

第 2 讲

初识 SLAM

本节目标

1. 理解一个视觉 SLAM 框架由哪几个模块组成，各模块的任务是什么。
2. 搭建编程环境，为开发和实验做准备。
3. 理解如何在 Linux 下编译并运行一个程序。如果它出了问题，我们又如何对它进行调试。
4. 掌握 cmake 的基本使用方法。

本讲概括地介绍一个视觉 SLAM 系统结构，作为后续章节的大纲。实践部分介绍环境搭建、程序基本知识，最后完成一个 Hello SLAM 程序。

Visual Odometry

视觉里程计

后端优化

Optimization

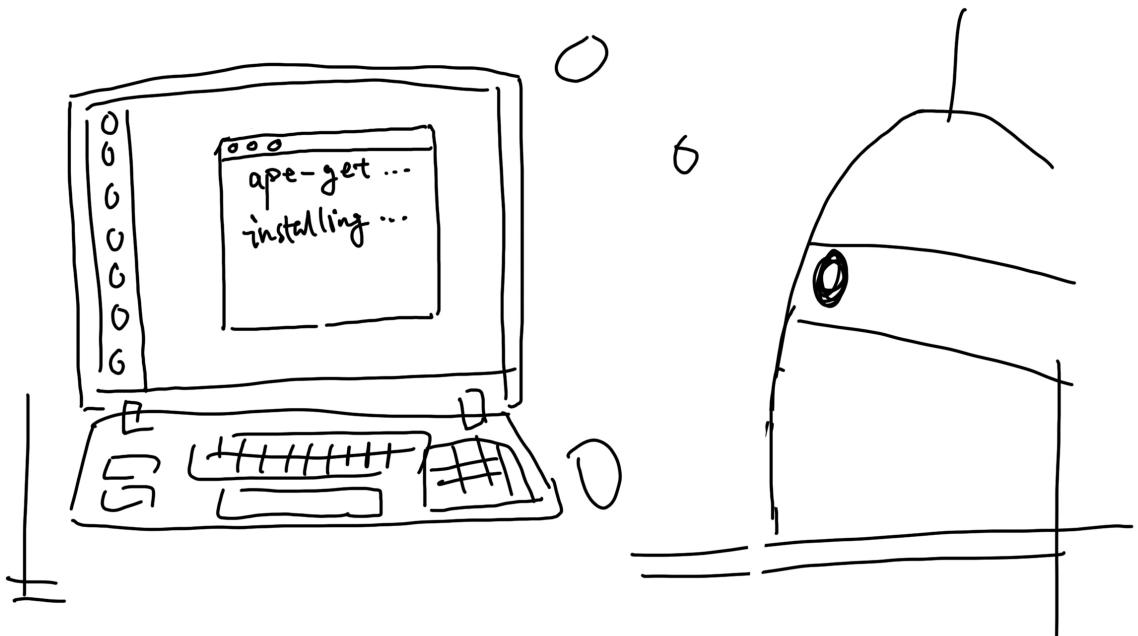
SLAM

圆环检测

Loop closure

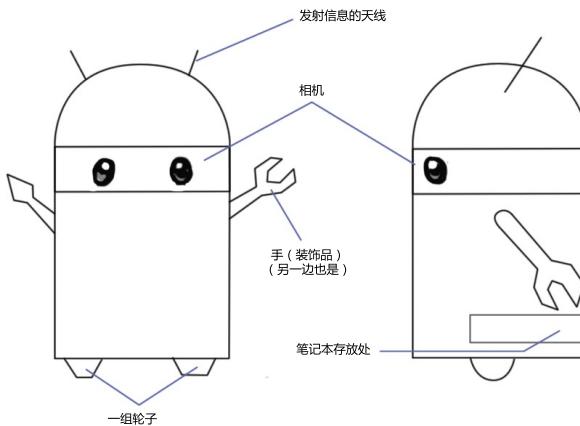
地图构建

Mapping



2.1 引子：小萝卜的例子

我发现上来就给出一列符号，讲“SLAM 的数学定义”这样的方式，会让初学者觉得难以接受。从一个实际的例子谈起会更好一些。假设我们组装了一台叫做“小萝卜”的机器人，大概长的像图 2-1 这个样子。（由于工程制图要求严格一些，所以这张图就不手绘了。）



小萝卜设计图

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

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

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

要移动首先得有轮子和电机，所以我们在小萝卜下方安装了轮子（足式机器人步态很复杂我们暂时不考虑）。有了轮子，机器人就能够四处行动了，但不加控制的话，小萝卜不知道行动的目标，只能四处乱走，更糟糕的情况下会撞上墙损坏自己。为了避免这种情况的发生，我们在它脑袋上安装了一个相机。安装相机的主要动机，是考虑到这样一个机器人和人类非常相似——从画面上一眼就能看出。有眼睛、大脑和四肢的人类，能够在任意环境里轻松自在地行走、探索，我们（天真地）也觉得机器人能够完成这件事。为了使小萝卜能够探索一个房间，它至少需要知道两件事：

1. 我在什么地方？——**定位**。
2. 周围环境是什么样？——**建图**。

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

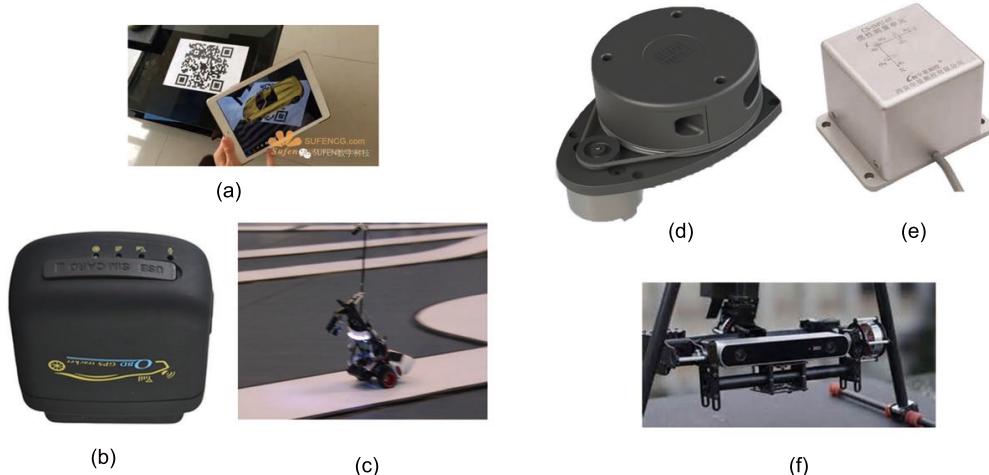


图 2-2 一些传感器的图片。(a) 利用二维码进行定位的增强现实软件；(b) GPS 定位装置；(c) 铺设导轨的小车；(d) 激光雷达；(e) IMU 单元；(f) 双目相机。

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

我们看到，这类传感器**约束了外部环境**。只有在这些约束满足时，基于它们的定位方案才能工作。反之，当约束无法满足时，我们就没法进行定位了。所以说，虽然这类传感器简单可靠，但它们无法提供一个普遍的，通用的解决方案。相对的，那些**携带于机器人本体上的**传感器，比如激光传感器、相机、轮式编码器、惯性测量单元（Inertial Measurement

Unit, IMU) 等等, 它们测到的通常都是一些间接的物理量而不是直接的位置数据。例如, 轮式编码器会测到轮子转动的角度、IMU 测量运动的角速度和加速度, 相机和激光则读取外部环境的某种观测数据。我们只能通过一些间接的手段, 从这些数据推算自己的位置。虽然这听上去是一种迂回战术, 但更明显的好处是, 它没有对环境提出任何要求, 使得这种定位方案可适用于未知环境。

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



图 2-3 形形色色的相机: 单目, 双目和深度相机。

视觉 SLAM 是本书的主题, 所以我们尤其关心小萝卜的眼睛能够做些什么事。SLAM 中使用的相机与我们平时见到的单反摄像头并不是同一个东西。它往往更加简单, 不携带昂贵的镜头, 以一定速率拍摄周围的环境, 形成一个连续的视频流。普通的摄像头能以每秒钟 30 张图片的速度采集图像, 高速相机则更快一些。**按照相机的工作方式, 我们把相机分为单目 (Monocular)、双目 (Stereo) 和深度相机 (RGB-D) 三个大类**, 如图 2-3 所示。直观看来, 单目相机只有一个摄像头, 双目有两个, 而 RGB-D 原理较复杂, 除了能够采集到彩色图片之外, 还能读出每个像素离相机的距离。它通常携带多个摄像头, 工作原理和普通相机不尽相同, 我们会在第五讲详细介绍它们的工作原理, 此处读者只需有一个直观概念即可。此外, SLAM 中还有全景相机 [7]、Event 相机 [8] 等特殊或新兴的种类。虽然偶尔能看到它们在 SLAM 中的应用, 不过到目前为止还没有成为主流。从样子上看, 小萝卜似乎是使用双目的^②。

^①不过实际当中我们都会有一个大概的范围, 例如室内和室外的区分。

^②因为画成单目会比较吓人。

我们分别来看一看各种相机用来做 SLAM 时会有什么特点。

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

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



图 2-4 单目视觉中的尴尬：不知道深度时，手掌上的人是真人还是模型？

由于单目相机只是三维空间的二维投影，所以，如果我们真想恢复三维结构，必须移动相机的视角。在单目 SLAM 中也是同样的原理。我们必须移动相机之后，才能估计它的

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

然而，即使我们知道了物体远近，它们仍然只是一个相对的值。想象我们在看电影时候，虽然能够知道电影场景中哪些物体比另一些大，但我们无法确定电影里那些物体的“真实尺度”：那些大楼是真实的高楼大厦，还是放在桌上的模型？而摧毁大厦的是真实怪兽，还是穿着特摄服装的演员？直观地说，如果把相机的运动和场景大小同时放大两倍，单目所看到的像是一样的。同样的，把这个大小乘以任意倍数，我们都将看到一样的景象。这说明了单目 SLAM 估计的轨迹和地图，将与真实的轨迹、地图，相差一个因子，也就是所谓的尺度 (Scale)^①。由于单目 SLAM 无法仅凭图像确定这个真实尺度，所以又称为尺度不确定性。

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



图 2-5 双目相机的数据：左眼图像，右眼图像。通过左右眼的差异，能够判断场景中物体离相机距离。

双目相机 (Stereo) 和深度相机 双目相机和深度相机的目的，在于通过某种手段测量物体离我们的距离，克服单目无法知道距离的缺点。如果知道了距离，场景的三维结构就可以通过单个图像恢复出来，也就消除了尺度不确定性。尽管都是为测量距离，但双目相机与深度相机测量深度的原理是不一样的。双目相机由两个单目相机组成，但这两个相机之间的距离（称为基线 (Baseline)）是已知的。我们通过这个基线来估计每个像素的空间位置——这和人眼非常相似。我们人类可以通过左右眼图像的差异，判断物体的远近，在计

^① 数学上的原因将会在视觉里程计章节中解释。

计算机上也是同样的道理。如果对双目相机进行拓展，也可以搭建多目相机，不过本质上并没有什么不同。

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



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

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

我们讨论了几种常见的相机，相信通过以上的说明，你应该对它们有了直观的理解。现

在，想象相机在场景中运动的过程，我们将得到一系列连续变化图像^①。视觉 SLAM 的目标，是通过这样的一些图像，进行定位和地图构建。这件事情并没有我们想象的那么简单。它不是某种算法，只要我们输入数据，就可以往外不断地输出定位和地图信息了。SLAM 需要一个完善的算法框架，而经过研究者们长期的研究工作，现有这个框架已经定型了。

2.2 经典视觉 SLAM 框架

下面我们来看经典的视觉 SLAM 框架，了解一下视觉 SLAM 究竟有哪几个模块组成，如图 2-7 所示。

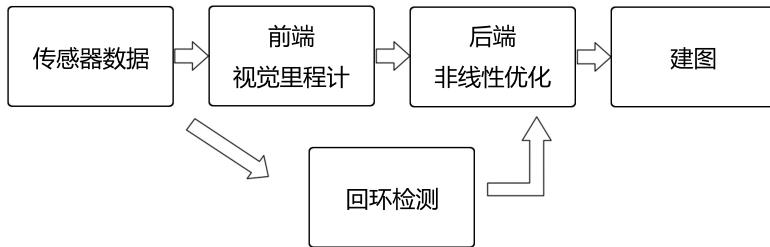


图 2-7 整体视觉 SLAM 流程图。

我们把整个视觉 SLAM 流程分为以下几步：

1. 传感器信息读取。在视觉 SLAM 中主要为相机图像信息的读取和预处理。如果在机器人中，还可能有码盘、惯性传感器等信息的读取和同步。
2. 视觉里程计 (Visual Odometry, VO)。视觉里程计任务是估算相邻图像间相机的运动，以及局部地图的样子。VO 又称为前端 (Front End)。
3. 后端优化 (Optimization)。后端接受不同时刻视觉里程计测量的相机位姿，以及回环检测的信息，对它们进行优化，得到全局一致的轨迹和地图。由于接在 VO 之后，又称为后端 (Back End)。
4. 回环检测 (Loop Closing)。回环检测判断机器人是否曾经到达过先前的位置。如果检测到回环，它会把信息供给后端进行处理。
5. 建图 (Mapping)。它根据估计的轨迹，建立与任务要求对应的地图。

经典的视觉 SLAM 框架是过去十几年内，研究者们总结的成果。这个框架本身，以及它所包含的算法已经基本定型，并且在许多视觉程序库和机器人程序库中已经提供。依靠这些算法，我们能够构建一个视觉 SLAM 系统，使之在正常的工作环境里实时进行定位与

^①你可以用手机录个小视频试试。

建图。因此，我们说，如果把工作环境限定在静态、刚体，光照变化不明显、没有人为干扰的场景，那么，这个 SLAM 系统是相当成熟的了 [9]。

读者可能还没有理解上面几个模块的概念，我们就来详细讲一讲各个模块具体的任务。但是，理性地理解它们的工作原理需要一些数学知识，我们放到本书的第二部分再进行。这里我们希望读者对各模块有一个直观的、定性的理解即可。

2.2.1 视觉里程计

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

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



图 2-8 相机拍摄到的图片与人眼反应的运动方向。

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

要讲清这个几何关系以及 VO 的实现方法，需要铺垫一些背景知识。我们在这里先让读者对 VO 有直观的概念，理性的理解需要在背景知识铺垫完成之后才能深入讨论。所以，现在我们只需知道，VO 能够通过相邻帧间的图像估计相机运动，并恢复场景的空间结构。

叫它为“里程计”是因为它和实际的里程计一样，只计算相邻时刻的运动，而和再往前的过去的信息没有关联。在这一点上，VO 就像一种只有很短时间记忆的物种一样。

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

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

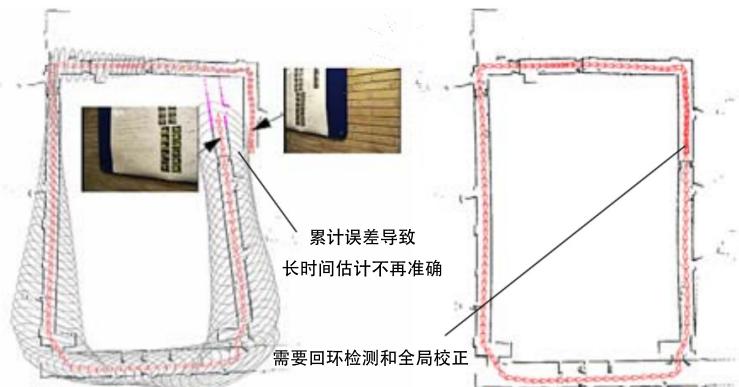


图 2-9 累计误差与回环检测的校正结果 [10]。

这也就是所谓的漂移 (Drift)。它将导致我们无法建立一致的地图。你会发现原本直的走廊变成了斜的，而原本 90 度的直角变成了歪的——这实在是一件很难令人忍受的事情！为了解决漂移问题，我们还需要两种技术：后端优化^①和回环检测。回环检测负责把“机器人回到原始位置”的事情检测出来，而后端优化则根据该信息，校正整个轨迹的形状。

^①更多时候称为后端 (Back End)。由于主要使用的是优化方法，故称为后端优化。

2.2.2 后端优化

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

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

从历史意义上来说，现在我们称之为后端优化的部分，很长一段时间直接被称为“SLAM 研究”。早期的 SLAM 问题是一个状态估计问题——正是后端优化要解决的东西。在最早提出 SLAM 的一系列论文中，当时的人们称它为“空间状态不确定性的估计”(Spatial Uncertainty) [4, 11]。虽然有一些晦涩，但也确实反映出了 SLAM 问题的本质：对运动主体自身和周围环境空间不确定性的估计。为了解决 SLAM，我们需要状态估计理论，把定位和建图的不确定性表达出来，然后采用滤波器或非线性优化，去估计状态的均值和不确定性（方差）。状态估计与非线性优化的具体内容将在第六章和十、十一两章介绍。让我们暂时跳过它的原理，继续往下说明。

2.2.3 回环检测

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

回环检测与“定位”和“建图”二者都有密切的关系。事实上，我们认为，地图存在的主要意义，是为了让机器人知晓自己到达过的地方。**为了实现回环检测，我们需要让机器人具有识别曾到达过的场景的能力。**它的实现手段有很多。例如像前面说的那样，我们可以在机器人下方设置一个标志物（如一张二维码图片）。只要它看到了这个标志，就知道自己回到了原点。但是，该标志物实质上是一种环境中的传感器，对应用环境提出了限

制（万一不能贴二维码怎么办呢？）。我们更希望机器人能使用携带的传感器——也就是图像本身，来完成这一任务。例如，我们可以判断图像间的相似性，来完成回环检测。这一点和人是相似的。当我们看到两张相似图片时，容易辨认它们来自同一个地方。如果回环检测成功，可以显著地减小累积误差。所以视觉回环检测，实质上是一种计算图像数据相似性的算法。由于图像的信息非常丰富，使得正确检测回环的难度也降低了不少。

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

2.2.4 建图

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

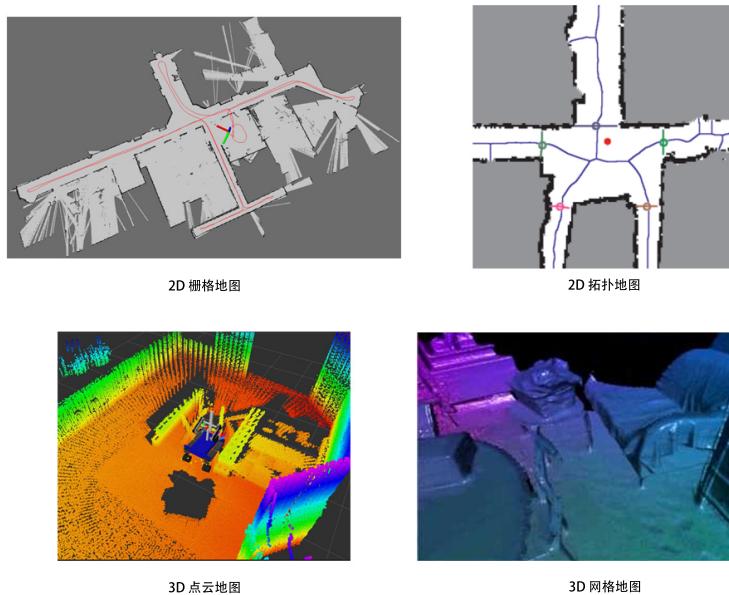


图 2-10 形形色色的地图：2D 棚格地图、拓扑地图以及 3D 点云地图和网格地图 [12]。

对于家用扫地机器人来说，这种主要在低矮平面里运动的机器人，只需要一个二维的地图，标记哪里可以通过，哪里存在障碍物，就够它在一定范围内导航了。而对于一个相机，它有六自由度的运动，我们至少需要一个三维的地图。有些时候，我们想要一个漂亮的重建结果，不仅是一组空间点，还需要带纹理的三角面片。另一些时候，我们又不关心

地图的样子，只需要知道“ A 点到 B 点可通过，而 B 到 C 不行”这样的事情。甚至，有时我们不需要地图，或者地图可以由其他人提供，例如行驶的车辆往往可以得到已经绘制好的当地地图。

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

度量地图（Metric Map）

度量地图强调精确地表示地图中物体的位置关系，通常我们用稀疏（Sparse）与稠密（Dense）对它们进行分类。稀疏地图进行了一定程度的抽象，并不需要表达所有的物体。例如，我们选择一部分具有代表意义的东西，称之为路标（Landmark），那么一张稀疏地图就是由路标组成地图，而不是路标的部分就可以忽略掉。相对的，稠密地图着重于建模所有看到的东西。对于定位来说，稀疏路标地图就足够了。而用于导航时，我们往往需要稠密的地图（否则撞上两个路标之间的墙怎么办？）。稠密地图通常按照某种分辨率，由许多个小块组成。二维度量地图是许多个小格子（Grid），三维则是许多小方块（Voxel）。一般地，一个小块含有占据、空闲、未知三种状态，以表达该格内是否有物体。当我们查询某个空间位置时，地图能够给出该位置是否可以通过的信息。这样的地图可以用于各种导航算法，如 A*, D*^① 等等，为机器人研究者们所重视。但是我们也看到，这种地图需要存储每一个格点的状态，耗费大量的存储空间，而且多数情况下地图的许多细节部分是无用的。另一方面，大规模度量地图有时会出现一致性问题。很小的一点转向误差，可能会导致两间屋子的墙出现重叠，使得地图失效。

拓扑地图（Topological Map）

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

^①https://en.wikipedia.org/wiki/A*_search_algorithm

2.3 SLAM 问题的数学表述

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

假设小萝卜正携带着某种传感器在未知环境里运动，怎么用数学语言描述这件事呢？首先，由于相机通常是在某些时刻采集数据的，所以我们也只关心这些时刻的位置和地图。这就把一段连续时间的运动变成了离散时刻 $t = 1, \dots, K$ 当中发生的事情。在这些时刻，用 \mathbf{x} 表示小萝卜自身的位置。于是各时刻的位置就记为 $\mathbf{x}_1, \dots, \mathbf{x}_K$ ，它们构成了小萝卜的轨迹。地图方面，我们设地图是由许多个路标（Landmark）组成的，而每个时刻，传感器会测量到一部分路标点，得到它们的观测数据。不妨设路标点一共有 N 个，用 $\mathbf{y}_1, \dots, \mathbf{y}_N$ 表示它们。

在这样设定中，“小萝卜携带着传感器在环境中运动”，由如下两件事情描述：

1. **什么是运动？** 我们要考虑从 $k - 1$ 时刻到 k 时刻，小萝卜的位置 \mathbf{x} 是如何变化的。
2. **什么是观测？** 假设小萝卜在 k 时刻，于 \mathbf{x}_k 处探测到了某一个路标 \mathbf{y}_j ，我们要考虑这件事情是如何用数学语言来描述的。

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

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k, \mathbf{w}_k). \quad (2.1)$$

这里 \mathbf{u}_k 是运动传感器的读数（有时也叫输入）， \mathbf{w}_k 为噪声。注意到，我们用一个一般函数 f 来描述这个过程，而不具体指明 f 的作用方式。这使得整个函数可以指代任意的运动传感器，成为一个通用的方程，而不必限定于某个特殊的传感器上。我们把它称为 **运动方程**。

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

$$\mathbf{z}_{k,j} = h(\mathbf{y}_j, \mathbf{x}_k, \mathbf{v}_{k,j}). \quad (2.2)$$

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

读者或许会说，我们用的函数 f, h ，似乎并没有具体地说明运动和观测是怎么回事？同时，这里的 x, y, z 又是什么东西呢？事实上，根据小萝卜的真实运动和传感器的种类，存在着若干种参数化方式（Parameterization）。什么叫参数化呢？举例来说，假设小萝卜在平面中运动，那么，它的位姿^①由两个位置和一个转角来描述，即 $\mathbf{x}_k = [x, y, \theta]_k^T$ 。同时，运动传感器能够测量到小萝卜在每两个时间间隔位置和转角的变化量 $\mathbf{u}_k = [\Delta x, \Delta y, \Delta \theta]_k^T$ ，那么，此时运动方程就可以具体化为：

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_k = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{k-1} + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \theta \end{bmatrix}_k + \mathbf{w}_k. \quad (2.3)$$

这是简单的线性关系。不过，并不是所有的传感器都直接能测量出位移和角度变化，所以也存在着其他形式更加复杂的运动方程，那时我们可能需要进行动力学分析。关于观测方程，比方说小萝卜携带着一个二维激光传感器。我们知道激光传感器观测一个 2D 路标点时，能够测到两个量：路标点与小萝卜本体之间的距离 r 和夹角 ϕ 。我们记路标点为 $\mathbf{y} = [p_x, p_y]^T$ （为保持简洁，省略了下标），观测数据为 $\mathbf{z} = [r, \phi]^T$ ，那么观测方程就具体化为：

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

考虑视觉 SLAM 时，传感器是相机，那么观测方程就是“对路标点拍摄后，得到了图像中的像素”的过程。这个过程牵涉到相机模型的描述，将在第五讲中详细介绍，这里暂时不细谈。

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

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

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

^①在本书中，我们以“位姿”这个词表示“位置”加上“姿态”。

状态估计问题的求解，与两个方程的具体形式，以及噪声服从哪种分布有关。我们按照运动和观测方程是否为线性，噪声是否服从高斯分布进行分类，分为线性/非线性和高斯/非高斯系统。其中线性高斯系统（Linear Gaussian, LG 系统）是最简单的，它的无偏的最优估计可以由卡尔曼滤波器（Kalman Filter, KF）给出。而在复杂的非线性非高斯系统（Non-Linear Non-Gaussian, NLNG 系统）中，我们会使用以扩展卡尔曼滤波器（Extended Kalman Filter, EKF）和非线性优化两大类方法去求解它。直至 21 世纪早期，以 EKF 为主的滤波器方法占据了 SLAM 中的主导地位。我们会在工作点处把系统线性化，并以预测——更新两大步骤进行求解（见第九讲）。最早的实时视觉 SLAM 系统即是基于 EKF[2] 开发的。随后，为了克服 EKF 的缺点（例如线性化误差和噪声高斯分布假设），人们开始使用粒子滤波器（Particle Filter）等其他滤波器，乃至使用非线性优化的方法。时至今日，主流视觉 SLAM 使用以图优化（Graph Optimization）为代表的优化技术进行状态估计 [13]。我们认为优化技术已经明显优于滤波器技术，只要计算资源允许，我们通常都偏向于使用优化方法（见第十、十一讲）。

相信读者应该对 SLAM 的数学模型有了大致的理解，然而我们仍需澄清一些问题。首先，我们要说明机器人位置 x 是什么。我们方才说位置是有些模糊的。也许读者能够理解在平面中运动的小萝卜，可以用两个坐标加一个转角的形式将位置参数化。然而，虽然我的漫画风格有些二次元，小萝卜更多时候是一个三维空间里的机器人。我们知道三维空间的运动由三个轴构成，所以小萝卜的运动要由三个轴上的平移，以及绕着三个轴的旋转来描述，这一共有六个自由度。那是否意味着我随便用一个 \mathbb{R}^6 中的向量就能描述它了呢？我们将发现事情并没有那么简单。对六自由度的位姿^①，如何表达它，如何优化它，都需要一定篇幅来介绍，这将是第三讲和第四讲的主要内容。随后，我们要说明在视觉 SLAM 中，观测方程如何参数化。换句话说，空间中的路标点是如何投影到一张照片上的。这需要解释相机的成像模型，我们将在第五章介绍。最后，当我们知道了这些信息，怎么求解上述方程？这需要非线性优化的知识，则是第六讲的内容。

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

^① 我们以后称它为位姿（Pose），以与位置进行区别。我们说的位姿，包含了旋转（Rotation）和平移（Translation）。