Creating Autonomous Vehicle Systems

Creating Autonomous Vehicle Systems
Shaoshan Liu, Liyun Li, Jie Tang, Shuang Wu, and Jean-Luc Gaudiot
www.morganclaypool.com

ISBN: 9781681730073 print
ISBN: 9781681730080 ebook
ISBN: 9781681731674 epub

DOI: 10.2200/S00787ED1V01Y201707CSL009

A Publication in the Morgan & Claypool Publishers series
SYNTHESIS LECTURES ON COMPUTER SCIENCE, #9

Series ISSN: 1932-1228 Print 1932-1686 Electronic

Creating Autonomous Vehicle Systems

Shaoshan Liu

PerceptIn

Liyun Li

Baidu, U.S.

Jie Tang

South China University of Technology

Shuang Wu

YiTu

Jean-Luc Gaudiot

University of California, Irvine

MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

This book is the first technical overview of autonomous vehicles written for a general computing and engineering audience. The authors share their practical experiences of creating autonomous vehicle systems. These systems are complex, consisting of three major subsystems: (1) algorithms for localization, perception, and planning and control; (2) client systems, such as the robotics operating system and hardware platform; and (3) the cloud platform, which includes data storage, simulation, high-definition (HD) mapping, and deep learning model training. The algorithm subsystem extracts meaningful information from sensor raw data to understand its environment and make decisions about its actions. The client subsystem integrates these algorithms to meet real-time and reliability requirements. The cloud platform provides offline computing and storage capabilities for autonomous vehicles. Using the cloud platform, we are able to test new algorithms and update the HD map—plus, train better recognition, tracking, and decision models.

This book consists of nine chapters. Chapter 1 provides an overview of autonomous vehicle systems; Chapter 2 focuses on localization technologies; Chapter 3 discusses traditional techniques used for perception; Chapter 4 discusses deep learning based techniques for perception; Chapter 5 introduces the planning and control sub-system, especially prediction and routing technologies; Chapter 6 focuses on motion planning and feedback control of the planning and control subsystem; Chapter 7 introduces reinforcement learning-based planning and control; Chapter 8 delves into the details of client systems design; and Chapter 9 provides the details of cloud platforms for autonomous driving.

This book should be useful to students, researchers, and practitioners alike. Whether you are an undergraduate or a graduate student interested in autonomous driving, you will find herein a comprehensive overview of the whole autonomous vehicle technology stack. If you are an autonomous driving practitioner, the many practical techniques introduced in this book will be of interest to you. Researchers will also find plenty of references for an effective, deeper exploration of the various technologies.

KEYWORDS

autonomous driving, driverless cars, perception, vehicle localization, planning and control, autonomous driving hardware platform, autonomous driving cloud infrastructures

Contents

Preface

Autonomous vehicles, be they on land, on water, or in the air, are upon us and are finding a myriad of new applications, from driverless taxi services to automatic airborne surveillance of sensitive remote areas. Continued technological advances in the past few decades have made these innovations possible, but the design problems which must be surmounted in order to provide useful, efficient, and, supremely importantly, safe operations of these independent units are equally numerous and daunting.

It is thus the purpose of this book to provide an overview of these problems and lead the reader through some common design solutions. High technological capabilities, complete integration of hardware and software, and deep synergy with resident platforms (such as cloud servers) are a must for an eventual successful deployment. The focus is on land vehicles, and more specifically cars in urban or country road environments, as well as off-road operations. The aim of this book is to address an audience of engineers, be they from the academic or the industrial side, with a survey of the problems, solutions, and future research issues they will encounter in the development of autonomous vehicles, from sensing, perception to action, and including support from cloud-based servers. A copious amount of bibliographic references completes the picture and will help the reader navigate through a jungle of past work.

STRUCTURE OF THE BOOK

A brief history of information technology and an overview of the algorithms behind autonomous driving systems, of the architecture of the systems, and of the support infrastructure needed is provided in Chapter 1. Localization, being one of the most important tasks in autonomous driving, is covered in Chapter 2 where the most common approaches are introduced. The principles, advantages, and drawbacks of GNSS, INS, LiDAR, and wheel odometry are described in detail and the integration of various versions of these strategies are discussed. As for detection, i.e., "understanding" the environment based on sensory data, it is described in Chapter 3, with an exploration of the various algorithms in use, including scene understanding, image flow, tracking, etc. The large datasets, highly complex computations required by image classification, object detection, semantic segmentation, etc. are best handled by the deep learning approaches to perception advocated for in Chapter 4, where applications to detection, semantic segmentation, and image flow are described in detail. Once the environment is understood by the autonomous vehicle, it must somehow predict future events (e.g., the motion of another vehicle in its vicinity) and plan its own route. This is the purpose of Chapter 5. Next (Chapter 6), comes an even more detailed level of decision making,

planning, and control. Feedback between modules with possibly orthogonal decisions as well as conflict resolution (e.g., one module could recommend a lane change, but another one has detected an obstacle in the lane in question) are covered with an emphasis on describing algorithms for behavioral decision making (e.g., Markov decision processes, scenario-based divide and conquer), and for motion planning. This is what leads us into Chapter 7 for a demonstration of the need to supplement the design with Reinforcement Learning-based Planning and Control for a complete integration of situational scenarios in the development of an autonomous system. Underneath it all, the on-board computing platform is the topic of Chapter 8. It includes an introductory description of the Robot Operating System, followed by an actual summary of the real hardware employed. The need for heterogeneous computing is introduced with a strong emphasis on meeting real-time computing requirements as well as on-board considerations (power consumption and heat dissipation). This means that a variety of processing units (general-purpose CPU, GPUs, FPGAs, etc.) must be used. Finally, Chapter 9 covers the infrastructure for the cloud platform used to "tie it all together" (i.e., provide services for distributed simulation tests for new algorithm deployment, offline deep learning model training, and High-Definition (HD) map generation).

CHAPTER 1

Introduction to Autonomous Driving

We are at the dawn of the future of autonomous driving. To understand what the future may be, we usually consult history, so let us start with that.

The beginning of information technology truly started in the 1960s, when Fairchild Semiconductors and Intel laid down the foundation layer of information technology by producing microprocessors, and as a side product, created Silicon Valley. Although microprocessor technologies greatly improved our productivity, the general public had limited access to these technologies. Then in the 1980s, Microsoft and Apple laid down the second layer of information technology by introducing Graphics User Interface, and realized the vision of "a PC/Mac in every home." Once everyone had access to computing power, in the 2000s, internet companies, represented by Google, laid down the third layer of information technology by connecting people and information. Through Google, for instance, the information providers and the information consumers can be indirectly connected. Then in the 2010s, the social network companies, such as Facebook and LinkedIn, laid down the fourth layer of information technology by effectively moving the human society to internet, and allowed people to directly connect to one another. After the population of the internet-based human society reached a significant scale, around 2015, the emergence of Uber and Airbnb laid down the fifth layer of information technology by providing services upon the internet-based human society, and forming an internet-based commerce society. Although Uber and Airbnb provided the means for us to efficiently access service providers through the internet, the services are still provided by humans.

1.1 AUTONOMOUS DRIVING TECHNOLOGIES OVERVIEW

As shown in Figure 1.1, autonomous driving is not one single technology, but rather a highly complex system that consists of many sub-systems. Let us break it into three major components: algorithms, including sensing, perception, and decision (which requires reasoning for complex cases); client systems, including the operating system and the hardware platform; and the cloud platform, including high-definition (HD) map, deep learning model training, simulation, and data storage.

The algorithm subsystem extracts meaningful information from sensor raw data to understand its environment and to make decisions about its future actions. The client systems integrate these algorithms together to meet real-time and reliability requirements. For example, if the camera generates data at 60 Hz, the client systems need to make sure that the longest stage of the processing pipeline takes less than 16 ms to complete. The cloud platform provides offline computing

and storage capabilities for autonomous cars. With the cloud platform, we are able to test new algorithms, update HD map, and train better recognition, tracking, and decision models.

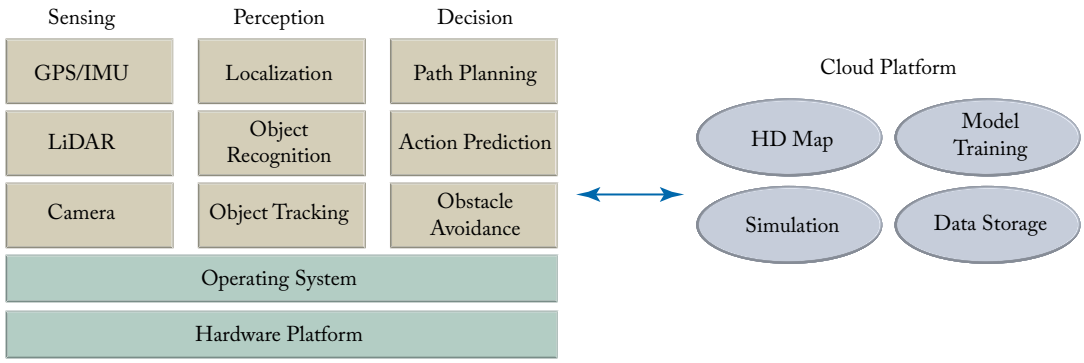

Figure 1.1: Autonomous driving system architecture overview.

1.2 AUTONOMOUS DRIVING ALGORITHMS

The algorithms component consists of sensing, that is extracting meaningful information from sensor raw data; perception, which is to localize the vehicle and to understand the current environment; and decision, in other words taking actions so as to reliably and safely reach target destinations.

1.2.1 SENSING

Normally, an autonomous car consists of several major sensors. Indeed, since each type of sensor presents advantages and drawbacks, in autonomous vehicles, the data from multiple sensors must be combined for increased reliability and safety. They can include the following.

- **GPS/IMU:** The GPS/IMU system helps the autonomous vehicle localize itself by reporting both inertial updates and a global position estimate at a high rate, e.g., 200 Hz. GPS is a fairly accurate localization sensor, but its update rate is slow, at about only 10 Hz, and thus not capable of providing real-time updates. However, IMU errors accumulate over time, leading to a corresponding degradation in the position estimates. Nonetheless, an IMU can provide updates more frequently, at or higher than 200 Hz. This should satisfy the real-time requirement. By combining both GPS and IMU, we can provide accurate and real-time updates for vehicle localization.

- **LiDAR:** LiDAR is used for mapping, localization, and obstacle avoidance. It works by bouncing a beam off surfaces and measures the reflection time to determine distance. Due to its high accuracy, LiDAR can be used to produce HD maps, to localize

a moving vehicle against HD maps, to detect obstacle ahead, etc. Normally, a LiDAR unit, such as Velodyne 64-beam laser, rotates at 10 Hz and takes about 1.3 million readings per second.

- **Cameras:** Cameras are mostly used for object recognition and object tracking tasks such as lane detection, traffic light detection, and pedestrian detection, etc. To enhance autonomous vehicle safety, existing implementations usually mount eight or more 1080p cameras around the car, such that we can use cameras to detect, recognize, and track objects in front of, behind, and on both sides of the vehicle. These cameras usually run at 60 Hz, and, when combined, would generate around 1.8 GB of raw data per second.

- **Radar and Sonar:** The radar and sonar system is mostly used for the last line of defense in obstacle avoidance. The data generated by radar and sonar shows the distance as well as velocity from the nearest object in front of the vehicle's path. Once we detect that an object is not far ahead, there may be a danger of a collision, then the autonomous vehicle should apply the brakes or turn to avoid the obstacle. Therefore, the data generated by radar and sonar does not require much processing and usually is fed directly to the control processor, and thus not through the main computation pipeline, to implement such "urgent" functions as swerving, applying the brakes, or pre-tensioning the seatbelts.

1.2.2 PERCEPTION

The sensor data is then fed into the perception stage to provide an understanding of the vehicle's environment. The three main tasks in autonomous driving perception are localization, object detection, and object tracking.

GPS/IMU can be used for localization, and, as mentioned above, GPS provides fairly accurate localization results but with a comparatively low update rate, while an IMU provides very fast updates at a cost of less accurate results. We can thus use Kalman Filter techniques to combine the advantages of the two and provide accurate and real-time position updates. As shown in Figure 1.2, it works as follows: the IMU updates the vehicle's position every 5 ms, but the error accumulates with time. Fortunately, every 100 ms, a GPS update is received, which helps correct the IMU error. By running this propagation and update model, the GPS/IMU combination can generate fast and accurate localization results. Nonetheless, we cannot solely rely on this combination for localization for three reasons: (1) the accuracy is only about one meter; (2) the GPS signal has multipath problems, meaning that the signal may bounce off buildings, introducing more noise; and (3) GPS requires an unobstructed view of the sky and would thus not work in environments such as tunnels.

Localization

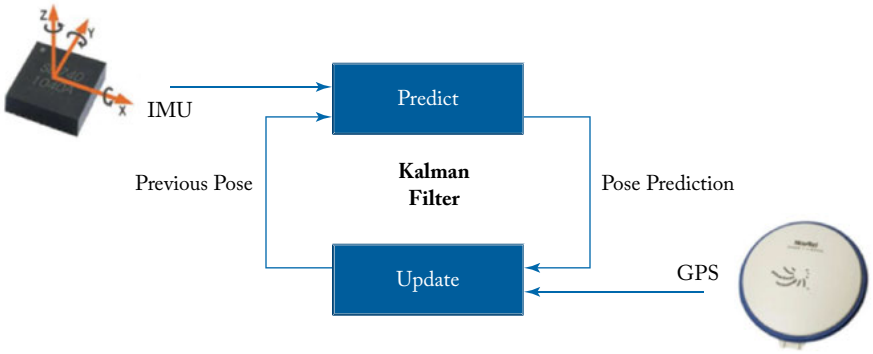

Figure 1.2: GPS/IMU localization.

Cameras can be used for localization too. Vision-based localization can be implemented as the following simplified pipeline: (1) by triangulating stereo image pairs, we first obtain a disparity map which can be used to derive depth information for each point; (2) by matching salient features between successive stereo image frames, we can establish correlations between feature points in different frames. We could then estimate the motion between the past two frames; and also, (3) by comparing the salient features against those in the known map, we could also derive the current position of the vehicle. However, such a vision-based localization approach is very sensitive to lighting conditions and, thus, this approach alone would not be reliable.

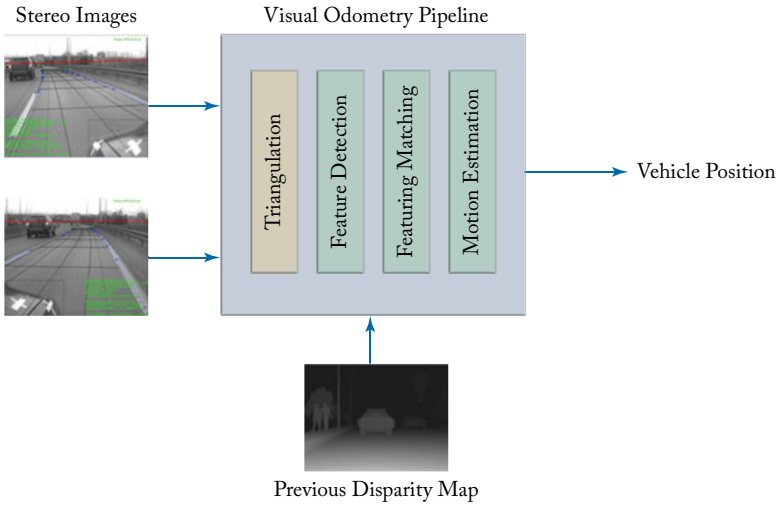

Figure 1.3: Stereo visual odometry.

This is why LiDAR approaches typically have recourse to particle filter techniques. The point clouds generated by LiDAR provide a "shape description" of the environment, but it is hard to differentiate individual points. By using a particle filter, the system compares a specific observed shape against the known map to reduce uncertainty. To localize a moving vehicle relative to these maps, we could apply a particle filter method to correlate the LIDAR measurements with the map. The particle filter method has been demonstrated to achieve real-time localization with 10-cm accuracy and to be effective in urban environments. However, LiDAR has its own problem: when there are many suspended particles in the air, such as rain drops and dust, the measurements may be extremely noisy. Therefore, as shown in Figure 1.4, to achieve reliable and accurate localization, we need a sensor-fusion process to combine the advantages of all sensors.

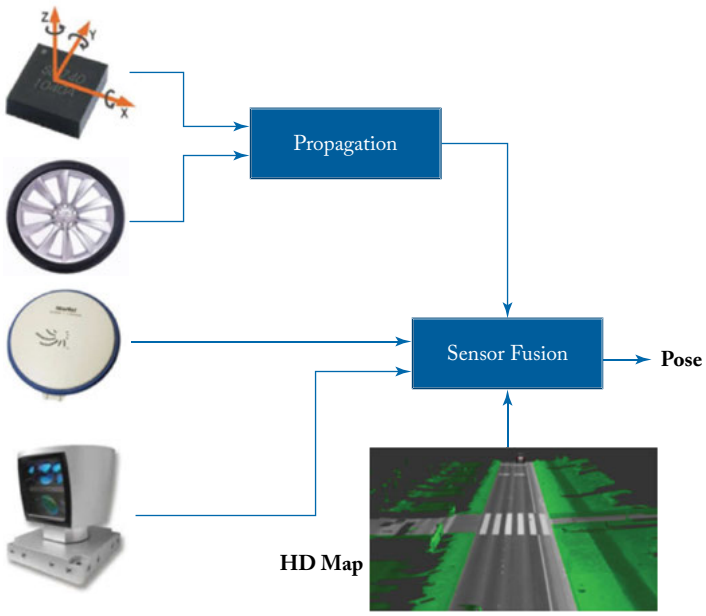

Figure 1.4: Sensor-fusion localization pipeline.

1.2.3 OBJECT RECOGNITION AND TRACKING

Originally LiDAR was used mostly to perform object detection and tracking tasks in Autonomous Vehicles, since LiDAR provides very accurate depth information. In recent years, however, we have seen the rapid development of Deep Learning technology, which achieves significant object detection and tracking accuracy. Convolution Neural Network (CNN) is a type of Deep Neural Network that is widely used in object recognition tasks. A general CNN evaluation pipeline usually consists of the following layers. (1) The Convolution Layer contains different filters to extract

different features from the input image. Each filter contains a set of "learnable" parameters that will be derived after the training stage. (2) The Activation Layer decides whether to activate the target neuron or not. (3) The Pooling Layer reduces the spatial size of the representation to reduce the number of parameters and consequently the computation in the network. (4) The Fully Connected Layer where neurons have full connections to all activations in the previous layer.

Object tracking refers to the automatic estimation of the trajectory of an object as it moves. After the object to track is identified using object recognition techniques, the goal of object tracking is to automatically track the trajectory of the object subsequently. This technology can be used to track nearby moving vehicles as well as people crossing the road to ensure that the current vehicle does not collide with these moving objects. In recent years, deep learning techniques have demonstrated advantages in object tracking compared to conventional computer vision techniques. Specifically, by using auxiliary natural images, a stacked Auto-Encoder can be trained offline to learn generic image features that are more robust against variations in viewpoints and vehicle positions. Then, the offline trained model can be applied for online tracking.

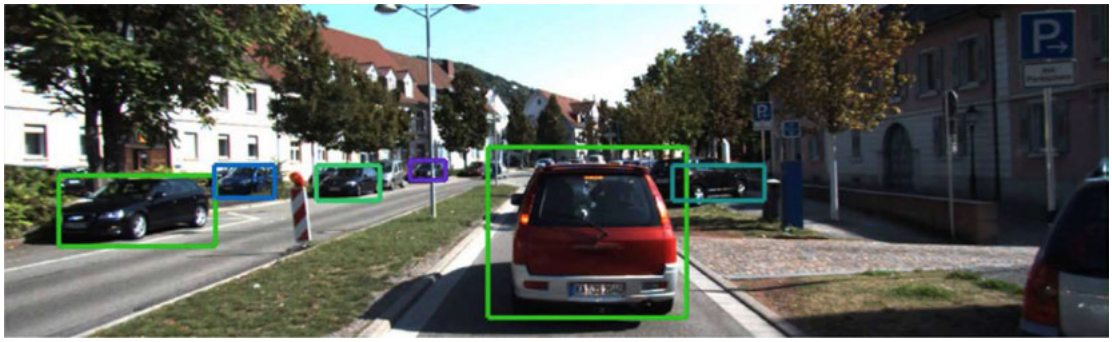

Figure 1.5: Object recognition and tracking [34], used with permission.

1.2.4 ACTION

Based on the understanding of the vehicle's environment, the decision stage can generate a safe and efficient action plan in real time.

Action Prediction

One of the main challenges for human drivers when navigating through traffic is to cope with the possible actions of other drivers which directly influence their own driving strategy. This is especially true when there are multiple lanes on the road or when the vehicle is at a traffic change point. To make sure that the vehicle travels safely in these environments, the decision unit generates predictions of nearby vehicles, and decides on an action plan based on these predictions. To predict

actions of other vehicles, one can generate a stochastic model of the reachable position sets of the other traffic participants, and associate these reachable sets with probability distributions.

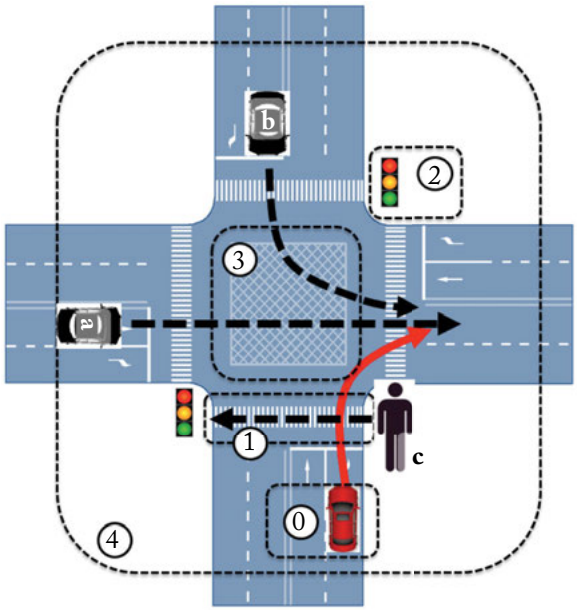

Figure 1.6: Action prediction.

Path Planning

Planning the path of an autonomous, agile vehicle in a dynamic environment is a very complex problem, especially when the vehicle is required to use its full maneuvering capabilities. A brute force approach would be to search all possible paths and utilize a cost function to identify the best path. However, the brute force approach would require enormous computation resources and may be unable to deliver navigation plans in real-time. In order to circumvent the computational complexity of deterministic, complete algorithms, probabilistic planners have been utilized to provide effective real-time path planning.

Obstacle Avoidance

As safety is the paramount concern in autonomous driving, we usually employ at least two levels of obstacle avoidance mechanisms to ensure that the vehicle would not collide with obstacles. The first level is proactive, and is based on traffic predictions. At runtime, the traffic prediction mechanism generates measures like time to collision or predicted minimum distance, and based on this

information, the obstacle avoidance mechanism is triggered to perform local path re-planning. If the proactive mechanism fails, the second level, the reactive mechanism, using radar data, would take over. Once radar detects an obstacle ahead of the path, it would override the current control to avoid the obstacles.

1.3 AUTONOMOUS DRIVING CLIENT SYSTEM

The client systems integrate the above-mentioned algorithms together to meet real-time and reliability requirements. Some of the challenges are as follows: the system needs to ensure that the processing pipeline is fast enough to process the enormous amount of sensor data generated; if a part of the system fails, it must be sufficiently robust to recover from the failure; and, in addition, it needs to perform all the computation under strict energy and resource constraints.

1.3.1 ROBOT OPERATING SYSTEM (ROS)

ROS is a powerful distributed computing framework tailored for robotics applications, and it has been widely used. As shown in Figure 1.7, each robotic task (such as localization), is hosted in a ROS node. ROS nodes can communicate with each other through topics and services. It is a great operating system for autonomous driving except that it suffers from several problems: (1) reliability: ROS has a single master and no monitor to recover failed nodes; (2) performance: when sending out broadcast messages, it duplicates the message multiple times, leading to performance degradation; and (3) security: it has no authentication and encryption mechanisms. Although ROS 2.0 promised to fix these problems, ROS 2.0 itself has not been extensively tested and many features are not yet available. Therefore, in order to use ROS in autonomous driving, we need to solve these problems first.

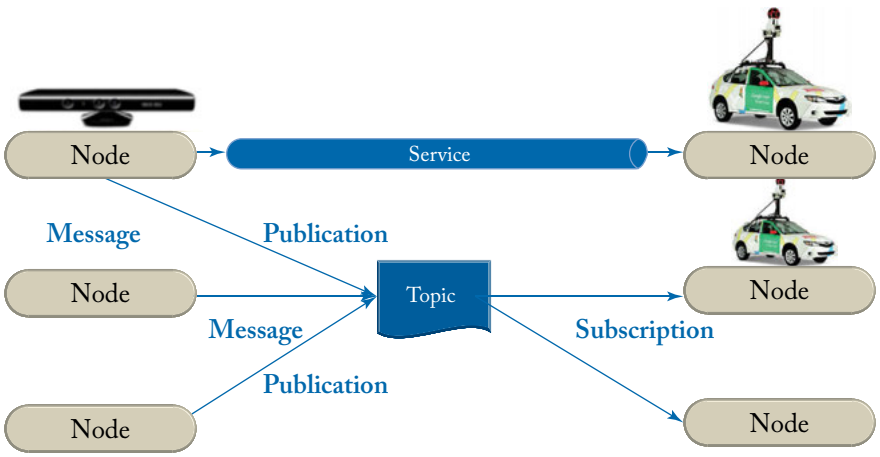

Figure 1.7: Robot operating system (ROS).

Reliability

The current ROS implementation has only one master node, when the master node crashes, the whole system would crash. This does not meet the safety requirement for autonomous driving. To fix this problem, we implemented a ZooKeeper-like mechanism in ROS. As shown in Figure 1.8, in this design, we have a main master node and a backup master node. In the case of main node failure, the backup node would take over, making sure that the system still runs without hiccups. In addition, this ZooKeeper mechanism monitors and restarts any failed nodes, making sure the whole ROS system is reliable.

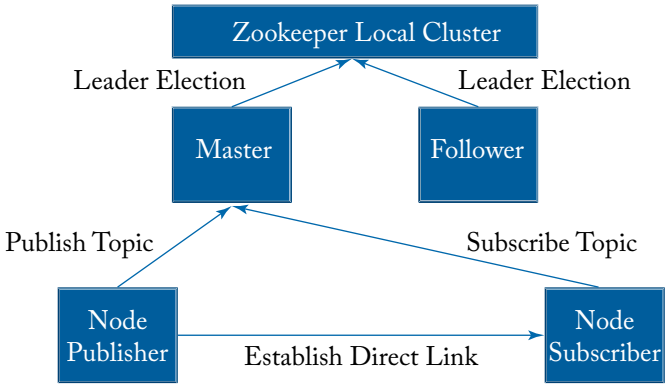

Figure 1.8: Zookeeper for ROS.

Performance

Performance is another problem with the current ROS implementation. The ROS nodes communicate often, and it is imperative to ensure the communication between nodes is efficient. First, it goes through the loop-back mechanism when local nodes communicate with each other. Each time it goes through the loop-back pipeline, a 20 ms overhead is introduced. To eliminate this overhead, for local node communication overhead, we used shared memory mechanism such that the message does not have to go through the TCPIP stack to get to the destination node. Second, when a ROS node broadcasts a message, the message gets copied multiple times, consuming significant bandwidth in the system. As shown in Figure 1.9, by switching to multicast mechanism, we greatly improved the throughput of the system.

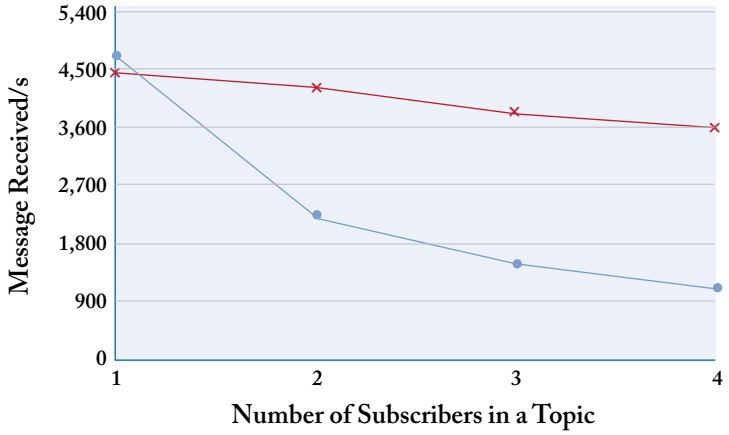

Figure 1.9: Multicast vs. broadcast in ROS.

Security

As we know, security is the most critical concern for ROS. Now imagine two scenarios: in the first scenario, a ROS node is kidnapped and continuously allocates memory until the system runs out of memory and starts killing other ROS nodes. In this scenario, the hacker successfully crashes the system. In the second scenario, since by default ROS messages are not encrypted, a hacker can easily eavesdrop the message between nodes and applies man-in-the-middle attacks. To fix the first problem, we can use Linux Container (LXC) to restrict the amount of resources used by each node, and also to provide a sandbox mechanism to protect the node from each other, therefore effectively preventing resource leaking. To fix the second problem, we can encrypt messages in communication, preventing messages being eavesdropped.

1.3.2 HARDWARE PLATFORM

To understand the challenges in designing hardware platform for autonomous driving, let us examine the computing platform implementation from a leading autonomous driving company. It consists of two compute boxes, each equipped with an Intel Xeon E5 processor and four to eight Nvidia K80 GPU accelerators. The second compute box performs exactly the same tasks and is used for reliability: in case the first box fails, the second box can immediately take over. In the worst case, when both boxes run at their peak, this would mean over 5000 W of power consumption which would consequently generate enormous amount of heat. Also, each box costs $20,000–$30,000, making the whole solution unaffordable to average consumers.

The power, heat dissipation, and cost requirements of this design prevents autonomous driving to reach the general public. To explore the edges of the envelope and understand how well an autonomous driving system could perform on an ARM mobile SoC, we implemented a simplified, vision-based autonomous driving system on a ARM-based mobile SoC with peak power consumption of 15 W. As it turns out, the performance was close to our requirements: the localization pipeline was capable of processing 25 images per second, almost keeping up with an image generation rate of 30 images per second. The deep learning pipeline was capable of performing 2–3 object recognition tasks per second. The planning and control pipeline could plan a path within 6 ms. With this system, we were able to drive the vehicle at around 5 mph without any loss of localization.

1.4 AUTONOMOUS DRIVING CLOUD PLATFORM

Autonomous vehicles are mobile systems, and therefore they need a cloud platform to provides supports. The two main functions provided by the cloud include distributed computing, and distributed storage. It has several applications, including simulation, which is used to verify new algorithms; HD map production; and deep learning model training. To build such a platform, we used Spark for distributed computing, OpenCL for heterogeneous computing, and Alluxio for in-memory storage. We have managed to deliver a reliable, low-latency, and high-throughput autonomous driving cloud by integrating Spark, OpenCL, and Alluxio together.

1.4.1 SIMULATION

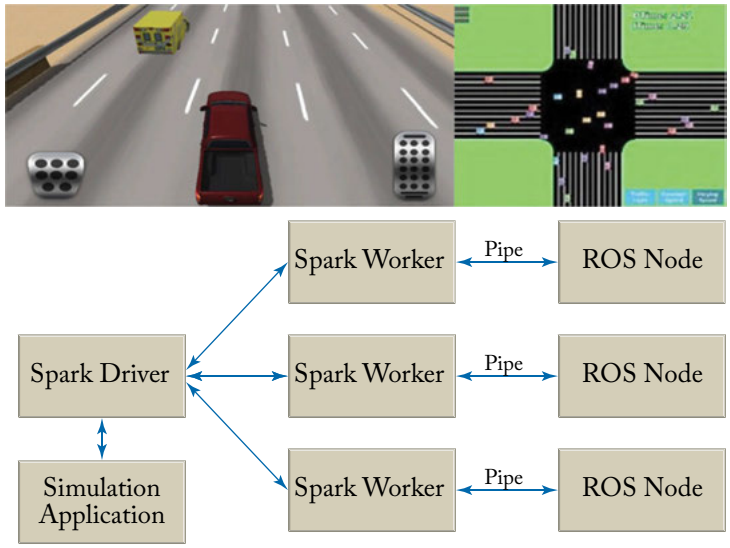

Figure 1.10: Spark and ROS-based simulation platform.

The first application of such a system is simulation. When we develop a new algorithm, and we need to test it thoroughly before we can deploy it on cars. If we were to test it on real cars, the cost would be enormous and the turn-around time would be too long. Therefore, we usually test it on simulators, such as replaying data through ROS nodes. However, if we were to test the new algorithm on a single machine, either it is going to take too long, or we do not have enough test coverage. As shown in Figure 1.10, to solve this problem, we have developed a distributed simulation platform. Such that we use Spark to manage distributed computing nodes, and on each node, we run a ROS replay instance. In an autonomous driving object recognition test set that we used, it took 3 h to run on a single server; by using the distributed system we developed, the test finished within 25 min when we scaled to 8 machines.

1.4.2 HD MAP PRODUCTION

As shown in Figure 1.11, HD map production is a complex process that involves many stages, including raw data processing, point cloud production, point cloud alignment, 2D reflectance map generation, HD map labeling, as well as the final map generation. Using Spark, we connected all these stages together in one Spark job. More importantly, Spark provides an in-memory computing mechanism, such that we do not have to store the intermediate data in hard disk, thus greatly reducing the performance of the map production process.

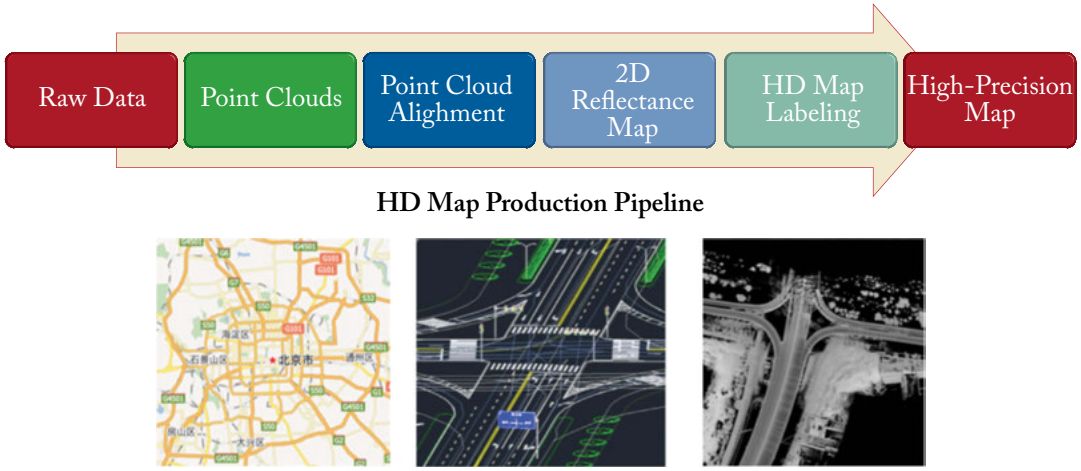

Figure 1.11: Cloud-based HD Map production.

1.4.3 DEEP LEARNING MODEL TRAINING

As we use different deep learning models in autonomous driving, it is imperative to provide updates to continuously improve the effectiveness and efficiency of these models. However, since the amount of raw data generated is enormous, we would not be able to achieve fast model training using single servers. To approach this problem, we have developed a highly scalable distributed deep learning system using Spark and Paddle (a deep learning platform recently open-sourced by Baidu). As shown in Figure 1.12, in the Spark driver we manage a Spark context and a Paddle context, and in each node, the Spark executor hosts a Paddler trainer instance. On top of that, we use Alluxio as a parameter server for this system. Using this system, we have achieved linear performance scaling as we added more resources, proving that the system is highly scalable.

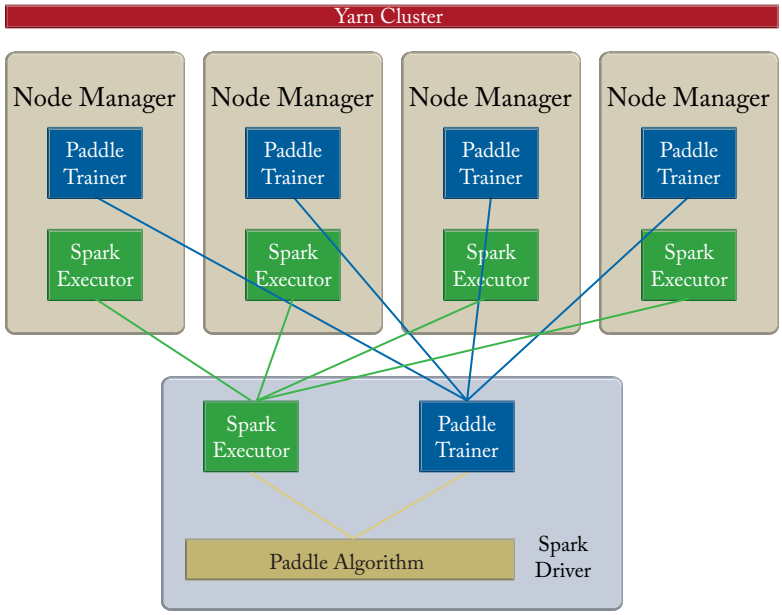

Figure 1.12: Distributed deep learning model training system.

1.5 IT IS JUST THE BEGINNING

Again, autonomous driving, or artificial intelligence in general, is not one single technology, it is an integration of many technologies. It demands innovations in algorithms, system integrations, cloud platforms. It is just the beginning and there are tons of opportunities in this era, I foresee that in 2020, we will officially start this AI-era and start seeing more and more AI-based products in the market.

The rest of the book is organized as follows. Localization, being one of the most important tasks in autonomous driving is covered in Chapter 2. As for detection, i.e., "understanding" the environment based on sensory data, this is described in Chapter 3, with an exploration of the various algorithms in use, including scene understanding, image flow, tracking, etc. The large datasets, highly complex computations required by image classification, object detection, semantic segmentation, etc., are best handled by the Deep Learning approaches to perception advocated in Chapter 4. Once the environment is understood by the autonomous vehicle, it must somehow predict future events (e.g., the motion of another vehicle in its vicinity) and plan its own route. This is the purpose of Chapter 5. Chapter 6 is comprised of an even more detailed level of decision making, planning, and control. Chapter 7 is a demonstration of the design with Reinforcement Learning-based Planning and Control for a complete integration of situational scenarios in the development of an autonomous system. Underneath it all, the on-board computing platform is the topic of Chapter 8. Finally, Chapter 9 covers the infrastructure for the cloud platform used to "tie it all together"

CHAPTER 2

Autonomous Vehicle Localization

Abstract

For autonomous vehicles, one of the most critical tasks is localization, i.e., the accurate and real-time determination of the unit's position. In this chapter we first study different localization techniques, including GNSS, LiDAR and High-Definition Maps, Visual Odometry, and other Dead Reckoning sensors. We also look into several real-world examples of applying sensor fusion techniques to combine multiple sensors to provide more accurate localization.

2.1 LOCALIZATION WITH GNSS

When human drive cars, we usually rely on the global navigation satellite system (GNSS) for localization. When it comes to autonomous vehicle localization, we also start with GNSS. In this section, we delve into the details of GNSS technologies and understand the pros and cons of GNSS when applying to autonomous driving.

2.1.1 GNSS OVERVIEW

The GNSS consist of several satellite systems: GPS, GLONASS, Galileo, and BeiDou. Here we use GPS as an example to provide an overview of GNSS. GPS provides coded satellite signals that can be processed in a GPS receiver, allowing the receiver to estimate position, velocity and time [1]. For this to work, GPS requires four satellite signals to compute positions in three dimensions and the time offset in the receiver clock. The deployment of these GPS satellites are dispersed in six orbital planes on almost circular orbits with an altitude of about 20,200 km above the surface of the Earth, inclined by 55° with respect to the equator and with orbital periods of approximately 11 hr 58 min.

The generated signals on board the satellites are derived from generation of a fundamental frequency f_o=10.23 MHz [1]. The signal is time stamped with atomic clocks with inaccuracy in the range of only 10–13 s over a day. Two carrier signals in the L-band, denoted L1 and L2, are generated by integer multiplications of f_o. The carriers L1 and L2 are bi-phase modulated by codes to provide satellite clock readings to the receiver and transmit information such as the orbital parameters. The codes consist of a sequence with the states +1 or −1, corresponding to the binary values 0 or 1. The bi-phase modulation is performed by a 180° shift in the carrier phase whenever

a change in the code state occurs. The satellite signals contain information on the satellite orbits, orbit perturbations, GPS time, satellite clock, ionospheric parameters, and system status messages, etc. The navigation message consists of 25 frames with each frame containing 1,500 bit and each frame is subdivided into 5 sub-frames with 300 bit.

The next critical piece of the GNSS system is the definition of reference coordinate system, which is crucial for the description of satellite motion, the modeling of observable and the interpretation of results. For GNSS to work, two reference systems are required: (a) space-fixed, inertial reference system for the description of satellite motion; and (b) earth-fixed, terrestrial reference system for the positions of the observation stations and for the description of results from satellite geodesy. The two systems are used and the transformation parameters between the space fixed and earth fixed are well known and used directly in the GNSS receiver and post processing software to compute the position of the receivers in the earth fixed system. Terrestrial reference system is defined by convention with three axes, where Z-axis coincides with the earth rotation axis as defined by the Conventional International Origin. The X-axis is associated with the mean Greenwich meridian, and the Y-axis is orthogonal to both Z and X axes and it completes the right-handed coordinate system. GPS has used the WGS84 as a reference system and with WGS84 associated a geocentric equipotential ellipsoid of revolution [2].

In recent years, the emergence of GNSS receivers supporting multiple constellations has kept steady pace with the increasing number of GNSS satellites in the sky in the past decade. With advancements in newer GNSS constellations, almost 100% of all new devices are expected to support multiple constellations. The benefits of supporting multiple constellations include increased availability, particularly in areas with shadowing; increased accuracy, more satellites in view improves accuracy; and improved robustness, as independent systems are harder to spoof.

2.1.2 GNSS ERROR ANALYSIS

Ideally, with GNSS, we can get perfect localization results with no error at all. However, there are multiple places where error can be introduced in GNSS. In this subsection, we review these potential error contributors.

- **Satellite Clocks:** Any tiny amount of inaccuracy of the atomic clocks in the GNSS satellites can result in a significant error in the position calculated by the receiver. Roughly, 10 ns of clock error results in 3 m of position error.

- **Orbit Errors:** GNSS satellites travel in very precise, well-known orbits. However, like the satellite clock, the orbits do vary a small amount. When the satellite orbit changes, the ground control system sends a correction to the satellites and the satellite ephemeris is updated. Even with the corrections from the GNSS ground control system, there are still small errors in the orbit that can result in up to ±2.5 m of position error.

- **Ionospheric Delay:** The ionosphere is the layer of atmosphere between 80 km and 600 km above the earth. This layer contains electrically charged particles called ions. These ions delay the satellite signals and can cause a significant amount of satellite position error (typically ±5 m). Ionospheric delay varies with solar activity, time of year, season, time of day and location. This makes it very difficult to predict how much ionospheric delay is impacting the calculated position. Ionospheric delay also varies based on the radio frequency of the signal passing through the ionosphere.

- **Tropospheric Delay:** The troposphere is the layer of atmosphere closest to the surface of the Earth. Variations in tropospheric delay are caused by the changing humidity, temperature and atmospheric pressure in the troposphere. Since tropospheric conditions are very similar within a local area, the base station and rover receivers experience very similar tropospheric delay. This allows RTK GNSS to compensate for tropospheric delay, which will be discussed in the next subsection.

- **Multipath:** Multipath occurs when a GNSS signal is reflected off an object, such as the wall of a building, to the GNSS antenna. Because the reflected signal travels farther to reach the antenna, the reflected signal arrives at the receiver slightly delayed. This delayed signal can cause the receiver to calculate an incorrect position.

We have summarized the error ranges of these contributing sources in Figure 2.1. For a more detailed discussion of these errors, please refer to [3, 4, 5, 6].

Contributing Source	Error Range
Satellite Clocks	±2 m
Orbit Errors	±2.5 m
Inospheric Delays	±5 m
Tropospheric Delays	±0.5 m
Receiver Noise	±0.3 m
Multipath	±1 m

Figure 2.1: GNSS system errors (based on [3]).

2.1.3 SATELLITE-BASED AUGMENTATION SYSTEMS

Satellite-Based Augmentation Systems (SBAS) complement existing GNSS to reduce measurement errors. SBAS compensate for certain disadvantages of GNSS in terms of accuracy, integrity, continuity, and availability. The SBAS concept is based on GNSS measurements by accurately located reference stations deployed across an entire continent. The GNSS errors are then transferred

to a computing center, which calculates differential corrections and integrity messages that are then broadcasted over the continent using geostationary satellites as an augmentation or overlay of the original GNSS message. SBAS messages are broadcast via geostationary satellites able to cover vast areas.

Several countries have implemented their own satellite-based augmentation system. Europe has the European Geostationary Navigation Overlay Service (EGNOS) which mainly covers the Europe. The U.S. has its Wide Area Augmentation System (WAAS). China has launched the BeiDou System (BDS) that provides its own SBAS implementation. Japan is covered by its Multi-functional Satellite Augmentation System (MSAS). India has launched its own SBAS program named GPS and GEO Augmented Navigation (GAGAN) to cover the Indian subcontinent. All of the systems comply with a common global standard and are therefore all compatible and interoperable.

Note that most commercial GNSS receivers provides SBAS function. In detail, the WAAS specification requires it to provide a position accuracy of 7.6 m or less for both lateral and vertical measurements, at least 95% of the time. Actual performance measurements of the system at specific locations have shown it typically provides better than 1.0 m laterally and 1.5 m vertically throughout most of the U.S.

2.1.4 REAL-TIME KINEMATIC AND DIFFERENTIAL GPS

Based on our experiences, most commercially available multi-constellation GNSS systems provide a localization accuracy no better than a 2-m radius. While this may be enough for human drivers, in order for an autonomous vehicle to follow a road, it needs to know where the road is. To stay in a specific lane, it needs to know where the lane is. For an autonomous vehicle to stay in a lane, the localization requirements are in the order of decimeters. Fortunately, Real-Time Kinematic (RTK) and Differential GNSS does provide decimeter level localization accuracy. In this subsection, we study how RTK and Differential GNSS works.

RTK GNSS achieves high accuracy by reducing errors in satellite clocks, imperfect orbits, inospheric delays, and trophospheric delays. Figure 2.2 shows the basic concept behind RTK GNSS, a good way to correct these GNSS errors is to set up a GNSS receiver on a station whose position is known exactly, a base station. The base station receiver calculates its position from satellite data and compares that position with its actual known position, and identifies the difference. The resulting error corrections can then be communicated from the base to the vehicle.

In detail, RTK uses carrier-based ranging and provides ranges (and therefore positions) that are orders of magnitude more precise than those available through code-based positioning. Code-based positioning is one processing technique that gathers data via a coarse acquisition code receiver, which uses the information contained in the satellite pseudo-random code to calculate posi-

tions. After differential correction, this processing technique results in 5-m accuracy. Carrier-based ranging is another processing technique that gathers data via a carrier phase receiver, which uses the radio carrier signal to calculate positions. The carrier signal, which has a much higher frequency than the pseudo-random code, is more accurate than using the pseudo-random code alone. The pseudo-random code narrows the reference then the carrier code narrows the reference even more. After differential correction, this processing technique results in sub-meter accuracy. Under carrier-based ranging, the range is calculated by determining the number of carrier cycles between the satellite and the vehicle, then multiplying this number by the carrier wavelength. The calculated ranges still include errors from such sources as satellite clock and ephemerides, and ionospheric and tropospheric delays. To eliminate these errors and to take advantage of the precision of carrier-based measurements, RTK performance requires measurements to be transmitted from the base station to the vehicle.

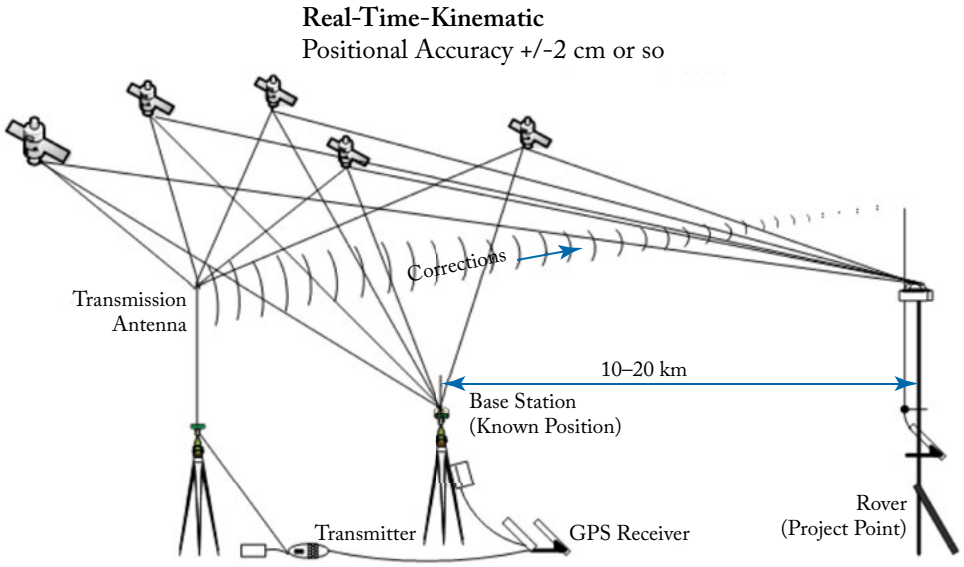

Figure 2.2: RTK GNSS (based on [46]).

With RTK GNSS, vehicles determine their position using algorithms that incorporate ambiguity resolution and differential correction. The position accuracy achievable by the vehicle depends on its distance from the base station and the accuracy of the differential corrections. Corrections are as accurate as the known location of the base station and the quality of the base station's satellite observations. Therefore, site selection is critical for minimizing environmental effects such as interference and multipath, as is the quality of the base station and vehicle receivers and antennas.

2.1.5 PRECISE POINT POSITIONING

Although RTK GNSS system provides sub-decimeter-level accuracy required to meet autonomous driving requirements, this solution often requires the users to deploy their own base stations, which are expensive to maintain. In this subsection, we study how Precise Point Positioning (PPP) GNSS system can help mitigate the problem [7, 8].

Figure 2.3 shows how a PPP GNSS solution works. Many reference stations are deployed worldwide, and these stations receive precise reference satellite orbits and reference GNSS satellite clocks in real time. These reference stations then calculate the corrections that should be applied to the satellite localization results. Once the corrections are calculated, they are delivered to the end users via satellite or over the Internet. The precise satellite positions and clocks minimize the satellite clock errors and orbit errors. We can then apply a dual-frequency GNSS receiver to remove the first-order effect of the ionosphere that is proportional to the carrier wave frequency. Therefore, the first-order ionospheric delay can be totally eliminated by using a combination of dual-frequency GNSS measurements. In addition, the tropospheric delay is corrected using the UNB model [9]: to achieve further accuracy, the residual tropospheric delay is estimated when estimating position and other unknowns [10]. By combining these techniques, PPP is capable of providing position solutions at the decimeter to centimeter level.

Specifically, the PPP algorithm uses as input code and phase observations from a dual-frequency receiver, and precise satellite orbits and clocks, in order to calculate precise receiver coordinates and clock. The observations coming from all the satellites are processed together in a filter, such as an Extended Kalman Filter (EKF). Position, receiver clock error, tropospheric delay, and carrier-phase ambiguities are estimated EKF states. EKF minimizes noise in the system and enables estimating position with centimeter-level accuracy. The estimates for the EKF states are improved with successive GNSS measurements, until they converge to stable and accurate values.

PPP differs from RTK positioning in the sense that it does not require access to observations from one or more close base stations and that PPP provides an absolute positioning instead of the location relative to the base station as RTK does. PPP just requires precise orbit and clock data, computed by a ground-processing center with measurements from reference stations from a relatively sparse station network. Note that PPP involves only a single GPS receiver and, therefore, no reference stations are needed in the vicinity of the user. Therefore, PPP can be regarded as a global position approach because its position solutions referred to a global reference frame. Hence, PPP provides much greater positioning consistency than the RTK approach in which position solutions are relative to the local base station or stations. Also, PPP is similar in structure to an SBAS system. Compared to SBAS, the key advantage provided by PPP is that it requires the availability of precise reference GNSS orbits and clocks in real-time, and thus achieving up to centimeter-level accuracy