CISC 322

Assignment 1
Conceptual Architecture of Apollo

Feb 14, 2022

Runfeng Qian (group leader)	18rq10@queensu.ca
Wonton Zhou	19bz6@queensu.ca
Ziheng Yang	18zy59@queensu.ca
Runze Lin	18rl30@queensu.ca
Junyu Yan	19jy28@queensu.ca
Haoyun Yang	18hy58@queensu.ca

Abstract

Baidu Apollo program provides a completely open, reliable and embracive software platform for the autonomous driving industry. A conceptual architecture for the Baidu Apollo, an open-source platform for autonomous driving cars, was concluded through hours of analysis and discussion in Apollo's Github codes, release notes and software components. The ultimate goal of determining the architecture design model of Apollo is to value its safety, performance, and most importantly its reliability.

Our team concluded that the most appropriate architecture design style of Apollo is a pipe and filter architecture style. The subsystems that formed the architecture are map engine, localization, perception, prediction, planning, control and lastly, human-machine interface modules. After an in-depth look through the conceptual architecture, details and uses of the software components are presented in this report to demonstrate the interactions between subsystems and components. Three sequence diagrams are also provided to further illustrate the interactions between modules in different use-cases.

Introduction & Overview

The Baidu Apollo, often referred to as Apollo open platform, is a revolutionary open-source software platform for the automotive industry and autonomous driving industry. The initiative aims to build a ubiquitous and cooperative ecosystem, leverage Baidu's technological advantages in artificial intelligence, and promote the development and popularization of autonomous driving technology. The program was named "Apollo," borrowing the meaning of the Apollo moon program. Apollo platform is a complete set of software and hardware and service systems, including four parts which are vehicle platform, hardware platform, software platform, cloud data services. This report only focuses on the software components of the platform as indicated in the guidelines.

Apollo's architecture has two forms of open autonomous capability: open code and open capability. Open ability is based on the ability provided by Baidu through API or SDK through standard disclosure. Open code is the same as the general traditional open and open-source software. The code is open, and everyone can use it and participate in the development together. The open range includes the perception system, path planning, vehicle control system and other essential components. Apollo opens code or capabilities for environment awareness, path planning, vehicle control, onboard operating systems and provides complete development and testing tools. The aim is to build a cooperative ecosystem, leverage Apollo's technological advantages in the field of artificial intelligence, empower open-source partners, and jointly promote the development and popularization of autonomous driving technology. By determining the conceptual architecture of Baidu Apollo, it was evident through the subsystems and components how the factors have led to success and popularity for the Apollo open-source platform. The main subsystems and components in the conceptual architecture are map engine, localization, perception and prediction modules, planning and control, human-machine interface. The dependencies between these subsystems and components formed an architecture that exhibits pipe and filter architecture styles. Open source, Accurate Perception and Simulation, Intelligent Control and planning with Apollo Dataset are some of the key features and benefits of the Apollo platform. Each key feature has reasons to be beneficial. For instance, the autonomous perception system is backed by both Baidu's big data and deep learning technologies, as well as a vast collection of real-world labelled driving data. Apollo's innovative open-source online platform and pipe and filter architecture style with a huge dataset are the crucial reasons it's one of the leading edges of the autonomous driving industry.

The stakeholders of the Baidu Apollo open-source autonomous driving platform include Baidu Inc, the developers of the autonomous industry around the globe and customers interested in self-driving cars or taxis.

Development Stages

The development stages and backgrounds of Apollo. For instance, during the development, the architecture analysis team constantly corporates and communicates with the design and implementation team on the software architecture structure. Hence the testing team and implementation team make the ideas generated by the architecture analysis team true. Once the software is completed for testing, the testing team of Apollo hence tests it on the Apollo studio, which is a simulation engine, to cope with and deal with different usage scenarios. If a software test fails, the testing team examines the information and then communicates with the implementation team, analyzes and finds the modules that caused the failure, and then tells the implementation team to improve on. Once the testing is completed, the maintenance team will maintain the software releases.

Derivation Process

The architecture proposed in this report arrived with a 3 step derivation process. The three steps were individual reasoning, group discussion, and finally the group summary. Every member of our team develops their own conceptual Architecture on their own first, including the architectural style and main modules. Then we group up together to share and argue the final architecture, discussing each one's main advantages and disadvantages. Each team member's architecture is slightly different from one another. The subsystems are all important ones to be included in a high-level architecture, and how these subsystems influence each other was also fully discussed. However, after the final discussion, we determined and concluded that the key subsystems are Map Engine, Localization, Perception, Prediction, Planning, Control and HMI, and the architectural style should be pipe and filter architectural style.

When analyzing and figuring out the interactions between all subsystems, we initially thought that there are 17 main modules. But after reviewing the course materials and the guidelines of this project, we figured that we should strictly follow the guidelines in which we should ignore the RTOS modules and the cloud service, also all the modules we choose should be the software components within the "Open Software Platform". After reviewing the guidelines, we reduced the number of modules in our architecture to 7 modules and we determined that the architectural style we choose should be the pipe and filter architecture. This also led us to derive the conceptual architecture that is diagrammed in Figure 1.

To further understand the architecture and the modules, we developed 3 use-cases with sequence diagrams to show the relations between these high-level modules. We dived into the GitHub pages and sources to determine their relations, including design documentation from blogs, videos, and discussion forums. After days of research and discussion, we finally concluded how the modules work together and how the sequence diagram should be introduced. Then we proposed our conclusion of this architecture report. Hence this is our whole derivation process.

Conceptual Architecture

After a long period of discussion and researching, we agreed that the most appropriate architecture style for autonomous driving car software is Pipe and Filter architecture style. Each component has to run in a specific order to process the data. All components have some output that can be used as input of another component.

The entire architecture has 7 main components: map engine, Localization, perception, prediction, planning and control. Map Engine gets the high-definition 3D map from the distant server and sends map information to the localization component. The localization component incorporates information from the Hardware part to obtain the precise location on the map. The perception module collects the environment information from sensors and receives the information of velocity and angular velocity of the host vehicle from the localization component. The Perception module analyzes and classifies the environment information, and sends the 3D obstacle tracks with the heading, velocity, classification information and traffic light detection and recognition information to the prediction component. After combining the outputs from Map Engine (Map information), Localization (precise location) and Perception module (physical obstacle information), the prediction module will calculate and output information of obstacles annotated with predicted trajectories and their priorities to the planning module. Moreover, the prediction component takes the planning trajectory of the previous computing cycle from the planning module as input for dynamic adjustment. After obtaining information about the autonomous vehicle's status (from localization component), map (from map engine), and surrounding obstacles (from prediction component), the planning module generates a feasible space-time trajectory for the autonomous vehicle and sends it to the control component. Depending on the planned trajectory (from the planning component) and the current state of the car (localization component), the control module uses different control algorithms to give the passengers as comfortable a driving experience as possible [5]. HMI provides a human-machine interaction interface, visualizes information from all other modules for easy manipulation.

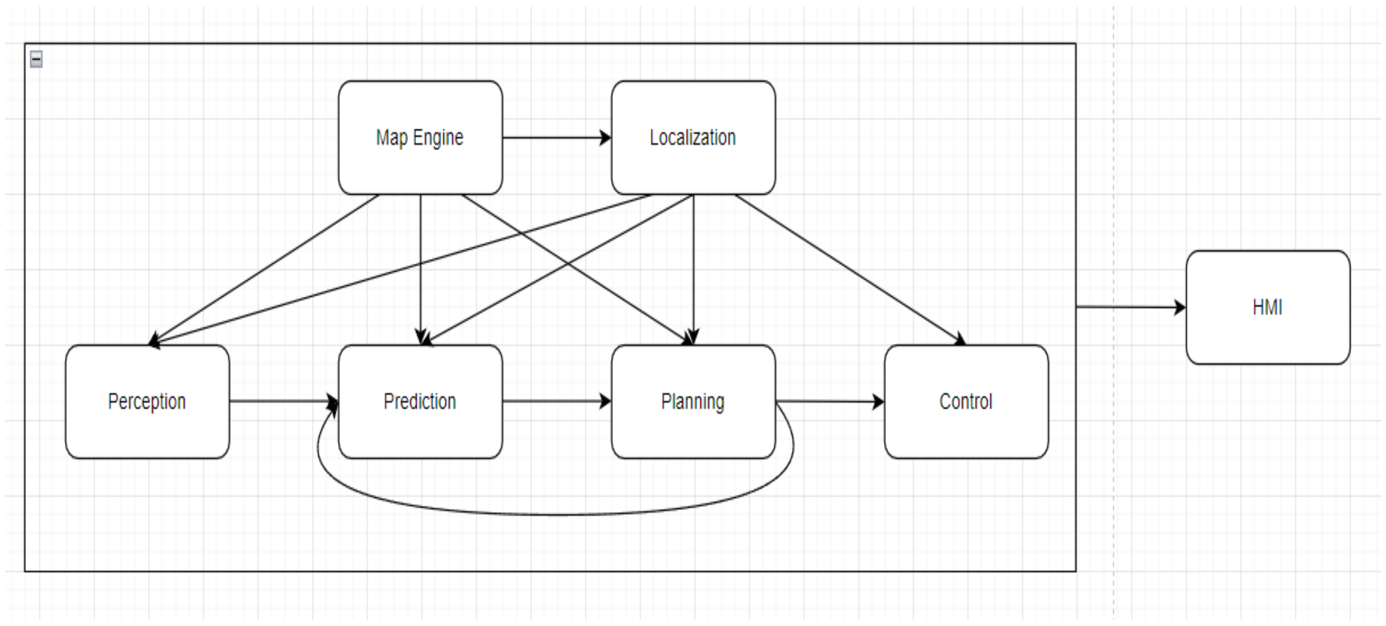


Figure 1: Conceptual architecture

Sub-Systems:

Map Engine

Map Engine is responsible for obtaining all kinds of Map data in the software and providing corresponding Map data function interfaces. Mainly used for queries, providing tailored structured information about road conditions. The tailored structured information is in the open drive format and provides a series of APIs for other modules to access and use.

Apollo's high-precision map adopts the open drive format, which is a unified map standard, which ensures the universality of the map. The main function provided by the map module is to read high-precision maps and convert them into Map objects in the apollo program. To further demonstrate, it indicates to read the open drive high-precision map in XML format into a format that the program can recognize and understand. The function that the map module does not implement is the production of high-precision maps.

The open drive format data is divided into map header information and structure. The header information mainly introduces the basic information of the map "version, time, projection method, map size, manufacturer, etc.". Structures are primarily different components of a road, including "crosswalks, intersection areas, lanes, parking observations, signals, give way signs, overlapping areas, no parking, speed bumps, roads, parking areas, curbside paths, or pedestrian walks. road".

Localization

The localization system is a comprehensive positioning solution with centimetre level accuracy based on GPS, IMU, HD map, and a variety of sensor inputs.

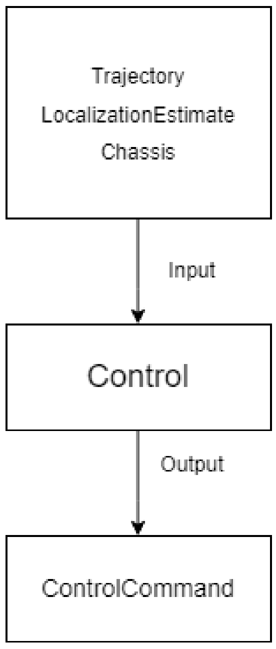
Specifically, this module provides localization services in two ways:

The RTK (Real-Time Kinematic) based method incorporates GPS and IMU (Inertial Measurement Unit) information and the multi-sensor fusion method which incorporates GPS, IMU, and LiDAR information.

The two inputs in the RTK localization method include the GPS(Global Positioning System) and the IMU(Inertial Measurement Unit). Contrastly, there are three inputs in the multi-sensor fusion localization method which are GPS, IMU, and LiDAR(Light Detection And Ranging Sensor). Either way, the localization outputs the precise and exact information of the vehicle's location.

Control

Figure 2: Control Module



The function of the Apollo control module is to calculate the accelerator, brake and steering wheel signals of the car according to the trajectory generated by the planning (planning module), and to control the car to drive according to the specified trajectory. The method adopted is to model the car based on the knowledge of car dynamics and kinematics to realize the control of the car. At present, Apollo mainly uses two control methods: PID control and model control. Input - Chassis (vehicle status information), LocalizationEstimate (location information), ADCTrajectory (trajectory planned by the planning module) Output - ControlCommand(Accelerator, Brake, Steering)

The Apollo intelligent vehicle control and canbus-proxy modules are precise, broadly applicable and adaptive to different environments. The modules handle different road conditions, speeds, vehicle types and CANbus protocols. Apollo provides waypoint following capability with a control accuracy of ~10 cm.

According to the planned trajectory and the current state of the car, the control module uses different control algorithms to bring the passengers as comfortable a driving experience as possible. The control module can work in normal and navigation mode. The input it requires is the planned space-time trajectory, the state of the autonomous vehicle, and the positioning information. After the control module receives the complete information, it will use the control algorithm to generate a series of control commands such as accelerator, brake, and steering.

Planning

Apollo vehicles are equipped with a planning system consisting of prediction, behaviour, and motion logic. The planning system adapts to real-time traffic conditions, resulting in precise trajectories that are both safe and comfortable. Currently, the planning system operates on a fixed route in both night/day conditions.

The planning module is responsible for planning a feasible spatiotemporal trajectory for the autonomous vehicle based on the results of the perception module and prediction module, the current vehicle information and road conditions, and this trajectory will be handed over to the control module. To accomplish this difficult task, the planning module needs the self-driving vehicle's own state, such as speed, position, orientation, acceleration, and other map-related information, such as road infrastructure, traffic lights, traffic signs, and navigation information around obstacles, Such as the possible future states of surrounding vehicles, pedestrians, non-motor vehicles, etc.

Apollo introduces two new planning modes: E2E mode and hybrid mode. Both modes employ a learning-based approach to planning future driving trajectories. The control module makes the vehicle through the accelerator, brake and steering wheel Follow the planned trajectory. The trajectory of the planning module is the short-term trajectory, that is, the trajectory of the vehicle in a short period of time, and the long-term trajectory is the navigation trajectory planned by the routing module, that is, the trajectory from the starting point to the destination. Following a short-term trajectory until the destination. This is also easy to understand. We turn on the navigation before driving and then drive according to the navigation. If there is a car in front, we will slow down or change lanes, overtake, avoid pedestrians, etc. This is a short-term trajectory, combining the above methods until we reach the destination.

Perception

The main function of the perception component is detection and classification, since the autonomous driving car needs to know the location of the obstacles and then classify them. There are two types of obstacles, which are static obstacles and dynamic obstacles. Static obstacles include walls, trees, buildings, etc. Dynamic obstacles include people, cars, etc. Various sensors, such as LiDAR, cameras and radar collect environmental data surrounding the vehicle. Using sensor fusion technology, perception algorithms and convolutional neural networks can determine in real-time the type, location, velocity and orientation of obstacles on the road.

This autonomous perception system is backed by both Baidu’s big data and deep learning technologies, as well as a vast collection of real-world labelled driving data. The large-scale deep-learning platform and GPU clusters drastically shorten the learning time for large quantities of data. Once trained, the new models are deployed onto the vehicle using over-the-air updates through the cloud. Artificial intelligence and data-driven solutions combine to enable Apollo’s perception system to continuously improve its detection and recognition capabilities, which provide accurate, stable, and reliable input for other autonomous system modules.

HMI

Apollo's HMI module provides a web application that helps developers visualize the output of other relevant autonomous driving modules, the vehicle's planning trajectory, car localization, chassis status etc. Specifically, the HMI module includes driving display and DreamView (for reviewing the automatic driving process), the main function is to visually check the driving status of the vehicle, test the module, provide debugging tools, and facilitate the driver to control the driving of the vehicle in real-time Wait.

HMI evaluates and monitors inputs from the following modules: Localization, Chassis, Planning, Monitor, Perception, Prediction, Routing. After monitoring, the HMI module outputs
a web-based dynamic 3D rendering of the monitored messages in a simulated world.

Prediction

Apollo’s prediction module receives obstacle information from the perception module, and the autonomous driving system must be able to predict what may happen in the scene for a short period in the future to make the correct movement as we do. In order to do that the module will also receive the localization information from the localization module and planning trajectory from the planning module in order to get the more precise output of the obstacle information with predicted trajectories and their priorities(ignore, caution and normal) [6].

The prediction module has 4 main functionalities: Container, Scenario, Evaluator, and Predictor. and these 4 main functionalities are also following the pipe and filter architecture style (They add the Scenario filter between container and evaluator in Apollo 3.5, which shows the advantages of this style: easily maintained and enhanced). Specifically, the Container stores input data, perception obstacles, localization of the vehicle and the planning, the Scenario analysis scenarios that include ego vehicles, they defined two scenarios, cruise and junction. Furthermore, the Evaluator predicts path and speed for the given obstacle and the Predictor generates predicted trajectories for obstacles, which is the final output.

Use Cases

Use Case 1: User enters the desired destination, Apollo program plans route accordingly.

dense line: control flow
dotted line: data flow
dest: destination

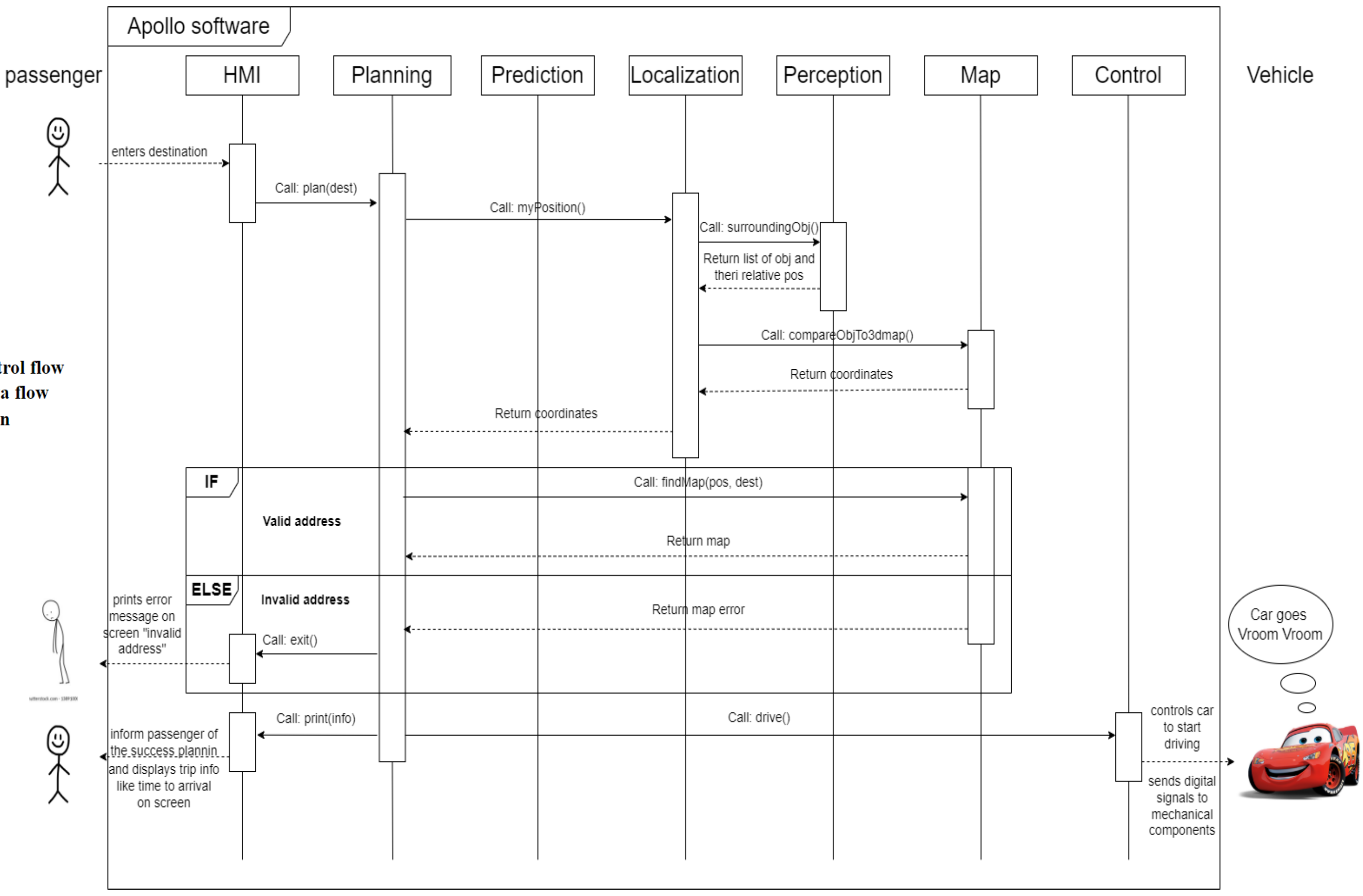


Figure 3: Sequence Diagram for a user using Apollo to plan to the desired destination

Figure 3 depicts the process of route planning to the user's desired destination. To begin this process, the user inputs the desired destination into HMI. The HMI then calls a route planning function from the Planning module. The Planning module calls Localization module to check the current position of the vehicle, in order for localization to work the module needs info from the perception and map module, Localization module checks the surrounding environment by calling the Perception module, Localization module also calls the Map Engine to get a detailed 3d map of the general area. And figures out the coordinates of the current location by comparing the surrounding features with the 3D map, the Map Engine module then returns the coordinates to the planning module. Then, the planning module calls the find function in the map module to get the coordinates of the desired destination. If the destination address is invalid planning exits execution and informs the user via HMI error message, if the address is good, planning calculates an optimal route. and sends instructions on how to follow the route to the control module via calling the drive function, drive function then sends digital signals to the mechanical car components beginning the trip. Meanwhile, planning informs the passenger via message through HMI on trip info like time till destination and current location.

Use Case 2: Apollo system encountered a closed road, proceed to recalculate the route.

dense line: control flow
dotted line: data flow

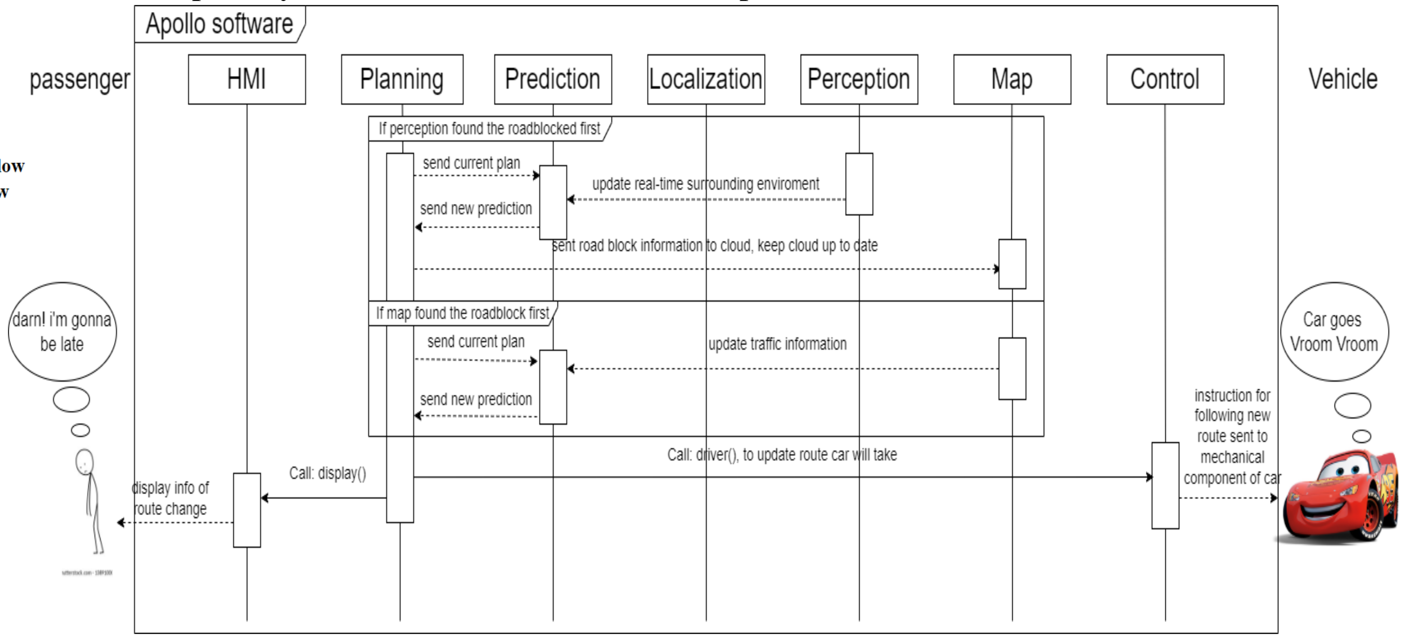

Apollo software
passenger
HMI
Planning
Prediction
Localization
Perception
Map
Control
Vehicle
If perception found the roadblocked first
send current plan
update real-time surrounding enviroment
send new prediction
sent road block information to cloud, keep cloud up to date
If map found the roadblock first
send current plan
update traffic information
send new prediction
Call: driver(), to update route car will take
Call: display()
display info of route change
instruction for following new route sent to mechanical component of car
darn! i'm gonna be late
Car goes Vroom Vroom

Figure 4: Sequence Diagram for Apollo rerouting after encountering a closed road

The case below assumes that the planning module already has info like current location, map and traffic info.

Figure 4 depicts a rerouting procedure after encountering a closed road. There are two start scenarios for this specific use case. In the first scenario, the perception module first gets hold of road blocked signs and objects via sensors and sends the data to be analyzed by the prediction module, prediction then uses the current plan provided by planning and info provided by perception to predict outcomes if the car follows the current plan. The prediction module then sends the predicted information to the planning module which promptly sends this info to the map module to inform and update the cloud maps. The second scenario starts out when the map module sends data of a closed road ahead to the prediction module, the prediction module then follows the same step taken in the first scenario and sends a troubling prediction to planning, the difference in this case being we do not need to update the map. When planning gets hold of prediction data it analyzes it and finds out the original plan will encounter stoppage/deadlock. The planning module will then call its internal sub-modules or functions to calculate a new route based on current location, destination, traffic information and map. It then updates the control module's drive function accounting for the new route, and informs the passenger via HMI about the closed road and the changed route.

Use Case 3: Apollo system encounters human activity on road ahead, takes precautions to ensure safety.

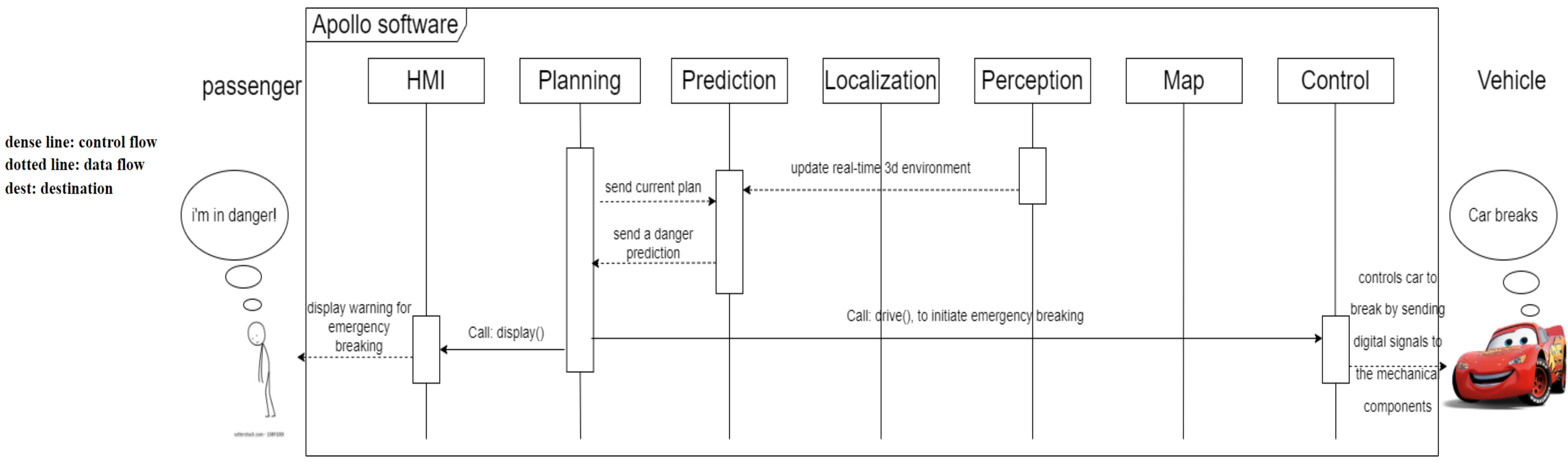


Figure 5: Sequence Diagram for Apollo reacting to the emergency situation of human presence on road ahead.

In every process cycle, the plan module and the perception module update the prediction module with new plans and objects picked up by the sensors (real-time). In our case above, during a cycle, after the prediction process got the current plan and surrounding environment, it recognizes a collision with a human in the road ahead if the car continues to follow the current plan. The prediction module then sends the prediction data to the planning module. The planning module then analyzes the prediction, recognizes the dangerous situation ahead and calls an emergency response function or sub-module within itself to react and prevent the problem from happening, the planning module then determines the appropriate actions like whether to change directions or in the example case breaking. planning then calls the drive function from the control module to initiate emergency action, which is relayed to the mechanical systems of the car. at the same time planning also informs the passenger of the emergency procedures that will be taken via the HMI.

Concurrency

For synchronization, all components can run at the same time. They do not have exclusive access to some specific data. From a macro perspective, when a self-driving car is moving, all modules are working.

However, some specific order of operation does exist between modules. The localization component can work without a map engine, but when it needs to find the precise location of the vehicle on the map, it must have the map information from the map engine. The prediction component can not work without the environment information from the perception module and the location from the localization component, it has to make predictions based on the vehicle's precise location and the status information of obstacles. Similarly, the control

component can not work with the planning module, it needs to keep the vehicle on the planned route. In addition, all four components (perception, prediction, planning, control) rely on the information from the localization component. This leads to a problem, since the location information may update in high frequency, if different modules are receiving different versions of the location information, this will raise some errors.
Moreover, prediction and planning components always cooperate with each other. The prediction component will take the planning trajectory of the previous computing cycle from the planning module as input for dynamic adjustment.

Development View

Task Division

Since the software platform has a pipe and filter architecture style, all components are relatively independent. They communicate through pipes with some input and output. Therefore, the implementation of each component can be assigned to an independent team. The input and output formats need to follow the same normative protocol.

Test Plan

Once the program is implemented, it will be tested on the Apollo cloud service platform for the ability to respond to and handle different usage scenarios. If it passes the test, a dedicated test team will conduct realistic scenario road tests. If the software fails the test, the test team will check the information stored in the BlackBox, analyze it and find the component(s) responsible for the failure and then tell the corresponding team for modification. The test team will also upload the failure analysis to the cloud, providing lessons for subsequent systems to avoid making the same mistakes again.

Maintenance

There should be a dedicated team for the maintenance of the software. The team can be a combination of one or more people from each implementation team since they understand their component the best. The modification of each component just needs to maintain the same input and output protocol. They will not affect other components as long as they correctly finish their own job.

System evolution

In Apr 2017, Apollo appeared on the market for the first time. Throughout the period between 2017 and 2022, Apollo has experienced lots of updates and all of them improve the quality of performance of autonomous techniques in various aspects. Apollo 1.0, which is the first update since Apollo is listed, allows vehicles to be tracked in a closed space by plugging a new tracking function. With the 1.0 version updated, vehicle owners will be able to use self-driving technology even in the underground parking area. Next, Apollo evolved to

version 1.5 in just three months. This update is considered to be revolutionary since Apollo optimized the cruise functionality. By combining with Lidar, vehicles with a 1.5 version update can have a better perception of the surroundings and can be tracked more accurately. In this version, the planning, perception, map engine and hardware all got improved. Moving to Apollo 2.0, the update in 2.0 mainly focused on security. It enhanced the camera, Lidar and Radar system to optimize the perception for obstacles on city traffic roads and highways. Also, vehicles with the 2.0 version can change lanes if needed and perceive the traffic light. In April and July of 2018, Apollo published versions 2.5 and 3.0 focusing on high-way autonomous driving and low-speed maintenance in closed areas. Next comes Apollo 3.5, 5.0 and 5.5. These three versions improve the perception module to a higher extent and support driving in more complex and various traffic circumstances. Apollo 6.0 is a huge update. This version realized the laser point cloud obstacle recognition model based on PointPillars and introduced semantic map-based imitative learning for the first time.
The newest version, Apollo 7.0 upgrades the Data pipelining to Apollo studio, optimizes the perception and prediction module by importing MaskPillars、SMOKE、Inter-TNT modules. The cloud platform Apollo studio can help developers to utilize the platform in an easier and more efficient way.

Glossary

LiDAR: Light Detection And Ranging Sensor, determines the distance by pointing a laser at an object and measuring the time it takes for the reflected light to return to the receiver.
ADCTrajectory: Trajectory planned by the planning module
RTK: Real-time Kinematic
IMU: Inertial Measurement Unit
GPS: Global position system
Pipe and filter: an architectural style, made up of a number of pipes (connections) and filters (module).
data flow: data transferred from one module to another.
control flow: a form of communication and process regulation between modules, usually involves function calls cross-module.

modules:
HMI: Human-Machine Interface provides the user with an interface to communicate to Apollo
Map Engine: provides map information.
Localization: module that gives the exact position of the car.
Control: this module controls the mechanical components of the car.
Prediction: this module studies and predicts the behaviour of all the obstacles detected by the perception module
Perception: This module incorporates the capability of using multiple cameras, radars (front and rear) and LiDARs to recognize obstacles and fuse their individual tracks to obtain a final tracklist.
Planning: the module uses logic to routing plans.

Conclusion

After days of diving into GitHub and all kinds of sources, our team has finally come to a conclusion. The conceptual architecture of the software aspects of the Baidu Apollo program is concluded as a pipe and filter architecture that involves seven high-level modules. The modules include perception, prediction, planning, map engine, localization, HMI and control. This architecture is easily maintained and enhanced(by adding more filters), also it allows main modules to run concurrently, which is the most suitable architecture for the software aspects of the Apollo program.

The advantage of our architecture in the software aspects of Apollo includes usability, supportability, concurrent execution which improve performance, maintenance and enhancement; New filters can be added or can replace older filters with improved ones. To further demonstrate, usability indicates that it’s easy to understand the overall structure, inputs and outputs. This architecture also supports deadlock analysis.

However, this architecture has pros and it also comes with cons. It had a poor performance with interactive systems, because of their transformational character. Also, excessive parsing and unparsing leads to loss of performance and increased complexity in writing the filters themselves.

In the future, we will stick to our findings and learn from our newly gained experiences. Then we will develop and demonstrate the concrete architecture of the Apollo, which is the next assignment and report to do. Also, we are going to optimize our sequence diagram for better understanding when we have feedback from other groups.

The Apollo project is a milestone that Baidu has taken another solid step in the system-level opening of artificial intelligence, and it is also the first system-level opening of autonomous driving technology worldwide. Hence we could have the opportunity to study this Apollo autonomous driving platform’s architecture to fulfill our knowledge base and gain more experience.

Lesson Learned

From this report, we expanded our horizons and learned more about software architecture. Have a deeper and more thorough understanding of large-scale software. And we have accumulated more experience in writing reports and drawings sequence and overview diagrams, and we are confident to do better and faster when we write a report next time.

From our team's experience and discussions, we can summarize what we have learned into the following skills: decision making, simplification, communication. Distinguishing the primary and secondary. For instance, don't waste time on unimportant decisions and work, learn to distinguish the primary and secondary tasks. Our team judges whether something is important by conceptual integrity. If we have decided on an idea at the beginning, then we stick to it, even if it is sometimes better to do it differently. This makes the overall concept clearer, improves understandability, and simplifies the difficulty to come up with the architecture for the project.

Decision-making is like prioritization. Some decisions are so critical that, if appropriate solutions are not taken early, there is a high risk of creating unsolvable problems later on. This is bad for us to do the project, team members can't continue to work until this decision is made. In this case, one of our team members had to wait to do the report until the architectural style of Apollo was being confirmed. We could have done this earlier.

Simplification is like divide and conquer. Break the big problem into smaller problems, then solve the small problems separately and verify that the small solutions match. Finally, take a step back and look at the overall situation.

Moreover, we have a deeper understanding of teamwork and communication. Our team leaders have learned how to properly distribute work, and we are allocating work and getting it done on time. We learned to have frequent team meetings to discuss complex and difficult issues. Teamwork allows us to complete team projects more efficiently.

References:

[1] Behere, S., & Törngren, M. (2016). A functional reference architecture for autonomous driving. Information and Software Technology, 73, 136–150. doi:10.1016/j.infsof.2015.12.008

[2] GitHub. 2022. GitHub - ApolloAuto/apollo: An open autonomous driving platform. [online] Available at: <https://github.com/ApolloAuto/apollo> [Accessed 20 February 2022].

[3] Leigh, B. and Duwe, R., 2019. Software Architecture of Autonomous Vehicles. ATZelectronics worldwide, 14(9), pp.48-51.

[4] GitHub. 2022. apollo/README.md at master · ApolloAuto/apollo. [online] Available at: <https://github.com/ApolloAuto/apollo/blob/master/modules/perception/README.md> [Accessed 20 February 2022].

[5] Wang, Y., Jiang, S., Lin, W., Cao, Y., Lin, L., Hu, J., ... & Luo, Q. (2020). A learning-based tune-free control framework for large scale autonomous driving system deployment. arXiv preprint arXiv:2011.04250.

[6] Ma, C., Xue, J., Liu, Y., Yang, J., Li, Y. and Zheng, N., 2018. Data-Driven State-Increment Statistical Model and Its Application in Autonomous Driving. IEEE Transactions on Intelligent Transportation Systems, 19(12), pp.3872-3882.

[7]Ofweek.com. 2022. 1925-2021 “The development history of self-driving autonomous vehicle. [online] Available at: <https://www.ofweek.com/auto/2021-07/ART-8500-7000-30507612.html> [Accessed 20 February 2022].