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.

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 **a**nd **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,

$$\mathbf{y} = \mathbf{A}\mathbf{x}. \tag{1}$$

   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