SLAM 中的几何与学习方法

Geometric and Learning Approaches Used in SLAM systems

盐粒

https://github.com/yanyan-li/SLAM-BOOK

updated 2021 年 6 月 11 日

目录

鲁迅

无穷的远方, 无数的人们
都和我有关

舟夜书所见
渣慎行

月黑见渔灯，孤光一点莹
微微风簇浪，散作满天星

辋川集 · 南🄵
王维

轻舟南🄵去，北🄵淼难即
隔浦望人家，遥遥不相识

叶芝

穿过我说谎的日子,
在阳光下抖掉枝条和花朵。
我现在可以枯萎而进入真理

说明：文章中的实验结果图片，除特殊说明以外，均来自我们自己的实验。读者可以引用，引用时，请说明出处。

```
@misc{gl-SLAM,
author = {Yanyan Li},
title = {SLAM中的几何与学习方法},
year = {2020},
publisher = {GitHub},
howpublished = {\url{https://github.com/yanyan-li/SLAM-BOOK}
}
```

1 书稿概述

1.1 执笔初衷

SLAM(Simultaneous Localization and Mapping) 关注传感器在环境中的定位与建图问题，其中定位部分研究的是传感器的六自由度位姿估计问题，而建图则是恢复未知环境中的三维信息。SLAM 系统提供的位姿和环境信息可以应用在增强/虚拟现实、机器人系统和无人驾驶系统中，为机器与环境的交互提供服务。

在 SLAM 社区，《Multiple View Geometry in Computer Vision》是一种字典、辞海一样的存在，它提供了丰富的理论知识，当我们对某一个概念不清楚的时候，可以去里面寻找答案。对于我们中文读者而言，《视觉 SLAM 十四讲》同样是一本非常精彩、扎实的书籍。高翔博士理论知识扎实、代码功力深厚，在书中深入浅出的把 SLAM 画卷展开在中国读者面前。与这些书籍相同的是，本书稿同样试图将作者自己对 SLAM 的理解重新梳理和归类，用更简单的话术将基础理论介绍给读者。在学习 SLAM 的过程中，作者和周围的一些朋友都曾有一种错觉:

> SLAM这个领域已经被挖掘完了，很难再取得突破.

产生这种错觉的主要原因可能是在于: 我们更多的是在学习 SLAM 的历史和基础理论，而不是去最新的进展中发觉尚未解决的问题。例如: ORB-SLAM 和 ORB-SLAM2 是非常重要、完整的 SLAM 系统，他们分别发表于 2015 年和 2017 年，到 2020 年 10 月，他们的引用量分别是 3300 和 2000。这意味着 SLAM 领的发展是十分迅速的。

基于此，本书稿的一个野心是为读者带去更新、更前沿的 SLAM 知识。通过梳理诸多新颖工作的 motivation 和他们关注的问题，我们在介绍充足的理论之后，努力的去探索这个领域的未来的可能性，以便在 SLAM 前沿研究中找到我们自己方向。

1.2 主要内容和面向读者

主要内容 对于不同的传感器，SLAM 系统在位姿估计和重建中应用的方法也大相径庭。对于依赖相机的（包括 RGB-D）系统，位姿估计可以分为 $2D-2D$，$2D-3D$ 和 $3D-3D$ 几种方式；对于激光雷达，有 $scan-scan$

和 $scan - map$ 的位姿估计策略；而对于 IMU 和 GPS，则由预积分和差分法等方法进行位姿估计。而重建部分则包括稀疏，半稠密和稠密几种方式。

近年来，深度学习在更高层次的视觉任务中取得瞩目的成绩，如：物体识别，语义分割等，这些课题曾是传统视觉无法或很难解决的任务。正是由于深度学习拓展了我们对高层次视觉任务的想象，越来越多的 SLAM 开始在他们的框架中融合学习的方法，来改进位姿估计的准确程度和环境重建的效果。但是深度学习是一个非常宽广的领域，和 SLAM 相关的课题只是它的一个分支。本书将会挑选和聚焦和 SLAM 相关的深度学习话题，我们希望能通过这本书来介绍 SLAM 系统中使用的几何和深度学习的方法，帮助读者掌握最新的进展。从整体上理解基于几何的 SLAM 系统的技术细节，同时还能了解深度学习相关课题对传统几何 SLAM 的影响。

本书首先分析单帧图像入手：单帧图像的几何特征提取，单帧图像的深度学习任务分析。通过相机模型和刚体变换的规则，我们把图像帧联系起来，构建帧与帧之间的关系。然后按照 SLAM 中的每一个重要环节来详细介绍相关的知识。几何与学习特征的处理、优化问题的构建与求解，以及环境地图的创建与维护。在每个环节中，我们都深入的介绍相关的理论与工程细节，力求本书为读者带来快速的提高。几何与学习就像是 SLAM 的一双翅膀，未来的 SLAM 将提供更加的鲁棒、更加丰富的信息。最后，我们一起探讨 SLAM 的发展方向，畅想这个领域未来蓝图。

在诸多种类的传感器中，本书稿主要和诸位读者一起探讨以相机为中心的 SLAM 系统，包括单目、双目 SLAM 系统、RGB-D SLAM 以及 VIO。围绕 RGB 相机，本书在介绍基础理论的同时，将深入探讨一下几个问题。

- 在传统的视觉几何中，如何使用点、线、面特征来提高视觉系统定位的精度和重建的效率。
- 深度学习在诸多视觉任务中取得了令人瞩目的表现，哪些学习信息可以被融合到 SLAM 系统中。
- 为关键的理论提供相关的代码，以及完整的 SLAM 系统。

面向读者 我们希望这本书稿能够给为研究 SLAM 的学生和相关从业者系统介绍几何特征与学习信息在 SLAM 系统中的提取和使用过程。更简单的说，如果诸位对《SLAM 视觉十四讲》感兴趣，那这本书稿就是为你而写的。

2 Hello SLAM

SLAM(Simultaneous Localization and Mapping) 关注传感器在环境中的定位与建图问题，其中定位部分研究的是传感器的 6D 的位姿估计问题，而建图则是试图恢复环境的 3D 信息。SLAM 系统提供的位姿和环境信息可以应用在增强/虚拟现实，无人驾驶系统中，为机器与环境的交互提供服务。

根据传感器的不同，SLAM 系统在位姿估计和重建中应用的方法也大相径庭。对于相机（包括 RGB-D），位姿估计可以分为 $2D-2D$，$2D-3D$ 和 $3D-3D$ 几种方式；对于激光雷达，有 $scan-scan$ 和 $scan-map$ 的位姿估计策略；而对于 IMU 和 GPS，则由预积分和差分法等方法进行位姿估计。而重建部分则包括稀疏，半稠密和稠密几种方式。

2.1 SLAM 研究方向的变迁

SLAM 领域经过二三十年的高速发展，逐渐成熟起来。每一篇划时代的论文都获得了巨大的引用量，通过阅读这些经典，我们可以感受整个 SLAM 研究方向的变迁。

2.1.1 基于特征的 SLAM 框架

对于基于特征法的 SLAM 系统，发表于 2007 年的 **PTAM** [16] 算是一个重要的里程碑，论文提出了基于多线程的 tracking 和 mapping 策略，以及关键帧的发展道路。其中 Mapping 部分是只处理关键帧的特征信息，并使用了局部光束法平差（Boundle Adjustment）来优化路标点和相机位姿。比起 2007 年之前的 SLAM 或视觉里程计系统相比，它可以实时维护几千个点的地图。

同样是基于点特征的系统，发表于 2017 年的 **ORB-SLAM2** [24] 是一个集大成的作品。这个系统是由 Raúl Mur-Artal 博士等人提出的一个完备的基于特征点的纯视觉实时 SLAM 系统，适用于单目、双目和 RGB-D 相机。作者在 2015 年第一次将单目版本（ORB-SLAM [23]）发表在 IEEE Transactions on Robotics（T-RO）期刊上，这是一个基于纯特征点的单目实时 SLAM 框架。两篇文章涉及的系统是非常完整的系统，通过多线程实现了稀疏地图、回环和优化环节，为 SLAM 在工业界和学术界的发展作出巨大贡献。

视觉SLAM十四讲 从理论到实践

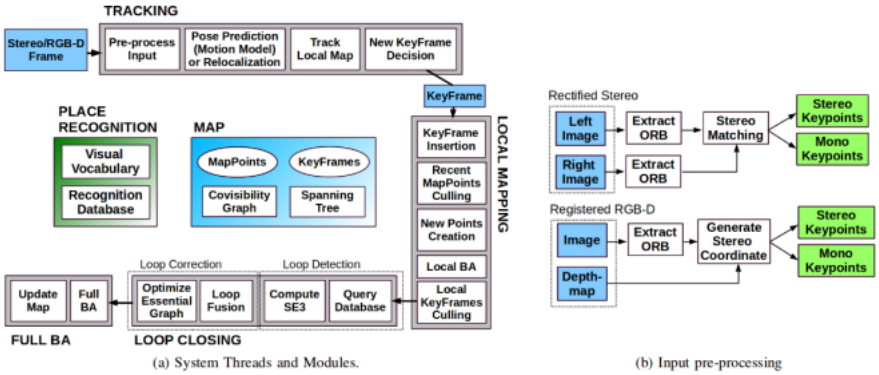


图 1: 特征法 SLAM（ORB-SLAM2）的经典框架。

在 ORB-SLAM（V1/V2）工程的基础上，图 5中的各个模块都吸引研究人员的注意力，最常见的就是为系统增加其他的特征信息。众所周知，点特征对于光照纹理都比较敏感，尤其是在纹理比较弱的地方，提取特征点的数目是很少的。为了应对低纹理场景中特征点能力的限制，很多点线，或是点线面的系统被提出来满足单目 [26]、双目 [9] 和 RGB-D 相机 [35]。随着新特征的加入，关键帧策略、地图和优化环节都会做相应的调整。

2.1.2 多几何特征

ORB-SLAM 系列利用的特征信息都是特征点，基于这些点特征系统，后续很都工作都尝试去增加新的特征类型。对于单目/双目传感器，早期的工作 PL-SLAM 在系统中增加直线信息；对于 RGB-D 传感器，则有工作尝试增加平面信息。在这些早期的工作中，增加的直线和平面一般主要是为了应对特征点较难提取的若纹理区域。由于没有充分利用这些信息的特点，即便是系统增加了特征的种类，SLAM 系统仍旧无法较大的提高系统的表现。

不同于传统的点-线-面系统，RGB-D SLAM [21] 进一步探索直线与直线，平面与平面之间的平行/垂直关系。由于，这种几何关系满足 Manhattan World 约束，这种约束带来了零漂移旋转估计和 3D 平移估计的策略。同时，在提取平面特征之后，我们对环境重建的能力也增强了，可以在 CPU 基础上对环境进行实时 mesh 重建。

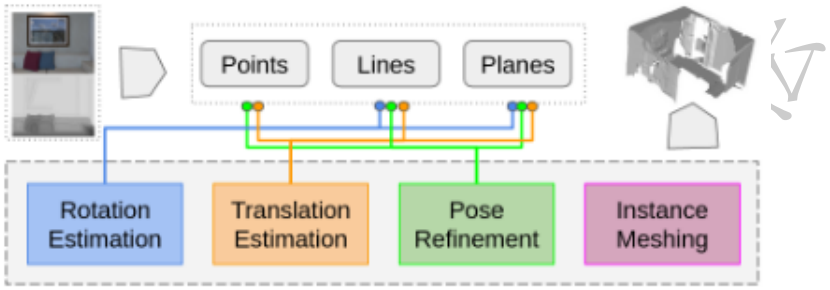


图 2: 多特征 SLAM 系统框架。

2.1.3 多传感器

由于运动模糊和位姿剧烈变化，快速运动情况为基于特征法的 SLAM 系统带来较大的挑战，因此研究人员通过增加传感器为系统增加更多的信息源。惯性测量单元（IMU）是一款非常便宜的设备，可以提供高帧率位姿估计结果。VINS-Mono [28] 是相机和 IMU 紧密耦合的经典系统，由于 IMU 可以高频的记录角速率和加速度信号，通过对高频信号进行预积分操作，系统可以稳定的获得两帧图像之间的相对位姿，然后通过相机带来的视觉特征的约束来实现位姿信息的优化。然后也有结合 IMU 与 ORB-SLAM 或 DSO 的工作 [34]，他们都取得了不错的效果。同样是加入线特征的策略，PL-VIO [10] 在 VINS-Mono 的基础上加入了线特征。

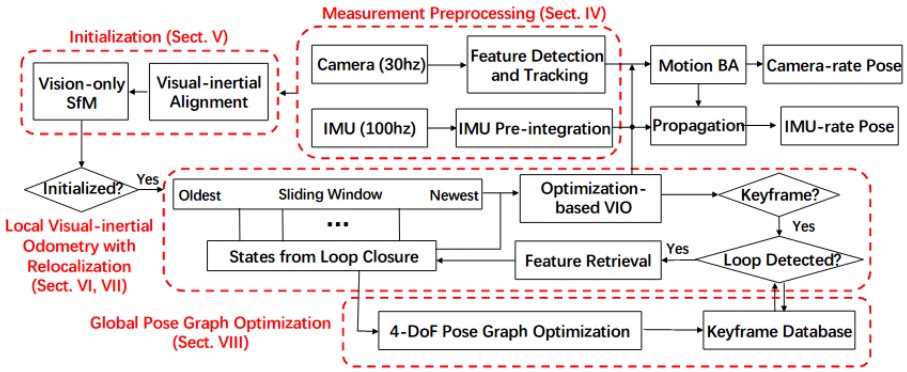


图 3: VIO 系统 (VINS-Mono) 经典框架。

2.1.4 几何与学习的融合

深度学习的出现帮助传统的 SLAM 系统拓展了传感器的边界，针对 SLAM 的目标，结合的目标主要是提升重建和位姿估计。比较早的结合工作有：DVSO、CNN-SLAM 这两篇工作都是发表在 2018 年，并且都是使用深度学习的估计的深度信息去恢复。

除了利用估计的深度信息帮助单目 SLAM 实现密集重建，还有一些新的工作直接利用深度学习提高位姿估计的效果。值得注意的是，深度学习提

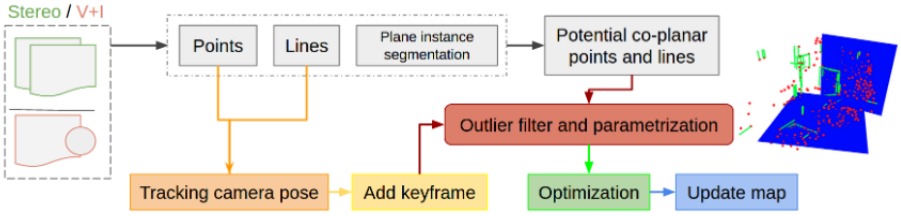


图 4: 基于学习平面的共面参数化方法（CP-Param [18]）。

供的特征并不像传统几何特征那样稳定，因此在为 SLAM 引入学习信息的时候，也要同时加入一个去伪存真的策略。

2.2 动手来写搭建开发环境

2.2.1 操作系统与开发软件

- Ubuntu 系统。很多系统都可以去开发 SLAM，但是方便的系统应该是 Linux 系统。Ubuntu 是 Linux 系统中的一种，系统简洁、操作方便，十分适合开发 SLAM、研究深度学习的朋友们。

- CLion + terminal。CLion 是一款非常优秀的 C++ 集成编译环境。可以在 Ubuntu 环境下使用，对于不熟悉 vim 的小伙伴，它非常容易使用，并且为学生用户提供免费使用的权利。

2.2.2 常用的开发库介绍

由于视觉前辈的卓越贡献，我们在编写 SLAM 系统的时候，不需要完全从底层编写。通过使用的常见开发库中的函数接口，我们就可以直接获得我们需要的阶段性结果。这些开发库提高了我们的开发效率，但是这些库是

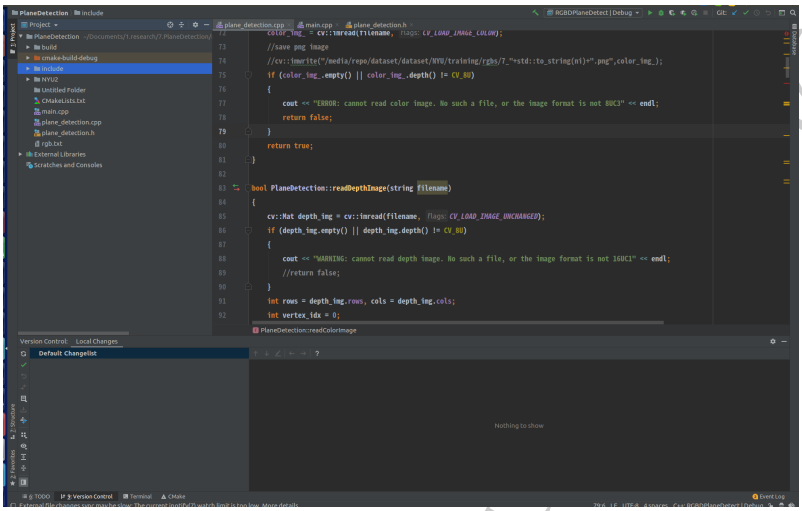


图 5: CLion 软件界面。

否可以免费用在商业代码中，还需要读者自己去研读相应库的开发者使用规范。

由于每个库本身都是内容极为丰富，例如 OpenCV 本身就有专门的书籍去介绍，因此我们这里只简单介绍与本书稿有关的部分，旨在让诸位对这些库的架构有一个大概的了解。

OpenCV

PCL

g2O

Pangolin

OpenGL

ROS ROS 系统是机器人领域很重要的平台，在大多数没有使用 ROS 的 SLAM 系统中，我们大概率都是使用 cmake 和 make 命令来完成代码编译，这种情况在 ROS 相关的系统发生了一些变化，通过对 cmake 进行扩展,ROS 系统构建了自己的编译工具 catkin. 要用 catkin 编译一个工程或软件包，只

高翔-SLAM 中的几何

需要用 $catkin_make$ 指令。一般当我们写完代码，执行一次 $catkin_make$ 进行编译, 调用系统自动完成编译和链接过程，构建生成目标文件。

Package 是 catkin 编译的基本单元，我们调用 $catkin_make$ 编译的对象就是一个个 ROS 的 package，也就是说任何 ROS 程序只有组织成 package 才能编译。所以 package 也是 ROS 源代码存放的地方，任何 ROS 的代码无论是 C++ 还是 Python 都要放到 package 中，这样才能正常的编译和运行.

盐粒-SLAM 中的几何与学习方法

3 几何视觉基础

计算机视觉基础是一个很庞大的课题，为了更贴近 SLAM 的研究主题，我们聚焦在视觉几何的部分，而这一章主要介绍 SLAM 中常用的图像/数据处理方法。

3.1 图像处理基础

3.1.1 RANSAC

随机抽样一致算法（Random Sample Consensus，RANSAC）[1]采用迭代的方式从一组包含外点的数据中估算出需求模型的参数。

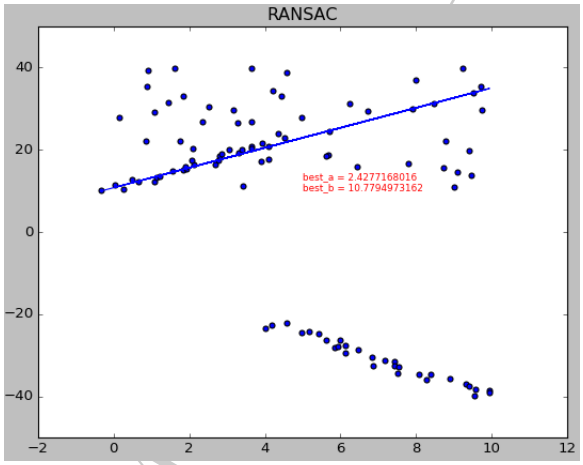


图 6: RANSAC 在 2D 数据中拟合直线。

- 随机从数据 D 中取最小抽样集 S，其中抽样点个数为 n，是模型初始化所需的最小样本数，基于抽样集 S 估计出模型 M。
- 计算最小抽样集 S 以外的数据与模型 M 之间的距离，距离小于阈值 t_d 的数据构成该模型的内点集 I_M。
- 若 I_M 中的个数大于阈值 t_n，则认为是找到模型 M 是该数据包含的正确模型。

[1] https://github.com/yanyan-li/SLAM-BOOK/tree/master/code/chapter 2

计算SLAM自学习方法

我们重复以上步骤多次，比较哪次计算中内群数量最多，最多内群点对应的模型就是该数据所要求模型。值得指出的是，图6中输入为两条直线，但是上述介绍的方式找到一个直线后就结束了。

3.2 几何特征提取

点线面特征被广泛的应用在在 SLAM 中，这些传统的特征提取与匹配方式, 经过长时间的发展, 已经逐渐稳定。

3.2.1 点特征提取与匹配

经常被使用的特征有 SIFT(Scale Invariant Feature Transformation) [19] 、SURF(Speeded up robust feature) [20] , ORB(Oriented fast and rotated BRIEF) [21] 等。这些特征通过不同的算法获得, 也有不同的应用场景。

SIFT 特征 SIFT 特征匹配算法是早期非常著名的特征描述和匹配算子, 为计算机视觉的发展起到了巨大的推动作用。这种特征可以很好的应对图像对之间的平移、旋转、尺度变换, 众多的实验表明了 SIFT 在上述情况下的鲁棒性。使用 SIFT 算子惊醒特征提取工作主要包括两个步骤: 分别是特征描述向量的生成, 如图 4 所示; 另一个是特征描述向量之间的匹配。同时这种两步方法也被后来众多的特征匹配算子继承, 甚至影响着使用深度学习完成匹配工作的网络框架。SIFT 算法可以很好的应对尺度、旋转情况下的

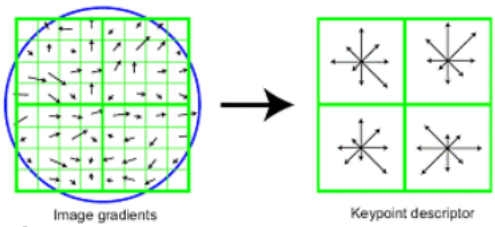


图 7: SIFT 特征提取的 128 维向量

匹配, 但是该算法建立在两幅图像满足相似变换这一假设。因此该算法适合处理相两幅图像邻角度小于 30°C 的图像, 对于较大倾角的倾斜图像,SIFT 只能提取很少量的特征, 同时伴随着较大比例的错误匹配, 因此 ASIFT 算子 [22] 被设计出来解决大倾角图像的匹配问题。

SURF 特征 SURF 算法同样是经历局部特征点的提取、描述和特征点的匹配, 但它一种稳健的具有尺度和旋转不变性的局部特征点检测和描述算法, 它降低了特征描述向量的维度, 同时这个算子更加容易实现并行处理。这种算子是对 SIFT 进行改进, 同样使用到图像空间金字塔等结构。不同于 SIFT 算法, 它使用黑塞矩阵 (Hessian Matrix) 和盒式滤波器去描述兴趣点 (Faster-Hessian Detector)。在特征的描述环节, 该方法首先是在兴趣点周围的圆形区域计算方向, 然后再根据选择的方向构建方形区域, 并从中提取特征点描述符。SURF 特征是一种具有较高准确度和速度的特征算子。

ORB 特征 由于 ORB-SLAM 系列和相关论文的影响, ORB 这种特征变得非常流行, 它运行速度非常的快, 我在比较好算力的台式机上从 $480*664$ 图片提取 ORB 特征并建立描述子需要 8ms 左右。那么 ORB 中有那些操作呢?

```
ComputePyramid(image);
ComputeKeyPointsOctTree(allKeypoints);
_descriptors.create(nkeypoints, 32, CV_8U);
```

首先, 我们将图片建立金字塔, 在金字塔每层图像上进行 FAST 特征提取。对于感兴趣的点, 如上图所示的 p 点, 以该点为中心, 3 像素为半径, 获得圆周上的 16 个像素。如果圆周上有连续 n 个像素点的灰度值大于或小于该点的灰度值, 则认为该点为 FAST 特征。

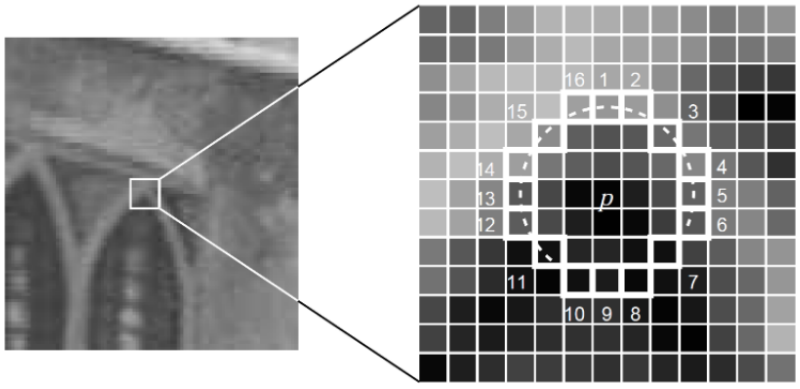


图 8: 感兴趣点 p 和圆周上的 16 个像素。 [33]

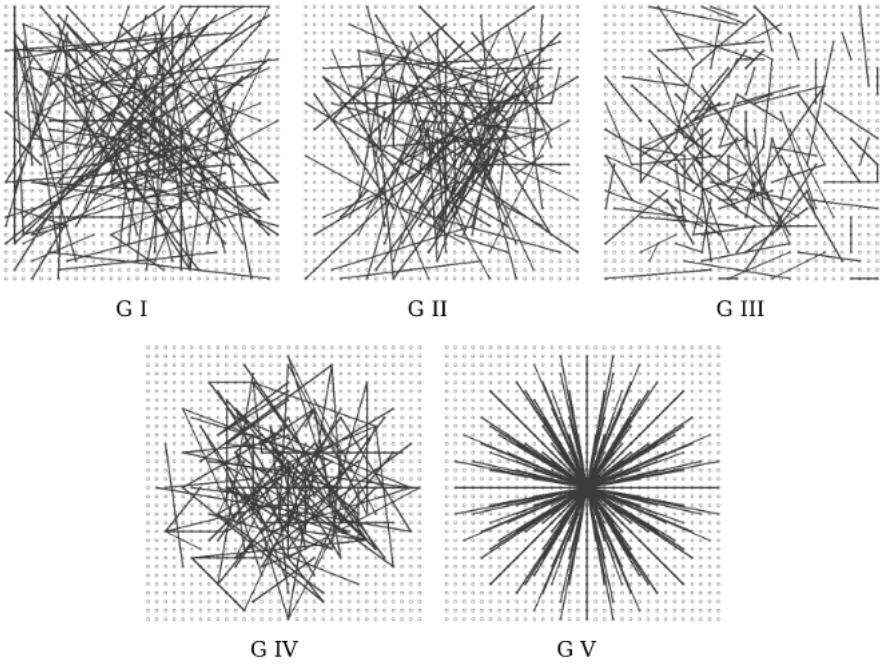


图 9: BRIEF [1] 不同选择点的方式。1. x,y 方向平均分布采样；2.x,y 均服从 Gauss(0,$S^2/25$) 各向同性采样；3.x 服从 Gauss(0,$S^2/25$)，y 服从 Gauss(0,$S^2/100$) 采样；4. x,y 从网格中随机获取；5.x 一直在 (0,0)，y 从网格中随机选取。

其次，我们要去除不好的 FAST 点。FAST 特征点和该点周围的 16 个像素输入决策树，筛选出来高质量的 FAST。对于临近的特征点，选择具有较大区分度的特征点，删除临近的其余点。

为了实现特征点的旋转不变性。对于特征点和 r 半径内的像素点，计算 r 半径范围内的质心，并建立从特征点位置到质心位置的向量，这个向量就是该特征点的方向。具体的计算过程如下所示：

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y) \tag{3.1}$$

这里的 x 和 y 表示特征点为中心，r 为半径的圆内点的坐标，他们范围是 $(-r,r)$。利用上面的动量公式：我们可以计算重心：

$$C = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}) \tag{3.2}$$

盗版-SJAM 中的）

因此围绕这个特征点的图像块的方向可以表示出来了：

$$\theta = atan2(m_{01}, m_{10}) \tag{3.3}$$

上述公式中的 $atan2()$ 是象限敏感（quadrant-aware）的反正切函数。

在获得特征点的位置和方向之后，我们一起来看如何为这个特征点建立描述子。ORB 在 BRIEF 的基础上改进出来自己的描述方法，我们先来回顾原始的 BRIEF 的过程。对于 $n = 256$ 个点对 (p_i, p_j)，每一对之间的灰度值 I 高低就产生了一个二值化结果。

$$r(p_i, p_j) = \begin{cases} 1 & I(p_i) < I(p_i) \\ 0 & I(p_i) > I(p_i) \end{cases} \tag{3.4}$$

因此 256 个像素对就构成了一个 256 维的向量。那么 BRIEF 是怎么选择这些点对呢？如图9所示，原论文给出了 5 种不同的思路，但是作者使用第二种：两个点服从高斯 $(0,S^2/25)$ 各向同性采样。

与传统 BRIEF 相比，ORB 为描述子增加了方向信息，其被称为 Steered BRIEF。原先的 n 个像素对构成了矩阵 S，

$$S = \begin{pmatrix} p_i^0 \cdots & p_i^1 \\ p_j^0 \cdots & p_j^1 \end{pmatrix} \tag{3.5}$$

把方向信息考虑在整体的描述子中，我们可以获得 ORB 特征构建的具有旋转不变性的描述子：

$$S_\theta = R_\theta S \tag{3.6}$$

上述公式中的 $R_\theta = \begin{bmatrix} cos\theta sin\theta \\ -sin\theta cos\theta \end{bmatrix}$。

3.2.2 线特征提取与匹配

LSD 是比较流行的一个直线提取方法，KNNMatch 做特征描述。同时，还有特征质量的简单筛选。

```
using namespace cv;
using namespace std;
using namespace cv::line_descriptor;
struct sort_descriptor_by_queryIdx
```

```
392     {
393         inline bool operator()(const vector<DMatch>&
394             a, const vector<DMatch>& b){
395     return ( a[0].queryIdx < b[0].queryIdx ); }
396     };
397     struct sort_lines_by_response
398     {
399         inline bool operator()(const KeyLine& a,
400             const KeyLine& b){
401              return ( a.response > b.response );
402         }
403     };
404
405     void ExtractLineSegment(const Mat &img, const Mat
406         &image2, vector<KeyLine> &keylines,vector<
407        KeyLine> &keylines2)
408     {
409         Mat mLdesc,mLdesc2;
410
411         vector<vector<DMatch>> lmatches;
412
413         Ptr<BinaryDescriptor> lbd = BinaryDescriptor
414             ::createBinaryDescriptor();
415         Ptr<line_descriptor::LSDDetector> lsd =
416             line_descriptor::LSDDetector::
417             createLSDDetector();
418
419         cout<<"extract lsd line segments"<<endl;
420         lsd->detect(img, keylines, 1.2,1);
421         lsd->detect(image2,keylines2,1.2,1);
422         int lsdNFeatures = 50;
423         cout<<"filter lines"<<endl;
424         if(keylines.size()>lsdNFeatures)
```

SLAM 中的几何与学习方法

```
425         {
426     sort(keylines.begin(), keylines.end(),
427             sort_lines_by_response());          keylines.resize(lsdNFeatures);
428             for( int i=0; i<lsdNFeatures; i++)
429                 keylines[i].class_id = i;
430         }
431         if(keylines2.size()>lsdNFeatures)
432         {
433             sort(keylines2.begin(), keylines2.end(),
434                sort_lines_by_response());
435             keylines2.resize(lsdNFeatures);
436             for(int i=0; i<lsdNFeatures; i++)
437                 keylines2[i].class_id = i;
438         }
439         cout<<"lbd describle"<<endl;
440         lbd->compute(img, keylines, mLdesc);
441         lbd->compute(image2,keylines2,mLdesc2);//计算
442            特征线段的描述子
443         BFMatcher* bfm = new BFMatcher(NORM_HAMMING,
444            false);
445         bfm->knnMatch(mLdesc, mLdesc2, lmatches, 2);
446         vector<DMatch> matches;
447         for(size_t i=0;i<lmatches.size();i++)
448         {
449             const DMatch& bestMatch = lmatches[i][0];
450             const DMatch& betterMatch = lmatches[i
451                ][1];
452             float  distanceRatio = bestMatch.distance
453                / betterMatch.distance;
454             if (distanceRatio < 0.75)
455                 matches.push_back(bestMatch);
456     }
457
```

std 盐-SLAM视觉中的几何与学习方

```
        cv::Mat outImg;
    std::vector<char> mask( lmatches.size(), 1 );
        drawLineMatches( img, keylines, image2, keylines2, matches, outImg, S
        Scalar::all( -1 ), mask,
                        DrawLinesMatchesFlags::
                            DEFAULT );

        imshow( "Matches", outImg );
        waitKey();

    }
```

最后提取的结果是下图展示的。最后大家觉得这种线提取方式，有什么优缺点，可以在下方评论或私信我，一起交流讨论。

图 10: LSD 直线提取与匹配结果

3.2.3 面特征提取与匹配

平面提取 平面提取一般是针对点云而言的，文章 [11] 总结了一个从点云到平面分割的基本方式。

- 首先是从 3D 点云计算 normal map.
- 第一步聚类，具有相似的 normal 方向时，分在一类 S。
- 空间点 × normal 产成距离，根据距离，再把这类等开成不同的 instance。

和上述方法 [11] 相比，AHCP [6] 提取平面的效果更好一些，当然计算也更复杂一些。每三个连续的 2D 点组成一个节点，就凑成了 a,b,c,d,e,...。通过 AHC 方法（块状聚类），找到 g 具有最小的 MSE，在把 g 与左右进行融合。通过将融合信息与阈值相比，删除误差大的组合。这种每三个点，step=1，就很容易通过 MSE 找出来边界。

AHC 是一个比较好的开源平面提取方案。这个方法分为四个阶段，首先是建立 Initial graph；AHC 分为两个 A，B 两个部分；粗糙的提取面，提取精细的面；如果不需要精细平面的时候，对于 640*480 的图片，可以达到 50Hz。

图 11: 平面提取结果。左图是深度图对应的 RGB 图像，中图是 AHC 方法提取的平面联通区域，右图是平面分割结果。

平面匹配 不像是点线特征，可以用一种特征来描述。平面的匹配大多是使用平面的法向量与原点到平面的距离来判断两平面的参数是不是一样。

3.3 几何特征之间的关系

在这个小节中，我们将特征 SLAM 系统中的约束罗列出来，主要的目标是介绍如何来形成这样的约束，至于如何在 SLAM 系统中利用这些约束，我们会在接下来的章节中逐一介绍。

我们从一帧图像上提取了点线面信息，通过探索单帧图像中特征之间的关系，我们可以获得更多的几何约束。

3.3.1 直线与直线平行-灭点

由于 3D 场景投影到图片的过程中，会产生投影变换，因此 3D 场景中平行的直线，在图片中会聚集于一点，这个点就成为是灭点（vanishing point）。

灭点在 SLAM 中有一些特别的用处，早期一些工作使用他们来做相机参数的计算，现在有些工作利用不同方向上的灭点的正交关系来进行结构化场景中的相机位姿追踪。

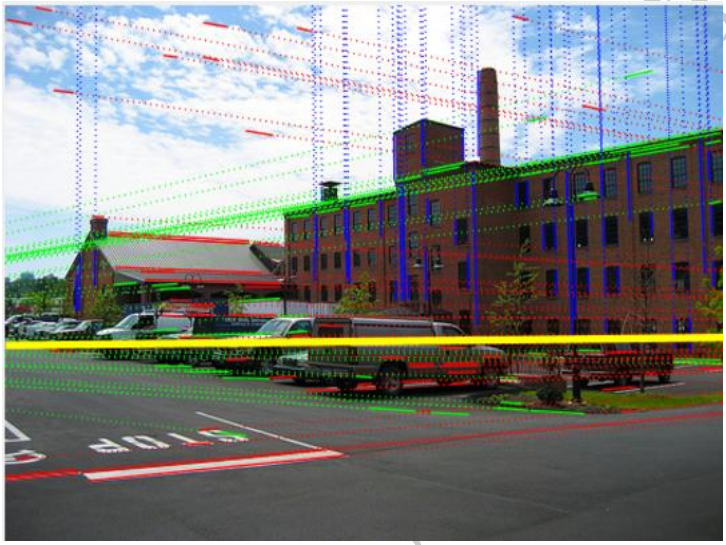


图 12: 灭点与平行直线。红绿蓝颜色代表三个方向上的平行直线, 黄色直线是灭线。

我们已经知道，2D 图像上的灭点是平行直线的交点。对于 2D 直线的方程 $ax + by + c = 0$, 我们可以通过联合两条直线的方程求解两条直线的交点 $p = (x, y)$。如果另外直线和上述两条直线平行, 那么 p 点距离该直线的距离应该是比较小的。由于像素误差的存在，平行直线不会是完全交于一点，因此存在很多方法（RANSAC,J-Linkage 等）[31] 去求出最合理的一个点。

3.3.2 平面与平面平行-灭线

3.3.3 平面与平面垂直-曼哈顿约束

如果在一副图像中找到两个垂直的平面，那这两个平面法向量就能够通过叉乘，获得（虚构）与这两个平面垂直的第三个平面，因此构成了一个正交平面系统，我们可以称其为曼哈顿世界（Manhattan World）。

与前面特征构成的约束不同，曼哈顿约束可以整体作为一种约束，这种约束在图像与图像之间可以发挥一作用。

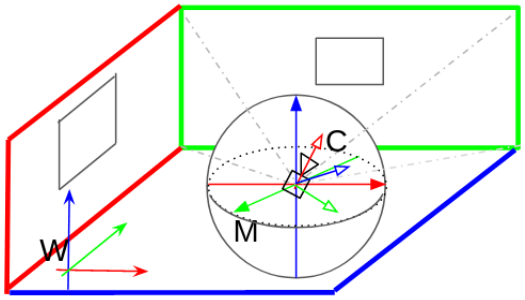


图 13: 曼哈顿世界。不同面之间具有垂直关系。

除了图像与图像之间的曼哈顿系统的约束之外，我们还可以将构成曼哈顿系统中的每个平面元素单独做出来平面级（非图像级）的约束。

$\sqrt{d}$

3.3.4 点在直线上-共线

3.3.5 点线在平面上-共面

点线在平面上带来的约束可以被成为共面约束，如图 14所示，空间点 P_i,P_j 和直线（n_l 与 n_π 的交线）都在平面（法向量 n_π 上）。

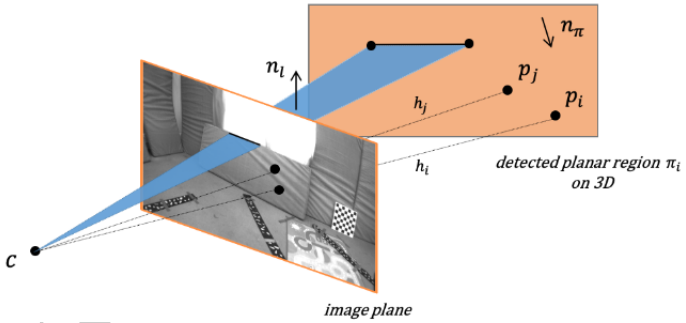


图 14: 点线共面几何 [18]。

共面约束产生的方程也是比较直接的，无非就是点和线的描述要满足平面的方程。

$$\begin{cases} n_x P_x + n_y P_y + n_z P_z + d = 0 \\ n_l \times n_\pi = dir(l) \end{cases} \tag{3.7}$$

上面的公式分别用简单直接的方式介绍点线构成的共面关系。这里点的表示是简单的 (P_x, P_y, P_z) 方式，直线的表示是用普朗克参数形式 (n_l, d) 的形式表示。值得说明的是，不同参数都是可以相互转化的，因此你可以将你喜欢的参数化形式用在共面约束中。

盐粒-SLAM 中的几何与学习方法

4 深度学习视觉基础

随着传感器技术的发展和计算能力的提高，价格便宜的 RGB-D 相机已经在很多场景中已经取得了很好的精度。其中 depth map 可以直接提供稠密的 3D 点云，他们对于场景理解，尤其是近距离场景，起到了重要的作用。但是这种传感器仍旧有很多限制，包括场景的大小、深度图空洞、边缘精度较差等。与此同时，深度学习迅速崛起，不仅在图片分类、物体检测、语义分割等任务上取得了突出的效果，而且在密集重建、场景补全、姿态估等傳統的 SLAM 问题上爆出巨大能力。当然很多人还会对深度学习嗤之以鼻，觉得网络缺少解释性。在笔者看来，大可不必这样。这种数据驱动的方法，虽然在很大的部分仍然未得到充分解释，但是他为计算机视觉带来的巨大想象空间是不容忽视的。

不管从传感器的价格，还是应用场景（高温、高压、极寒、较远距离）的通用性来说，单目相机永远保持着他独特的优势。当我们无法使用主动光获取深度图时，能否利用这种学习的方法来获得深度图呢？实际上，国内外的研究学者已经在这个问题上取得了很不错的效果，本文就是要对众多的理论、方法进行梳理，分析其中主要方法的逻辑和优缺点，以便后来的读者可以更简单的理解问题、贡献智慧。

4.1 经典网络回顾

当前深度学习已经在很多任务取得了很多令人瞩目的成果，其实早期学习任务是从分类开始的，然后基于学习任务很多网络结构被提出来，这些经典的网络深度的探索了基于卷积神经网络的特征提取工作，如 VGG 系列，$ResNet$ 系列，$DenseNet$ 系列等。这些经典的网络如今被用在各种新的学习任务中，支持网络的优秀表现。因此我们先从分类网络开始，重新理解网络的设计与一些 module 的设计初衷，汲取炼丹经验。

输入： $32 \times 32 = 1024$ 的手写字体图片。这些手写字体包含 0 9 数字，这个问题也就是相当于是一个 10 个类别的图片。输入一个数字，找到合适的分类。

激活函数 ReLU

$tanh()$ 函数: 饱和的非线性函数在梯度下降的时候要比非饱和的非线性函数慢得多，在 AlexNet 中使用 ReLU 函数作为激活函数, 达到相同准确率的耗时更少。

浙江-SLAM中的几何与学习方法

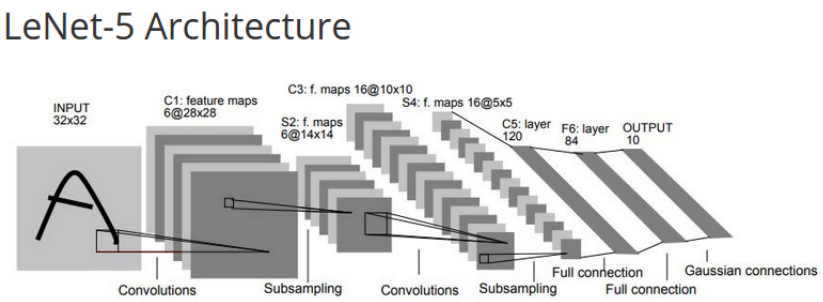


图 15: LeNet-5

卷积	pooling	全连接层
c1:5×5 * 6	s2: MaxPool, s=2	F6
c3:5×5 * 6	s4: MaxPool, s=2	output
c5:5×5 * 120		

表 1: LeNet-5 网络参数

但是：由于 ReLU 函数没有一个有限的值域区间，所以在 ReLU 之后需要进行归一化处理，LRN（**Local Response Normalization**）

可以看到 Vgg-16 有 16 个隐藏层（13 个卷积层和 3 个全连接层），还有 Vgg-19 和 Vgg-64 版本。整个网络都使用了同样大小的卷积核（3x3）和 max pooling（2x2）。

论文探讨了一个 7×7 的卷积核与 3 个 3×3 的区别：增加感受野 + 减少参数量。后面很多论文都改用了 3×3 卷积核。

GoogleNet 是一个很深的网络，它通过 inception 结构堆叠起来，如下图所示：

里面有不同尺度的卷积核，也是第一次看到 1×1 的卷积核，这种操作有调整维度的效果，在 mobilenet 中，也可以看到。

Vgg 系列和 GoogleNet 给大家的感觉就是更深的网络，学习能力更强，实际由于梯度消失/爆炸问题，单纯的加深是不能很好提高学习能力的。

为了解决因网络深度增加而产生的性能下降问题，residual module 被提出来。**思路是:** 不是通过堆叠几层网络直接去拟合一个函数 H（x），而是拟合一个残差项，相当于把一个复杂的函数拆成了两项，这样学习起来就变得简单了。

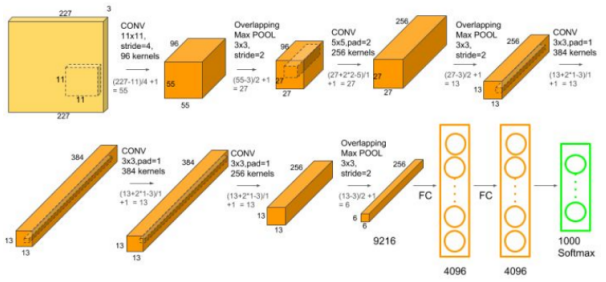


图 16: AlexNet

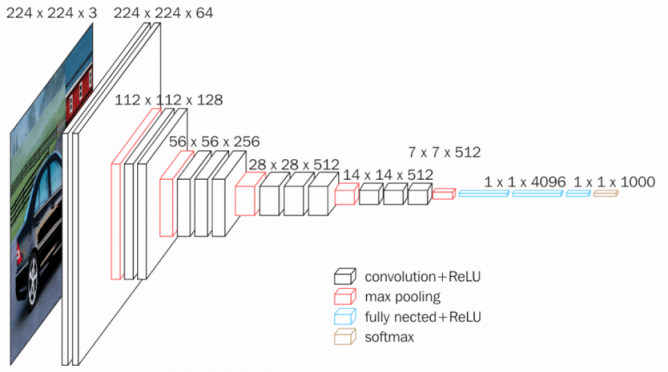


图 17: Caption

F(x)+x 能够很容易通过" shortcut connections" 来实现。可以看到 Inception 结构更关注网络的宽度，而 ResNet 关注的更多的是网络的深度。DenseNet 认为上述直接相加的方式会 ** 阻碍信息在网络的流动 **，所以很简单的实现就是通过一个模块将一段时间内所有的 feature map 直接拼接起来。

Bottleneck Layers

不断的拼接 feature map 就会导致后面通道数过多，参数过多。这里就可以用到 1×1 先降维度，在 3×3 卷积操作，就能维持参数的平稳增加。BN-ReLU-Conv1x1-BN-ReLU-Conv3x3

Batch Normalization 在神经网络训练开始前，我们都会对输入数据做一个归一化处理，其原因是训练过程本质就是为了学习数据分布的过程，一旦训练数据与测试数据的分布不同，那么网络的泛化能力也大大降低。

同样的道理，对于每一个 batch，如果他们的分布各不相同，那么网络

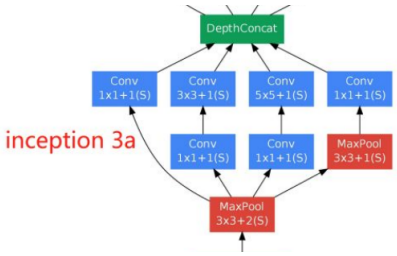


图 18: Vgg-16

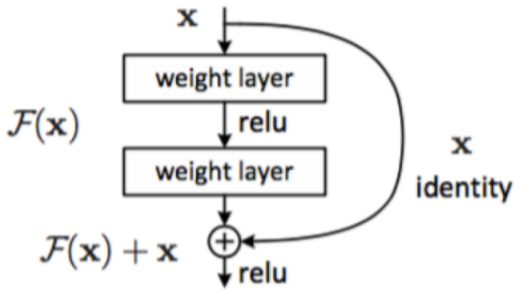


图 19: Caption

就要在每次迭代都去学习适应不同的分布，这样将会大大降低网络的训练速度。所以在网络每一层输入的时候，我们可以考虑插入 BN 层。基于相机的 RGB 影像可以提供环境中的结构信息和纹理信息，他们被广泛应用在计算机视觉和摄影测量的众多问题中。机器人和无人驾驶汽车利用相机传感器采集图像序列，在未知环境中的姿态估计和环境重建工作。

4.2 优化器与学习率

前面章节，我们描述了经典的网络结构。网络为什么能完成这些视觉任务呢？其实一个网络是有很多“未知数/待优化数”够成的，卷积层有卷积核，全连接层有权重与偏置，激活函数可以调整非线性程...。把输入信息经过网络前向传播，就可以得到网络预测的结果。如果这个结果离我们的预期有一些距离（loss），我们就把这个距离反向逐层的调整那些网络“未知数/待优化数”，以减少网络预测结果和实际结果之间的差距。这个反向调整网络参数的过程就是网络训练的过程。

面对这个复杂的优化过程，常见的深度学习框架（PyTorch/TensorFlow）

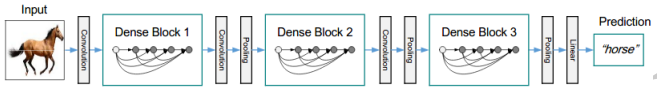


图 20: DenseNet

已经帮我们写好了求导的过程，我们需要在网络构建之处选择合适的优化器，并设置相关的参数就可以了。接下来，我们来看几种常见的优化器：

梯度下降 (Gradient Descent). 梯度是有所优化问题都会见到的，它是所有参数的偏导构成的一个向量，这个向量指引着极值方向。在 SLAM 中，我们同样有优化问题，不过由于 SLAM 存在稀疏性 [20]，所以我们有特殊的求解器。

随机梯度下降方法（SGD，Stochastic Gradient Descent）是梯度下降中常见的一种优化器。在参数更新时，该方法不需要计算整个样本的梯度，而是只随机选取其中一个样本计算梯度，因此这个方法训练有很快的训练速度。但是这种简单操作显然也有很多问题，包括：学习率的选择需要经过多次尝试，容易陷入极小值等。

为了减少震荡和避免局部最优，我们可以为 SGD 加上 Momentum。在梯度方向不变的维度上，加速参数更新；当梯度方向有所改变时，通过增加动量使参数更新速度变慢。

$$v_{t+1} = \mu * v_t + g_{t+1} \tag{4.1}$$

$$p_{t+1} = p_t - lr * v_{t+1} \tag{4.2}$$

这里 p，g，v 以及 μ 分别表示待更新的参数，梯度，下降速度和动量。

```
    if cfg.method == 'sgd':
        optimizer = torch.optim.SGD(filter(lambda p:
            p.requires_grad, parameters),lr=cfg.lr,
            momentum=0.9, weight_decay=cfg.
            weight_decay)
```

我们可以在优化的过程中让学习率逐渐下降，例如我们可以让每经历 5 个 epoch, 学习率就可以得到原来学习率的 80%。

作者-SLAM 中的几何与学习方法

```
    if epoch%5 == 0:
        for p in optimizer.param_groups:
            p['lr'] *= 0.8
    print(optimizer.state_dict()['param_groups'][0]['
        lr'])
```

学习率自适应优化算法 为优化器选择合适的学习率是非常困难的问题，因此学界提出很多自适应的优化器，如：Adam 算法、RMSProp 算法，以及 AdaDelta。

Adam (Adaptive Moment Estimation) [15] 是应用很广泛的方法，它有动量和自适应学习率两个特性，一般全局学习率设置为 $lr = 1e-4$.

```
    if cfg.method == 'adam':
        optimizer = torch.optim.Adam(filter(lambda p:
            p.requires_grad, parameters),lr=cfg.lr,
            weight_decay=cfg.weight_decay)

    elif cfg.method == 'rmsprop':
        optimizer = torch.optim.RMSprop(filter(lambda
            p: p.requires_grad, parameters),lr=cfg.lr
            , weight_decay=cfg.weight_decay)
    elif cfg.method == 'adadelta':
        optimizer =torch.optim.Adadelta(filter(lambda
            p: p.requires_grad, parameters),
        lr=cfg.lr,   weight_decay=cfg.weight_decay)
```

4.3 常用的度量方法

不同的学习问题，有不同的规则去衡量模型的效果。这里我们介绍几种在分类问题中常见的度量方式。我们通过一个例子，来介绍精度 True-Negative, False-Negative, True-Positive, False-Positive, 精确率（Precision），召回率（Recall），准确率（Accuracy）.

浙江SAM中的几何学习方法

对于二分类问题，这些统计概念都很好理解，但是对于多分类问题，有点模糊。在这个三分类问题中，我们分别有 class1，class2, class3，并获得网络的预测值，以及数据集的真实值。然后我们就可以统计出以下三张表：

		Predicted		
		class1	class2	class3
Actual	class1	852	83	268
	class2	74	850	273
	class3	216	337	505

图 21: 分类结果统计。

	TP	FN	FP
class1	852	83+268	74+216
class2	850	74+273	83+337
class3	505	337+216	268+273

图 22: 统计 TP，FN 和 FP 对应的信息。

	Precision (TP/(TP+FP))	Recall TP/(TP+FN)
Parallel	0.74	0.71
Perpendicular	0.66	0.71
non-relationship	0.48	0.48

图 23: 计算精确率和召回率。

盐粒-SL

5 基于卷积网络的学习任务

5.1 点线面的预测

5.1.1 预测特征点

GCNv2: Efficient Correspondence Prediction for Real-Time SLAM 2019 ral. 网络提取特征点，并提供描述子。实验来看，比 orb 能更好的应对快速运动。

5.1.2 预测特征线

Robust Line Segments Matching via Graph Convolution Networks. 刚出来的文章，建议搭建关注 graph，可能一大波论文正在袭来。这篇文章已经开源。

5.1.3 平面实例分割

5.2 物体检测与分割

物体检测（object detection）可能涉及到物体的识别、分割等。对于识别问题，也就是判断物体的种类，是一种分类问题。而物体的分割，有基于 bounding box 的分割，也有当下流行的实例分割（instance segmentation）。

5.2.1 分类学习基础

分类和回归被认为是两种基础的学习任务，在研究一个学习任务之前，我们都要思考，这个问题可以被理解成一种什么任务。对于分类问题，一般是对不同的物体进行概念上的归类，可以针对整个图片，也可以针对图片中的局部图像块。

手写数字识别是一个基本的分类问题，对于输入的数字图片，我们要通过网络来识别数字。我们从这个问题入手，来理解分类任务。

MNIST 数据集是一种识别手写数字常用的数据集，每张图片的大小是 28*28，并且每个图片都有对应的 ground truth label，即图片对应的真实数字。这种进点的数据集，已经被集成在大多数深度学习框架中了，如 PyTorch, TensorFolow，大家可以在工程中通过简单的方式直接下载使用。对于这种简单的任务，我们可以建立一种多层感知机（Multi-layer perception）来完成

视觉SLAM 中的几何与学习方法

图 24: Minist 数字手写体数据集

工作。下面的代码中可以看出，网络有 3 层全链接层。将图像展开成 1×784 维的向量，经过隐藏层，然后输出 10 个值。

```
class MLP(torch.nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = torch.nn.Linear(784,512)
        self.fc2 = torch.nn.Linear(512,128)
        self.fc3 = torch.nn.Linear(128,10)

    def forward(self,input):
        input = input.view(-1,28*28)
        dout = torch.nn.functional.relu(self.fc1(
            input))
        dout = torch.nn.functional.relu(self.fc2(
            dout))
        return self.fc3(dout)
model = MLP().cuda()
print(model)
```

基于上述的网络结构，我们就建立了数据之间的传递方式。除了数据传递的方式，我们还要明确优化器、学习率、损失函数等。

```
# loss func and optim
optimizer = torch.optim.SGD(model.parameters(),lr
    =0.01,momentum=0.9)
criterion = torch.nn.CrossEntropyLoss().cuda

for x in range(10):
```

```
        for i,data in enumerate(train_dataset):
    optimizer.zero_grad()
            (inputs,labels) = data
            inputs = torch.autograd.Variable(inputs).
                cuda()
            labels = torch.autograd.Variable(labels).
                cuda()

            outputs = model(inputs)

            loss = criterion(outputs,labels)
            loss.backward()

            optimizer.step()
```

MLP 是一种简单的网络结构，对于复杂的任务，可以通过在它的前面增加卷积层来提高网络的学习能力。

5.2.2 分类网络中常见策略

1.LeNet-5：每一个卷积核都会形成一个特征图，3 个通道则是每个通道是不同的卷积核，但是最后是将三通道的卷积值相加，最后变成一个通道，所以 5*5*64 的卷积核，感知范围是 5*5，最后出来是 64 层（个特征图）。每个卷积核都包括 w（权重）和 b(bias 偏置)。LeNet-5 最初用于手写数字识别。

2.AlexNet：5 个卷积层和 3 个全连接层最后输出层是 1000 类的 Softmax。

使用如下创新：

1）ReLU 非线性激活函数

2）多 GPU 训练

3）局部响应归一化

4）重叠池化

3. 降低过拟合的方法：

1）数据增强：对图像数据进行变换

基于SLAM中的几何与学习方法

2）Dropout 随机的关闭一定比例的节点（神经元）

4.VGGnet: 最后输出为 4096 维，全连接层是 1000 维（1000 类别）。最后一层是 Softmax 层

隐层使用 ReLU

5.GoogleNet:2014ILSVRC 分类任务冠军。22 层

创新点：用全局平均池化层取代全连接层，借鉴了 NIN（network in network）的做法

MLP 网络能够更好的拟合局部特征，也增强了输入局部的表达能力，NIN 不在分类层前使用全连接，而是采用全局平均池化。

提高深度网络效果的方式是：增大网络尺寸。但是带来了更多的参数和计算资源需求。

改善方案：

1）引入稀疏性

2）Hebbin 原则：两个神经元同步激发，则他们之间的权重增加，如果单独激发，则权重减少。

GoogleNet 就是利用 Inception 自动构建非一致结构的神经网络。

ResNet：残差网络 shortcut 一定程度上解决了过深模型梯度发散导致无法训练的问题。

DenseNet 从第一层开始每层都作为后面各层的输入。

Dual Path Network：双通道神经网络

5.3 深度图与法向量预测

5.3.1 监督学习

FCN [22] 是一篇无法避开的文章，他的 google scholar 引用已经达到 9600 左右。这是一个划时代的工作，他的出现为深度学习进入像素领域的操作实现了思想解放，大量的基于像素的语义分割的工作随之而来。回头来看 FCN 提出来的反卷积（deconvolution）操作是对特征图的上采样，实验效果也别后来的方法 [CRF][] 刷新，但是他对领域的贡献，已经反应在引用量上了。回到深度预测的话题，David et al. [4] 早在 2014 就提出使用深度学习做这项工作，他使用了非常基础的网络结构，如图一所示。框架分为两个部分：coarse net 和 fine net，前一个网络通过使用较大卷积核子和全链接层试图获得更加全局的结构（structure）信息，第二个网络完全使用卷积操作，并在把 coarseNet 的结果也作为一种 feature map 输入。这两个

深度SLAM中的几何与学习方法

网络也是分开训练的。在这个框架基础上，David et al. 提出了更加复杂的框架 [3], 来同时处理 depth prediction, Surface normals(SN) and semantic segmentation. CoarseNet 和 FineNet 经过少许变化之后，输入 Scale3 以获得更细致的结果。

另外，论文 [3] 在 [4] 深度估计的 Loss 基础上进行修改，直接使用深度之差（predicted and GT）代替对数化的差，同时考虑差在水平、竖直图像上的梯度。

作者表示，这种设计不仅可以使预测图像接近 GT，同时还有助于保持局部结构的相似。

除此之外，[3] 同样给出了 SN 和 semantic labels 的 LOSS 表示：对于 SN，作者计算每个像素处的点积（predicted and GT），并对整体图像求平均。对于 Semantic 部分，作者使用 softmax 分类器进行逐项素预测，不同 channels 之间使用 cross-entropy 给出最后预测。数据在监督学习中起着重要的作用,David 的工作是在 NYU-Depth 上训练的。Wang et al.[8] 提出了一种结构来进行 normal estimation. 不同的是, 他们考虑了环境特点（man-made, Manhattan World）和中间表述（房间布局等），并提出一种全局和局部结合的网络，并使用融合策略统一他们。[37] 提出了一种在 Physically-Based Rendering 的渲染方法，并利用 SUNCG [30] 产生很多 synthetic 数据, 作者使用这些数据去训练网络，并通过 NYU 数据做 finetuning 之后，效果更好。这种渲染方法被说成是以后实时渲染的趋势，我不太了解，还是回到数据集和模型。这篇文章的框架是比较简单，使用 VGG16 提取特征，然后是全卷积网络进行 normal 估计，Loss 则是沿用了 David Eigen 的设计，训练方法是 RMSprop。同时这个工作也涉及到目标边界检测，并取得不错的效果。接着这个工作，作者发表了为 RGB-D 数据进行深度补全（deep completion）的论文 [36]。论文首先是利用之前工作 [30]，通过单帧图像获得 SN 和物体边界（occlusion boundaries），然后通过一个优化函数，可以训练 SN，OB 与深度之间的关系。作者指出，对整体图像进行训练比仅训练洞处的像素效果更好。

同样使用 FCN 结构的还有，Iro [17] 的论文，今天还在 chair 遇见她。这是她和 Christian 完成的工作。Christian 已经去 VGG 做 PostDoc 了，接触比较少，但是很喜欢他的安静与真诚。他们的工作是使用 ResNet-50 来提取特征，并且获得了很好的效果。后来，Keisuke 和 Iro 连接了 Iro-Net and LSD-SLAM 一起做了 CNN-SLAM [32], 虽然网络可以很好的估计深度

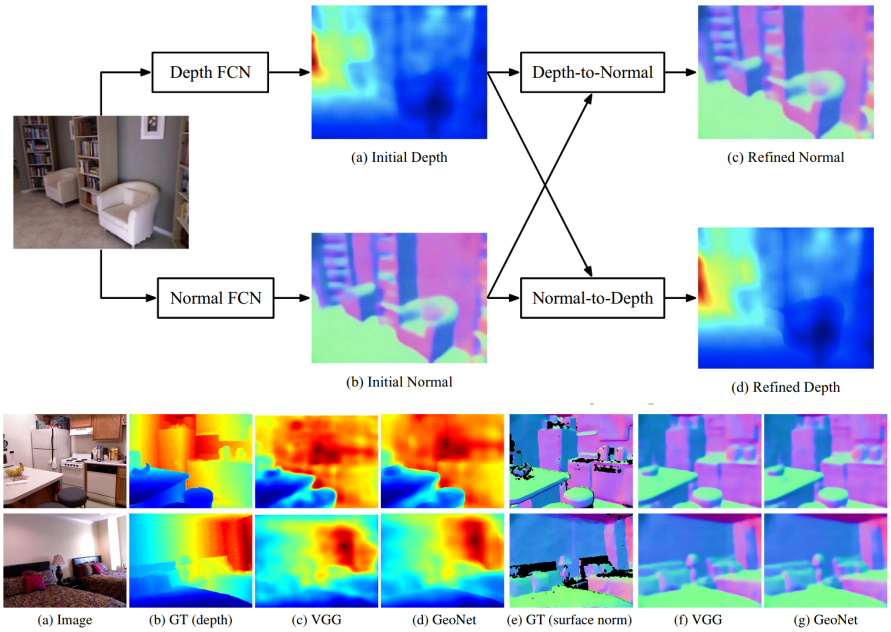


图 25: Geo-Net [27] 的网络框架与训练结果。

图，但是计算速度却达不到 real-time，因此 CNN-SLAM 只对 keyframe 做深度估计。

上面这些工作确实是比较久远了，网络结构没有那么好，显得比较笨重。我们来看一下比较轻量级的网络：mobileNet V2 [29].mobileNetV2 提供了一个语义分割的框架。论文指出在与其他网略表现相近的情况下，mobileNet V2 的速度和超参的数量有很大的优势。而原因主要是，该网络将标准卷积分解成一个深度卷积和一个点卷积（1*1)。对于实时性要求的 SLAM 领域，这个网络结构具有较大的意义。论文 [29] 指出，联合 DeepLabv3 和 MNetV2 feature map 可以较 MNetV1 减少参数，并保持相似表现。这篇文章没有估计深度图，但是网络结构可以有很好的借鉴意义。Geo-Net [27] 和 Deep ordinal [7] 是朋友推荐的两篇，第一篇是港中文 Prof. Jiaya Jia 的团队，第二篇是悉尼大学的 Prof. Dacheng Tao。GeoNet 考虑了 depth map 与 normals 之间的紧密关系，建立两个 steam 的网络结构。

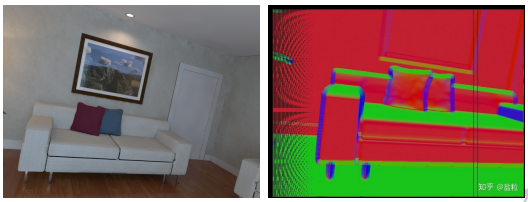


图 26: 制作数据集

5.3.2 动手训练一个法向量预测网络

在 David Eigen 的 depth prediction,surface normals predition 的论文网页中，给出了 NYU depth 的 normal 训练集，场景也比较丰富，不过数据量太少了。

制作数据集 这里是我们自己做的训练集合，使用 ICL 的深度图来制作自己的法向量数据集。

```
void Frame::EstimateSurfaceNormalGradient(const
    cv::Mat &imDepth, const double &timeStamp,cv::
    Mat &K)
{
    //ofstream surfaceNormal;
    //surfaceNormal.open("surfaceN.txt", ios::out
        | ios::trunc );
    //PointCloud::Ptr tmp( new PointCloud() );
    vector<SurfaceNormal> surfaceNormalVector;

    int cellsize=10;
    cv::Mat mPointCLoud=cv::Mat::zeros(imDepth.
        rows,imDepth.cols,CV_32FC3);//x,y,z

    for ( int m=0; m<imDepth.rows; m++ )
    {
        for ( int n=0; n<imDepth.cols; n++ )
        {
```

```
854                 float d = imDepth.ptr<float>(m)[n];
855     if(d<0.001){ continue;}                 mPointCLoud.at
856                 <cv::Vec3f>(m,n)[2]=d;
857
858                 mPointCLoud.at<cv::Vec3f>(m,n)[0]= (n
859                     - K.at<float>(0,2))*d / K.at<
860                     float>(0,0);
861
862                 mPointCLoud.at<cv::Vec3f>(m,n)[1]= (m
863                     - K.at<float>(1,2))*d / K.at<
864                     float>(1,1);
865             }
866         }
867
868         //读取中心点 和周围的四个点
869         //创建Mat
870         cv::Mat mTangeMask=cv::Mat::zeros(imDepth.
871             rows,imDepth.cols,CV_8U);
872         cv::Mat tangeMaskIntegralTemp;cv:: Mat
873             mUTangeMapIntegralTemp;Mat
874             mVTangeMapIntegralTemp;
875
876         cv::Mat tangeMaskIntegral;cv:: Mat
877             mUTangeMapIntegral;Mat mVTangeMapIntegral;
878
879         cv::Mat mUTangeMap=cv::Mat::zeros(imDepth.
880             rows,imDepth.cols,CV_32FC3);
881         cv::Mat mVTangeMap=cv::Mat::zeros(imDepth.
882             rows,imDepth.cols,CV_32FC3);
883         for ( int m=1; m<imDepth.rows-1; m++ )
884         {
885             for ( int n=1; n<imDepth.cols-1; n++ )
886             {
```

```
887                 //根据深度，获取有效点
888     }     }
889         //求intergral
890         integral(mTangeMask,tangeMaskIntegralTemp);
891         integral(mUTangeMap,mUTangeMapIntegralTemp,
892             CV_32FC3);
893 
894         integral(mVTangeMap,mVTangeMapIntegralTemp,
895             CV_32FC3);
896         //去掉积分图的padding
897 
898         tangeMaskIntegralTemp(cv::Range(1,imDepth.
899             rows-3), cv::Range(1,imDepth.cols-3)).
900             copyTo(tangeMaskIntegral);
901         mUTangeMapIntegralTemp(cv::Range(1,imDepth.
902             rows-3), cv::Range(1,imDepth.cols-3)).
903             copyTo(mUTangeMapIntegral);
904         mVTangeMapIntegralTemp(cv::Range(1,imDepth.
905             rows-3), cv::Range(1,imDepth.cols-3)).
906             copyTo(mVTangeMapIntegral);
907 
908         int cellCount=0;
909         for (int v=cellsize+1; v<imDepth.rows-3;v++
910             )
911         {
912         //处理每个cell
913         }
914         vSurfaceNormal=surfaceNormalVector;
915     }
916 
```

917 **完成一个神经网络** 其实我的工作不是一个语义分割的任务，由于目前正
918 在准备 IROS2019，如果后面能接受，我在详细的介绍。训练集和测试集是
919 相同 size 的图片，一个是 RGB，一个是 mask。

论文-SLAM 中的应用与学习方法

训练集是 RGB 图像和 mask, 把图像尺寸压缩成 160× 160 大小。

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456,
        0.406], std=[0.229, 0.224, 0.225])])

class BagDataset(Dataset):

    def __init__(self, transform=None):
        self.transform = transform

    def __len__(self):
        return len(os.listdir('bag_data'))

    def __getitem__(self, idx):
        img_name = os.listdir('bag_data')[idx]
        imgA = cv2.imread('bag_data/'+img_name)
        imgA = cv2.resize(imgA, (160, 160))
        imgB = cv2.imread('bag_data_msk/'+
            img_name, 0)
        imgB = cv2.resize(imgB, (160, 160))
        imgB = imgB/255
        imgB = imgB.astype('uint8')
        imgB = onehot(imgB, 2)
        imgB = imgB.transpose(2,0,1)
        imgB = torch.FloatTensor(imgB)
        #print(imgB.shape)
        if self.transform:
            imgA = self.transform(imgA)

        return imgA, imgB

bag = BagDataset(transform)
```


train_size在SIAM论文中的几何与学习方

```
    train_size = int(0.9 * len(bag))test_size = len(
bag) - train_size
    train_dataset, test_dataset = random_split(bag, [
        train_size, test_size])

    train_dataloader = DataLoader(train_dataset,
        batch_size=4, shuffle=True, num_workers=4)
    test_dataloader = DataLoader(test_dataset,
        batch_size=4, shuffle=True, num_workers=4)
```

在开始网络结构之前，还是要回顾一下 feature map 的尺寸是如何变化的，我们需要保持两段 size 的一致。

$$Out_{height} = \frac{Input_{height} - kernel + 2 \times padiing}{stride} + 1 \tag{5.1}$$

我们还是使用预训练好的 VGG16 做为卷积层，这里要输出五个 size 的 feature map, 用来反卷积回来。

```
    class VGG16(nn.Module):
        def __init__(self):
            super(VGG16, self).__init__()
            self.conv1_1 = nn.Conv2d(3, 64, 3,
                padding=100)
            self.relu1_1 = nn.ReLU(inplace=True)
            self.conv1_2 = nn.Conv2d(64, 64, 3,
                padding=1)
            self.relu1_2 = nn.ReLU(inplace=True)
            self.pool1 = nn.MaxPool2d(2, stride=2,
                ceil_mode=True) # 1/2

    class FCNs(nn.Module):

        def __init__(self, pretrained_net, n_class):
            super().__init__()
```

```
986             self.n_class = n_class
987         self.pretrained_net = pretrained_net        self.
988             relu    = nn.ReLU(inplace=True)
989             self.deconv1 = nn.ConvTranspose2d(512,
990                 512, kernel_size=3, stride=2, padding
991                 =1, dilation=1, output_padding=1)
992             self.bn1     = nn.BatchNorm2d(512)
993             self.deconv2 = nn.ConvTranspose2d(512,
994                 256, kernel_size=3, stride=2, padding
995                 =1, dilation=1, output_padding=1)
996             self.bn2     = nn.BatchNorm2d(256)
997             self.deconv3 = nn.ConvTranspose2d(256,
998                 128, kernel_size=3, stride=2, padding
999                 =1, dilation=1, output_padding=1)
1000            self.bn3     = nn.BatchNorm2d(128)
1001            self.deconv4 = nn.ConvTranspose2d(128,
1002                64, kernel_size=3, stride=2, padding
1003                =1, dilation=1, output_padding=1)
1004            self.bn4     = nn.BatchNorm2d(64)
1005            self.deconv5 = nn.ConvTranspose2d(64, 32,
1006                 kernel_size=3, stride=2, padding=1,
1007                dilation=1, output_padding=1)
1008            self.bn5     = nn.BatchNorm2d(32)
1009            self.classifier = nn.Conv2d(32, n_class,
1010             kernel_size=1)
1011            # classifier is 1x1 conv, to reduce
1012                channels from 32 to n_class
1013
1014        def forward(self, x):
1015            output = self.pretrained_net(x)
1016            x5 = output['x5']
1017            x4 = output['x4']
1018            x3 = output['x3']
```

X1 论SLAM定中的几何与学习方

```
            x2 = output['x2']
    x1 = output['x1']
            score = self.bn1(self.relu(self.deconv1(
                x5)))
            score = score + x4
            score = self.bn2(self.relu(self.deconv2(
                score)))
            score = score + x3
            score = self.bn3(self.relu(self.deconv3(
                score)))
            score = score + x2
            score = self.bn4(self.relu(self.deconv4(
                score)))
            score = score + x1
            score = self.bn5(self.relu(self.deconv5(
                score)))
            score = self.classifier(score)
            return score
```

```
    def train(epo_num=50, show_vgg_params=False):

        device = torch.device('cuda' if torch.cuda.
            is_available() else 'cpu')

        vgg_model = VGGNet(requires_grad=True,
            show_params=show_vgg_params)
        fcn_model = FCNs(pretrained_net=vgg_model,
            n_class=2)
        fcn_model = fcn_model.to(device)
        criterion = nn.BCELoss().to(device)
        optimizer = optim.SGD(fcn_model.parameters(),
            lr=1e-2, momentum=0.7)
```

```
        all_train_iter_loss = []
    all_test_iter_loss = []
        # start timing
        prev_time = datetime.now()
        for epo in range(epo_num):

            train_loss = 0
            fcn_model.train()
            for index, (bag, bag_msk) in enumerate(
                train_dataloader):

                bag = bag.to(device)
                bag_msk = bag_msk.to(device)

                optimizer.zero_grad()
                output = fcn_model(bag)
                output = torch.sigmoid(output) #
                    output.shape is torch.Size([4, 2,
                    160, 160])
                loss = criterion(output, bag_msk)
                loss.backward()
                iter_loss = loss.item()
                all_train_iter_loss.append(iter_loss)
                train_loss += iter_loss
                optimizer.step()

                output_np = output.cpu().detach().
                    numpy().copy() # output_np.shape =
                     (4, 2, 160, 160)
                output_np = np.argmin(output_np, axis
                    =1)
                bag_msk_np = bag_msk.cpu().detach().
                    numpy().copy() # bag_msk_np.shape
```

bag_msk_np.npz-SLAM实战中的几何与

```
                    = (4, 2, 160, 160)
    bag_msk_np = np.argmin(bag_msk_np, axis=1)
                if np.mod(index, 15) == 0:
                    print('epoch {}, {}/{},train loss
                        is {}'.format(epo, index, len
                        (train_dataloader), iter_loss)
                        )
                                # plt.subplot(1, 2, 1)
                # plt.imshow(np.squeeze(bag_msk_np[0,
                    ...]), 'gray')
                # plt.subplot(1, 2, 2)
                # plt.imshow(np.squeeze(output_np[0,
                    ...]), 'gray')
    # plt.pause(0.5)
```

5.3.3 无监督学习

monodepth 和 monodepthv2。

盖雅-SLAM 中的几何与学习方法

6 基于图网络的学习任务

图网络对于做 SLAM 研究的同学有着特殊的意义，因为我们很早就利用 graph 中的节点和边来表示相机（路标点）和观测，相比与 SLAM 优化中的图，这里介绍的图网络相同点，也有不同之处。一方面他们处理的数据结构是相同的，所谓数据结构就是任意可能的连接关系导致的非结构化数据。另一方便，不想 SLAM 中单一的节点形式，图网络中的节点由 deep feature 形成，并且可以表示任意的内容。由于图网络的资料不像卷积网络那么多，因此我们先花些功夫来介绍基础知识，然后再运用到具体的视觉任务中。

6.1 图网络简介

不同于图像数据（像素结构化排列），图（graph）是由节点与边构成的非结构化的数据形式，他们形态各异，是一种非结构化的数据。图网络的提出就是处理这样的数据结构。PyTorch geometric 是一个基于 pytorch 的图网络处理库，里面封装了处理图网络需要用到的基础结构。基于这个图网路

图 27: PyTorch 是基于 PyTorch 的图网络库

库，我们介绍相关的细节信息。

1. 输入数据 单个的图被描述为 torch_geometric.data.Data, 他有以下属性：

我们对上面 data 中的变量进行赋值，就可以创建一个简单的图网路：

```
import torch
from torch_geometric.data import Data

# 边的连接情况： 0->1  1->0  1->2  2->1
```

data.x (节点的特征)	shape: [num_nodes,num_node_features]
data.edge_index（图的连接情况）	shape: [2, num_edges]
data.edge_attr （边的特征）	shape: [num_edges, num_edge_features]
data.y	
data.pos	shape: [num_nodes,num_dimensions]

表 2: 图网络中的参数

```
edge_index = torch.tensor([[0, 1, 1, 2], [1, 0,
    2, 1]], dtype=torch.long)
# 这是每个节点的特征x = torch.tensor([[-1], [0],
[1]], dtype=torch.float)

# 构建Graph的数据
data = Data(x=x, edge_index=edge_index)
>>> Data(edge_index=[2, 4], x=[3, 1])
```

Data 常用属性 Data 类中已经为大家封装好了一些常用的属性，可以调用相应的函数获得你想要的信息：

```
print(data.keys)
>>> ['x', 'edge_index']

# 节点的特征
print(data['x'])
>>> tensor([[-1.0],
            [0.0],
            [1.0]])

for key, item in data:
    print("{} found in data".format(key))
>>> x found in data
>>> edge_index found in data
```

盐-SLAM 组内的几何与学习

```
'edge_attr' in data>>> False

data.num_nodes
>>> 3

data.num_edges
>>> 4

data.num_node_features
>>> 1

data.contains_isolated_nodes()
>>> False

data.contains_self_loops()
>>> False

data.is_directed()
>>> False

# Transfer data object to GPU.
device = torch.device('cuda')
data = data.to(device)
```

如何把点云数据转成图数据 ShapeNet dataset (包含了 17,000 3D 形状的点云和每个点被分成 16 类)，可以通过 torch$_g$*eometric.datasets*

```
from torch_geometric.datasets import ShapeNet

dataset = ShapeNet(root='/tmp/ShapeNet',
    categories=['Airplane'])
```

习题-SLAM书中的几何与学习方

```
    dataset[0]
>>> Data(pos=[2518, 3], y=[2518])
```

如何从点云转成 graph 呢? geometric 也提供了数据转换接口。

```
import torch_geometric.transforms as T
from torch_geometric.datasets import ShapeNet

# 产生最近邻方式产生 graph
dataset = ShapeNet(root='/tmp/ShapeNet',
    categories=['Airplane'],
                   pre_transform=T.KNNGraph(k=6)
                       )

dataset[0]
>>> Data(edge_index=[2, 15108], pos=[2518, 3], y
    =[2518])

# 使用随机增强出来graph
dataset = ShapeNet(root='/tmp/ShapeNet',
    categories=['Airplane'],
                   pre_transform=T.KNNGraph(k=6)
                       ,
                   transform=T.RandomTranslate
                       (0.01))

dataset[0]
>>> Data(edge_index=[2, 15108], pos=[2518, 3], y
    =[2518])
```

可以看到 Data 的形状就转过去了。

构建第一个图网络 前面在知乎 “SLAM 烹饪术” 上，我们介绍了基础的图网络概念，这里我们来实现一个两层 GCN，可以看到，我们直接调用库提供的 GCN 层就可以了，非常的便捷。在前向传输的过程中，我们为第一层

作者-SLAM 中的几何与学习方法

增加一个 ReLU 和 dropout，来增加网络的线性能力和泛化能力。

```
import torch
import torch.nn.functional as F
from torch_geometric.nn import GCNConv

class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = GCNConv(dataset.
            num_node_features, 16)
        self.conv2 = GCNConv(16, dataset.
            num_classes)

    def forward(self, data):
        # 数据data，获得x, edge_index
        x, edge_index = data.x, data.edge_index

        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, training=self.training)
        x = self.conv2(x, edge_index)

        return F.log_softmax(x, dim=1)
```

写好了网络结构，下面是训练代码的主体，和卷积网络一样，里面包含了优化器选择与设置等内容。

```
device = torch.device('cuda' if torch.cuda.
    is_available() else 'cpu')
model = Net().to(device)
data = dataset[0].to(device)
optimizer = torch.optim.Adam(model.parameters(),
    lr=0.01, weight_decay=5e-4)
```

for 在SIAM的中的几何与学习方

```
    model.train()
    for epoch in range(200):    optimizer.zero_grad()
        out = model(data)
        loss = F.nll_loss(out[data.train_mask], data.
            y[data.train_mask])
        loss.backward()
        optimizer.step()

    # 模型测试部分
    model.eval()
    _, pred = model(data).max(dim=1)
    correct = int(pred[data.test_mask].eq(data.y[data
        .test_mask]).sum().item())
    acc = correct / int(data.test_mask.sum())
    print('Accuracy: {:.4f}'.format(acc))
    >>> Accuracy: 0.8150
```

6.2 如何定义一个图网络层

在本节开讲之前，我们再回顾一下图网络的独特之处

> Generalizing the convolution operator to irregular domains is typically expressed as a neighborhood aggregation or message passing scheme.

那么如何实现邻近聚合和信息传递策略，就是本文讲述的内容。

$$x_i^k = r^k(x_i^{k-1}, \dot{\rho}^k(x_i^{k-1}, x_j^{k-1}, e_{ji})) \tag{6.1}$$

在 PyTorch Geometric 提供的 MessagePassing 基类中，已经实现了消息自动传递机制，作者只需要去定义少数的几个函数，如 message(), update(), ，以及信息聚合的策略，如 aggr="addquot;, aggr="meanquot; 或 aggr="maxquot;.

监控-SLAM 中的几何与学习方法

GCN 代码实现 下面我们来实现一个下面公式表示的 GCN 层，

$$x_i^{(k)} = \sum_{j \in N(i)\, i} \frac{1}{\sqrt{deg(i)} \cdot \sqrt{deg(j)}} (\Theta \cdot x_j^{(k-1)}) \tag{6.2}$$

这里 Θ 表示权重矩阵 (weight matrix)。上述公式可以清楚的看出相邻节点是如何传递到该节点的，

step1. 首先使用权重矩阵来处理 $x_j^{(k-1)}$，

step2. 然后消除 degree 的影响，

step3. 最后再相加。

```
import torch
from torch_geometric.nn import MessagePassing
from torch_geometric.utils import add_self_loops,
    degree

class GCNConv(MessagePassing):  ##继承自
    MessagePassing
    def __init__(self, in_channels, out_channels)
        :
        super(GCNConv, self).__init__(aggr='add')
            # 使用"Add"策略  step3
        self.lin = torch.nn.Linear(in_channels,
            out_channels)

    def forward(self, x, edge_index):
        # x 是输入graph输入, N个节点, 每节点大小
            in_channels: 形状 [N, in_channels]
        # edge_index 是连接关系, shape [2, E]

        # 自我连接, Add self-loops to the
            adjacency matrix.
        edge_index, _ = add_self_loops(edge_index
            , num_nodes=x.size(0))

        # Step 1: 线性转换这个 节点特征矩阵.
```

半监SLAM粒中的几何与学习方

```
            x = self.lin(x)
    # Step 2: 计算连接度, 并消除影响          row, col
            = edge_index
            deg = degree(col, x.size(0), dtype=x.
                dtype)
            deg_inv_sqrt = deg.pow(-0.5)
            norm = deg_inv_sqrt[row] * deg_inv_sqrt[
                col]

            # 搞定 Start propagating messages.
            return self.propagate(edge_index, x=x,
                norm=norm)

        def message(self, x_j, norm):
            # x_j has shape [E, out_channels]

            # Step 4: Normalize node features.
            return norm.view(-1, 1) * x_j
```

当我们调用 propagete() 函数时，系统会自动去调用 massage()，aggregate() 和 update() 函数。上面的 massage() 函数会把输出节点特征限制在 (-1,1) 之间。

上述过程就是一个完整的 GCN 信息传递和更新过程。这个简单的 GCN 可以直接在你的网络中调用。

```
conv = GCNConv(16, 32)
x = conv(x, edge_index)
```

边代码实现 我们还是先来看边卷积层的数学定义，

$$x_i^{(k)} = max_{j \in N(i)} h_\Theta(x_i^{(k-1)}, x_j^{(k-1)} - x_i^{(k-1)}) \tag{6.3}$$

这里 h_Θ 可以看作是一个 MLP。我们同样使用 messagepassing 类来写这个边卷积层。

from 北京-SIAT-张量的几何与学习方

```
import torch
from torch.nn import Sequential as Seq, Linear,
ReLUfrom torch_geometric.nn import MessagePassing

class EdgeConv(MessagePassing):
    def __init__(self, in_channels, out_channels)
        :
        super(EdgeConv, self).__init__(aggr='max
            ') #  "Max" aggregation策略.
        # 定义MLP
        self.mlp = Seq(Linear(2 * in_channels,
            out_channels),
                       ReLU(),
                       Linear(out_channels,
                           out_channels))

    def forward(self, x, edge_index):
        # x has shape [N, in_channels]
        # edge_index has shape [2, E]

        return self.propagate(edge_index, x=x)

    def message(self, x_i, x_j):
        # x_i has shape [E, in_channels]
        # x_j has shape [E, in_channels]

        tmp = torch.cat([x_i, x_j - x_i], dim=1)
            # tmp has shape [E, 2 * in_channels]
        return self.mlp(tmp)
```

在 message 函数中，我们使用 mlp () 函数处理获得的 tmp，这个 tmp 包含该节点本身的 feature，以及相邻节点的一个关系特征。

图数据-SLAM中的几何与学习方法

7 传感器模型

7.1 相机模型

简单来说，相机模型就是把环境中的东西转变成观察信息的过程。不同的相机种类也有不同的相机模型，我们在这里介绍最简单的小孔成像模型。

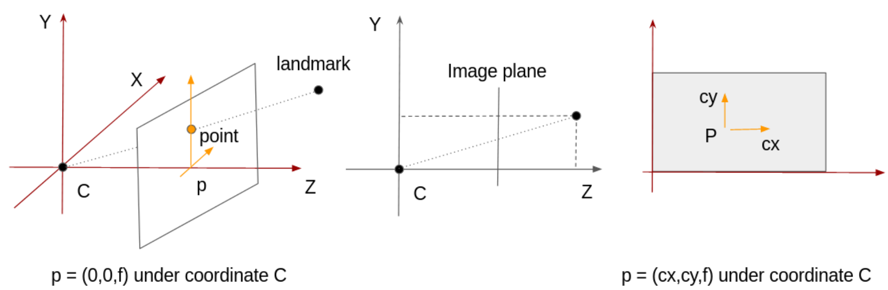


图 28: 小孔成像相机模型。

如图28所示，平面 $z = f$ 被成为是图像平面（image plane or focal plane）。把 landmark 点 $(X, Y, Z)^T$ 投影在图像平面上 (图28中)：

$$(X, Y, Z)^T -> (fX/Z, fY/Z)^T \tag{7.1}$$

由于图片中心的偏移，图片上投影点更加普遍的表示方法是 $(fX/Z + c_x, fY/Z + c_y)$。使用齐次坐标表示：

$$(X, Y, Z, 1)^T -> (fX + Zc_x, fY + Zc_y, Z)^T = \begin{bmatrix} f & s & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{7.2}$$

在上述公式中就可以看到相机的内参矩阵 K，其中 s 是偏斜参数（skew parameter）在大多数开源数据集中都是 0。当 landmark 的表达是在世界坐标系时，就可以通过刚体变化来转换坐标系。在图29中，我们使用向量关系，$\vec{wc} + \vec{cx} = \vec{wx}$，来描述 landmark 在不同坐标系下的转化关系。

$$X_c = R_{cw}(X_w - C_w) \tag{7.3}$$

X_w 和 C_w 表示世界坐标系下的 landmark 和相机像主点坐标。$X_w - C_w$ 可以表示向量 $C\vec{X}$ 在世界坐标系观点下的形式，这个向量在相机坐标系下的观测通过 R_{cw} 来转换。

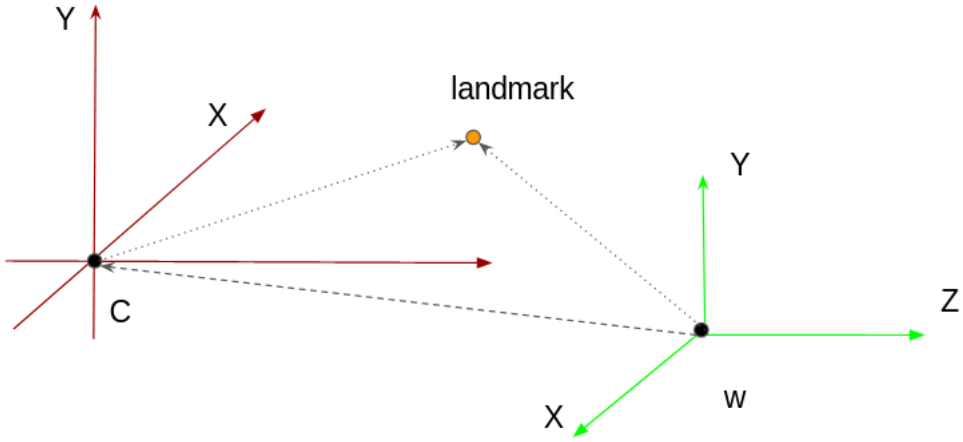


图 29: 路标点在不同坐标系下的表示。

因此，我们利用齐次坐标表示这个过程。

$$X_c = \begin{bmatrix} R & -R_{3\times3}C_{w3\times1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{7.4}$$

通过把上述公式与公式7.2联系起来，我们就可以获得世界坐标系到图像平面的过程。

$$x = KX_c = KR[I \quad |-C]X_w \tag{7.5}$$

因此，我们就构建了针孔相机模型 $P = KX_c = KR[I \quad |-C]$。

7.2 惯性导航模型

IMU（Inertial Measurement Unit）的中文名称是惯性测量单元，它由加速度计和陀螺仪这两部分组成，我们先抛开坐标系的干扰，可以简单的感受到加速度计提供的是加速度信号，而陀螺仪提供的是角速度信号。由此我们大概可以理解加速度随着时间积分成速度，然后获得位移运动，角速度则

视觉SLAM中的几何

在时间的作用下产生了旋转运动，而上述两者的结合貌似可以算出来携带 IMU 载体的六自由度位姿。

上述朦胧的感受暗示这个思路貌似可以实现，但是我们都知道，离开坐标系来谈位姿估计就是耍流氓。所以，我们进一步去看这种所谓的加速度和角速度是在哪里定义的。随着我们揭开这个系统输入信息的面纱，就能十分自然的过度到传统的捷联导航和预计分的核心算法。

相较于传统的相机传感器，IMU 是一种高帧率设备，在获得每一帧 IMU 数据的时候，加速度计 $a_{i,i+1}$ 获得值是从前一帧到当前帧的加速度信息。不仅如此，加速度同样受到了重力的影响。对习惯视觉 VO/SLAM 的同学来说，位姿估计一般是以第一帧图像的单位阵为世界坐标系，关系是发生在这个假定的世界坐标系中的，因此完全不用考虑和真实世界的关系。但是 IMU 中加速度计采集信息是受到重力影响的，因此这中加速度也被称之为**包含反向重力加速度的比力**。除了上述问题之外，传感器都存在噪声，噪声在长时间累积之后，同样或带来巨大的漂移问题。

陀螺仪和加速度计测量模型:

$$\begin{cases} \hat{w}_{wb}^{b}(t) = & w_{wb}^{b}(t) + b_g(t) + \theta_g(t) \\ \hat{a}^{b}(t) = & R_b^{wT}(a^w - g^w) + b_a(t) + \theta_a(t) \end{cases} \tag{7.6}$$

其中 $b_g()$ 和 $b_a()$ 是随时间变化的偏差，$\theta_g(t)$ 和 $\theta_a(t)$ 是白噪声项。

7.3 激光雷达模型

盐粒-SLAM 中的几何学习方法

8 位姿估计的模型

8.1 2D-2D 位姿估计

完成图片的匹配同时, 我们就在相邻的 RGB 图像之间获得了同名点的信息，一对同名点对应着 3D 空间中同一个路标点。图 30 可以看出, 图像的匹配结果中还是存在着大量的错误匹配线。如果使用这些错误匹配计算, 那么将使得后面图像的相对位姿估计出现错误。因此我们必须去除错误的匹配对，而常用的方法是 RANSAC 方式。

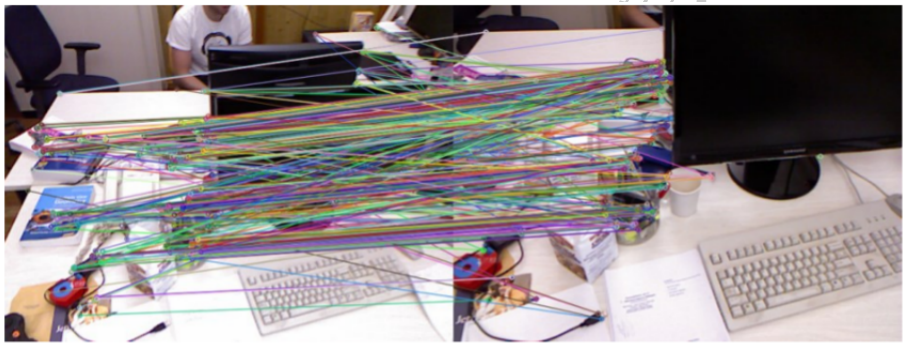


图 30: 彩色通道的特征提取与匹配

对于完成匹配的两张影像之间, 我们可以获得两张影像的相对运动情况, 如下图所示, 两张影像的相主点分别是 O_1 和 O_2，他们共同观测到空间中的点 $X = (x, y, z)$, 并且 X 在两张影像上的投影分别是 x_1，x_2。由于已经知道 x_1 与 x_2 的匹配关系以及相机之间存在刚体变换, 假设 O_2 通过旋转矩阵 R 和平移矩阵 t, 就可以构建起两张影像之间的关系。

基础矩阵 相机的内参矩阵是 K，对于 P_1 和 P_2，我们可以计算出 X 在两幅影像上的重投影点：$x_1 = KX$ 和 $x_2 = K(RX + t)$。因此，$K^{-1}x_2 = (RK^{-1}x_1 + t) = RK^{-1}x_1 + t$，等式两边同时与 t 左叉乘, 则

$$t \times K^{-1}x_2 = t \times RK^{-1}x_1 \tag{8.1}$$

把 $t\times$ 记做 $t_{[]}$, 则公式变形为 $t_{[]}K^{-1}x_2 = t_{[]}RK^{-1}x_1$, 同时两边再点成 $(K^{-1}x_2)^T$,

$$x_2^T K^{-T} t_{[]} R K^{-1} x_1 = 0 \tag{8.2}$$

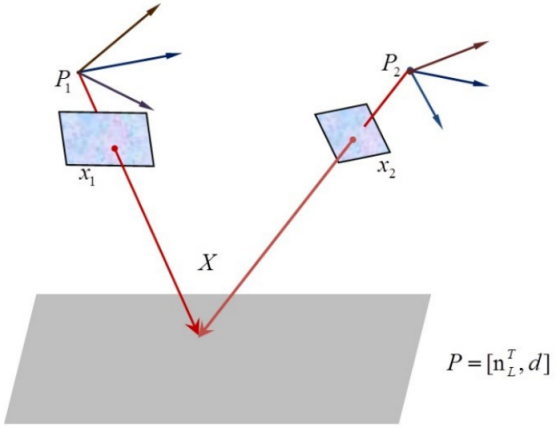


图 31: 空间点属于平面区域

上面的公式构建起了一对同名点的关系。我们把 $K^{-T}t_{[]}RK^{-1}$ 称为基础矩阵 F，F 矩阵的大小是 3×3。

$$x_2^T F x_1 = 0 \tag{8.3}$$

通常情况下,F 矩阵中的包含 8 个独立参数，因此至少八个同名点对就可完成 F 矩阵的求解，如果特征点对的数量多于 8，我们就可以通过奇异值 (SVD) 分解的方法获得最优的旋转矩阵 R 和平移矩阵 t。

单应矩阵 如图31所示，如果 x_1 和 x_2 对应的路标点落在平面区域，那么他们之间的变化关系将会退化成单应矩阵：

$$x_1 = H_\pi x_2 \tag{8.4}$$

通过矩阵变化，我们构建起了最初的两张影像的位姿变化关系，同时可以把两个相机的共同观测作为系统的局部地图维护起来，当接下来的图像进入系统时，可以通过新图像与局部点云之间的转换关系，进行新的图像帧的位姿判定的条件。这样就会大大增加系统对于场景的鲁棒性。

8.2 3D-2D 位姿估计

局部地图中的路标点具有世界坐标系，他们是最近多帧图像所观察的一个场景。由于机器人运动的连续性，当前帧和最近多帧具有较多的相同观测

信息, 因此, 我们可以通过局部地图构建一个 3D 点云和 2D 图像之间的关系。同时确定的关键帧可以为局部地图带来新的路标点, 实现局部地图的更新。

PnP(Perspective-n-Point) 算法就是一个常用来解算 3D 和 2D 关系的方法, 这个算法不需要满足上面的对极约束, 可以通过点云和图像点的齐次坐标之间的关系来获得旋转矩阵 R 和平移矩阵 t。

对于 PnP 算法的解法, 我们选择是用直接线性变换进行求解。我们首先来推导 3D 点云和 2D 图像特征点的关联过程，然后获得求解关系。公式8.5中使用齐次坐标系表示,

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K \begin{pmatrix} \hat{X}_c \\ \hat{Y}_c \\ 1 \end{pmatrix} \tag{8.5}$$

这里 $(\hat{X}_c, \hat{Y}_c, 1) = (X_c/Z_c, Y_c/Z_c, 1)$，其中 (X_c, Y_c, Z_c) 是 3D 点 P 在相机 C 坐标系下的坐标。K 是相机的内参。

由于局部点云中提供的 3D 点的坐标是在世界坐标系下，他们和当前相机坐标系下的关系正是由旋转 R_{cw} 和平移矩阵 t_{cw} 构成,cw 则表示从世界坐标系 w 到相机坐标系 c。

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = R_{cw} \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + t_{cw} \tag{8.6}$$

因此我们就可以构造出当前帧上的 2D 特征点 $(u, v\ 1)$ 与世界坐标系下对应的 3D 路标点 $(X_w, Y_w, Z_w\ 1)$ 的关系。这个关系由相机的旋转矩阵、平移矩阵和相机内参构成，它可以表示为 $A = \begin{bmatrix} t_1 & t_2 & t_3 & t_4 \\ t_5 & t_6 & t_7 & t_8 \\ t_9 & t_{10} & t_{11} & t_{12} \end{bmatrix} = \begin{bmatrix} A_1^T \\ A_2^T \\ A_3^T \end{bmatrix}$。在相机内参无法获得的情况下，我们可以通过求解 12 个未知参数获得，如果提供了相机的内参矩阵，未知数的个数将变成 6 个。

$$s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{bmatrix} t_1 & t_2 & t_3 & t_4 \\ t_5 & t_6 & t_7 & t_8 \\ t_9 & t_{10} & t_{11} & t_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{pmatrix} A_1^T \\ A_2^T \\ A_3^T \end{pmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{8.7}$$

盗版SLAM中的视觉与学习方法

其中 s 表示尺度信息，$s = r_7X + r_8Y + r_9Z + t_3 = A_3^T P$, 故,

$$\begin{aligned} A_1^T P - A_3^T P u = 0 \\ A_2^T P - A_3^T P v = 0 \end{aligned} \tag{8.8}$$

上面的公式 8.8可知, 每一对 3D-2D 关系，可以获得两个方程, 因此我们至少需要六个关系即可获得 12 个方程, 并求解出来 R,t 的关系。与对极几何中的 F 矩阵求解类似, 我们在构成超定方程时, 可以使用 SVD 分解获得最优的 R 和 t 矩阵。

在 OpenCV 中提供了不同的 PnP 求解的函数，对于 P3P，EPnP, 以及需要使用 LM 方法优化 3D-2D 之间的最小投影误差等问题，可以使用 solvePnP 这个函数。这里 ojectPoints 是 3D 点云数组，imagePoints 是与之对应的 2D 图像像素点数组，cameraMatrix 是相机内参矩阵，distCoeffs 表示相机畸变参数（空输入默认为畸变为零畸变)。

```
C++: bool solvePnP(InputArray objectPoints,
    InputArray imagePoints, InputArray
    cameraMatrix, InputArray distCoeffs,
    OutputArray rvec, OutputArray tvec, bool
    useExtrinsicGuess=false, int flags=ITERATIVE )

Python: cv2.solvePnP(objectPoints, imagePoints,
    cameraMatrix, distCoeffs[, rvec[, tvec[,
    useExtrinsicGuess[, flags]]]]) → retval, rvec,
     tvec
```

除此之外, OpenCV 还提供基于 RANSAC 的 PnP 函数 solvePnPRansac, 这个方法对含有外点 (outliers) 的输入数据更加鲁棒。可以看到除了 slovePnP 方法中的参数以外，还有 iterationsCount 表示迭代的次数，reprojectionError 表示判断外点的阈值，minInliersCount 表示当内点数超过这个数是，就认为结果足够可靠，进而结束迭代。

```
C++: void solvePnPRansac(InputArray objectPoints,
     InputArray imagePoints, InputArray
```


Python视觉SLAM教中的几何学习方

```
        cameraMatrix, InputArray distCoeffs,
        OutputArray rvec, OutputArray tvec, bool
        useExtrinsicGuess=false, int iterationsCount
        =100, float reprojectionError=8.0, int
        minInliersCount=100, OutputArray inliers=
        noArray(), int flags=ITERATIVE )
    Python: cv2.solvePnPRansac(objectPoints,
    imagePoints, cameraMatrix, distCoeffs[, rvec[, tvec[, useExtrinsicGuess[,
    reprojectionError[, minInliersCount[, inliers[, flags]]]]]]]) → rvec, tv
```

此时我们就可以获得当前帧的位姿关系, 并将当前帧存储进入轨迹图, 并且更新局部地图和全局地图。局部地图的维护主要在于控制路标点的数量, 在相对位姿信息的确定过程中, 随着局部地图的增大而增大。因此, 我们在局部地图中加入新的路标点的同时, 必须剔除过时的路标点, 以保持局部地图的较轻规模。

在有些里程计和 SLAM 系统中, 不仅涉及关键帧的选取同时也有关键帧的剔除功能, 在确定关键帧之后, 如果后面遇到更合适的关键帧, 我们可以把冗余的关键帧进行踢出, 这样可以进一步减少系统的压力。限于篇幅, 本文对于关键帧的剔除工作不做进一步的介绍。

8.3 3D-3D 位姿估计

8.4 位姿解耦估计方法

前面的方法都是同时计算出来 6D 的位姿中旋转和平移信息。有些研究者称，位姿估计中的漂移问题主要是来自旋转漂移，这里具体的原因，感兴趣的同学可以去尝试证明一下，应该是很有意义的一件事。

位姿解耦算法一般是将旋转矩阵和平移矩阵分开来解算，并提供一个更加精确的旋转矩阵。如何能提供更加准确的旋转矩阵呢？在结构化场景中，如室内场景、走廊等人造环境，存在特殊的几何现象：平行与垂直的直线和平面。这种环境被归结成曼哈顿世界（Manhattan World）假设和亚特兰大世界（Atlanta World）假设。

[12–14] 提出了 RGB-D 相机下的解偶位姿估计的框架，利用深度图计算出来平面的法向量，然后使用 mean-shift 聚类方法，找出来环境中正交平面，然后去追踪正交平面与正交平面之间的旋转关系。由于每次计算旋

西北工业大学 NPU 中国人的学习方法

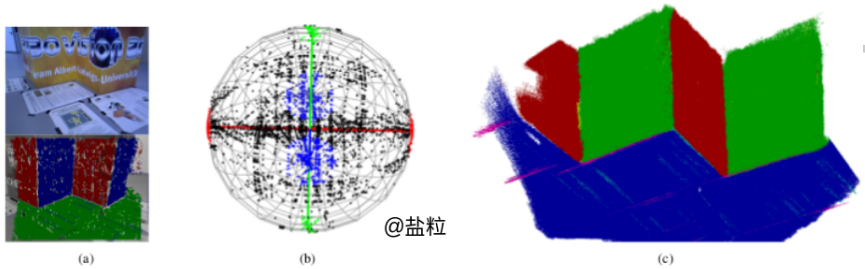


图 32: 曼哈顿假设下的定位与重建。

转，都是和第一次看到的曼哈顿世界相比较，因此从理论上说，这是一种没有漂移（drift-free）的旋转矩阵估计方式。没有漂移并不是意味着，没有误差，再每次 rotation 估计中依然会存在误差，只是这种误差不会积累起来了。

8.4.1 曼哈顿和亚特兰大假设

曼哈顿世界是一种存在正交几何信息的一种特殊环境，如常见的室内环境 (如图32 (a) 所示), 墙面之间满足垂直关系，前面与地板和天花板之间同样满足垂直关系。在曼哈顿世界中，这样的一个正交结构是全局环境中的主要结构。与曼哈顿不同，亚特兰大世界中包含多个曼哈顿世界的场景，即某一个局部满足单一的曼哈顿世界，然而全局包含很多个不同的曼哈顿世界场景，但是这些场景满足相同的重力方向。

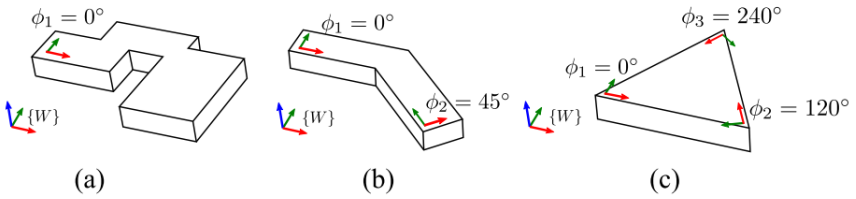


图 33: 环境示意图 [39]。(a) 曼哈顿世界假设; (b) 和 (c) 是亚特兰大世界假设。

8.4.2 基于特殊环境结构的无漂旋转估计

前面我们已经介绍了两种常见的环境，在这样的环境运行时，SLAM 系统可以利用环境中特殊的几何关系来完成相机的追踪和环境重建。

通过什么样的几何元素才能展示这种特殊的环境呢？当然是直线与平面信息了。对于不同的传感器，我们能使用的信息也受到相应的限制。对于单目或双面传感器，比较直接的方式就是使用直线和灭点来呈现正交信息，而 RGB-D 相机，我们就可以同时使用直线和平面信息。下面我们针对 RGB-D 输入，介绍在曼哈顿假设下的无漂旋转矩阵的估计算法。前面章节已经介绍了直线和平面的提取方法，以及法向量和灭点的计算方法，这里我们直接使用相关的信息。

首先我们建立一个曼哈顿坐标系 M（如图32所示，并计算当前帧和曼哈顿坐标系的相对旋转矩阵 R_{C_1M}，表示从 Manhattan coordinate M 到第一帧 C_1。我们可以假设 R_{C_1M} 初始为单位矩阵，然后通过迭代的方式逐渐优化出来结果。

如图32(b)，首先把 C_1 帧对应的法向量 v_k^s 投影到一个单位高斯球上 [38] [14]，这个高斯球也同时对应这 R_{C_1M} 的三个方向的正交轴 x, y, z。然后把单位法向量投影到每个轴 r_n 对应的切平面上。为了移除噪声，只取切平面上靠近轴心的圆形区域，并获得 $v_{kn}^{s'}$ 和 $v_{kn}^{d'}$。

随后，在切平面上使用 mean-shift 聚类方法，

$$s_n^{'} = \frac{\sum_{in} e^{-c\|m_{in}^{'}\|^2} m_{in}^{'}}{\sum_{in} e^{-c\|m_{in}^{'}\|^2}} \tag{8.9}$$

上述公式中 c 是调整高斯核宽度的参数。完成聚类之后，我们就在切平面上获得了 r_n 对应的 2D 中心点 s_n。

$$\hat{r_n} = Qs_n \tag{8.10}$$

这里 $Q = [r_{mod(n+1,3)}, r_{mod(n+2,3)}, r_{mod(n,3)}]$,$mod()$ 是取模函数。经过迭代，我们可以获得合理的旋转矩阵 $R_{C_1M} = [\hat{r_1}, \hat{r_2}, \hat{r_3}]$。

由于每一帧都是按照上述类似的方式进行旋转估计，这样第 i 帧的旋转矩阵的估计方式，就不再依赖前一帧或前面的关键帧，而是直接估计 R_{C_iM}，因此就实现了漂移的阻断，完成了 SLAM 中常见的漂移问题。估计出来旋转矩阵之后，我们还是可以使用之前的方式估计相机的平移矩阵。

视觉SLAM中的几何学方法

8.5 ORB-SLAM2 中的位姿策略

如上图所示，系统的初始化是在这个函数中完成，初始化根据传感器的不同，分为单目和 stereo 两种方式，其中将 RGB-D 转化出来双目信息。

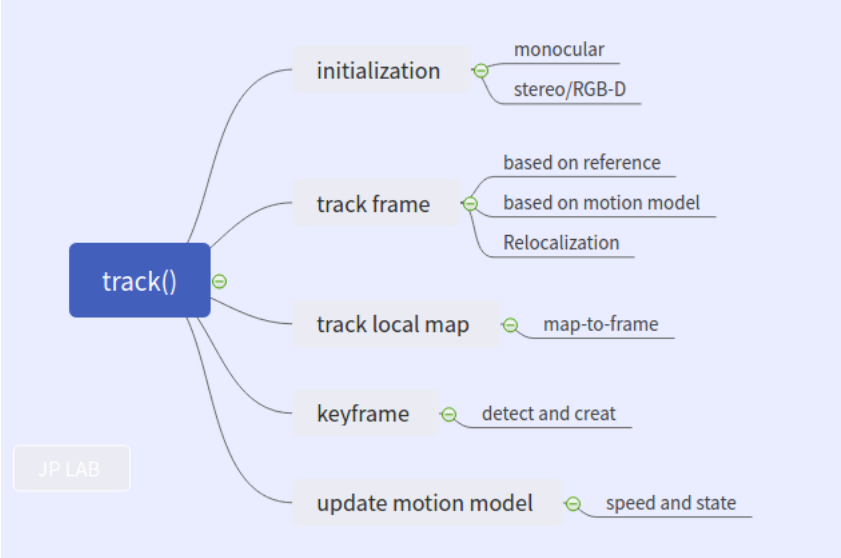


图 34: ORB-SLAM2 中用到的位姿估计模型

MonocularInitialization();

StereoInitialization(); 大家都知道，单目系统为了构建一个稳定的地图，有比较复杂的初始化过程。双目则是直接启动。后面我们会专门介绍单目初始化模块。初始化结束之后，会建立关键帧，全局地图。

然后会进行 frame-to-frame, map-to-frame 的位姿估计，当然如果丢失，则会进行 relocalization。

完成对的当前帧的 track, 就会建立起来当填充前帧中 2D 特征对应的 3D landmark。然后去判断是否要加入关键帧。并对追踪状态进行更新。下面，详细介绍追踪的过程。

其实，ORB-SLAM2 中的追踪策略是非常值得我们借鉴的。首先是给当前帧一个初始的位姿。通过 track reference 或 track motion model 来获得用重投影误差去优化当前帧的 Pose。每个结构里面，按照当前系统状态是否 OK，分成下面三种情况：

OK：

速度值不存在，或者初始化后的一帧，或者重定位后的一帧，使用参考帧进行追踪：bOK = TrackReferenceKeyFrame();

在追踪正常的情况下，存在运动的速度值，使用运动模型进行追踪：TrackWithMotionModel();

! OK：

Relocalization()

```
    bool Tracking::TrackReferenceKeyFrame()
{
    // Compute Bag of Words vector
    cout<<"track reference keyframe"<<endl;
    mCurrentFrame.ComputeBoW();
    ORBmatcher matcher(0.7,true);
    vector<MapPoint*> vpMapPointMatches;
//找到 vpMapPointMatches, nmatches
    int nmatches = matcher.SearchByBoW(
        mpReferenceKF,mCurrentFrame,
        vpMapPointMatches);

    if(nmatches<15)
        return false;
    cout<<"存在足够匹配点"<<endl;
    mCurrentFrame.mvpMapPoints = vpMapPointMatches
        ;
    mCurrentFrame.SetPose(mLastFrame.mTcw);

    // 通过优化3D-2D的重投影误差来获得位姿
    Optimizer::PoseOptimization(&mCurrentFrame);

    cout<<" Discard outliers"<<endl;
    // 剔除优化后的outlier匹配点（MapPoints）
    int nmatchesMap = 0;
    for(int i =0; i<mCurrentFrame.N; i++)
    {
```

```
            if(mCurrentFrame.mvpMapPoints[i])
    {           if(mCurrentFrame.mvbOutlier[i])
                {
                    MapPoint* pMP = mCurrentFrame.
                        mvpMapPoints[i];

                    mCurrentFrame.mvpMapPoints[i]=
                        static_cast<MapPoint*>(NULL);
                    mCurrentFrame.mvbOutlier[i]=false;
                    pMP->mbTrackInView = false;
                    pMP->mnLastFrameSeen =
                        mCurrentFrame.mnId;
                    nmatches--;
                }
                else if(mCurrentFrame.mvpMapPoints[i
                    ]->Observations()>0)
                    nmatchesMap++;
            }
        }

        return nmatchesMap>=10;
    }
```

tracking 成功后接下来就开始调用 TrackLocalMap() 函数来优化位姿了。

TrackLocalMap() 包括下面几个函数：UpdateLocalMap()：更新局部地图中的 keyframes 和 Local Points searchLocalPoints()：获得局部地图与当前帧的匹配 PoseOptimization()：最小化重投影误差来优化位姿

8.6 VIO 中的位姿估计

随着硬件设备的发展，VIO 相机定位系统也被大家熟知，其全称是 Visual-Inertial Odometry, 即视觉惯性里程计。由于 IMU 的加入，让系统在图像质量不佳，甚至是无图像状态下仍然能够进行一段时间稳定的位姿

视觉-SLAM 中的几何与学习方法

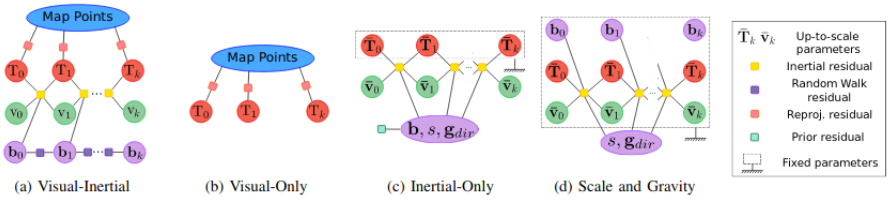


图 35: 不同优化模型中的因子图表示 [from ORBSLAM3 [2]].

估计。在 VIO 中一般有是世界坐标系、相机坐标系和 IMU 坐标系，其中 IMU 坐标系可以通过一个刚体变变化成相机坐标系。VIO 系统初始化的程度对后续的定位精度有重要的影响，因此我们首先来看如何将视觉部分和 IMU 部分通过初始化过程耦合起来。

如图 35所示，不同的追踪系统建立不同的因子图，可以看到 Visual-inertial 系统上部分就是常见的视觉图结构，这部分通过重投影误差来连接路标点和关键帧，不同的是（a）中还有 inertial residual 和 walk residual。然后，我们接下来的目标就是构建上述残差项。IMU 速度、偏差和相机位姿构成了惯性残差，预计分中的偏差构成了行动残差。这些量就是我们要在后续优化过程中计算的量了。

8.6.1 初始化

整个初始化部分分为视觉地图初始化、IMU 中的参数初始化，以及传感器耦合三个阶段。基于视觉的地图初始化部分，我们在本章前面的几节已经对不同传感器下的位姿估计和地图构建做了详细的介绍。因此，此处我们假设地图和刚开始的几帧已经完成了估计，我们直接进行 IMU 的初始化工作。

一、**角速度偏差的估计**。我们在前面的章节介绍的 IMU 的观测方程，在角速度和加速度的观测信息包含了偏差（bias），因此我们从观测信息中把偏差去掉之后，就会更加接近真实值。连续的两帧图像在世界坐标系下的旋转矩阵为 $R_{w,i}$ 和 $R_{w,i+1}$，因此两帧之间的相对旋转运动就可以获得 $R_{i,i+1} = R_{w,i}^T * R_{w,i+1}$。于此同时，通过预积分我们同样可获得两帧之间的旋转运动 ΔR_{imu}，我们可以通过两个不同模型估计出来的同一个旋转运动

盗版SLAM中的IMU学习

来描述角速度的偏差。

$$R_{i,i+1} = \Delta R_{imu} \begin{bmatrix} 1 \\ \frac{1}{2} J_{b_{wi}}^{\Delta R_{imu}} \Delta b_{wi} \end{bmatrix} \tag{8.11}$$

在初始化过程中，我们有 m 帧图像，因此联立所有相邻图像帧便得到一系列关于角速度偏差的方程，并且可以实现求解。

二、**速度，重力估计**。除了旋转矩阵构成的约束之外，图像和 IMU 之间还有相同的平移矩阵的约束，这个约束用估计每一个图像帧的速度，重力参数 g 和尺度 s。在 VINS-Mono 中，

$$st_{wb} = st_{wc} + R_{wc} t_{cb} \tag{8.12}$$

利用 IMU 预积分理论，我们可以获得两帧之间的运动速度和位姿表达式，

$$\Delta V_{i,i+1} = R_{wb_i}^T (R_{wb_{i+1}} - R_{wbi} V_i + g\Delta t) \tag{8.13}$$

$$\Delta t_{i,i+1} = R_{wb_i}^T (t_{wb_{i+1}} - t_{wbi} - R_{wbk} V_k \Delta t + \frac{1}{2} g \Delta t^2) \tag{8.14}$$

我们把 IMU 和相机之间平移关系（公式 8.12）和 IMU 两帧相对平移联系起来就可以获得：

$$\Delta t_{i,i+1} = R_{wb_i}^T (s(t_{wc_{k+1}} - t_{wck}) - (R_{wc_{k+1}} - R_{wbk}) t_{bc} - R_{wbk} V_k \Delta t + \frac{1}{2} g \Delta t^2) \tag{8.15}$$

方程中还剩余的未知量 χ_I 是每个帧位的速度，以及统一的重力和尺度，

$$\chi_I = [v_{b0}^{b0}, v_{b1}^{b1}, \ldots, v_{bi}^{bi}, g^{c0}, s] \tag{8.16}$$

然后，连立公式 8.12和8.15，我们构建起来

$$H_{b_{k+1}^{b_k}} \chi_I = z_{b_{k+1}^{b_k}} \tag{8.17}$$

其中 $H_{b_{k+1}^{b_k}}$ 可以表示为：

$$H_{b_{k+1}^{b_k}} = \begin{bmatrix} -I\Delta t_k & 0 & \frac{1}{2} R_{c0}^{bk} & R_{c0}^{bk} (t_{ck+1}^{c0} - t_{ck}^{c0}) \\ -I & R_{c0}^{bk} R_{bk+1}^{c0} & R_{c0}^{bk} \Delta t_k & 0 \end{bmatrix} \tag{8.18}$$

$$z_{bk+1}^{bk} = \begin{bmatrix} \alpha_{bk+1}^{bk} - t_c^b + R_{c0}^{bk} R_{bk+1}^{c0} t_c^b \\ \beta_{bk+1}^{bk} \end{bmatrix}. \tag{8.19}$$

盲源-SLAM 中的几何与学习方法

9 地图创建

前面章节介绍了传感器位姿的估计，这里我们介绍 SLAM 另外一个目标-地图重建。对于未知环境的重建，是机器和环境进行交互的基础。在常见的 SLAM 系统中，地图主要分为稀疏点云、半稠密点云和稠密点云。

9.1 稀疏地图重建

稀疏点云是指使用一部分特殊几何信息来描述 3D 环境，一般来说这些稀疏点是梯度变化比较明显的部分。我们先讲解如何利用特征法来进行重建，然后再讨论直接法。

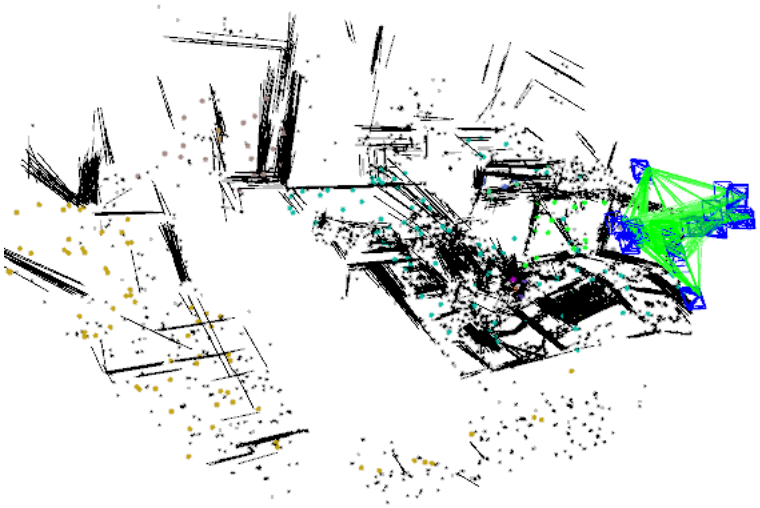


图 36: 点线面构成的稀疏点云。

9.1.1 稀疏点重建

特征法和直接法都可以获得稀疏点特征。特征法是从图片中提取特征点、线和面特征，并利用特征信息来计算相机位姿和重建环境信息。前面章节介绍了基本的特征提取与匹配知识，读者可以根据自己的需求选择相应的特征，因此这里我们直接使用这些特征来完成重建。

三角化 如图37所示，在两幅图像上分别提取同名特征点 x_1 和 x_2，而他们都是对空间路标点 X 的观测信息。我们的目标就是利用 2D 特征点来恢复空间 3D 路标点，众多的路标点则构成了稀疏点云。现在我们已经获得了同

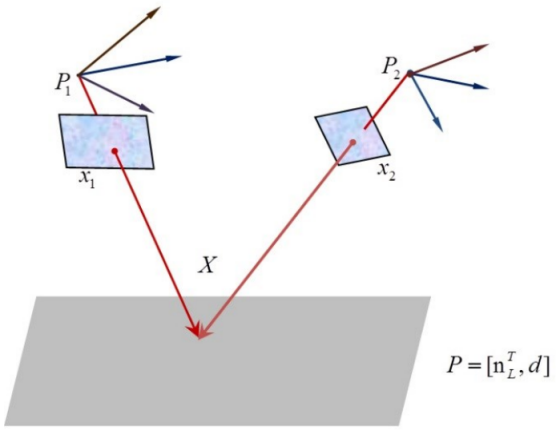


图 37: 特征点的重建

名点 $x_1 = (u_1, v_1)$ 和 $x_2 = (u_2, v_2)$，接下来就是通过简单线性三角化方法来重建稀疏点 X。

$$\begin{aligned} x_1 &= P_1 X_w \\ x_2 &= P_2 X_w \end{aligned} \tag{9.1}$$

这里 P_1 和 P_2 是两个相机对应的矩阵。我们构成一个矩阵 A，并且 $AX_w = 0$。

$$AX_w = \begin{bmatrix} x_1 \times P_1 X_w \\ x_2 \times P_2 X_w \end{bmatrix} \tag{9.2}$$

通过对矩阵 A 进行 SVD 分解，可以获得 3D 点的坐标。

```
cv::Mat A(4,4,CV_32F);
A.row(0) = kp1.pt.x*P1.row(2)-P1.row(0);
A.row(1) = kp1.pt.y*P1.row(2)-P1.row(1);
A.row(2) = kp2.pt.x*P2.row(2)-P2.row(0);
A.row(3) = kp2.pt.y*P2.row(2)-P2.row(1);
cv::Mat u,w,vt;
```

3D视觉SLAM学习方案 中的，

```
    cv::SVD::compute(A,w,u,vt,cv::SVD::MODIFY_A| cv::
        SVD::FULL_UV);
    x3D = vt.row(3).t();x3D = x3D.rowRange(0,3)/x3D.
    at<float>(3);
```

视差角 前文已经从理论上说明任意同名点都可以通过三角化操作实现 3D 点的重建。事实上，由于特征点和位姿信息都存在误差，当同名点之间视差角太小的时候，误差更容易被放大，因此，我们在重建的地图点的时候，尽量过滤掉这种情况。

首先，我们获得两个 2D 点在归一化平面上的坐标 $xn1$ 和 $xn2$，这两个点可以看成是相机到这个点的向量，通过相机到世界坐标系的旋转矩阵 R_{wc}，可以获得世界坐标系下的两条光线 $ray1$ 和 $ray2$，并计算光线之间的余弦值。3D 点在世界坐标系下的光线

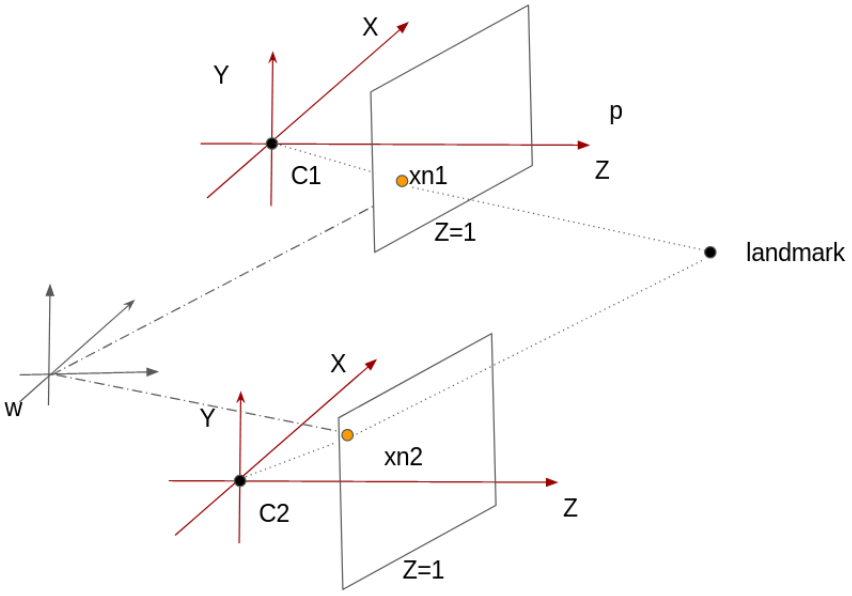


图 38: Caption

```
    // 世界坐标系下 观察光线
    cv::Mat ray1 = Rwc1*xn1;
```

```
    cv::Mat ray2 = Rwc2*xn2;
    // 计算在世界坐标系下，两个坐标向量间的余弦值
    const float cosParallaxRays = ray1.dot(ray2)
    /(cv::norm(ray1)*cv::norm(ray2));

    float cosParallaxStereo = cosParallaxRays+1;
    //ORB-SLAM使用的判断条件
    if(cosParallaxRays<cosParallaxStereo &&
       cosParallaxRays>0 && cosParallaxRays<0.9998)
```

9.1.2 线特征的重建。

对于直线的 3D 重建，我们可以使用类似于点的方式。如图39所示，3D 直线 L 的两个端点 X_s 和 X_e。利用点的方式，公式9.3，我们可以把两个端点投影到图像上，可以获得 $x_s = P_1 X_s$，然后点 x_s 在 l_1 和 l_2 上，因此我们可以建立

$$\begin{aligned} l_1 x_1 &= l_1 P_1 X_s = 0 \\ l_2 x_2 &= l_2 P_2 X_s = 0 \end{aligned} \tag{9.3}$$

其中 l_1 和 l_2 是 2D 直线的方程，他们可以使用 2D 直线的端点计算出来。

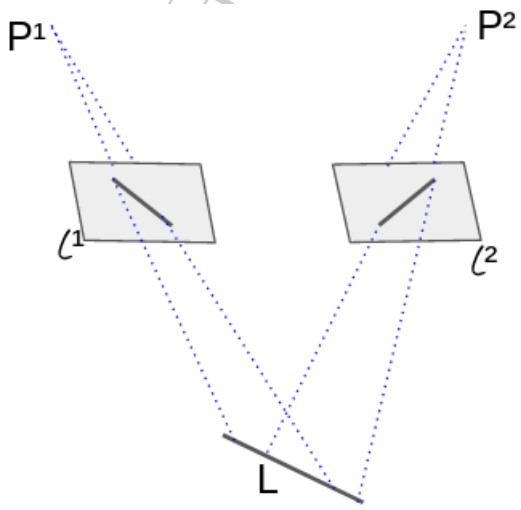


图 39: 特征线的重建

可见公式9.3是不足以求解出来 X_s 的，我们再把 l_1(或 l_2) 的端点加入到方程中，就像是特征点在公式9.3中的表现一样，$l_{1n} = P_1 X_n$，这里 n 表示端点 s 或 e。

```
// 起始点s
cv::Mat A(4,4,CV_32F);
A.row(0) = klF1.t()*M1;
A.row(1) = klF2.t()*M2;
A.row(2) = StartC1.at<float>(0)*Tcw1.row(2)-Tcw1.
    row(0);

A.row(3) = StartC1.at<float>(1)*Tcw1.row(2)-Tcw1.
    row(1);
cv::Mat w1, u1, vt1;
cv::SVD::compute(A, w1, u1, vt1, cv::SVD::
    MODIFY_A| cv::SVD::FULL_UV);

s3D = vt1.row(3).t();
```

视差角 与点重建过程中的处理方式相同，我们计算直线的中点，并考虑中点与中点之间的视差角。通过这种方式，在视差角太小情况下，我们可以避免 3D 直线的重建。

9.2 稠密地图重建

上面介绍了稀疏点云中点线重建，他们都只考虑到一张图片中一小部分信息。而稠密重建的目标是获得每一个像素的深度信息，然后利用像素坐标和深度表示出来。DTAM 是利用 GPU 的一款单目直接法稠密重建系统，作者 Newcombe Richard 专注与稠密重建，并在 2014 年发表了 KinectFusion，这两篇文章都是 SLAM 历史上的里程碑。这里我们先简单的分析 DTAM 的稠密重建思路，详细的优化过程推荐阅读论文原文。在直接法中，我们依靠光度误差来实现参考帧 I_r 和当前帧 I_m 的位姿估计，也就是文中的亮度

视觉SLAM中的几何与学习方法

图 40: 从左到右是深度恢复粗糙到精致的过程。图片来源 DTAM [25]

一致性假设（brightness constancy assumption）。

$$\rho_r(I_m, u, d) = I_r(u) - I_m(\pi(KT_{mr}\pi^{-1}(u, d))) \tag{9.4}$$

公式中 $\pi()$ 是将 3D 投影成 2D。这样就建立起了 d 和 T_{mr} 的光度误差方程。

$$C_r(u, d) = \frac{1}{|I(r)|}\sum_m ||\rho_r(I_m, u, d)|| \tag{9.5}$$

将所有的关键帧的光度误差联合起来构建造优化方程，就可与优化出合适的地图和位姿信息。

根据地图的表现形式来分，稠密重建可以分为 mesh,SDF,surfel。下面我们一一介绍。

9.2.1 Mesh

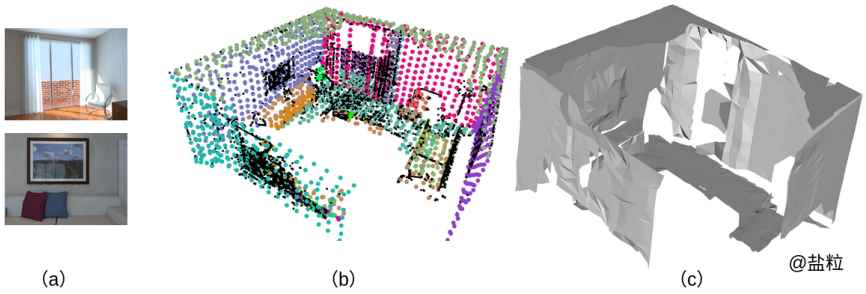


图 41: 基于 mesh 的地图重建。(a) 是场景图片；(b) 是稀疏点云地图；(c) 是 mesh 的密集重建。

mesh 是在点的基础上构网。

盐粒S 中的几何与学习为

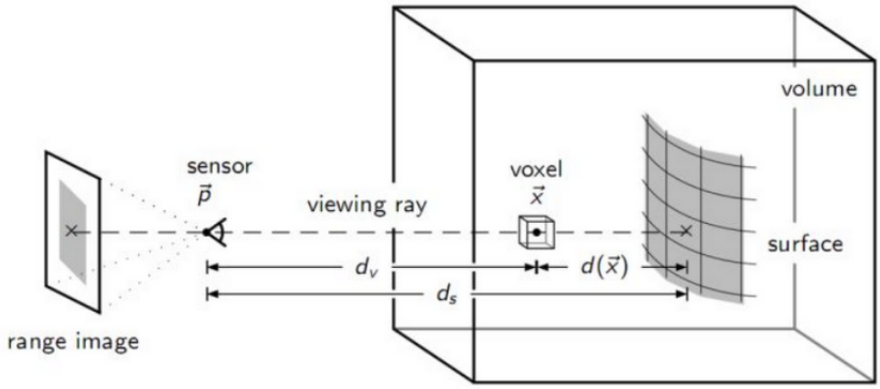


图 42: TSDF 算法示意图

9.2.2 TSDF

TSDF（truncated signed distance function）是一种密集重建中计算隐势面常用的表示方式，其中文全称为“基于截断的带符号距离函数”。TSDF 是 SDF 的一种改进方法，通过在 SDF 中增加表面到体素的截断距离来实现计算量的下降。

TSDF 算法把即将重建的空间看成是一个大的立方体区域 (volume)，并将大的立方体划分成许多小的立方体，我们称之为体素（voxel, 可以看出这个词就是 volume pixel 的意思)。

在 SLAM 或 SfM 问题中，我们通过不同的形式来获得空间信息，如空间三维点 P_w，然后将该点初始成一个体素，很多体素堆放在一起就是我们的重建目标。

1. 如何将当前帧融入到模型中 在讨论这个问题之前，我们先介绍 TSDF 算法中需要用到哪些信息来定义一个 voxel.

- 该 voxel 到最近表面的带符号距离，记作 tsdf(x)
- 体素涉及到更新问题，以此需要记录每次更新的权重，记作 w 。

我们抛开计算 3D 点的不同情况，从比较简单的 RGB-D 数据出发，深度信息由第 i 帧深度数据提供，并获得了相机位姿矩阵 T_{wc}，然后通过 $T_{wc}^T P_w$ 我们可以物体表面信息 P 在世界坐标系下的坐标转化到相机坐标系下 P_c。

深蓝-SLAM 中的几何与

然后利用相机内参矩阵，找到该点对应在当前帧中的位置信息，一遍求取当前帧新观测信息到已知表面的距离信息。

具体来说，深度图可以知道当前帧给出的深度值 d，对因表面上的点距离当前帧相机原点 O 的距离通过下式计算。

$$d_s = distance(O - P_c) \tag{9.6}$$

由相机内参矩阵，反投影 P_c 点求深度图像中的对应像素点 v，我们就可以获得改像素点对应的深度值 d_v。因此我们就获得了当前帧观测点的 sdf 距离。

$$sdf(p) = d_v - d_s \tag{9.7}$$

与 sdf 不同，TSDF 通过对距离设置一个截断阈值，构造了新的截断函数，

$$tsdf(p) = \begin{cases} sdf(d)/|u| & d < u \\ s \end{cases} \tag{9.8}$$

在计算 sdf 距离之后，我们还需要更新权重 $w(p)$ 的计算公式：

$$w(p) = cos(\theta)/d_s$$

其中 θ 是投影光线与表面法向量的夹角。

TSDF 的全局更新地图中的体素 x 的信息记为 $TSDF(x)$ 和 $W(x)$，这个点对应在当前帧中的信息是 $tsdf(x)$ 和 $w(x)$，新的帧进入之后，我们对观测到的信息进行更新：

$$\begin{array}{ll} TSDF(x) = & \frac{W(x)TSDF(x)+w(x)tsdfx}{W(x)+w(x)} \\ W(x) = & W(x) + w(x) \end{array} \tag{9.9}$$

2. 物体表面 物体表面在计算 tsdf 中有很重的作用，用来计算新加入信息的截断距离。一般使用 marching cubes 算法在去寻找 distance 加权和为 0 的等值表面。

9.2.3 Surfel

盖若-SLAM 中的几何与学习方法

10 非线性优化

前面的章节介绍了特征的提取匹配、相机姿态求解和全局地图的构建，但是他们获得的相机姿态和路标点存在误差，并且误差会随着时间的增加逐渐积累。因此本章介绍光束法平差模型对上述信息进行优化，以获得准确的相机姿态求解和全局地图，平差模型解算结果会直接影响机器人、无人机所提供服务的质量。首先，我们描述基于点线面特征的光束法平差模型，然后介绍光束法平差模型常用的解算方法。

10.1 光束法平差模型分析

光束法平差（bundle adjustment，BA）是三维重建和视觉定位中非常关键的一步，在获得相机内参数、外参、影像特征点和空间 3D 物点坐标初始值之后，可以使用 BA 对包括相机的内参和外参在内的上述数据进行整体的优化、消除误差并获得精确的视觉信息。

光束法平差被认为是摄影测量和计算机视觉中的黄金定律，而数学本质上来说，光束法平差目标方程模型就是非线性最小二乘模型，并且是关于相机姿态 (ξ) 与三维路标信息 ($Landmark$) 的最小二乘优化问题。相机姿态包含平移矩阵 R 和旋转矩阵 t，而常用的三维路标信息有点 P、线 L 和面 Q。

10.1.1 点特征的误差方程

一组同名像点对应着一个空中的三维点 $P_j = (x, y, z)$，通过计算三维点在影像上的重投影与对应的 2D 特征之间的距离，我们可以获得点的冲投影误差方程，

$$r^p_{k,j} = p_k - \pi(R_{k,j}P_j + t_{k,j}) \tag{10.1}$$

公式中的 $\pi()$ 就是投影过程，f_x 与 f_y 则是相机的内参信息。在优化过程中，误差方程对于相机位姿的的 Jacobian 矩阵可以表示为，

$$\frac{\partial e^p_{k,j}}{\partial \xi} = \begin{bmatrix} \frac{xyf_x}{z^2} & -\frac{z^2+x^2}{z^2}f_x & \frac{yf_x}{z} & -\frac{f_x}{z} & 0 & \frac{xf_x}{z^2} \\ \frac{z^2+y^2}{z^2}f_y & -\frac{xyf_y}{z^2} & -\frac{xf_y}{z} & 0 & -\frac{f_y}{z} & \frac{yf_y}{z^2} \end{bmatrix} \tag{10.2}$$

北京-SLAM 中的几何与学习方法

10.1.2 线特征的误差方程

与特征点相比，直线可以被表示为两个端点 p_{start} 和 p_{end}。首先我们利用这两个端点计算直线的方程，通过单位化处理，可以显出直线长度带来的影响。

$$l = \frac{p_{start} \times p_{end}}{\|p_{start}\|\|p_{end}\|} = (a, b, c) \tag{10.3}$$

类似与点的重投影误差，我们来计算线特征的误差方程。在 2D 图像上获得直线方程之后，通过将空间 3D 直线的端点（P_{start} 和 P_{end}）重投影回来。对于每一个端点 $P_x, x \in [start, end]$, 误差方程可以被描述为：

$$e^l_{k,j} = l\pi(R_{k,j}P_x + t_{k,j}) \tag{10.4}$$

同理，上述误差方程带来的 Jacobian 矩阵为

$$\frac{\partial e^l_{k,j}}{\partial \xi} = \begin{bmatrix} -\frac{f_y^2 z^2 + axyf_x + by^2 f_y}{z^2}, & \frac{az^2 f_x + ax^2 f_x + bxyf_y}{z^2}, & -\frac{ayf_x - bxf_y}{z}, \\ \frac{af_x}{z}, & \frac{bf_y}{z}, & -\frac{axf_x + byf_y}{z^2} \end{bmatrix} \tag{10.5}$$

10.1.3 面特征的误差方程

平面信息可以在人造环境中随处可见，常用的平面表示方式是海森表示 $n_\pi P + d = 0$, 这种方式可以很直观的平面信息，但是存在过参的问题，因此我们使用 $q(\pi) = (\phi, \psi, d)$ 来描述平面，这里 ϕ 和 ψ 是平面法向量的 azimuth 和 elevation，d 表示点到平面的距离。

$$q(\pi) = (\phi = arctan(\frac{n_y}{n_x}), \psi = arcsin(n_z), d). \tag{10.6}$$

所以，我们可以描述两个匹配平面（π_k 和 π_x）之间的误差方程，

$$e^\pi_{k,\pi_x} = q(\pi_k) - q(T^{-T}_{cw}\pi_x) \tag{10.7}$$

这里 $T^{-T}_{cw} = (R_{cw}, t_{cw})$ 是从世界坐标系到相机坐标系下的转化矩阵，R_{cw} 用来旋转平面的法向量，t_{cw} 用来处理点到平面的距离。

同理，我们可以获得面误差方程(10.7)带来的 Jacobian 矩阵，

$$\frac{\partial e^\pi_{k,\pi_x}}{\partial \xi} = \begin{bmatrix} \frac{n_{cx}n_{cz}}{n_{cx}^2+n_{cy}^2} & \frac{n_{cy}n_{cz}}{n_{cx}^2+n_{cy}^2} & -1 & 0 & 0 & 0 \\ \frac{n_{cy}}{\sqrt{1-n_{cz}^2}} & \frac{n_{cx}}{\sqrt{1-n_{cz}^2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & n_{cx} & n_{cy} & n_{cz} \end{bmatrix} \tag{10.8}$$

盲源SLAM中的几何与学习方法

10.1.4 构建光束法平差模型

$$
\begin{aligned}
t* = & argmin \sum_{j}^{M} \rho_p \left({e_{k,j}^p}^T \Lambda_{p_{k,j}} e_{k,j}^p \right) \\
& + \rho_l \left({e_{k,P_x}^l}^T \Lambda_{p_{k,P_x}} e_{k,P_x}^l \right) + \rho_\pi \left({e_{k,\pi_x}^\pi}^T \Lambda_{k,\pi_x} e_{k,\pi_x}^\pi \right)
\end{aligned} \tag{10.9}
$$

在公式10.9中，Λ 是一个协方差矩阵，同一个传感器观场景进行观测的时候，我们认为误差之间是相互独立的的，故通常作为一个对角阵存在，在实际的工作中通常取单位阵。通过对目标方程的解算，我们最终可以获得优化的结果。

10.2 经典的光束法平差解算方法

在非线性优化问题的诸多解法中，牛顿法因为具备二阶收敛速度而得到广泛的应用，但是在迭代求解的过程中，牛顿法因计算复杂度过高、海森矩阵稠密等问题，无法应用在实际的光束法平差问题中。因此，我们要有光束法平差问题的专门解算方法。

在光束法平差模型解算放法中，高斯牛顿（Gauss-Newton，GN）方法通过忽略牛顿方法中产生的高阶导数很好的降低了算法的复杂度，作为一种具有快速收敛性的收敛方法，被广泛用来解算光束法平差中，获得精准的相机位姿信息和 3D 特征点信息。但是 GN 对初值的精度要求较高，在初值不好的情况下，将面临发散的危险。列温伯格-马夸尔特（Leverberg-Marquart，LM）方法通过增加阻尼项，保持了收敛的鲁棒性，但是这种方法使用置信区间来尝试阻尼的值，不能准确的获得下降方向，因此需要多次迭代才能完成收敛。这两种方法是最广泛使用的解算方法，除此之外还有共轭梯度法、拟牛顿等解算方法。

10.2.1 高斯牛顿方法

高斯牛顿方法从牛顿方法简化而来，因此在剖析高斯牛顿方法发散原因之前，本文首先介绍牛顿优化方法。通过牛顿法的介绍，可以更加容易理解高斯牛顿发散的数学机理。在数值优化领域，牛顿法因具有较快的收敛速度，被用来求解无约束优化问题。这个方法的主要思路是在迭代点处进行泰勒展开，使用展开式对目标方程进行近似。因此非线性问题就可以转化为线

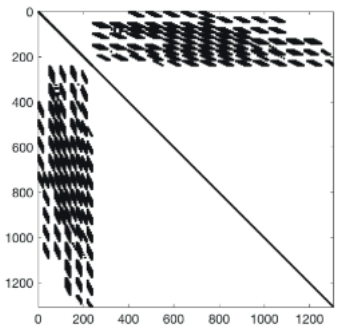


图 43: 海森矩阵的对称稀疏结构。

性问题，并使用求二次模型的极小点的方式来获取新的迭代点。通过多次迭代，达到满足的精度为止。对公式10.9进行一阶泰勒展开，这里我们只展开公式中特征点的部分

$$z(x) = f(x_{k+1}) + g_k(x - x_{k+1}) + \frac{(x - x_{k+1})^T G(x - x_{k+1})}{2} \tag{10.10}$$

这里 $G = \Delta^2 f(x_{k+1})$，$g_{k+1} = \Delta f(x_{k+1})$。对于高斯解法，通过对上述公式求导，并令导数为零，就可以获得极值点，

$$\Delta z(x) = \Delta f(f(x_{k+1})) + G(x - x_{k+1}) = 0 \tag{10.11}$$

如果 G^{-1} 是非奇异阵，那么 $x_{k+1} = x_k - G^{-1} g_{k+1}$，这样就可以实现迭代更新。在之前已经描述过了，牛顿法具有不低于二阶的收敛速度，但是该方法在遇到 BA 问题时，求解 Hessian 矩阵工作量巨大，同时不能保证目标函数的 Hessian 矩阵在每个迭代点处都能保持正定。为了利用牛顿法的优势，避免不足，出现了 Gauss-Newton 等 BA 解法。

正常情况下 $Hd = J(x)^T F(x)$,其中 H 为 Hessian 矩阵,$H = J(x)^T J(x) + \sum F_i(x)^2$。在 Gauss-Newton 方法中，忽略公式的第二项，使用 $J(x)^T J(x)$ 来代替 Hesse 矩阵，求解方程的下降方向。因此，可以看出缺少高阶导数信息的高斯牛顿算法在迭代求解过程中必须满足 Hessian 矩阵正定的条件，进而要求 Jacobian 矩阵必须是列满秩，不然算法失效。而 Jacobian 矩阵是相机与三维特征点之间的表示方式是随着输入参数的变化而变化的。因此对 Jacobian 矩阵的要求就变成了对观测初值的要求，这就是高斯牛顿方法对初值敏感的根本原因。

西北工业大学 计算机学院 SLAM 中的几何与学习方法

Algotithm 1: Guass-Newton

Input: A vector function f: R^m ->R^n n>m,
A measure vector $x \epsilon R^n$ 初值 P_0
Output: a vector P^+ =argmin$\|f(p)\|^2$ $f(p) = x - r(p)$
K =0 $A = J^T J g = J^T f(p)$ $B_0 = A$

$$Ad_k^{GN} = -J_k^T f(x_k)$$
$$\partial = 1$$
$$x := x + d_k^{GN}$$

图 44: 高斯牛顿法的伪码算法过程

值得说明的是，虽然高斯牛顿法不是非常的鲁棒，但是中对称稀疏的海森矩阵形式（如图43），有利的推动了光束法平差模型的使用，因为这种稀疏结构可以通过矩阵知识很快速的被计算出来，即使是面对数万维度大小的矩阵。

10.2.2 Leverberg-Marquart

有上面内容可知，对于 GN 方法，使用 $J^T J$ 矩阵近似 Hessian 矩阵很容易发生奇异，因此，光束法平差问题常常无法解算。为了克服 GN 方法的缺点，LM 方法被提出来，并作为应用最为广泛的方法求解 BA 模型。LM 算法可以表示为公式

$$(J^T J + \lambda I) d_k^{LM} = -J^T g_k \tag{10.12}$$

其中 I 矩阵表示单位阵,LM 算法的原理是在近似 Hessian 矩阵上的对角线上加上一个阻尼值，从而近似 Hessian 矩阵的奇异问题。通过观察公式10.12，LM 方法是一种结构化的方法，其中包含两个部分：$J^T J$ 与 λI，可以通过调整 λ 的大小，调节 $J^T J$ 与 λ 之间的比重。当 λ 特别大时，$J^T J$ 可以忽略，即公式10.12可以变成梯度下降方法公式：

$$d_k^{LM} = -\frac{1}{\lambda} J^T g_k \tag{10.13}$$

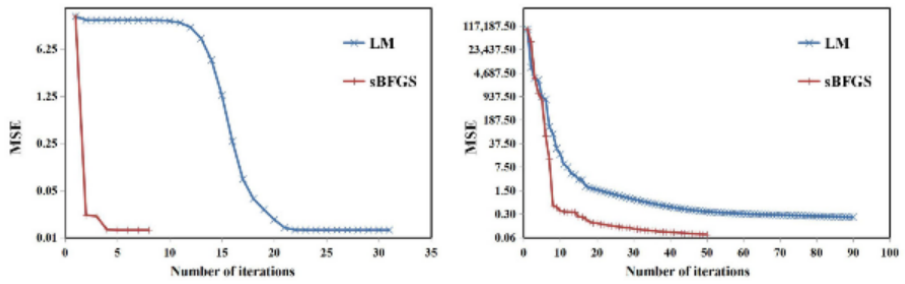


图 45: 两种方法收敛结果。其中红色稀疏结构化的 BFGS 方法的收敛结果，蓝色为 LM 法收敛结果。

当 λ 特别小时候，λI 可以忽略，求解方法即变成 GN 方法。因此可以把 LM 方法看做是最速梯度下降法和高斯牛顿法的结合。使用置信区间来更新 λ，如果当前次迭代可以得到比前一次迭代更小的优化结果，此时公式10.12就需要减小 λ 的大小（$\lambda = \lambda \cdot t, t = max(\rho, \frac{1}{3})$）；假如不满足条件时，需要增大 λ 值（$\lambda = \lambda \cdot v, v = 2v$）。

进入的优化环节的初值情况是无法控制的，与高斯牛顿的初值敏感特点一样，LM 方法对于初值较差的情况，LM 同样是被动的调整阻尼值，不能根据当前迭代的实际情况来完成阻尼的更新，因此 LM 方法常常在需要较多次的迭代才能完成。从图45的实验结果可以看出，LM 迭代方向并不是稳定的，会不会在迭代初期经历近似平原区域，还要看光束平差模型中优化的输入数据。

10.2.3 稀疏结构化 BFGS 方法

在上述篇幅分析了常见的解算方法：高斯牛顿方法和 LM 方法，清楚的把这两种方法的优势与缺陷，及其成因展现在读者眼中。针对上述问题，稀疏结构化 BFGS（sBFGS）[20] 通过探索海森矩阵中高阶信息的保留方式，提出一个保持更好下降方向的解算方法。

BFGS 正定保持特性 BFGS 是一种常用的拟牛顿方法，由于该方法经过秩 2 修正，因此在迭代求解过程中，对于输入的正定矩阵，在满足情况的条

武汉大学测绘学院

件下，能保持正定性。

$$A_{k+1} = A_k - \frac{A_k r_k r_k^T A_k}{r_k^T A_k r_k} + \frac{z_k z_k^T}{z_k^T r_k} \quad (z_k^T r_k > 0) \tag{10.14}$$

其中，$r_k = X_{k+1} - X_k$ 是两次迭代之间的残差向量。另外，z_k 是雅克比矩阵的差，$z_k = (J_{k+1}^T - J_k^T)||r_{k+1}||$。

结构化的 BFGS 方法 在上面的分析中我们明白高斯牛顿方法无法保持矩阵正定，现在我们为其加入一个正定矩阵，使得在任何时候都能够保持正定，这样可以解算任意情况的初值。如下式所示：

$$B_{k+1} = J_{k+1}^T J_{k+1} + A_{k+1} \tag{10.15}$$

在本方法中通过 BFGS 算法来更新，具体的证明过程可以参考优化书籍，不再赘述。

由于 BFGS 对于正定性质的保持能力，假设为正定矩阵，使用 BFGS 算法更新时，只要，并且那么可以保障的正定性，进而为正定矩阵，保证了下降方向始终是正确的。

$$A_{k+1} = \begin{cases} A_k - \dfrac{A_k s_k s_k^T A_k}{s_k^T A_k s_k} + \dfrac{z_k z_k^T}{z_k^T s_k} & if(z_k^T s_k > \upsilon) \\ A_k & otherwise \end{cases} \tag{10.16}$$

因此上面的式子中，使用 A_{k+1} 近似 Hessian 矩阵中的含有高阶导的部分，我们通过 BFGS 方法来更好的模拟 Hessian 矩阵。通过公式 10.16获得的 A_{k+1} 是一个密集矩阵，由于 BA 问题涉及到的矩阵的维数非常大，因此对于密集矩阵的求解将会非常困难。因此本文根据 Hessian 矩阵的低阶导数信息的稀疏化高阶导数矩阵，因此可以实现稀疏化的 A_{k+1}，记做 A_{k+1}^s。

$$B_k^s = \begin{cases} J_{k+1}^T J_{k+1} + A_{k+1}^s & if(z_k^T s_k > \upsilon) \\ J_{k+1}^T J_{k+1} + ||r_k|| I & otherwise \end{cases} \tag{10.17}$$

如公式 10.18所示，可以通过上面的稀疏化的 B_k^s 矩阵，在求解下降方向以后，可以顺利的对光束法平差问题进行解算。与 LM 方法相比，本方法在迭代点远离真值时，依然能保持超线性收敛速度，随着迭代的进行，当前迭代点逐渐接近真值，本方法中的高阶导数矩阵的作用就比较微弱，本方法以高斯牛顿方法进行收敛，同样保持超线性收敛速度。

$$B_k^s d_k^{sBFGS} = -J^T g_k \tag{10.18}$$

武汉大学测绘学院 SLAM 中的几何与学习方法

1956 其中 d_k^{sBFGS} 表示 sBFGS 方法下降方向。

盖粒-SLAM 中的几何与学习方法

11 优化中的位姿描述

11.1 四元数

11.2 李群与李代数

12 优化中的特征描述

12.1 点特征-逆深度

12.2 线特征

众所周知，线特征比点特征可以提供更多的约束条件，并且在某些场合下比点特征要鲁棒。但是如果我们想要把线特征加入到 SLAM 系统中面临的是参数化和优化求导。这篇文章介绍了贺博的 PL-VIO 中线特征的参数化和求雅克比。

下面我们就来详细介绍一下空间直线的两种参数化方法：Plücker 参数化方法，直线正交表示方法。为什么需要两种参数化方法呢？因为空间中的直线有 4 个自由度，而 Plücker 参数化方法需要使用 6 个参数表示直线，这样就会导致过参数化，过参数化在优化的时候就需要采用带约束的优化，不太方便。于是引入了可以用 4 个参数更新直线的正交表示来方便优化。这两种参数化方法可以很方便的相互转换，所以我们可以在 SLAM 系统中同时使用这两种参数化形式，在初始化和进行空间变换的时候使用 Plücker 坐标，在优化的时候使用正交表示。下面我们就来详细了解这两种参数化方法和优化的雅克比求导。

1、Plücker 参数化方法

在 3D 空间中，直线 [公式] 的 Plücker 坐标表示为 [公式] ，其中 [公式] 是直线的方向向量，[公式] 是由直线和光心构成的平面 [公式] 的法向量。Plücker 坐标是过参数化的，比如 [公式] 和 [公式] 之间存在约束关系（[公式])。于是 Plücker 坐标不能直接使用无约束的优化。不过这种使用法向量和方向向量的表现形式在初始化直线和进行空间变换的时候很方便。所以在 SLAM 系统中我们可以使用 Plücker 坐标来初始化和变换，具体的初始化和变换方法如下。

图 1 线特征的 Plücker 坐标 (a) Plücker 线坐标系 (b) 新观测到的线的初始化当给定从世界坐标系 [公式] 到相机 [公式] 的变换矩阵 [公式] ，可以

北京SLAM中的人们学习方法

通过下面的形式将世界坐标系和相机坐标系下的 Plücker 线坐标进行变换：

[公式]

上式中 [公式] 表示的是三维向量的反对称矩阵，[公式] 表示的是从世界坐标 [公式] 到相机坐标 [公式] 的 Plücker 线坐标的变换矩阵。

[公式] 当我们从不同的两帧相机观测到一个新的线路标的时候，Plücker 坐标初始化方式也很简单。如图 1b 中所示，直线 [公式] 在两帧图像 [公式] 和 [公式] 中表示为两个线段 [公式] 和 [公式] 。线段 [公式] 在归一化平面可以被两个端点表示，[公式] 和 [公式] 。再加上坐标原点 [公式] ，这三个点可以确定一个平面: [公式]

[公式]

其中：[公式]

然后对同一个空间直线在两帧相机平面的投影和相机光心可以确定两个平面 [公式] 和 [公式] ，通过两个平面可以确定唯一的空间直线。有了这两个平面我们可以得到对偶 Plücker 矩阵 [公式]

这里对偶矩阵的内容可以看《多视图几何》[公式]

然后从对偶矩阵中可以提取出 Plücker 坐标 [公式] 。法向量 [公式] 和方向向量 [公式] 都不要求是单位向量。

2、正交表示法

由于 3D 空间的直线只有 4 个自由度，使用 Plücker 坐标这种过参数化的表示形式在优化的时候不方便，所以我们引入四个参数的正交表示 [公式] 。Plücker 坐标和正交表示之间可以很方便的互相转换，我们之后会分别介绍如何从 Plücker 坐标到正交表示和正交表示怎么变换成 Plücker 坐标。首先我们需要知道了直线的 Plücker 坐标 [公式] ，然后对 [公式] 进行 QR 分解，得到：

[公式]

分解得到的第一项是正交矩阵 [公式] ，是一个旋转矩阵。所表示的是相机坐标系到直线坐标系的旋转。其中直线坐标系的定义如下：用直线的方向向量以及直线和光心组成平面的法向量作为坐标的两个轴，再用他们叉乘得到的向量作为第三个轴，所以

[公式]

其中 [公式] 代表的是相机坐标系到直线坐标系在 [公式] ，[公式] 和 [公式] 轴的旋转角。

到此我们得到了正交表示的第一项，第二项需要做一些小变换。由于将

视觉SLAM十四讲 从理论到实践

[公式] 结合之后只有一个自由度，所以我们可以用三角函数矩阵参数化：

[公式]

上式中的 [公式] 是旋转角。由于坐标原点（相机光心）到 3D 直线的距离是 [公式] ，所以 [公式] 包含了距离信息 [公式] 的。根据 [公式] 和 [公式] 的定义可以看出，4 个自由度包括旋转的 3 个自由度和距离的一个自由度。在优化的时候，我们使用 [公式] 作为空间直线更新的最小表示。

正交表示到 Plücker 坐标之间的变换可以通过下面的方式计算出来：

[公式]

其中 [公式] 代表的是 [公式] 矩阵的第 i 列。虽然 [公式] 和 [公式] 有一个尺度的差，但是它们表示的是同一条空间直线。

3、直线的观测模型和误差

图 2 空间直线投影到像素平面要想知道线特征的观测模型，我们需要知道线特征从归一化平面到像素平面的投影内参矩阵 [公式] 。如图 2，点 C 和 D 是直线 [公式] 上两点，点 c 和 d 是它们在像素平面上的投影。[公式] ,K 是相机的内参矩阵。[公式] ，[公式] 。那么有

[公式]

上式表明，直线的线投影只和法向量有关和方向向量无关。

关于投影的误差，我们不可以直接从两幅图像的线段中得到，因为同一条直线在不同图像线段的长度和大小都是不一样的。衡量线的投影误差必须从空间中重投影回当前的图像中才能定义误差。在给定世界坐标系下的空间直线 [公式] 和正交表示 [公式] ，我们首先使用 [公式] 将直线变换到相机 [公式] 坐标下。然后再将直线投影到成像平面上得到投影线段 [公式] ，然后我们就得到了线的投影误差。我们将线的投影误差定义为图像中观测线段的端点到从空间重投影回像素平面的预测直线的距离。

[公式]

其中 [公式] 和 [公式] 是图像中观测到的线段端点，[公式] 是重投影的预测的直线。

4、误差雅克比推导

如果要优化的话，需要知道误差的雅克比矩阵：

线特征在 VIO 下根据链式求导法则：

[公式]

其中第一项 [公式] ，因为

[公式]

东南大学SLAM中的几何与学习方法

所以：

[公式]

第二项 [公式] ，因为

[公式]

所以：

[公式]

最后一项矩阵包含两个部分，一个是相机坐标系下线特征对的旋转和平移的误差导数，第二个是直线对正交表示的四个参数增量的导数

第一部分中，

[公式]

在 VIO 中，如果要计算线特征的重投影误差，需要将在世界坐标系 [公式] 下的线特征变换到 IMU 坐标系 [公式] 下，再用外参数 [公式] 变换到相机坐标系 [公式] 下。所以

[公式]

其中 [公式]

[公式] 线特征 [公式] 只优化状态变量中的位移和旋转，所以只需要对位移和旋转求导，其他都是零。下面我们来具体分析旋转和位移的求导。首先是线特征对旋转的求导：

[公式]

然后是线特征对位移的求导：

[公式]

第二部分中 [公式] ，先解释第一个 [公式]

[公式]

所以 [公式]

然后后面的 [公式] 有两种思路，先介绍第一种：

[公式]

其中 [公式] 对 [公式] 和 [公式] 求导，因为 [公式] ，所以

[公式]

[公式]

然后是 [公式] 对 [公式] 和 [公式] 对 [公式] 的求导，

因为 [公式] ，所以

[公式]

所以，可得

视觉-SLAM中的几何与学习方法

[公式]

12.3 面特征

众所周知，SLAM 中添加线和面这样的高级特征可以增加系统的鲁棒性，但是在使用面特征的时候有两个问题：面特征怎么去提取？提取出来的面特征怎么参数化和优化？在这篇文章中，我们会给出几种常用的面特征参数化方法。如果有错误或者疏漏的地方，欢迎大家批评指正。

Hesse 形式对于面特征来说，大家都知道的一种参数化形式就是 Hesse 形式，用一个平面的单位法向量和平面距离原点的距离来表示：

[公式]

其中 [公式] 表示平面上的一点。在使用 Hesse 形式的时候，我们也可以很方便的把平面表示在不同坐标系下进行变换。假设 [公式] 是世界坐标系下的平面方程，[公式] 是本地坐标系下平面方程，[公式] 是本地坐标到世界坐标的旋转，[公式] 是本地坐标到世界坐标的平移，则有

[公式]

那么我们是不是就可以愉快的使用这种方式呢。答，不可以。因为 SLAM 中有一个优化的问题，优化的时候上面这种 Hesse 形式会导致过参数化（overparametrization）的问题。

过参数化问题什么是过参数化问题呢？三维空间中的平面只有 3 个自由度，但是上面的 Hesse 形式有 4 个参数，这种参数化之后参数的个数大于实际表示的自由度的的表现形式就被成为过参数化（overparametrization）。过参数化的形式会引发什么样的问题呢？对于高斯-牛顿优化问题来说，如果是过参数化的表现形式，优化过程中计算出的 Hessian 矩阵会不满秩，也就不能求逆。所以对于过参数化的问题，一种解决办法就是列文伯格-马夸尔特算法 (L-M) 提供的正则化 (regularization)。然而，由于代价函数在某些方向上是平面，因而存在无穷多的解，会使得算法的收敛速度变为梯度下降法的线性收敛速度，而不是高斯-牛顿法的二次收敛速度，因此会导致收敛的速度变慢。除此之外，过参数化还不可以使用信赖区域法。所以下面我们就来介绍几种平面参数化的方法。

球坐标形式其实平面的 Hesse 参数化形式可以被分成两个部分，一个单位法向量部分和一个距离部分。平面的 Hesse 形式的过参数化就是因为单位法向量部分有 3 个参数，但是实际只有两个自由度导致的。那么我们就可以从这个法向量着手解决过参的问题。单位法向量可以被看成是一个单

位圆球上的一点。那么我们就可以用两个角度 [公式] 来参数化这个点，从而表示出这个单位法向量。那么我们就可以把 Hesse 形式变换成如下的形式：[公式]

图 1 半球坐标系 g2o 中也实现了这种面特征的参数化形式的节点。这种方式我们可以便捷的计算出 Hesse 形式对其的雅克比：

[公式]

但是这种形式的参数化虽然是最小参数化形式，但是在优化的时候会产生奇异性。当 [公式] 的时候，上面的雅克比的秩会从 3 降到 2。

切平面形式那么上面的球坐标的形式会产生奇异性，还有没有更好的形式呢，也是有的。比如 GTSAM 里面表示面的方式用切平面的方式来更新单位法向量，同样也可以被用来优化单位法向量。和上面类似，在单位球平面上找到该法向量对应的点的切平面。然后就可以任意找到一对正交单位法向量，这两个法向量乘以单位就可以表示这个平面上的任意一点。

[公式]

法向量的优化参数就可以被表示为切平面上这两个正交向量的大小系数。然后优化的时候求雅克比的话也很方便。不过这种参数化形式一般只用来对法向量方向进行微调。

图 2 切平面最近点（Closest Point，CP）这是黄国权黄老师团队在论文 LIPS: LiDAR-Inertial 3D Plane SLAM Patrick 中提出的一种平面参数化方法。这种方式是用在平面上距离当前表示坐标系原点最近的 3D 坐标点来表示平面。这种表示方式有两个优点：1. 是最小化参数形式；2. 具有真实的物理意义。

[公式]

但是需要说明的是，这种参数化方式在平面参数 d 为 0 的时候会有奇异性。所以一般情况需要将平面参数化在检测到平面的坐标系之下，这样就可以消除奇异性。

图 3 最近点单位四元数这是 CMU 的 Michael Kaess 教授在论文 Simultaneous Localization and Mapping with Infinite Planes 中提出的方法。对于单位四元数来说，这是一种采用了 4 个参数，但是只有 3 个自由度，同时只需要 3 个参数去更新的参数化形式。那么这种形式很巧合的和平面有 4 个参数，但是只有 3 个自由度在数学上高度吻合。那么如果我们可以把平面的 4 个参数投影到单位四元数平面上，那么就可以用单位四元数去优化平面了。

视觉SLAM中的几何学习方法

所以我们只需要把 Hesse 形式进行一个小小的变形就可以得到单位四元数化的平面参数化形式啦。

[公式]

对于这种参数化形式，主要是用来优化不同坐标系下的平面的变换的误差。只能整个平面的方程一起更新。同时这种参数化形式的更新和其他方式的参数化不太一样，四元数的参数化更新和旋转矩阵的参数更新是一样的，是对应到李代数去更新的。所以这种参数化形式有个令人诟病的地方就是：这种参数化形式的几何意义并不明确。

利用四元数去优化平面方程构建误差一般采用以下形式：

[公式]

退化二次曲面最后给大家介绍一种特殊的表示方法。这是 g2o 的作者最近在论文 Unified Representation and Registration of Heterogeneous Sets of Geometric Primitives 中提出的一种可以同时用一种参数化形式同时表示点线面的参数化形式。大神这么做的意义是想把所有特征都统一起来，也可以方便的构建出不同特征之间的约束（比如，点到面，点到线，线到面）。本人认为这应该也是未来多特征 SLAM 的一个大的研究方向吧。一个二次曲面可以通过下面这种方式参数化，其中 [公式] 是二次曲面上的点，[公式] 是二次曲面的原点，[公式] 是一个 3×3 的对称矩阵。

[公式]

对 [公式] 做一个特征值分解可以得到 [公式] 。其中 [公式] 的每一列都是二次曲面坐标系的一个轴，[公式] 是包含了 [公式] 的特征值的一个对角阵。

这种表示方法是把一个点看成是一个半径为 0 的一个球；一条直线看成是一个半径为 0 的一个圆筒；一个平面看成是两个匹配上的平面。所以它们对应的分解得到的 [公式] 的形式分别如下表所示。（1 代表该特征值和其他特征值相等，而 0 代表特征值为 0）

所以对于任意一个点线面的特征实体 [公式] ，可以用下面 3 个部分来参数化：

[公式]

那么两个特征之间的变换也可以从旋转和平移中简单的得到。

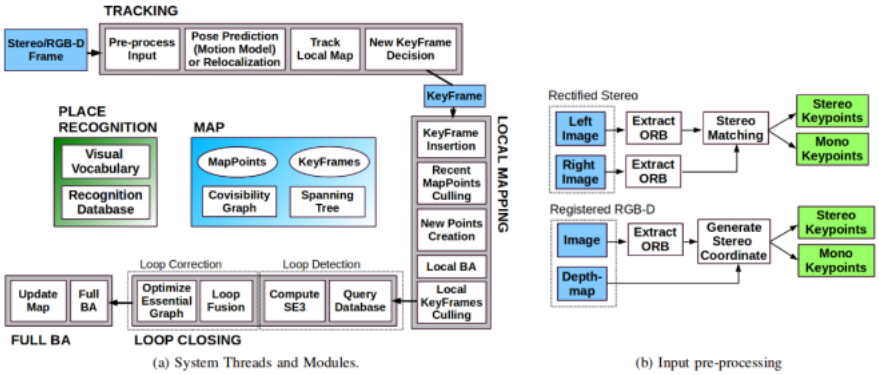
[公式]

一些其他形式最后，还有很多种方法没有被提到，比如 VINS on wheels 中的一个 2 自由度的四元数加距离，Probabilistic plane fitting in 3d and an

paper-SLAM 中的几何学公式

application to robotic mapping 和 Probabilistic RGB-D odometry based on points, lines and planes under depth uncertainty 中分别对 Hesse 形式的四个参数除以一个参数导致可以使用 3 个参数最小表示平面。就不在此赘述了。因为它们都有一定的限制条件，并不大众化。

盖粒-SLAM 中的几何与学习方法



(a) System Threads and Modules. (b) Input pre-processing

图 46: ORB-SLAM2 系统框图。[24]

13 经典的纯几何 SLAM

13.1 ORBSLAM (V1 and V2)

ORB-SLAM2 [24] 是由 Raúl Mur-Artal 博士等人提出的一个完备的基于特征点的纯视觉实时 SLAM 系统，适用于单目、双目和 RGB-D 相机，发表于 2017 年。作者在 2015 年第一次将单目版本（ORB-SLAM [23]）发表在 IEEE Transactions on Robotics（ToR）期刊上，这是一个基于纯特征点的单目实时 SLAM 框架。总体来说，该 ORB-SLAM 的两篇文章是基于特征的 SLAM 系统的大成之作，系统本身非常的完整，包含了稀疏地图，回环和优化环节。ORBSLAM2 将传感器从单目扩展到双目和 RGB-D，成为了迄今为止影响力最大的特征法 SLAM 系统之一，为 SLAM 在工业界和学术界的发展作出巨大贡献。

在特征匹配、跟踪和回环检测等所有环节全部采用统一的 ORB 描述子，通过地图复用，有效提高了系统的鲁棒性。该系统分为四个模块，分别是前端、局部建图和回环检测以及闭环，系统使用 3 个线程。。下面我们着重介绍 ORB-SLAM2 中重要的几个模块。

13.1.1 单目系统的初始化

系统有两个初始化模块，双目和 RGB-D 共用一个模块，单目有自己的一个初始化模块，在追踪开始后，根据输入图像的样式，进去相应的初始化

程序。相比于双目和 RGB-D，单目不具有尺度，因此更加的复杂，这里我们介绍单目的初始化过程。首先是对连续的两帧（F_i 和 F_{i+1}）提取特征点，如果他们满足特征匹配数量的约束。就进一步计算两帧之间的位姿。如果，不是连续的两帧那么就重新开始。

为了提升初始化的精度，作者同时估计两帧图像之间的单应矩阵（Homograph）和基础矩阵（Fundamental），单应矩阵可以用来处理平面之间/较小视差角的旋转与平移，基础矩阵则能处理更普遍的特征点。对于两帧图像之间的同名点，系统同时估计上述的两个矩阵 H_{cr} 和 $F(cr)$，其中 c 和 r 分别表示当前帧和参考帧。对于同名点随机选择八个点，并建立很多组。然后通过公式13.1将这两个矩阵分别将一帧图像中同名点转换到另一帧图像上，来计算同名点在每帧图像上的像素距离。

$$S_M = \sum_i (\rho_M(d^2_{cr}(x^i_c, x^i_r, M)) + \rho_M(d^2_{rc}(x^i_c, x^i_r, M))) \tag{13.1}$$

这里 i 表示同名点的数量，M 为 H_{cr} 或 $F(cr)$ 矩阵中的一个。最后，利用公式13.2来比较两个误差的大小，并选择一个矩阵。

$$R_H = \frac{S_H}{S_H + S_F} \tag{13.2}$$

在 ORB-SLAM 中，如果 $R_H > 0.045$ 选择单应矩阵，否则选择基础矩阵来初始化两帧图像之间的位姿。在完成位姿之后，系统利用该信息和同名点信息进行路标点的三角化（Triangulation）工作，创建稀疏地图。

13.1.2 姿态估计与优化

对于相机姿态的估计是 SLAM 系统的一个重要任务,这个环节为 AR/VR，机器人和无人驾驶等应用提供支持。在 ORB-SLAM2 中，利用相机基线来实现 RGB-D 向双目相机的转化。

- 系统在 2D 特征提取中为了使特征分布更加均匀，将图像分块提取特征。对于双目和 RGB-D 相机，系统可以直接获得特征点的 3D 信息，而单目相机需要在后面进行三角化操作。
- 完成当前帧的特征提取之后，系统通过追踪参考帧或使用运动模型的方式获得当前帧的初始 6D 位姿。两个方法都是先给当前相机的位姿赋一个值，然后再用重投影误差去优化。然后使用局部地图再优化一遍，局部建图模块是利用前端输入的关键帧插入局部地图进行局部优化。

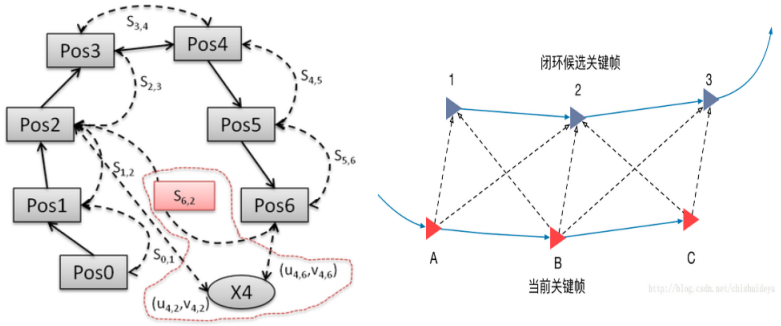


图 47: 闭环检测检测

- 如果检测出来是关键帧，另一个线程基于共视图的优化。

ORB-SLAM 的这种相机位姿估计策略非常鲁棒，尤其是当检测的特征点充足时，可以实现高精度的位姿估计。

13.1.3 基于词袋的闭环检测

对于基于 Frame-to-Frame 和 Map-to-Frame 的 SLAM 系统，随着追踪时间的增长，系统也在不断积累着误差，这种现象被称之为系统漂移（drift）。回环检测（Loop Closure）是一种消除漂移的有效的解决方案。闭环检测是在历史轨迹中查找是否存在与新关键帧具有相同观测的和约束的关键帧，筛选过程主要包括位置识别和几何验证。

在 ORBSLAM2 中，闭环检测线程通过词袋法 (DBoW2) 加速闭环匹配帧的筛选。这种方法的目的是用一种更加抽象、更加高效的特征集合来描述一幅图像。通过 DBoW2, 作者训练了一个较大的特征字典。对于每一帧图像中提取的特征，只要在字典树中逐层查找，最后都能找到与之对应的单词。最后可以使用一个分布或直方图来准确的描述这个图像。从图47 (a) 中可以看到 Pose2 和 Pose6 之间的共同观测。并通过共同观测计算两帧图像之间的旋转矩阵和平移矩阵，对于单目相机，还要通过 Sim3 优化尺度。然后在上一步求得的 Sim3 和对应同名点的基础上，纠正了当前帧以及所有有同视关系的关键帧的位姿，以及路标点。

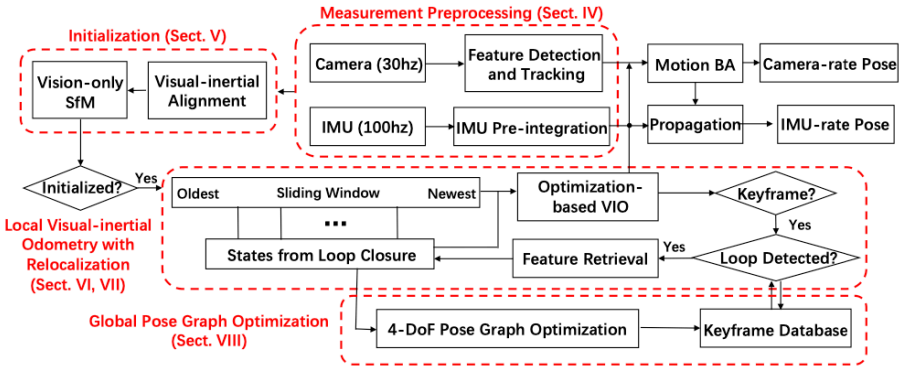


图 48: VINS-Mono 的系统框架。[cite]

13.2 VINS 系列

VINS 系列主要指港科沈绍杰教授团队所提出的 VIO 相关论文和开源系统，其中更以 VINS-Mono 为代表，这篇文章的引用量迅速上升至 1000（Google Scholar, 06.2021），成为了 VIO 领域短期之内避不开的重要论文。这里我们一起来回顾 VINS-Mono 所提出来的经典框架。

13.2.1 系统概述

如图48所示，系统首先分别以 30hz 和 100hz 的帧率输入单目图片和 IMU 信息，使用 KLT 光流算法跟踪每帧图像提取的 Harris 角点，IMU 部分的处理方式是预积分。系统运行起来后，第一步是初始化程序。无论是 VINS 还是后面的 ORB-SLAM3，系统都是先进行纯视觉的初始化，视觉初始化成功之后再进行视觉和 IMU 的共同优化。在追踪过程中，该系统通过使用优化模型，构建包含特征点约束、IMU 预积分约束和闭环检测约束的优化方程，来求解滑窗内所有帧的位置、速度、旋转和 IMU 的 bias。需要注意的是，闭环检测中用到的图像帧是滑窗中选出来的关键帧，通过 DBoW 方法监测到闭环后，对涉及闭环的关键帧进行全局优化。

13.2.2 预积分和视觉约束

前面在介绍 ORB-SLAM 系列的时候，介绍了特征提取

盲源-SLAM中的闭环与后

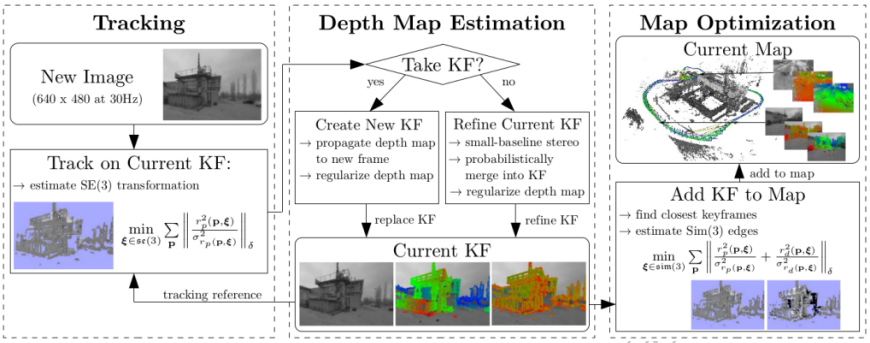


图 49: Pipeline of LSD-SLAM [5]

13.3 LSD-SLAM

LSD-SLAM(large-scale direct monocular SLAM) [5] 是发表在 ECCV2014 的直接法的 SLAM。与 ORB-SLAM 等所用的特征法不同，直接法视觉里程计（VO）是直接利用图像像素点的灰度信息来构图与定位，克服了特征点提取方法的局限性，可以使用图像上的所有信息。该方法在特征点稀少的环境下仍能达到很高的定位精度与鲁棒性，而且提供了更多的环境几何信息，这在机器人和增强现实额应用中都是非常有意义的。

这篇文章的第一作者 Jakob Engel 对直接法的发展作出了很大的贡献。本篇文章提出的方法能够构建大尺度的，全局一致性的环境地图。除了能够基于直接图像配准得到高度准确的姿态估计外，还能够将三维环境地图实时重构为关键帧的姿态图和对应的半稠密的深度图。这些都是通过对大量像素点对之间的基线立体配准结果滤波后得到的。算法提出了计算尺度漂移的公式，即便是当图像序列的场景尺度变化较大时也能够适用。

13.3.1 整体流程

算法有三个主要组成部分，分别为图像跟踪、深度图估计和地图优化。如图 49所示。

- 图像跟踪：连续跟踪从相机获取到的新“图像帧”。也就是说用前一帧图像帧作为初始姿态，估算出当前参考关键帧和新图像帧之间的刚体变换 $\xi \in se(3)$。
- 深度图估计：使用被跟踪的“图像帧”，要么对当前关键帧深度更新，

要么替换当前关键帧。深度更新是基于像素小基线立体配准的滤波方式，同时耦合对深度地图的正则化。如果相机移动足够远，就初始化新的关键帧，并把现存相近的关键帧图像点投影到新建立的关键帧上。

- 地图优化模块：一旦关键帧被当前的图像替代，它的深度信息将不会再被进一步优化，而是通过地图优化模块插入到全局地图中。为了检测闭环和尺度漂移，采用尺度感知的直接图像配准方法来估计当前帧与现有邻近关键帧之间的相似性变换。

13.3.2 光度误差与深度误差

关键帧是 LSD-SLAM 系统中重要的媒介。启动 LSD-SLAM 系统时，只需要初始化首帧关键帧即可，而关键帧深度信息初步设定为一个方差很大的随机变量。在算法运行最开始的几秒钟，一旦摄像头运动了足够的平移量，LSD-SLAM 算法就会“锁定”到某个特定的深度配置，经过几个关键帧的传递之后，就会收敛到正确的深度配置。

$$E(\xi) = \sum_i (I_{ref}(p_i) - I(w(p_i, D_{ref}(p_i), \xi)))^2 \tag{13.3}$$

公式 13.3是当前帧 I 与参考帧 I_{ref} 之间的光度误差。其中 $w(p_i, D_{ref}(p_i), \xi)$ 是将这些像素重投影到参考帧 I_{ref}。一旦新图像帧被选择成为关键帧，把上一帧关键帧的兴趣点投影到新创建的关键帧上得到这一帧的兴趣点。之后，深度图被平均至逆深度为 1，就可以使用相似变换 sim(3) 来计算关键帧之间的边，因为相似变换 sim(3) 可以较好地结合关键帧之间的尺度缩放差异。

除了包含光度测量残差 r_p，该论文还引入深度残差（depth residual）r_d 来惩罚关键帧之间的逆深度偏差，能直接够估计出帧间的相似变换。LSD-SLAM 论文中使用的误差函数 $E_(\xi_{ji})$，

$$E(\xi_{ji}) = \sum || \frac{r_p^2(p, \xi_{ji})}{\sigma^2_{r_p(p,\xi_{ji})}} + \frac{r_d^2(p, \xi_{ji})}{\sigma^2_{r_d(p,\xi_{ji})}} || \tag{13.4}$$

其中 r_d 是深度惨差，$\sigma_{r_p}^{-2}$ 和 $\sigma_{r_d}^{-2}$ 分别是光度残差和深度惨差的逆方差。

13.4 SVO

全称 fast Semi-direct monocular Visual Odometry（半直接视觉里程计），是苏黎世大学机器人感知组的克里斯蒂安. 弗斯特（Christian Forster

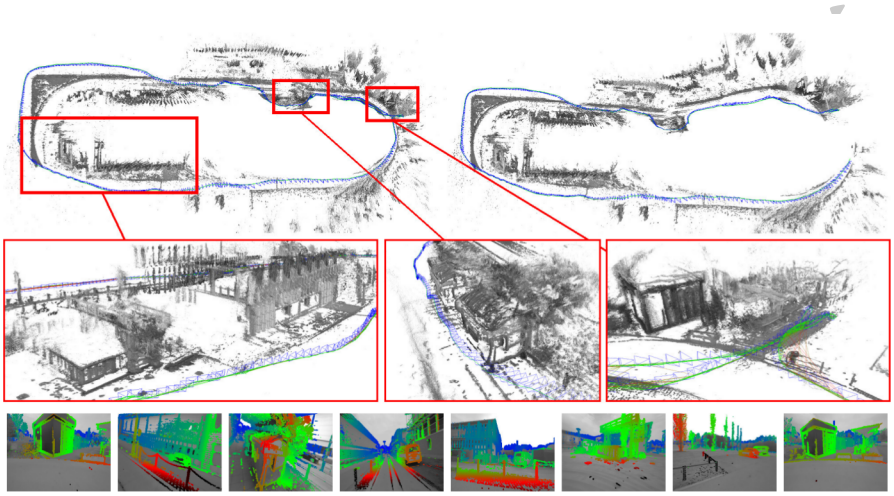


图 50: LSD-SLAM 中的闭环检测模块。左图是闭环检测的效果，右图是没有闭环检测。[5]

等人于 2014 年 ICRA 会议上发表的工作。2016 年扩展了多相机和 IMU 之后，写成期刊论文，称为 SVO 2.0。与基于特征法的 ORB-SLAM 和直接法的 LSD-SLAM 不同，SVO 提取了稀疏点，但是使用直接法来做图像之间的对齐，故称它为半直接法。高翔博士在 LDSO [8] 中同样使用特征点和光度，特征的主要作用是实现闭环检测，只不过现在半稠密的概念已经不如以往流行了。SVO 结合了直接法和特征点法，但是特征点只是从关键帧中提取，以保证系统的运行速度。SVO 算法分成两部分：位姿估计，深度估计。下文详细介绍这两个模块。

13.4.1 位姿估计

对稀疏的特征块使用 direct method 配准，获取相机位姿；通过获取的位姿预测参考帧中的特征块在当前帧中的位置，由于深度估计的不准导致获取的位姿也存在偏差，从而使得预测的特征块位置不准。由于预测的特征块位置和真实位置很近，所以可以使用牛顿迭代法对这个特征块的预测位置进行优化。特征块的预测位置得到优化，说明之前使用直接法预测的有问题。利用这个优化后的特征块预测位置，再次使用直接法，对相机位姿 (pose) 以及特征点位置 (structure) 进行优化。而特征点值从关键帧中提取，文章使用重投影误差来做 pose refinement。

SLAM 十四讲

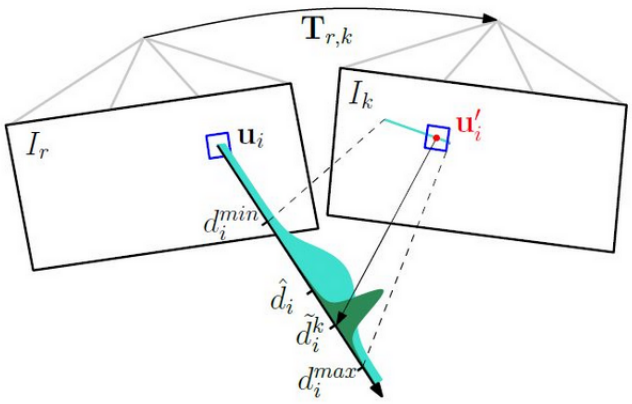


图 51: 深度的高斯-均匀分布。

13.4.2 深度估计

处了 tracking 线程，svo 还有一个 mapping 线程。2d 特征的深度估计是在这个线程中完成的。SVO 中使用高斯-均匀分布对逆深度信息建模。如果当前帧被选为关键帧，就提取关键帧上的新特征点，这些特征点并未有对应的深度信息，因此需要获得计算深度。如果进来一个普通帧，就用普通帧的信息，更新所有种子点的概率分布。于是你从这条极线的一个端点走到另一个端点，把每处的图像块都和参考的去比较，就可以找到正确的匹配。如果某个种子点的深度分布已经收敛，就把它放到地图中，供追踪线程使用。

14 深度 SLAM

深度 SLAM 用来描述融合几何与深度学习的 SLAM 系统。

14.1 Structure-SLAM

该文章提出一个单目 SLAM 系统，使用深度学习网络进行表面法向量（surface normal）估计，然后使用曼哈顿世界（Manhattan World）模型，来估计相机的旋转矩阵，然后利用点线特征估计平移向量。获得旋转、平移矩阵和局部地图之后，通过地图对相机姿态进行一次优化。论文使用 TUM RGB-D 和 ICL NUIM 数据集进行测试。

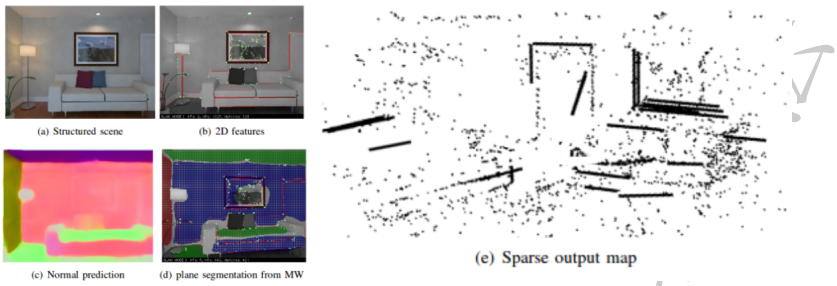


图 52: 效果图

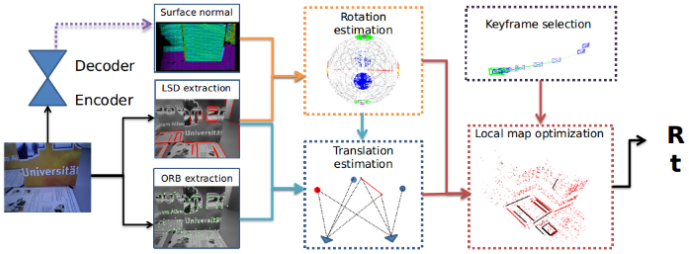


Fig. 2. Proposed SLAM framework (StructureSLAM). In the front-end, the encoder-decoder network predicts dense surface normals. In parallel, point and line features are extracted from the RGB image. In the back-end, first the scene structure in the form of normals and lines is used to estimate the global rotation of the camera. Then, the remaining 3-DoF for the translation are obtained using point and line features. The initial pose estimate is validated and refined using the local map. Keyframes are selected based on the availability of point and line features.

图 53: Structure-SLAM 框架 [19]

2. 基于曼哈顿世界假设的旋转矩阵估计

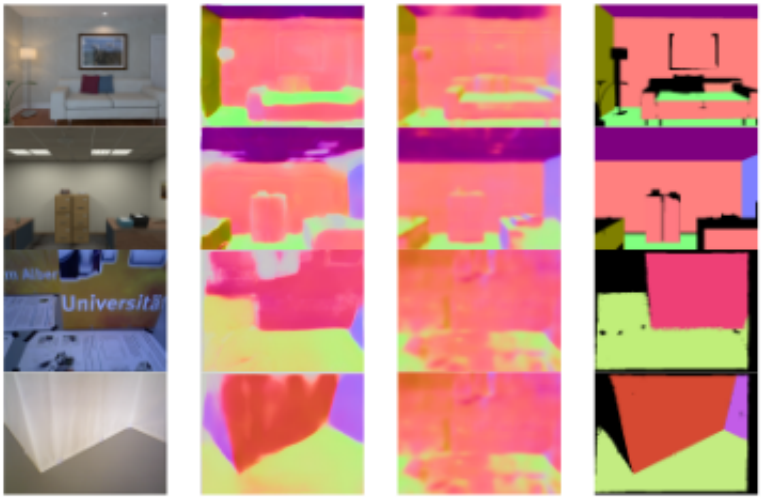
什么是曼哈顿世界假设 曼哈顿假设：如下图所示，环境中存在垂直/正交的信息，如地板、天花板、墙面等。通过判断相机和曼哈顿世界之间的旋转矩阵，进而可以获得相机与相机之间的相对旋转矩阵。

旋转矩阵估计 首先是进行 normal 估计，第四列是使用深度图计算出来的平面分割结果。二三列是网络基于 RGB 图估计出来的结果。

将估计的法向量结果投影在一个单位球（unit sphere）上，如下如所示，一部分法向量会聚集在一起（平行平面），由于估计误差，也会有分散在球面上的法向量。该文章使用聚类方法-mean shift，对球面上的法向量进行聚类。

基于 sphere mean shift 的处理方式如下：

开始假设最开始的 $R_{c_0,M}$ 是单位矩阵，然后我们将法向量投影在 x,y,z



(a) Input (b) Ours (c) GeoNet[14] (d) AHC [36]

Fig. 4. Results of normal prediction on ICL-NUIM (top) and TUM-RGBD (bottom) scenes for different approaches.

图 54: Caption

轴的切平面上，然后在 2D 且平面上进行 mean-shift 聚类：

$$s_n^{'} = \frac{\sum_{in} e^{-c\|m_{in}^{'}\|^2} m_{in}^{'}}{\sum_{in} e^{-c\|m_{in}^{'}\|^2}} \tag{14.1}$$

聚类结果之后，我们把聚类结果投影回 3D 空间 s_n，形成两个坐标系之间的夹角。

$$\hat{r}_n = Q_n s_n \tag{14.2}$$

这里 $Q_n = [r_{mod(n,3)}, r_{mod(n+1,3)}, r_{mod(n+2,3)}]$，其中 $mod()$ 是一个取模操作，经过一个迭代过程，我们可以获得 $R_{C_1M} = [\hat{r}_1, \hat{r}_2, \hat{r}_3]^T$. 这样就更新出来了 $R_{c_{k+1},M}$，也就可以获得 $R_{c_{k+1},w} = R_{c_{k+1},M} * R_{M,w}$

Note 我们获得 $R_{c_{k+1},w}$，但是它不依赖与前面的图像帧或关键帧，而是一个不变的量 $R_{M,w}$. 因此这就是一个很有趣的概念：**drift-free rotation estimation**.

首发SLAM中的人

平移向量估计 由于这里只计算 3 自由度的平移向量，因此能更好的适应 low-textured 的场景。

$$e^p_{k,j} = p_k - \pi(R_{k,j}P_j + t_{k,j}) \tag{14.3}$$

$$e^l_{k,j} = l\pi(R_{k,j}P_x + t_{k,j}) \tag{14.4}$$

这是使用点线特征来完成平移向量的计算。

盖粒-SLAM 中的几何与学习方法

参考文献

[1] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

[2] Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. Orb-slam3: An accurate open-source library for visual, visual-inertial and multi-map slam. *arXiv preprint arXiv:2007.11898*, 2020.

[3] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE international conference on computer vision*, pages 2650–2658, 2015.

[4] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, pages 2366–2374, 2014.

[5] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European conference on computer vision*, pages 834–849. Springer, 2014.

[6] Chen Feng, Yuichi Taguchi, and Vineet R Kamat. Fast plane extraction in organized point clouds using agglomerative hierarchical clustering. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6218–6225. IEEE, 2014.

[7] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. Deep ordinal regression network for monocular depth estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2002–2011, 2018.

[8] Xiang Gao, Rui Wang, Nikolaus Demmel, and Daniel Cremers. Ldso: Direct sparse odometry with loop closure. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2198–2204. IEEE, 2018.

视觉SLAM中的几何与学习方法

[9] Ruben Gomez-Ojeda, Francisco-Angel Moreno, David Zuñiga-Noël, Davide Scaramuzza, and Javier Gonzalez-Jimenez. Pl-slam: a stereo slam system through the combination of points and line segments. *IEEE Transactions on Robotics*, 35(3):734–746, 2019.

[10] Yijia He, Ji Zhao, Yue Guo, Wenhao He, and Kui Yuan. Pl-vio: Tightly-coupled monocular visual–inertial odometry using point and line features. *Sensors*, 18(4):1159, 2018.

[11] Dirk Holz, Stefan Holzer, Radu Bogdan Rusu, and Sven Behnke. Real-time plane segmentation using rgb-d cameras. In *Robot Soccer World Cup*, pages 306–317. Springer, 2011.

[12] Pyojin Kim, Brian Coltin, and H Jin Kim. Linear rgb-d slam for planar environments. In *European Conference on Computer Vision (ECCV)*, pages 333–348, 2018.

[13] Pyojin Kim, Brian Coltin, and H Jin Kim. Visual odometry with drift-free rotation estimation using indoor scene regularities. In *British Machine Vision Conference (BMVC)*, 2017.

[14] Pyojin Kim, Brian Coltin, and H Jin Kim. Low-drift visual odometry in structured environments by decoupling rotational and translational motion. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 7247–7253. IEEE, 2018.

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *2007 6th IEEE and ACM international symposium on mixed and augmented reality*, pages 225–234. IEEE, 2007.

[17] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. In *2016 Fourth international conference on 3D vision (3DV)*, pages 239–248. IEEE, 2016.

北航SLAM中的几何与学习方法

[18] Xin Li, Yanyan Li, Evin Pnar Ornek, Jinlong Lin, and Federico Tombari. Co-planar parametrization for stereo-slam and visual-inertial odometry. *IEEE Robotics and Automation Letters*, 2020.

[19] Yanyan Li, Nikolas Brasch, Yida Wang, Nassir Navab, and Federico Tombari. Structure-slam: Low-drift monocular slam in indoor environments. *IEEE Robotics and Automation Letters*, 5(4):6583–6590, 2020.

[20] Yanyan Li, Shiyue Fan, Yanbiao Sun, Qiang Wang, and Shanlin Sun. Bundle adjustment method using sparse bfgs solution. *Remote Sensing Letters*, 9(8):789–798, 2018.

[21] Yanyan Li, Raza Yunus, Nikolas Brasch, Nassir Navab, and Federico Tombari. Rgb-d slam with structural regularities. *arXiv preprint arXiv:2010.07997*, 2020.

[22] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[23] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[24] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[25] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *2011 international conference on computer vision*, pages 2320–2327. IEEE, 2011.

[26] Albert Pumarola, Alexander Vakhitov, Antonio Agudo, Alberto Sanfeliu, and Francese Moreno-Noguer. Pl-slam: Real-time monocular visual slam with points and lines. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 4503–4508. IEEE, 2017.

北斗SLAM中的几何与学习方法

[27] Xiaojuan Qi, Renjie Liao, Zhengzhe Liu, Raquel Urtasun, and Jiaya Jia. Geonet: Geometric neural network for joint depth and surface normal estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 283–291, 2018.

[28] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[29] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[30] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1746–1754, 2017.

[31] Jean-Philippe Tardif. Non-iterative approach for fast and accurate vanishing point detection. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1250–1257. IEEE.

[32] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassir Navab. Cnn-slam: Real-time dense monocular slam with learned depth prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6243–6252, 2017.

[33] Deepak Geetha Viswanathan. Features from accelerated segment test (fast).

[34] L. von Stumberg, V. Usenko, and D. Cremers. Direct sparse visual-inertial odometry using dynamic marginalization. May 2018.

[35] Xiaoyu Zhang, Wei Wang, Xianyu Qi, Ziwei Liao, and Ran Wei. Point-plane slam using supposed planes for indoor environments. *Sensors*, 19(17):3795, 2019.

北京-SLAM 中的几何与学习方法

[36] Yinda Zhang and Thomas Funkhouser. Deep depth completion of a single rgb-d image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 175–185, 2018.

[37] Yinda Zhang, Shuran Song, Ersin Yumer, Manolis Savva, Joon-Young Lee, Hailin Jin, and Thomas Funkhouser. Physically-based rendering for indoor scene understanding using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5287–5295, 2017.

[38] Yi Zhou, Laurent Kneip, Cristian Rodriguez, and Hongdong Li. Divide and conquer: Efficient density-based tracking of 3d sensors in manhattan worlds. In *Asian Conference on Computer Vision (ACCV)*, pages 3–19. Springer, 2016.

[39] Danping Zou, Yuanxin Wu, Ling Pei, Haibin Ling, and Wenxian Yu. Structvio: visual-inertial odometry with structural regularity of man-made environments. *IEEE Transactions on Robotics*, 35(4):999–1013, 2019.

盖雅-SLAM 中的几何学习方法