SLAM14讲—02初识SLAM

2.1 引子

传感器分类：一类传感器是携带于机器人本体上的，例如机器人的轮式编码器、相机、激光等（测量的通常是间接的物理量，对环境没有任何要求，**适用于未知环境**）；另一类是安装于环境中的，例如导轨、二维码标志等（此类约束了外部环境，当约束无法满足时，就无法进行定位；即**不能提供给一个普遍的、通用的解决方案**）。

相机分类：单目（Monocular）、双目（Stereo）、深度相机（GEB-D）。
其中深度相机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。

1.单目相机

拍摄的照片本质上是拍照时的场景在相机的成像平面上留下一个**投影，以二维的形式反映了三维的世界**。丢失了场景的**深度**，我们无法通过单个图片来计算场景中物体离我们的**距离感（空间感）**。
我们必须移动相机才能估计物体的**运动和结构**（远近和大小）。当相机移动时，物体在图像上的运动，形成了**时差**，我们可以借此定量判断物体的远近。
单目SLAM估计的轨迹和地图与真实的轨迹和地图相差一个因子，即尺度。 故单目SLAM存在**尺度不确定性**。
==>单目相机的缺陷：平移之后才能计算深度+无法确定真实尺度

2.双目相机和深度相机

均通过某种首段测量物体离我们的距离，克服单目无法知道距离的缺点。

- 双目相机
  由两个相机组成，两者之间的距离（即**基线**）已知。通过左右图像的差异来判断物体的远近。基线距离越大，测量到的就越远。双目相机的距离估计不依赖其他设备，室内外均可使用。
    - 双目相机的缺陷：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，且时差的计算非常消耗资源，需要GPU和FPGA设备加速才能实时输出。
- 深度相机
  通过红外结构光或Time-of-Flight(ToF)原理，通过主动向物体发射光并接受返回的光，测出物体离相机的距离。节省计算量，主要用于室内。
    - 深度相机的缺陷：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问题。

2.2经典视觉SLAM框架

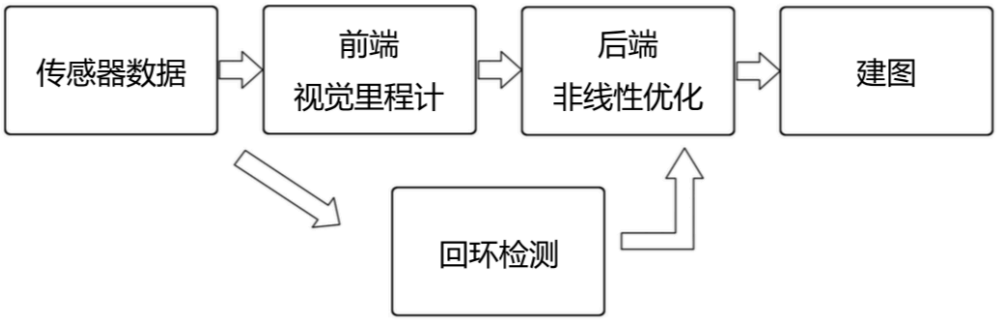

图 2-7　整体视觉 SLAM 流程图。

1.传感器信息读取。主要为相机图像信息的读取和预处理等。
2.前端：视觉里程计（**Visual Odometry，VO**）。估算相邻图像间相机的运动，以及局部地图的样子。
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。
4.回环检测（**Loop Closing**）。检测机器人是否曾经到达过先前位置。
5.建图（**Mapping**）。
在工作环境限定在静态、刚体，光照变化不明显、没有认为干扰的场景，这个SLAM系统是相当成熟的。

2.2.1视觉里程计

视觉里程计不可避免会产生累计漂移（**Accumulating Drift**），这是由于视觉里程计只估计两个图像间运动造成的。每次的估计都有一定的误差，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，一段时间后估计将不在准确。
为了解决漂移问题，我们需要后端优化和回环检测。回环检测负责把“机器人回到原始位置”的事情检测出来，而后端优化则根据该信息，校正整个轨迹的形状。

2.2.2后端优化

笼统地说，后端优化主要指处理SLAM过程中噪声的问题。后端优化要考虑的问题，是如何从这些带有噪声的数据中，估计整个系统的状态（包括机器人自身的轨迹及地图），以及这个状态估计的不确定性有多大——即最大后验概率估计（Maximum-a-Posterior,MAP)。
在视觉SLAM中，前端和计算机视觉研究领域相关，比如图像的特征提取与匹配，后端则主要是滤波和非线性优化算法。
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）。

2.2.3回环检测

主要解决位置估计随时间漂移的问题。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。
实现方法：①对应用环境提出限制：如在机器人下方设置一个标志物（如二维码图片），只要它看到了这个标志，就知道自己回到了远点。②使用携带的传感器——即图像本身来完成：如判断图像间的相似性来完成回环检测。
回环检测实际上是一种计算图像数据相似性的算法。若我们有充分且正确的回环检测，就可以消除累计误差，得到全局一致的轨迹和地图。

2.2.4建图

地图的分类：度量地图和拓扑地图。

- 度量地图
  度量地图强调精确地表示地图中物体的位置关系，通常用稀疏（Sparse）与稠密（Dense）对其分类。

稀疏地图进行了一定程度的抽象，并不需要表达所有的物体，例如我们可以选择一部分具有代表意义的东西——路标，而直接忽略不是路标的部分。可用于定位。
稠密地图着重于建模所有看到的东西。可用于导航。稠密地图通常按照某种分辨率，由许多小块组成（二位是小个子，三维是小方块），每一个小块含有占据、空闲、未知三个状态，以表达该格内是否有物体。
    - 缺陷：存储每一个格点的状态会耗费大量的存储空间，且多数情况下地图的许多细节部分是无用的。另外，大规模度量地图有时会出现一致性问题，即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
拓扑地图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，而不考虑路程。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。
    - 缺陷：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，仍有待考究。

2.3SLAM问题的数学表达

数学语言的描述：

- 离散时刻t=1,...,K
- $\mathbf{x}$表示机器人自身的位置，各时刻的位置记为$\mathbf{x}_1,...,\mathbf{x}_K$
- 地图由路标组成，每个时刻传感器会测量到一部分路标点得到观测数据，设共有N个路标点，用$\mathbf{y}_1,...,\mathbf{y}_N$来表示。

在这样设定中，“机器人携带者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，机器人的位置$\mathbf{x}$是如何变化的。
2. 什么是观测？假设机器人在k时刻，于是$\mathbf{x}_k$处探测到了某一个路标$\mathbf{y}_j$，我们要考虑这件事情如何用数学语言来描述。

运动方程

$\mathbf{x}k = f(\mathbf{x}_{k-1},\mathbf{u}_k,\mathbf{w}_k)$（2.1）
这里$\mathbf{u}k$是运动传感器的读数（有时又叫做输入），$\mathbf{w}k$为噪声。

观测方程

观测方程描述的事，当机器人在$\mathbf{x}_k$位置上看到某个路标点$\mathbf{y}_j$，产生了一个观测数据$\mathbf{z}_{k,j}$。可用以下函数h来描述：
$\mathbf{z}_{k,j} = h(\mathbf{y}_j, \mathbf{x}_k, \mathbf{v}_{k,j})$（2.2）
这里$\mathbf{v}_{k,j}$是这次观测的噪声。

参数化

例1：假设机器人在平面中运动，那么它的位姿由两个位置和一个转角来描述，即$\mathbf{x}_k = [x,y,\theta]^T_k$。同时，运动传感器能够测量到机器人在每两个时间间隔位置和转角的变化量$\mathbf{u}_k = [\triangle x,\triangle y,\triangle \theta]^T_k$，那么，此时运动方程可以具体化为：

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

不是所有的传感器都能直接测量出位移和角度变化，还可能存在其他形式更加复杂的运动方程，需要进行动力学分析。
例2：机器人携带一个二维激光传感器，在观测一个2D路标点时，能够测量到：路标点和机器人本体之间的距离r和夹角φ。记路标点为$\mathbf{y} = [p_x, p_y]^T$，观测数据为$\mathbf{z} = [r, \varphi]^T$，那么观测方程就具体化为：

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

SLAM过程可总结为两个基本方程

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

SLAM问题建模成为一个**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？
该问题的求解，与两个方程的具体形式，以及噪声服从那种分布有关。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**系统分布。

- 线性高斯系统（Linear Gaussian,LG系统）
  最为简单，其无偏最优估计可以由卡尔曼滤波器（Kalman Filter，KF）给出。
- 非线性非高斯系统（Non-Linear Non-Gaussian，NLNG系统）
  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求解

2.4实践：编程基础(具体程序详见/home/luyangsiyi/slambook/ch2)

2.4.1安装Linux操作系统

2.4.2Hello SLAM

1. 编写helloSLAM.cpp文件；
2. 终端输入下述代码进行编译：

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

3.运行程序

```
./a.out
```

2.4.3使用cmake

在一个cmake工程中，用cmake命令生程一个makefile文件，然后用make命令，根据这个makefile文件的内容编译整个工程。
建立CMakeLists.txt文件以便cmake：

```
#声明要求的camke最低版本
cmake_minimum_required(VERSION 2.8)
#申明一个cmake工具
project (HelloSLAM)

```

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

然后在根目录输入以下代码：

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

2.4.4使用库

1. 书写libHelloSLAM.cpp
2. 在CMakeLists.txt中添加如下代码

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

该命令即告诉cmake将文件编译成一个叫“hello”的库，最后会生成libhello.a的静态库
3. 若要生成共享库，则使用如下命令

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

输出文件为libhello_shared.so
静态库每次被调用都会生成一个副本，而共享库则只有一个副本，更省空间
4. 为了让别人使用这个库，还需要提供头文件
编写头文件libHelloSLAM.cpp
5. 编写可执行程序useHello.cpp调用该头文件，开头代码为：

```
#include "libHelloSLAM.h"
```

6.在CMakeLists.txt中添加一个可执行程序的生成命令，简介到刚才的共享库上

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

总结上述工作：
1. 首先，程序代码由头文件和源文件组成；
2. 带有main函数的源文件编译成可执行程序，其他的编译成库文件
3.如果可执行程序想调用库文件中的函数，它需要参考该库提供的头文件，以明白调用的格式。同时，要把可执行程序链接到库文件上。

2.4.5使用IDE

具体方法可参照
https://blog.csdn.net/lch_vison/article/details/79112450