

Introduction to Visual SLAM

Xiang Gao · Tao Zhang

Introduction to Visual SLAM

From Theory to Practice



中国工信出版集团



電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>



Springer

Xiang Gao
Tsinghua University
Beijing, China

Tao Zhang
Tsinghua University
Beijing, China

ISBN 978-981-16-4938-7 ISBN 978-981-16-4939-4 (eBook)
<https://doi.org/10.1007/978-981-16-4939-4>

Jointly published with Publishing House of Electronics Industry
The print edition is not for sale in China (Mainland). Customers from China (Mainland) please order the print book from: Publishing House of Electronics Industry.

Translation from the Chinese Simplified language edition: 视觉SLAM十四讲: 从理论到实践 (第2版) by Xiang Gao, and Tao Zhang. © Publishing House of Electronics Industry 2019. Published by Publishing House of Electronics Industry. All Rights Reserved.

© Publishing House of Electronics Industry 2021

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publishers, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publishers nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publishers remain neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

To my beloved Lilian and Shenghan

Preface

What is This Book Talking About?

This book introduces visual SLAM, and it is probably the first Chinese book solely focused on this specific topic. With a lot of help from the commit, it was translated into English in 2020.

So, what is SLAM?

SLAM stands for **S**imultaneous **L**ocalization and **M**apping. It usually refers to a robot or a moving rigid body, equipped with a specific **sensor**, estimates its **motion** and builds a **model** (certain kinds of description) of the surrounding environment, without a *priori* information [1]. If the sensor referred to here is mainly a camera, it is called **Visual SLAM**.

Visual SLAM is the subject of this book. We deliberately put a long definition into one single sentence so that the readers can have a clear concept. First of all, SLAM aims at solving the *localization* and *map building* issues at the same time. In other words, it is a problem of how to estimate the location of a sensor itself, while estimating the model of the environment. So how to achieve it? SLAM requires a good understanding of sensor information. A sensor can observe the external world in a particular form, but the specific approaches for utilizing such observations are usually different. And, why is this problem worth spending an entire book to discuss? Because it is difficult, especially if we want to do SLAM in **real-time** and **without any prior knowledge**. When we talk about visual SLAM, we need to estimate the trajectory and map based on a set of continuous images (which form a video sequence).

This seems to be quite intuitive. When we human beings enter an unfamiliar environment, aren't we doing exactly the same thing? So, the question is whether we can write programs and make computers do so.

At the birth of computer vision, people imagined that one-day computers could act like humans, watching and observing the world and understanding the surrounding environment. The ability to explore unknown areas is a beautiful and romantic dream, attracting numerous researchers striving on this problem day and night [2]. We thought that this would not be that difficult, but the progress turned out to be not

as smooth as expected. Flowers, trees, insects, birds, and animals are recorded so differently in computers: they are just numerical matrices consisted of numbers. To make computers understand the contents of images is as difficult as making us humans understand those blocks of numbers. We didn't even know how we understand images, nor do we know how to make computers do so. However, after decades of struggling, we finally started to see signs of success—through Artificial Intelligence (AI) and Machine Learning (ML) technologies, which gradually enable computers to recognize objects, faces, voices, texts, although in a way (probabilistic modeling) that is still so different from us.

On the other hand, after nearly three decades of development in SLAM, our cameras begin to capture their movements and know their positions. However, there is still a massive gap between the capability of computers and humans. Researchers have successfully built a variety of real-time SLAM systems. Some of them can efficiently track their locations, and others can even do the three-dimensional reconstruction in real-time.

This is really difficult, but we have made remarkable progress. What's more exciting is that, in recent years, we have seen the emergence of a large number of SLAM-related applications. The sensor location could be very useful in many areas: indoor sweeping machines and mobile robots, self-driving cars, Unmanned Aerial Vehicles (UAVs), Virtual Reality (VR), and Augmented Reality (AR). SLAM is so important. Without it, the sweeping machine cannot maneuver in a room autonomously but wandering blindly instead; domestic robots cannot follow instructions to accurately reach a specific room; Virtual reality devices will always be limited within a seat. If none of these innovations could be seen in real life, what a pity it would be. Today's researchers and developers are increasingly aware of the importance of SLAM technology. SLAM has over 30 years of research history, and it has been a hot topic in both robotics and computer vision communities. Since the twenty-first century, visual SLAM technology has undergone a significant change and breakthrough in both theory and practice and is gradually moving from laboratories into the real-world. At the same time, we regretfully find that, at least in the Chinese language, SLAM-related papers and books are still very scarce, making many beginners of this area unable to get started smoothly. Although SLAM's theoretical framework has basically become mature, implementing a complete SLAM system is still very challenging and requires a high level of technical expertise. Researchers new to the area have to spend a long time learning a significant amount of scattered knowledge and often have to go through several detours to get close to the real core.

This book systematically explains the visual SLAM technology. We hope that it will (at least partially) fill the current gap. We will detail SLAM's theoretical background, system architecture, and the various mainstream modules. At the same time, we emphasize the practice: all the essential algorithms introduced in this book will be provided with runnable code that can be tested by yourself so that readers can reach a more in-depth understanding. Visual SLAM, after all, is a technology for real applications. Although the mathematical theory can be beautiful, if you cannot convert it into code, it will be like a castle in the air, bringing little practical impact.

We believe that practice brings real knowledge (and true love). After getting your hands dirty with the algorithms, you can truly understand SLAM and claim that you have fallen in love with SLAM research.

Since its inception in 1986 [3], SLAM has been a hot research topic in robotics. It is very difficult to provide a complete introduction to all the algorithms and their variants in the SLAM history, and we consider it unnecessary as well. This book will first introduce the background knowledge, such as the 3D geometry, computer vision, state estimation theory, and Lie Group/Lie algebra. We will show the trunk of the SLAM tree and omit those complicated and oddly-shaped leaves. We think this is effective. If the reader can master the trunk's essence, they have already gained the ability to explore the frontier research details. So we aim to help SLAM beginners quickly grow into qualified researchers and developers. On the other hand, even if you are already an experienced SLAM researcher, this book may reveal areas that you are unfamiliar with and provide you with new insights.

There have already been a few SLAM-related books around, such as *Probabilistic Robotics* [4], *Multiple View Geometry in Computer Vision* [2], and *State Estimation for Robotics: A Matrix-Lie-Group Approach* [5]. They provide rich content, comprehensive discussions, and rigorous derivations, and therefore are the most popular textbooks among SLAM researchers. However, there are two critical issues: Firstly, the purpose of these books is often to introduce the fundamental mathematical theory, with SLAM being only one of its applications. Therefore, they cannot be considered as specifically visual SLAM focused. Secondly, they place great emphasis on mathematical theory but are relatively weak in programming. This makes readers still fumbling when trying to apply the knowledge they learn from the books. Our belief is one can only claim a real understanding of a problem only after coding, debugging, and tweaking algorithms and parameters with his own hands.

This book will introduce the history, theory, algorithms, and research status in SLAM and explain a complete SLAM system by decomposing it into several modules: *visual odometry*, *backend optimization*, *map building*, and *loop closure detection*. We will accompany the readers step by step to implement each core algorithm, discuss why they are effective, under what situations they are ill-conditioned, and guide them by running the code on your own machines. You will be exposed to the critical mathematical theory and programming knowledge and will use various libraries including *Eigen*, *OpenCV*, *PCL*, *g2o*, and *Ceres*, and learn their usage in Linux.

Well, enough talking, wish you a pleasant journey!

How to Use This Book?

This book is entitled as *Introduction to Visual SLAM: From Theory to Practice*. We will organize the contents into lectures like studying in a classroom. Each lecture focuses on one specific topic, organized in a logical order. Each chapter will include both a *theoretical* part and a *practical* part, with the theoretical usually coming

first. We will introduce the mathematics essential to understand the algorithms, and most of the time in a narrative way, rather than in a *definition, theorem, inference* approach adopted by most mathematical textbooks. We think this will be much easier to understand, but of course, with the price of being less rigorous sometimes. In practical parts, we will provide code, discuss the various components' meaning, and demonstrate some experimental results. So, when you see chapters with the word practice in the title, you should turn on your computer and start to program with us, joyfully.

The book can be divided into two parts: The first part will be mainly focused on fundamental math knowledge, which contains

1. Preface (the one you are reading now), introducing the book's contents and structure.
2. Lecture 1: an overview of a SLAM system. It describes each module of a typical SLAM system and explains what to do and how to do it. The practice section introduces basic C++ programming in a Linux environment and the use of an IDE.
3. Lecture 2: rigid body motion in 3D space. You will learn about rotation matrices, quaternions, Euler angles and practice them with the *Eigen* library.
4. Lecture 3: Lie group and Lie algebra. It doesn't matter if you have never heard of them. You will learn the basics of the Lie group and manipulate them with *Sophus*.
5. Lecture 4: pinhole camera model and image expression in computer. You will use *OpenCV* to retrieve the camera's intrinsic and extrinsic parameters and generate a point cloud using the depth information through Point Cloud Library (*PCL*).
6. Lecture 5: nonlinear optimization, including state estimation, least squares, and gradient descent methods, e.g., Gauss-Newton and Levenburg-Marquardt method. You will solve a curve-fitting problem using the *Ceres* and *g2o* library. From lecture 6, we will be discussing SLAM algorithms, starting with visual odometry (VO) and followed by the map building problems:
7. Lecture 6: feature-based visual odometry, which is currently the mainstream in VO. Contents include feature extraction and matching, epipolar geometry calculation, Perspective-n-Point (PnP) algorithm, Iterative Closest Point (ICP) algorithm, and Bundle Adjustment (BA), etc. You will run these algorithms either by calling *OpenCV* functions or constructing your own optimization problem in *Ceres* and *g2o*.
8. Lecture 7: direct (or intensity-based) method for VO. You will learn the optical flow principle and the direct method. The practice part is about writing single-layer and multi-layer optical flow and direct method to implement a two-view VO.
9. Lecture 8: backend optimization. We will discuss Bundle Adjustment in detail and show the relationship between its sparse structure and the corresponding graph model. You will use *Ceres* and *g2o* separately to solve the same BA problem.

10. Lecture 9: pose graph in the backend optimization. Pose graph is a more compact representation for BA, which converts all map points into constraints between keyframes. You will use *g2o* to optimize a pose graph.
11. Lecture 10: loop closure detection, mainly Bag-of-Word (BoW) based method. You will use DBoW3 to train a dictionary from images and detect loops in videos.
12. Lecture 11: map building. We will discuss how to estimate the depth of pixels in monocular SLAM (and show why they are unreliable). Compared with monocular depth estimation, building a dense map with RGB-D cameras is much easier. You will write programs for epipolar line search and patch matching to estimate depth from monocular images and then build a point cloud map and octagonal treemap from RGB-D data.
13. Lecture 12: a practice chapter for stereo VO. You will build a visual odometry framework by yourself by integrating the previously learned knowledge and solve problems such as frame and map point management, keyframe selection, and optimization control.
14. Lecture 13: current open-source SLAM projects and future development direction. We believe that after reading the previous chapters, you can understand other people's approaches easily and be capable of achieving new ideas of your own.

Finally, if you don't understand what we are talking about at all, congratulations! This book is right for you!

Source Code

All source code in this book is hosted on Github: <https://github.com/gaoxiang12/slambook2>

Note the *slambook2* refers to the second version in which we added a lot of extra experiments.

Check out the English version by: `git checkout -b en origin-en`

It is strongly recommended that readers download them for viewing at any time. The code is divided into chapters. For example, the contents of the 7th lecture will be placed in folder *ch7*. Some of the small libraries used in the book can be found in the "3rdparty" folder as compressed packages. For large and medium-sized libraries like *OpenCV*, we will introduce their installation methods when they first appear. If you have any questions regarding the code, click the *issue* button on GitHub to submit. If there is indeed a problem with the code, we will correct them in time. If you are not accustomed to using Git, you can also click the *Download* button on the right side to download a zipped file to your local drive.

Targeted Readers

This book is for students and researchers interested in SLAM. Reading this book needs specific prerequisites, and we assume that you have the following knowledge:

- **Calculus, Linear Algebra, Probability Theory.** These are the fundamental mathematical knowledge that most readers should have learned during undergraduate study. You should at least understand what a matrix and a vector are, and what it means by doing differentiation and integration. For more advanced mathematical knowledge required, we will introduce in this book as we proceed.
- **Basic C++ Programming.** As we will be using C++ as our major programming language, it is recommended that the readers are at least familiar with its basic concepts and syntax. For example, you should know what a class is, how to use the C++ standard library, how to use template classes, etc. We will try our best to avoid using tricks, but we really cannot avert them in certain situations. We will also adopt some of the C++11 standards, but don't worry. They will be explained if necessary.
- **Linux Basics.** Our development environment is Linux instead of Windows, and we will only provide source code for Linux. We believe that mastering Linux is an essential skill for SLAM researchers, and please don't ask for Windows-related issues. After going through this book's contents, we think you will agree with us.¹ In Linux, the configuration of related libraries is so convenient, and you will gradually appreciate the benefit of mastering it. If you have never used a Linux system, it will be beneficial to find some Linux learning materials and spend some time reading them (the first few chapters of an introductory book should be sufficient). We do not ask readers to have superb Linux operating skills, but we do hope readers know how to find a terminal and enter a code directory. There are some self-test questions on Linux at the end of this chapter. If you have answers to them, you should be able to quickly understand the code in this book.

Readers interested in SLAM but do not have the knowledge mentioned above may find it difficult to proceed with this book. If you do not understand the basics of C++, you can read some introductory books such as *C++ Primer Plus*. If you do not have the relevant math knowledge, we also suggest reading some relevant math textbooks first. Nevertheless, most readers who have completed undergraduate study should already have the necessary mathematical backgrounds. Regarding the code, we recommend that you spend time typing them by yourself and tweaking the parameters to see how they affect outputs. This will be very helpful.

This book can be used as a textbook for SLAM-related courses or as self-study materials.

¹ Linux is not that popular in China as our computer science education starts very lately around the 1990s.

Style

This book covers both mathematical theory and programming implementation. Therefore, for the convenience of reading, we will be using different layouts to distinguish the contents.

1. Mathematical formulas will be listed separately, and important formulas will be assigned with an equation number on the right end of the line, for example,

$$\mathbf{y} = \mathbf{A}\mathbf{x}. \quad (1)$$

Italics are used for scalars like a . Bold symbols are used for vectors and matrices like \mathbf{a} , \mathbf{A} . Hollow bold represents special sets, e.g., the real number set \mathbb{R} and the integer set \mathbb{Z} . Gothic is used for Lie Algebra, e.g., $\mathfrak{se}(3)$.

2. Source code will be framed into boxes, using a smaller font size, with line numbers on the left. If a code block is long, the box may continue to the next page:

Listing 1 Code example:

```

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

3. When the code block is too long or contains repeated parts with previously listed code, it is not appropriate to be listed entirely. We will only give the important parts and mark them with *part*. Therefore, we strongly recommend that readers download all the source code on GitHub and complete the exercises to better understand the book.
4. Due to typographical reasons, the book's code may be slightly different from the code in GitHub. In that case, please use the code on GitHub.
5. For each of the libraries we use, it will be explained in detail when first appearing but not repeated in the follow-up. Therefore, it is recommended that readers read this book in order.
6. A *goal of study* part will be presented at the beginning of each lecture. A summary and some exercises will be given at the end. The cited references are listed at the end of the book.
7. The chapters with an asterisk mark in front are optional readings, and readers can read them according to their interests. Skipping them will not hinder the understanding of subsequent chapters.
8. Important contents will be marked in **bold** or *italic*, as we are already accustomed to.

9. Most of the experiments we designed are demonstrative. Understanding them does not mean that you are already familiar with the entire library. Otherwise, this book will be an *OpenCV* or *PCL* document. So we recommend that you spend time on yourselves in further exploring the important libraries frequently used in the book.
10. The book's exercises and optional readings may require you to search for additional materials, so you need to learn to use search engines.

Exercises (Self-test Questions)

1. Suppose we have a linear equation $\mathbf{Ax} = \mathbf{b}$. If \mathbf{A} and \mathbf{b} are known, how to solve the \mathbf{x} ? What are the requirements for \mathbf{A} and \mathbf{b} if we want a unique \mathbf{x} ? (Hint: check the rank of \mathbf{A} and \mathbf{b}).
2. What is a Gaussian distribution? What does it look like in a one-dimensional case? How about in a high-dimensional case?
3. What is the **class** in C++? Do you know STL? Have you ever used them?
4. How do you write a C++ program? (It's completely fine if your answer is "using Visual C++ 6.0"²).
5. Do you know the C++11 standard? Which new features have you heard of or used? Are you familiar with any other standard?
6. Do you know Linux? Have you used at least one of the popular distributions (not including Android), such as Ubuntu?
7. What is the directory structure of Linux? What basic commands do you know? (e.g., *ls*, *cat*, etc.)
8. How to install the software in Ubuntu (without using the Software Center)? What directories are software usually installed under? If you only know the fuzzy name of a software (for example, you want to install a library with the word "eigen" in its name), how to search it?
9. *Spend an hour learning *vim*. You will be using it sooner or later. You can *vimtutor* into a terminal and read through its contents. We do not require you to operate it very skillfully, as long as you can use it to edit the code in the process of learning this book. Do not waste time on its plugins for now. Do not try to turn *vim* into an IDE. We will only use it for text editing in this book.

Beijing, China

Xiang Gao
Tao Zhang

² As I know, many of our undergraduate students are still using this version of VC++ in the university.

Acknowledgments

In the process of writing this book, a large number of documents and papers have been referenced. Most of the theoretical knowledge of mathematics is the result of previous research, not my original creation. A small part of the experimental design also comes from various open-source code demonstration programs, but most of them are written by myself. In addition, there are some pictures taken from published journals or conference papers, which have been cited in the text. Unexplained images are either original or fetched from the Internet. I don't want to infringe anyone's picture copyright. If readers find any problems, please contact me to modify it.

As I'm not a native English speaker, the translation work is based on Google translation and some afterward modifications. If you think the quality of translation can be improved and willing to do this, please contact me or send an issue on Github. Any help will be welcome!

My friends, Dr. Yi Liu and Qinrei Yan, helped me a lot in the Chinese edition of this book. And I also thank them very much about this. Thanks for the following friend's help in the translation time: Nicolas Rosa, Carrie (Yan Ran), Collen Jones, Hong Ma. And also, thanks for your attention and support!

Please contact me through GitHub or email: gao.xiang.thu@gmail.com.

Contents

Part I Fundamental Knowledge

1	Introduction to SLAM	3
1.1	Meet “Little Carrot”	3
1.1.1	Monocular Camera	6
1.1.2	Stereo Cameras and RGB-D Cameras	8
1.2	Classical Visual SLAM Framework	10
1.2.1	Visual Odometry	11
1.2.2	Backend Optimization	13
1.2.3	Loop Closing	14
1.2.4	Mapping	15
1.3	Mathematical Formulation of SLAM Problems	17
1.4	Practice: Basics	20
1.4.1	Installing Linux	20
1.4.2	Hello SLAM	22
1.4.3	Use CMake	23
1.4.4	Use Libraries	25
1.4.5	Use IDE	27
2	3D Rigid Body Motion	33
2.1	Rotation Matrix	33
2.1.1	Points, Vectors, and Coordinate Systems	33
2.1.2	Euclidean Transforms Between Coordinate Systems	35
2.1.3	Transform Matrix and Homogeneous Coordinates	38
2.2	Practice: Use <i>Eigen</i>	40
2.3	Rotation Vectors and the Euler Angles	44
2.3.1	Rotation Vectors	44
2.3.2	Euler Angles	46
2.4	Quaternions	48
2.4.1	Quaternion Operations	49
2.4.2	Use Quaternion to Represent a Rotation	51

2.4.3	Conversion of Quaternions to Other Rotation Representations	51
2.5	Affine and Projective Transformation	53
2.6	Practice: <i>Eigen</i> Geometry Module	55
2.6.1	Data Structure of the <i>Eigen</i> Geometry Module	55
2.6.2	Coordinate Transformation Example	57
2.7	Visualization Demo	58
2.7.1	Plotting Trajectory	58
2.7.2	Displaying Camera Pose	60
3	Lie Group and Lie Algebra	63
3.1	Basics of Lie Group and Lie Algebra	63
3.1.1	Group	64
3.1.2	Introduction of the Lie Algebra	65
3.1.3	The Definition of Lie Algebra	67
3.1.4	Lie Algebra $\mathfrak{so}(3)$	68
3.1.5	Lie Algebra $\mathfrak{se}(3)$	68
3.2	Exponential and Logarithmic Mapping	69
3.2.1	Exponential Map of $SO(3)$	69
3.2.2	Exponential Map of $SE(3)$	71
3.3	Lie Algebra Derivation and Perturbation Model	72
3.3.1	BCH Formula and Its Approximation	72
3.3.2	Derivative on $SO(3)$	75
3.3.3	Derivative Model	76
3.3.4	Perturbation Model	77
3.3.5	Derivative on $SE(3)$	78
3.4	Practice: Sophus	79
3.4.1	Basic Usage of Sophus	79
3.4.2	Example: Evaluating the Trajectory	81
3.5	Similar Transform Group and Its Lie Algebra	84
3.6	Summary	85
4	Cameras and Images	87
4.1	Pinhole Camera Models	87
4.1.1	Pinhole Camera Geometry	88
4.1.2	Distortion	91
4.1.3	Stereo Cameras	94
4.1.4	RGB-D Cameras	96
4.2	Images	97
4.3	Practice: Images in Computer Vision	99
4.3.1	Basic Usage of OpenCV	99
4.3.2	Basic OpenCV Images Operations	100
4.3.3	Image Undistortion	103
4.4	Practice: 3D Vision	104
4.4.1	Stereo Vision	104
4.4.2	RGB-D Vision	105

5	Nonlinear Optimization	109
5.1	State Estimation	110
5.1.1	From Batch State Estimation to Least-Square	110
5.1.2	Introduction to Least-Squares	112
5.1.3	Example: Batch State Estimation	114
5.2	Nonlinear Least-Square Problem	116
5.2.1	The First and Second-Order Method	117
5.2.2	The Gauss-Newton Method	118
5.2.3	The Levenberg-Marquardt Method	120
5.2.4	Conclusion	122
5.3	Practice: Curve Fitting	123
5.3.1	Curve Fitting with Gauss-Newton	123
5.3.2	Curve Fitting with Google Ceres	126
5.3.3	Curve Fitting with <i>g2o</i>	132
5.4	Summary	138

Part II SLAM Technologies

6	Visual Odometry: Part I	143
6.1	Feature Method	143
6.1.1	ORB Feature	146
6.1.2	Feature Matching	149
6.2	Practice: Feature Extraction and Matching	151
6.2.1	ORB Features in OpenCV	152
6.2.2	ORB Features from Scratch	154
6.2.3	Calculate the Camera Motion	157
6.3	2D–2D: Epipolar Geometry	157
6.3.1	Epipolar Constraints	157
6.3.2	Essential Matrix	160
6.3.3	Homography	162
6.4	Practice: Solving Camera Motion with Epipolar Constraints	165
6.4.1	Discussion	167
6.5	Triangulation	169
6.6	Practice: Triangulation	170
6.6.1	Triangulation with OpenCV	170
6.6.2	Discussion	171
6.7	3D–2D PnP	173
6.7.1	Direct Linear Transformation	173
6.7.2	P3P	175
6.7.3	Solve PnP by Minimizing the Reprojection Error	177
6.8	Practice: Solving PnP	181
6.8.1	Use EPnP to Solve the Pose	181
6.8.2	Pose Estimation from Scratch	182
6.8.3	Optimization by <i>g2o</i>	183
6.9	3D–3D Iterative Closest Point (ICP)	187

6.9.1	Using Linear Algebra (SVD)	188
6.9.2	Using Non-linear Optimization	190
6.10	Practice: Solving ICP	191
6.10.1	Using SVD	191
6.10.2	Using Non-linear Optimization	192
6.11	Summary	194
7	Visual Odometry: Part II	197
7.1	The Motivation of the Direct Method	197
7.2	2D Optical Flow	199
7.2.1	Lucas-Kanade Optical Flow	199
7.3	Practice: LK Optical Flow	201
7.3.1	LK Flow in OpenCV	201
7.3.2	Optical Flow with Gauss-Newton Method	202
7.3.3	Summary of the Optical Flow Practice	208
7.4	Direct Method	208
7.4.1	Derivation of the Direct Method	208
7.4.2	Discussion of Direct Method	211
7.5	Practice: Direct method	212
7.5.1	Single-Layer Direct Method	212
7.5.2	Multi-layer Direct Method	215
7.5.3	Discussion	216
7.5.4	Advantages and Disadvantages of the Direct Method	219
8	Filters and Optimization Approaches: Part I	223
8.1	Introduction	223
8.1.1	State Estimation from Probabilistic Perspective	223
8.1.2	Linear Systems and the Kalman Filter	226
8.1.3	Nonlinear Systems and the EKF	229
8.1.4	Discussion About KF and EKF	231
8.2	Bundle Adjustment and Graph Optimization	233
8.2.1	The Projection Model and Cost Function	233
8.2.2	Solving Bundle Adjustment	234
8.2.3	Sparsity	236
8.2.4	Minimal Example of BA	237
8.2.5	Schur Trick	240
8.2.6	Robust Kernels	243
8.2.7	Summary	244
8.3	Practice: BA with Ceres	245
8.3.1	BAL Dataset	245
8.3.2	Solving BA in Ceres	246
8.4	Practice: BA with <i>g2o</i>	249
8.5	Summary	252

9	Filters and Optimization Approaches: Part II	255
9.1	Sliding Window Filter and Optimization	255
9.1.1	Controlling the Structure of BA	255
9.1.2	Sliding Window	257
9.2	Pose Graph Optimization	260
9.2.1	Definition of Pose Graph	260
9.2.2	Residuals and Jacobians	261
9.3	Practice: Pose Graph	263
9.3.1	Pose Graph Using <i>g2o</i> Built-in Classes	263
9.3.2	Pose Graph Using Sophus	266
9.4	Summary	271
10	Loop Closure	273
10.1	Loop Closure and Detection	273
10.1.1	Why Loop Closure Is Needed	273
10.1.2	How to Close the Loops	275
10.1.3	Precision and Recall	276
10.2	Bag of Words	278
10.3	Train the Dictionary	280
10.3.1	The Structure of Dictionary	280
10.3.2	Practice: Creating the Dictionary	281
10.4	Calculate the Similarity	284
10.4.1	Theoretical Part	284
10.4.2	Practice Part	285
10.5	Discussion About the Experiment	288
10.5.1	Increasing the Dictionary Scale	288
10.5.2	Similarity Score Processing	290
10.5.3	Processing the Keyframes	290
10.5.4	Validation of the Detected Loops	291
10.5.5	Relationship with Machine Learning	291
11	Dense Reconstruction	293
11.1	Brief Introduction	293
11.2	Monocular Dense Reconstruction	296
11.2.1	Stereo Vision	296
11.2.2	Epipolar Line Search and Block Matching	297
11.2.3	Gaussian Depth Filters	299
11.3	Practice: Monocular Dense Reconstruction	302
11.3.1	Discussion	310
11.3.2	Pixel Gradients	310
11.3.3	Inverse Depth Filter	311
11.3.4	Pre-Transform the Image	313
11.3.5	Parallel Computing	314
11.3.6	Other Improvements	314
11.4	Dense RGB-D Mapping	315
11.4.1	Practice: RGB-D Point Cloud Mapping	316

11.4.2	Building Meshes from Point Cloud	320
11.4.3	Octo-Mapping	322
11.4.4	Practice: Octo-mapping	325
11.5	*TSDF and RGB-D Fusion Series	327
11.6	Summary	330
12	Practice: Stereo Visual Odometry	331
12.1	Why Do We Have a Separate Engineering Chapter?	331
12.2	Framework	333
12.2.1	Data Structure	333
12.2.2	Pipeline	334
12.3	Implementation	335
12.3.1	Implement the Basic Data Structure	335
12.3.2	Implement the Frontend	339
12.3.3	Implement the Backend	341
12.4	Experiment Results	344
13	Discussions and Outlook	347
13.1	Open-Source Implementations	347
13.1.1	MonoSLAM	348
13.1.2	PTAM	349
13.1.3	ORB-SLAM Series	351
13.1.4	LSD-SLAM	353
13.1.5	SVO	354
13.1.6	RTAB-MAP	356
13.1.7	Others	356
13.2	SLAM in Future	357
13.2.1	IMU Integrated VSLAM	358
13.2.2	Semantic SLAM	359
	Appendix A: Gaussian Distribution	363
	Appendix B: Matrix Derivatives	367
	References	369