GROUP 24 Diana Candy: Conceptual Architecture Report of Baidu Apollo

Linus Zuo (18xz44@queensu.ca)

Peiyang Huang (18ph18@queensu.ca)

Shengqi Yuan (18sy52@queensu.ca)

Alex Wang (18tw7@queensu.ca)

Yifan Zhu (18yz153@queensu.ca)

Shen Ye (18ys121@queensu.ca)

CISC 322

Software Architecture

February 19, 2022

Contents

1 Abstract

This is a report on a conceptual architecture of the Apollo 7.0 system, an intelligent driving technology affiliated with Baidu. Apollo is an open, complete and secure software platform provided by Baidu to partners in the automotive industry and in the field of autonomous driving, helping them to quickly build their own complete autonomous driving system by combining vehicles and hardware systems. We can understand that the internal operation and technical implementation of software need to have a framework, which is software architecture. System architecture refers to the concept and architecture of computer system design, describing the design principles that computers are actually doing. Through the analysis of the newly released Apollo 7.0 system, we learned that this framework has a complex system architecture after multiple generations of updates and iterations. The content of the report in this article contains only a limited scope.

Through research, our group believes that Apollo 7.0 implements Pipe and filter and multi-process architecture. This software system could be divided into 5 sub-systems:

1. Perception
2. Strategizing
3. Navigation
4. Execution
5. Human-Computer interaction

The breakdown and explanation of these subsystems will be discussed in further detail within this report.

2 Introduction and Overview

In modern society, autopilot is always a important study topice, there are many solutions and approaches of this topic. Which Baidu Apollo is one of them.

As an autopilot software system, Apollo is designed to deal with complex and changeable road environment. As the second "brain" of the car, Apollo is not only able to design navigation route as traditional navigation system, it is also able to automatically avoid obstacles, make adjustments according to changes in traffic lights, and send and let the vehicle execute the corresponding instructions.

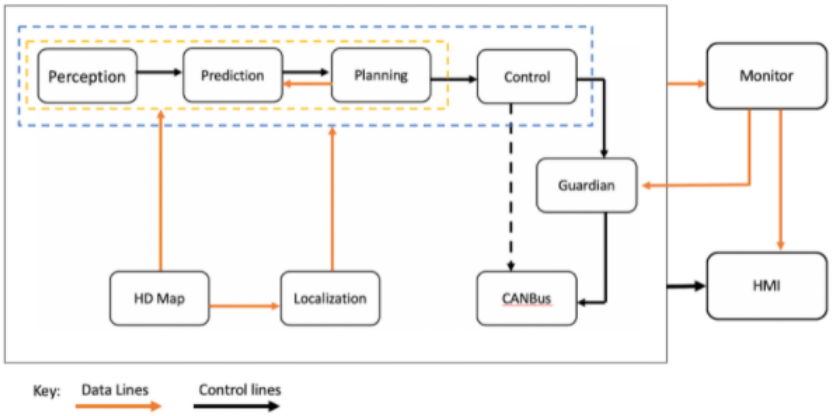
Code, documentation and the evolution of the system can be found in the Apollo's GitHub and Gitee repo. This provides us important opportunity to learn how this system has been developed, and how it work. Through the functional analysis and structure of each sub-module, our team structured the entire software system into 5 subsystems: Perception, Strategizing, Navigation, Execution, and Human-Computer Interaction. These sub-system work together in

a **Pipe-and-Filter Architecture**, at the same time, it also has features from **multi-process architecture** to support the multi-processing requirement of autopilot.

This report, will first provide a brief overview, of each module under different sub-systems. Then we will provide the information of the evolution of this system, the concurrency of this whole system, the way of control data flows. We will also analyse some important use cases, and discuss the implications of resposibility among developers.

3 Architecture

There are many core parts in the Apollo system. This includes Map Engine, Localization, Perception, Prediction, Planning, Control and Human Machine Interface (HMI) in the Open Software Platform.



Navigation Sub-system contains 2 main modules: Map and Localization

The map engine queries the map data from the cloud service and provides those data to Localization and other modules. The map engine is also responsible for collecting data from the environment.

The localization module provides localization services. There are two ways to provide localization services. First, the Global Positioning System and Inertial Measurement Unit work together to provide localization services. Second, The Global Positioning System, Inertial Measurement Unit and Light Detection And Ranging Sensor work together to provide localization services.

Perception contains 1 main module: Perception

The perception module identifies the world surrounding the autonomous vehicle. There are two important submodules inside perception: obstacle detection and traffic light detection. The perception module work with 5 cameras

(2 front, 2 on either side and 1 rear) and 2 radars (front and rear) along with 3 16-line LiDARs (2 rear and 1 front) and 1 128-line LiDAR to recognize obstacles and fuse their individual tracks to obtain a final tracklist. The obstacle detection submodule detects, classifies and tracks obstacles. It also predicts obstacle motion and position information (e.g., heading and velocity). The traffic light submodule detects traffic lights and recognizes traffic lights.

Strategizing contains 2 main modules: Prediction and Planning

The prediction module predicts the future motion trajectories of the perceived obstacles. It works with the obstacles data from the perception module, the localization data from the localization module and the planning trajectory of the previous computing cycle from the planning module.

The planning module will provide a collision-free and comfortable trajectory. It works with the data from the localization module, Perception module, Prediction module, HD Map from the map engine module, routing module and task_manager. The planning module takes the prediction module result. The planning module plans the route based on that information. Also, the planning module takes the output from routing. In some scenarios, the planning module has to recompute a new route to replace the current route. The planning module also requires information about the vehicle, such as location from the localization module.

Excecution Sub-system contains 3 main modules: control, guardian, CanBus

The control module takes the route from the planning module and generates the control command (steering, throttle, brake) to pass to CanBus. Also, the Guardian module will prevent the control from sending the command to CanBus when there is some failure. The Control module can work both in normal and navigation modes.

Human-Computer interaction Sub-system contains 3 modules: control, guardian, CanBus

Human Machine Interface (H.M.I.) is a web application. It visualizes all the information from each module and allows drivers to check the status of all the modules. It takes the messages from Localization, Chassis, Planning, Monitor, Perception, Prediction. It helps developers visualize the output of relevant driving modules. It is a dynamic 3D rendering of the monitored messages in a simulated world. Developers can send some commands to Guardian to stop the command from Control to CanBus.

4 Evolution of the System

The initial version of Apollo implemented autonomous GPS waypoint following. Major features include adding GPS/IMU GNSS driver support, recording and replaying trajectories using HMI, visualizing vehicle trajectories using

DreamView, and adding debugging tools modules/tools/. In the improvement of localization and control, RTK technology is used to improve the accuracy of GPS position measurement in the positioning technology. In terms of control technology, longitudinal calibration table and LQR technology, the earliest and most mature state-space design method in modern control theory, are used. Vehicles with this version do not perceive obstacles in close proximity. Neither can they drive on public roads or areas without GPS signals.

Apollo's version 1.5 began to support vehicles autonomously cruising on fixed lanes. Major features include adding HD Maps Engine API, adding Velodyne 64 lidar driver support and adding new debugging tools. At the same time, add routing, perception, prediction, planning and end-to-end components, where the perception component is 3D point cloud based obstacle detection and tracking with GPU support; the prediction components are deep neural network MLP prediction model and Handling multiple predictors of different classes of obstacles; planning components is traffic law module, multiple iteration functions for DP and QP optimization of path and speed; end-to-end components is Mixed deep neural network models with convolutional LSTM in longitudinal and FCNN in lateral. At the same time, the HMI and DreamView functions released in version 1.0 have been improved to realize the operation of real-time traffic display and traffic scene playback. Vehicles with this version do not detect traffic lights and will not stop at red traffic lights. Neither will they change lanes on the road.

Apollo's version 2.0 supports autonomous driving of vehicles on simple urban roads. The vehicle is able to cruise and avoid collisions with obstacles, stop at traffic lights and change lanes when needed to reach its destination. Major features include adding traffic light detection, adding obstacle classification and supporting obstacle categories: vehicle, pedestrian, cyclist and unknown, adding point cloud-based localization algorithm and RTK fusion, adding MPC-based control algorithm, adding RNN model for traffic prediction, add USB camera and radar driver support, add additional debugging tools. At the same time, upgrade planning capabilities to change lanes to reach destinations, integrate HMI and DreamView, redesign DreamView and upgrade it with additional debug visualization tools. Subsequently, the release docker image upgrade is added via a secure OTA. This version of the vehicle can drive autonomously at low to moderate speeds on simple city roads.

Apollo's version 2.5 allows the vehicle to autonomously run on geo-fenced highways. Vehicles are able to do lane-keeping cruises and avoid collisions with the leading vehicles. Major features include adding HD map data collection tool, adding vision-based perception with obstacle and lane mark detections, adding a relative map to support ACC and lane-keeping for planning and control. At the same time, upgrade localization based on multiple sensor fusion (MSF), upgrade DreamView with more visualization features and make docker files available. Vehicles with this version can drive autonomously on highways at higher speed with limited HD map support. The highway needs to have clear white painted lane marks with minimum curvatures. The performance of vision-based perception will degrade significantly at night or with strong light flares.

Apollo's version 3.0 enables an L4 product-level solution that allows vehicles to drive in a closed venue setting at a low speed. Automakers can now leverage this one-stop solution for autonomous driving without having to customize it on their own. Major features include adding a new safety module called guardian, adding Apollo sensor unit (ASU), a new gatekeeper - ultrasonic sensor. At the same time, version 3.0 enhanced surveillance module, hardware service layer that will now act as a platform and not a product anymore, it is giving developers the flexibility to integrate their own hardware. The perception module got a great improvement, the CIPV detection and Tailgating function is to detect the vehicle in front of the ego-car and estimate its trajectory in order to follow and keep the lane more efficiently when the lane detection is unreliable. Asynchronous sensor fusion Unlike previous versions, Perception in Apollo 3.0 is able to integrate all information and data points by asynchronously fusing LiDAR, radar and camera data. Such conditions allow for more comprehensive data capture and reflect a more realistic sensor environment. Online Pose Estimation This new function estimates the pose of the ego vehicle for each frame. This feature helps understand driving through bumps or slopes on the road with a more accurate 3D scene. Perception in Apollo 3.0 can now be used with the new ultrasonic sensor. Outputs can be used for automatic emergency braking (AEB) and vertical/vertical parking. Vehicles with this version can drive autonomously in complex urban road conditions including both residential and downtown areas.

Apollo's version 3.5 is capable of navigating through complex driving scenarios such as residential and downtown areas. With 360-degree visibility and upgraded perception algorithms to handle the changing conditions of urban roads, the car is more secure and aware. Major features include branding new runtime framework-Apollo CyberRT which is specifically targeted towards autonomous driving, adding new perception algorithms and new localization algorithms, adding new scenario-based planning with a new planning algorithm, open space planner. Vehicles with this version can drive autonomously in complex urban road conditions including both residential and downtown areas.

Apollo's version 5.0 is an effort to support volume production for Geo-Fenced Autonomous Driving. The car now has 360-degree visibility, along with an upgraded perception deep learning model to handle the changing conditions of complex road scenarios, making the car more secure and aware. Scenario-based planning has been enhanced to support additional scenarios like pullovers and crossing bare intersections. Major features include adding new perception algorithms again, adding sensor calibration service, adding map data verification tools, prediction evaluators and Apollo synthetic data set. At the same time, version 5.0 is also a brand new data pipeline service, including vehicle calibration. Upgrading scenario-based planning with a new planning algorithm, open space planner and new scenarios supported, include intersection - stop sign, traffic light, bare intersection and part - valet, pull over. Vehicles with this version can drive autonomously in complex urban road conditions including both residential and downtown areas.

Apollo's version 5.5 enhances the complex urban road autonomous driving

capabilities of previous Apollo releases, by introducing curb-to-curb driving support. With this new addition, Apollo is now a leap closer to fully autonomous urban road driving. The car has complete 360-degree visibility, along with an upgraded perception deep learning model a brand new prediction model to handle the changing conditions of the complex road and junction scenarios, making the car more secure and aware. New Planning scenarios have been introduced to support curb-side functionality. Major features include adding a brand new storytelling module, Scenario-based planning with a new planning scenario to support curb-to-curb driving, also add a prediction model including semantic LSTM evaluator and extrapolation predictor. Control module improved by update model reference adaptive control (MRAC) and control profiling service. Vehicles with version 5.5 installed have the same driving abilities as version 5.0.

Apollo's version 6.0 incorporates new deep learning models to enhance the capabilities for certain Apollo modules. This version works seamlessly with new additions of data pipeline services to better serve Apollo developers. Apollo 6.0 is also the first version to integrate certain features as a demonstration of our continuous exploration and experimentation efforts towards driverless technology. Major features include adding brand new data pipeline services that low-speed obstacle prediction model training service with semantic map support, PointPillars based obstacle detection model training service, Control profiling service, Vehicle dynamic model training service, Open space planner profiling service, Complete control parameter auto-tune service were added. The new version also upgraded deep learning models including PointPillars based obstacle detection model, semantic map-based pedestrian prediction model, learning-based trajectory planning model. Version also supports driverless research, which includes a remote control interface with DreamView integration and audio based emergency vehicle detection system.

Apollo's version 7.0 incorporates 3 brand-new deep learning models to enhance the capabilities for Apollo Perception and Prediction modules. Apollo Studio is introduced in this version, combined with Data Pipeline, to provide a one-stop online development platform to better serve Apollo developers. Apollo 7.0 also publishes the PnC reinforcement learning model training and simulation evaluation service based on previous simulation services. Major features include adding brand new deep learning models, that Mask-Pillars obstacle detection model based on PointPillars, Inter-TNT prediction model based on interactive prediction & planning evaluator and Camera obstacle detection model based on SMOKE were included.

5 Concurrency

The idea of concurrency is ubiquitous in the Apollo scheduling system. Because autonomous driving and human safety are so closely related, Apollo's scheduling system must be very practical and not allow asynchronous tasks to preempt it. In Apollo, a mechanism like Co-routine is introduced.

The two main scheduling strategies used by Apollo are the classic strategy

and the choreography strategy. By default, Apollo uses the classic policy, and the scheduling model uses a multi-priority queue, which means that tasks of the same priority are in the same queue, and the system will prioritize the tasks with higher priority. On the contrary, the choreography policy can analyze the priority relationship at the beginning and configure different tasks to run on different processors. The concurrency here is reflected in the fact that tasks of the same nature are placed on the same processor as much as possible to get the best results (e.g. high frequency & short time-consuming tasks).

Perception is one of the most important parts of Apollo autonomous driving, which will use the vehicle's external sensors including LIDAR, cameras, Radar millimetre wave, etc. to sense the surrounding environment. Concurrency of which has always been reflected. Obstacle perception and traffic light perception are the two main parts of the Apollo Perception system. Among them, concurrency in obstacle perception is reflected in the simultaneous management and correlation of obstacle results from different sensors, and finally the integration of obstacle velocities. Even though there are no obvious concurrency threads in stoplight perception, obstacle perception and stoplight perception need to run concurrently to ensure that the vehicle can be operated properly.

6 Controls and Data Flows

As mentioned earlier, Apollo is a pipe-and-filter architecture with features from multi-process architecture, which natively supports the communication between separate processes.

Most sensor data, such as camera data, lidar data, microphone data, etc., is collected by the perception component. After the driver component has this data collected, it will be transmitted to different components for their use. The apollo system provides three different ways for components to communicate:

1. topic: topic is the basic communication method. It provides plenty of topic flags, able to fit communication needs between all components.

2. story: the story is a high-level communication method, it is able to create complex scenarios for multiple components. And the component outside the story is also able to get this information by subscribe.

3. bridge: bridge module provides a way for apollo components to communicate with 3rd party components by using UDP socket.

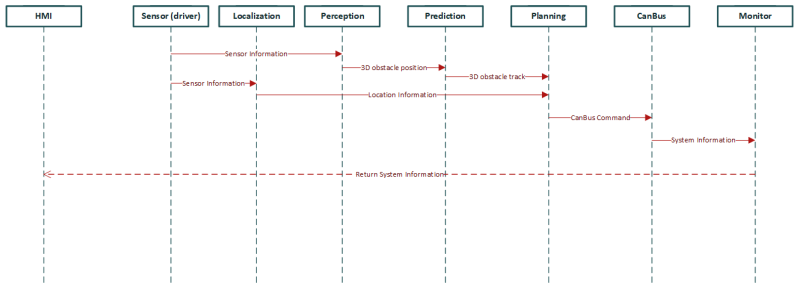
Since this is a pipe-and-filter architecture, the data collected by the sensor will flow through different components and be processed as a control command. After the data is collected by the perception component, it will be sent to the prediction component together with the map data from the map component. The prediction component will calculate the collision prediction information, after the calculation, the collision prediction information will be sent to the planning component for route planning.

The planning component will collect data from multiple sources, including localization, perception, map, prediction, routing, task_manager. The planning component will handle and calculate all this information, and specify the navigation route (from the route component) and obstacle avoidance plan (from the prediction component). Then the route and plan will be sent to CanBus as a command, these commands might go through the guardian component for security purposes. Finally, the CanBus component will execute the command to make the vehicle drive correctly.

There are also components provides the external interface for the third party software to communicate with the Apollo system. One of the important components is the bridge mentioned above. The bridge module use UDP socket, which is a great idea for the autopilot, since compare to TCP, UDP can provides shorter latency, which is the key of autopiloting.

7 Use Cases

1. Automatic obstacle avoidance:



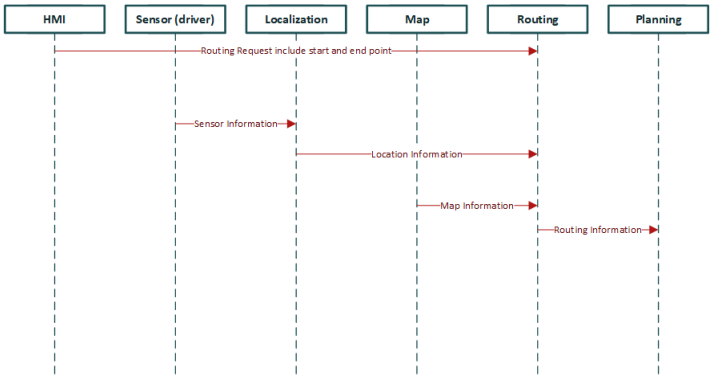
For the use case of automatic obstacle avoidance use case, it uses the following steps which will fit the pipe-and-filter architecture.

Firstly, the native sensor information will be collected by the driver module, and it will be transmitted to the perception module for processing. The perception module will process the native sensor information and generate the 3D obstacle information (by the sensor) data.

Then the 3D obstacle information data will be transmitted into the prediction model. In prediction model will analyze the 3D obstacle information (by the sensor), and calculate the prediction. When the prediction module has done its part, it will generate the 3D obstacle track. This information will be sent to the planning module.

The planning module will collect data from multiple different modules, in the obstacle avoidance use case, it will collect the 3D obstacle track from prediction and car's location information from the localization module. The planning module will collect this information and calculate how the vehicle will travel next as the CanBus command. This command will be transmitted to the CanBus module and be executed by it.

2. Navigation route planning:



The use case of navigation route planning use case uses the following steps which will fit the pipe-and-filter architecture.

In the beginning, the navigation request will be sent from the H.M.I. module by the user, after that the routing module will begin to wait for the information of location and map.

The location information will be first collected by the sensor (driver module), after that, this information will be sent to the localization module. The localization module will calculate the location data from the sensor data, after it has done the calculation, the location data will be sent to the routing module.

On the other hand, the map will send its information to the routing module together with the location data calculated by the location module. After this data is collected, the routing module will begin to calculate the navigation route plan. After the routing module is done, this information will be sent to the planning module and wait for planning and execution.

8 Implications of Responsibility Division of Developers

When facing the Pipe and filter architecture, we know that different architectures will make developers face different task assignments. We know that the Pipe and filter architecture needs to add a large number of independent filters, which may be reduced performance due to excessive computational overhead. but auto-driving systems need to maintain computing performance to ensure the safety of users, so in the division of responsibilities, it is necessary to call out redundant manpower to optimize performance and ensure the quality of operation. The second point is that the interactive transformation of the pipe architecture is difficult because a large number of filters are required, so the filter must be considered as an independent individual for analysis, so the development team needs more personnel to be responsible for the transformation.

In addition, in the code analysis of Apollo 7.0, we found that the number of modules is large (because of its pipe and filter module), so if an error occurs in a certain link, a chain reaction is prone to occur, and a large number of errors occur. Not only to the bad side, we also need to know the advantages, this structure is easy to maintain and enhance, which is convenient for developers to reduce manpower and time in maintenance.

The current apollo version has hundreds of GitHub main contributors and a large number of apollo associated developers, and there are more than 4,000 developers discussing in the official statistics of the cooperation group. After joining the developer's discussion group, we learned that the management of the office staff is vetted and verified to ensure that no malicious changes are made by malicious people. Officials said that they have established reviewers separately to review the uploaded code and the code submitted by the people who are required to cooperate and the company, and will conduct independent operations for review.

9 Conclusion

Although our architecture is not complex, we think we have finished structuring Apollo System's main functions. As the "brain" of this system. The Perception will listen and collect the data from sensor, together with the information from Navigation sub-system these information will be handled by Strategizing sub-system. The Strategizing sub-system will send command and the Execution sub-system will handle and execute it. The Human-Computer Interaction sub-system provide the interface for human driver users to control and monitor the whole system.

These subsystems work in together is Pipe-and-Filter order to provide safe and stable autonomous driving functions for Apollo System.

10 What we learned?

A autopilot system might seems complex, but it is able to be deconstructed into multiple sub-systems for easier analysis. Software systems are not static, it sub-systems, sub-modules might change from time to time, there might be new modules added, new sub-system added, some modules and subsystems may be removed or integrated.

At the same time, a complex software system does not necessarily follow a single software structure, and may also have the characteristics of other software structures on top of a main software structure.

11 Reference

1. Apollo Development Team. 2021. Baidu Apollo System. [Source Code] https://github.com/ApolloAuto/apollo
2. Apollo Development Team. 2021. Baidu Apollo System. [Changelogs] https://github.com/ApolloAuto/apollo
3. Apollo Development Team. 2021. Baidu Apollo System. [Documentations] https://github.com/ApolloAuto/apollo