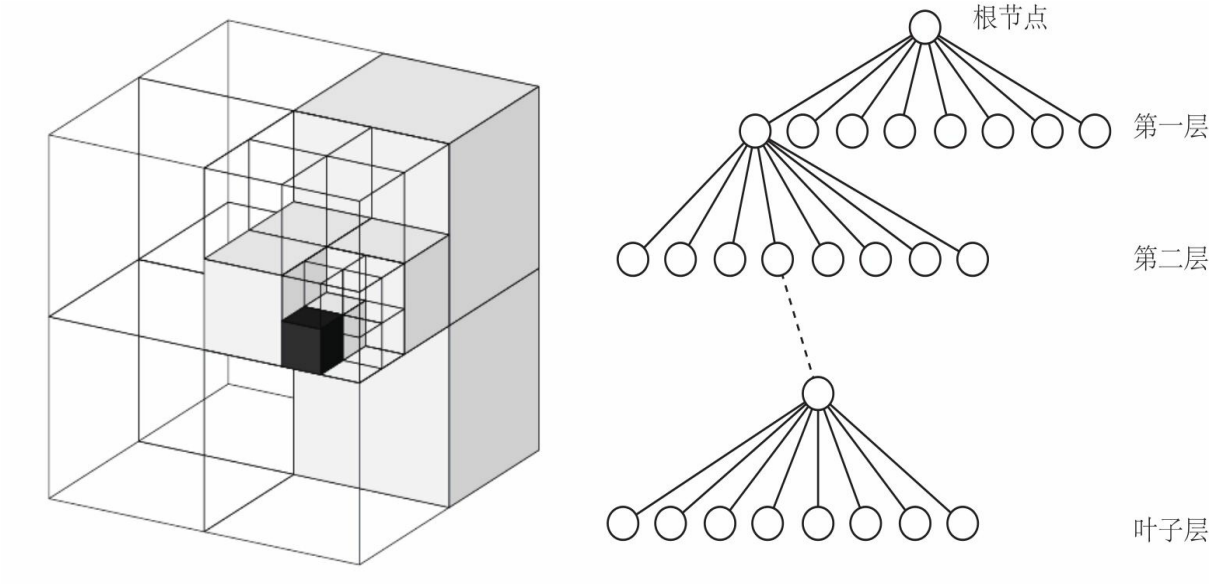


图13-10　八叉树示意图

前面说八叉树的节点存储了它是否被占据的信息。从点云层面来讲，我们自然可以用0表示空白，1表示被占据。这种0-1的表示可以用一个比特来存储，节省空间，不过显得有些过于简单了。由于噪声的影响，我们可能会看到某个点一会儿为0，一会儿为1；或者大部分时刻为0，小部分时刻为1；或者除了“是”“否”两种情况之外，还有一个“未知”的状态。能否更精细地描述这件事呢？我们会选择用概率形式表达某节点是否被占据的事情。比如，用一个浮点数$x\in[0,1]$来表达。这个x一开始取0.5。如果不断观测到它被占据，那么让这个值不断增加；反之，如果不断观测到它是空白，那就让它不断减小即可。

通过这种方式，我们动态地建模了地图中的障碍物信息。不过，现在的方式有一点小问题：如果让x不断增加或减小，它可能跑到[0,1]区间之外，带来处理上的不便。所以我们不是直接用概率来描述某节点被占据，而是用概率对数值（Log-odds）来描述。设$y\in\mathbb{R}$为概率对数值，x为0～1的概率，那么它们之间的变换由logit变换描述：

$$y=\operatorname{logit}(x)=\log\left(\frac{x}{1-x}\right). \tag{13.17}$$

其反变换为

$$x = \text{logit}^{-1}(y) = \frac{\exp(y)}{\exp(y)+1}. \tag{13.18}$$

可以看到，当y从-∞变到+∞时，x相应地从0变到了1。而当y取0时，x取到0.5。因此，我们不妨存储y来表达节点是否被占据。当不断观测到“占据”时，让y增加一个值；否则就让y减小一个值。当查询概率时，再用逆logit变换，将y转换至概率即可。用数学形式来说，设某节点为n，观测数据为z。那么从开始到t时刻某节点的概率对数值为$L(n|z_{1:t})$，t+1时刻为

$$L(n|z_{1:t+1}) = L(n|z_{1:t-1}) + L(n|z_t). \tag{13.19}$$

如果写成概率形式而不是概率对数形式，就会有一点复杂：

$$P(n|z_{1:T}) = \left[1 + \frac{1-P(n|z_T)}{P(n|z_T)} \frac{1-P(n|z_{1:T-1})}{P(n|z_{1:T-1})} \frac{P(n)}{1-P(n)}\right]^{-1}. \tag{13.20}$$

有了对数概率，我们就可以根据RGB-D数据更新整个八叉树地图了。假设我们在RGB-D图像中观测到某个像素带有深度d，这说明了一件事：我们在深度值对应的空间点上观察到了一个占据数据，并且，从相机光心出发到这个点的线段上，应该是没有物体的（否则会被遮挡）。利用这个信息，可以很好地对八叉树地图进行更新，并且能处理运动的结构。

13.5.3 实践：八叉树地图

下面通过程序演示一下octomap的建图过程。首先，请读者安装octomap库：https://github.com/OctoMap/octomap。Octomap库主要包含octomap地图与octovis（一个可视化程序），二者都是cmake工程。其主要依赖doxygen，可通过如下命令安装：

```
sudo apt-get install doxygen
```

我们直接来演示如何通过前面的5张图像生成八叉树地图，然后将

书籍汇总下载 http://t.cn/RDIAj5D

它画出来。

slambook/ch13/dense_RGBD/octomap_mapping.cpp（片段）

```
#include <iostream>
#include <fstream>
using namespace std;

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>

#include <octomap/octomap.h>    // for octomap

#include <Eigen/Geometry>
#include <boost/format.hpp>  // for formating strings

int main( int argc, char** argv )
{
    // 图像和位姿读取部分略
    cout<<"正在将图像转换为 Octomap ..."<<endl;

    // octomap tree
```

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

```
    octomap::OcTree tree( 0.05 ); // 参数为分辨率

    for ( int i=0; i<5; i++ )
    {
        cout<<"转换图像中: "<<i+1<<endl;
        cv::Mat color = colorImgs[i];
        cv::Mat depth = depthImgs[i];
        Eigen::Isometry3d T = poses[i];

        octomap::Pointcloud cloud;  // the point cloud in octomap

        for ( int v=0; v<color.rows; v++ )
            for ( int u=0; u<color.cols; u++ )
            {
                unsigned int d = depth.ptr<unsigned short> ( v )[u]; // 深度值
                if ( d==0 ) continue; // 为0表示没有测量到
                if ( d >= 7000 ) continue; // 深度太大时不稳定，去掉
                Eigen::Vector3d point;
                point[2] = double(d)/depthScale;
                point[0] = (u-cx)*point[2]/fx;
                point[1] = (v-cy)*point[2]/fy;
                Eigen::Vector3d pointWorld = T*point;
                // 将世界坐标系的点放入点云
                cloud.push_back( pointWorld[0], pointWorld[1], pointWorld[2] );
            }

        // 将点云存入八叉树地图，给定原点，这样可以计算投射线
        tree.insertPointCloud( cloud, octomap::point3d( T(0,3), T(1,3), T(2,3) ) );
    }

    // 更新中间节点的占据信息并写入磁盘
    tree.updateInnerOccupancy();
    cout<<"saving octomap ... "<<endl;
    tree.writeBinary( "octomap.bt" );
    return 0;
}
```

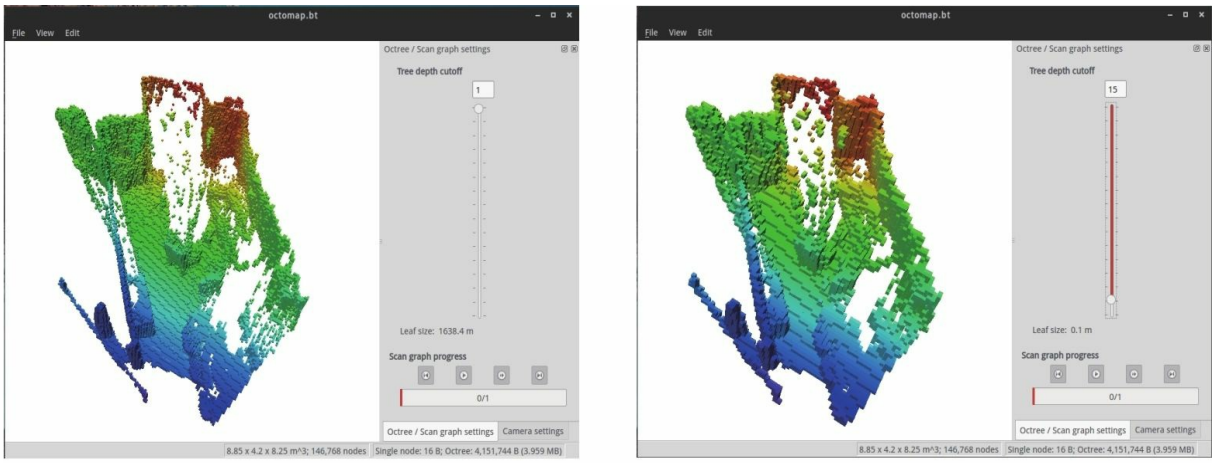
我们使用了octomap::OcTree来构建整张地图。实际上，octomap提

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

供了许多种八叉树：有带地图的，有带占据信息的，你也可以自己定义每个节点需要携带哪些变量。简单起见，我们使用了不带颜色信息的、最基本的八叉树地图。

Ocotmap内部提供了一个点云结构。它比PCL的点云稍微简单一些，只携带点的空间位置信息。我们根据RGB-D图像和相机位姿信息，先将点的坐标转至世界坐标，然后放入octomap的点云，最后交给八叉树地图。之后，octomap会根据之前介绍的投影信息，更新内部的占据概率，最后保存成压缩后的八叉树地图。我们把生成的地图存成octomap.bt文件。在之前编译octovis时，我们实际上安装了一个可视化程序，即octovis。现在，调用它打开地图文件，就能看到地图的实际样子了。

图13-11显示了我们构建的地图结果。由于我们没有在地图中加入颜色信息，所以一开始打开地图时将是灰色的，按“1”键可以根据高度信息进行染色。读者可以熟悉一下octovis的操作界面，包括地图的查看、旋转、缩放等。



八叉树地图（0.05米分辨率）　　八叉树地图（0.1米分辨率）

图13-11　八叉树地图在不同分辨率下的显示结果

在右侧有八叉树的深度限制条，这里可以调节地图的分辨率。由于我们构造时使用的默认深度是16层，所以这里显示16层即最高分辨率，也就是每个小块的边长为0.05米。当我们将深度减少一层时，八叉树的叶子节点往上提一层，每个小块的边长就增加一倍，变成0.1米。可以看到，我们能够很容易地调节地图分辨率以适应不同的场合。

书籍汇总下载 http://t.cn/RDIAj5D

Octomap还有一些可以探索的地方，例如，我们可以方便地查询任意点的占据概率，以此设计在地图中进行导航的方法[119]。读者亦可比较点云地图与八叉树地图的文件大小。上一节生成的点云地图的磁盘文件大约为6.9MB，而octomap只有56KB，连点云地图的百分之一都不到，可以有效地建模较大的场景。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

13.6 *TSDF地图和Fusion系列

在本讲的最后，我们介绍一个与SLAM非常相似但又有稍许不同的研究方向：实时三维重建。本节内容牵涉到GPU编程，并未提供参考例子，所以作为可选的阅读材料。

在前面的地图模型中，以定位为主体。地图的拼接是作为后续加工步骤放在SLAM框架中的。这种框架成为主流的原因是，定位算法可以满足实时性的需求，而地图的加工可以在关键帧处进行处理，无须实时响应。定位通常是轻量级的，特别是当使用稀疏特征或稀疏直接法时；相应地，地图的表达与存储则是重量级的。它们的规模和计算需求较大，不利于实时处理。特别是稠密地图，往往只能在关键帧层面进行计算。

但是，现有做法中，我们并没有对稠密地图进行优化。比如，当两幅图像都观察到同一把椅子时，我们只根据两幅图像的位姿把两处的点云进行叠加，生成了地图。由于位姿估计通常是带有误差的，这种直接拼接往往不够准确，比如同一把椅子的点云无法完美地叠加在一起。这时候，地图中会出现这把椅子的两个重影——这种现象有时候被形象地称为“鬼影”。

这种现象显然不是我们想要的，我们希望重建结果是光滑的、完整的，符合我们对地图的认识的。在这种思想下，出现了一种以“建图”为主体，而定位居于次要地位的做法，也就是本节要介绍的实时三维重建。由于三维重建把重建准确地图作为主要目标，所以基本都需要利用GPU进行加速，甚至需要非常高级的GPU或多个GPU进行并行加速，通常需要较重的计算设备。与之相反，SLAM是朝轻量级、小型化方向发展，有些方案甚至放弃了建图和回环检测部分，只保留了视觉里程计。而实时重建则正在朝大规模、大型动态场景的重建方向发展。

自从RGB-D传感器出现以来，利用RGB-D图像进行实时重建形成了一个重要的发展方向，陆续产生了Kinect Fusion[120]、Dynamic Fusion[121]、Elastic Fusion[122]、Fusion4D[123]、Volumn Deform[124]等成果。其中，Kinect Fusion完成了基本的模型重建，但仅限于小型场景；

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

后续的工作则是将它向大型的、运动的，甚至变形场景下拓展。我们把它们看成实时重建一个大类的工作，但由于种类繁多，不可能详细讨论每一种的工作原理。图13-12展示了一部分重建结果，可以看到这些建模结果非常精细，比单纯拼接点云要细腻很多。

我们就以经典的TSDF地图为代表进行介绍。TSDF是Truncated Signed Distance Function的缩写，不妨译作截断符号距离函数。虽然把“函数”称为“地图”似乎不太妥当，然而在没有更好的翻译之前，我们还是暂时称它为TSDF地图、TSDF重建等，只要不产生理解上的偏差即可。

与八叉树相似，TSDF地图也是一种网格式（或者说，方块式）的地图，如图13-13所示。先选定要建模的三维空间，比如$3\times3\times3m^3$那么大，按照一定分辨率将这个空间分成许多小块，存储每个小块内部的信息。不同的是，TSDF地图整个存储在显存而不是内存中。利用GPU的并行特性，我们可以并行地对每个体素进行计算和更新，而不像CPU遍历内存区域那样不得不串行。

每个TSDF体素内，存储了该小块与距其最近的物体表面的距离。如果小块在该物体表面的前方，它就有一个正的值；反之，如果该小块位于表面之后，那么就为负值。由于物体表面通常是很薄的一层，所以就把值太大的和太小的都取成1和-1，这样就得到了截断之后的距离，也就是所谓的TSDF。那么按照定义，TSDF为0的地方就是表面本身——或者，由于数值误差的存在，TSDF由负号变成正号的地方就是表面本身。在图13-13的下部，我们看到一个类似于人脸的表面出现在了TSDF改变符号的地方。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

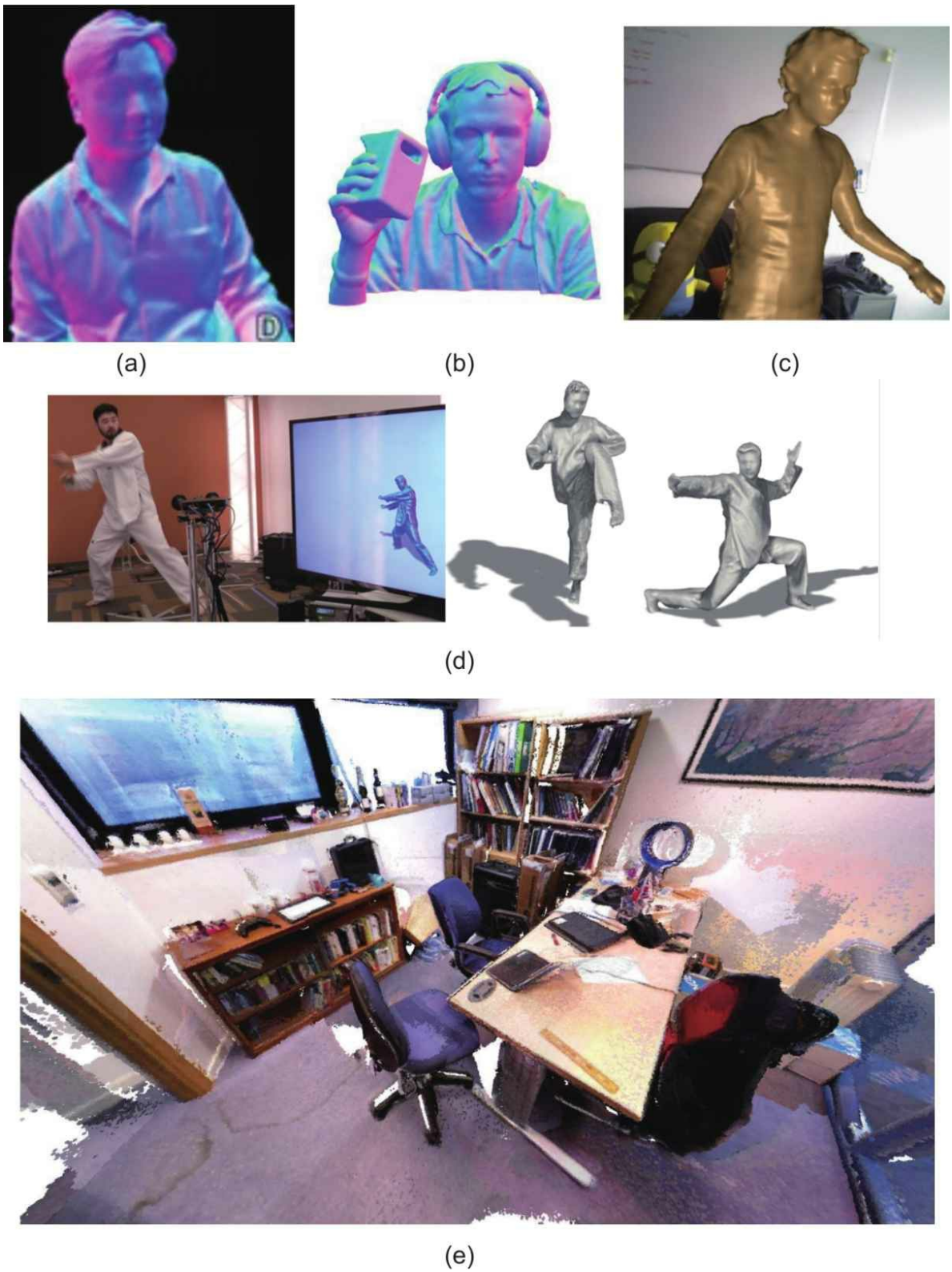
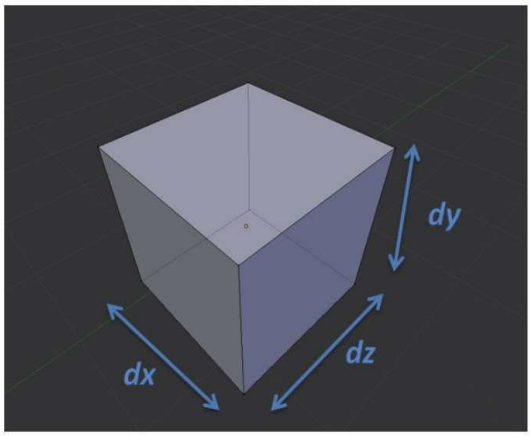
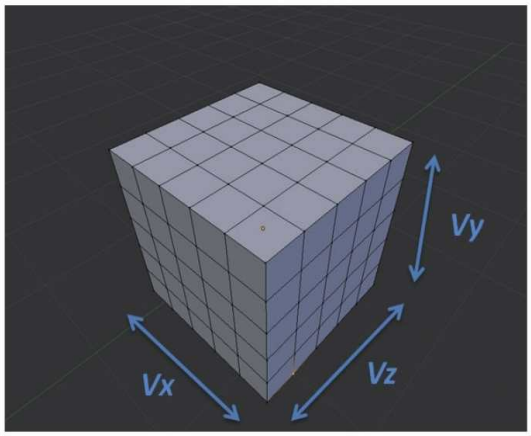


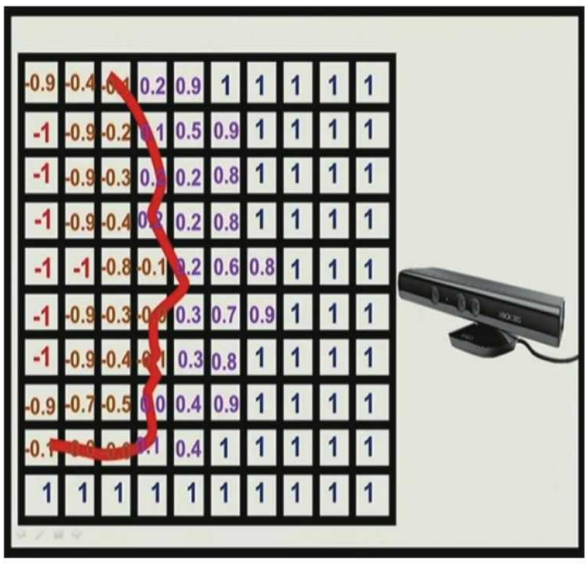
图13-12　各种实时三维重建的模型。（a）Kinect Fusion；（b）Dynamic Fusion；（c）Volumn Deform；（d）Fusion4D；（e）Elastic Fusion。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728





dx = dy = dz = 3 [meters]　　Vx = Vy = Vz = {32, 64, 128, 256, 512} [voxels]



相机观察到物体表面时形成的截断距离值

图13-13　TSDF示意图。

TSDF亦有“定位”与“建图”两个问题，与SLAM非常相似，不过具体的形式与本书前面几讲介绍的稍有不同。在这里，定位问题主要指如何把当前的RGB-D图像与GPU中的TSDF地图进行比较，估计相机位姿。而建图问题就是如何根据估计的相机位姿，对TSDF地图进行更新。传统做法中，我们还会对RGB-D图像进行一次双边贝叶斯滤波，以去除深度图中的噪声。

TSDF的定位类似于前面介绍的ICP，不过由于GPU的并行化，我们可以对整张深度图和TSDF地图进行ICP计算，而不必像传统视觉里程计

书籍汇总下载 http://t.cn/RDIAj5D

那样必须先计算特征点。同时，由于TSDF没有颜色信息，意味着我们可以只使用深度图，不使用彩色图，就完成位姿估计，这在一定程度上摆脱了视觉里程计算法对光照和纹理的依赖性，使得RGB-D重建更加稳健[6]。另一方面，建图部分亦是一种并行地对TSDF中的数值进行更新的过程，使得所估计的表面更加平滑可靠。由于我们并不过多介绍GPU相关的内容，所以具体的方法就不展开细说了，请感兴趣的读者参阅相关文献。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

13.7 小结

本讲介绍了一些常见的地图类型，尤其是稠密地图形式。我们看到根据单目或双目可以构建稠密地图，而RGB-D地图则往往更加容易、稳定一些。本讲的地图偏重于度量地图，而拓扑地图形式由于和SLAM研究差别比较大，所以没有详细地展开探讨。

习题

1.推导式(13.6)。

2.把本讲的稠密深度估计改成半稠密，你可以先把梯度明显的地方筛选出来。

3.*把本讲演示的单目稠密重建代码从正深度改成逆深度，并添加仿射变换。你的实验效果是否有改进？

4.你能论证如何在八叉树中进行导航或路径规划吗？

5.研究文献[120]，探讨TSDF地图是如何进行位姿估计和更新的。它和我们之前讲过的定位建图算法有何异同？

6.*研究均匀-高斯混合滤波器的原理与实现。

---

[1]正式点叫Fragile。

[2]反之，如果运动不知道，那么极线也无法确定。

[3]整体亮一些可能由环境光照变亮或相机曝光参数升高导致。

[4]请注意，稠密深度估计运行比较费时，如果你的计算机比较老，请耐心等候一段时间。

[5]u,v的不确定性取决于图像的分辨率。

[6]不过话说回来，对深度图就更加依赖了。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

第14讲　SLAM：现在与未来

主要目标

1.了解经典的SLAM实现方案。

2.通过实验，比较各种SLAM方案的异同。

3.探讨SLAM的未来发展方向。

前面介绍了一个SLAM系统中的各个模块的工作原理，这是研究者们多年工作的结晶。目前，除了这些理论框架之外，我们也积累了许多优秀的开源SLAM方案。不过，由于它们大部分的实现都比较复杂，不适合作为初学者的上手材料，所以我们放到了本书的最后加以介绍。相信读者通过阅读之前的内容，应该能明白其基本原理。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

14.1 当前的开源方案

本讲是全书的总结，我们将带着读者去看看现有的SLAM方案能做到怎样的程度。特别地，我们重点关注那些提供开源实现的方案。在SLAM研究领域，能见到开源方案是很不容易的。往往论文中介绍理论只占20%的内容，其他80%都写在代码中，是论文里没有提到的。正是这些研究者们的无私奉献，推动了整个SLAM行业的快速前进，使后续研究者有了更高的起点。在我们开始做SLAM之前，应该对相似的方案有深入的了解，然后再进行自己的研究，这样才会更有意义。

本讲的前半部分将带领读者参观一下当前的视觉SLAM方案，评述其历史地位和优缺点。表14-1列举了一些常见的开源SLAM方案，读者可以选择感兴趣的方案进行研究和实验。限于篇幅，我们只选了一部分有代表性的方案，这肯定是不全面的。在后半部分，我们将探讨未来可能的一些发展方向，并给出当前的一些研究成果。

表14-1常用开源SLAM方案

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

方案名称	传感器形式	地址
MonoSLAM	单目	https://github.com/hanmekim/SceneLib2
PTAM	单目	http://www.robots.ox.ac.uk/~gk/PTAM/
ORB-SLAM	单目为主	http://webdiis.unizar.es/~raulmur/orbslam/
LSD-SLAM	单目为主	http://vision.in.tum.de/research/vslam/lsdslam
SVO	单目	https://github.com/uzh-rpg/rpg_svo
DTAM	RGB-D	https://github.com/anuranbaka/OpenDTAM
DVO	RGB-D	https://github.com/tum-vision/dvo_slam
DSO	单目	https://github.com/JakobEngel/dso
RTAB-MAP	双目/RGB-D	https://github.com/introlab/rtabmap
RGBD-SLAM-V2	RGB-D	https://github.com/felixendres/rgbdslam_v2
Elastic Fusion	RGB-D	https://github.com/mp3guy/ElasticFusion
Hector SLAM	激光	http://wiki.ros.org/hector_slam
GMapping	激光	http://wiki.ros.org/gmapping
OKVIS	多目 +IMU	https://github.com/ethz-asl/okvis
ROVIO	单目 +IMU	https://github.com/ethz-asl/rovio

14.1.1 MonoSLAM

说到视觉SLAM，很多研究者第一个想到的是A.J.Davison的单目SLAM工作[2,125]。Davison教授是视觉SLAM研究领域的先驱，他在2007年提出的MonoSLAM是第一个实时的单目视觉SLAM系统[2]，被认为是许多工作的发源地[1]。MonoSLAM以扩展卡尔曼滤波为后端，追踪前端非常稀疏的特征点。由于EKF在早期SLAM中占据着明显主导地位，所以MonoSLAM亦是建立在EKF的基础之上，以相机的当前状态和所有路标点为状态量，更新其均值和协方差。

图14-1所示是MonoSLAM在运行时的情形。可以看到，单目相机在一幅图像当中追踪了非常稀疏的特征点（且用到了主动追踪技术）。在EKF中，每个特征点的位置服从高斯分布，所以我们能够以一个椭球的形式表达它的均值和不确定性。在该图的右半部分，我们可以找到一些在空间中分布着的小球。它们在某个方向上显得越长，说明在该方向的位置就越不确定。我们可以想象，如果一个特征点收敛，我们应该能看

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

到它从一个很长的椭球（相机Z方向上非常不确定）最后变成一个小点的样子。

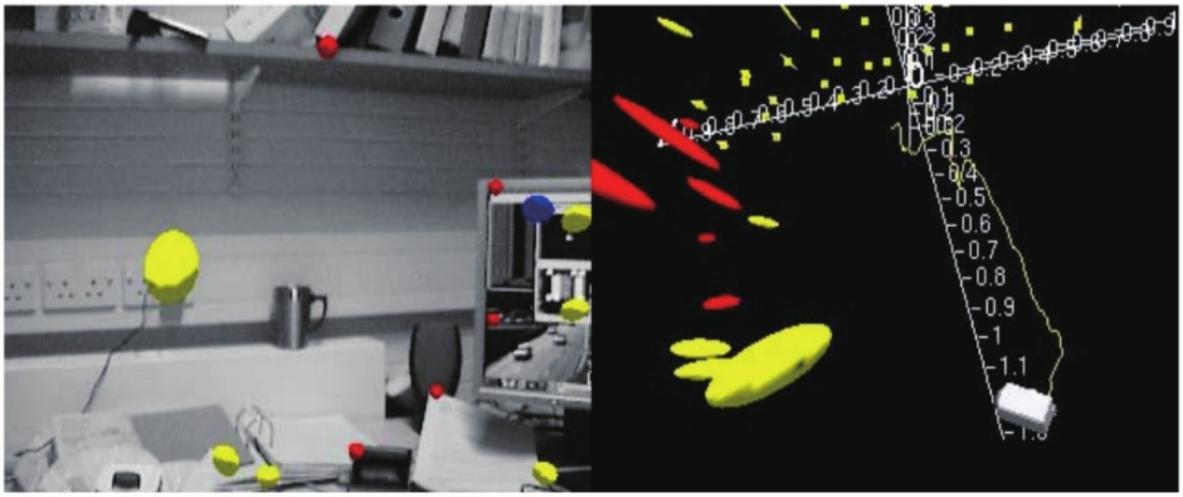


图14-1　MonoSLAM的运行时截图。左侧：追踪特征点在图像中的表示；右侧：特征点在三维空间中的表示。

这种做法在今天看来固然存在许多弊端，但在当时已经是里程碑式的工作了，因为在此之前的视觉SLAM系统基本不能在线运行，只能靠机器人携带相机采集数据，再离线地进行定位与建图。计算机性能的进步，以及用稀疏的方式处理图像，加在一起才使得一个SLAM系统能够在线地运行。从现代的角度来看，MonoSLAM存在诸如应用场景很窄，路标数量有限，稀疏特征点非常容易丢失的情况，对它的开发也已经停止，取而代之的是更先进的理论和编程工具。不过这并不妨碍我们对前人工作的理解和尊敬。

14.1.2　PTAM

2007年，Klein等人提出了PTAM（Parallel Tracking and Mapping）[81]，这也是视觉SLAM发展过程中的重要事件。PTAM的重要意义在于以下两点：

1.PTAM提出并实现了跟踪与建图过程的并行化。我们现在已然清楚，跟踪部分需要实时响应图像数据，而对地图的优化则没必要实时地计算。后端优化可以在后台慢慢进行，然后在必要的时候进行线程同步即可。这是视觉SLAM中首次区分出前后端的概念，引领了后来许多视

书籍汇总下载 http://t.cn/RDIAj5D

觉SLAM系统的设计（我们现在看到的SLAM多半都分前后端）。

2.PTAM是第一个使用非线性优化，而不是使用传统的滤波器作为后端的方案。它引入了关键帧机制：我们不必精细地处理每一幅图像，而是把几个关键图像串起来，然后优化其轨迹和地图。早期的SLAM大多数使用EKF滤波器或其变种，以及粒子滤波器等；在PTAM之后，视觉SLAM研究逐渐转向了以非线性优化为主导的后端。由于之前人们未认识到后端优化的稀疏性，所以觉得优化后端无法实时处理那样大规模的数据，而PTAM则是一个显著的反例。

PTAM同时是一个增强现实软件，演示了酷炫的AR效果（如图14-2所示）。根据PTAM估计的相机位姿，我们可以在一个虚拟的平面上放置虚拟物体，看起来就像在真实的场景中一样。

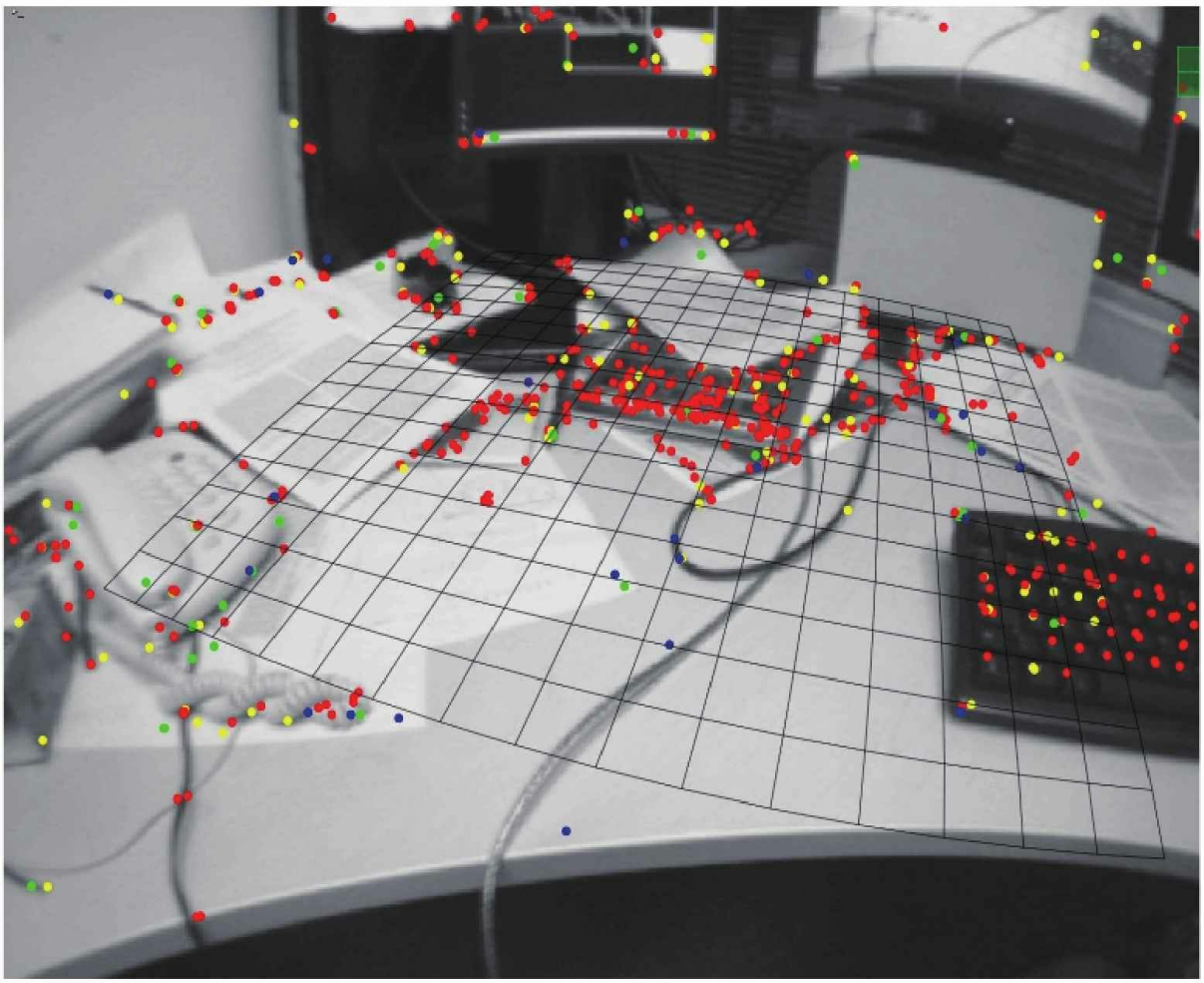


图14-2　PTAM的演示截图。它既可以提供实时的定位和建图，也可以在虚拟平面上叠加虚拟物体。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

不过，从现代的眼光看来，PTAM也算是早期的结合AR的SLAM工作之一。与许多早期工作相似，存在着明显的缺陷：场景小，跟踪容易丢失，等等。这些又在后续的方案中得以修正。

14.1.3 ORB-SLAM

介绍了历史上的几种方案之后，我们来看现代的一些SLAM系统。ORB-SLAM是PTAM的继承者中非常有名的一位[73]（见图14-3）。它提出于2015年，是现代SLAM系统中做得非常完善、非常易用的系统之一（如果不是最完善易用的话）。ORB-SLAM代表着主流的特征点SLAM的一个高峰。相比于之前的工作，ORB-SLAM具有以下几条明显的优势：

1.支持单目、双目、RGB-D三种模式。这使得无论我们拿到了哪种常见的传感器，都可以先放到ORB-SLAM上测试一下，它具有良好的泛用性。

2.整个系统围绕ORB特征进行计算，包括视觉里程计与回环检测的ORB字典。它体现出ORB特征是现阶段计算平台的一种优秀的效率与精度之间的折中方式。ORB不像SIFT或SURF那样费时，在CPU上面即可实时计算；相比Harris角点等简单角点特征，又具有良好的旋转和缩放不变性。并且，ORB提供描述子，使我们在大范围运动时能够进行回环检测和重定位。

3.ORB的回环检测是它的亮点。优秀的回环检测算法保证了ORB-SLAM有效地防止累积误差，并且在丢失之后还能迅速找回，这一点许多现有的SLAM系统都不够完善。为此，ORB-SLAM在运行之前必须加载一个很大的ORB字典文件[2]。

4.ORB-SLAM创新式地使用了三个线程完成SLAM：实时跟踪特征点的Tracking线程，局部Bundle Adjustment的优化线程（Co-visibility Graph，俗称小图），以及全局Pose Graph的回环检测与优化线程（Essential Graph俗称大图）。其中，Tracking线程负责对每幅新来的图像提取ORB特征点，并与最近的关键帧进行比较，计算特征点的位置并粗略估计相机位姿。小图线程求解一个Bundle Adjustment问题，它包括局部空间内的特征点与相机位姿。这个线程负责求解更精细的相机位姿

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

与特征点空间位置。不过，仅有前两个线程，只完成了一个比较好的视觉里程计。第三个线程，也就是大图线程，对全局的地图与关键帧进行回环检测，消除累积误差。由于全局地图中的地图点太多，所以这个线程的优化不包括地图点，而只有相机位姿组成的位姿图。

继PTAM的双线程结构之后，ORB-SLAM的三线程结构取得了非常好的跟踪和建图效果，能够保证轨迹与地图的全局一致性。这种三线程结构也将被后续的研究者认同和采用。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

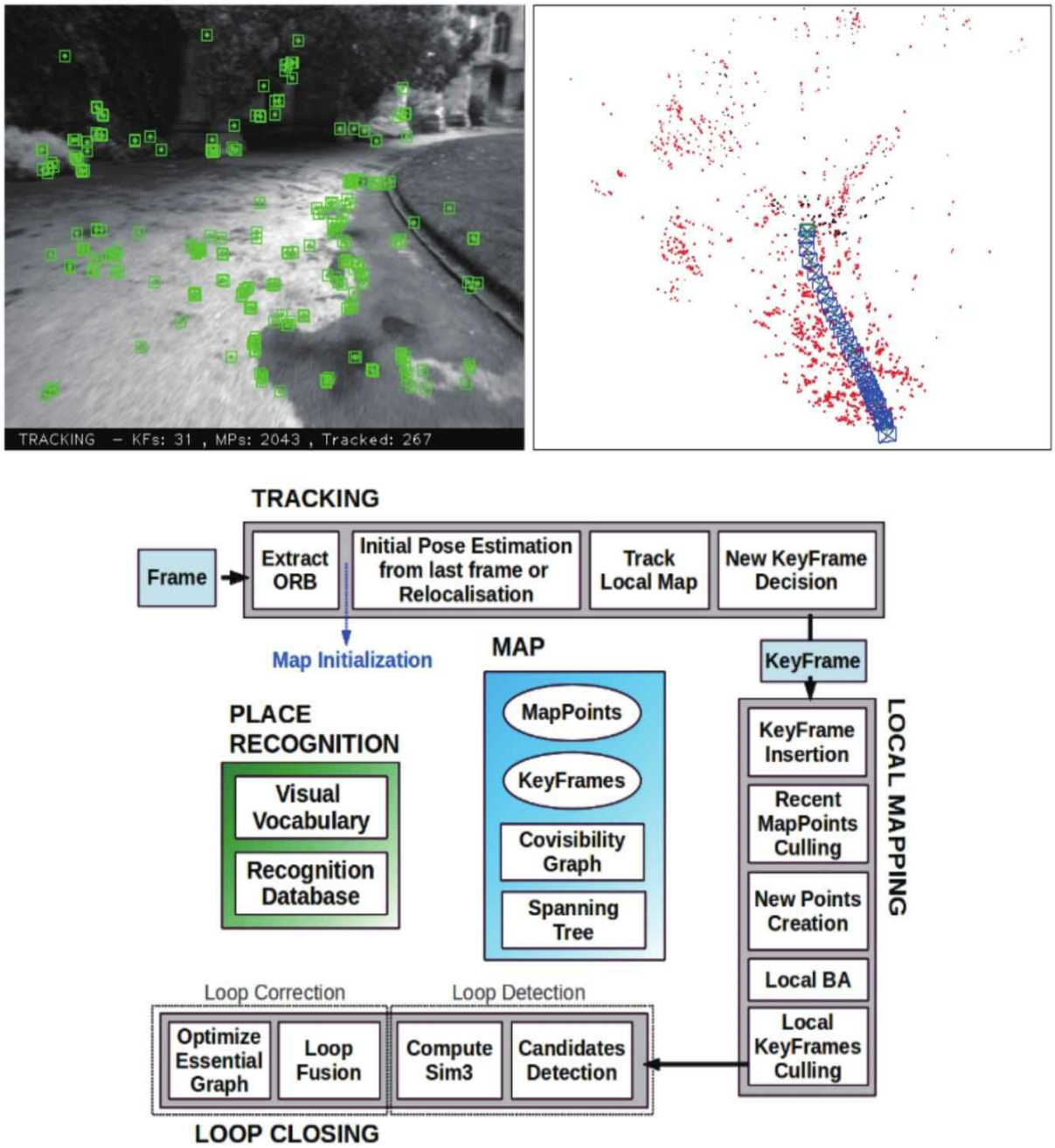


图14-3　ORB-SLAM运行截图。左侧为图像与追踪到的特征点，右侧为相机轨迹与建模的特征点地图。下方为其标志性的三线程结构。

5.ORB-SLAM围绕特征点进行了不少的优化。例如，在OpenCV的特征提取基础上保证了特征点的均匀分布，在优化位姿时使用了一种循环优化4遍以得到更多正确匹配的方法，比PTAM更为宽松的关键帧选取策略，等等。这些细小的改进使得ORB-SLAM具有远超其他方案的稳健性：即使对于较差的场景，较差的标定内参，ORB-SLAM都能够顺利

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

地工作。

上述这些优势使得ORB-SLAM在特征点SLAM中达到顶峰，许多研究工作都以ORB-SLAM作为标准，或者在它的基础上进行后续的开发。它的代码以清晰易读著称，有着完善的注释，可供后来的研究者进一步理解。

当然，ORB-SLAM也存在一些不足之处。首先，由于整个SLAM系统都采用特征点进行计算，我们必须对每幅图像都计算一遍ORB特征，这是非常耗时的。ORB-SLAM的三线程结构也给CPU带来了较重的负担，使得它只有在当前PC架构的CPU上才能实时运算，移植到嵌入式设备上则有一定困难。其次，ORB-SLAM的建图为稀疏特征点，目前还没有开放存储和读取地图后重新定位的功能（虽然从实现上来讲并不困难）。根据我们在建图部分的分析，稀疏特征点地图只能满足我们对定位的需求，而无法提供导航、避障、交互等诸多功能。然而，如果我们仅用ORB-SLAM处理定位问题，似乎又显得有些过于重量级了。相比之下，另外一些方案提供了更为轻量级的定位，使我们能够在低端的处理器上运行SLAM，或者让CPU有余力处理其他的事务。

14.1.4 LSD-SLAM

LSD-SLAM（Large Scale Direct monocular SLAM）是J.Engle等人于2014年提出的SLAM工作[59,57]。类比于ORB-SLAM之于特征点，LSD-SLAM则标志着单目直接法在SLAM中的成功应用。LSD-SLAM的核心贡献是将直接法应用到了半稠密的单目SLAM中。它不仅不需要计算特征点，还能构建半稠密的地图——这里半稠密的意思主要是指估计梯度明显的像素位置。它的主要优点如下：

1.LSD-SLAM的直接法是针对像素进行的。作者有创见地提出了像素梯度与直接法的关系，以及像素梯度与极线方向在稠密重建中的角度关系。这些在本书的第8讲和第13讲均有讨论。不过，LSD-SLAM是在单目图像进行半稠密的跟踪，实现原理要比本书的例程更加复杂。

2.LSD-SLAM在CPU上实现了半稠密场景的重建，这在之前的方案中是很少见到的。基于特征点的方法只能是稀疏的，而进行稠密重建的方案大多要使用RGBD传感器，或者使用GPU构建稠密地图[126]。TUM

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

计算机视觉组在多年对直接法研究的基础上，实现了这种CPU上的实时半稠密SLAM。

3.之前也说过，LSD-SLAM的半稠密追踪使用了一些精妙的手段来保证追踪的实时性与稳定性。例如，LSD-SLAM既不是利用单个像素，也不是利用图像块，而是在极线上等距离取5个点，度量其SSD；在深度估计时，LSD-SLAM首先用随机数初始化深度，在估计完后又把深度均值归一化，以调整尺度；在度量深度不确定性时，不仅考虑三角化的几何关系，而且考虑了极线与深度的夹角，归纳成一个光度不确定性项；关键帧之间的约束使用了相似变换群及与之对应的李代数$\zeta \in sim(3)$显式地表达出尺度，在后端优化中可以将不同尺度的场景考虑进来，减小了尺度飘移现象。

图14-4显示了LSD的运行情况。我们可以观察一下这种微妙的半稠密地图是怎样一种介于稀疏地图与稠密地图之间的形式。半稠密地图建模了灰度图中有明显梯度的部分，显示在地图中，很大一部分都是物体的边缘或表面上带纹理的部分。LSD-SLAM对它们进行跟踪并建立关键帧，最后优化得到这样的地图。看起来比稀疏的地图具有更多的信息，但又不像稠密地图那样拥有完整的表面（稠密地图一般认为无法仅用CPU实现实时性）。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

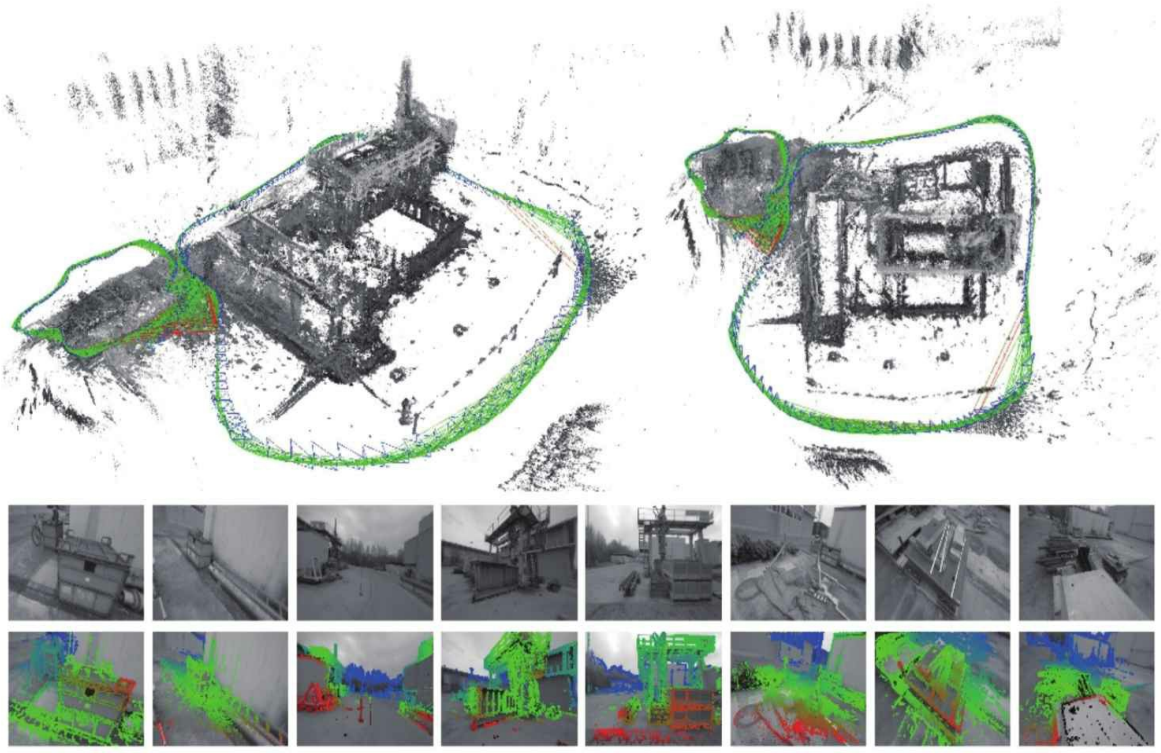


图14-4　LSD-SLAM运行图片。上半部分为估计的轨迹与地图，下半部分为图像中被建模的部分，即具有较好的像素梯度的部分。

由于LSD-SLAM使用了直接法进行跟踪，所以它既有直接法的优点（对特征缺失区域不敏感），也继承了直接法的缺点。例如，LSD-SLAM对相机内参和曝光非常敏感，并且在相机快速运动时容易丢失。另外，在回环检测部分，由于目前并没有基于直接法实现的回环检测方式，因此LSD-SLAM必须依赖于特征点方法进行回环检测，尚未完全摆脱特征点的计算。

14.1.5　SVO

SVO是Semi-direct Visual Odoemtry的缩写[56]。它是由Forster等人于2014年提出的一种基于稀疏直接法的视觉里程计。按作者的称呼应该叫“半直接”法，然而按照本书的理念框架，称为“稀疏直接法”可能更好一些。半直接在原文中的意思是指特征点与直接法的混合使用：SVO跟踪了一些关键点（角点，没有描述子），然后像直接法那样，根据这些关键点周围的信息估计相机运动及其位置（如图14-4所示）。在实现

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

中，SVO使用了关键点周围的4×4的小块进行块匹配，估计相机自身的运动。

相比于其他方案，SVO的最大优势是速度极快。由于使用稀疏的直接法，它既不必费力去计算描述子，也不必处理像稠密和半稠密那么多的信息，因此，即使在低端计算平台上也能达到实时性，而在PC平台上则可以达到每秒100多帧的速度。在后续的SVO 2.0中，速度更达到了惊人的每秒400帧。这使得SVO非常适用于计算平台受限的场合，例如无人机、手持AR/VR设备的定位。无人机也是作者开发SVO的目标应用平台。

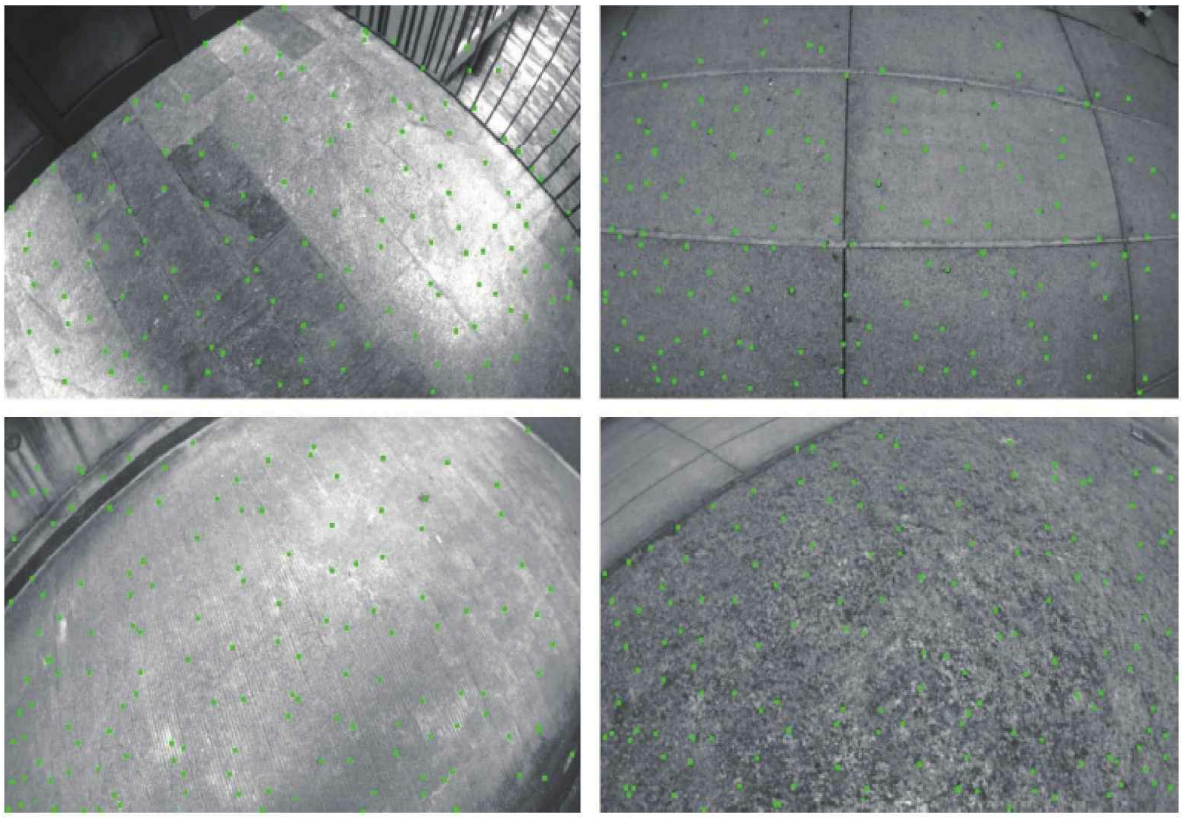


图14-5　SVO跟踪关键点的图片。

SVO的另一创新之处是提出了深度滤波器的概念，并推导了基于均匀-高斯混合分布的深度滤波器。这在本书的第13讲有提及，但由于原理较为复杂，我们没有详细解释。SVO将这种滤波器用于关键点的位置估计，并使用了逆深度作为参数化形式，使之能够更好地计算特征点位置。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

开源版的SVO代码清晰易读，十分适合读者作为第一个SLAM实例进行分析。不过，开源版SVO也存在一些问题：

1.由于目标应用平台为无人机的俯视相机，其视野内的物体主要是地面，而且相机的运动主要为水平和上下的移动，SVO的许多细节是围绕这个应用设计的，这使得它在平视相机中表现不佳。例如，SVO在单目初始化时，使用了分解*H*矩阵而不是传统的*F*或*E*矩阵的方式，这需要假设特征点位于平面上。该假设对俯视相机是成立的，但对平视相机通常是不成立的，可能导致初始化失败。再如，SVO在关键帧选择时，使用了平移量作为确定新的关键帧的策略，而没有考虑旋转量。这同样在无人机俯视配置下是有效的，但在平视相机中则会容易丢失。所以，如果读者想要在平视相机中使用SVO，必须自己加以修改。

2.SVO为了速度和轻量化，舍弃了后端优化和回环检测部分，也基本没有建图功能。这意味着SVO的位姿估计必然存在累积误差，而且丢失后不太容易进行重定位（因为没有描述子用来回环检测）。所以，我们称它为一个VO，而不是称它为完整的SLAM。

14.1.6 RTAB-MAP

介绍了几款单目SLAM方案后，我们再来看一些RGB-D传感器上的SLAM方案。相比于单目和双目，RGB-D SLAM的原理要简单很多（尽管实现上不一定），而且能够在CPU上实时建立稠密的地图。

RTAB-MAP（Real Time Appearance-Based Mapping）[107]是RGB-D SLAM中比较经典的一个方案。它实现了RGB-D SLAM中所有应该有的东西：基于特征的视觉里程计、基于词袋的回环检测、后端的位姿图优化，以及点云和三角网格地图。因此，RTABMAP给出了一套完整的（但有些庞大的）RGB-D SLAM方案。目前我们已经可以直接从ROS中获得其二进制程序，此外，在Google Project Tango上也可以获取其App使用（如图14-6所示）。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

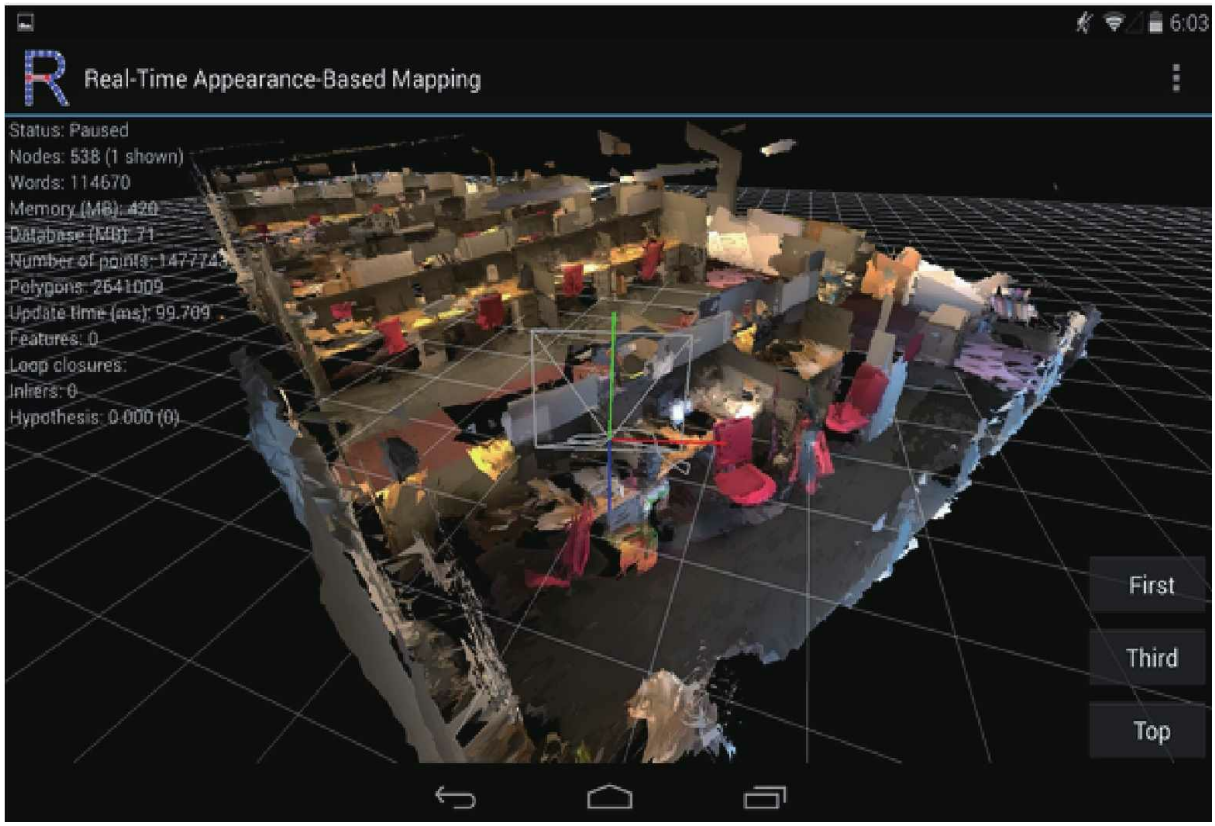


图14-6　RTAB-MAP在Google Project Tango上的运行样例。

RTAB-MAP支持一些常见的RGB-D和双目传感器，像Kinect、Xtion等，且提供实时的定位和建图功能。不过由于集成度较高，使得其他开发者在它的基础上进行二次开发变得困难，所以RTAB-MAP更适合作为SLAM应用而非研究使用。

14.1.7　其他

除了这些开源方案之外，读者还能在openslam.org之类的网站上找到许多其他的研究，例如，DVO-SLAM[127]、RGBD-SLAM-V2[88]、DSO[58]，以及一些Kinect Fusion相关的工作，等等。随着时代发展，更新颖、更优秀的开源SLAM作品亦将出现在人们的视野中，限于篇幅这里就不逐一介绍了。

书籍汇总下载 http://t.cn/RDIAj5D

14.2 未来的SLAM话题

看过了现有的方案，我们再来讨论一些未来的发展方向[3]。大体上讲，SLAM将来的发展趋势有两大类：一是朝轻量级、小型化方向发展，让SLAM能够在嵌入式或手机等小型设备上良好运行，然后考虑以它为底层功能的应用。毕竟，大部分场合中，我们的真正目的都是实现机器人、AR/VR设备的功能，比如说运动、导航、教学、娱乐，而SLAM是为上层应用提供自身的一个位姿估计。在这些应用中，我们不希望SLAM占用所有计算资源，所以对SLAM的小型化和轻量化有非常强烈的需求。另一方面则是利用高性能计算设备，实现精密的三维重建、场景理解等功能。在这些应用中，我们的目的是完美地重建场景，而对于计算资源和设备的便携性则没有多大限制。由于可以利用GPU，这个方向和深度学习亦有结合点。

14.2.1 视觉+惯性导航SLAM

首先，我们要谈一个有很强应用背景的方向：视觉-惯性导航融合SLAM方案。实际的机器人也好，硬件设备也好，通常都不会只携带一种传感器，往往是多种传感器的融合。学术界的研究人员喜爱“大而且干净的问题”（Big Clean Problem），比如说仅用单个摄像头实现视觉SLAM。但产业界的朋友们则更注重让算法更加实用，不得不面对一些复杂而琐碎的场景。在这种应用背景下，用视觉与惯性导航融合进行SLAM成为了一个关注热点。

惯性传感器（IMU）能够测量传感器本体的角速度和加速度，被认为与相机传感器具有明显的互补性，而且十分有潜力在融合之后得到更完善的SLAM系统[128]。为什么这么说呢？

1.IMU虽然可以测得角速度和加速度，但这些量都存在明显的漂移（Drift），使得积分两次得到的位姿数据非常不可靠。好比说，我们将IMU放在桌上不动，用它的读数积分得到的位姿也会漂出十万八千里。但是，对于短时间内的快速运动，IMU能够提供一些较好的估计。这正

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

是相机的弱点。

当运动过快时，（卷帘快门的）相机会出现运动模糊，或者两帧之间重叠区域太少以至于无法进行特征匹配，所以纯视觉SLAM非常害怕快速的运动。而有了IMU，即使在相机数据无效的那段时间内，我们也能保持一个较好的位姿估计，这是纯视觉SLAM无法做到的。

2.相比于IMU，相机数据基本不会有漂移。如果相机放在原地固定不动，那么（在静态场景下）视觉SLAM的位姿估计也是固定不动的。所以，相机数据可以有效地估计并修正IMU读数中的漂移，使得在慢速运动后的位姿估计依然有效。

3.当图像发生变化时，本质上我们没法知道是相机自身发生了运动，还是外界条件发生了变化，所以纯视觉SLAM难以处理动态的障碍物。而IMU能够感受到自己的运动信息，从某种程度上减轻动态物体的影响。

总而言之，我们看到IMU为快速运动提供了较好的解决方式，而相机又能在慢速运动下解决IMU的漂移问题——在这个意义下，它们二者是互补的。

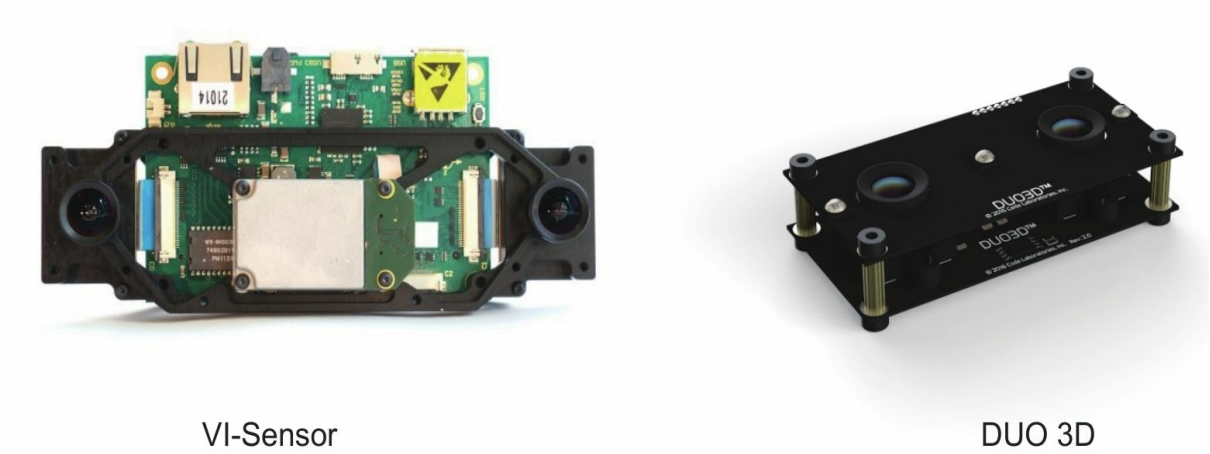


图14-7　越来越多的相机开始集成IMU设备。

当然，虽然说得很好听，不管是理论还是实践，VIO（VisualInertialOdometry）都是相当复杂的。其复杂性主要来源于IMU测量加速度和角速度这两个量的事实，所以不得不引入运动学计算。目前VIO的框架已经定型为两大类：松耦合（Loosely　Coupled）和

书籍汇总下载 http://t.cn/RDIAj5D

紧耦合（Tightly Coupled）[129]。松耦合是指IMU和相机分别进行自身的运动估计，然后对其位姿估计结果进行融合。紧耦合是指把IMU的状态与相机的状态合并在一起，共同构建运动方程和观测方程，然后进行状态估计——这和我们之前介绍的理论非常相似。我们可以预见，紧耦合理论也必将分为基于滤波和基于优化两个方向。在滤波方面，传统的EKF[130]以及改进的MSCKF（Multi-State Constraint KF）[131]都取得了一定的成果，研究者对EKF也进行了深入的讨论（例如能观性[132]）；优化方面亦有相应的方案[74,133]。值得一提的是，尽管在纯视觉SLAM中优化方法已经占了主流，但在VIO中，由于IMU的数据频率非常高，对状态进行优化需要的计算量就更大，因此目前仍处于滤波与优化并存的阶段[134,60]。由于过于复杂，限于篇幅，这里就只能大概地介绍一下这个方向了。

VIO为将来SLAM的小型化与低成本化提供了一个非常有效的方向。而且结合稀疏直接法，我们有望在低端硬件上取得良好的SLAM或VO效果，是非常有前景的。

14.2.2 语义SLAM

SLAM的另一个大方向就是和深度学习技术结合。到目前为止，SLAM的方案都处于特征点或者像素的层级。关于这些特征点或像素到底来自于什么东西，我们一无所知。这使得计算机视觉中的SLAM与我们人类的做法不怎么相似，至少我们自己从来看不到特征点，也不会去根据特征点判断自身的运动方向。我们看到的是一个个物体，通过左右眼判断它们的远近，然后基于它们在图像当中的运动推测相机的移动。

很久之前，研究者就试图将物体信息结合到SLAM中。例如文献[135,136,137,138]中就曾把物体识别与视觉SLAM结合起来，构建带物体标签的地图。另一方面，把标签信息引入到BA或优化端的目标函数和约束中，我们可以结合特征点的位置与标签信息进行优化[139]。这些工作都可以称为语义SLAM。综合来说，SLAM和语义的结合点主要有两个方面[9]：

1.语义帮助SLAM。传统的物体识别、分割算法往往只考虑一幅

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

图，而在SLAM中我们拥有一台移动的相机。如果我们把运动过程中的图片都带上物体标签，就能得到一个带有标签的地图。另外，物体信息亦可为回环检测、BA优化带来更多的条件。

2.SLAM帮助语义。物体识别和分割都需要大量的训练数据。要让分类器识别各个角度的物体，需要从不同视角采集该物体的数据，然后进行人工标定，非常辛苦。而SLAM中，由于我们可以估计相机的运动，可以自动地计算物体在图像中的位置，节省人工标定的成本。如果有自动生成的带高质量标注的样本数据，能够很大程度上加速分类器的训练过程。



图14-8　语义SLAM的一些结果，左图和右图分别来自文献[138,140]。

在深度学习广泛应用之前，我们只能利用支持向量机、条件随机场等传统工具对物体或场景进行分割和识别，或者直接将观测数据与数据库中的样本进行比较[108,140]，尝试构建语义地图[138,141,142,143]。由于这些工具本身在分类正确率上存在限制，所以效果也往往不尽如人意。随着深度学习的发展，我们开始使用网络，越来越准确地对图像进行识别、检测和分割[144,145,146,147,148,149]。这为构建准确的语义地图打下了更好的基础[150]。我们正看到，逐渐开始有学者将神经网络方法引入到SLAM中的物体识别和分割，甚至SLAM本身的位姿估计与回环检测中[151,152,153]。虽然这些方法目前还没有成为主流，但将SLAM与深度学习结合来处理图像，亦是一个很有前景的研究方向。

14.2.3　SLAM的未来

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

除此之外，基于线/面特征的SLAM[154,155,156]、动态场景下的SLAM[157,158,159]、多机器人的SLAM[160,67,161]，等等，都是研究者感兴趣并发力的地方。按照[9]的观点，视觉SLAM经过了三个大时代：提出问题、寻找算法、完善算法。而我们目前正处于第三个时代，面对着如何在已有的框架中进一步改善，使视觉SLAM系统能够在各种干扰的条件下稳定运行。这一步需要许多研究者的不懈努力。

当然，没有人能够预测未来，我们也说不准会不会突然有一天，整个框架都被新的技术推倒重写。不过即使是那样，今天我们的付出仍将是有意义的。没有今天的研究，也就不会有将来的发展。最后，希望读者能在读完本书之后，对现有的整个SLAM系统有了充分的认识。我们也期待你能够为SLAM研究做出贡献！

习题

1.选择本讲提到的任意一个开源SLAM系统，在你的机器上编译运行它，直观体验其过程。

2.你应该已经能够看懂绝大多数SLAM相关论文了。拿起纸和笔，开始你的研究吧！

---

[1]这是他博士期间工作的延续。他现在也在致力于将SLAM小型化、低功率化。

[2]目前开源版ORB-SLAM使用了文本格式的字典，改成二进制格式字典之后可以加速不少。

[3]这里有一部分是笔者个人的理解，不一定完全正确。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

附录A　高斯分布的

这里总结一下常见的高斯分布的性质，它在本书的很多地方都会用到。

A.1高斯分布

如果一个随机变量*x*服从高斯分布$N(\mu,\sigma)$，那么它的概率密度函数为

$$p(x)=\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right). \tag{A.1}$$

其高维形式为

$$p(x)=\frac{1}{\sqrt{(2\pi)^N\det(\boldsymbol{\Sigma})}}\exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right). \tag{A.2}$$

A.2高斯分布的运算

A.2.1线性运算

设两个独立的高斯分布：

$$\boldsymbol{x}\sim N(\boldsymbol{\mu}_x,\boldsymbol{\Sigma}_{xx}),\quad \boldsymbol{y}\sim N(\boldsymbol{\mu}_y,\boldsymbol{\Sigma}_{yy}),$$

那么，它们的和仍是高斯分布：

$$\boldsymbol{x}+\boldsymbol{y}\sim N(\boldsymbol{\mu}_x+\boldsymbol{\mu}_y,\boldsymbol{\Sigma}_{xx}+\boldsymbol{\Sigma}_{yy}). \tag{A.3}$$

如果以常数*a*乘以*x*，那么*ax*满足：

$$a\boldsymbol{x}\sim N(a\boldsymbol{\mu}_x,a^2\boldsymbol{\Sigma}_{xx}). \tag{A.4}$$

如果取*y*=*Ax*，那么*y*满足：

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

$$\boldsymbol{y} \sim N(\boldsymbol{G}\boldsymbol{\mu}_x, \boldsymbol{G}\boldsymbol{\Sigma}_{xx}\boldsymbol{G}^{\mathrm{T}}). \tag{A.5}$$

A.2.2乘积

设两个高斯分布的乘积满足$p(xy)=N(\mu,\Sigma)$，那么：

$$\begin{aligned}\boldsymbol{\Sigma}^{-1} &= \boldsymbol{\Sigma}_{xx}^{-1} + \boldsymbol{\Sigma}_{yy}^{-1} \\ \boldsymbol{\Sigma}\boldsymbol{\mu} &= \boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\mu}_x + \boldsymbol{\Sigma}_{yy}^{-1}\boldsymbol{\mu}_y.\end{aligned} \tag{A.6}$$

该公式可以推广到任意多个高斯分布之乘积。

A.2.3复合运算

同样考虑x和y，若其不独立，则其复合分布为

$$p(\boldsymbol{x},\boldsymbol{y}) = N\left(\begin{bmatrix}\boldsymbol{\mu}_x \\ \boldsymbol{\mu}_y\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_{xx} & \boldsymbol{\Sigma}_{xy} \\ \boldsymbol{\Sigma}_{yx} & \boldsymbol{\Sigma}_{yy}\end{bmatrix}\right). \tag{A.7}$$

由条件分布展开式$p(x,y)=p(x|y)p(y)$可以推出，条件概率$p(x|y)$满足：

$$p\left(\boldsymbol{x}|\boldsymbol{y}\right) = N\left(\boldsymbol{\mu}_x + \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}\left(\boldsymbol{y}-\boldsymbol{\mu}_y\right), \boldsymbol{\Sigma}_{xx} - \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}\boldsymbol{\Sigma}_{yx}\right). \tag{A.8}$$

A.3复合的例子

下面举一个和卡尔曼滤波器相关的例子。考虑随机变量$x\sim N(\mu_x,\Sigma_{xx})$，另一变量$y$满足：

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} + \boldsymbol{w} \tag{A.9}$$

其中A,b为线性变量的系数矩阵和偏移量，w为噪声项，为零均值的高斯分布：$w\sim N(0,R)$。

我们来看y的分布。根据前面的介绍，可以推出：

$$p\left(\boldsymbol{y}\right) = N\left(\boldsymbol{A}\boldsymbol{\mu}_x + \boldsymbol{b}, \boldsymbol{R} + \boldsymbol{A}\boldsymbol{\Sigma}_{xx}\boldsymbol{A}^{\mathrm{T}} + \boldsymbol{R}\right). \tag{A.10}$$

这为卡尔曼滤波器的预测部分提供了理论基础。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

附录B　ROS入门

ROS是机器人研究领域一个广为探讨的主题。为了避免使本书阅读门槛太高，我们没有在正文和例程中提到它。然而近年来，ROS正逐步在各大高校的学生中间得到推广，渐渐为人们所熟知和接受，所以这里也介绍一下ROS，希望对读者能有所帮助。

B.1ROS是什么

ROS（Robot Operating System）是Willow Garage公司于2007年发布的一个开源机器人操作系统，它为软件开发人员开发机器人应用程序提供了许多优秀的工具和库。同时，还有优秀的开发者不断地为它贡献代码。本质上，ROS并不是一个真正意义上的操作系统，而更像是基于操作系统之上的一个软件包。它提供了众多在实际机器人中可能遇到的算法：导航、通信、路径规划，等等。

ROS的版本代号是按照字母顺序来编排的，并随着Ubuntu系统发布更新。通常一个ROS版本会支持两到三个Ubuntu系统版本。ROS从Box Turtle开始，截止到本书写作时（2016年），已经更新到了Kinetic Kame（见图B-1）。同时，ROS也已经彻底重构，推出了实时性更强的2.0版本。

ROS支持很多操作系统，支持最完善的是Ubuntu及其衍生版本（Kubuntu、Linux　Mint、Ubuntu　GNOME等），对其他Linux发布版本、Windows等的支持也有，不过没有那么完善。因此，推荐读者使用Ubuntu操作系统来进行开发和研究。

ROS支持目前被广泛使用的面向对象的编程语言C++，以及脚本语言Python。你可以选择自己喜欢的语言进行开发。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D



图B-1　ROS各版本命名方式

B.2ROS的特点

ROS的设计初衷，就是使机器人开发能够像计算机开发一样，屏蔽底层硬件及其接口的不一致性，最终使得软件可以复用。

而软件复用也正是软件工程优美性最集中的体现之一，ROS能够以统一消息格式来使得大家只需要关注算法层面的设计，而底层硬件的根本目的是接收各种各样的消息，如图像、数据等。各个硬件厂商将接收到的数据都统一到ROS所规定的统一消息格式下，即可让用户方便地使用各种开源的机器人相关算法。

在第14讲中提到的常见的开源SLAM方案中，ORB-SLAM、ORB-SLAM2、LSDSLAM、SVO、DVO、RTAB-MAP、RGBD-SLAM-V2、Hector SLAM、Gmapping、ROVIO等均有ROS版本的开源代码，你可以很方便地在ROS中运行、调试和修改它们。

在调试SLAM程序时，数据的来源通常有3种：传感器、数据集，以及bag文件。若手头没有相应的传感器，通常就需要利用虚拟的数据来跑SLAM程序。其中，最方便的方式当属利用ROS下的bag文件发布topic，然后SLAM程序就可以监视topic发出的数据，就像使用真实的传感器采集数据一样。后面我们会简单介绍一下如何利用bag文件来模拟真实的传感器数据。

B.3如何快速上手ROS

ROS有完善的维基系统。因而，按照官网的介绍在机器上安装对应

电子书寻找看手相qq37391319 钉钉或微信 pythontesting　　书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

版本的ROS：http://wiki.ros.org/ROS/Installation；然后，阅读ROS自带的教学程序即可。你会学习到ROS的基本概念、主题的发布和订阅，以及用Python和C++控制它们。如果你觉得麻烦，也可以使用针对ROS定制的Ubuntu：http://www.aicrobo.com/ubuntu_for_ros.html。

除了基本知识之外，你还可以学到一些ROS的常用工具，例如：

1.rqt。rqt是ROS下的一个软件框架，它以插件的方式提供了各种各样方便好用的GUI（用户图形界面）。rqt的功能非常强大，可以实时地查看ROS中流动的消息。

2.rosbag。rosbag是ROS提供的一个非常好用的录制及播放topic数据的工具。当你想实际跑一下SLAM程序，但囿于手头没有实际的传感器时，可以考虑使用公开提供的bag文件来进行图像或者数据的模拟，这种方式与使用一个真实的传感器在感觉上并无不同。rosbag的使用方式请参考ROS的维基页面。此外，许多公开数据集也会提供bag格式的数据文件。

3.rviz。rviz是ROS提供的可视化模块，你可以通过它实时地查看ROS中的图像、点云、地图、规划的路径，等等，从而更方便地调试程序。

我们相信，机器人的硬件层面和软件层面一定都会向着统一架构的方向前行，而ROS正是软件架构层面标准化一个重要的里程碑。其中，ROS 1.x在之前被大量用于实验室的研究，或者公司产品demo的研发阶段，而ROS2则解决了ROS实时性的问题，未来很有可能被直接用于实际产品的研发，为推进工业级机器人和服务机器人的应用做出重要的贡献。

本附录概述性地介绍了有关ROS的历史、优点，以及如何利用ROS中的一些可视化工具来辅助SLAM程序开发等。我们希望读者系统地学习ROS，并使用ROS开发自己的SLAM程序。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

参考文献

[1]L.Haomin,Z.Guofeng,andB.Hujun,“Asurveyofmonocularsimultaneous *of Computer-Aided Design and Compute Graphics*,vol.28,no.6,pp.855–868,2016.in Chinese.

[2]A.Davison,I.Reid,N.Molton,and O.Stasse,“Monoslam:Real-time single camera SLAM,”*IEEE Transactions on Pattern Analysis and Machine Intelligence*,vol.29,no.6,pp.1052–1067,2007.

[3]R.Hartley and A.Zisserman,*Multiple View Geometry in Computer Vision*.Cambridge university press,2003.

[4]R.C.Smith and P.Cheeseman,“On the representation and estimation of spatial uncertainty,”*International Journal of Robotics Research*,vol.5,no.4,pp.56–68,1986.

[5]S.Thrun,W.Burgard,and D.Fox,*Probabilistic robotics*.MIT Press,2005.

[6]T.Barfoot,“State estimation for robotics:A matrix lie group approach,”2016.

[7]A.Pretto,E.Menegatti,andE.Pagello,“Omnidirectionaldenselarge-scalemappingandnavigation based on meaningful triangulation,”*2011 IEEE International Conference on Robotics and Automation(ICRA 2011)*,pp.3289–96,2011.

[8]B.RueckauerandT.Delbruck,“Evaluationofevent-basedalgorithmsforoptical fl owwithgroundtruth from inertial measurement sensor,”*Frontiers in neuroscience*,vol.10,2016.

[9]C.Cesar,L.Carlone,H.C.,Y.Latif,D.Scaramuzza,J.Neira,I.D.Reid,andL. future of simultaneous localization and mapping:Towards the robust-perception age,”*arXiv preprint arXiv:1606.05830*,2016.

[10]P.Newman and K.Ho,“Slam-loop closing with visually salient

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

features,”in *proceedings of the 2005 IEEE International Conference on Robotics and Automation*,pp.635–642,IEEE,2005.

[11]R.Smith,M.Self,and P.Cheeseman,“Estimating uncertain spatial relationships in robotics,”in *Autonomous robot vehicles*,pp.167–193,Springer,1990.

[12]P.Beeson,J.Modayil,andB.Kuipers,“Factoringthemappingproblem:M building in the hybrid spatial semantic hierarchy,”*International Journal of Robotics Research*,vol.29,no.4,pp.428–459,2010.

[13]H.Strasdat,J.M.Montiel,and A.J.Davison,“Visual slam:Why fi lter?,”*Image and Vision Computing*,vol.30,no.2,pp.65–77,2012.

[14]M.Liang,H.Min,and R.Luo,“Graph-based slam:A survey,”*ROBOT*,vol.35,no.4,pp.500–512,2013.in Chinese.

[15]J.Fuentes-Pacheco,J.Ruiz-Ascencio,and J.M.Rendón-Mancha,“Visual simultaneous localization and mapping:a survey,”*Arti fi cial Intelligence Review*,vol.43,no.1,pp.55–81,2015.

[16]J.Boal,Á.Sánchez-Miralles,and Á.Arranz,“Topological simultaneous localization and mapping:a survey,”*Robotica*,vol.32,pp.803–821,2014.

[17]S.Y.Chen,“Kalman fi lter for robot vision:A survey,”*IEEE Transactions on Industrial Electronics*,vol.59,no.11,pp.4409–4420,2012.

[18]Z.Chen,J.Samarabandu,andR.Rodrigo,“Recentadvancesinsimultaneo using computer vision,”*Advanced Robotics*,vol.21,no.3-4,pp.233–265,2007.

[19]J.Stuelpnagel,“On the parametrization of the three-dimensional rotation group,”*SIAM Review*,vol.6,no.4,pp.422–430,1964.

[20]V.S.Varadarajan,*Lie groups,Lie algebras,and their representations*,vol.102.Springer Science&Business Media,2013.

[21]H.Strasdat,*Localaccuracyandglobalconsistencyfore ffi cientvisualslam*.PhDthesis,Citeseer,2012.

[22]S.Agarwal,N.Snavely,I.Simon,S.M.Seitz,and R.Szeliski,“Building rome in a day,”in *2009 IEEE 12th international conference on computer*

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

vision,pp.72–79,IEEE,2009.

[23]J.Nocedal and S.Wright,*Numerical Optimization*.Springer Science&Business Media,2006.

[24]M.I.Lourakis and A.A.Argyros,“Sba:A software package for generic sparse bundle adjustment,”*ACM Transactions on Mathematical Software(TOMS)*,vol.36,no.1,p.2,2009.

[25]G.Sibley,“Relativebundleadjustment,”*DepartmentofEngineeringScie*

[26]B.Triggs,P.F.McLauchlan,R.I.Hartley,and A.W.Fitzgibbon,“Bundle adjustment:a modern synthesis,”in *Vision algorithms:theory and practice*,pp.298–372,Springer,2000.

[27]S.Agarwal,K.Mierle,and Others,“Ceres solver.”http://ceres-solver.org.

[28]R.Kummerle,G.Grisetti,H.Strasdat,K.Konolige,and W.Burgard,“G2o:a general framework for graph optimization,”in *IEEE International Conference on Robotics and Automation(ICRA)*,pp.3607–3613,IEEE,2011.

[29]Wikipedia,“Feature(computervision).”"https://en.wikipedia.org/wiki/ [Online;accessed 09-July-2016].

[30]D.G.Lowe,“Distinctive image features from scale-invariant keypoints,”*International Journal of Computer Vision*,vol.60,no.2,pp.91–110,2004.

[31]H.Bay,T.Tuytelaars,and L.Van Gool,“Surf:Speeded up robust features,”in *Computer Vision–ECCV 2006*,pp.404–417,Springer,2006.

[32]E.Rublee,V.Rabaud,K.Konolige,and G.Bradski,“Orb:an e ffi cient alternative to siftor surf,”in *2011 IEEE International Conference on Computer Vision(ICCV)*,pp.2564–2571,IEEE,2011.

[33]E.Rosten and T.Drummond,“Machine learning for high-speed corner detection,”in *European conference on computer vision*,pp.430–443,Springer,2006.

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

[34]M.Calonder,V.Lepetit,C.Strecha,and P.Fua,“Brief:Binary robust independent elementary features,”in *European conference on computer vision*,pp.778–792,Springer,2010.

[35]P.L.Rosin,“Measuringcornerproperties,”*ComputerVisionandImageU* 307,1999.

[36]M.Muja and D.G.Lowe,“Fast approximate nearest neighbors with automatic algorithm con fi guration.,”in *VISAPP(1)*,pp.331–340,2009.

[37]J.Sturm,N.Engelhard,F.Endres,W.Burgard,and D.Cremers,“A benchmark for the evaluation of rgb-d SLAM systems,”in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS)*,pp.573–580,IEEE,2012.

[38]R.I.Hartley,“In defense of the eight-point algorithm,”*IEEE Transactions on pattern analysis and machine intelligence*,vol.19,no.6,pp.580–593,1997.

[39]H.C.Longuet-Higgins,“A computer algorithm for reconstructing a scene from two projections,”*Readings in Computer Vision:Issues,Problems,Principles,and Paradigms,MA Fischler and O.Firschein,eds*,pp.61–62,1987.

[40]H.Li and R.Hartley,“Five-point motion estimation made easy,”in *18th International Conference on Pattern Recognition(ICPR’06)*,vol.1,pp.630–633,IEEE,2006.

[41]D.Nistér,“An e ffi cient solution to the fi ve-point relative pose problem,”*IEEE Transactions on Pattern Analysis and Machine Intelligence*,vol.26,no.6,pp.756–770,2004.

[42]O.D.Faugeras and F.Lustman,“Motion and structure from motion in a piecewise planar environment,”*InternationalJournal ofPattern RecognitionandArti fi cial Intelligence*,vol.2,no.03,pp.485–508,1988.

[43]Z.Zhang and A.R.Hanson,“3d reconstruction based on homography mapping,”*ARPA Image Understanding Workshop*,pp.1007–1012,1996.

[44]E.Malis and M.Vargas,*Deeper understanding of the homography*

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

decomposition for vision-based control.PhD thesis,INRIA,2007.

[45]X.-S.Gao,X.-R.Hou,J.Tang,and H.-F.Cheng,“Complete solution classi fi cation for the perspective-three-point problem,”*IEEE Transactions on Pattern Analysis and Machine Intelligence*,vol.25,pp.930–943,Aug 2003.

[46]V.Lepetit,F.Moreno-Noguer,and P.Fua,“Epnp:An accurate o(n)solution to the pnp problem,”*International Journal of Computer Vision*,vol.81,no.2,pp.155–166,2008.

[47]A.Penate-Sanchez,J.Andrade-Cetto,and F.Moreno-Noguer,“Exhaustive linearization for robust camera pose and focal length estimation,”*IEEE Transactions on Pattern Analysis and Machine Intelligence*,vol.35,no.10,pp.2387–2400,2013.

[48]L.Chen,C.W.Armstrong,and D.D.Raftopoulos,“An investigation on the accuracy of threedimensional space reconstruction using the direct linear transformation technique,”*Journal of Biomechanics*,vol.27,no.4,pp.493–500,1994.

[49]iplimage,“P3p(blog).”"http://iplimage.com/blog/p3p-perspective-point-overview/",2016.

[50]K.S.Arun,T.S.Huang,and S.D.Blostein,“Least-squares fi tting of two 3-d point sets,”*Pattern Analysis and Machine Intelligence,IEEE Transactions on*,no.5,pp.698–700,1987.

[51]F.Pomerleau,F.Colas,and R.Siegwart,“A review of point cloud registration algorithms for mobile robotics,”*Foundations and Trends in Robotics(FnTROB)*,vol.4,no.1,pp.1–104,2015.

[52]O.D.Faugeras and M.Hebert,“The representation,recognition,and locating of 3-d objects,”*The International Journal of Robotics Research*,vol.5,no.3,pp.27–52,1986.

[53]B.K.Horn,“Closed-form solution of absolute orientation using unit quaternions,”*JOSA A*,vol.4,no.4,pp.629–642,1987.

[54]G.C.Sharp,S.W.Lee,andD.K.Wehe,“Icpregistrationusinginvariantfeat*on Pattern Analysis and Machine Intelligence*,vol.24,no.1,pp.90–102,2002.

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

[55]G.Silveira,E.Malis,and P.Rives,“An e ffi cient direct approach to visual slam,”*IEEE Transactions on Robotics*,vol.24,no.5,pp.969–979,2008.

[56]C.Forster,M.Pizzoli,and D.Scaramuzza,“Svo:Fast semi-direct monocular visual odometry,”in *Robotics and Automation(ICRA),2014 IEEE International Conference on*(rs,ed.),pp.15–22,IEEE,2014.

[57]J.Engel,T.Schöps,and D.Cremers,“Lsd-slam:Large-scale direct monocular slam,”in *Computer Vision–ECCV 2014*,pp.834–849,Springer,2014.

[58]J.Engel,V.Koltun,and D.Cremers,“Direct sparse odometry,”*arXiv preprint arXiv:1607.02565*,2016.

[59]J.Engel,J.Sturm,and D.Cremers,“Semi-dense visual odometry for a monocular camera,”in *Proceedings of the IEEE International Conference on Computer Vision*,pp.1449–1456,2013.

[60]V.Usenko,J.Engel,J.Stueckler,and D.Cremers,“Direct visual-inertial odometry with stereo cameras,”in *IEEE International Conference on Robotics and Automation(ICRA)*,May 2016.

[61]Wikipedia,“Randomsampleconsensus.”"https://en.wikipedia.org/wik [Online;accessed 09-July-2016].

[62]V.Sujan and S.Dubowsky,“E ffi cient information-based visual robotic mapping in unstructured environments,”*International Journal of Robotics Research*,vol.24,no.4,pp.275–293,2005.

[63]F.Janabi-Shari fi and M.Marey,“A kalman- fi lter-based method for pose estimation in visual servoing,”*IEEE Transactions on Robotics*,vol.26,no.5,pp.939–947,2010.

[64]S.Li and P.Ni,“Square-root unscented kalman fi lter based simultaneous localization and mapping,”in *Information and Automation(ICIA),2010 IEEE International Conference on*,pp.2384–2388,IEEE,2010.

[65]R.Sim,P.Elinas,and J.Little,“A study of the rao-blackwellised particle fi lter for e ffi cient and accurate vision-based slam,”*International*

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

Journal of Computer Vision,vol.74,no.3,pp.303–318,2007.

[66]J.S.Lee,S.Y.Nam,and W.K.Chung,“Robust rbpf-slam for indoor mobile robots using sonar sensors in non-static environments,”*Advanced Robotics*,vol.25,no.9-10,pp.1227–1248,2011.

[67]A.Gil,O.Reinoso,M.Ballesta,and M.Julia,“Multi-robot visual slam using a rao-blackwellized particle fi lter,”*Robotics and Autonomous Systems*,vol.58,no.1,pp.68–80,2010.

[68]G.Sibley,L.Matthies,and G.Sukhatme,“Sliding window fi lter with application to planetary landing,”*Journal of Field Robotics*,vol.27,no.5,pp.587–608,2010.

[69]L.M.Paz,J.D.Tardós,and J.Neira,“Divide and conquer:Ekf slam in o(n),”*IEEE Transactions on Robotics*,vol.24,no.5,pp.1107–1120,2008.

[70]O.G.Grasa,J.Civera,and J.Montiel,“Ekf monocular slam with relocalization for laparoscopic sequences,”in *Robotics and Automation(ICRA),2011 IEEE International Conference on*,pp.4816–4821,IEEE,2011.

[71]E.Süli and D.F.Mayers,*An Introduction to Numerical Analysis*.Cambridge university press,2003.

[72]L.Polok,V.Ila,M.Solony,P.Smrz,and P.Zemcik,“Incremental block cholesky factorization for nonlinear least squares in robotics.,”in *Robotics:Science and Systems*,2013.

[73]R.Mur-Artal,J.Montiel,and J.D.Tardos,“Orb-slam:a versatile and accurate monocular slam system,”*arXiv preprint arXiv:1502.00956*,2015.

[74]S.Leutenegger,S.Lynen,M.Bosse,R.Siegwart,and P.Furgale,“Keyframe-based visual–inertial odometry using nonlinear optimization,”*The International Journal of Robotics Research*,vol.34,no.3,pp.314–334,2015.

[75]“Bundle adjustment in the large.”http://grail.cs.washington.edu/projects/bal/.

[76]J.Sherman and W.J.Morrison,“Adjustment of an inverse matrix

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

corresponding to a change in one element of a given matrix,"*The Annals of Mathematical Statistics*,vol.21,no.1,pp.124–127,1950.

[77]H.Strasdat,A.J.Davison,J.M.M.Montiel,and K.Konolige,"Double window optimisation for constant time visual SLAM,"*2011 IEEE International Conference On Computer Vision(ICCV)*,pp.2352–2359,2011.

[78]G.Dubbelman and B.Browning,"Cop-slam:Closed-form online pose-chain optimization for visual slam,"*Robotics,IEEE Transactions on*,vol.31,pp.1194–1213,Oct 2015.

[79]D.Lee and H.Myung,"Solution to the slam problem in low dynamic environments using a pose graph and an rgb-d sensor,"*Sensors*,vol.14,no.7,pp.12467–12496,2014.

[80]Y.Latif,C.Cadena,and J.Neira,"Robust loop closing over time for pose graph slam,"*The International Journal of Robotics Research*,vol.32,no.14,pp.1611–1626,2013.

[81]G.Klein and D.Murray,"Parallel tracking and mapping for small ar workspaces,"in *Mixed and Augmented Reality,2007.ISMAR 2007.6th IEEE and ACM International Symposium on*,pp.225–234,IEEE,2007.

[82]D.Koller and N.Friedman,*Probabilistic graphical models:principles and techniques*.MIT press,2009.

[83]M.Kaess,A.Ranganathan,and F.Dellaert,"isam:Incremental smoothing and mapping,"*IEEE Transactions on Robotics*,vol.24,no.6,pp.1365–1378,2008.

[84]M.Kaess,H.Johannsson,R.Roberts,V.Ila,J.J.Leonard,and F.Dellaert,"isam2:Incremental smoothing and mapping using the bayes tree,"*The International Journal of Robotics Research*,p.0278364911430419,2011.

[85]D.M.Rosen,M.Kaess,and J.J.Leonard,"Rise:An incremental trust-region method for robust onlinesparseleast-squaresestimation,"*IEEETransactionsonRobotics*,vol.30,no.5,pp.1091–1108,2014.

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

[86]J.Sola,“Course on slam.”https://github.com/joansola/slamtb/raw/graph/courseSLAM.pdf,2016.

[87]F.Dellaert,“Factor graphs and gtsam:A hands-on introduction,”2012.

[88]F.Endres,J.Hess,J.Sturm,D.Cremers,andW.Burgard,“3-dmappingwithanrgb-dcamera,”*IEEE Transactions on Robotics*,vol.30,no.1,pp.177–187,2014.

[89]D.Hahnel,W.Burgard,D.Fox,and S.Thrun,“An e ffi cient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements,”in *Intelligent Robots and Systems,2003. (IROS2003).Proceedings.2003IEEE/RSJInternationalConferenceon*,vol.1,pp.2 211,IEEE,2003.

[90]I.Ulrich and I.Nourbakhsh,“Appearance-based place recognition for topological localization,”in *Robotics and Automation,2000.Proceedings.ICRA’00.IEEE International Conference on*,vol.2,pp.1023–1029,Ieee,2000.

[91]X.Gao and T.Zhang,“Robust rgb-d simultaneous localization and mapping using planar point features,”*Robotics and Autonomous Systems*,vol.72,pp.1–14,2015.

[92]S.Lloyd,“Leastsquaresquantizationinpcm,”*IEEEtransactionsoninforr* 137,1982.

[93]D.Arthur and S.Vassilvitskii,“K-means++:The advantages of careful seeding,”in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*,pp.1027–1035,Society for Industrial and Applied Mathematics,2007.

[94]M.Cummins and P.Newman,“Fab-map:Probabilistic localization and mapping in the space of appearance,”*The International Journal of Robotics Research*,vol.27,no.6,pp.647–665,2008.

[95]M.Cummins and P.Newman,“Accelerating fab-MAP with concentration inequalities,”*IEEE Transactions On Robotics*,vol.26,no.6,pp.1042–1050,2010.

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

[96]M.CumminsandP.Newman,“Appearance-onlyslamatlargescalewithfab-map2.0,”*International Journal of Robotics Research*,vol.30,no.9,pp.1100–1123,2011.

[97]C.ChowandC.Liu,“Approximatingdiscreteprobabilitydistributionswit *transactions on Information Theory*,vol.14,no.3,pp.462–467,1968.

[98]D.Galvez-Lopez and J.D.Tardos,“Bags of binary words for fast place recognition in image sequences,”*IEEE Transactions On Robotics*,vol.28,no.5,pp.1188–1197,2012.

[99]J.L.Bentley,“Multidimensionalbinarysearchtreesusedforassociativese *of the ACM*,vol.18,no.9,pp.509–517,1975.

[100]J.SivicandA.Zisserman,“Videogoogle:Atextretrievalapproachtoobje *Computer Vision,2003.Proceedings.Ninth IEEE International Conference on*,pp.1470–1477,IEEE,2003.

[101]S.Robertson,“Understandinginversedocumentfrequency:ontheoretic *of documentation*,vol.60,no.5,pp.503–520,2004.

[102]D.Nister and H.Stewenius,“Scalable recognition with a vocabulary tree,”in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition(CVPR’06)*,vol.2,pp.2161–2168,IEEE,2006.

[103]C.Cadena,D.Galvez-Lopez,J.D.Tardos,and J.Neira,“Robust place recognition with stereo sequences,”*IEEE Transactions on Robotics*,vol.28,no.4,pp.871–885,2012.

[104]X.Gao and T.Zhang,“Loop closure detection for visual slam systems using deep neural networks,”in *Control Conference(CCC),2015 34th Chinese*,pp.5851–5856,IEEE,2015.

[105]X.GaoandT.Zhang,“Unsupervisedlearningtodetectloopsusingdeepne slam system,”*Autonomous Robots*,pp.1–18,2015.

[106]B.Williams,M.Cummins,J.Neira,P.Newman,I.Reid,andJ.Tardós,“A techniques in monocular slam,”*Robotics and Autonomous Systems*,vol.57,no.12,pp.1188–1197,2009.

[107]M.LabbéandF.Michaud,“Onlinegloballoopclosuredetectionforlarge-

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

scalemulti-
sessiongraphbasedslam,”in*2014IEEE/RSJInternationalConferenceonIntelliger*
2666,IEEE,2014.

[108]R.F.Salas-Moreno,R.A.Newcombe,H.Strasdat,P.H.J.Kelly,and A.J.Davison,“Slam++:Simultaneous localisation and mapping at the level of objects,”*2013 IEEE Conference on Computer Vision and Pattern Recognition(CVPR)*,pp.1352–9,2013.

[109]M.Pizzoli,C.Forster,andD.Scaramuzza,“Remode:Probabilistic,monc in real time,”in *2014 IEEE International Conference on Robotics and Automation(ICRA)*,pp.2609–2616,IEEE,2014.

[110]“Correlation based similarity measure-summary.”https://siddhantahuja.wordpress.com/tag/stereo-matching/.

[111]H.HirschmullerandD.Scharstein,“Evaluationofcostfunctionsforstere *Conference on Computer Vision and Pattern Recognition*,pp.1–8,IEEE,2007.

[112]G.VogiatzisandC.Hernández,“Video-based,real-timemulti-viewstereo,”*ImageandVisionComputing*,vol.29,no.7,pp.434–441,2011.

[113]A.Handa,R.A.Newcombe,A.Angeli,and A.J.Davison,“Real-time camera tracking:When is high frame-rate best?,”in *European Conference on Computer Vision*,pp.222–235,Springer,2012.

[114]J.Montiel,J.Civera,andA.J.Davison,“Uni fi edinversedepthparametrizationformonocularslam,”*analysis*,vol.9,p.1,2006.

[115]J.Civera,A.J.Davison,and J.M.Montiel,“Inverse depth parametrization for monocular slam,”*IEEE transactions on robotics*,vol.24,no.5,pp.932–945,2008.

[116]M.Kazhdan,M.Bolitho,and H.Hoppe,“Poisson surface reconstruction,”in *Proceedings of the fourth Eurographics symposium on Geometry processing*,vol.7,2006.

[117]J.Stuckler and S.Behnke,“Multi-resolution surfel maps for e ffi cient dense 3d modeling and tracking,”*Journal of Visual Communication and Image Representation*,vol.25,no.1,pp.137–147,2014.

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

[118]A.Hornung,K.M.Wurm,M.Bennewitz,C.Stachniss,and W.Burgard,“Octomap:An e ffi cient probabilistic3dmappingframeworkbasedonoctrees,”*AutonomousRobots*,vol.34 206,2013.

[119]M.Burri,H.Oleynikova,M.W.Achtelik,and R.Siegwart,“Real-time visual-inertial mapping,re-localization and planning onboard mavs in unknown environments,”in *Intelligent Robots and Systems(IROS),2015 IEEE/RSJ International Conference on*,pp.1872–1878,IEEE,2015.

[120]R.A.Newcombe,A.J.Davison,S.Izadi,P.Kohli,O.Hilliges,J.Shotton,I A.Fitzgibbon,“Kinectfusion:Real-time dense surface mapping and tracking,”in *2011 10th IEEE international symposium on Mixed and augmented reality(ISMAR)*,pp.127–136,IEEE,2011.

[121]R.A.Newcombe,D.Fox,and S.M.Seitz,“Dynamicfusion:Reconstruction and tracking of nonrigid scenes in real-time,”in *Proceedings of the IEEE conference on computer vision and pattern recognition*,pp.343–352,2015.

[122]T.Whelan,S.Leutenegger,R.F.Salas-Moreno,B.Glocker,and A.J.Davison,“Elasticfusion:Dense slam without a pose graph,”*Proc.Robotics:Science and Systems,Rome,Italy*,2015.

[123]M.Dou,S.Khamis,Y.Degtyarev,P.Davidson,S.R.Fanello,A.Kowdle,S *al.*,“Fusion4d:real-time performance capture of challenging scenes,”*ACM Transactions on Graphics(TOG)*,vol.35,no.4,p.114,2016.

[124]M.Innmann,M.Zollhöfer,M.Nießner,C.Theobalt,and M.Stamminger,“Volumedeform:Realtime volumetric non-rigid reconstruction,”*arXiv preprint arXiv:1603.08161*,2016.

[125]A.J.Davison,“Real-timesimultaneouslocalisationandmappingwithasinglecamera,”in*Computer Vision,2003.Proceedings.Ninth IEEE International Conference on*,pp.1403–1410,IEEE,2003.

[126]C.Kerl,J.Sturm,andD.Cremers,“Robustodometryestimationforrgb-dcameras,”in*Roboticsand Automation(ICRA),2013 IEEE International*

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

Conference on,pp.3748–3754,IEEE,2013.

[127]C.Kerl,J.Sturm,and D.Cremers,“Dense visual slam for rgb-d cameras,”in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*,pp.2100–2106,IEEE,2013.

[128]J.Gui,D.Gu,S.Wang,and H.Hu,“A review of visual inertial odometry from fi ltering and optimisation perspectives,”*Advanced Robotics*,vol.29,pp.1289–1301,Oct 18 2015.

[129]A.Martinelli,“Closed-formsolutionofvisual-inertialstructurefrommotion,”*InternationalJournal of Computer Vision*,vol.106,no.2,pp.138–152,2014.

[130]M.Bloesch,S.Omari,M.Hutter,and R.Siegwart,“Robust visual inertial odometry using a direct ekf-based approach,”in *Intelligent Robots and Systems(IROS),2015 IEEE/RSJ International Conference on*,pp.298–304,IEEE,2015.

[131]M.Li and A.I.Mourikis,“High-precision,consistent ekf-based visual-inertial odometry,”*International Journal of Robotics Research*,vol.32,pp.690–711,MAY 2013.

[132]G.Huang,M.Kaess,andJ.J.Leonard,“Towardsconsistentvisual-inertialnavigation,”in*2014IEEE International Conference on Robotics and Automation(icra)*,IEEE International Conference on RoboticsandAutomationICRA,pp.4926–4933,2014.IEEEInternationalConferenceonRobotics and Automation(ICRA),Hong Kong,PEOPLES R CHINA,MAY 31-JUN 07,2014.

[133]C.Forster,L.Carlone,F.Dellaert,andD.Scaramuzza,“Imupreintegratic ffi cient visual-inertial maximum-a-posteriori estimation,”in *Robotics:Science and Systems XI*,no.EPFLCONF-214687,2015.

[134]M.Tkocz and K.Janschek,“Towards consistent state and covariance initialization for monocular slam fi lters,”*Journal of Intelligent&Robotic Systems*,vol.80,pp.475–489,DEC 2015.

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

[135]A.NüchterandJ.Hertzberg,“Towardssemanticmapsformobilerobots,’ *Systems*,vol.56,no.11,pp.915–926,2008.

[136]J.Civera,D.Gálvez-López,L.Riazuelo,J.D.Tardós,and J.Montiel,“Towards semantic slam using a monocular camera,”in *Intelligent Robots and Systems(IROS),2011 IEEE/RSJ International Conference on*,pp.1277–1284,IEEE,2011.

[137]H.S.Koppula,A.Anand,T.Joachims,and A.Saxena,“Semantic labeling of 3d point clouds for indoor scenes,”in *Advances in Neural Information Processing Systems*,pp.244–252,2011.

[138]A.Anand,H.S.Koppula,T.Joachims,and A.Saxena,“Contextually guided semantic labeling and search for three-dimensional point clouds,”*The International Journal of Robotics Research*,p.0278364912461538,2012.

[139]N.Fioraio and L.Di Stefano,“Joint detection,tracking and mapping by semantic bundle adjustment,”*2013 IEEE Conference on Computer Vision and Pattern Recognition(CVPR)*,pp.1538–45,2013.

[140]R.F.Salas-Moreno,B.Glocken,P.H.Kelly,and A.J.Davison,“Dense planar slam,”in *Mixed and Augmented Reality(ISMAR),2014 IEEE International Symposium on*,pp.157–164,IEEE,2014.

[141]J.Stückler,N.Biresev,and S.Behnke,“Semantic mapping using object-class segmentation of rgb-d images,”in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*,pp.3005–3010,IEEE,2012.

[142]I.Kostavelis and A.Gasteratos,“Learning spatially semantic representations for cognitive robot navigation,”*Robotics and Autonomous Systems*,vol.61,no.12,pp.1460–1475,2013.

[143]C.Couprie,C.Farabet,L.Najman,and Y.LeCun,“Indoor semantic segmentation using depth information,”*arXiv preprint arXiv:1301.3572*,2013.

[144]J.Deng,W.Dong,R.Socher,L.-J.Li,K.Li,and L.Fei-Fei,“Imagenet:A large-scale hierarchical image database,”in *CVPR09*,2009.

[145]A.Krizhevsky,I.Sutskever,and G.E.Hinton,“Imagenet classi fi cation with deep convolutional neural networks,”in *Advances in neural*

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

information processing systems,pp.1097–1105,2012.

[146]K.He,X.Zhang,S.Ren,and J.Sun,“Deep residual learning for image recognition,”*arXiv preprint arXiv:1512.03385*,2015.

[147]S.Ren,K.He,R.Girshick,andJ.Sun,“Fasterr-cnn:Towardsreal-timeobjectdetectionwithregion proposal networks,”in *Advances in neural information processing systems*,pp.91–99,2015.

[148]J.Long,E.Shelhamer,and T.Darrell,“Fully convolutional networks for semantic segmentation,”*arXiv preprint arXiv:1411.4038*,2014.

[149]S.Zheng,S.Jayasumana,B.Romera-Paredes,V.Vineet,Z.Su,D.Du,C.Huang,and P.Torr,“Conditional random fi elds as recurrent neural networks,”in *International Conference on Computer Vision(ICCV)*,2015.

[150]S.Gupta,P.Arbeláez,R.Girshick,and J.Malik,“Indoor scene understanding with rgb-d images:Bottom-up segmentation,object detection and semantic segmentation,”*International Journal of Computer Vision*,pp.1–17,2014.

[151]K.Konda and R.Memisevic,“Learning visual odometry with a convolutional network,”in *International Conference on Computer Vision Theory and Applications*,2015.

[152]A.Kendall,M.Grimes,and R.Cipolla,“Posenet:A convolutional network for real-time 6-dof camera relocalization,”in *Proceedings of the IEEE International Conference on Computer Vision*,pp.2938–2946,2015.

[153]Y.Hou,H.Zhang,and S.Zhou,“Convolutional neural network-based image representation for visual loop closure detection,”*arXiv preprint arXiv:1504.05241*,2015.

[154]S.Y.An,J.G.Kang,L.K.Lee,and S.Y.Oh,“Line segment-based indoor mapping with salient line feature extraction,”*Advanced Robotics*,vol.26,no.5-6,pp.437–460,2012.

[155]H.Zhou,D.Zou,L.Pei,R.Ying,P.Liu,andW.Yu,“Structslam:Visualsla lines,”*Vehicular Technology,IEEE Transactions on*,vol.64,pp.1364–

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

1375,April 2015.

[156]D.Benedettelli,A.Garulli,and A.Giannitrapani,“Cooperative slam using m-space representation of linear features,”*Robotics and Autonomous Systems*,vol.60,no.10,pp.1267–1278,2012.

[157]J.P.Saarinen,H.Andreasson,T.Stoyanov,and A.J.Lilienthal,“3d normal distributions transform occupancy maps:An e ffi cient representation for mapping in dynamic environments,”*The International Journal of Robotics Research*,vol.32,no.14,pp.1627–1644,2013.

[158]W.Maddern,M.Milford,and G.Wyeth,“Cat-slam:probabilistic localisation and mapping using a continuousappearance-basedtrajectory,”*InternationalJournalofRoboticsResearch*,vol.31,no.4SI,pp.42 451,2012.

[159]H.Wang,Z.-G.Hou,L.Cheng,and M.Tan,“Online mapping with a mobile robot in dynamic and unknownenvironments,”*InternationalJournalofModelling,Identi fi cationandControl*,vol.4,no.4,pp.415–423,2008.

[160]D.Zou and P.Tan,“Coslam:Collaborative visual SLAM in dynamic environments,”*IEEE Transactions On Pattern Analysis And Machine Intelligence*,vol.35,no.2,pp.354–366,2013.

[161]T.A.Vidal-Calleja,C.Berger,J.Sola,andS.Lacroix,“Largescalemultiplerobotvisualmapping heterogeneous landmarks in semi-structured terrain,”*Robotics and Autonomous Systems*,vol.59,no.9,pp.654–674,2011.

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D



《TensorFlow 实战》
黄文坚 唐源 著



《第一本无人驾驶技术书》（封面待定）
刘少山 唐洁 吴双 宋力杭 著

三步加入“人工智能交流群”，实时获取资源共享
1. 扫码添加小编为微信好友。
2. 申请验证时输入“AI”。
3. 小编将邀请你加入“人工智能交流群”。

电子书寻找看手相qq37391319 钉钉或微信 pythontesting
书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

电子书寻找看手相qq37391319 钉钉或微信 pythontesting 书籍免费下载qq群6089740 钉钉群21745728

书籍汇总下载 http://t.cn/RDIAj5D

Broadview®
www.broadview.com.cn
博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版

隆重向读者推荐《视觉SLAM十四讲：从理论到实践》。一方面这本书是业界少有的涵盖从基础理论到代码实例，系统性讲解SLAM的书；另一方面，本书的作者和地平线颇有渊源，高翔曾经是我们的算法实习生，颜沁睿是自动驾驶算法工程师，都是在SLAM领域非常杰出的青年专家，走在技术实践的最前沿。在移动互联网大潮之后，自动驾驶、无人机、服务机器人等人工智能硬件会成为下一个产业爆发点，其中最关键的技术之一就是动态定位和环境建模的SLAM技术。本书是国内最新、最有价值的有关SLAM技术的书籍，适合有志于从事机器人技术的研究生和工程师，一定会让读者很有收获。

地平线机器人创始人，中国人工智能学会副秘书长

我在新加坡和加拿大给学生讲视觉SLAM的时候常常觉得缺乏一本适合初学者的教材。高翔博士的《视觉SLAM十四讲：从理论到实践》从最基础的四元数、李代数讲起，涵盖了卡尔曼滤波、Bundle Adjustment、Pose-Graph等高级优化工具。书中更有最近十多年成功系统的概述，从2003年的MonoSLAM直到2016年的ORB-SLAM。通篇既有清晰的理论叙述，又辅以大量示例程序，是一本非常好的视觉SLAM教材。

谭平　加拿大西蒙弗雷泽大学终身教授

视觉SLAM随着近年增强现实、无人驾驶等应用的兴起而重新获得重大关注。视觉SLAM属于计算机视觉和机器人研究的交叉领域，因此涉及的基础知识广而分散。国内专门的研究机构相对较少，因此学生入门的门槛较高。幸运的是，本书不仅有深入浅出的讲解，同时注重理论和实战的结合，大大降低了国内学生和相关从业者的进入门槛。因此，本书非常值得初学者学习实践。

刘海伟　网易感知与智能中心增强现实算法架构师

作者的这本书既是通俗有趣的高科技演义，又是足以指导研发实践的翔实教程，对国内SLAM界而言可谓意义重大。我甚至发现有不少目前圈内的一流人才都是因为看了本书的早期章节才决定进入这个行业并快速成长起来的。

本书所涵盖的知识面、技术细节，甚至是某些宝贵的最优实践经验，对国内刚刚起步的虚拟现实和增强现实（VR/AR）、无人机、无人车、机器人等行业，都将产生深远影响！

时驰（Chris）博士　uSens凌感科技联合创始人/首席运营官

博文视点Broadview
新浪微博 weibo.com
@博文视点Broadview

策划编辑：郑柳洁
责任编辑：白　涛
欢迎投稿
邮箱：zhenglj@phei.com.cn
微信号：Alinamercy

上架建议：人工智能
ISBN 978-7-121-31104-8
9 787121 311048 >
定价：75.00元

电子书寻找看手相qq37391319 钉钉或微信 pythontesting
书籍免费下载qq群6089740 钉钉群21745728