CISC 322/326 Assignment 1: Report
Apollo: Conceptual Architecture Analysis

February 12, 2022

TEAM17 404 NOT FOUND
Qianran Liao (17ql32@queensu.ca)
Liangjin Lin (18ll51@queensu.ca)
Yihong Li (19yl31@queensu.ca)
Yiru Zhong (16yz139@queensu.ca)
Zitao Wang (18zw54@queensu.ca)
Ziyuan Yu (16zy18@queensu.ca)

Abstract

Baidu Apollo is an open-source platform for autonomous driving systems, with navigation pilot, valet parking, intelligent traffic signals, and intelligent vehicle infrastructure cooperation. It makes its systems open source and modifiable in a way that allows partners to use and customize the system to their own needs and desires. This article analyzes and discusses the conceptual architecture of Apollo open-source autonomous driving platform, which is a detailed study of the abstract architecture of Apollo open software platform, summarizes its derivation process, and explains the interaction of components and modules. It will be analyzed from several perspectives, including system evolution, architecture style, subsystems, control and data flow of the different parts, concurrency, and use case analysis. In addition to them, limitations and lessons learned are presented as well. All of these will be discussed in this report.

Introduction and Overview

Autonomous driving is set to be as life-changing as the invention of the motor vehicle itself. Nowadays, most of the giant companies in the IT industry and car industry worldwide have put great effort into the development projects of autonomous vehicles. The autonomous driving platform developed involves perception and artificial intelligence incorporating automatic functions like braking and propulsion (Behere and Törngren).

In order to achieve autonomous operation of a vehicle in urban situations with unpredictable traffic, several real-time systems must interoperate, including environment perception, localization, planning, and control. In addition, a robust vehicle platform with appropriate sensors, computational hardware, networking, and software infrastructure is essential (Levinson et al.).

Baidu's Apollo open platform is an open-source platform for autonomous driving, with over well-known 50 partners worldwide. It provides a complete software and hardware service solution, including cloud data service, software, vehicles, and hardware platform, to accelerate autonomous vehicles' development, testing, and deployment. Baidu provides open-source code and capabilities in obstacle perception, trajectory planning, vehicle control, vehicle operating system and other aspects, as well as a complete set of testing tools and other functions ("Baidu Apollo").

This paper aims to analyze the conceptual architecture of the Baidu Apollo open-source autonomous driving platform. This analysis paper conducts a comprehensive study mainly on the abstract architecture of the open software platform of Apollo, outlines the derivation process, explains the rationales for the components and modules and their interactions. The conceptual architecture analysis can be broken into 6 sections: 1) derivation process, 2) architecture style, 3) subsystems, 4) control and data flow, 5) use case analysis and 6) lessons learned and limitations. Sequence diagrams and use case diagrams are used to illustrate the system of Apollo and support the arguments of this analysis paper in the visualization aspect. Through the research, we concluded that the high-level of the Apollo architectural style is based on layered style. It also used pipe/filter style to realize functionality.

Derivation Process

At the beginning stage of analyzing, we learned as much as possible about Apollo through online searching, including the background of autonomous driving and its conceptual structure. Although we had a preliminary understanding of the Apollo autonomous driving platform at this time, the conceptual architecture was still the part that required us to investigate more deeply. By reading the documentation of the Apollo system, we formed a basic image of the structure of the conceptual architecture. There are seven modules to realize the basic functionalities: Map, Localization, Perception, Planning, Control, and HMI. There are also two extra modules to confirm the whole software system is running normally, which are not shown in the Figure 1, Guardian and Monitor.

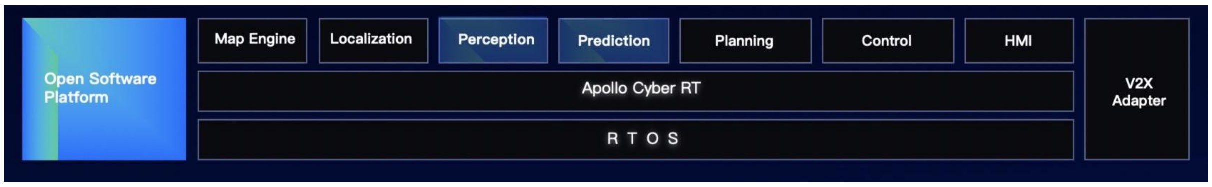

Figure. 1 Apollo Open Software Platform Structure
Source: Apollo. Apollo 7.0, https://github.com/ApolloAuto/apollo

To explore the interaction between different system components, we found a diagram of software overview from the official GitHub site of Apollo, illustrating data and control flow throughout the system.

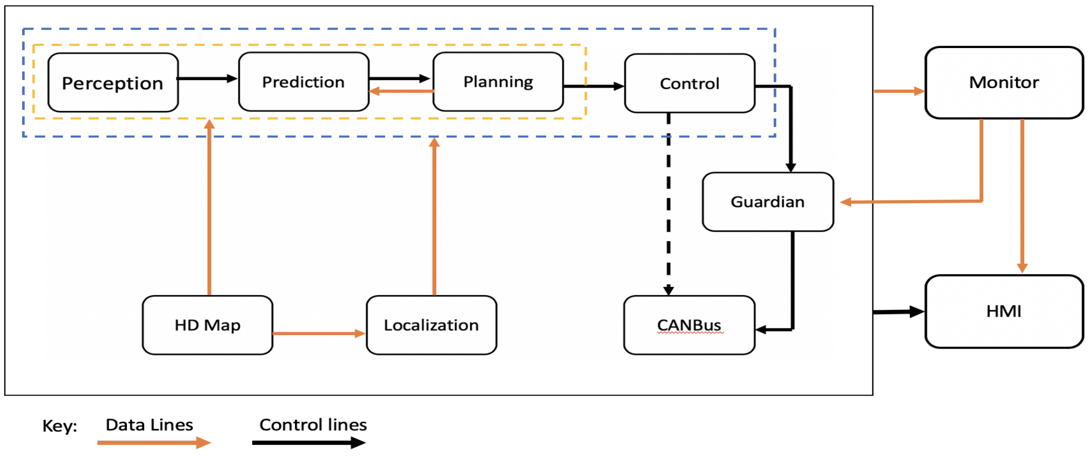

Figure. 2 Data and Control Flow of System
Source: Apollo. Software overview, https://github.com/ApolloAuto/apollo

Based on Figure 2, we found the documentation of each subsystem showing their functionalities and started to investigate and solve the problem of the architecture style. The data flow accords with the Layered style characteristic. The dependency is from the top tier, HMI through all the logic modules in the mid-tier to the bottom tier, HD MAP (which has storage of map database) and CANBus (the end of the software system which sends the command to hardware system).

After gathering more documentation for subsystems in the logic tier, we discovered that the data flow through perception, prediction and planning is suitable for the Pipe and Filter style. The data stream from the MAP database and localization module transfers to the filter formed by perception, prediction and planning modules to output to the control module.

To better understand the input and output for every component, we learned the Cyber RT framework, which allows high concurrency, low latency, and high throughput in autonomous driving. After studying the Cyber framework, it is easier to understand the coding part and clarify the input and output data throughout modules. Under the Cyber framework, every component can publish their message through a node or subscribe to other messages from other nodes. It accords to the characteristics of implicit invocation style, which involve a loosely-coupled collection of components, each of which carries out some operation and may in the process enable other operations.

Conceptual Architecture

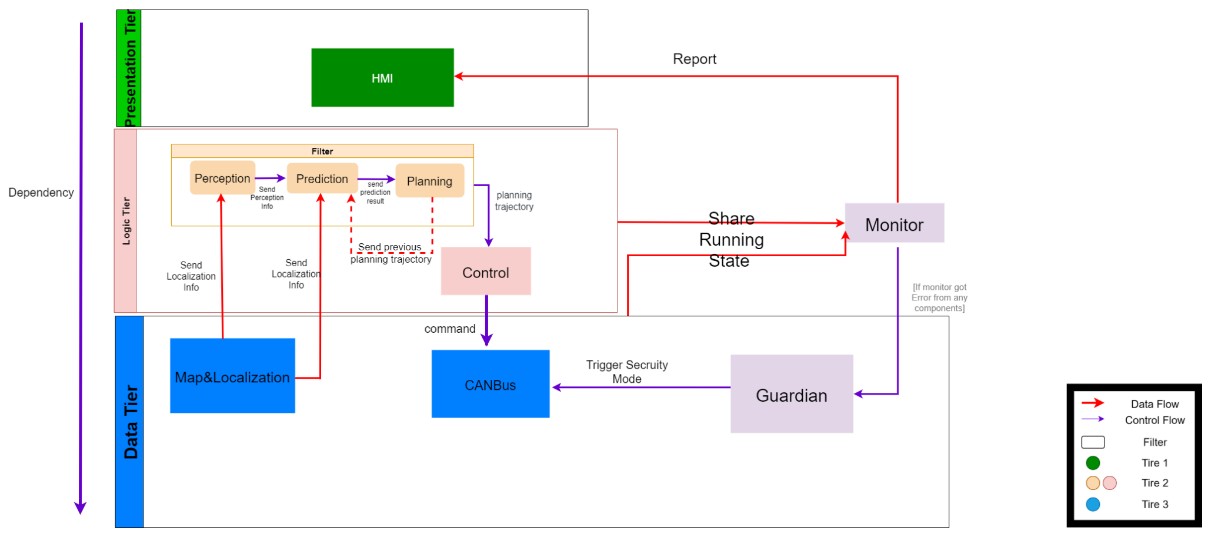

Figure 3. Conceptual Architecture

Architechture Style

Throughout the whole framework of Apollo, the high-level architecture is based on the layered/multitier style. The order of tiers from top to bottom shows the dependencies between different tiers. The top tier is the presentation tier which includes the HMI (Human Machine Interface) or Dream view. It is the presentation tier faced to users which helps developers visualize the output of other modules. The mid-tier is the logic tier which includes all the subsystems and modules involved in algorithm, arithmetic and logic operation functions. The bottom tier includes the base modules, which only receive instructions from the logic tier and give data back.

The entire logic tier is based on the Pipe Filter style. The pipeline part is data flow transfer from the Map module through three concurrency filters (perception prediction planning). The data output is the trajectories calculated by planning.

Subsystem Decomposition

Apollo is an open-source framework for autonomous driving which is developed by Baidu. The advantage of Apollo is that it has a high-performance and flexible architecture that can make the development, testing, and deployment process of autonomous driving more efficient.

Cyber RT

Cyber is an open-source runtime framework designed to improve concurrency and throughput, reduce latency based on a centralized computing model and is used in Apollo to replace the ROS messaging middleware. Under the Cyber framework, every component has one node with input and output objects based on client/server or reader/writer. Each module only contains one node. Client/server objects support interactive communication between two nodes. Reader/writer only support publish-subscribe style communication.

Perception

The perception module is indispensable for the autonomous driving system to obtain external information. This module mainly senses the current environment such as obstacles, traffic lights, lane lines, traffic signs, cars and pedestrians encountered during driving by calling hardware devices such as cameras, LIDAR and radar, and outputs information such as traffic light indications as well as distance, speed and position to the prediction module.

Prediction

When the prediction module receives the information from the perception module, it will be the first to start working. The prediction module needs to make a correct and appropriate judgment about what may happen in the scene in the future period. For this purpose, it will work with the planning module, the localization module, etc., to send some appropriate instructions to the car.

A complex predictive mathematical model is built in the prediction module. This mathematical model analyzes the received information and combines it with the current scene to give the possible velocity and path probabilities of each object and then predicts the object's trajectory.

Planning

The planning module is responsible for planning a feasible motion trajectory for the autonomous vehicle. The planning module needs to collect the current state of the car such as speed, position, direction, acceleration, etc., the information of maps such as road infrastructure, traffic lights, traffic signs and navigation information, and the information from the prediction module such as surrounding vehicles, pedestrians, non-motorized vehicles and other possible future states.

Referring to the documentation related to the planning module, compared to previous versions, "Apollo 7.0 adds a new dead-end scenario with the addition of "three-point turns", which increases the ability to drive in and out of vehicles and expands the operational boundaries of the urban road network. The "Three Point Turn" feature is based on the Open Space Planner framework and includes the following components: dead-end scene conversion, open space ROI construction, and "Three Point Turn" trajectory planning."

The planning module plans driving trajectories for autonomous vehicles based on the perceived situation, which is different from the routing module, which refers to short-term planning.

Control

The control module controls the vehicle's trajectory along the planned generated path, and it sends mechanical control commands to the CANBus module to achieve control of the vehicle. Depending on the planning control and the current state of the car, the control module uses different control input methods, which facilitates a more comfortable driving experience. The module can work in normal and navigation modes. It requires the information output by the planning module, the status of the autonomous vehicle and positioning information. After receiving the complete information, the control module uses a control algorithm to generate a series of control commands such as throttle, brake, steering, etc.

CANBus

The CANBus module is used to transmit control commands to the vehicle chassis, collect the vehicle chassis's status information, and send it back to the control module, etc., to correct possible deviations.

Localization

The positioning module integrates GPS, IMU LiDAR and other hardware devices in order to output the accurate position of the vehicle. It provides positioning services in two ways: RTK (Real Time Kinematics), which integrates GPS and IMU (Inertial Measurement Unit) information and a multi-perceptron fusion scheme that adds LiDAR information to RTK.

HMI

The HMI human-machine interface, including the driving display and DreamView (for viewing the autopilot process), has the main functions of visually showing the driver the driving status of the vehicle, testing the modules, providing debugging tools, facilitating real-time driver control of the vehicle's driving situation, etc.

Monitor

The monitor component is responsible for receiving data from all components and will pass it to the HMI component to provide information to the driver to ensure proper component operation. In the event of a module failure, the monitor sends an alert to Guardian, and Guardian decides to take action to prevent an accident or software crash.

Guardian

Guardian is a newly added module. It is like an action center that makes decisions based on the data sent by Monitor. 1. When all components are working properly, the guardian does not block the control flow, and control signals are sent normally to CANBus components for regular driving purposes. 2. When Monitor detects a component failure, it transmits a signal to the guardian that is like a trigger that will block the control signal to CANBus only when triggered and stop the vehicle under three conditions depending on the situation.

1) If the Ultrasonic sensor (Perception component) is working normally and no obstacle is detected near the vehicle, the guardian will stop the vehicle slowly.

2) If the sensor does not respond, the vehicle will be forced to stop by hard braking.

3) If the information from the Monitor to the HMI is not intervened by the driver within 10 seconds, the vehicle will be forced to stop by hard braking.

Other modules

StoryTelling is used to isolate and manage complex situations. Considering that some situations may be dangerous in real-world experiments, this module creates virtual scenarios that can trigger multiple module actions.

Calibration is the module which is used for the calibration of sensor coordinates and sensor fusion in the perception module.

Common is a collection of some commonly used basic functions.

Driver module is a collection of radar, lidar, GPS, CANBus, camera and other drivers.

V2x (Vehicle to Everything) module attempts to enable information interaction between vehicles and all objects that may affect them, so that this can reduce accidents, slow down traffic jams, reduce environmental pollution and provide other information services.

System Envolvement

The 1.0 Version of Baidu Apollo works only in an enclosed venue such as a test track or parking lot. In the following 1.5 version, Apollo focused on fixed lane cruising features and added a map engine module, perception module, and planning module. The map engine module helps the vehicles more accurately map their current position. The perception module was added to improve the ability of vehicles to perceive their surroundings. The vehicle with this version with the planning module could plan its trajectory for safer maneuvering on its lane.

Later 2.0 version of Apollo improved the three newly-released features and allowed the vehicles to drive on simple urban roads. Vehicles can cruise on roads safely, avoid collisions with obstacles, stop at traffic lights, and change lanes if needed to reach their destination. The improvement in the perception module and the planning module was implemented in the 2.5 version of Apollo. Vehicles can maintain lane control, cruise and avoid collisions with vehicles ahead of them on geo-fenced highways.

The system of open software platforms had no significant change in Apollo 3.0. More complex driving scenarios, such as downtown and residential areas, can be navigated by vehicles with Apollo 3.5. The system upgraded perception algorithms to handle the changing conditions of urban roads, making the car more secure and aware. The scenario-based planning module can navigate complex scenarios, including unprotected turns and narrow streets often found in residential areas and roads with stop signs. Besides, Apollo Cyber RT was introduced in version 3.5. The Apollo Cyber RT is designed as a centralized and parallel computing model, enabling high concurrency of task execution, low latency and high throughput. It results from years of building a framework that answers the highest performance requirements of autonomous driving solutions.

The perception module was upgraded to a deep learning model to handle the changing conditions. Scenario-based planning has been enhanced to support additional scenarios like pull-over and crossing bare intersections. A brand-new prediction model was added in the 5.5 Version. Apollo 7.0 incorporates 3 brand new deep learning models to enhance the capabilities for Apollo Perception and Prediction modules.

Control and data flow

The main components of the conceptual architecture establish the basic sequence diagram order. In the use case, this process starts when the user logs into the Autopilot system and turns on Autopilot. When Autopilot is turned on, the HML module checks the vehicle's status, tests other modules, and monitors the operation of each component of the vehicle in real-time.

When there are no operational problems, it sends data to the perception component, capturing the surroundings through a camera. The camera can identify any captured road information and traffic participants and draws a 3D image to return to the monitor. The perception component transmits the data to the decision module, which simulates the judgments made by the environment and transmits them to the manipulation component. A set of routes is then planned to match the actual conditions. The planned route is played back on the monitor as a 3D image. The control module captures a series of planned road data and prepares it for execution, sending the final decision to the CANBus component, which passes the control commands to the vehicle's hardware interface to control the vehicle's hardware for autonomous driving. At the same time, the Map & Localization component continuously updates the vehicle's latest position in real-time and sends back position information to the HMI component so that the planned route can be better updated. The monitor will also send back the vehicle driving status report to the decision module to check the vehicle driving status.

In addition, the guardian is the safety module for autonomous driving. The monitor transmits all data information to the guardian component and monitors it in real-time to intervene to ensure the safety of the vehicle when a component failure or abnormal behavior is detected.

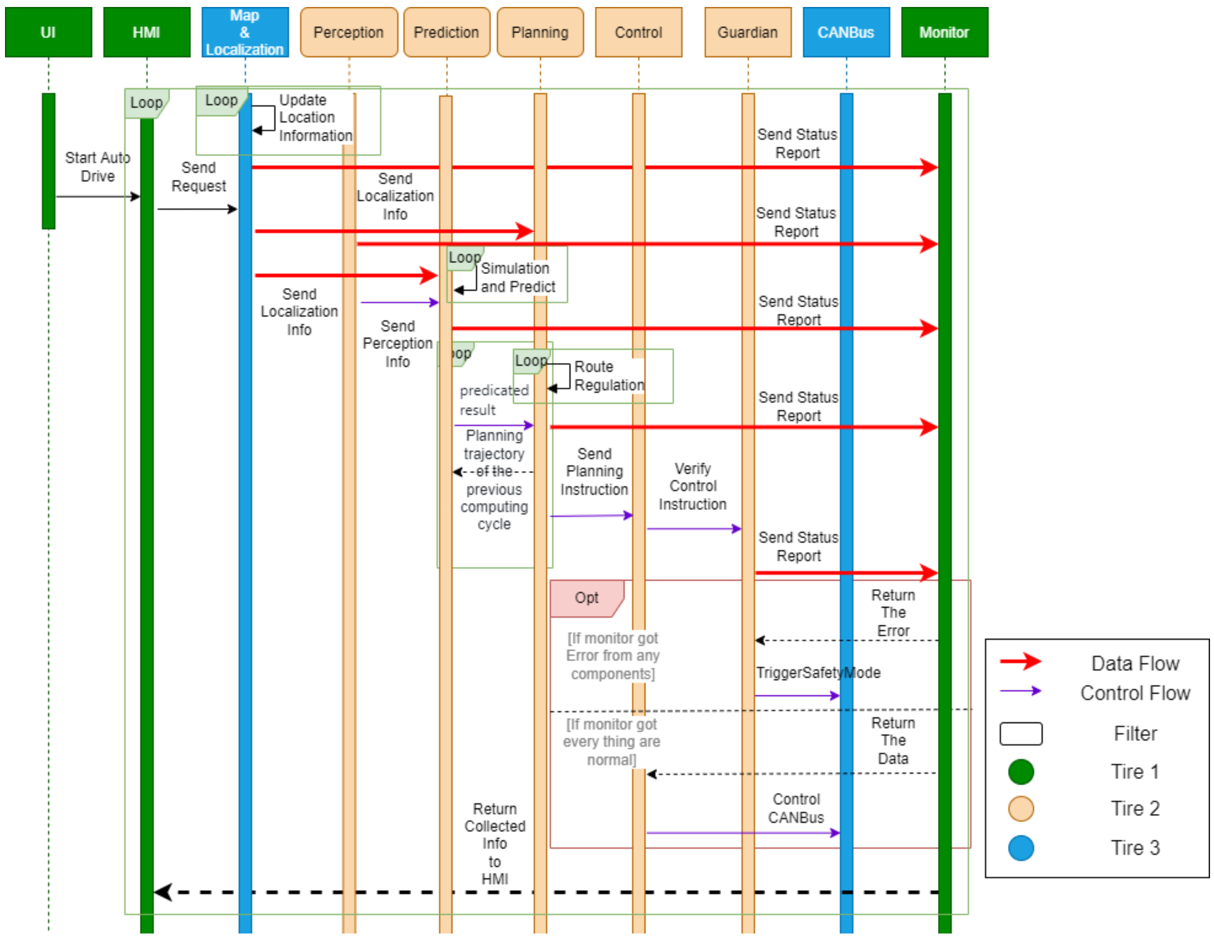

Figure 4. Sequence Diagram of Apollo's conceptual Architecture

Concurrency

For a system, concurrency means that it has multiple parts and can execute simultaneously in any order without affecting each other and the outcome. Concurrency refers to the parallel execution of order-independent, potentially interactive computer units or components, which are the decomposition of a program, algorithm, or problem.

Apollo uses a multi-process architecture. Apollo's architecture and modes of action result from the cooperation of multiple processes that interact with each other. Notably, there is concurrency between the components that form the architecture. Each module exists in the form of components and has its process. Moreover, each process is responsible for processing different data seemingly simultaneously. The processed data is transmitted between components by the communication system using data channels. As a result, each component constantly receives new data and handles different tasks simultaneously. Therefore, if a component crashes, this is detected by the monitor component, which then sends a warning to Guardian to intervene with the CANbus component. Without concurrency, the crash of one vehicle component will cause the entire vehicle to fail, resulting in an accident. Furthermore, when much external information needs to be processed, the components cannot process the information quickly without concurrency, and the whole subsystem will run so slowly that the system's response time will be slowed down. Inefficiency can significantly negatively impact the safety and performance of autonomous driving.

The processes in the multi-process architecture apply multi-threaded architecture, meaning that multi-process systems execute multiple processes simultaneously, while multi-threaded systems execute multiple threads of a process simultaneously. That is, multi-threaded systems can further improve the concurrency of multi-process systems.

Therefore, the Apollo system needs concurrency to ensure a very high utilization rate of external resources and fast program response speed in order to minimize the potential safety hazard of autonomous driving to the outside world and itself and improve system performance.

Developer contributes

Apollo is an open-source platform that will help partners in the automotive industry and autonomous driving quickly build their autonomous driving systems combining vehicles and hardware systems. The platform provides cloud service, open software, hardware support and open vehicle certificate service to partners and users. According to Simon Sword's article, a complete development system should be separated into the technical lead, software developers, and software tests. Besides, the administrators and collaborating contributors are also two stakeholders because Apollo open software platform is an open-source project throughout the GitHub platform.

The technical lead is responsible for translating business requirements into technological solutions (the development team leader). The software developer is subdivided into two parts, the front-end and back-end, which are responsible..... Besides, the software test ensures that the software solution fulfills business requirements and is bug, error, and defect-free.

External Interfaces

HD-Map

HD-Map is used for querying, providing specially structured information about road conditions

Use Cases

Pull Over

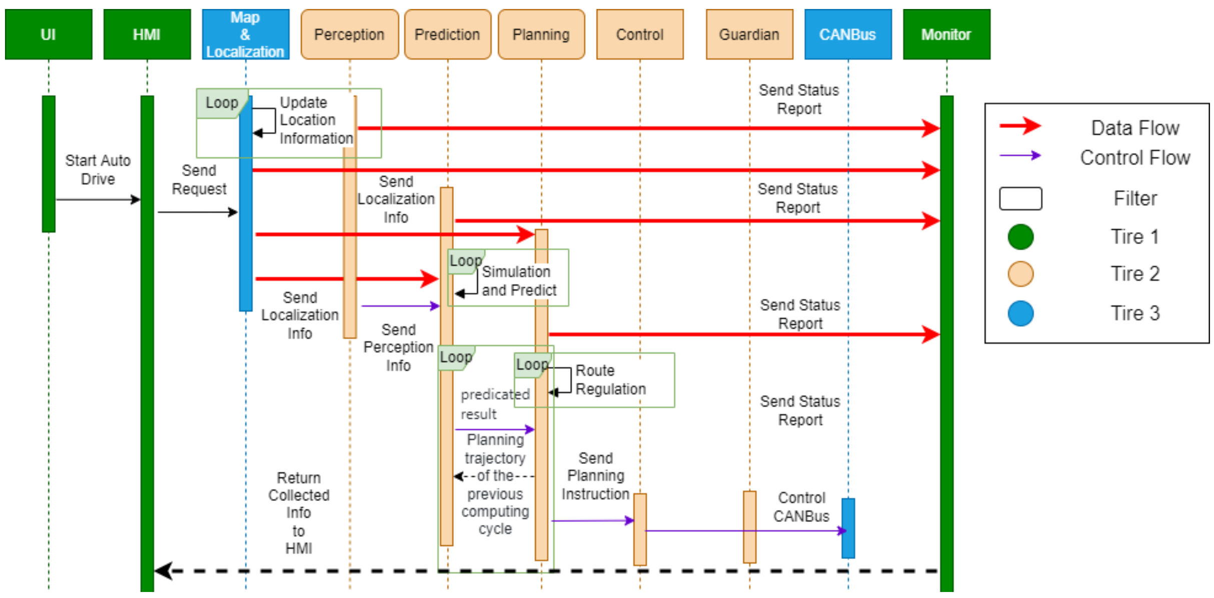

Figure 5. Sequence diagram of Use Case 1: Pull Over

When the vehicle needs to pull over when driving, the Map and Localization components will send localization information to the Planning component to find a parking area on the map and confirm the target localization. The planning component determines if the endpoint is suitable for parking, if the endpoint is not at an intersection, and if the endpoint lane allows pulling over parking.

After the planning component determines that parking is allowed, the perception component will capture the road surface information through the camera and determine which driving road the vehicle is on. When the vehicle is within a certain distance from the endpoint, the prediction component sends a message to the planning component, starting to execute the shortest distance driving and pull over the parking plan. When the vehicle reaches the stopping point, the camera continues to capture road information and transmits it to the prediction component to determine the road conditions at the stopping location, such as whether there will be pedestrians, bicycles, and other obstacles, and then transmits the data to the planning component. When there is no problem, the planning component executes the pullover and sends the planning data to the control component. After verification, the guardian detects no data abnormalities and can send data to the CANBus component to control the vehicle to execute the pullover parking.

Emergency Braking

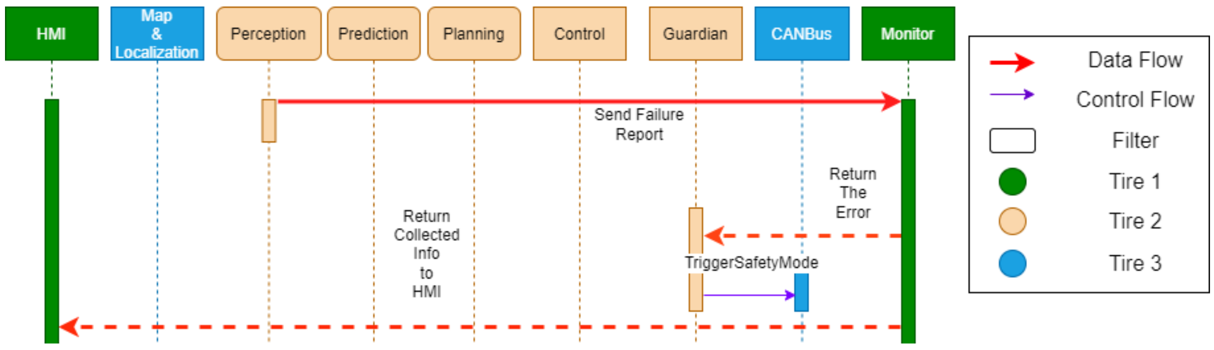

Figure 5. Sequence Diagram of Use Case 2: Emergency Braking

While the vehicle is in motion, the perception component continuously monitors road information such as camera captures and sensors around the vehicle to remain sensitive. When all modules are functioning normally, the Guardian module allows the data to flow normally, and then the control module transmits signals to the CANBus module to control the vehicle normally.

When a failure occurs in any component of the whole software system, the Monitor detects the failure and sends the data stream to the Guardian module, allowing the Guardian module to resolve the crash. In this case, the Guardian module will directly prevent the control signal from reaching the CANBus module and intervene with the vehicle hardware to force the vehicle to stop moving by applying hard brakes.

Lessons Learned and Limitation

The development and application of driverless systems is a significant milestone in the invention and use of vehicles. Apollo platform is still in the way of ensuring the vehicles drive safely in all real-life scenarios that have more complex traffic situations.

Although the Apollo platform looks very promising, there are still some limitations in its use:

1) There is no formal law to accept driverless vehicles worldwide. According to the law, driving a motor vehicle should obtain a driver's license, and the driver must be a human being, so driverless cars on the road are illegal. In addition, if a driverless car is operated improperly and causes an accident, there is no office to pursue the responsibility that should be.

2) Currently, Baidu's ACE intelligent traffic engine is implemented in 30 cities such as Beijing, Guangzhou, Shanghai and Chongqing, but there is not enough artificial intelligence to support full-scene driverless autonomous driving. Road

conditions, weather, environment and various natural and unnatural factors cannot be handled well. The current autonomous driving has not yet reached a reasonably mature point. So the popularity of L4 level autonomous driving still has limitations.

Lessons learned from our analysis of Apollo's open-source autonomous driving platform:

1) For the smooth operation of AI, it is necessary to provide a large amount of data during the research and development to face unexpected situations.
2) Consider all factors before research and development, including funding, developers, and sites.

Conclusions

In conclusion, the function and interaction of modules, subsystems, and components make the Apollo system stable and efficient performance. Multiple architectural styles lay the architectural foundation for integrating modules within the Apollo system. The system's evolution shows how modules and subsystems have been refined and improved over the years. In short, as an open-source autonomous driving platform, we have analyzed and described the Apollo autonomous driving system comprehensively and in detail from different aspects and perspectives. Apollo provides an open, complete, and secure software platform to the automotive industry and autonomous driving partners. Although Apollo autonomous driving has a potential future, there are always limitations. If future AI technology can replace the current road control system, autonomous driving technology will be more mature and safe.

Glossary

Message: The smallest data unit in the Cyber framework.

Actor: can also be interpreted as the “platform”. Which is initiated by users and starts the autodrive system.

Dreamview: a simulator which can play back the data during the automatic driving process.

Autopilot: an autopilot is a system used to control the path of an aircraft, marine craft or spacecraft without requiring constant manual control by a human operator.

CAN Bus: Controller Area Network (CAN bus) is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other’s applications without a host computer.

HMI: The Human Machine Interface (HMI) is the connector between the driver and the vehicle.

ROS: Robot Operating System (ROS) provides tools for accessing data from sensors of the vehicle, and it is the messaging middleware of the car.

GPS: Global Positioning System, is a global navigation satellite system that provides location, velocity and time synchronization.

3D: Three-dimensional

ROI: Region of interest is the rough translation of “relevant measurement range”. The term is used to refer to the relevant section of a measurement curve. This area can then be regarded preferably statistically.

References

Apollo Cyber RT framework. Apollo, https://apollo.auto/cyber.html. Accessed 20 February 20, 2022

Apollo_5.5_Software_Architecture.md. Apollo 5.5 Software Architecture, commit f7d1f94f5342737f33b52132bbd5a98243fd820b. Apollo. 2020. Github, https://github.com/ApolloAuto/apollo/blob/master/docs/specs/Apollo_5.5_Software_Architecture.md

Baidu Apollo open source autonomous driving platform. neousys technology, https://www.neousys-tech.com/en/discover/fanless-in-vehicle-pc/baidu-apollo-open-source-autonomous-driving-platform. Accessed 20 February 20, 2022.

Behere, Sagar & Martin Törngren. "A functional reference architecture for autonomous driving." Information and Software Technology, Vol. 73, 2016, pp. 136-150.

Levinson, Jesse, et al., "Towards fully autonomous driving: Systems and algorithms," 2011 IEEE Intelligent Vehicles Symposium (IV), 2011, pp. 163-168, doi:10.1109/IVS.2011.5940562.

README.md. *Apollo architecture*, commit 43cfadd13ba3b8465dbcdb021acfd49f38260d2a. Apollo, 2021. *Github,* https://github.com/ApolloAuto/apollo#architecture.

Software development team roles and responsibilities: Atlas. Atlas Computer Systems Ltd, 19 Jan. 2020, https://www.atlascode.com/blog/software-development-project-roles-and-responsibilities/.