SLAM in Robotics and Autonumous Driving

Xiang Gao

Last update: March 21, 2024

To my beloved Lilian and Shenghan

Contents

Preface

About this book

Well, autonomous driving is a such cool stuff, isn't it?

You've probably seen scenes of self-driving cars in science fiction movies. In these vehicles, the steering wheel turns by itself, the throttle and brakes are controlled automatically, freeing people from the monotony of driving so they can enjoy their time more freely. In fact, some Level 2 vehicles have already achieved partial self-driving capabilities in simple road conditions. They can help drivers keep the vehicle centered in the lane or maintain a certain distance from the preceding vehicle. These systems are called **Advanced Driver Assistance Systems (ADAS)**. And for more advanced self-driving systems (Level 4), computers can fully take over, not only assisting drivers but also controlling buses, delivery vehicles, robots, robotic dogs, and even bicycles, enabling many functionalities we never imagined. Over time, these sci-fi-like scenes have gradually become reality. Self-driving jobs have also emerged as a new industry, attracting young talents from all sectors of society. It's truly an exciting development!

The field of autonomous driving encompasses many emerging technologies, with Simultaneous Localization and Mapping (SLAM) being a key focus. Since completing my PhD, I have been involved in the research and development of SLAM in the autonomous driving industry. It's an intriguing area because whether it's large passenger vehicles, small low-speed vehicles, or even sweepers, SLAM is certainly a fundamental technology. Most of the automation features we see are actually embedded within the vehicle's map data. For instance, the map might instruct the vehicle to turn left at the upcoming intersection and merge into the right lane of the opposite road; or, for cleaning a plaza ahead, the vehicle should drive along the right boundary in circles while avoiding the flower bed area in the middle. To achieve these functionalities, we need to integrate various types of data such as GPS, inertial navigation, laser point clouds, visual images, etc., to construct maps and then perform localization on these maps.

If you work in this area, you'll find that it brings together people with diverse backgrounds. People working on inertial navigation are familiar with strapdown inertial navigation systems. They enjoy writing matrix and vector computation programs on a small embedded CPU, often scratching their heads over errors of one or two points aboue accuracy. Those involved in processing laser point clouds engage in meticulous map reconstruction, displaying beautiful three-dimensional point clouds on the screen. Meanwhile, those working on vision spend their days manipulating images on the imaging plane, producing impressive results but not focusing much on accuracy issues. Well, I mean, accuracy is of course an important issue, but the accuracy on a two-dimensional imaging plane is not the same as the accuracy of three-dimensional world points or localization accuracy. Cameras usually don't have fixed accuracy metrics like other sen-

sors. They can either focus on nearby objects for local high precision or distant objects for a broad field of view, just like the difference between a microscope and a telescope. Thanks to the collaboration among these individuals and effective communication from management, vehicles are able to navigate the roads stably. However, most of the time, we aren't entirely clear about what other people are doing or how they're doing it. This is one of the motivations for me to write this book.

I hope to introduce readers to the localization and mapping technologies related to autonomous driving and robotics in this book, which include the sensors we use in our daily lives. Although there is currently no unified opinion on what constitutes a vehicle or a robot, they can even be seen as wheeled smartphones. However, these intelligent machines all use similar sensors, and the underlying theories are basically the same. I hope that through this book, researchers in this field can enhance mutual understanding, or colleagues and students outside the field can understand what work we are doing. I believe many people will be interested in these technologies.

Content of This Book

This book introduces SLAM-related technologies used in autonomous driving and robotics. The SLAM discussed here is quite general. We will cover sensors and processing methods related to localization and mapping. A typical autonomous vehicle will include various sensors such as IMU, wheel encoders, vehicle speed sensors, and multi-line laser sensors, so our localization and mapping will also involve methods for these sensors. Therefore, readers will encounter topics like basic algorithms for inertial navigation, filters represented by Kalman filtering, laser point cloud matching methods, trajectory fusion algorithms, and more in this book. Overall, we will introduce these contents in the following order:

1. The first part covers **Basic Mathematical Knowledge**. We will start with basic coordinate system definitions, rotational geometry, and quickly introduce some mathematical background knowledge used in this book. Since most of this background knowledge can be found in other books and materials, we will only provide a brief introduction. Chapter 1 of Part 1 provides an overview of autonomous driving, Chapter 2 introduces basic geometry and kinematics, Chapter 3 covers the error Kalman filter used for integrated navigation, and Chapter 4 introduces pre-integration systems and optimization methods. Readers don't need to worry about the specialized terminology mentioned here; we will delve into them in detail in specific chapters.

2. The second part is about **Laser Localization and Mapping**. This section introduces 2D and 3D laser localization technologies, with the former mainly used in robots represented by sweepers, and the latter being one of the foundational technologies for autonomous driving vehicles. We will detail representative techniques for processing laser point clouds and demonstrate their applications through code implementation. Chapter 5 of Part 2 introduces basic point cloud processing algorithms (nearest neighbor structures, KD trees, etc.), Chapter 6 discusses 2D laser localization and mapping, and Chapter 7 covers 3D laser localization and mapping.

3. The third part focuses on **Application**. We will discuss the process of building high-precision point cloud maps for autonomous driving and how to use point

cloud maps for real-time localization. Chapter 8 introduces tightly coupled laser-inertial odometry methods, Chapter 9 presents offline point cloud mapping systems, and Chapter 10 introduces online fusion localization systems.

Similar to my previous book[1], this book emphasizes the unity of theory and practice and pays close attention to the implementation of principles in code. All algorithms mentioned in this book will have code implementations provided in the corresponding chapters. Readers will work with us to implement those important and foundational algorithmic structures in this field from scratch, using modern programming techniques and fully exploiting parallelization principles to make our algorithms run smoother than classical implementations. Consequently, we will not limit ourselves to specific implementations of open-source code. For example, we will avoid discussing what LOAM does from line X to line Y or which library Cartographer references in a particular cpp file. We will refrain from discussing engineering details like thread pools or parameter file formats. Yes, such discussions can be too detailed, and everyone's implementation may differ. We will strive to retain only the core algorithmic code, allowing readers to debug and understand the entire process themselves.

In terms of style, I will continue to use my familiar writing style. Readers who are familiar with me should quickly adapt, while those who are not should not find it overly challenging. I hope my writing is as clear and straightforward as a conversation. Throughout the introduction of content, I hope the reading process reflects a complete train of thought, rather than simply compiling information together. Although this writing style may result in some verbosity, I believe it is beneficial.

The majority of the key contents in this book will be accompanied by corresponding implementation code. This is one of the major features of this book. I believe that for a comprehensive book, providing code that demonstrates the concepts is always a wise choice. However, despite our efforts to streamline the code, the code section of this book is still much larger than my previous book.

Here is our code repository:

```
https://github.com/gaoxiang12/slam_in_autonomous_driving
```

Please checkout the "en" branch if you only want to read English comments.

All code and data for this book are open-source and freely accessible to readers. The PDF file of this book will be continuously updated within the code repository. We use C++ as the primary programming language. Please don't ask me why I didn't use more concise languages like Python or Matlab because the programs running in actual vehicles or robots are still primarily C++ programs, and I don't want our experiments to deviate too far from industrial applications. Please note that the code, errata, and other files for this book will be updated on GitHub first, while the published version of the book may have a certain lag time due to the printing schedule of the publishing house. If readers find any discrepancies between the content in the book and the code repository, please consider the implementation in the code repository as authoritative.

We welcome readers to ask questions or answer questions from other readers in the code repository of this book. We encourage readers to communicate in English to facilitate sharing your experiences with international friends.

[1] `https://github.com/gaoxiang12/slambook-en`

How to Use This Book

The content of this book follows a process of gradually deepening complexity, but even basic topics like those in Chapter 2 require some groundwork. Personally, I hope this book serves as a sequel to my previous book [1] (https://github.com/gaoxiang12/slambook-en). You should at least read the first 6 chapters of that book to familiarize yourself with some basic mathematical principles and the basic usage of optimization libraries. However, if readers have not read that book, you should at least have knowledge in the following areas:

- Basic undergraduate-level mathematics such as calculus, linear algebra, and probability theory.
- Mathematics at the graduate level: optimization, matrix theory, a small amount of knowledge about Lie groups and Lie algebras.
- Computer science: Linux system operations, C++ programming language.

If readers find certain parts of this book difficult to understand, they can refer to corresponding reference books for supplementary learning. Overall, this book will be slightly more challenging and the pace of introduction will be somewhat faster.

The code for this book is organized by chapter. For example, the code for Chapter 3 will be located in src/ch3. The code for each chapter will be compiled into separate library files and executable files. Additionally, shared code will be placed in src/common (such as some common structures, message definitions, UI, etc.). There is a certain degree of dependency between the code of different chapters, with later chapters reusing the results of earlier chapters. The code for this book needs to be compiled using ROS, but the actual running and testing processes do not require the use of ROS mechanisms; only ROS data packages are used for storage. Readers only need to understand the installation process of ROS and do not need to familiarize themselves with the details of ROS in advance.

Notations

The mathematical symbols in this book follow the international standard. In general, scalars are expressed in slanted font, such as a; matrices and vectors are represented in bold font, such as $\mathbf{A}$; special sets are denoted in hollow sans-serif font, such as $\mathbb{R}$; and Lie algebra-related sets are expressed in Gothic font, such as $\mathfrak{so}(3)$. We aim to maintain consistency throughout the book in terms of symbols, with additional explanations provided where ambiguity may arise.

Relationship with Other Books and Papers

Autonomous driving localization technology involves many active research fields. For example, Professor Barfoot's "State Estimation for Robotics" [2] focuses on introducing state estimation theory. Its Chinese translation was also translated by our team. On the one hand, it provides a comparative introduction to the differences and similarities between traditional filtering theory and modern optimization theory, and on the other hand, it provides an excellent introduction to Lie groups and Lie algebra for engineering readers. This book will partially use some conclusions from the book on state estimation, mainly the part related to Lie groups and Lie algebras, to support some of our formula derivations.

Professor Ma's "An invitation to 3-d vision: from images to geometric models" [3] is also an excellent book that introduces knowledge of 3D vision, with many similarities in the basic knowledge of 3D geometry.

Joan Solà's "Quaternion Kinematics for the Error-State Kalman Filter" [4] provides a very concise and precise theory of quaternion-based error Kalman filters. Although it is not lengthy, it discusses quaternions and Kalman filters very thoroughly, and most derivations in this field are based on this material. This book will also use some of its results, but we will mainly derive various filter formulas based on Lie groups rather than quaternion forms.

Professor Thrun's "Probabilistic Robotics" [5] is also a well-known classic book in the field of robotics. It introduces some results related to SLAM in the field of robotics, and provides a very detailed introduction to traditional filters, 2D grid maps, and other content. This book will also introduce 2D grid localization and mapping methods, with the theoretical part also referencing this book's content.

Professor Qin Yongyuan and Professor Yan Gongmin's works in the field of inertial navigation, including "Inertial Navigation" [6], "Kalman Filter Algorithm and Combination Navigation Principle for Strapdown Inertial Navigation" [7], and "Inertial Instrument Testing and Data Analysis" [8], are classic textbooks in this field, and many teachers and students studying inertial navigation will refer to their derivation process. This book also refers to these books in the field of inertial navigation, but compared to specialized textbooks on inertial navigation, the content introduced in this book will be relatively basic. We mainly introduce the basic principles of inertial navigation, without involving complex parameter compensation or discussions on various subdivided motion states. However, in contrast, the preintegration principle and nonlinear optimization part introduced in this book are not fully introduced in these traditional textbooks.

Finally, compared to the books and materials mentioned above, the biggest feature of this book is still the unity of code and theory. It can be said that most books are for reading, while this book can be **executed**. I believe that understanding many algorithmic aspects requires readers to participate in the debugging and running process.

Environment

This book uses Ubuntu 20.04 as the experimental environment. Readers can use their personal computers as development environments. If familiar with Docker, they can also use Docker environments. The book primarily utilizes **C++17** as the C++ standard, which may be relatively new to some readers. Older machines or environments may not necessarily support it well. We recommend readers to use software environments above Ubuntu 20.04 to run the code in this book; otherwise, you may need to address some minor issues regarding C++ standard support.

This book comes with a considerable amount of test data, which is quite large (approximately 270GB). We suggest that readers allocate at least 100GB of space to run the code in this book. Readers can download the test data through the links provided in the book's repository.

Acknowledgement

1. Considering the confidentiality of geographical information, this book avoids using domestic data and tends to use open-source datasets worldwide. Readers can consider the trajectories or point clouds provided in the book as data in a general

spatial coordinate system, without concerning themselves with the actual geographical locations of this data.

2. Similarly, unless necessary, the data provided in this book will not specify geographical information such as place names or ranges. Readers can regard them as general road, plaza, or building scenes.

3. Some of the images used in this book are sourced from internet search engines and are used solely for educational purposes, with no intention of infringing on the original authors' copyrights. Some of the images used in this book may contain logos of commercial companies or may be images used in promotional materials for some companies. These images are sourced from public search engines and do not imply any cooperation or competition relationship between the authors and the companies. The author will strive to obtain authorization for images that may have commercial copyrights. If there is any dispute, please inform us.

4. Each chapter of this book uses datasets from different sources, mainly including the NCLT dataset from the University of Michigan [9], the UTBM dataset from Montbéliard, France [10], and the UrbanLoco dataset mainly from Hong Kong, China [11] (ULHK), among others. This book has designed a unified interface for them programmatically, making it convenient for readers to test the performance of algorithms on different datasets.

5. The English version of this book is translated with the help of ChatGPT and I would like to thank their great work here.

Part I

Basic Knowledge


笔者也刚买了新能源车，看着车自己开，真有趣！
京A8CDEFG

Chapter 1

Autonomous Driving

1.1 Autonomous Driving Technologies

1.1.1 Autonomous Driving Capabilities and Grading

Autonomous driving, as the name suggests, is the study of enabling vehicles to drive by itself. If you were to design an autonomous driving system, where would you start?

Although the internal structure of a car is highly complex, what humans actually need to operate is simply looking ahead, manipulating the steering wheel, accelerator, and brake pedals. If a computer program also learns to send signals to the steering wheel and pedals based on information from camera images, does it qualify as learning autonomous driving? If so, how should this program be designed?

In a naive conception, to enable a car to autonomously drive, one should first observe how humans perform driving behaviors. Humans primarily use vision to judge the relationships between their own vehicles and surrounding vehicles, pedestrians, and roads. They determine the direction of travel by observing lane markings and then use maps to determine long-term route planning. Similarly, autonomous vehicles should possess these abilities. Let's make a simple analogy:

1. Autonomous vehicles should be able to identify the types of surrounding vehicles and pedestrians in real time, recognize road signs and signals, such as common lane markings, traffic lights, and traffic signs. This is called the **perception** capability of the vehicle [12, 13].

2. The vehicle should be able to determine its own direction and position, as well as the positional relationship between itself and the aforementioned elements. This is also known as the **localization** capability of the vehicle [14–16].

3. The vehicle should be able to control the throttle, brake, steering wheel, and other actuators based on the aforementioned signal recognition results, and plan short-term and long-term driving routes. This is called the **planning and control** (P&C) capability of the vehicle [17, 18].

However, despite addressing the same problems, the capabilities of humans and computers differ greatly. Throughout the long history of evolution, humans have developed extremely powerful spatial **perception** abilities. We can understand the vast majority of objects in our field of vision in an instant, with minimal errors. We also

possess strong learning abilities; even when encountering unfamiliar objects, we instinctively avoid them. We can quickly understand the structure of the road ahead in any weather and scene, and even drive normally on roads without lane markings. We can also communicate with surrounding vehicles through lights and sounds, and predict their actions based on the behavior of other vehicles. In some defensive driving techniques, we can even infer potential dangers in blind spots. Due to these powerful understanding abilities, we can drive vehicles freely based solely on vision, without precise position and attitude information, unlike autonomous vehicles, which require expensive ranging devices like LiDAR, high-definition maps, and high-precision positioning to precisely control vehicle behavior (Figure 1-1).

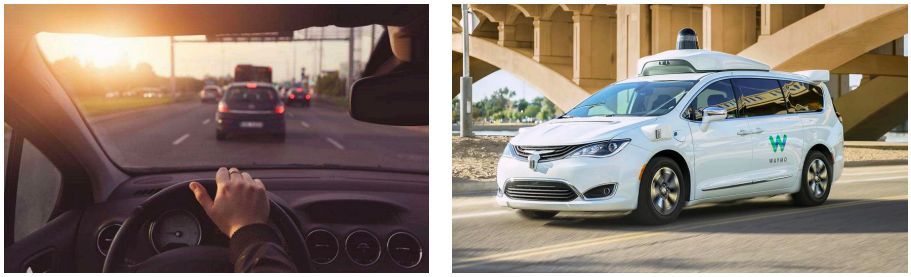


Figure 1-1: Cars driven by humans versus cars driven autonomously. Humans can drive solely based on vision, while autonomous vehicles currently rely on high-precision ranging devices and backend high-precision map services.

If we were to draw a comparison between birds and airplanes, our human driving abilities would resemble the effortless and natural flight of birds in the sky. However, for aircraft designers, should they make airplanes flap their wings like birds? The reality is quite different. Aircraft possess sophisticated control devices to manipulate airflow over their wings to generate the necessary lift; they also have precise measuring instruments to determine their own attitude and use modern control methods to maintain the desired orientation. The relationship between autonomous and human-driven vehicles is very similar to that of airplanes and birds. We can draw upon certain human abilities to design autonomous driving systems, but the resulting autonomous vehicles will inevitably differ significantly from human-driven ones. We don't necessarily need autonomous vehicles to behave exactly like human-driven ones. Autonomous vehicles should have their own design and operational logic.

In fact, autonomous driving vehicles are already capable of providing automated driving experiences. In several major cities in China (such as Beijing, Shanghai, Changsha, Chongqing, Wuhan, Shenzhen, among others), autonomous taxis (Robotaxis) has been opened to the public and can be experienced at any time. If you were to sit in an autonomous vehicle, you would notice that the steering wheel turns automatically, and braking and acceleration do not require human intervention. You might realize that if the vehicle were entirely controlled by a computer, there would be no need to install those steering wheels, pedals, or central control panels in the car. Additionally, you can see on the vehicle screen the planned route, surrounding vehicles and pedestrians, and data provided by HD maps. Since 2018, these vehicles have undergone several years of pilot testing but have not yet reached mainstream consumers.

On the other hand, if you were looking to buy a vehicle soon, most vehicles would advertise their autonomous driving features. They can automatically maintain a fixed

speed on highways, eliminating the need for you to operate the accelerator; they can also automatically steer to keep the vehicle in the center of the lane, relieving you of steering duties. Some vehicles even offer features like lane change assistance or automatic lane changing, so you can leave lane changing to the vehicle. They can even handle automatic following in congested traffic conditions. These features indeed assist drivers in vehicle operation. Moreover, these vehicles are tangible and can be purchased at widely accepted prices. Most of them use purely visual or predominantly visual sensor solutions.

Are all of these considered "autonomous driving"? If so, why is it difficult to purchase the former now, while the latter can appear directly on the consumer market?

This is precisely the situation that autonomous driving faces today: if we want to achieve driving capabilities similar to humans and have vehicles drive entirely on their own without the need for drivers, it requires a hefty price to implement such functionality. This cost could be attributed to laser radar sensors, backend map services, machine costs, research and development investments, and so on. When vehicles no longer require drivers, many business models will change accordingly. Taxi companies will no longer need drivers, food delivery companies will no longer need delivery personnel, and all operational personnel will only need to maintain autonomous driving vehicles. On the other hand, if we want to keep the price of vehicles within the reach of consumer-grade vehicles by using inexpensive, practical sensors, then we must acknowledge the current lack of reliability in existing algorithms, necessitating human driver supervision and intervention at any time, unable to completely replace human drivers. In fact, the entire autonomous driving industry is currently exploring along these two paths. The former path is highly likely to lead to fully autonomous driving, but currently, it seems too expensive to be directly targeted at consumers; the latter path is known as the "incremental" route, which is being used by various vehicle manufacturers, but it still has a significant distance from fully autonomous driving and is not suitable for tasks that require fully autonomous driving capabilities.

This divergence in paths reminds us that sometimes when discussing autonomous driving, different parties may not necessarily be talking about the same tasks. In fact, if we only care about "some" autonomous driving tasks, such as automatic following, lane keeping, and automatic lane changing, they do not require very complex technology or sensors. Although we sometimes refer to these vehicles as "autonomous driving" vehicles, and their manufacturers are also willing to claim that they are "fully autonomous driving," they still belong to the category of "assisted driving" functions according to standards. This also reminds us that there should be a clear and standardized delineation of autonomous driving capabilities.

In the international arena, researchers have long classified vehicles into five levels of autonomy, from Level 1 to Level 5 (SAE classification[1]), as shown in Table 1-1. Similar to this, China has its own "Classification of Automotive Driving Automation"[2], the summary of which is presented in 1-2. Generally, the various standards for classifying autonomous driving capabilities are primarily based on the following two points:

1. Whether the system requires human intervention, also known as **intervention**. **Assisted driving systems** require the driver to take over when intervention is needed, while **autonomous driving systems** strive to operate without the need for human intervention, allowing the removal of driver equipments such as steering

[1]Classification by the Society of Automotive Engineers, see SAE Standard J3016-202104: `https://www.sae.org/standards/content/j3016_202104`.

[2]See: GB/T: 40429-2021.

wheels and pedals. This is the key distinction between Level 2 and above Level 3 autonomous driving capabilities.

2. Whether the system operates in limited scenarios or can function in most normal scenarios relative to human drivers. This is the key difference between Level 4 and Level 5.

Table 1-1: SAE Automated Driving Levels

Level	L0	L1	L2	L3	L4	L5
Responsiblity	Driver			Computer		
Intervention	at all times			when required	not required	
Typical Func.	AEB BSD LDW ALC	LCC ACC	LCC+ACC	Traffic Jam Driving Auto Parking Auto Summoning	Robotaxi Robotruck	all conditions

Abbreviations: LDW: Lane Departure Warning, ACC: Adaptive Cruise Control, LCC: Lane Centering Control, BSD: Blind Spot Detection, AEB: Automatic Emergency Braking, ALC: Auto Lane Change.

Table 1-2: China Classification of Automotive Driving Automation

Driving Level	Name	Main Contents
L0	Emergency Assist.	Detection and response capabilities for certain events
L1	Partial Driver Assist.	Continuous execution of lateral and longitudinal control
L2	Combined Driver Assist.	Continuous execution of lateral and longitudinal control
L3	Conditionally Auto. Driving	Continuous execution of all driving tasks
L4	Highly Auto. Driving	User may refrain from taking over
L5	Fully Auto. Driving	Can autonomously drive in any environment

Therefore, although there are five to six levels in terms of classification, for professionals in the field of autonomous driving, the main concern lies in Levels 2 and 4. Level 2 autonomous driving vehicles can be directly targeted at consumers, enhancing driving comfort to a certain extent based on traditional vehicles. The functionalities currently achieved at Level 2 are not far from current laws and regulations and are gradually being popularized in some new car models. On the other hand, Level 4 vehicles should be capable of unmanned driving in most scenarios, able to address some business needs for automation, which is also considered by many researchers as a form of "driverless" mode. Although within the broad definition, both Level 2 and Level 4 are part of autonomous driving, in practical terms, there are fundamental differences in module design and implementation between them. Level 4 autonomous driving is most concerned with the **intervention rate**, requiring that the vehicle cannot be taken over by human without cause, thus placing high demands on various algorithm performances. On the other hand, all functions of Level 2 autonomous driving can be taken over by humans, thus emphasizing the recognition of which scenarios are **valid scenarios**, where Level 2 functionalities can be activated. Due to human intervention, Level 2 is much more tolerant of most algorithm metrics, placing greater emphasis on the presence of functions rather than complete automation.

1.1.2 Typical L4 Tasks

L2 or L4, that is the question.

However, this question shouldn't be answered by technical personnel. We should first ask, does a certain type of vehicle really need to drive completely autonomously?

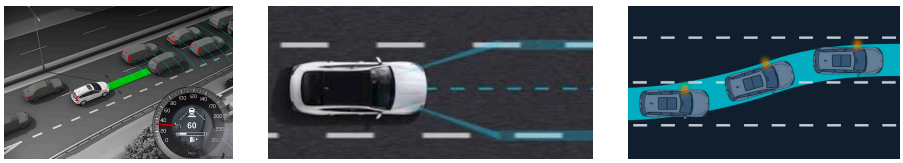


Figure 1-2: Typical L2 tasks: ACC, LCC, ALC.

Figure 1-3: Some applications of L4 passenger vehicles: autonomous taxis, buses, trucks, mining trucks

In some scenarios, the answer is “yes”. **Automation** is the core functionality of these vehicles. Without full automation, these vehicles lose their raison d’être. In other scenarios, we might say “not necessarily.” What we need more is safe driving with computers helping to **alleviate burdens**. If computers can provide more advanced functions, we are willing to accept them, but we also need to consider the cost of these functions. If the cost is too high, consumers won’t buy it.

The former belongs to typical L4 applications, and the entity doesn’t have to be limited to **vehicles**. In a broad sense, as long as a chassis carries sensors and has a certain level of automation, it can be considered a form of autonomous driving vehicle. From this perspective, whether the entity is a car or carries passengers is not the key to distinguishing autonomous driving. Whether the tasks they perform require **full automation** is the key to distinguishing their autonomous driving capabilities. For example, an autonomous delivery vehicle’s primary function is to autonomously deliver items to users. If this function isn’t fully automated and still requires a driver’s cooperation, then the business loses its main feature. An autonomous cleaning vehicle’s main function is to automatically cover cleaning in fixed scenes. If this process still requires human involvement, it’s not meaningful. For these businesses, **removing the driver** is their most important feature, so they belong to core L4 applications. These vehicles are not designed with driver cabins or driver positions (Figure 1-4).

Figure 1-4: Low-speed L4 applications: sweepers, delivery bots, patrol robot

On the other hand, we can also ask, do taxis need autonomous driving? Do trucks need autonomous driving? If there are no drivers, taxis can be operated solely by taxi companies, and trucks can also be operated solely by logistics companies, without the need to recruit drivers, only requiring vehicle maintenance. This business model is called Robotaxi and Robotruck, which is a completely different way from existing commercial models (see Figure 1-3). It imposes strict requirements on autonomous driving. Once a vehicle experiences a failure and requires human intervention, in the absence of a driver on board, it's difficult to have a human driver intervene in a timely manner, and a takeover event could easily turn into an accident.

Applications such as Robotaxi, Robotruck, Robobus, etc., technically also belong to L4 autonomous driving. Compared to autonomous driving vehicles for cleaning, delivery, inspection, etc., they have higher requirements for autonomous driving safety, stricter requirements for overall vehicle system stability, lower tolerance for risks and failures, and require more perception, high-precision positioning, mapping, and other technical support. If a low-speed vehicle experiences a failure, it won't directly lead to casualties, and most of the time, it can be remotely managed by technical personnel. However, if a passenger-carrying vehicle experiences an accident, the consequences can easily have substantial impacts at the company or even the community level. So, we ask, can the current level of technology support applications like Robotaxi? Unfortunately, this question currently does not have a definitive answer.

On one hand, autonomous driving systems are complex systems, unlike traditional electronic or mechanical switches, where it's easy to provide a functional safety verification plan or give clear reasons for failures. If an autonomous driving vehicle fails to recognize a vehicle in front and collides, within the current theoretical framework, we can't explain **why the system failed to recognize the vehicle ahead**. It's just a phenomenon that occurs in reality. Perhaps if a certain value in a certain program is increased by 0.001, this phenomenon won't occur, but it may make it impossible to recognize vehicles of a different color in different weather conditions. There are hundreds of millions of parameters like this, none of them have names, and they are connected and calculated in a way arbitrarily stipulated by humans. It's difficult to attribute the occurrence of a specific result to a parameter being too large or too small, or the calculation sequence between them not being reasonable enough. A brake system is almost impossible to fail, but a perception system is almost impossible to be 100% correct. In short, autonomous driving systems are difficult to precisely analyze what failure at a certain point may lead to what phenomenon, and provide convincing reasons, like traditional mechanical and electronic systems.

On the other hand, if it's difficult to verify the safety of autonomous driving vehicles from a theoretical level, can we statistically measure the stability of autonomous driving systems at an experimental level? This is indeed what many autonomous driving companies are doing now. Most L4 autonomous driving companies will statistically analyze the relationship between vehicle mileage and takeover times, for example, cal-

Table 1-3: 2021 report of the California Department of Motor Vehicles (DMV) on autonomous driving

Company	Num. of cars	Num. of Invent.	Miles	MPI
Waymo	693	292	2325843	7965
Cruise	138	21	876105	41719
Pony.ai	38	21	305617	14553
Zoox	85	21	155125	7387
Nuro	15	23	59100	2570
Mercedes-Benz	17	272	58613	215
WeRide	14	3	57966	19322
AutoX	44	1	50108	50108
DiDi	12	1	40745	40745
Argo AI	13	1	36734	36734
DeepRoute.ai	2	2	30872	15436
Nvidia	6	82	28004	342
Toyota	4	419	13959	33
Apple	37	663	13272	20
Aurora	7	9	12647	1405
Lyft	23	23	11200	487
Almotive	2	106	2976	28
Gatik ai	3	6	1924	321
Qualcomm	3	143	1635	11
Apollo	5	1	1468	1468
SF Motors	2	61	875	14
Nissan	5	17	508	39
Valeo	2	205	336	2
Easymile	1	222	320	1
Udelv	1	46	60	1
Inceptio.ai	2	0	39	-
UATC	3	31	14	0.5

culating the **miles per intervention** (MPI)[3], to measure the stability of the system. In the 2021 report of the California Department of Motor Vehicles (DMV) on autonomous driving, the MPI of some Chinese companies has reached the level of thousands to tens of thousands of kilometers (see Table 1-3)[4]. We generally believe that MPI is indeed an indicator of the overall autonomous driving capability of a vehicle, but so far, there is no very open and fair MPI testing method, and what we can see more are self-testing reports from various companies. They lack unified testing environments and standardized criteria for when to intervene. In terms of quantity and mileage, compared to traditional mass-produced vehicles, most L4 autonomous driving companies only have fleets consisting of dozens or hundreds of vehicles, and their road test scenarios are usually relatively simple. Compared to traditional manufacturers with monthly sales of tens of thousands of vehicles, their accumulated test mileage and number of scenarios are very limited.

The dilemma of whether to have vehicles first or autonomous driving technology first is facing all L4 autonomous driving companies. Without a sufficient number of vehicles, it's difficult to prove that an autonomous driving system is stable enough, and thus challenging to truly implement businesses like Robotaxi; however, if we focus solely on the vehicle itself without paying attention to autonomous driving technology, it's also challenging to attract enough technical talents and cultivate the team's tech-

[3]Sometimes also called miles per disengagement, MPD.
[4]Source: `http://www.evinchina.com/articleshow-217.html`

nical capabilities. Several years ago, it was a common problem that L4 technology companies didn't understand vehicles, and vehicle manufacturers didn't understand autonomous driving. Doing both requires enormous scale and determination, while collaboration between two different companies requires a high level of trust. Fortunately, major vehicle manufacturers have begun to pay attention to autonomous driving-related businesses (although currently focused on the easier-to-implement L2 business), and L2-related functionalities have been integrated into many vehicle models. Many practitioners also believe that as autonomous driving technology matures, L2 functionalities will gradually become richer and gradually approach L4 capabilities. Suppliers of autonomous driving-related sensors, algorithms, chips, and hardware will also play active roles in various vehicle models.

Even after overcoming technical issues, businesses like Robotaxi still face practical legal and social issues. After all, if L4 vehicles are widely deployed, other drivers will face the challenge of how to interact with driverless vehicles[5]. Will driverless vehicles understand the intentions of other vehicles changing lanes or overtaking? Will they avoid vehicles driving in the wrong direction? Will they detour around road sections undergoing sudden construction? Will they recognize children who have fallen on the road? Legally, how should the responsibility of driverless vehicles be defined if they collide with other vehicles? Should the responsibility lie with the developer of the driverless vehicle? Should the developers be held accountable if the collision occurs because the perception system failed to correctly identify pedestrians, or because the map annotation personnel incorrectly annotated the speed limit information of a road section, or because the satellite signal was weak that day, causing the vehicle to enter the wrong lane? These are questions that may be encountered in reality but are difficult to answer. In vehicles with human drivers, most safety responsibilities ultimately fall on the driver. Once the subject becomes a driverless vehicle, these responsibilities are difficult to distribute among various modules. It's unlikely that any company currently has the ability to assume these responsibilities on behalf of autonomous driving development companies. In short, discussions on many legal, ethical, and social issues related to autonomous driving will continue for many years to come.

However, autonomous driving still represents the direction of future technological advancement. Its overall prospects are promising, albeit with inevitable challenges along the way. Future passenger vehicles, low-speed vehicles, and robots will become increasingly intelligent, taking on more and more tasks in daily life. A series of technical issues derived from autonomous driving will also be fully addressed and discussed by researchers in various fields. Many things that appeared in science fiction movies will gradually become familiar landscapes in the coming years. For example, restaurant robots that the author fantasized about during their studies were once among the representatives of science fiction, but now they are widely present in major shopping malls, and major suppliers have begun price wars. Will autonomous driving vehicles also experience this situation in the coming years? We wait and see.

1.2 Localization and Mapping in Autonomous Driving

1.2.1 Why L4 Needs Localization and Mapping

This book primarily discusses the localization and mapping technologies in L4 autonomous driving. Before delving into the details, readers might ask a fundamental

[5]Throughout this book, the term **driverless vehicles** generally refers to L4 autonomous driving vehicles.

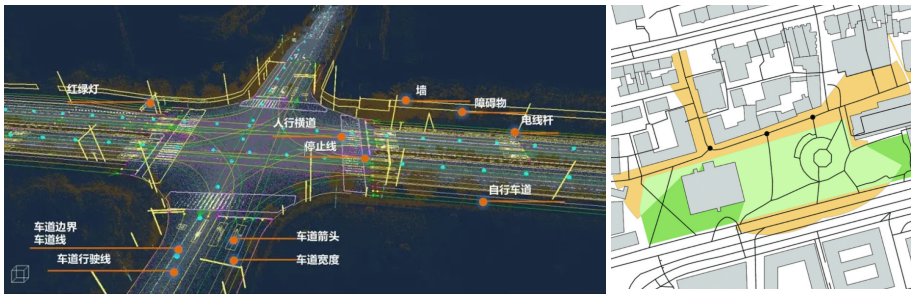


Figure 1-5: Differences between high-definition maps and traditional electronic navigation maps. Navigation maps represent roads and intersections using polygons and vectors, while high-definition maps also accurately mark lane positions, stop line positions, and provide detailed information about surrounding objects.

question: why does autonomous driving require localization and mapping?

That's an excellent question. My answer is, if high-precision localization and mapping weren't necessary for autonomous driving, that would be ideal. Unfortunately, at the current technological level, achieving low intervention rates in L4 autonomous driving still requires the use of high-precision localization and mapping. Conversely, if we're discussing L2 autonomous driving, which doesn't prioritize intervention rates, high-precision localization and mapping may not be necessary (although some L2 systems are also using high-precision maps) [19–21]. This contradicts the current level of intelligence and reliability. The smarter it is, the less reliable it becomes; the more reliable it is, typically meaning simpler in structure, the less intelligent it is. If we choose to accept the results of AI, we must also accept the mistakes made by AI. As for when L4 is needed and when L2 is needed, we've discussed this earlier.

Why does L4 need high-definition maps? This is because L4 and L2 have different goals. L2 autonomous driving doesn't concern itself with intervention rates, whereas achieving low intervention rates is the primary goal of L4, which determines that their technological paths have fundamental differences. L4-related technologies exhibit strong determinism. Many behaviors that may seem acceptable at the L2 level, such as turning into the wrong lane at an intersection or misreading a traffic light, would result in intervention in L4 and are not allowed to occur. Consequently, L2 leans more toward real-time perception and may even use perception results directly to construct a bird's eye view (BEV) [22–24], while L4 relies on offline maps [25–27]. In simple terms, L2 is more like "driving while looking at the road," whereas L4 is "driving in the map in the mind." "Driving while looking at the road" has the advantage of very direct logical relationships, closer to human behavior, but the disadvantage is dealing with the limitations and uncertainties of computer detection results. Vehicle cameras typically only see the ground lane lines a few dozen meters ahead. They may also be obscured by other vehicles, submerged in puddles, or, due to lighting conditions, shadows of roadside guardrails may be mistaken for lane lines. These outcomes could affect the vehicle's autonomous driving behavior, and these uncertainties must be considered at the control and decision-making levels. In contrast, high-definition maps in L4 are meticulously annotated by humans, with accurate positions for each lane. It's predetermined which lane they will continue onto, which direction to turn at intersections, and which traffic light to look at in three-dimensional space [28]. Even if no lane lines are visible in real-time images, L4 vehicles can accurately follow straight lines in the map

[29]. This approach comes with two costs: first, we need to create such a high-definition map in advance; second, we need to know our accurate position in this map [30, 31].

High-precision localization and mapping represent a strict, accurate concept. On the flip side, it brings about rigid, cumbersome business burdens. Maps essentially transform those limited, uncertain perception elements into static, precise data information through manual or post-processing methods (Figure 1-5). Maps carry out similar tasks to real-time perception but can provide correct results across unlimited ranges, greatly reducing the burden of perception [32]. Therefore, some have said that maps are a cheating sensor, essentially providing the answers directly to autonomous driving vehicles. With high-definition maps, the vehicle's burden of perception can be significantly alleviated. We only need to focus on dynamic pedestrian and vehicle information, without worrying about the shape and topology of road lanes. However, the balance between maps and perception is constantly changing. Some companies' high-definition maps are richer than others, even including information about obstacles, flowerbed shapes, etc., while others' solutions require the perception module to detect this information. Perhaps in the near future, the balance between maps and perception will change with technological iterations.

In current L4 autonomous driving solutions, most task elements are tied to maps. When users want to drive from point A to point B in a city, the autonomous driving vehicle first generates a lane-level path from point A to point B on the map. This path is different from the common navigation systems we're familiar with; it's at the lane level. The navigation system calculates which road, which intersection, and which lane to turn into. When executing autonomous driving tasks, the vehicle also strives to ensure that the actual executed path aligns with the results of high-definition map navigation. For this reason, the vehicle needs to know its real-time position on the map, which requires high-precision localization within lanes.

1.2.2 Contents and Production of High-Definition Maps

High-definition maps are essentially **structured data** [33]. Their basic elements consist of sections of lanes in the real world, as depicted in Figure 1-6. Various questions about lanes can be asked, such as:

1. What is the geometric shape of this lane? Is it straight, with bends, or curved?
2. Which lane is to its left, and which is to its right?
3. What is the speed limit? Is it a straight lane or a turning lane?
4. Is it a motorized lane or a non-motorized lane?
5. Which lanes is it connected to? Are they sequentially connected, or are there forks or mergers?

Questions of this sort are easily described and stored using **structures** in programs. Various programming and markup languages support struct syntax, making high-definition map software compatible with multiple languages, including JSON, Protocol Buffers (protobuf), XML, and others. Describing a complete lane's information can be quite extensive and challenging. Researchers worldwide have therefore developed standards for high-definition maps. Common standards include OpenDrive [34], LaneLet2 [35], and Apollo OpenDrive. These standards define ways to describe a lane, an intersection, and specifics of various traffic lights. Readers can refer to these standards for a comprehensive understanding of field information.

Figure 1-6: Common lane information in high-definition maps

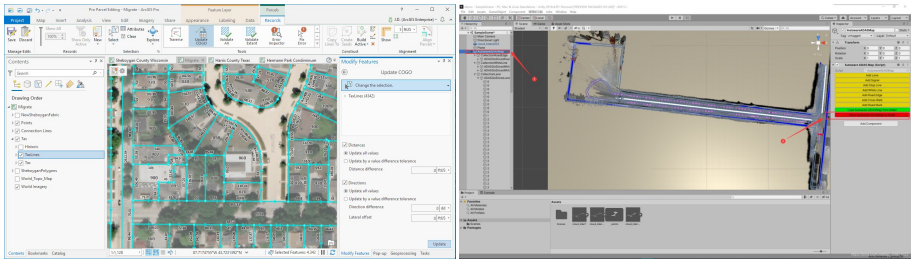


Figure 1-7: Common high-definition map editing software: ArcGIS and Autoware Map Tool

Most geometric elements in high-definition maps are described by points. For instance, a lane can be described by its center reference line plus its width, or by lines marking its left and right boundaries, which are composed of a series of lower-level points. The coordinates of each point can be expressed in terms of latitude and longitude or other global coordinate systems discussed later. Typically, they are represented by floating-point numbers. On the other hand, area-like elements can be described by polygons formed by multiple points, such as parking lots or buildings. Once this information is exported to files, it can be used for map rendering or for vehicle navigation and control.

Since high-definition maps are essentially made up of these lines and information, can't we generate them freely? Certainly, we can. We can even draw a road on paper and say its speed limit is 60 kilometers per hour; this can be considered a high-definition map, albeit with limited practical use. High-definition maps on computers are usually generated using specialized drawing software (such as ArcGIS, Autoware Map Tool, as seen in Figure 1-7), and some companies may develop their own drawing software. You can certainly start from a blank area and draw a virtual map. However, if we want the map to correspond to the real world, we need to find a way to first obtain the three-dimensional structure or two-dimensional aerial view of the real world. These serve as the data source for real-world high-definition maps.

The two-dimensional or three-dimensional data from the real world mainly come from the following sources:

1. Satellite imagery from remote sensing satellites. Satellite imagery can be obtained in various mapping software, but the highest resolution of civilian satellite

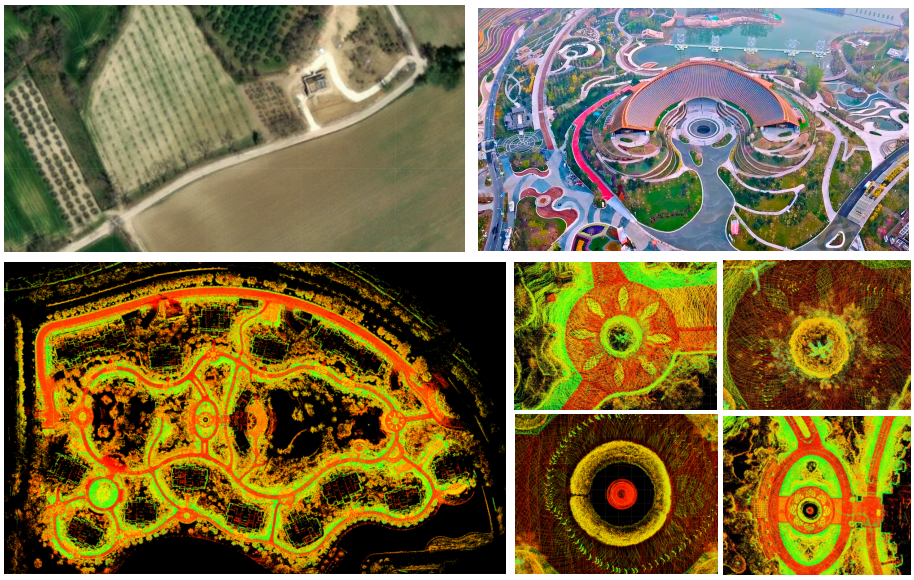


Figure 1-8: Data sources for high-definition maps: satellite imagery, drone aerial imagery, laser-generated point cloud maps (global and local)

imagery is at the level of a few meters, and images become blurred at around 10 meters. While they cover the entire globe and can be used for annotating electronic navigation maps, they are insufficient for high-definition maps, making it difficult to discern the specific positions of lanes and road surface details (see Figure 1-8, top left).

2. Aerial imagery from drones. Drones equipped with high-precision positioning devices can take top-down photographs at any altitude, which are then stitched together to create aerial imagery. This imagery can be highly detailed, but its coverage is limited, and flying drones is prohibited in many areas.

3. Three-dimensional map reconstruction using sensors carried by autonomous vehicles. The most common method is to use onboard LiDAR sensors to construct three-dimensional point clouds of the scene. These point clouds reflect the three-dimensional structure and brightness information of the scene and can effectively serve as a reference for map drawing. Since autonomous vehicles have relatively unrestricted ranges of travel, most autonomous driving companies use this method to construct point cloud maps [36, 37].

Figure 1-8 shows several different data sources. They follow a simple logic: the closer they are, the clearer they appear. Compared to remote sensing satellites orbiting tens of thousands of kilometers above the Earth, drones can hover tens of meters above the ground, while cars can directly capture images or measure distances in front of objects. Satellite images often struggle to discern road surface details, while drone images and LiDAR point clouds can reflect the texture of the road, the shapes of tree trunks and trees, and various small objects in the scene. Based on vehicle point clouds, we can annotate various detailed objects. If time permits, we can even annotate details such as the texture of tiles and the positions of tree trunks in high-definition maps.

1.3 Introduction to SLAM in this Book

The next question is how to use sensors onboard vehicles for high-precision point cloud reconstruction? What principles underlie this type of three-dimensional reconstruction? Apart from annotating maps, what other purposes do they serve? We will explore all of these questions throughout the content of this book. We will see that high-precision point clouds are the result of the combined action of a series of sensors. Their prices range from hundreds to hundreds of thousands, each serving different purposes. The core of the three-dimensional reconstruction system lies in estimating the vehicle's position and attitude at each moment, involving the vehicle's own **kinematic theory** and the **state estimation theory** used to estimate the vehicle's state using sensors. The following chapters of this book will introduce various sensor principles and methods in a certain order. The rough sequence is as follows:

1. Firstly, we need to introduce some basic geometric knowledge, including the sensor coordinate systems on the vehicle and the coordinate systems of the Earth. At the same time, we will review the basic knowledge of state estimation theory, including the Kalman filter and nonlinear optimization theory. This part of the knowledge has been introduced in my previous book [1], so we will not elaborate further but only review. In particular, this book will mainly use properties on $\mathrm{SO}(3)$ when dealing with rotation variables. Readers need to maintain a certain proficiency in this part of the content. This is the content of Chapter 2 of this book.

2. Chapters 3 and 4 will introduce two mainstream methods for processing **inertial measurement unit** (IMU) data. Chapter 3 introduces the classical Error State Kalman Filter (ESKF) and handles rotations in $\mathrm{SO}(3)$, while Chapter 4 mainly introduces preintegration methods. Since IMUs do not directly measure the physical state of the vehicle (translation and rotation) but rather measure their derivatives along the time axis (angular velocity and acceleration), we must introduce the differential relationship of the vehicle state and their various properties after integration. These are the main contents of Chapters 3 and 4.

3. Chapters 5 to 7 introduce the content of laser SLAM. Chapter 5 mainly covers basic point cloud processing methods, including how to represent laser point clouds, how to find their nearest neighbors, and how to rasterize them. We will implement some classical data structures ourselves. Chapters 6 and 7 respectively introduce 2D and 3D laser SLAM methods. We will first implement some laser registration methods: 2D and 3D ICP, NDT, probability grid, etc., then manage them using sub-map methods, and finally add loop detection to form a complete SLAM system. Chapter 7 also introduces loosely coupled laser-inertial odometry.

4. Chapters 8 to 10 introduce typical SLAM applications. Chapter 8 implements a tightly coupled laser-inertial odometry, each using an iterative error Kalman filter and a preintegration optimizer. Chapter 9 introduces offline point cloud map construction methods. We will rewrite some key algorithms into easily parallelized offline programs. Chapter 10 introduces methods for high-precision localization in existing point cloud maps. We will divide the point cloud map into small blocks in space and then use filters to achieve fusion localization of point clouds and inertial data.

The above is the main content of this book. We mainly focus on the application of SLAM in autonomous driving around the aspects of inertial navigation and laser point clouds. Most vehicles on open roads or park roads can be mapped and located in this way. However, this book will not delve into the annotation part of maps in detail because they are mainly drawn manually and do not involve much algorithmic content[6].

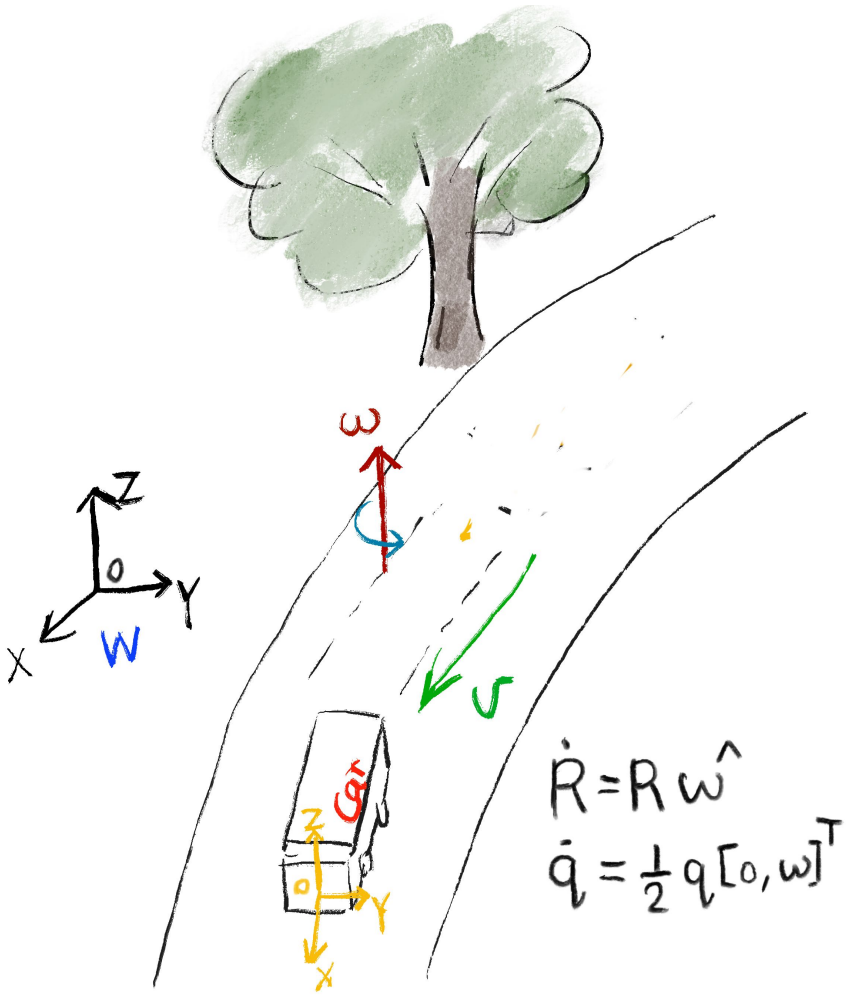
Except for this chapter, the end of each chapter will include a certain number of exercises. Readers should allocate time for exercises according to their own learning progress.

[6]The automatic generation of high-definition maps is also a key research direction in the field of autonomous driving, but the methods are quite different from the content covered in this book, and the results are not yet mature. We will not delve into this topic. Readers can refer to related literature, such as [38, 39].

Chapter 2

Quick Review of Basic Mathematical Concepts

Before delving into various sensor processing methods, let's review some fundamental mathematical concepts. This book largely follows the notation conventions established in "Introduction to Visual SLAM"[1]. To avoid redundancy, we must assume that readers are already familiar with the basic geometric knowledge presented in that book. This book does not elaborate on processes such as quaternion-to-rotation-matrix transformations in detail; instead, it briefly mentions their conclusions for readers to refer back to at any time. For topics not extensively covered in [1], this chapter provides additional explanations and derivations as appropriate.



$$\dot{R} = R\hat{\omega}$$

$$\dot{q} = \frac{1}{2} q[0, \omega]^T$$

因为汽车在运动，
要考虑它随时间的变化。

2.1 Geometry

2.1.1 Coordinate Systems

To describe the position and orientation of an autonomous vehicle, we should first define various coordinate systems for it. Firstly, we assume the existence of a fixed coordinate system in the world, known as the **world coordinate system** or **inertial coordinate system**. There are several ways to define this coordinate system in the real world, but in principle, it can be simply considered as a fixed coordinate system. When the vehicle moves in the world coordinate system, there exists a transformation relationship between the vehicle's own coordinate system (referred to as the **body coordinate system** or **body frame**) and the world system. This transformation relationship changes over time, allowing us to define the vehicle's **linear velocity**, **angular velocity**, **acceleration**, and other physical quantities. This constitutes the motion process of the vehicle.

However, explaining what linear velocity, angular velocity, and especially attitude mean from a mathematical perspective is not so intuitive. The attitude of the vehicle is usually described by a **rotation matrix** or **quaternion**. When they change over time, how many-dimensional vectors should be used to describe the angular velocity? How does the angular velocity vector act on the rotation matrix or quaternion? Are there any formal differences in their various definitions? Are they essentially the same? These are the questions this chapter aims to answer.

A three-dimensional coordinate system is composed of three vectors in space. Typically, we choose a set of unit orthogonal vectors to form a reference frame. For instance, if $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$ are the three vectors of the world coordinate system, it means that these three vectors have a length of 1 and their inner products are 0. In this case, we say that a coordinate system (reference frame) $E = \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ has been chosen. Then, any three-dimensional spatial vector $\mathbf{a}$ can be represented in this reference frame as:

$$\mathbf{a} = a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + a_3\mathbf{e}_3, \tag{2.1}$$

where (a_1, a_2, a_3) are the coordinates of the vector $\mathbf{a}$.

Please note that even without specifying a reference frame and coordinates, various operations can be performed between vectors. For example, the following operations can be performed between two vectors $\mathbf{a}$ and $\mathbf{b}$:

1. **Addition and subtraction**. The result of vector addition or subtraction is still a vector, following the parallelogram rule:

   $$\mathbf{c} = \mathbf{a} \pm \mathbf{b}. \tag{2.2}$$

   If the vectors have coordinates, the components are simply added or subtracted.

2. **Scalar multiplication**. Multiplying a vector by any scalar $k \in \mathbb{R}$ scales the vector:

   $$\mathbf{b} = k\mathbf{a}, \tag{2.3}$$

   resulting in another vector. When $\mathbf{a}$ has coordinates, these coordinates are scaled accordingly.

3. **Taking the length**. We can compute the length of a vector, denoted as:

   $$\|\mathbf{a}\|. \tag{2.4}$$

The length yields a scalar value. Mathematically, a vector's length can be zero or negative, for instance, in Minkowski space, but in the physical world of autonomous driving, we are concerned with vectors in Euclidean space, where their length is always non-negative.

4. **Dot product**. The dot product of two vectors yields the product of their lengths times the cosine of the angle between them, resulting in a scalar:

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\|\|\mathbf{b}\| \cos \langle \mathbf{a}, \mathbf{b} \rangle , \tag{2.5}$$

where if vectors have coordinates, the dot product results from the sum of the products of their respective components.

5. **Cross product**. The cross product of two vectors is another vector whose direction is perpendicular to the plane formed by the two vectors, and its magnitude is the product of their lengths times the sine of the angle between them. If vectors $\mathbf{a}, \mathbf{b}$ are defined in the $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ frame, the cross product is written as:

$$\mathbf{a} \times \mathbf{b} = \begin{Vmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{Vmatrix} = \begin{bmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{bmatrix} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \mathbf{b} \triangleq \mathbf{a}^{\wedge} \mathbf{b}. \tag{2.6}$$

The cross product can also be expressed as the usual matrix-vector multiplication, which requires expressing the first vector in a **skew-symmetric** matrix form[1]. We use the $^{\wedge}$ symbol to define this transformation:

$$\mathbf{a}^{\wedge} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} = \mathbf{A}. \tag{2.7}$$

Note that this operator is a **one-one mapping**, meaning that for any vector, there exists a unique corresponding skew-symmetric matrix, and vice versa. We use the $^{\vee}$ symbol to denote the mapping from skew-symmetric matrix to vector:

$$\mathbf{A}^{\vee} = \mathbf{a}. \tag{2.8}$$

The skew-symmetric matrix operator is a symbol widely used in the subsequent text; readers should pay attention to this notation. In other literature, it might also be denoted as $\mathbf{a}_{\times}, \mathbf{a}^{\times}, [\mathbf{a}]_{\times}, \hat{\mathbf{a}}$[40], all of which have the same meaning. This book uniformly adopts the $\wedge$ and $\vee$ symbols on the upper right, as they appear more concise.

Lastly, even when a reference frame is not specified, vectors can undergo the aforementioned operations. Their results are independent of the choice of reference frame. If a reference frame and coordinates are specified, then the aforementioned computations can also be represented using numerical values of coordinates.

An autonomous vehicle is equipped with various types of sensors. We typically assume that each sensor has its own reference frame, and their respective axis directions are defined according to the usage habits of each sensor. For example, in Figure 2-1, the IMU, 64-line lidar, and camera of the vehicle all define their own reference frames.

[1] A skew-symmetric matrix satisfies $\mathbf{A}^{\top} = -\mathbf{A}$.

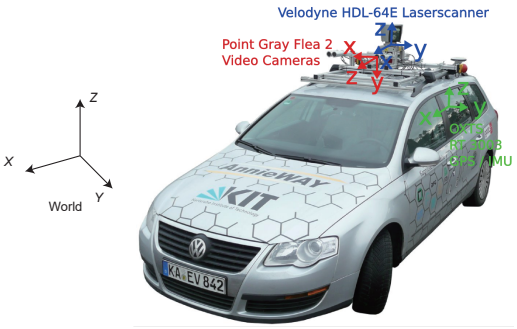


Figure 2-1: The body and world coordinate systems of a typical autonomous vehicle sensors

The vehicle body generally uses the **front-left-up**[2] or **right-front-up** order to define its coordinate system, while the camera coordinate system commonly adopts the **right-down-front** order. Consequently, there exist rotation and translation relationships between the coordinate systems of various sensors, which we characterize using rotation matrices and translation vectors.

Assuming a point $\mathbf{p}$ in the world coordinate system has coordinates $\mathbf{p}_w$, and its coordinates in the vehicle body coordinate system are $\mathbf{p}_b$, then we define the rotation matrix $\mathbf{R}_{wb}$ and the translation vector $\mathbf{t}_{wb}$, such that:

$$\mathbf{p}_w = \mathbf{R}_{wb}\mathbf{p}_b + \mathbf{t}_{wb}, \tag{2.9}$$

It is crucial for readers to understand the approach here. The key points are as follows:

1. Firstly, we define the transformation relationship between **coordinates**. $\mathbf{R}_{wb}$ and $\mathbf{t}_{wb}$ are used to handle coordinate transformations between vectors. Some materials deal with transformations between **coordinate axes** (or bases), interpreting rotation and translation as a transformation of a **coordinate axis** from one position to another. This definition is opposite to that of this book[3], so please be careful.

2. We can directly write $\mathbf{R}_{wb}, \mathbf{t}_{wb}$ as a **transformation matrix** $\mathbf{T}_{wb}$, expressing coordinate transformations in homogeneous form:

   $$\mathbf{p}_w = \mathbf{T}_{wb}\mathbf{p}_b. \tag{2.10}$$

   This transforms the discussion into properties of the transformation matrix $\mathbf{T}$. The specific form of $\mathbf{T}$ is:

   $$\mathbf{T}_{wb} = \begin{bmatrix} \mathbf{R}_{wb} & \mathbf{t}_{wb} \\ \mathbf{0} & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4}. \tag{2.11}$$

   s However, since the subsequent discussion will involve the IMU, which does not directly measure the differential of $\mathbf{T}$, we prefer to separate $\mathbf{R}$ and $\mathbf{t}$ rather than express them in the form of a transformation matrix.

[2]The convention "front-left-up" refers to the X axis pointing forward, the Y axis pointing left, and the Z axis pointing up, following the right-hand rule. The convention for "right-front-up" is analogous.

[3]One deals with transformations of coordinates, while the other deals with transformations of bases. Readers should be cautious.

3. Our subscript reading order is **from right to left**, meaning the subscript wb is right-multiplied with b to obtain variables in the w system. This makes writing and reading more fluid and intuitive. Different books handle the superscript and subscript of coordinate systems differently. Some write them on the left, some write them above, and some books even have four different markings (superscript, subscript, left, right) for one variable. This book uniformly uses the subscript wb to define various variables. Since all variables have the subscript wb, we omit these subscripts in the vast majority of content to strive for simplicity. We also need to discuss various variables at different times or iteration numbers, which will introduce subscripts related to time or iteration numbers. If combined with coordinate system subscripts, readers would have to face a large number of formulas with various superscripts and subscripts.

All three-dimensional rotation matrices form the **Special Orthogonal Group** ($\mathrm{SO}(3)$). It is a 3×3 real matrix that satisfies:

- The rotation matrix is an orthogonal matrix: $\mathbf{R}^\top = \mathbf{R}^{-1}$.
- The determinant of the rotation matrix is 1: $\det(\mathbf{R}) = 1$.

Additionally, a rotation matrix can also be represented by **quaternions** or **rotation vectors**. Below, we will review their definitions and conversion relationships.

2.1.2 Rotation Vectors

Rotation vectors, also known as **angle-axis**, correspond to the Lie algebra $\mathfrak{so}(3)$ of $\mathrm{SO}(3)$. Since $\mathfrak{so}(3)$ is the tangent space of $\mathrm{SO}(3)$, as we will see later, rotation vectors can also be used to express angular velocities.

Let's denote a rotation vector as $\mathbf{w} \in \mathbb{R}^3$, and it can be decomposed into direction and magnitude: $\mathbf{w} = \theta\mathbf{n}$. The conversion relationship from rotation vector to rotation matrix can be described by **Rodrigues' formula** or the exponential map on $\mathrm{SO}(3)$:

$$\mathbf{R} = \cos\theta\mathbf{I} + (1 - \cos\theta)\,\mathbf{n}\mathbf{n}^\top + \sin\theta\mathbf{n}^\wedge = \exp(\mathbf{w}^\wedge). \tag{2.12}$$

Here, exp can also be expanded using Taylor series and simplified to the formula on the left. To simplify notation, we denote the uppercase Exp as:

$$\mathrm{Exp}(\mathbf{w}) = \exp(\mathbf{w}^\wedge), \tag{2.13}$$

which eliminates one $^\wedge$ symbol, making the formula appear more concise in complex expressions.

Conversely, the conversion relationship from rotation matrix to rotation vector can be described by the logarithmic map:

$$\mathbf{w} = \log(\mathbf{R})^\vee = \mathrm{Log}(\mathbf{R}). \tag{2.14}$$

The computation method for the angle-axis is as follows. For the angle θ, we have:

$$\theta = \arccos\left(\frac{\mathrm{tr}(\mathbf{R}) - 1}{2}\right). \tag{2.15}$$

And the axis $\mathbf{n}$ is the unit eigenvector of $\mathbf{R}$ corresponding to the eigenvalue 1:

$$\mathbf{R}\mathbf{n} = \mathbf{n}. \tag{2.16}$$

2.1.3 Quaternions

Three-dimensional rotations can also be described by unit quaternions. Quaternions, also known as expanded complex numbers, consist of a real part and three imaginary parts. This book uses Hamiltonian quaternions[4], defined as:

$$\mathbf{q} = q_0 + q_1 i + q_2 j + q_3 k, \tag{2.17}$$

where q_0 is the real part, and q_1, q_2, q_3 are the imaginary parts. The imaginary units i, j, k satisfy the following multiplication rules:

$$\begin{cases} i^2 = j^2 = k^2 = -1 \\ ij = k, ji = -k \\ jk = i, kj = -i \\ ki = j, ik = -j \end{cases}. \tag{2.18}$$

To simplify notation, the three imaginary parts can be represented as a vector, and the quaternion can be expressed as a combination of scalar part s and vector part $\mathbf{v}$:

$$\mathbf{q} = [s, \mathbf{v}]^\top. \tag{2.19}$$

Using the vector part, a compact form of quaternion multiplication can be written.

Following the multiplication rules of quaternions, several commonly used quaternion calculation methods can be derived. We list them below.

1. *Addition and Subtraction*

   The addition and subtraction of quaternions $\mathbf{q}_a$ and $\mathbf{q}_b$ are given by:

   $$\mathbf{q}_a \pm \mathbf{q}_b = \left[s_a \pm s_b, \mathbf{v}_a \pm \mathbf{v}_b\right]^\top. \tag{2.20}$$

2. *Multiplication*

   Multiplication involves multiplying each term of $\mathbf{q}_a$ by each term of $\mathbf{q}_b$ and then summing them up, while following the rules defined by Equation (2.18). It can be expressed as:

   $$\begin{aligned} \mathbf{q}_a \mathbf{q}_b = & s_a s_b - x_a x_b - y_a y_b - z_a z_b \\ & + (s_a x_b + x_a s_b + y_a z_b - z_a y_b)\, i \\ & + (s_a y_b - x_a z_b + y_a s_b + z_a x_b)\, j \\ & + (s_a z_b + x_a y_b - y_a x_b + z_a s_b)\, k. \end{aligned} \tag{2.21}$$

   Though slightly complex, this form is well-structured. Expressing it in vector form and using inner and outer product operations results in a more concise expression:

   $$\mathbf{q}_a \mathbf{q}_b = \left[s_a s_b - \mathbf{v}_a^\top \mathbf{v}_b, s_a \mathbf{v}_b + s_b \mathbf{v}_a + \mathbf{v}_a \times \mathbf{v}_b\right]^\top. \tag{2.22}$$

   Under this definition of multiplication, the product of two real quaternions is still real, which is consistent with complex numbers. However, note that quaternion multiplication is usually non-commutative due to the presence of the cross-product term, unless $\mathbf{v}_a$ and $\mathbf{v}_b$ are collinear in $\mathbb{R}^3$, in which case the cross-product term becomes zero.

[4] According to different tastes, there are slight variations in the definition of quaternions. Hamiltonian is the most common and intuitive way of defining quaternions.

This book does not deliberately distinguish between standard multiplication and quaternion multiplication. Some materials may use symbols such as $\otimes$ to differentiate quaternion multiplication, but this book consistently uses standard multiplication. Quaternions are not multiplied with ordinary vectors or matrices, so **the meaning of multiplication should be clear**.

3. *Norm*

   The norm of a quaternion is defined as:

$$\|\mathbf{q}_a\| = \sqrt{s_a^2 + x_a^2 + y_a^2 + z_a^2}. \tag{2.23}$$

   It can be verified that the norm of the product of two quaternions equals the product of their norms, which ensures that the product of unit quaternions remains a unit quaternion:

$$\|\mathbf{q}_a \mathbf{q}_b\| = \|\mathbf{q}_a\| \|\mathbf{q}_b\|. \tag{2.24}$$

4. *Conjugate*

   The conjugate of a quaternion is obtained by negating the imaginary parts:

$$\mathbf{q}_a^* = s_a - x_a i - y_a j - z_a k = [s_a, -\mathbf{v}_a]^\top. \tag{2.25}$$

   Multiplying a quaternion by its conjugate yields a real quaternion with a real part equal to the square of its norm:

$$\mathbf{q}^* \mathbf{q} = \mathbf{q}\mathbf{q}^* = [s^2 + \mathbf{v}^\top \mathbf{v}, \mathbf{0}]^\top. \tag{2.26}$$

5. *Inverse*

   The inverse of a quaternion is given by:

$$\mathbf{q}^{-1} = \mathbf{q}^* / \|\mathbf{q}\|^2. \tag{2.27}$$

   According to this definition, the product of a quaternion and its inverse yields a real quaternion $\mathbf{1}$:

$$\mathbf{q}\mathbf{q}^{-1} = \mathbf{q}^{-1}\mathbf{q} = \mathbf{1}. \tag{2.28}$$

   If $\mathbf{q}$ is a unit quaternion, its inverse and conjugate are the same. Moreover, the inverse of a product obeys a property similar to matrices:

$$(\mathbf{q}_a \mathbf{q}_b)^{-1} = \mathbf{q}_b^{-1} \mathbf{q}_a^{-1}. \tag{2.29}$$

6. *Scalar Multiplication*

   Similar to vectors, quaternions can be multiplied by scalars:

$$k\mathbf{q} = [ks, k\mathbf{v}]^\top. \tag{2.30}$$

Representing Rotation with Quaternions

Rotation of a point can be expressed using quaternions. Let's assume a three-dimensional point in space $\mathbf{p} = [x, y, z] \in \mathbb{R}^3$, and a rotation specified by a unit quaternion $\mathbf{q}$. The point $\mathbf{p}$ undergoes a rotation to become $\mathbf{p}'$. If described using matrices, then $\mathbf{p}' = \mathbf{R}\mathbf{p}$. But how can we express this relationship using quaternions?

Firstly, represent the three-dimensional space point using a quaternion:

$$\mathbf{p} = [0, x, y, z]^\top = [0, \mathbf{v}]^\top. \tag{2.31}$$

This is equivalent to associating the three imaginary parts of the quaternion with the three axes in space. Then, the rotated point $\mathbf{p}'$ can be expressed as the following product:

$$\mathbf{p}' = \mathbf{q}\mathbf{p}\mathbf{q}^{-1}. \tag{2.32}$$

Here, the multiplication is quaternion multiplication, resulting in another quaternion. Finally, extract the imaginary part of $\mathbf{p}'$ to obtain the coordinates of the point after rotation. It can be verified that the real part of the computed result is 0, hence it is a pure imaginary quaternion.

Conversion from Quaternion to Rotation Matrix and Rotation Vector

Any unit quaternion describes a rotation, which can also be described using a rotation matrix or rotation vector. Now, let's examine the relationship between quaternions and rotation vectors, rotation matrices.

Before diving into that, it's worth mentioning that quaternion multiplication can also be expressed as a form of matrix multiplication. Let $\mathbf{q} = [s, \mathbf{v}]^\top$ be a quaternion. Define the following symbols $^+$ and $^\oplus$ as follows[41]:

$$\mathbf{q}^+ = \begin{bmatrix} s & -\mathbf{v}^\top \\ \mathbf{v} & s\mathbf{I} + \mathbf{v}^\wedge \end{bmatrix}, \quad \mathbf{q}^\oplus = \begin{bmatrix} s & -\mathbf{v}^\top \\ \mathbf{v} & s\mathbf{I} - \mathbf{v}^\wedge \end{bmatrix}, \tag{2.33}$$

where these symbols map quaternions into a 4×4 matrix. Thus, quaternion multiplication can be written in matrix form as follows:

$$\mathbf{q}_1^+ \mathbf{q}_2 = \begin{bmatrix} s_1 & -\mathbf{v}_1^\top \\ \mathbf{v}_1 & s_1\mathbf{I} + \mathbf{v}_1^\wedge \end{bmatrix} \begin{bmatrix} s_2 \\ \mathbf{v}_2 \end{bmatrix} = \begin{bmatrix} -\mathbf{v}_1^\top \mathbf{v}_2 + s_1 s_2 \\ s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + \mathbf{v}_1^\wedge \mathbf{v}_2 \end{bmatrix} = \mathbf{q}_1\mathbf{q}_2. \tag{2.34}$$

Similarly, it can be proven that:

$$\mathbf{q}_1\mathbf{q}_2 = \mathbf{q}_1^+\mathbf{q}_2 = \mathbf{q}_2^\oplus \mathbf{q}_1. \tag{2.35}$$

Then, let's consider the problem of rotating a point in space using quaternions. According to the previous discussion, we have:

$$\begin{aligned} \mathbf{p}' &= \mathbf{q}\mathbf{p}\mathbf{q}^{-1} = \mathbf{q}^+\mathbf{p}^+\mathbf{q}^{-1} \\ &= \mathbf{q}^+\mathbf{q}^{-1^\oplus}\mathbf{p}. \end{aligned} \tag{2.36}$$

Substituting the matrices corresponding to the two symbols, we obtain:

$$\mathbf{q}^+\left(\mathbf{q}^{-1}\right)^\oplus = \begin{bmatrix} s & -\mathbf{v}^\top \\ \mathbf{v} & s\mathbf{I} + \mathbf{v}^\wedge \end{bmatrix} \begin{bmatrix} s & \mathbf{v}^\top \\ -\mathbf{v} & s\mathbf{I} + \mathbf{v}^\wedge \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0}^\top & \mathbf{v}\mathbf{v}^\top + s^2\mathbf{I} + 2s\mathbf{v}^\wedge + (\mathbf{v}^\wedge)^2 \end{bmatrix}. \tag{2.37}$$

Since both $\mathbf{p}'$ and $\mathbf{p}$ are purely imaginary quaternions, the lower right corner of this matrix actually gives the transformation from quaternion to rotation matrix:

$$\mathbf{R} = \mathbf{v}\mathbf{v}^\top + s^2\mathbf{I} + 2s\mathbf{v}^\wedge + (\mathbf{v}^\wedge)^2. \tag{2.38}$$

To obtain the conversion formula from quaternion to rotation vector, take the trace of both sides of the above equation:

$$\begin{aligned} \mathrm{tr}(\mathbf{R}) &= \mathrm{tr}(\mathbf{v}\mathbf{v}^\top) + 3s^2 + 2s \cdot 0 + \mathrm{tr}((\mathbf{v}^\wedge)^2) \\ &= v_1^2 + v_2^2 + v_3^2 + 3s^2 - 2(v_1^2 + v_2^2 + v_3^2) \\ &= (1 - s^2) + 3s^2 - 2(1 - s^2) \\ &= 4s^2 - 1. \end{aligned} \tag{2.39}$$

Also, from Eq.(2.15), we have:

$$\begin{aligned} \theta &= \arccos(\frac{\mathrm{tr}(\mathbf{R}) - 1}{2}) \\ &= \arccos(2s^2 - 1). \end{aligned} \tag{2.40}$$

Thus,

$$\cos\theta = 2s^2 - 1 = 2\cos^2\frac{\theta}{2} - 1, \tag{2.41}$$

so:

$$\theta = 2\arccos s. \tag{2.42}$$

Regarding the rotation axis, if we substitute $\mathbf{q}$'s imaginary part for $\mathbf{p}$ in Eq.(2.36), it's easy to see that the vector formed by the imaginary part of $\mathbf{q}$ remains fixed during rotation, constituting the rotation axis. Thus, by normalizing it by its magnitude, we obtain the rotation axis. In summary, the conversion formula from quaternion to rotation vector can be expressed as follows:

$$\begin{cases} \theta = 2\arccos s \\ [n_x, n_y, n_z]^\top = \mathbf{v}^\top / \sin\frac{\theta}{2} \end{cases}. \tag{2.43}$$

Since quaternions require only four values to represent rotation, most programs choose quaternions as the underlying representation for rotations. They may provide interfaces for matrix operations, such as the previously mentioned $\vee$ or log operations, or interfaces for quaternions, such as retrieving the four components of a quaternion, and so on. When using these programs, we can simply use these matrix interfaces without concerning ourselves with their underlying storage format.

2.1.4 Lie Group and Lie Algebra

Three-dimensional rotations form the three-dimensional rotation group $\mathrm{SO}(3)$, with its corresponding Lie algebra denoted as $\mathfrak{so}(3)$; three-dimensional transformations form the three-dimensional transformation group $\mathrm{SE}(3)$, with its corresponding Lie algebra denoted as $\mathfrak{se}(3)$.

The mapping from Lie algebra elements to Lie group elements is known as the exponential mapping. For $\mathfrak{so}(3)$ to $\mathrm{SO}(3)$, the exponential mapping is given by:

$$\exp(\boldsymbol{\phi}^\wedge) = \mathbf{R}, \tag{2.44}$$

where the specific computation is provided by the Rodrigues' formula (2.12). The inverse mapping, known as the logarithm mapping, is denoted as:

$$\boldsymbol{\phi} = \log(\mathbf{R})^{\vee}, \tag{2.45}$$

with specific computation provided by equations (2.15) and (2.16).

We mainly utilize the combination of $\mathrm{SO}(3)$ with translation vectors to derive subsequent motion equations, filtering relationships, etc. We omit the introduction of $\mathrm{SE}(3)$ and $\mathfrak{se}(3)$.

2.1.5 BCH Linear Approximation on $\mathrm{SO}(3)$

The Baker-Campbell-Hausdorff (BCH) formula [42] provides a relationship between the addition of small quantities in the Lie algebra and the multiplication of small quantities in the Lie group, which is widely used for linearization of various functions. Here, we only present the conclusions.

In $\mathrm{SO}(3)$, for a rotation $\mathbf{R}$ (corresponding to the Lie algebra $\boldsymbol{\phi}$), left-multiplying it by a small rotation, denoted as $\Delta\mathbf{R}$, with the corresponding Lie algebra $\Delta\boldsymbol{\phi}$, results in $\Delta\mathbf{R} \cdot \mathbf{R}$ on the Lie group. According to the BCH approximation, in the Lie algebra, it is $\mathbf{J}_l^{-1}(\boldsymbol{\phi})\Delta\boldsymbol{\phi} + \boldsymbol{\phi}$. Thus, we can simply write:

$$\exp\left(\Delta\boldsymbol{\phi}^{\wedge}\right)\exp\left(\boldsymbol{\phi}^{\wedge}\right) = \exp\left(\left(\boldsymbol{\phi} + \mathbf{J}_l^{-1}\left(\boldsymbol{\phi}\right)\Delta\boldsymbol{\phi}\right)^{\wedge}\right). \tag{2.46}$$

Conversely, if we perform addition in the Lie algebra, adding $\Delta\boldsymbol{\phi}$ to $\boldsymbol{\phi}$, it can be approximated as multiplication with left and right Jacobians on the Lie group:

$$\exp\left(\left(\boldsymbol{\phi} + \Delta\boldsymbol{\phi}\right)^{\wedge}\right) = \exp\left(\left(\mathbf{J}_l(\boldsymbol{\phi})\Delta\boldsymbol{\phi}\right)^{\wedge}\right)\exp\left(\boldsymbol{\phi}^{\wedge}\right) = \exp\left(\boldsymbol{\phi}^{\wedge}\right)\exp\left(\left(\mathbf{J}_r(\boldsymbol{\phi})\Delta\boldsymbol{\phi}\right)^{\wedge}\right). \tag{2.47}$$

Where the left Jacobian for $\mathrm{SO}(3)$ is given by:

$$\mathbf{J}_l(\theta\mathbf{a}) = \frac{\sin\theta}{\theta}\mathbf{I} + (1 - \frac{\sin\theta}{\theta})\mathbf{a}\mathbf{a}^{\top} + \frac{1-\cos\theta}{\theta}\mathbf{a}^{\wedge} \tag{2.48}$$

$$\mathbf{J}_l^{-1}(\theta\mathbf{a}) = \frac{\theta}{2}\cot\frac{\theta}{2}\mathbf{I} + \left(1 - \frac{\theta}{2}\cot\frac{\theta}{2}\right)\mathbf{a}\mathbf{a}^{\top} - \frac{\theta}{2}\mathbf{a}^{\wedge}. \tag{2.49}$$

And the right Jacobian for $\mathrm{SO}(3)$ is:

$$\mathbf{J}_r(\boldsymbol{\phi}) = \mathbf{J}_l(-\boldsymbol{\phi}). \tag{2.50}$$

Since the Lie algebra $\boldsymbol{\phi}$ and $\mathbf{R}$ can be easily associated, sometimes we also simply denote $\mathbf{J}_r(\boldsymbol{\phi})$ as $\mathbf{J}_r(\mathbf{R})$ instead of $\mathbf{J}_r(\mathrm{Log}(\mathbf{R}))$. This can make the formulas look more concise. In many cases, we also omit the part inside the parentheses of $\mathbf{J}_r(\boldsymbol{\phi})$ and directly write $\mathbf{J}_r$ and $\mathbf{J}_l$.

All of the above content has been introduced in [1] already. If readers are interested in their detailed derivation process, please check [1], [2] or [4]. This book will directly use the conclusions introduced above.

2.2 Kinematics

Now let's consider a three-dimensional object in motion over time. In this section, we will explore various perspectives on expressing three-dimensional kinematics, which will be correlated with subsequent chapters. Examining three-dimensional kinematics will lead to a series of interesting discussions. Join us as we delve into it.

2.2.1 Kinematics from the Perspective of Lie Groups

Earlier, we discussed how the rotation and translation of an object can be described by $\mathbf{R}$ and $\mathbf{t}$, respectively (here, we omit the subscript wb denoting the coordinate frame). When they vary continuously with time, they become functions of time, $\mathbf{R}(t)$ and $\mathbf{t}(t)$. Obviously, the translational part is trivial, just a function with the codomain $\mathbb{R}^3$. Therefore, we focus on the rotational part.

Let's assume that $\mathbf{R}$ varies with time, i.e., $\mathbf{R}(t)$. According to the property of $\mathbf{R}$ being an orthogonal matrix:

$$\mathbf{R}^\top \mathbf{R} = \mathbf{I}, \tag{2.51}$$

it is not difficult to observe:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\mathbf{R}^\top \mathbf{R}\right) = \dot{\mathbf{R}}^\top \mathbf{R} + \mathbf{R}^\top \dot{\mathbf{R}} = \mathbf{0}, \tag{2.52}$$

which implies:

$$\mathbf{R}^\top \dot{\mathbf{R}} = -(\mathbf{R}^\top \dot{\mathbf{R}})^\top. \tag{2.53}$$

It can be seen that $\mathbf{R}^\top \dot{\mathbf{R}}$ is a skew-symmetric matrix, and a skew-symmetric matrix can be expressed in vector form using the skew-symmetric symbol $^\wedge$. Let's take $\boldsymbol{\omega}^\wedge \in \mathbb{R}^{3\times 3} = \mathbf{R}^\top \dot{\mathbf{R}}$, then we can write $\mathbf{R}$ in the form of a differential equation:

$$\dot{\mathbf{R}} = \mathbf{R}\boldsymbol{\omega}^\wedge. \tag{2.54}$$

This equation is also known as the **Poisson equation**[43]. It's worth noting that we could also start from $\mathbf{R}\mathbf{R}^\top = \mathbf{I}$, define $\boldsymbol{\omega}^\wedge = \dot{\mathbf{R}}\mathbf{R}^\top$, and obtain the result $\dot{\mathbf{R}} = \boldsymbol{\omega}^\wedge \mathbf{R}$. These two forms are essentially equivalent, just different in appearance.

If we only consider instantaneous changes, then at a fixed time t, $\boldsymbol{\omega}$ can be considered constant. In physical terms, we call $\boldsymbol{\omega}$ the **instantaneous angular velocity**. Given an initial rotation matrix $\mathbf{R}(t_0)$ at time t_0, the solution to the above differential equation is:

$$\mathbf{R}(t) = \mathbf{R}(t_0)\exp(\boldsymbol{\omega}^\wedge (t - t_0)). \tag{2.55}$$

If readers are familiar with the knowledge of Lie groups and Lie algebras, it's easy to recognize that Equation (2.55) represents the exponential mapping on $\mathrm{SO}(3)$. Let $\Delta t = t - t_0$, then this equation can also be written as:

$$\mathbf{R}(t) = \mathbf{R}(t_0)\mathrm{Exp}(\boldsymbol{\omega}\Delta t). \tag{2.56}$$

From another perspective, we can also expand $\mathbf{R}(t)$ around time t_0 using Taylor series, and the first-order approximation is:

$$\begin{aligned} \mathbf{R}(t_0 + \Delta t) &\approx \mathbf{R}(t_0) + \dot{\mathbf{R}}(t_0)\Delta t \\ &= \mathbf{R}(t_0) + \mathbf{R}(t_0)\boldsymbol{\omega}^\wedge \Delta t \\ &= \mathbf{R}(t_0)(\mathbf{I} + \boldsymbol{\omega}^\wedge \Delta t). \end{aligned} \tag{2.57}$$

This reveals the approximate form of the exponential mapping:

$$\mathrm{Exp}(\boldsymbol{\omega}\Delta t) = \mathbf{I} + \boldsymbol{\omega}^\wedge \Delta t + \frac{1}{2}(\boldsymbol{\omega}^\wedge \Delta t)^2 + \dots \tag{2.58}$$

By comparing the above equations, we can see that:

1. Equation (2.56) is the discrete-time form of Equation (2.55).
2. Equation (2.57) is the linear approximation of Equation (2.56).

These two sets of equations are very useful in dealing with angular velocities, and we will continue to use them in the subsequent discussion.

2.2.2 Kinematics from the Perspective of Quaternions

Now let's examine how the kinematic equations change if we use quaternions to represent rotations. This is an alternative description of the same problem from a different perspective. Investigating this issue can help us establish connections between different mathematical representations. We know that the rotation of a vector by quaternions should take the form given by Equation (2.36), and quaternions themselves carry the unit constraint $\mathbf{q}\mathbf{q}^* = \mathbf{q}^*\mathbf{q} = \mathbf{1}$. Similar to the case of $\mathrm{SO}(3)$, starting from $\mathbf{q}^*\mathbf{q} = \mathbf{1}$, if we differentiate both sides with respect to time, we get:

$$\dot{\mathbf{q}^*}\mathbf{q} + \mathbf{q}^*\dot{\mathbf{q}} = \mathbf{0}, \tag{2.59}$$

which leads to:

$$\mathbf{q}^*\dot{\mathbf{q}} = -\dot{\mathbf{q}^*}\mathbf{q} = -(\mathbf{q}^*\dot{\mathbf{q}})^*. \tag{2.60}$$

Thus, $\mathbf{q}^*\dot{\mathbf{q}}$ is a pure quaternion (with zero real part). We can denote a pure quaternion as $\boldsymbol{\varpi} = [0, \underbrace{\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \boldsymbol{\omega}_3}_{\boldsymbol{\omega}}]^\top \in \mathcal{Q}$, so we have:

$$\mathbf{q}^*\dot{\mathbf{q}} = \boldsymbol{\varpi}. \tag{2.61}$$

Multiplying both sides by $\mathbf{q}$, we get:

$$\dot{\mathbf{q}} = \mathbf{q}\boldsymbol{\varpi}. \tag{2.62}$$

This equation is very similar to Equation (2.54). Analogous to the case of $\mathrm{SO}(3)$, we can also discuss the instantaneous angular velocity, Lie algebra, exponential mapping, and logarithmic mapping near time t. When considering instantaneous changes, we can treat $\boldsymbol{\varpi}$ as a constant value, so the solution to the above differential equation is:

$$\mathbf{q}(t) = \mathbf{q}(t_0)\exp(\boldsymbol{\varpi}\Delta t), \tag{2.63}$$

where we used the quaternion exponential mapping. Let's take a brief pause in the derivation and introduce the quaternion exponential mapping in the usual sense.

For any pure quaternion $\boldsymbol{\varpi} = [0, \boldsymbol{\omega}]^\top \in \mathcal{Q}$, its exponential mapping is defined as:

$$\exp(\boldsymbol{\varpi}) = \sum_{k=0}^{\infty} \frac{1}{k!}\boldsymbol{\varpi}^k. \tag{2.64}$$

Separating its direction and magnitude, let $\boldsymbol{\varpi} = \mathbf{u}\theta$, where θ is the magnitude of $\boldsymbol{\varpi}$ and $\mathbf{u}$ is the unit imaginary quaternion. Since $\mathbf{u}$ is an unit imaginary quaternion, we have:

$$\mathbf{u}^2 = -\mathbf{1}, \quad \mathbf{u}^3 = -\mathbf{u}, \tag{2.65}$$

which is similar to the self-multiplication property of unit imaginary numbers and can be used to simplify higher-order terms. Using this property, we can derive:

$$\begin{aligned}
\exp(\mathbf{u}\theta) &= 1 + \mathbf{u}\theta - \frac{1}{2!}\theta^2 - \frac{1}{3!}\theta^3\mathbf{u} + \frac{1}{4!}\theta^4 + \dots \\
&= \underbrace{\left(1 - \frac{1}{2!}\theta^2 + \frac{1}{4!}\theta^4 - \dots\right)}_{\cos\theta} + \underbrace{\left(\theta - \frac{1}{3!}\theta^3 + \frac{1}{5!}\theta^5 - \dots\right)}_{\sin\theta}\mathbf{u} \\
&= \cos\theta + \mathbf{u}\sin\theta.
\end{aligned} \tag{2.66}$$

This formula is very similar to Euler's formula for complex numbers:

$$\exp(i\theta) = \cos\theta + i\sin\theta, \tag{2.67}$$

and it's indeed its extension to quaternions.

Substituting the pure imaginary ϖ, we obtain:

$$\exp(\varpi) = [\cos\theta, \mathbf{u}\sin\theta]^\top. \tag{2.68}$$

Also, because ϖ is an imaginary quaternion, we have:

$$\|\exp(\varpi)\| = \cos^2\theta + \sin^2\theta\|\mathbf{u}\|^2 = 1. \tag{2.69}$$

So, the result of the exponential mapping of an imaginary quaternion is a unit quaternion, which is also a mapping relationship between unit quaternions and pure quaternions. We can also think of the imaginary quaternion ϖ as a quaternion form of the Lie algebra. Therefore, an obvious question arises: what is the relationship between the quaternion form of the Lie algebra and the rotation vector form of the Lie algebra?

2.2.3 Conversion between Lie Algebra of Quaternions and Rotation Vectors

Consider a rotation matrix $\mathbf{R}$ and its rotation vector $\boldsymbol{\phi}$. Obviously, their relationship is described by the exponential mapping:

$$\mathbf{R} = \mathrm{Exp}(\boldsymbol{\phi}) = \mathrm{Exp}(\theta\mathbf{n}), \tag{2.70}$$

where $\mathbf{n}$ is the direction of the rotation vector and θ is its magnitude. We also assume that this rotation can be expressed by $\mathbf{q} = \mathrm{Exp}(\varpi)$, where ϖ is a pure imaginary quaternion $[0, \boldsymbol{\omega}]^\top$. Now let's examine the transformation relationship between these two representations.

From Equation (2.43), we know that the quaternion corresponding to $\mathbf{R}$ is:

$$\mathbf{q} = [\cos\frac{\theta}{2}, \mathbf{n}\sin\frac{\theta}{2}], \tag{2.71}$$

By comparing with Equation (2.68), it's easy to see the relationship between ϖ and $\boldsymbol{\phi}$:

$$\varpi = [0, \frac{1}{2}\boldsymbol{\phi}]^\top, \quad \text{or } \boldsymbol{\omega} = \frac{1}{2}\boldsymbol{\phi}. \tag{2.72}$$

We miraculously discover that the angular velocity expressed by quaternions is exactly half of the $\mathrm{SO}(3)$ Lie algebra! This is because when using quaternions to rotate a vector, we need to multiply corresponding parts twice. Due to this "half" relationship, the Lie algebra corresponding to quaternions is slightly different from $\mathfrak{so}(3)$. To maintain the continuity of derivation and writing, we use a unified Exp relationship to combine the two definitions. In summary, for a three-dimensional instantaneous angular velocity (or the update quantity of an optimization function) $\boldsymbol{\omega} \in \mathbb{R}^3$, we define its kinematic form on $\mathrm{SO}(3)$ as:

$$\dot{\mathbf{R}} = \mathbf{R}\boldsymbol{\omega}^\wedge \tag{2.73}$$

Its corresponding exponential mapping is:

$$\mathbf{R} = \mathrm{Exp}(\boldsymbol{\omega}) = \exp(\boldsymbol{\omega}^\wedge), \tag{2.74}$$

or, if this quantity is the update amount of a pure imaginary quaternion (typically obtained from solving an optimization function), then the corresponding quaternion should only update half of it. According to the definition in Equation (2.62), the quaternion kinematic equation and exponential mapping can be written as:

$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q}[0, \boldsymbol{\omega}]^\top, \tag{2.75}$$

which can usually be simplified as[5]:

$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q}\boldsymbol{\omega}, \tag{2.76}$$

Here, the coefficient $1/2$ is used to unify the definition of angular velocity on $\mathrm{SO}(3)$ with quaternion angular velocity, so this equation differs from Equation (2.62). Readers should also note that this equation implies that the three-dimensional vector $\boldsymbol{\omega}$ is first converted to a quaternion before multiplying with $\mathbf{q}$, rather than directly multiplying $\mathbf{q}$ by $\boldsymbol{\omega}$.

The quaternion exponential mapping can also be written similarly as:

$$\mathbf{q} = \exp(\frac{1}{2}[0, \boldsymbol{\omega}]^\top) \triangleq \mathrm{Exp}(\boldsymbol{\omega}). \tag{2.77}$$

If $\boldsymbol{\omega}$ is small, then $\cos(\frac{\theta}{2}) \approx 1, \mathbf{n}\sin\frac{\theta}{2} \approx \mathbf{n}\frac{\theta}{2}$, and the exponential mapping has a simplified form:

$$\mathrm{Exp}(\boldsymbol{\omega}) \approx [1, \frac{1}{2}\boldsymbol{\omega}], \tag{2.78}$$

So the quaternion update formula can be simplified as[6]:

$$\mathbf{q}\mathrm{Exp}(\boldsymbol{\omega}) \approx \mathbf{q}[1, \frac{1}{2}\boldsymbol{\omega}], \tag{2.79}$$

However, a significant disadvantage of this equation compared to the update equation on $\mathrm{SO}(3)$ is that the quaternion on the right-hand side is not a unit quaternion, so after long-term updates, it needs to be re-normalized [4]. This problem does not exist for rotation matrices, as $\mathrm{Exp}(\boldsymbol{\omega})$ is always a rotation matrix.

Thus far, we have introduced the kinematics from the perspectives of $\mathrm{SO}(3)$ and quaternions, as well as their conversion relationship. With this relationship, we can use either rotation matrices or quaternions when writing the motion equations of a vehicle or when calculating the Jacobian matrices of optimization problems, just remember the coefficient of $1/2$. We can also mix quaternions and rotation matrices, just remember that when updating variables, quaternions only need to be updated by half.

In addition, we can also consider kinematics at the level of the Lie algebra $\mathfrak{so}(3)$. For the translation part, it can be viewed as independent three-dimensional variables or collectively considered in $\mathrm{SE}(3)$. In practice, these expressions are interchangeable without essential differences, but there may be differences in the ease of operation. We introduce several other expressions in this section, but only one of them will be detailed in subsequent chapters.

[5] Note that the meaning of $\boldsymbol{\omega}$ has changed here. In the previous equation, it was a three-dimensional vector, while in the next equation, it is a quaternion.

[6] In this equation, there is no need to change the definition of $\boldsymbol{\omega}$, it is still a three-dimensional vector.

2.2.4 Other Kinematic Representations

Kinematics on $\mathfrak{so}(3)$

To give some physical meaning to mathematical symbols, we will use $\boldsymbol{\omega}$ to express angular velocity and $\boldsymbol{\phi}$ to express rotation vectors in the future. The Rodrigues formula tells us that $\mathbf{R} = \mathrm{Exp}(\boldsymbol{\phi})$. Now we want to examine the derivative of $\boldsymbol{\phi}$ with respect to time and its relationship with the instantaneous angular velocity $\boldsymbol{\omega}$.

The BCH formula gives the relationship between increments on the Lie group and the Lie algebra. Assuming that at time t to $t + \Delta t$, $\boldsymbol{\phi}(t)$ changes to $\boldsymbol{\phi}(t) + \Delta\boldsymbol{\phi}$ on $\mathfrak{so}(3)$, and at the same time $\mathrm{SO}(3)$ changes from $\mathbf{R}$ to $\mathbf{R} \cdot \Delta\mathbf{R}$, then according to the BCH approximation, we have:

$$\Delta\mathbf{R} = \mathrm{Exp}(\mathbf{J}_r \Delta\boldsymbol{\phi}), \tag{2.80}$$

On the $\mathrm{SO}(3)$ level, according to the definition of angular velocity, we have $\dot{\mathbf{R}} = \mathbf{R}\boldsymbol{\omega}^\wedge$, so:

$$\begin{aligned} \mathbf{R}\boldsymbol{\omega}^\wedge &= \dot{\mathbf{R}} = \lim_{\Delta t \to 0} \frac{\mathbf{R}(t+\Delta t) - \mathbf{R}(t)}{\Delta t} \\ &\approx \lim_{\Delta t \to 0} \frac{\mathbf{R}(t)\mathrm{Exp}(\mathbf{J}_r \Delta\boldsymbol{\phi}) - \mathbf{R}(t)}{\Delta t} \\ &\approx \lim_{\Delta t \to 0} \frac{\mathbf{R}(t)(\mathrm{Exp}(\mathbf{J}_r \Delta\boldsymbol{\phi}) - \mathbf{I})}{\Delta t} = \mathbf{R}(\mathbf{J}_r \dot{\boldsymbol{\phi}})^\wedge, \end{aligned} \tag{2.81}$$

where the last equality requires a Taylor expansion of the Exp function. By comparing the left and right sides, we easily obtain:

$$\boldsymbol{\omega} = \mathbf{J}_r \dot{\boldsymbol{\phi}}, \tag{2.82}$$

or:

$$\dot{\boldsymbol{\phi}} = \mathbf{J}_r^{-1} \boldsymbol{\omega}. \tag{2.83}$$

This shows the relationship between the time derivative on $\mathfrak{so}(3)$ and the instantaneous angular velocity on $\mathrm{SO}(3)$. In principle, we can also use this quantity to derive subsequent filters or optimizers. However, its physical meaning is not as intuitive as $\boldsymbol{\omega}$, so very few people actually choose this method.

Kinematics on $\mathrm{SO}(3) + \mathbf{t}$

We can incorporate linear velocity into consideration. For example, let $\mathbf{v} = \dot{\mathbf{t}}$, then the system kinematic equations can be written as:

$$\dot{\mathbf{R}} = \mathbf{R}\boldsymbol{\omega}^\wedge, \quad \dot{\mathbf{t}} = \mathbf{v}. \tag{2.84}$$

This approach is the simplest and most intuitive, and is widely adopted.

Kinematics on $\mathrm{SE}(3)$

We can also derive kinematics on $\mathrm{SE}(3)$ and make it consistent with the exponential mapping on $\mathrm{SO}(3)$. This requires some modifications to the linear velocity part. Let the transformation matrix be:

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \in \mathrm{SE}(3), \tag{2.85}$$

then its time derivative is:

$$\dot{\mathbf{T}} = \begin{bmatrix} \dot{\mathbf{R}} & \dot{\mathbf{t}} \\ \mathbf{0}^\top & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{R}\boldsymbol{\omega}^\wedge & \mathbf{v} \\ \mathbf{0}^\top & 0 \end{bmatrix}. \tag{2.86}$$

For instance, to achieve the kinematics on $\mathrm{SE}(3)$ in the right-multiplication model, we want to obtain the form $\dot{\mathbf{T}} = \mathbf{T}\boldsymbol{\xi}^\wedge$[7], let $\boldsymbol{\xi} = [\boldsymbol{\rho}, \boldsymbol{\phi}]^\top$, then:

$$\begin{bmatrix} \mathbf{R}\boldsymbol{\omega}^\wedge & \mathbf{v} \\ \mathbf{0}^\top & 0 \end{bmatrix} = \mathbf{T} \begin{bmatrix} \boldsymbol{\phi}^\wedge & \boldsymbol{\rho} \\ \mathbf{0}^\top & 0 \end{bmatrix}. \tag{2.87}$$

It is not difficult to derive:

$$\boldsymbol{\phi} = \boldsymbol{\omega}, \quad \boldsymbol{\rho} = \boldsymbol{R}^\top \mathbf{v}. \tag{2.88}$$

Therefore, by defining $\boldsymbol{\xi} = [\mathbf{R}^\top \mathbf{v}, \boldsymbol{\omega}]^\top$, we can obtain the kinematics on $\mathrm{SE}(3)$ as:

$$\dot{\mathbf{T}} = \mathbf{T}\boldsymbol{\xi}^\wedge. \tag{2.89}$$

Kinematics on $\mathfrak{se}(3)$

Let the Lie algebra be $\boldsymbol{\varphi}$. To derive the kinematics of $\boldsymbol{\varphi}$, we still use the approach in Section 2.2.1. According to the BCH approximation, when $\boldsymbol{\varphi}$ increases by $\Delta\boldsymbol{\varphi}$, $\mathbf{T}$ right-multiplies $\Delta\mathbf{T}$. Analogously to the previous approach, we can write:

$$\dot{\mathbf{T}} = \mathbf{T}\boldsymbol{\xi}^\wedge = \lim_{\Delta t \to 0} \frac{\mathbf{T}(t)\,\mathrm{Exp}(\mathcal{J}_r \Delta\boldsymbol{\varphi}) - \mathbf{T}(t)}{\Delta t} \tag{2.90}$$

$$= \mathbf{T}\mathcal{J}_r \dot{\boldsymbol{\varphi}}^\wedge. \tag{2.91}$$

Here, some intermediate steps are omitted. Finally, we obtain:

$$\dot{\boldsymbol{\varphi}} = \mathcal{J}_r^{-1} \boldsymbol{\xi}. \tag{2.92}$$

This can also be used to characterize the kinematics on the Lie algebra. However, in these expressions, only the $\mathrm{SO}(3) + \mathbf{t}$ representation corresponds to the actual physical meaning, and the other representations require some degree of transformation. In a large number of papers, researchers default to using the simplest kinematic representation, i.e., rotation plus translation. In practice, there is no need to introduce unnecessary complications in theory, so we **default to using kinematics with rotation and translation**, but the rotation representation can freely use either $\mathrm{SO}(3)$ or quaternions (corresponding to different update quantities).

2.2.5 Linear Velocity and Acceleration

Now let's consider the transformation relationship of linear velocity and acceleration between different coordinate systems. For simplicity, we consider the transformation of linear velocity and acceleration between two coordinate systems with **only rotational relationship**.

Consider coordinate systems 1 and 2. A certain vector $\mathbf{p}$ has coordinates $\mathbf{p}_1, \mathbf{p}_2$ in the two systems, and it is obvious that they satisfy the relationship $\mathbf{p}_1 = \mathbf{R}_{12}\mathbf{p}_2$, which is a simple geometric relationship.

[7] The $\wedge$ symbol on $\mathrm{SE}(3)$ is defined as: $\boldsymbol{\xi}^\wedge = \begin{bmatrix} \boldsymbol{\phi}^\wedge & \boldsymbol{\rho}^\wedge \\ \mathbf{0}^\top & 0 \end{bmatrix} \in \mathbb{R}^{4\times 4}$, where $\boldsymbol{\phi}$ is the rotation part and $\boldsymbol{\rho}$ is the translation part.

Now we consider the case where $\mathbf{p}$ varies with time, while the two coordinate systems also undergo rotation. We want to remind the reader that **the velocity vector of p in the two systems is different**; it is not the expression of the same vector in different coordinate systems. Let's see why.

Taking the time derivative of the above equation, we have:

$$\begin{aligned}\dot{\mathbf{p}}_1 &= \dot{\mathbf{R}}_{12}\mathbf{p}_2 + \mathbf{R}_{12}\dot{\mathbf{p}}_2 \\ &= \mathbf{R}_{12}\boldsymbol{\omega}^{\wedge}\mathbf{p}_2 + \mathbf{R}_{12}\dot{\mathbf{p}}_2 \\ &= \mathbf{R}_{12}(\boldsymbol{\omega}^{\wedge}\mathbf{p}_2 + \dot{\mathbf{p}}_2).\end{aligned} \tag{2.93}$$

Traditionally, we denote $\dot{\mathbf{p}}_1 = \mathbf{v}_1, \dot{\mathbf{p}}_2 = \mathbf{v}_2$, then we can obtain the transformation equation for the two linear velocities:

$$\mathbf{v}_1 = \mathbf{R}_{12}(\boldsymbol{\omega}^{\wedge}\mathbf{p}_2 + \mathbf{v}_2). \tag{2.94}$$

We can see that there is actually a relationship between the two velocity vectors and the angular velocity. At this point, we do not speak of **different coordinate expressions of a velocity vector**, but rather **the transformation of velocity vectors in two coordinate systems**.

Continuing to take the time derivative of the above equation, we obtain:

$$\begin{aligned}\dot{\mathbf{v}}_1 &= \dot{\mathbf{R}}_{12}\left(\boldsymbol{\omega}^{\wedge}\mathbf{p}_2 + \mathbf{v}_2\right) + \mathbf{R}_{12}\left(\dot{\boldsymbol{\omega}}^{\wedge}\mathbf{p}_2 + \boldsymbol{\omega}^{\wedge}\dot{\mathbf{p}}_2 + \dot{\mathbf{v}}_2\right) \\ &= \mathbf{R}_{12}\left(\boldsymbol{\omega}^{\wedge}\boldsymbol{\omega}^{\wedge}\mathbf{p}_2 + \boldsymbol{\omega}^{\wedge}\mathbf{v}_2 + \dot{\boldsymbol{\omega}}^{\wedge}\mathbf{p}_2 + \boldsymbol{\omega}^{\wedge}\dot{\mathbf{p}}_2 + \dot{\mathbf{v}}_2\right) \\ &= \mathbf{R}_{12}\left(\dot{\mathbf{v}}_2 + 2\boldsymbol{\omega}^{\wedge}\mathbf{v}_2 + \dot{\boldsymbol{\omega}}^{\wedge}\mathbf{p}_2 + \boldsymbol{\omega}^{\wedge}\boldsymbol{\omega}^{\wedge}\mathbf{p}_2\right).\end{aligned} \tag{2.95}$$

Defining $\mathbf{a}_1 = \dot{\mathbf{v}}_1, \mathbf{a}_2 = \dot{\mathbf{v}}_2$, the equation can be written as:

$$\mathbf{a}_1 = \mathbf{R}_{12}(\underbrace{\mathbf{a}_2}_{\text{acceleration}} + \underbrace{2\boldsymbol{\omega}^{\wedge}\mathbf{v}_2}_{\text{Coriolis acceleration}} + \underbrace{\dot{\boldsymbol{\omega}}^{\wedge}\mathbf{p}_2}_{\text{angular acceleration}} + \underbrace{\boldsymbol{\omega}^{\wedge}\boldsymbol{\omega}^{\wedge}\mathbf{p}_2}_{\text{centripetal acceleration}}). \tag{2.96}$$

This equation gives the transformation between the expressions of acceleration in the two systems. It can be seen that due to the motion relationship between the two systems themselves, the transformation of acceleration is more complex than that of velocity, and it requires considering the angular velocity and angular acceleration between the two systems. Fortunately, these terms have special names to help readers remember. Moreover, in practical processing, since measurement sensors can only measure discrete values, in low-precision applications, we usually choose to ignore the last three terms and only retain the simplest transformation relationship.

In addition, in practical vehicles, we usually take system 1 and system 2 as the world coordinate system and the vehicle coordinate system, respectively. If we consider a moving point in the vehicle coordinate system, it is obvious that the linear velocity of this point in the vehicle system and in the world system are not the same vector, and it should be related to the rotation of the vehicle. These two linear velocities should satisfy the transformation relationship described in this section. However, we don't often talk about a moving point in the vehicle. More often, we discuss the **velocity of the vehicle itself**, which is the velocity of the vehicle body origin in the world system (the velocity of the vehicle origin in the vehicle system is always zero and has no practical meaning). This velocity is defined in the world system and is denoted as $\mathbf{v}_w$. If left-multiplied by $\mathbf{R}_{bw}$, this vector can also be transformed into the vehicle coordinate system, denoted as $\mathbf{v}_b$. We call $\mathbf{v}_b$ the **body velocity**, which essentially means the result of transforming the velocity vector in the world system into the vehicle coordinate system, and it can

be measured by various sensors (such as the vehicle speed sensor, rotation sensor on the wheels, etc.). Note that this transformation relationship is different from equation (2.94), one represents the relationship between different vectors, and the other represents the coordinate transformation relationship of the same vector. Please pay attention to their differences.

2.2.6 Perturbation and Jacobian Matrices

When we left-multiply or right-multiply increments on a Lie group (whether represented by rotation matrices or quaternions), there exists a corresponding increment on the Lie algebra. Due to the existence of the Baker-Campbell-Hausdorff formula, there will be a Jacobian matrix between these two increments in the sense of first-order linear approximation. Obviously, this Jacobian matrix will differ depending on the representation method or the definition of increment addition. Below, we discuss some feasible choices and definition methods, and provide some common methods for calculating Jacobians.

If we want to differentiate functions containing rotations or transformations, this derivative can be defined either at the vector level, i.e., $\mathfrak{so}(3)$ and $\mathfrak{se}(3)$, or at the perturbation level, which means left-multiplying or right-multiplying perturbations on the original $\mathbf{R}, \mathbf{T}, \mathbf{q}$, and then differentiating with respect to the perturbation. In most cases, differentiating with respect to perturbations is a more concise and clear approach. Below, we discuss the differences in formulas when perturbing rotation matrices or quaternions separately.

Example: Rotation of Vectors

Consider a vector $\mathbf{a}$, and let's rotate it. Rotation can be expressed by either a rotation matrix $\mathbf{R}$ or a quaternion $\mathbf{q}$. Thus, the rotation of $\mathbf{a}$ can be written as $\mathbf{Ra}$ in the sense of matrix multiplication, or $\mathbf{qaq}^*$ in the sense of quaternion multiplication.

Firstly, the derivation with respect to $\mathbf{a}$ itself is trivial[8], and there is no need to elaborate[9]:

$$\frac{\partial \mathbf{Ra}}{\partial \mathbf{a}} = \frac{\partial \left(\mathbf{qaq}^*\right)}{\partial \mathbf{a}} = \mathbf{R}. \tag{2.97}$$

The derivation with respect to $\mathbf{R}$ or $\mathbf{q}$ depends on the definition method. Generally, we can choose to derive with respect to the four elements of $\mathbf{q}$ itself or the Lie algebra corresponding to $\mathbf{R}$, but the Jacobian matrix corresponding to the perturbation model will be simpler. Moreover, the perturbation model is divided into left perturbation and right perturbation, and there are different definition methods for $\mathbf{R}$ and $\mathbf{q}$. Earlier in this book, the right perturbation method was used when introducing angular velocity, so here we also consider right perturbation for $\mathbf{R}$[10]. Let the perturbation quantity be ϕ,

[8] That is, it can be calculated using standard matrix derivative rules without additional conversion procedures. Readers with a certain level of matrix knowledge should be able to see this on their own.

[9] We still omit the transpose symbol at the denominator to maintain the simplicity of the formula.

[10] There is no essential difference between left and right perturbation. However, the expression of velocity or acceleration quantities in different coordinate systems may differ. According to the convention described earlier in this book, we mainly use the wb sequence to express the transformation relationship. In this case, the symbols for angular velocity, velocity, etc., are consistent with the measured values. To accommodate this expression method, we use the right perturbation model when deriving. If the reader still cannot understand the reason here, they can reconsider it later in the following text.

then[11]:

$$\begin{aligned}\frac{\partial \mathbf{Ra}}{\partial \mathbf{R}} &= \lim_{\boldsymbol{\phi}\to\mathbf{0}} \frac{\mathbf{R}\mathrm{Exp}\left(\boldsymbol{\phi}\right)\mathbf{a} - \mathbf{Ra}}{\boldsymbol{\phi}} \\ &= \lim_{\boldsymbol{\phi}\to\mathbf{0}} \frac{\mathbf{R}\left(\mathbf{I}+\boldsymbol{\phi}^{\wedge}\right)\mathbf{a} - \mathbf{Ra}}{\boldsymbol{\phi}} = -\mathbf{Ra}^{\wedge}.\end{aligned} \tag{2.98}$$

Similarly, although it is not impossible to derive with respect to the quaternion itself, it is still relatively cumbersome. Reference [4] provides a method for deriving with respect to $\mathbf{q}$ without detailed derivation. Suppose $\mathbf{q} = [w, \mathbf{v}]$, then the partial derivatives with respect to the real part and the imaginary part of $\mathbf{q}$ yield:

$$\frac{\partial \mathbf{qaq}^{*}}{\partial \mathbf{q}} = 2\left[w\mathbf{a} + \mathbf{v}^{\wedge}\mathbf{a}, \mathbf{v}^{\top}\mathbf{a}\mathbf{I}_3 + \mathbf{va}^{\top} - \mathbf{av}^{\top} - w\mathbf{a}^{\wedge}\right] \in \mathbb{R}^{3\times 4}. \tag{2.99}$$

This is evidently too complex. However, we can perturb $\mathbf{q}$. Let the perturbation quantity be $\boldsymbol{\omega} \in \mathbb{R}^3$, in order to be consistent with $\mathrm{SO}(3)$, we right-multiply $\mathbf{q}$ by $\frac{1}{2}[1, \boldsymbol{\omega}]^{\top}$, then, since the size of the perturbation on the rotation matrix is still the same, its Jacobian matrix should also be consistent:

$$\frac{\partial \mathbf{Ra}}{\partial \boldsymbol{\omega}} = -\mathbf{Ra}^{\wedge}. \tag{2.100}$$

This example tells us that in practical operations, whether rotating with $\mathbf{q}$ or $\mathbf{R}$, **we can use the same Jacobian**. If the perturbation quantity is our optimization variable, then just **update accordingly** when updating the optimization variables, instead of separately deriving Jacobian matrices for the two representation methods.

Example: Composition of Rotations

Now, let's consider the composition of rotations. We want to find the derivative of $\mathrm{Log}(\mathbf{R}_1\mathbf{R}_2)$ with respect to $\mathbf{R}_1$. We cannot directly differentiate $\mathbf{R}_1\mathbf{R}_2$ with respect to $\mathbf{R}_1$ or $\mathbf{R}_2$ because that would involve differentiating a matrix with respect to another matrix, which is not feasible without introducing tensors. Thus, we must introduce the Log operator to ensure we are dealing with vector-to-vector derivatives.

When perturbing $\mathbf{R}_1$, we can derive:

$$\begin{aligned}\frac{\partial \mathrm{Log}\left(\mathbf{R}_1\mathbf{R}_2\right)}{\partial \mathbf{R}_1} &= \lim_{\boldsymbol{\phi}\to 0} \frac{\mathrm{Log}\left(\mathbf{R}_1\mathrm{Exp}\left(\boldsymbol{\phi}\right)\mathbf{R}_2\right) - \mathrm{Log}\left(\mathbf{R}_1\mathbf{R}_2\right)}{\boldsymbol{\phi}} \\ &= \lim_{\boldsymbol{\phi}\to 0} \frac{\mathrm{Log}\left(\mathbf{R}_1\mathbf{R}_2\mathrm{Exp}\left(\mathbf{R}_2^{\top}\boldsymbol{\phi}\right)\right) - \mathrm{Log}\left(\mathbf{R}_1\mathbf{R}_2\right)}{\boldsymbol{\phi}} \\ &= \mathbf{J}_r^{-1}(\mathrm{Log}(\mathbf{R}_1\mathbf{R}_2))\mathbf{R}_2^{\top}.\end{aligned} \tag{2.101}$$

Here, the second line uses the adjoint property of $\mathrm{SO}(3)$:

$$\mathbf{R}^{\top}\mathrm{Exp}(\boldsymbol{\phi})\mathbf{R} = \mathrm{Exp}(\mathbf{R}^{\top}\boldsymbol{\phi}), \tag{2.102}$$

and the third line uses the first-order approximation of the Baker-Campbell-Hausdorff formula:

$$\mathrm{Log}\left(\mathbf{R}_1\mathbf{R}_2\mathrm{Exp}\left(\mathbf{R}_2^{\top}\boldsymbol{\phi}\right)\right) = \mathrm{Log}(\mathbf{R}_1\mathbf{R}_2) + \mathbf{J}_r^{-1}(\mathbf{R}_1\mathbf{R}_2)\mathrm{Log}(\mathrm{Exp}(\mathbf{R}_2^{\top}\boldsymbol{\phi})). \tag{2.103}$$

[11] This equation can be denoted as $\frac{\partial \mathbf{Ra}}{\partial \mathbf{R}}$ or $\frac{\partial \mathbf{Ra}}{\partial \boldsymbol{\phi}}$ on the left side.

Similarly, when perturbing $\mathbf{R}_2$, we can obtain:

$$\frac{\partial \mathrm{Log}\left(\mathbf{R}_1\mathbf{R}_2\right)}{\partial \mathbf{R}_2} = \mathbf{J}_r^{-1}(\mathrm{Log}(\mathbf{R}_1\mathbf{R}_2)). \tag{2.104}$$

Equations (2.101) and (2.104) serve as the basis for many complex function derivatives. It's essential for readers to grasp them. In practical applications, it's common to encounter compositions involving rotation matrices and other matrices or vectors. Many composite formulas can be derived using the above two equations.

2.3 Kinematics Example: Circular Motion

Below we demonstrate the differences in handling angular velocity using quaternions and rotation matrices through some practical examples.

When driving a car, if we maintain a constant speed and fix the steering wheel at a certain angle, the car should trace out a circular path. Now consider: how can we simulate this in a program?

Clearly, such a vehicle should have a fixed angular velocity. Assuming **forward-left-up** as the coordinate system, the angular velocity vector $\boldsymbol{\omega}$ of the vehicle should point towards the Z direction. The previously mentioned **constant speed** implies that in the vehicle's coordinate system, the velocity vector should be fixed pointing forward $\mathbf{v}_b = [v_x, 0, 0]^\top$. Of course, its velocity in the world frame is not solely along the X-axis because it is also turning simultaneously. Now let's implement a program to simulate this vehicle's motion. We'll use both rotation matrices and quaternions to handle the vehicle's rotation.

Listing 2.1: src/ch2/motion.cc

```
#include <gflags/gflags.h>
#include <glog/logging.h>

#include "common/eigen_types.h"
#include "common/math_utils.h"
#include "tools/ui/pangolin_window.h"

/// This program demonstrates a vehicle undergoing circular motion
/// Angular velocity and linear velocity of the vehicle can be set in flags

DEFINE_double(angular_velocity, 10.0, "Angular velocity in degrees per second");
DEFINE_double(linear_velocity, 5.0, "Linear velocity of the vehicle in m/s");
DEFINE_bool(use_quaternion, false, "Whether to use quaternion calculations");

int main(int argc, char** argv) {
  google::InitGoogleLogging(argv[0]);
  FLAGS_stderrthreshold = google::INFO;
  FLAGS_colorlogtostderr = true;
  google::ParseCommandLineFlags(&argc, &argv, true);

  /// Visualization
  sad::ui::PangolinWindow ui;
  if (ui.Init() == false) {
    return -1;
  }

  double angular_velocity_rad = FLAGS_angular_velocity * sad::math::kDEG2RAD;  // Angular velocity in radians
  SE3 pose;   // Pose represented by TWB
  Vec3d omega(0, 0, angular_velocity_rad);         // Angular velocity vector
  Vec3d v_body(FLAGS_linear_velocity, 0, 0);     // Body frame velocity
  const double dt = 0.05;     // Time for each update

  while (ui.ShouldQuit() == false) {
```

```
    // Update position
    Vec3d v_world = pose.so3() * v_body;
    pose.translation() += v_world * dt;

    // Update rotation
    if (FLAGS_use_quaternion) {
      Quatd q = pose.unit_quaternion() * Quatd(1, 0.5 * omega[0] * dt, 0.5 * omega
          [1] * dt, 0.5 * omega[2] * dt);
      q.normalize();
      pose.so3() = SO3(q);
    } else {
      pose.so3() = pose.so3() * SO3::exp(omega * dt);
    }

    LOG(INFO) << "pose: " << pose.translation().transpose();
    ui.UpdateNavState(sad::NavStated(0, pose, v_world));

    usleep(dt * 1e6);
  }

  ui.Quit();
  return 0;
}
```

Since this is the first program appearing in this book, let's provide a more complete version of it. Subsequent programs will only display the core code.

Throughout this book, we use GLog for managing logs and Gflags for managing program parameters. This program accepts user-specified angular velocity and linear velocity magnitudes, as well as the choice between using rotation matrices or quaternions for processing. Here's what we do:

1. Firstly, we convert the user-provided angular velocity to radians and linear velocity to v_{body} in the vehicle's body frame. We set the simulation time interval to 0.05 seconds.

2. During each update, we calculate the velocity in the world frame. For this, we need to know the orientation of the vehicle, so we extract the pose variable's pose and right-multiply it by the body velocity.

3. Then we update the vehicle's state. If the user specifies quaternion representation, we use Equation (2.79); otherwise, we update the self-attitude using Equation (2.56).

4. Finally, we pass the calculated pose to the UI for display and wait for a time interval.

To visualize the effect of this program in real-time, we provide a UI interface for readers. By updating the current pose and velocity in the UI interface, they will be displayed in real-time in a 3D window, as shown in Figure 2-2. After compiling the code in this chapter, readers can execute the program using:

Listing 2.2: Terminal Input:

```
./bin/motion
```

To change parameters or calculation methods, simply fill in the GFlags:

Listing 2.3: Terminal Input:

```
bin/motion --use_quaternion=true --angular_velocity=15
```

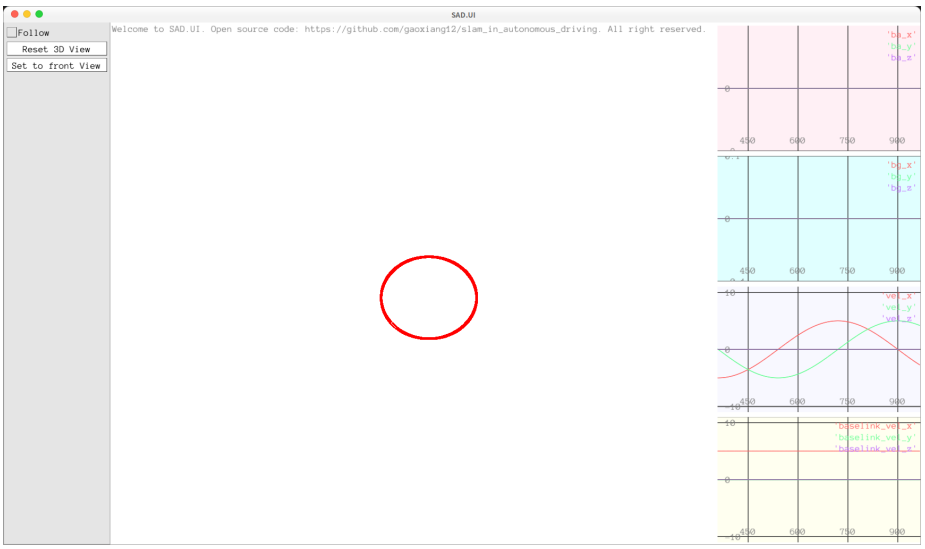


Figure 2-2: Kinematic simulation of a vehicle undergoing circular motion

Most programs in this book can be executed in a similar manner. Readers can familiarize themselves with the code style of this book using this section's program. Through this experiment, we observe the vehicle tracing out a complete circular path. Its velocity in the world frame resembles trigonometric functions, while the velocity in the body frame remains fixed along the X axis. Whether using quaternions or rotation matrices, there's no essential difference in handling kinematics. Readers can utilize this section's program to demonstrate common free fall or parabolic motion. These are left as exercises for the readers.

2.4 Filters and Optimization

Here we review the basic principles of filters and their connection to optimization methods. We'll start with state estimation.

2.4.1 State Estimation and Least Squares

SLAM problems, localization problems, or mapping problems can all be summarized as state estimation problems. A typical discrete-time state estimation problem consists of a set of motion equations and a set of observation equations:

$$\begin{cases} \mathbf{x}_k = \mathbf{f}\left(\mathbf{x}_{k-1}, \mathbf{u}_k\right) + \mathbf{w}_k, & k = 1, \ldots, N \\ \mathbf{z}_k = \mathbf{h}\left(\mathbf{x}_k\right) + \mathbf{v}_k, \end{cases} \tag{2.105}$$

where $\mathbf{f}$ is called the motion equation, $\mathbf{h}$ is called the observation equation, $\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k)$, $\mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_k)$ are Gaussian-distributed random noises. If $\mathbf{f}$ and $\mathbf{h}$ are assumed to be linear functions, we obtain the state estimation problem of a Linear Gaussian (LG) system:

$$\begin{cases} \mathbf{x}_k = \mathbf{A}_k \mathbf{x}_{k-1} + \mathbf{u}_k + \mathbf{w}_k \\ \mathbf{z}_k = \mathbf{C}_k \mathbf{x}_k + \mathbf{v}_k \end{cases}. \tag{2.106}$$

Here $\mathbf{A}_k$ and $\mathbf{C}_k$ are the system's transition matrix and observation matrix, respectively. LG systems represent the simplest form of state estimation problems, and their unbiased optimal estimation is provided by the **Kalman Filter** (KF) [2, 44].

2.4.2 Kalman Filter

The Kalman filter describes how to recursively estimate the state from one time step to the next. It consists of two steps: **prediction** and **update**. The prediction step propagates the motion equation, while the update step corrects the result from the previous step. Let $\mathbf{x}_{k-1}, \mathbf{P}_{k-1}$ denote the state estimate and its covariance matrix at time $k-1$, where $\mathbf{x}_{k-1}$ is the mean and $\mathbf{P}_{k-1}$ is the estimated covariance matrix.

1. Prediction:

$$\mathbf{x}_{k,\text{pred}} = \mathbf{A}_k \mathbf{x}_{k-1} + \mathbf{u}_k, \quad \mathbf{P}_{k,\text{pred}} = \mathbf{A}_k \mathbf{P}_{k-1} \mathbf{A}_k^\top + \mathbf{R}_k. \tag{2.107}$$

2. Update: First, calculate $\mathbf{K}$, also known as the **Kalman gain**.

$$\mathbf{K}_k = \mathbf{P}_{k,\text{pred}} \mathbf{C}_k^\top \left(\mathbf{C}_k \mathbf{P}_{k,\text{pred}} \mathbf{C}_k^\top + \mathbf{Q}_k \right)^{-1}. \tag{2.108}$$

Then compute the posterior distribution.

$$\begin{aligned} \mathbf{x}_k &= \mathbf{x}_{k,\text{pred}} + \mathbf{K}_k \left(\mathbf{z}_k - \mathbf{C}_k \mathbf{x}_{k,\text{pred}} \right), \\ \mathbf{P}_k &= \left(\mathbf{I} - \mathbf{K}_k \mathbf{C}_k \right) \mathbf{P}_{k,\text{pred}}. \end{aligned} \tag{2.109}$$

Here, the subscript "pred" denotes the predicted result. Different books may use various symbols to express the difference between it and the final estimate, such as $\hat{\mathbf{x}}, \mathbf{x}^*, \check{\mathbf{x}}$, and so on. In this book, we use subscript notation to distinguish predicted variables.

We won't delve into the derivation process of the linear Kalman filter. However, we would like to remind readers that in linear systems, various methods (Bayesian filters, Kalman filters, least squares, gain optimization, etc.) will all reach the same conclusion, so the Kalman filter can be derived from various methods, such as:

1. Derivation from the perspective of gain optimization, i.e., assuming the optimal estimate takes the form of $\mathbf{x}_{k,\text{pred}} + \mathbf{K}_k (\mathbf{z}_k - \mathbf{C}_k \mathbf{x}_{k,\text{pred}})$ and then finding the optimal $\mathbf{K}_k$. This derivation method is the simplest and is the preferred method for most similar materials.

2. Derivation from the Bayesian filter, which requires the use of linear transformations and marginalization of Gaussian distributions. This is also the preferred method in [5] and is our approach in [1].

3. Derivation from Maximum A Posteriori (MAP) estimation, which only requires basic linear algebra.

4. Derivation from batch MAP solution, using Cholesky decomposition to distinguish between forward and backward processes, and deriving the Kalman filter from the forward process. This is the preferred method in [2], which has the advantage of showing the connection between the Kalman filter and the Rauch-Tung-Stribel Smoother method (as well as the connection with batch MAP), but the downside is that the derivation process is more complex and requires a lot of space to write it.

Different from traditional Kalman filters, this book uniformly employs Lie group and Lie algebra methods to handle Kalman filters. Since we need to consider motion equations, the state variable $\mathbf{x}$ not only includes position and orientation but also includes other variables such as velocity and sensor biases. Thus, such a high-dimensional $\mathbf{x}$ lies on a high-dimensional manifold $\mathcal{M}$, known as the **Kalman Filter on Manifold** [45]. In the next chapter, we will see that this manifold-based approach is more concise than methods based on Euler angles or raw quaternion components.

2.4.3 Nonlinear Systems

In nonlinear systems, the preferred approach is to linearize $\mathbf{f}$ and $\mathbf{h}$ (**linearization**). Linearization essentially involves finding a **Taylor expansion** of a function around a fixed point and retaining only the first-order coefficients. Linearization is a widely used theory in the subsequent discussion. If linearization is performed on a general vector function $\mathbf{f}(\mathbf{x})$ at point $\mathbf{x}_0$, the result should be:

$$\mathbf{f}(\mathbf{x}_0 + \Delta\mathbf{x}) = \mathbf{f}(\mathbf{x}_0) + \mathbf{J}\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^\top\mathbf{H}\Delta\mathbf{x} + O(\Delta\mathbf{x}^2), \tag{2.110}$$

where $\mathbf{J}$ is the **Jacobian matrix**, and $\mathbf{H}$ is the **Hessian matrix**, which are the two most important matrices in linearization. If only the first-order term is retained, $\mathbf{f}(\mathbf{x})$ can be approximated as:

$$\mathbf{f}(\mathbf{x}_0 + \Delta\mathbf{x}) \approx \mathbf{f}(\mathbf{x}_0) + \mathbf{J}\Delta\mathbf{x}. \tag{2.111}$$

We can linearize the motion equations and observation equations of nonlinear systems, and then apply the conclusions of the Kalman filter to nonlinear systems, resulting in the **Extended Kalman Filter** (EKF). Without discussing the definition of $\mathbf{x}$ and the detailed forms of each matrix, a general EKF can be described as follows.

First, linearize the motion equations around the previous state to obtain:

$$\mathbf{x}_k \approx \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{F}_k\Delta\mathbf{x}_k + \mathbf{w}_k, \tag{2.112}$$

where $\mathbf{F}$ is the Jacobian matrix relative to the previous state. This matrix is mainly used to compute the predicted covariance. As for the predicted mean value, it can be obtained by substituting $\mathbf{x}_{k-1}$ into $\mathbf{f}$. This forms the prediction process of the EKF:

$$\mathbf{x}_{k,\text{pred}} = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_k), \quad \mathbf{P}_{k,\text{pred}} = \mathbf{F}_k\mathbf{P}_{k-1}\mathbf{F}_k^\top + \mathbf{R}_k. \tag{2.113}$$

It should be noted that this actually assumes that **a Gaussian distributed state variable remains Gaussian distributed after passing through a nonlinear function**. This is actually an approximation and may differ significantly from the actual situation. For the observation equation, linearize it around $\mathbf{x}_{k,\text{pred}}$ to obtain:

$$\mathbf{z}_k \approx \mathbf{h}(\mathbf{x}_{k,\text{pred}}) + \mathbf{H}_k(\mathbf{x}_k - \mathbf{x}_{k,\text{pred}}) + \mathbf{n}_k, \tag{2.114}$$

then substitute it into the Kalman filter's gain formula and update equation to obtain the update process of the EKF:

$$\mathbf{K}_k = \mathbf{P}_{k,\text{pred}}\mathbf{H}_k^\top (\mathbf{H}_k\mathbf{P}_{k,\text{pred}}\mathbf{H}_k^\top + \mathbf{Q}_k)^{-1}, \tag{2.115}$$

$$\mathbf{x}_k = \mathbf{x}_{k,\text{pred}} + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}_k\mathbf{x}_{k,\text{pred}}), \tag{2.116}$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{C}_k)\mathbf{P}_{k,\text{pred}}. \tag{2.117}$$

Comparing the KF and EKF, we find that they are basically the same in terms of formulas, except that the coefficients matrices of the EKF are not fixed and can change with the linearization point.

In this way, we have quickly reviewed the formulas of KF and EKF. However, the discussion here does not elaborate on how to calculate each matrix when $\mathbf{x}$ contains variables such as displacement, rotation, velocity, etc. In particular, if the rotation in $\mathbf{x}$ is represented in the form of $\mathbf{R}$, we actually cannot directly write expressions like $\mathbf{x}_k - \mathbf{x}_{k-1}$ or $\mathbf{x}_k - \mathbf{x}_{k,\text{pred}}$, and should use left and right perturbation models to handle these terms. After introducing Lie groups and Lie algebras, how should the EKF be modified is the problem we will discuss in Chapter **??**.

2.4.4 Graph Optimization

On the other hand, both the motion equations and observation equations can be viewed as residuals between a state variable $\mathbf{x}$ and the kinematic inputs or observations. This is a perspective of **batch least squares**:

$$\mathbf{e}_{\text{motion}} = \mathbf{x}_k - \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}) \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k), \tag{2.118}$$

$$\mathbf{e}_{\text{obs}} = \mathbf{z}_k - \mathbf{h}(\mathbf{x}_k) \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_k). \tag{2.119}$$

The optimal state estimation in the filter can be seen as a least squares problem with respect to the error terms:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_k \left(\mathbf{e}_k^\top \boldsymbol{\Omega}_k^{-1} \mathbf{e}_k\right). \tag{2.120}$$

Here, $\mathbf{e}_k$ represents the k-th error term, and $\boldsymbol{\Omega}_k$ is the covariance matrix of this error. The error $\mathbf{e}_k$ can be replaced by the motion error or observation error mentioned above, but the least squares algorithm does not deliberately distinguish between kinematic errors or observation errors. For a least squares problem composed of a general error function $\mathbf{e}_k$, we can use **iterative optimization** methods for solving. The overall idea is as follows:

1. First, start from some initial value of $\mathbf{x}$, for example $\mathbf{x}_0$.
2. Suppose the i-th iteration value is $\mathbf{x}_i$. Then, linearize the error function at $\mathbf{x}_i$:

$$\mathbf{e}_k(\mathbf{x}_i + \Delta\mathbf{x}) \approx \mathbf{e}_k(\mathbf{x}_i) + \mathbf{J}_{k,i}\Delta\mathbf{x}_i, \tag{2.121}$$

   where the linearization matrix is $\mathbf{J}_{k,i}$.

3. Use methods like Gauss-Newton or similar to solve for the increment $\Delta\mathbf{x}_i$. For example, the linear equation solved by Gauss-Newton is:

$$\sum_k (\mathbf{J}_{k,i}\boldsymbol{\Omega}_k^{-1}\mathbf{J}_{k,i}^\top)\Delta\mathbf{x}_i = -\sum_k (\mathbf{J}_{k,i}\boldsymbol{\Omega}_k^{-1}\mathbf{e}_k). \tag{2.122}$$

4. Update $\mathbf{x}_i$ to obtain the next iteration value: $\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta\mathbf{x}_i$.

5. Check if the algorithm has converged. If it has, exit; if not, proceed to the next iteration.

Optimization methods are intricately linked with filtering methods. They yield the same results in linear systems, but generally not in nonlinear systems. The main reasons are as follows:

1. Optimization methods involve an iterative process, while the EKF does not.

2. The iterative process continuously computes Jacobian matrices at new linearization points $\mathbf{x}_i$, whereas the EKF's Jacobian matrix is only computed once at the prediction position.

3. The EKF also distinguishes $\mathbf{x}_{k,\text{pred}}$, treating the prediction process and observation process separately, while optimization methods do not have $\mathbf{x}_{k,\text{pred}}$ and treat state variables uniformly.

An important question is, if we ignore the third point above and regard the Kalman filter as a nonlinear optimization problem, how many optimization variables and error functions should the Kalman filter have? The answer is: two optimization variables and three error functions. The two optimization variables refer to $\mathbf{x}_{k-1}$ and $\mathbf{x}_k$, while the three error functions are:

1. The prior error from the $k-1$ time step state $\mathbf{x}_{k-1}$, which follows its prior Gaussian distribution. If we assume $\mathbf{x}_{k-1} \sim \mathcal{N}(\bar{\mathbf{x}}_{k-1}, \mathbf{P}_{k-1})$, then a **prior error** is generated:
$$\mathbf{e}_{\text{prior}} = \mathbf{x}_{k-1} - \bar{\mathbf{x}}_{k-1} \sim \mathcal{N}(0, \mathbf{P}_{k-1}). \tag{2.123}$$

2. The **motion error** from $k-1$ to k.

3. The **observation error** at time step k.

The latter two are already listed in Equation (2.118). Thus, the Kalman filter and the optimization problem are equivalent (see Figure 2-3). However, their actual solution processes differ. The EKF does not update $\mathbf{x}_{k-1}$ but only calculates the change of $\mathbf{x}_k$, while the optimizer treats all states equally. On the other hand, the EKF also updates the covariance matrix $\mathbf{P}_k$, while a regular optimizer only computes the mean part $\mathbf{x}_k$. If we want to obtain $\mathbf{P}_k$, we also need to perform **marginalization** on the optimization problem.

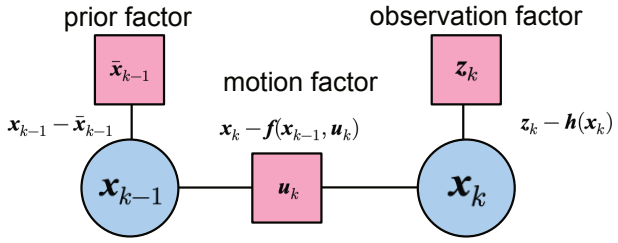


Figure 2-3: Kalman Filter and Graph Optimization Models

In the field of SLAM, optimization problems are described using graph models, corresponding to **graph optimization** or **factor graph**. Factor graph models can further introduce methods from **probabilistic graphical models** for solving. This book does not deliberately distinguish between the concepts of graph optimization and factor graph, as they usually do not have much difference in practical operation. However, comparing and discussing Kalman filters with graph optimization methods is one of the focuses of this book. We will implement classic EKF and graph optimization methods to solve problems involving inertial navigation, GPS, and laser point clouds. We will also extend EKF to Iterated Extended Kalman Filter (IEKF) to handle cases where there are nearest neighbor issues in the observation model (observation models with nearest neighbor issues may change in equation number and form during the iteration process, rather than simply linearizing at different points).

2.5 Summary

This section introduced various common coordinate systems and kinematic theories to the readers, focusing on two methods for handling kinematics: quaternions and $\mathrm{SO}(3)$. We discussed their similarities and differences. We also reviewed the basic equations of KF and EKF, and discussed some differences between them and graph optimization.

The content of this section is primarily a review. In the next section, we will introduce ESKF in detail and provide implementations as well as animated demonstrations.

Exercises

1. Calculate
$$\frac{\partial \mathbf{R}^{-1}\mathbf{p}}{\partial \mathbf{R}}$$
using both left and right perturbation models.

2. Calculate
$$\frac{\partial \mathbf{R}_1\mathbf{R}_2^{-1}}{\partial \mathbf{R}_2}$$
using both left and right perturbation models.

3. Modify the experiment in Section 2.3 to include parabolic motion with rotation. The object should rotate along the Z-axis while having an initial horizontal linear velocity and being subjected to gravity acceleration in the $-Z$ direction. Design a program and complete the animated demonstration.

Bibliography

[1] X. Gao and T. Zhang, *Introduction to visual SLAM: from theory to practice*. Springer Nature, 2021.

[2] T. D. Barfoot, *State estimation for robotics*. Cambridge University Press, 2017.

[3] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry, *An invitation to 3-d vision: from images to geometric models*, vol. 26. Springer Science & Business Media, 2012.

[4] J. Solà, "Quaternion kinematics for the error-state kalman filter," *CoRR*, vol. abs/1711.02508, 2017.

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

[6] Y. Qin, *Inertial Navigation*. Science Press, 2014.

[7] G. Yan and J. Weng, *Kalman Filter Algorithm and Combination Navigation Principle for Strapdown Inertial Navigation*. Northwestern Polytechnical University Press, 2019.

[8] S. L. Gongmin Yan and Y. Qin, *Inertial Instrument Testing and Data Analysis*. National Defense Industry Press, 2012.

[9] N. Carlevaris-Bianco, A. K. Ushani, and R. M. Eustice, "University of Michigan North Campus long-term vision and lidar dataset," *International Journal of Robotics Research*, vol. 35, no. 9, pp. 1023–1035, 2015.

[10] Z. Yan, L. Sun, T. Krajnik, and Y. Ruichek, "EU long-term dataset with multiple sensors for autonomous driving," in *Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[11] W. Wen, Y. Zhou, G. Zhang, S. Fahandezh-Saadi, X. Bai, W. Zhan, M. Tomizuka, and L.-T. Hsu, "Urbanloco: A full sensor suite dataset for mapping and localization in urban scenes," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2310–2316, IEEE, 2020.

[12] R. Qian, D. Garg, Y. Wang, Y. You, S. Belongie, B. Hariharan, M. Campbell, K. Q. Weinberger, and W.-L. Chao, "End-to-end pseudo-lidar for image-based 3d object detection," 2020.

[13] Y. Zhang, J. Wang, X. Wang, and J. M. Dolan, "Road-segmentation-based curb detection method for self-driving via a 3d-lidar sensor," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 12, pp. 3981–3991, 2018.

[14] P. Wei, X. Wang, and Y. Guo, "3d-lidar feature based localization for autonomous vehicles," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pp. 288–293, 2020.

[15] C. Hungar, S. Jürgens, D. Wilbers, and F. Köster, "Map-based localization with factor graphs for automated driving using non-semantic lidar features," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–6, 2020.

[16] L. Wang, Y. Zhang, and J. Wang, "Map-based localization method for autonomous vehicles using 3d-lidar," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 276 – 281, 2017. 20th IFAC World Congress.

[17] L. Sun, C. Peng, W. Zhan, and M. Tomizuka, "A fast integrated planning and control framework for autonomous driving via imitation learning," in *Dynamic Systems and Control Conference*, vol. 51913, p. V003T37A012, American Society of Mechanical Engineers, 2018.

[18] I. B. Viana, H. Kanchwala, K. Ahiska, and N. Aouf, "A comparison of trajectory planning and control frameworks for cooperative autonomous driving," *Journal of Dynamic Systems, Measurement, and Control*, vol. 143, no. 7, 2021.

[19] L. Zhang, F. Tafazzoli, G. Krehl, R. Xu, T. Rehfeld, M. Schier, and A. Seal, "Hierarchical road topology learning for urban map-less driving," 2021.

[20] T. Ort, K. Murthy, R. Banerjee, S. K. Gottipati, D. Bhatt, I. Gilitschenski, L. Paull, and D. Rus, "Maplite: Autonomous intersection navigation without a detailed prior map," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 556–563, 2020.

[21] Y. B. Can, A. Liniger, O. Unal, D. Paudel, and L. V. Gool, "Understanding bird's-eye view semantic hd-maps using an onboard monocular camera," 2020.

[22] A. Sun, "A perception centered self-driving system without hd maps," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 10, 2020.

[23] M. H. Ng, K. Radia, J. Chen, D. Wang, I. Gog, and J. E. Gonzalez, "Bev-seg: Bird's eye view semantic segmentation using geometry and semantic point cloud," 2020.

[24] N. Hendy, C. Sloan, F. Tian, P. Duan, N. Charchut, Y. Xie, C. Wang, and J. Philbin, "Fishing net: Future inference of semantic heatmaps in grids," 2020.

[25] J. Levinson and S. Thrun, "Robust vehicle localization in urban environments using probabilistic maps," in *2010 IEEE international conference on robotics and automation*, pp. 4372–4378, IEEE, 2010.

[26] R. W. Wolcott and R. M. Eustice, "Visual localization within lidar maps for automated urban driving," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 176–183, IEEE, 2014.

[27] R. Matthaei, G. Bagschik, and M. Maurer, "Map-relative localization in lane-level maps for adas and autonomous driving," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pp. 49–55, IEEE, 2014.

[28] F. Ghallabi, F. Nashashibi, G. El-Haj-Shhade, and M.-A. Mittet, "Lidar-based lane marking detection for vehicle positioning in an hd map," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 2209–2214, IEEE, 2018.

[29] B. Yang, M. Liang, and R. Urtasun, "Hdnet: Exploiting hd maps for 3d object detection," in *Conference on Robot Learning*, pp. 146–155, PMLR, 2018.

[30] R. Spangenberg, D. Goehring, and R. Rojas, "Pole-based localization for autonomous vehicles in urban scenarios," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2161–2166, 2016.

[31] J. Levinson, M. Montemerlo, and S. Thrun, "Map-based precision vehicle localization in urban environments," in *Robotics: science and systems*, vol. 4, p. 1, Atlanta, GA, USA, 2007.

[32] H. G. Seif and X. Hu, "Autonomous driving in the icity—hd maps as a key challenge of the automotive industry," *Engineering*, vol. 2, no. 2, pp. 159–162, 2016.

[33] Y. Zhou, Y. Takeda, M. Tomizuka, and W. Zhan, "Automatic construction of lane-level hd maps for urban scenes," *arXiv preprint arXiv:2107.10972*, 2021.

[34] M. Dupuis, M. Strobl, and H. Grezlikowski, "Opendrive 2010 and beyond–status and future of the de facto standard for the description of road networks," in *Proc. of the Driving Simulation Conference Europe*, pp. 231–242, 2010.

[35] F. Poggenhans, J.-H. Pauls, J. Janosovits, S. Orf, M. Naumann, F. Kuhnt, and M. Mayr, "Lanelet2: A high-definition map framework for the future of automated driving," in *2018 21st international conference on intelligent transportation systems (ITSC)*, pp. 1672–1679, IEEE, 2018.

[36] F. Ghallabi, G. El-Haj-Shhade, M.-A. Mittet, and F. Nashashibi, "Lidar-based road signs detection for vehicle localization in an hd map," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1484–1490, IEEE, 2019.

[37] W.-C. Ma, I. Tartavull, I. A. Bârsan, S. Wang, M. Bai, G. Mattyus, N. Homayounfar, S. K. Lakshmikanth, A. Pokrovsky, and R. Urtasun, "Exploiting sparse semantic hd maps for self-driving vehicle localization," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5304–5311, IEEE, 2019.

[38] M. Elhousni, Y. Lyu, Z. Zhang, and X. Huang, "Automatic building and labeling of hd maps with deep learning," 2020.

[39] B. Liao, S. Chen, X. Wang, T. Cheng, Q. Zhang, W. Liu, and C. Huang, "Maptr: Structured modeling and learning for online vectorized hd map construction," *arXiv preprint arXiv:2208.14437*, 2022.

[40] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[41] T. Barfoot, J. R. Forbes, and P. T. Furgale, "Pose estimation using linearized rotations and quaternion algebra," *Acta Astronautica*, vol. 68, no. 1-2, pp. 101–112, 2011.

[42] R. Gilmore, "Baker-campbell-hausdorff formulas," *Journal of Mathematical Physics*, vol. 15, no. 12, pp. 2090–2092, 1974.

[43] H. E. Rauch, F. Tung, and C. T. Striebel, "Maximum likelihood estimates of linear dynamic systems," *AIAA journal*, vol. 3, no. 8, pp. 1445–1450, 1965.

[44] P. Zarchan, *Progress in astronautics and aeronautics: fundamentals of Kalman filtering: a practical approach*, vol. 208. Aiaa, 2005.

[45] D. He, W. Xu, and F. Zhang, "Kalman filters on differentiable manifolds," *arXiv preprint arXiv:2102.03804*, 2021.