14 Lectures on Visual SLAM: From Theory to Practice

Xiang Gao, Tao Zhang, Qinrui Yan and Yi Liu

June 10, 2019

Contents

Preface for English Version

A lot of friends at github asked me about this English version. I'm really sorry it takes so long to do the translation, and I'm glad to make it public available to help the readers. I encountered some issues on math equation in the web pages. Since the book is originally written in LaTeX, I'm going to release the LaTeX source along with the compiled pdf. You can directly access the pdf version for English book, and probably the publishing house is going to help me do the paper version.

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

Xiang

Chapter 1

Preface

1.1 What is this book about?

This is a book introducing visual SLAM, and it is probably the first Chinese book solely focused on this specific topic.

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 own **motion** and builds a **model** (certain kinds of description) of the surrounding environment, without a *priori* information[1]. If the sensor referred 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 "positioning" 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? This requires a good understanding of sensor information. A sensor can observe the external world in a certain form, but the specific approaches for utilizing such observations are usually different. And, why is this problem worth spending an entire book to discuss? Simply because it is difficult, especially if we want to do SLAM in **real time** and **without any a priory 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).

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 human, watching and observing the world, and understanding the surrounding environment. The ability of exploring unknown areas is a wonderful and romantic dream, attracting numerous researchers striving on this problem day and night [**?**]. 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 simply matrices consisted of numbers. To make computers understand the contents of images, is as difficult as making us human 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 identify 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, although there is still a huge gap between the capability of computers and human. Researchers have successfully built a variety of real-time SLAM systems. Some of them can efficiently track their own locations, and others can even do 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 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, automatic driving cars, Unmanned Aerial Vehicles (UAVs) in the air, 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 can not follow instructions to reach a certain room accurately; Virtual Reality will always be limited within a prepared space. 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 the 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 21st century, visual SLAM technology has undergone a significant change and breakthrough in both theory and practice, and is gradually moving from laboratories into 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 the theoretical framework of SLAM has basically become mature, to implement a complete SLAM system is still very challenging and requires 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 a number of detours to get close to the real core.

This book systematically explains the visual SLAM technology. We hope that it will (at least in part) fill the current gap. We will detail SLAM's theoretical background, system architecture, and the various mainstream modules. At the same time, we place great emphasis on practice: all the important algorithms introduced in this book will be provided with runnable code that can be tested by yourself, so that readers can reach a deeper understanding. Visual SLAM, after all, is a technology for application. Although the mathematical theory can be beautiful, if you are not able to convert it into lines of code, it will be like a castle in the air, which brings little practical impact. We believe that practice verifies true knowledge, and practice tests true passion. Only 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 [2], SLAM has been a hot research topic in robotics. It is very difficult to give a complete introduction to all the algorithms and their variants in the SLAM history, and we consider it as unnecessary as well. This book will be firstly introducing the background knowledge, such as projective geometry, computer vision, state estimation theory, Lie Group and Lie algebra, etc. On top of that, we will be showing the trunk of the SLAM tree, and omitting those complicated and oddly-shaped leaves. We think this is effective. If the reader can master the

essence of the trunk, they have already gained the ability to explore the details of the research frontier. So our aim is 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 still reveal areas that you are unfamiliar with, and may provide you with new insights.

There have already been a few SLAM-related books around, such as "Probabilistic Robotics" [3], "Multiple View Geometry in Computer Vision" [4], "State Estimation for Robotics: A Matrix-Lie-Group Approach"[**?**], etc. They provide rich contents, comprehensive discussions and rigorous derivations, and therefore are the most popular textbooks among SLAM researchers. However, there are two important 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: only after coding, debugging and tweaking algorithms and parameters with his own hands, one can claim real understanding of a problem.

In this book, we will be introducing the history, theory, algorithms and research status in SLAM, and explaining a complete SLAM system by decomposing it into several modules: visual odometry, back-end optimization, map building, and loop closure detection. We will be accompanying the readers step by step to implement the core algorithms of each module, explore why they are effective, under what situations they are ill-conditioned, and guide them through running the code on their 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 master their use in the Linux operating system.

Well, enough talking, wish you a pleasant journey!

1.2 How to use this book?

This book is entitled "14 Lectures on Visual SLAM". As the name suggests, we will organize the contents into "lectures" like we are learning 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 a price of being less rigorous sometimes. In practical parts, we will provide code and discuss the meaning of the various parts, 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 the fundamental math knowledge, which contains:

1. Lecture 1: preface (the one you are reading now), introducing the contents and structure of the book.

2. Lecture 2: an overview of a SLAM system. It describes each module of a SLAM system and explains what they do and how they do it. The practice

section introduces basic C++ programming in Linux environment and the use of an IDE.

3. Lecture 3: rigid body motion in 3D space. You will learn knowledge about rotation matrices, quaternions, Euler angles, and practice them with the Eigen library.

4. Lecture 4: Lie group and Lie algebra. It doesn't matter if you have never heard of them. You will learn the basics of Lie group, and manipulate them with Sophus.

5. Lecture 5: pinhole camera model and image expression in computer. You will use OpenCV to retrieve camera's intrinsic and extrinsic parameters, and then generate a point cloud using the depth information through PCL (Point Cloud Library).

6. Lecture 6: nonlinear optimization, including state estimation, least squares and gradient descent methods, e.g. Gauss-Newton and Levenburg-Marquardt. You will solve a curve fitting problem using the Ceres and g2o library.

   From lecture 7, we will be discussing SLAM algorithms, starting with Visual Odometry (VO) and followed by the map building problems:

7. Lecture 7: 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 by constructing you own optimization problem in Ceres and g2o.

8. Lecture 8: direct (or intensity-based) method for VO. You will learn the principle of optical flow and direct method, and then use g2o to achieve a simple RGB-D direct method based VO (the optimization in most direct VO algorithms will be more complicated).

9. Lecture 9: back-end 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 a same BA problem.

10. Lecture 10: pose graph in the back-end optimization. Pose graph is a more compact representation for BA which marginalizes all map points into constraints between keyframes. You will use g2o and gtsam to optimize a pose graph.

11. Lecture 11: 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 12: map building. We will discuss how to estimate the depth of feature points 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 tree map from RGB-D data.

13. Lecture 13: a practice chapter for VO. You will build a visual odometer framework by yourself by integrating the previously learned knowledge, and solve problems such as frame and map point management, key frame selection and optimization control.

14. Lecture 14: current open source SLAM projects and future development direction. We believe that after reading the previous chapters, you will be able to understand other people's approaches easily, and be capable to achieve 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!

1.3 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 which I added a lot of extra experiments.

It is strongly recommended that readers download them for viewing at any time. The code is divided by chapters, for example, the contents of the 7th lecture will be placed in folder "ch7". In addition, some of the small libraries used in the book can be found in the "3rd party" 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 "Issues" button on GitHub to submit. If there is indeed a problem with the code, we will make changes in a timely manner. Even if your understanding is biased, we will still reply as much as possible. If you are not accustomed to using Git, you can also click the button on the right which contains the word "download" to download a zipped file to your local drive.

1.4 Oriented readers

This book is for students and researchers interested in SLAM. Reading this book needs certain prerequisites, 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 in certain situations we really can not avert. In addition, we will adopt some of C++ 11 standard, but don't worry, they will be explained as they appear.

- 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 take it to begin. After going through the contents of this book, we believe 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 if you can find some Linux learning materials and spend some time reading them (to master Linux basics, 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 at least know how to fire an 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 shouldn't have much problem in understanding the code in this book.

Readers interested in SLAM but do not have the above mentioned knowledge 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 that you read some relevant math textbooks first. Nevertheless, we think that most readers who have completed undergraduate study should already have the necessary mathematical arsenal. 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, but also suitable as extra-curricular self-study materials.

1.5 Style

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

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}. \tag{1.1}$$

   Italics are used for scalars, e.g., a. Bold italics are used for vectors and matrices, e.g.$\boldsymbol{a}, \boldsymbol{A}$. Hollow bold represents special sets, e.g. real number $\mathbb{R}$ and 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:

```
1  #include <iostream>
2  using namespace std;
3
```

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

```
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 **important snippets** and mark it with "Snippet". 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 code shown in the book may be slightly different from the code hosted on GitHub. In that case please use the code on GitHub.

5. For each of the libraries we use, it will be explained in details when first appearing, but not repeated in the follow-up. Therefore, it is recommended that readers read this book in order.

6. An abstract 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 interest. 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. 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.

1.6 Acknowledgments

The online English version of this book is currently public available and open source. The Chinese version

1.7 Exercises (self-test questions)

1. There is a linear equation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, if $\boldsymbol{A}$ and $\boldsymbol{b}$ are known, how to solve for $\boldsymbol{x}$? What are the requirements for $\boldsymbol{A}$ and $\boldsymbol{b}$ if we want an unique $\boldsymbol{x}$? (Hint: check the rank of $\boldsymbol{A}$ and $\boldsymbol{b}$).

2. What is a Gaussian distribution? What does it look like in one-dimensional case? How about in high-dimensional case?

3. Do you know what a **class** is 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" [①]. As long as you have C++ or C programming experience, you are in good hand).

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 flavor (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 a 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 a word "eigen" in its name), how would you do it?

9. *Spend an hour learning Vim, you will be using it sooner or later. You can type "vimtutor" into an 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, do not try to turn Vim into an IDE for now, we will only use it for text editing in this book.

[①] As I know many of our undergraduate students are still using this VC++ 6.0 in the university.

Chapter 2

First Glance of Visual SLAM

Goal of Study

1. Understand which modules a visual SLAM framework consists of, and what task each module carries out.
2. Set up the programming environment, and prepare for experiments.
3. Understand how to compile and run a program under Linux. If there is a problem, how to debug it.
4. Learn the basic usage of cmake.

2.1 Introduction

This lecture summarizes the structure of a visual SLAM system as an outline of subsequent chapters. Practice part introduces the fundamentals of environment setup and program development. We will make a small "Hello SLAM" program at the end.

2.2 Meet "Little Carrot"

Suppose we assembled a robot called *Little Carrot*, as shown in the following figure:

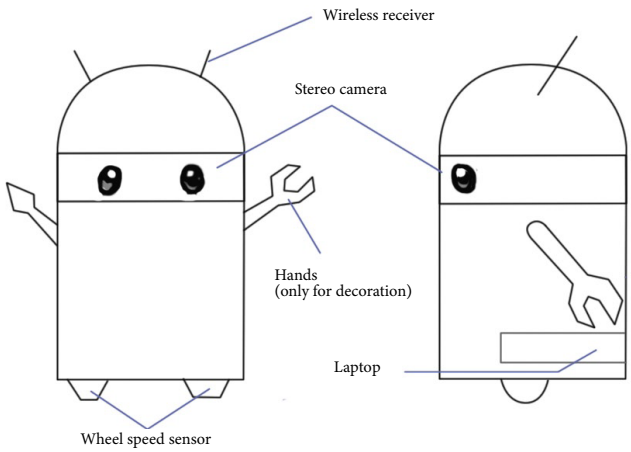

Figure 2-1: The sketch of robot *Little Carrot*

Although it looks a bit like the Android robot, it has nothing to do with the Android system. We put a laptop into its trunk (so that we can debug programs at any time). So, what is our robot capable to do?

We hope Little Carrot has the ability of *autonomous moving*. Although there are many *robots* placed statically on desktops, capable of chatting with people or playing music, but a tablet computer nowadays can also deliver the same tasks. As a robot, we hope Little Carrot can move freely in a room. Wherever we say hello to it, it can come to us right away.

First of all, such a robot needs wheels and motors to move, so we installed wheels under Little Carrot (gait control for humanoid robots is very complicated, which we will not be considering here). Now with the wheels, the robot is able to move, but without an effective navigation system, Little Carrot does not know where a target of action is, and it can do nothing but wander around blindly. Even worse, it may hit a wall and cause damage. In order to avoid this, we installed cameras on its head, with the intuition that such a robot *should look similar to human*. Certainly, with eyes, brains and limbs, human can walk freely and explore any environment, so we (somehow naively) think that our robot should be able to achieve it too. Well, in order to make Little Carrot able to explore a room, we find it at least needs to know two things:

1. Where am I? - It's about *localization*.

2. What is the surrounding environment like? -It's about *map building*.

Localization and *map building*, can be seen as the perception in both inward and outward directions. As a completely autonomous robot, Little Carrot need not only to understand its own *state* (i.e. the location), but also the external *environment* (i.e. the map). Of course, there are many different approaches to solve these two problems. For example, we can lay guiding rails on the floor of the room, or paster a lot of artificial markers such as QR code pictures on the wall, or mount radio positioning devices on the table. If you are outdoor, you can also install a GNSS receiver (like the one in a cell phone or a car) on the head of Little Carrot. With these devices, can we claim that the positioning problem has been resolved? Let's categorize these sensors (see Fig. 2-2) into two classes.

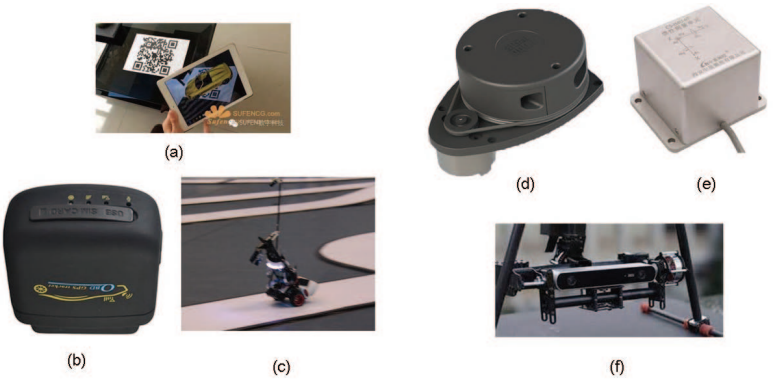

Figure 2-2: Different kinds of sensors: (a) QR code (b) GNSS receiver (c) guiding rails (d) Laser range finder (e) Inertial measurement unit (f) stereo camera

The first class are *non-intrusive* sensors which are completely self-contained inside a robot, such as wheel encoders, cameras, laser scanners, etc. They do not assume an cooperative environment around the robot. The other class are *intrusive* sensors depending on a prepared environment, such as the above mentioned guiding rails, QR codes, etc. Intrusive sensors can usually locate a robot directly, solving the positioning problem in a simple and effective manner. However, since they require changes on the environment, the scope of usage is often limited within a certain degree. For example, if there is no GPS signal, or guiding rails cannot be laid, what should we do in those cases?

We can see that the intrusive sensors place certain *constraints* to the external environment. A localization system based on them can only function properly when those constraints are met in the real world. Otherwise, the localization approach cannot be carried out anymore, like GPS positioning system normally doesn't work well in indoor environments. Therefore, although this type of sensor is simple and reliable, they do not work as a general solution. In contrast, non-intrusive sensors, such as laser scanners, cameras, wheel encoders, Inertial Measurement Units (IMUs), etc., can only observe indirect physical quantities rather than the direct locations. For example, a wheel encoder measures the wheel rotation angle, an IMU measures the angular velocity and the acceleration, a camera or a laser scanner observe the external environment in a certain form like point-clouds and images. We have to apply algorithms to infer positions from these indirect observations. While this sounds like a roundabout tactic, the more obvious benefit is that it does not make any

demands on the environment, making it possible for this localization framework to be applied to an unknown environment. Therefore, they are called as self-localization in many research area.

Looking back at the SLAM definitions discussed earlier, we emphasized an *unknown environment* in SLAM problems. In theory, we should not presume which environment the Little Carrot will be used (but in reality we will have a rough range, such as indoor or outdoor), which means that we can not assume that the external sensors like GPS can work smoothly. Therefore, the use of portable non-intrusive sensors to achieve SLAM is our main focus. In particular, when talking about visual SLAM, we generally refer to the using of *cameras* to solve the localization and map building problems.

Visual SLAM is the main subject of this book, so we are particularly interested in what the Little Carrot's eyes can do. The cameras used in SLAM are different from the commonly seen SLR cameras. It is often much simpler and does not carry expensive lens. It shoots at the surrounding environment at a certain rate, forming a continuous video stream. An ordinary camera can capture images at 30 frames per second, while high-speed cameras can do faster. The camera can be roughly divided into three categories: Monocular, Stereo and RGB-D, as shown by the following figure 2-3. Intuitively, a monocular camera has only one camera, a stereo camera has two. The principle of a RGB-D camera is more complex, in addition to being able to collect color images, it can also measure the distance of the scene from the camera for each pixel. RGB-D cameras usually carry multiple cameras, and may adopt a variety of different working principles. In the fifth lecture, we will detail their working principles, and readers just need an intuitive impression for now. In addition, there are also specialty and emerging camera types can be applied to SLAM, such as panorama camera [5], event camera [6]. Although they are occasionally seen in SLAM applications, so far they have not become the mainstream. From the appearance we can infer that Little Carrot seems to carry a stereo camera.

Figure 2-3: Different kinds of cameras: monocular, RGB-D and stereo.

Now, let's take a look at the pros and cons of using different type of camera for SLAM.

Monocular Camera

The SLAM system that uses only one camera is called Monocular SLAM. This sensor structure is particularly simple, and the cost is particularly low, therefore the monocular SLAM has been very attractive to researchers. You must have seen the output data of a monocular camera: photo. Yes, as a photo, what are its characteristics?

A photo is essentially a *projection* of a scene onto a camera's imaging plane. It reflects a three-dimensional world in a two-dimensional form. Obviously, there is one dimension lost during this projection process, which is the so-called depth (or distance). In a monocular case, we can not obtain the *distance* between objects in the scene and the camera by using a single image. Later we will see that this distance is actually critical for SLAM. Because we human have seen a large number of images, we formed a natural sense of distances for most scenes, and this can help us determine the distance relationship among the objects in the image. For example, we can recognize objects in the image and correlate them with their approximate size obtained from daily experience. The close objects will occlude the distant objects; the sun, the moon and other celestial objects are infinitely far away; an object will have shadow if it is under sunlight. This common sense can help us determine the distance of objects, but there are also certain cases that confuse us, and we can no longer determine the distance and true size of an object. The following figure 2-4 is shown as an example. In this image, we can not determine whether the figures are real person or small toys purely based on the image itself. Unless we change our view angle, explore the three-dimensional structure of the scene. In other words, from a single image, we can not determine the true size of an object. It may be a big but far away object, but it may also be a close but small object. They may appear to be the same size in an image due to the perspective projection effect.

Figure 2-4: We cannot tell if the people are real humans or just small toys from a single image

Bibliography

[1] A. Davison, I. Reid, N. Molton, and O. Stasse, "Monoslam: Real-time single camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.

[2] R. C. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *International Journal of Robotics Research*, vol. 5, no. 4, pp. 56–68, 1986.

[3] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT Press, 2005.

[4] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge university press, 2003.

[5] A. Pretto, E. Menegatti, and E. Pagello, "Omnidirectional dense large-scale mapping and navigation based on meaningful triangulation," *2011 IEEE International Conference on Robotics and Automation (ICRA 2011)*, pp. 3289–96, 2011.

[6] B. Rueckauer and T. Delbruck, "Evaluation of event-based algorithms for optical flow with ground-truth from inertial measurement sensor," *Frontiers in neuroscience*, vol. 10, 2016.