视觉 SLAM 笔记 *

易鹏

中山大学

内部版本号：V1.12.028 (内测版)

2022 年 9 月 29 日

*本笔记已经开源，可以免费下载，github 地址：https://github.com/Lovely-XPP/Notebook

gitee 分流地址：https://gitee.com/sysu_xpp/Notebook

目录

第 1 章　SLAM 简介

1.1　SLAM 的目标

SLAM 的全称为 Simultaneous Localization and Mapping，中文翻译为即时定位于建图，所以 SLAM 的目标就是两个：

(1) 我在什么地方？——**定位**

(2) 周围环境怎么样？——**建图**

1.2　视觉 SLAM

1.2.1　SLAM 传感器

机器的定位有两类传感器。

(1) **机器本身携带**：轮式编码器、相机、激光传感器、IMU 等

(2) **安装于环境**：导轨、二维码标识等

由于环境中的传感器受限于环境的条件，而 GPS 在室内没有信号，SLAM 就是为了解决在任意未知环境中进行定位。这里主要讲视觉 SLAM 的工作。

1.2.2　视觉 SLAM 的传感器

视觉 SLAM 主要的传感器就是**相机**，照片本质上是场景在相机的成像平面上的**投影**。它以**二维的形式记录了三维的世界**，在这个过程中丢掉了场景的一个维度：**深度**（距离）。相机主要有三类：

(1) **单目相机**

- 组成：只使用一个摄像头。
- 原理：在单张图像中，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。所以如果要恢复三维结构，只能改变相机的视角。所以在单目 SLAM 中，我们必须移动相机才能估计它的**运动**（Motion），同时估计场景中物体的远近和大小，称为**结构**（Structure）。从图像的变化可以得到相机的相对运动状态，同时我们还知道：**近处的物体移动块，远处的物体移动慢，极远处（无穷远处）的物体（如太阳、月亮）看上去是不动的**。所以物体在图像上的运动就形成了**视差**（Disparity），通过视差就可以定量判断物体的距离远近。
- 缺点：由于单张图像无法确定深度，所以得到的物体远近仅仅是一个相对值。也就是说，如果把相机的运动和场景放大同样的倍数，单目相机得到的像是一样的。所以，单目 SLAM 估计的轨迹和地图将与真实的

轨迹和地图相差一个因子，即**尺度**（Scale）。由于单目 SLAM 无法仅仅通过图像确定真实尺度，所以又称为**尺度不确定性**（Scale Ambiguity）。

(2) **双目相机**

- 目的：通过某种手段测量物体与相机之间的距离，克服单目相继无法知道距离的缺点。
- 组成：两个单目相机，且两个相机之间的距离（**基线**）已知。
- 原理：和人眼相似，通过左右眼图像的差异判断物体的远近，具体定量用基线确定。
- 缺点：需要大量计算才能近似（不太可靠）估计每一个像素点的深度，且深度的量程和精度受基线和分辨率所限，视差的计算需要消耗大量的计算资源。
- 应用：室内室外

(3) **深度相机**

- 目的：通过某种手段测量物体与相机之间的距离，克服单目相继无法知道距离的缺点。
- 组成：深度相机又称 **RGB-D 相机**，由激光传感器等组成。
- 原理：通过红外结构光或 Time-of-Flight（ToF）原理，像激光传感器那样，主动向物体发射并接受返回的光，测出物体与相机的距离，此为通过物理测量手段进行测量，可以节省大量的计算资源。
- 缺点：测量范围窄、噪声大、视野小、易受日光干扰、无法测量透射物质。
- 应用：室内

1.3 经典的视觉 SLAM 框架

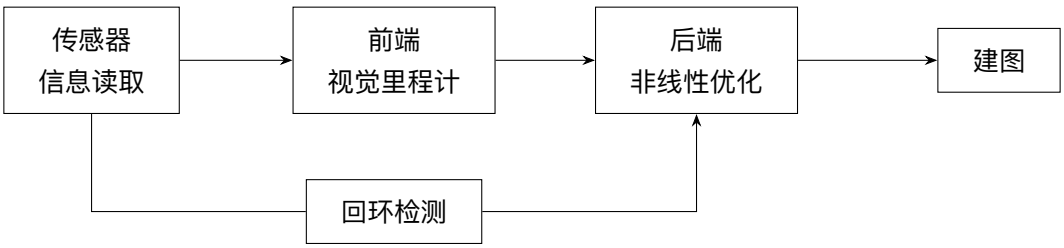

图 1.1: 经典的视觉 SLAM 框架

如图 1.1 所示，经典的视觉 SLAM 框架包括以下几个流程。

1. **传感器信息读取**：相机图像信息的读取和预处理。在机器人中，还可能包括码盘、惯性传感器等信息的读取和同步。
2. **前端视觉里程计**（Visual Odometery，**VO**）：估算相邻图像间相机运动，以及局部地图的样子。VO 又称为**前端**（Front End）。
3. **后端非线性优化**（Optimization）：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 Closure Detection）：回环检测判断机器人是否到达过先前的位置。如果检测到回环，它会把信息提供给后端进行处理。
5. **建图**：根据估计的轨迹，建立与任务要求对应的地图。

在静态、刚体、光照变化不明显、没有人为干扰的场景下，视觉 SLAM 技术已经相当成熟。

1.3.1 视觉里程计

视觉里程计是通过**相邻帧间的图像估计相机运动**，再通过**相机与空间点的几何关系**来估计当前时刻的相机运动，然后将相邻时刻的运动“串起来”，可以估计任意时刻的相机运动并恢复整个场景的空间结构。它称为“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（不限于 2 帧，可以是 5 ~ 10 帧），而和过去的信息没有关联。

但是，由于视觉里程计在最简单的情况下只估计极少量图像间的运动，每个时刻的估计都产生一定的误差，导致先前的误差会传递到下一时刻，误差会随时间不断累积，这就是**累积漂移**（Accumulating Drift）。这将导致无法建立一致的地图。为了解决漂移问题，我们还需要后端优化和回环检测。回环检测负责吧“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。

1.3.2 后端优化

后端优化主要指助理 SLAM 过程中的**噪声**问题。任何传感器都会带一定的噪声，为此我们需要估计传感器带有多大的噪声，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，以及这个状态轨迹的不确定性有多大——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。

前端给后端提供待优化的数据，后端只关心数据而不关心数据来自哪里。**在视觉 SLAM 中，前端和计算机视觉研究更为相关，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算法**。

后端：借助状态估计理论，把定位和建图的不确定性表达出来，然后采用滤波器或非线性优化，估计状态的均值和不确定性（方差）。

1.3.3 回环检测

回环检测又称闭环检测，主要解决**位置估计随时间漂移**的问题。这需要让机器人具有**识别到过的场景**的能力。视觉 SLAM 一般采用**判断图像间的相似性**的方法来完成回环检测。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。机器人判断回到原点后，将数据重新更新为当时不含累积漂移的数据，同时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。所以，如果我们又充分而且正确的回环检测，则可以消除累积误差，得到全局一致的轨迹和地图。

1.3.4 建图

建图是指构件**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。大体上讲，地图可以分为**度量地图**和**拓扑地图**两种。

(1) **度量地图（Metric Map）**

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

- **稀疏地图**：稀疏地图进行了一定程度的抽象，并不需要表达所有物体。例如，我们选择一部分具有代表意义的东西，称之为**路标**（Landmark），那么一张稀疏地图就是由路标组成的地图，而不是路标的部分就可以忽略。

- **稠密地图**：稠密地图着重于建模所有看到的东西。稠密地图通常按照某种分辨率，由许多个小块组成，在二维度量地图中体现为许多个小**格子**（Grid），而在三维度量地图中则体现为许多小**方块**（Voxel）。通常一个小块含有占据、空间、未知三种状态，以表达该格内是否有物体。当查询某个空间位置时，地图能够给出该位置是否可以通过的信息。一般导航就是使用稠密地图。缺点：耗费大量储存空间且很多细节是不需要的，而且大规模度量地图可能会出现一致性问题，很小的误差就可以导致物体重叠，使地图失效。

(2) **拓扑地图（Topological Map）**

相比于度量地图的精确性，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，只考虑节点之间的连通性，而不关注节点之间是如何连通的。

优点：放松了地图对精确位置的需要，去掉了地图的细节，表达更紧凑。

缺点：不擅长表达具有复杂结构的地图，如何将实际地图转换为拓扑地图，如何使用拓扑地图进行导航与路径规划，都是比较困难的。

1.4 SLAM 问题的数学表达

假设一个机器人携带一些传感器在位置环境里面运动。由于传感器是离散采样，记采样的时刻为（连续时间离散化）$t = 1, \cdots, K$，机器人在各个时刻的位置记为 $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_K$。假设地图是由许多个**路标**组成的，传感器在各个时刻测量到一部分路标点。假设路标点一共有 N 个，记为 $\boldsymbol{y}_1, \cdots, \boldsymbol{y}_N$。

我们需要描述的两个问题：运动与观测。

1.4.1 运动的数学表达

什么是运动？即考察 $k-1 \to k$ 时刻，机器人位置 $\boldsymbol{x}$ 的变化情况。机器人的运动状况是由上一个时刻的位置 $\boldsymbol{x}_{k-1}$ 以及运动传感器的读数或输入 $\boldsymbol{u}_k$，当然只要是传感器都会有噪声，所以噪声 $\boldsymbol{\omega}_k$ 也会对当前时刻的位置造成影响。综上，我们可以用一个函数来表示通用、抽象的数学模型。

定义 1.1 运动方程

运动方程定义如下：

$$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k, \boldsymbol{\omega}_k) \tag{1.1}$$

噪声的存在使得这个模型变成了随机模型。因为每次运动过程中的噪声是随机的，如果忽略噪声的影响，只根据指令来估计位置，可能与实际位置相差很远。

1.4.2 观测的数学表达

什么是观测？即考察在 k 时刻时机器人在 $\boldsymbol{x}_k$ 处观测到了某一个路标 $\boldsymbol{y}_j$，如何描述这个事件。同样地，我们可以用一个抽象的函数来表示这个数学模型。

定义 1.2 观测方程

观测方程定义如下：

$$\boldsymbol{z}_{k,j} = h(\boldsymbol{y}_j, \boldsymbol{x}_k, \boldsymbol{v}_{k,j}) \tag{1.2}$$

其中，$\boldsymbol{v}_{k,j}$ 是观测中的噪声。

1.4.3 参数化实例

对于不同的真实运动和传感器种类，存在很多**参数化**（Parameterization）方式。例如，假设机器人在平面内运动，那么它的位姿（位置和姿态）由两个位置参数和一个转角参数来描述，即

$$\boldsymbol{x}_k = \begin{bmatrix} x_1 \\ x_2 \\ \theta \end{bmatrix}_k \tag{1.3}$$

其中，x_1, x_2 为两个轴上的位置，而 θ 为转角。同时，输入的指令是由两个时间间隔位置和转角的变化量来描述，即

$$\boldsymbol{u}_k = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta\theta \end{bmatrix}_k \tag{1.4}$$

于是运动凡尘就可以具体化为

$$\begin{bmatrix} x_1 \\ x_2 \\ \theta \end{bmatrix}_k = \begin{bmatrix} x_1 \\ x_2 \\ \theta \end{bmatrix}_{k-1} + \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta\theta \end{bmatrix}_k + \boldsymbol{\omega}_k \tag{1.5}$$

在这种情况下，运动方程就是简单的线性关系。但是，不是所有输入指令都是位移和角度的变化量。例如“油门”或“控制杆”的输入就是速度或加速度量，所以运动方程是多样化的，具体需要进行动力学分析。

关于观测方程，以二维激光传感器为例。激光传感器观测 2D 路标时能够测量两个量：路标点与机器人之间的距离 r 和夹角 ϕ。记路标点，位姿和观测数据分别为

$$\boldsymbol{y}_j = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}_j \qquad \boldsymbol{x}_k = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_k \qquad \boldsymbol{z}_{k,j} = \begin{bmatrix} r_{k,j} \\ \phi_{k,j} \end{bmatrix} \tag{1.6}$$

那么观测方程可以写为

$$\begin{bmatrix} r_{k,j} \\ \phi_{k,j} \end{bmatrix} = \begin{bmatrix} \sqrt{(y_{1,j} - x_{1,k})^2 + (y_{2,j} - x_{2,k})^2} \\ \arctan\left(\dfrac{y_{2,j} - x_{2,k}}{y_{1,j} - x_{1,k}}\right) \end{bmatrix} + \boldsymbol{v}_{k,j} \tag{1.7}$$

考虑视觉 SLAM 时，传感器是相机，那么观测方程就是“对路标点拍摄后，得到图像中的像素”的过程。

所以，针对不同的传感器，运动方程和观测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。

定义 1.3 SLAM 基本方程

SLAM 的两个基本方程为：

$$\begin{cases} \boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k, \boldsymbol{\omega}_k) & k = 1, \cdots, K \\ \boldsymbol{z}_{k,j} = h(\boldsymbol{y}_j, \boldsymbol{x}_k, \boldsymbol{v}_{k,j}) & (k,j) \in \mathcal{O} \end{cases} \tag{1.8}$$

其中，$\mathcal{O}$ 是一个集合，记录这在哪个时刻观察到了哪个路标，通常不是每个路标在每个时刻都能看到的——我们在单个时刻很可能只看到一小部分。

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

1.4.4 问题总结

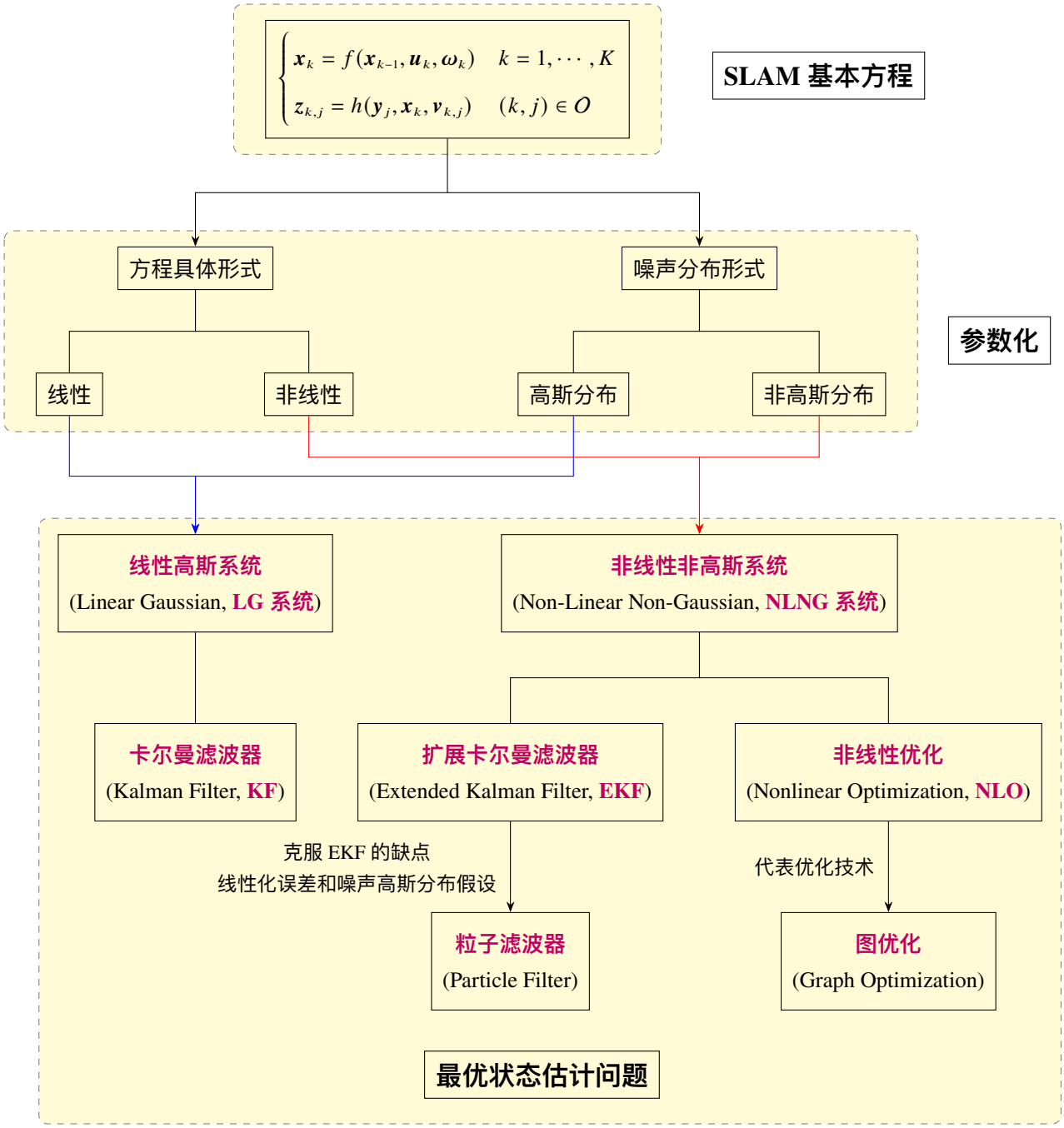

图 1.2: SLAM 问题数学表述总结图

1.5 环境配置与 CMake 基础

基本 Linux 环境：Ubuntu 20.04，安装教程略。

1.5.1 环境配置

1. git

```
sudo apt install git
```

2. vim

```
sudo apt install vim
```

3. Slambook2

```
cd ~ && git clone https://github.com/gaoxiang12/slambook2/
```

4. 编译器

```
sudo apt install gcc g++ cmake
```

5. Eigen3

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

6. Pangolin

- 系统依赖

```
sudo apt install libglew-dev libboost-dev libboost-thread-dev libboost-fileystem-dev
```

- 克隆仓库

```
cd 3rdparty # if current dir is 3rdparty, skip this command
git clone https://github.com/stevenlovegrove/Pangolin.git
```

- CMake & Install

```
cd Pangolin
mkdir build && cd build
cmake ..
make -j4
sudo make install
```

7. fmt

- 克隆仓库

```
cd 3rdparty # if current dir is 3rdparty, skip this command
git clone https://github.com/fmtlib/fmt.git
cd fmt
git checkout b6f4ceae
```

- CMake & Install

```
cd fmt # if current dir is fmt, skip this command
mkdir build && cd build
cmake ..
make -j4
sudo make install
```

8. Sophus

- 克隆仓库

```
cd 3rdparty # if current dir is 3rdparty, skip this command
git clone https://github.com/strasdat/Sophus
```

- CMake & Install

```
cd Sophus # if current dir is Sophus, skip this command
mkdir build && cd build
cmake ..
make -j4
sudo make install
```

9. g2o

- 系统依赖

```
cd 3rdparty # if current dir is 3rdparty, skip this command
sudo apt install libqt4-dev qt5-qmake libqglviewer-dev-qt5 libsuitesparse-dev libcxsparse3
    libcholmod3
```

- 克隆仓库

```
git clone https://github.com/RainerKuemmerle/g2o
cd g2o
git checkout 9b41a4ea
```

- CMake & Install

```
cd g2o # if current dir is g2o, skip this command
mkdir build && cd build
cmake ..
make -j4
sudo make install
```

10. DBow3

- 克隆仓库

```
cd 3rdparty # if current dir is 3rdparty, skip this command
git clone https://github.com/rmsalinas/DBow3
cd DBow3
git checkout c5ae539a
```

- CMake & Install

```
cd DBow3 # if current dir is DBow3, skip this command
mkdir build && cd build
cmake ..
make -j4
sudo make install
```

11. OpenCV 3

- 系统依赖

```
sudo apt install unzip build-essential libgtk2.0-dev libvtk6-dev libjpeg-dev libtiff5-dev
    libjasper-dev libopenexr-dev libtbb-dev
```

- 克隆仓库

```
cd 3rdparty # if current dir is 3rdparty, skip this command
wget https://github.com/opencv/opencv/archive/3.4.16.zip -O opencv3.zip
unzip -q opencv3.zip
```

- CMake & Install

```
cd opencv-3.4.16
mkdir build && cd build
cmake ..
make -j4
sudo make install
```

1.5.2 CMake 基本介绍

理论上，任意一个 C++ 程序都可以用 g++ 来编译。但是当程序规模越来越大时，一个工程可能有许多个文件夹和源文件，这时输入的编译命令将越来越长。通常，一个小型 C++ 项目可能还有十几个类，各类之间还存在复杂的依赖关系。其中一部分要编译成可执行文件，另一部分编译成库文件。如果仅依靠 g++ 命令，则需要输入大量的编译指令，整个编译过程会变得异常繁琐。因此，对于 C++ 项目，使用一些工程管理工具会更加高效，我们采用 CMake 管理源代码。

在一个 CMake 工程中，我们会用 cmake 命令生成一个 makefile 文件，然后，用 make 命令根据这个 makefile 文件内容编译整个工程。CMake 主要通过 CMakeLists.txt 文件来进行管理。以下用一个示例文件来说明 CMakeLists 的语法。

```
#声明要求的cmake最低版本
cmake_minimum_required(VERSION 3.2.0)

# 指定cmake编译策略
if(POLICY CMP0048)
    cmake_policy(SET CMP0048 NEW)
endif(POLICY CMP0048)

# 声明一个cmake工程，包含版本和语言类型
project(HelloSLAM VERSION 1.0.3 LANGUAGES CXX)

# gcc 编译添加选项
# add_definitions(-std=c++11) # 指定c++格式
add_definitions(-g) # 添加其他编译选项 -g 代表debug

# 头文件目录
include_directories(${PROJECT_SOURCE_DIR}/include)

# 寻找包
find_package(OPENCV 3 REQUIRED)

# 设置输出文件目录
# 静态库输出
set(CMAKE_ARCHIVE_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/Build/Lib)
# 动态库输出
set(CMAKE_LIBRARY_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/Build/Lib)
```

```
# 可执行二进制文件输出
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/Build)

# 需要额外的链接库文件
# link_directories(${PROJECT_SOURCE_DIR})

# 编译代码，SHARED代表动态库，STATIC代表静态库
# 编译动态库
add_library(hello_shared SHARED xxxxx) # xxxxx 为动态库源代码路径
# 编译静态库
# add_library(hello_static STATIC xxxxx) # xxxxx 为静态库源代码路径
# 库依赖连接
# add_library(hello_shared XXX)
# add_library(hello_static XXX)

# 生成可执行文件
add_executable(helloSLAM xxxxx) # xxxxx 为可执行文件源代码路径
target_link_libraries(helloSLAM hello_shared) # 链接动态库
#target_link_libraries(hello_static hello_shared) # 链接静态库
```

插图目录

表格目录

索引

Y

Z