Conceptual Architecture report on Apollo

Minxuan Li 20144519 18ml51@queensu.ca
Li Bowen 20144417 18bl29@queensu.ca
Steven Wen 20144322 18yw85@queensu.ca
Yuehan Qi 20167259 18yq36@queensu.ca
Aolin Zhou 20058020 16az31@queensu.ca
Queen’s University
CISC322
Instructor: Prof. Bram Adams

Table of Content

1. Abstract

In this paper, we researched and searched for information on Baidu Apollo Autopilot in this study. We aim to understand Baidu Apollo Auto driving technology's functionality, software system, data control, and system concurrency and understand it better with some examples and sequence diagrams. In order to gain a comprehensive and accurate understanding of the Apollo project, we use survey and literature research methods. Eventually, through relevant information and group discussions, we came up with the architecture of Apollo.

2. Introduction and overview

We divide the introduction into three aspects to have a better overview of the whole project.

Background and purpose

At the 1939 World's Fair, General Motors created the first self-driving car model. It was an electric vehicle guided by radio-controlled electromagnetic fields and operated from magnetized metal spikes embedded in the roadway. This model turned into a reality in 1958. Since then, more car companies have wanted to develop their system to relax the driving experience. Nowadays, traditional car companies are developing automatic driving vehicles, but some tech companies also want to be involved. And so does Baidu. In 2017, they announced "Apollo," an open-source platform to help car industries build their autonomous driving system by combining their hardware with "Apollo ." Apollo comes from the "Apollo program," wishing it could be a milestone for the self-driving industry.

Apollo's environment perception, path planning, vehicle control, and in-vehicle operating system are the core modules that rely on each other. The developer needs to configure all the modules before testing the vehicle. As for now, Apollo has released its 7.0 version, which has three brand new deep learning models to enhance the capabilities for Apollo Perception and Prediction modules which helps it drive on complex urban roads.

Organization of the paper

The paper is mainly focused on the architecture of Apollo and the architecture is divided into six parts.

Beginning with the functionalities, there are ten main functionalities with the Apollo autonomous driving system: perception, prediction, planning, control, HD Map, localization, Guardian, CANbus, Monitor, and HMI. Monitor and HMI interact with all other modules via publish/subscribe style. The perception, prediction, and planning modules closely communicate in control and data flow. The HD Map and localization act as support modules to transfer information to the perception, prediction, and planning modules.

When finding the history and evolution of the Apollo, the project started in 2017 and updated for many improved functionalities and added features, and through 11 iterations the latest version was Apollo 7.0, the milestone for Apollo.

During the development process, the Apollo utilizes the robot operating system as a framework. This framework provides the convenience of easy accessing sensor data for each node, processing data, and generating the response to communicate to each other. However, ROT underlies some defects that may affect the performance and latency of the autonomous driving vehicle, such as single point failure. Thus, Apollo improved it by making three main areas of change, ROS Decentralization Feature, Communication Performance Optimization, and Data Compatibility Extension.

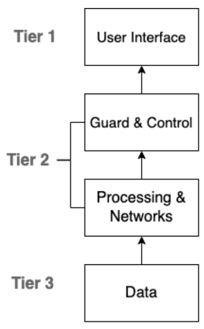
When discussing concurrency, there are two main strategies that Cyber RT can use are orchestration mode and the classic mode. The orchestration takes a distinct feature of autonomous driving tasks as the foundation. When applying the orchestration strategy, Systems are pre-configured which CPU each job executes on. Once the task scheduling is complete, it only needs to be executed in sequence. This can reduce the complexity of thread switching and the cost of Cache Miss during the system's unpredictability. The classic strategy applies the first-in-first-out mechanism. Cyber RT implements a multi-priority queue mechanism to support the priority distinction of tasks, with work assigned to a particular queue based on importance.

After concurrency we discussed four distinct responsibilities in the project design and decision-making process. Users are the first to submit requirements and feedback. The second is a software developer, a contributor who oversees calculating a task's cost and deadline. Committers are the third group, and they are involved in design discussions and code quality. The Project Management Committee is the last one, and it makes decisions.

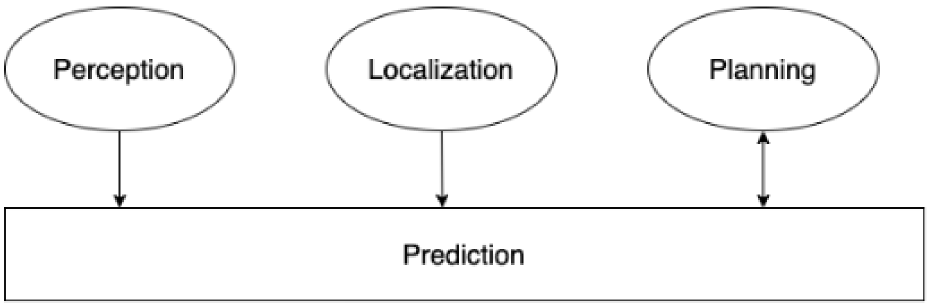
Finally, the report discusses two types of sequence diagrams: the general case of Apollo smart transportation and Apollo's traffic light perception. The planning module will be sending the plan to the control module based on the data. The control module will operate the CANbus. The processing module will take in the pre-processed data and push out the result for the planning module. Then the perception module's output will be taken into the prediction module, which will subscribe to the location and planning information.

High-level architecture and subsystem

First, there are three tiers in the Apollo software system. The topmost tier, the presentation tier, is where the user interface translates tasks and feedback to users so that they can easily understand. It sends signals to the tier below it. Then Tier 2 interprets the signals as commands to process. The 2nd tier consists of two layers. Each of them contributes to the logic-related process of the system. Tier 3 is the data tier where the information about the system is produced, passed to its upper-tier for processing, and eventually returned to the user.



The processing system suits the pub-sub style, in which the prediction module subscribes to localization, planning and perception obstacle messages, as shown below.



3. Derivation process

During the progress, the assignment 0, each of our members has assigned a specific module to research and collect information. Even though we spent a lot of time learning and studying the Apollo autonomous driving system, we still did not realize the core constitutions of software architecture regarding this system. It was not until we discussed and derived a picture of its conceptual architecture. The Apollo autonomous driving is an open-source platform designed and developed by Baidu developers based on a system with the central Apollo system at the upper level, the middleware at the middle level, and the Linux operating system at the lower level. The higher-order architecture we came up with is as shown in figure 1, where the higher modules depend on the modules below it.

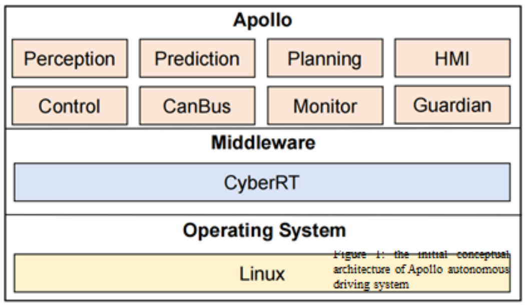


Figure 1: the initial conceptual architecture of Apollo autonomous driving system

This layered architecture did make sense to us until we encountered a more comprehensive overview of what higher-level architecture would look like in figure 2. As shown in figure 2, there are ten main modules in this architecture, each module is responsible for a specific and critical task, and the modules closer to the HMI depend on the modules closer to the perception. Since these modules are within the same architecture level, we thought that each depends on the other by receiving the input and producing the output to another module. This corresponds to the style of pipe and filter. For example, the prediction module can serve as a filter, and its connections between perception and planning can serve as pipes. The prediction module processes input that it received and performs transformations on data. The connector pipes connected components next to each other. This architecture provides some flexibility and advantages suited to the Apollo autonomous driving system. First, it provides loose coupling of components or filters. Secondly, it is in benefit of concurrent processing. However, after we thoroughly reconsidered its disadvantage, we thought it might not be suited to the Apollo. Since its disadvantage of reduced performance during the long-running task is fatal, especially in an autonomous driving system, various modules such as perception and prediction have to be operating all the time to keep the vehicle under control. Therefore, we give up on this style and declare another architecture style of publish-subscribe. It was inspired by how the Robot Operating System functions. Each module in the Apollo Autonomous Operating System is instantiated as a node and runs on the ROS middleware in a publish-subscribe fashion. Consequently, we came to an architecture style of eclectic layered and publish-subscribe style for the Apollo autonomous driving system.

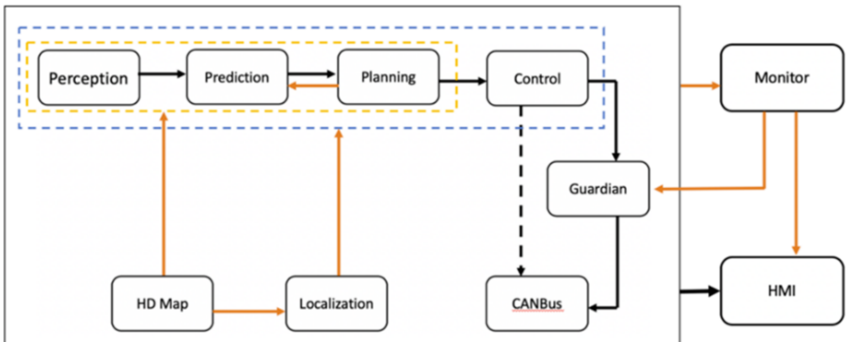


Figure 2: the overview of the Apollo conceptual architecture

4. Architecture

4.1 System Functionality

There are 10 modules in the system of Apollo, in which each of them represent a functional view.

- Perception

The perception module works on the external environment where the vehicle operates with accuracy. It will detect the obstacles in the surrounding environment and recognize the traffic lights. Then it will output the tracking of obstacles heading, velocity and classification information for prediction. Three new deep learning models enhance the perception and prediction modules in the recently updated Apollo 7.0.

- Prediction

Receiving from the obstacles perceived by the perception module, the prediction module will process it then give an estimate of possible future trajectory movements. The localization module also contributes to the prediction module's status updates. Whenever a localization is updated, a signal is sent to the prediction module. This triggers an internal update.

- HD map

It provides information for the perception, prediction, and planning flow and transports data to the Localization component.

- Localization

Works for the perception, prediction, planning and control flow. It is responsible for determining the vehicle's location concerning a global map. With various inputs from HD Map, GPS and sensors, the localization system can provide positioning solutions comprehensively with centimeter level accuracy.

- Planning

Further movements are predicted, trajectory generation and motion logic is applied. Nearly every module in the architecture is involved. Each driving use case is treated as a particular driving scenario based on road conditions in the planning module, and it is responsible for providing a safe and collision-free trajectory for the control module to execute. In the newest version, the planning module adds a “three-point turn” dead-end scenario, which increases the capabilities of driving in and out and expands the operation boundary of the urban road network.

- Control

The control module takes in the planning trajectory from the planning module. Then it generates control commands like steering, throttle, and brake to CANbus.

- CANbus

The CANbus module takes control commands and sends the vehicle’s detailed chassis status back to control.

- Guardian

The guardian module works as an if-else statement between the control module and the CANbus module to prevent the car from an emergency when a module goes down. If all modules work accurately, the guardian will process all the data coming from the control module to CANbus. Otherwise, when the monitor detects a module crash, it will send signals to the guardian to intercept the control signals reaching CANbus so that the car would stop.

- Monitor

The monitor module surveys the working condition of the system. It receives data from multiple modules and checks if they all work without any issue. Then it passes on the information to HMI for the driver to view.

- HMI

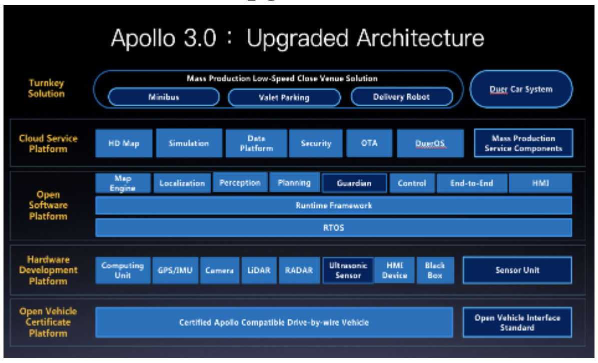
The HMI module outputs the visualization of relevant driving modules. It displays the hardware status of the car and the switch to turn on/off modules to the driver.

4.2 System Evolution

Because there are many versions of the history of Apollo from its beginnings to the present, we will discuss only the four most important to its development in this paper.

Apollo 3.0: production level closed venue AD

In Apollo 2.5, the purpose is to allow vehicles to drive autonomously through highways. However, the primary aim in Apollo 3.0 focuses on maintaining lane control, driving, and avoiding collisions with vehicles on the road in a low-speed environment, and the difference and upgrade between 2.5 and 3.0 are based on this aspect.



In Apollo 3.0, the system adds more modules and upgrades the old module.

First, we will introduce the new module Guardian. Due to the more complex road conditions in the city, driving safety issues need more attention. A new module called Guardian has launched a primary focus on safety, and it ensures the vehicle's safety by carefully bringing the vehicle to a stop during emergencies.

The monitor is a surveillance system and works closely with the Guardian. If the monitor detects any software malfunction, it sends in an alert to The human driver is alerted to take over by a guardian and an auditory and visual warning. Guardian can bring the vehicle to a safe halt(gradual barking).

It also executes hard brakes under two scenarios:

1. Ultrasonic sensor malfunction: Guardian constantly checks with an ultrasonic sensor. If the ultrasonic sensor fails to respond or malfunctions, Guardian sends a hard brake signal to CANbus to avoid a possible collision, regardless of the vehicle's speed.
2. The human driver failed to take control of the vehicle: as previously said, when the monitor identifies any possible hazard, it works with HMI to send a visual and aural warning to the human driver, in addition to requesting Guardian to execute a soft stop. The Guardian would instruct a hard brake if the human driver failed to stop for a brief period of time。

Secondly, we will talk about the update for Perception. The perception module introduced a few features to provide more diverse functionalities and a more reliable, robust perception in AV performance in Apollo 3.0.

CIPV detection: This feature enables the ego-car to detect the vehicle before it and estimate its trajectory. When the lane detection feature is inaccurate, this results in more efficient tailgating and lane keeping.

Asynchronous sensor fusion: By asynchronously fusing LiDAR, radar, and camera data, Perception can condense all the information and data points. As a result, more comprehensive and dynamic modes are possible, as well as more realistic sensor environments.

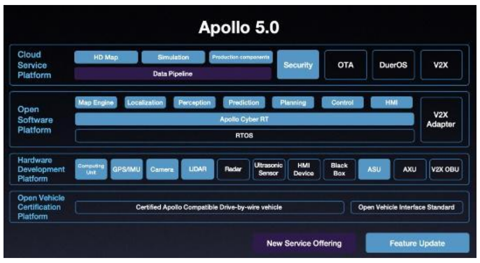
Online pose estimation: This feature can estimate the pose of an ego-car for every single frame, which helps to drive through bumps and slopes with more accurate 3D scene understanding.

Ultrasonic sensors: Perception now works with ultrasonic sensors. The output can be used for AEB and vertical/perpendicular parking.

Complete lane line detection: This feature provides more accurate and lone-range lane line detection. The benefit of this inclusion is that it makes lane-keeping much safer and provides more accurate predictions.

Apollo 5.0: AD empowering production

In Apollo 5.0, there are several newly released features such as a Smart data recorder, Apollo data pipeline, and large-scale cloud processing. Apart from all these interesting new service offerings and technologies, Apollo 5.0 also included important software system enhancements in the Perception, Prediction, and Planning modules.

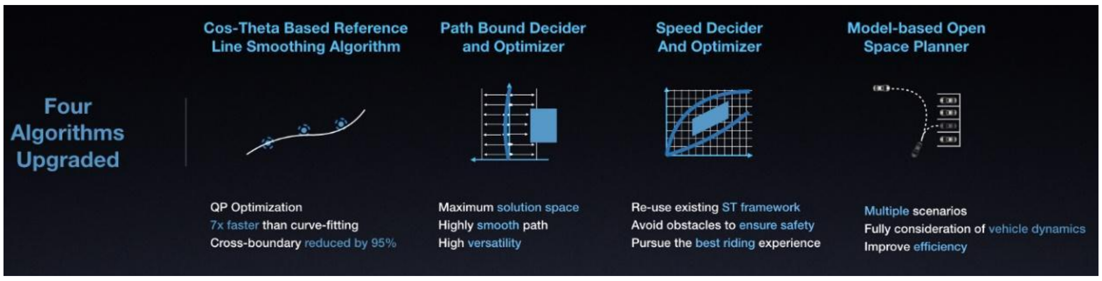


Perception: The Apollo 5.0 apollo perception module incorporates multiple cameras, radars and LiDARs to detect obstacles and fuse their tracks to obtain a final trajectory. Utilizing the ego-motion of the self-driving vehicle, this sub-module estimates the trace of obstacles, including a closest-in-path obstacle. From a deep neural network, the detector can detect each instance of lane lines, classify numerous lane kinds, and recognize fork and split lane lines.

In addition, the Apollo team created an offline camera calibration tool that allows the community to quickly adjust the orientation of cameras with a single button press.

Prediction: The upgraded prediction module can now model-building to different types of obstacles. Road conditions are also captured and assessed as part of the modeling process. Semantic maps are created to help create models like behavior prediction, pat prediction, and so on by adding road features to the deep learning network. Simultaneously, these models are optimized based on priority, with a more elaborate model being employed to deal with higher-priority impediments. The model scheduling framework will automatically select the most appropriate model to pursue the best prediction outcomes possible once such hierarchical architecture has passed the scene analysis, intent understanding, and job prioritizing.

Planning: The four fundamental algorithms in Apollo 5.0 have been upgraded significantly.


Four Algorithms Upgraded
Cos-Theta Based Reference Line Smoothing Algorithm
QP Optimization
7x faster than curve-fitting
Cross-boundary reduced by 95%
Path Bound Decider and Optimizer
Maximum solution space
Highly smooth path
High versatility
Speed Decider And Optimizer
Re-use existing ST framework
Avoid obstacles to ensure safety
Pursue the best riding experience
Model-based Open Space Planner
Multiple scenarios
Fully consideration of vehicle dynamics
Improve efficiency

To optimize trajectory problems, the planning module converts the world coordinate system into a reference-line coordinate system. A smooth reference line is therefore crucial to optimizing trajectory problems. Apollo 5.0 uses a discrete-point, optimization-based reference-line smoothing algorithm that replaces the previous curve fitting method, which is seven times faster and results in a more consistent smoothing effect. Furthermore, the new algorithm is highly versatile and can be used to solve the path planning problem for many different driving scenarios, ensuring that crossing the boundary was reduced by 95%.

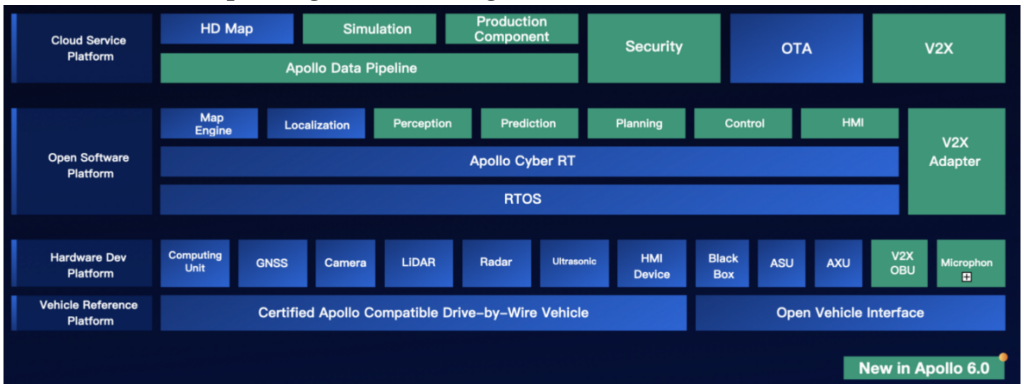
Apollo 6.0: towards fully driverless technology

Apollo 6.0 incorporates new deep learning models to enhance the capabilities for specific Apollo modules, including the following three ones:

Perception**:** Point Pillars model for object detection from LiDAR point clouds data, optimized by TensorRT;

Prediction**:** Semantic map based pedestrian prediction model to fit the pedestrian behavior in the environment better;

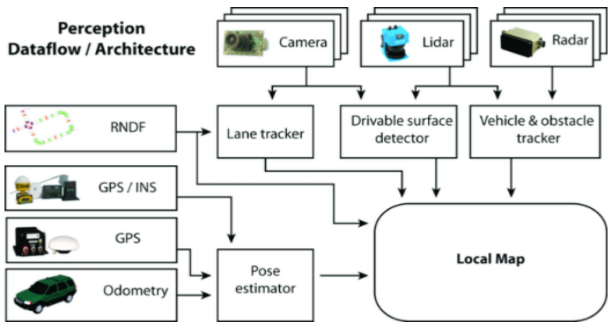
Planning**:** Learning-based trajectory planning model to imitate driver behaviors in side-passing slow-moving obstacles.



Apollo 7.0

Apollo 7.0 represents a milestone in the evolution of Apollo from code to tool and from open-source platform to tool platform. The new version of Apollo 7.0 provides a series of upgrades based on the four open-source platforms, which include cloud service, open-source software, hardware development, and vehicle authentication. Upgrades include a one-stop cloud platform, Apollo Studio, a leading simulation service, and highly efficient new models.

4.3 Control and data flow


Perception Dataflow / Architecture
Camera
Lidar
Radar
RNDF
GPS / INS
GPS
Odometry
Lane tracker
Drivable surface detector
Vehicle & obstacle tracker
Pose estimator
Local Map

By the CANbus, the planning engine controls The Apollo Autonomous vehicles dynamics. This Apollo AV system depends on different hardware such as LiDAR, cameras, radar sensors, and Global Navigation Satellite System to retrieve much real-time information on the road. This information could be pedestrians, moving vehicles, obstacles, road conditions. Upon receiving the data, the ADS utilizes various trained machine learning models to assist in the decision-making of real-time driving. For the sake of high-precision computation, different modules of Localization, Perception and Planning modules function as independent input sources, while the output sources are working together as an architecture style of peer-to-peer and supported by RPC network application.

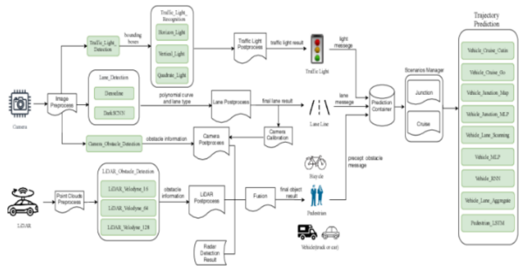
The ROS is a set of open-source software libraries and tools for writing robotic applications. The Apollo ADS is based on ROS to improve its performance and stability: ROS Decentralization Feature, Communication Performance Optimization, and Data Compatibility Extension. In the ROS, there are a series of individual nodes communicating by a publish or subscribe messaging model. When the sensors collect the real-time data, the sensor driver can be implemented as a node to transfer data in published message form to other nodes of filter logging systems. The ROS also provides a master node to enable inter-communication between each node.

Since the ROS system contains a primary node to enable other nodes to communicate, communication failure may be risky when the controller node is down. Moreover, this system structure lacks a recovery mechanism. Therefore, Apollo removes the centralized network structure and implements a complete peer-to-peer network structure using the RTPS service discovery protocol.

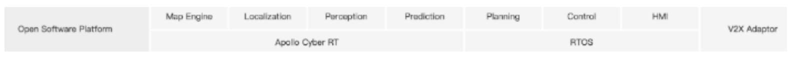
After an upgrade of the interface, different module versions may be incompatible, and the Apollo ADS makes another change to the ROS to improve this. The data format of the ROS changes to Google's Protocol Buffers format data to address this. Protocol Buffers is a free and open-source, cross-platform data format used for serializing data. Due to its data storage and communication protocol, it resolves the disadvantage of incompatibility of modules.

Nodes can communicate with each other to publish and subscribe to Topics. Camera modules, for instance, capture data and transmit it to different modules in the system. While some modules are registering to this camera module, the camera module can publish its data to its subscriber in a publish and subscribe manner.

Many sensors collect a lot of data at an extremely high frequency in a real-time autonomous driving environment that could pose a considerable pressure of processing the data. Therefore, high data transmission efficiency might be required. The Apollo ADS solves this by introducing a shared memory technique, which reduces data copies and improves communication performance of the latency and throughput between nodes.



4.4 Concurrency



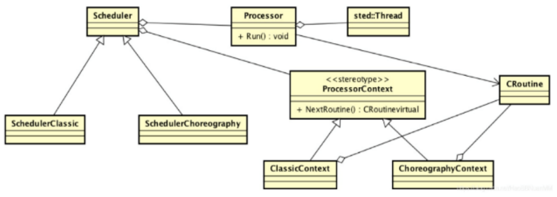
Concurrency arises when several process threads in the operating system execute multiple instructions simultaneously. The threads of a running process constantly communicate via shared memory or message forwarding. Concurrency causes resource sharing, leading to deadlocks and resource scarcity issues. It aids with process coordination, memory allocation, and execution schedule to maximize throughput. Through the Apollo Open Platform, the concurrency can be found in the Apollo Cyber RT framework.

The Apollo Cyber RT framework is the world's first open-source, high-performance runtime framework intended exclusively for autonomous driving technology development. The release of the Apollo 3.5 open-source platform was initially introduced. The Apollo Cyber RT is a centralized and parallel computing system that allows high task concurrency, low latency, and high throughput. It is the result of years of development work to design a framework that fulfils the strictest performance criteria for autonomous driving systems. There are several modules in Apollo Cyber RT. The focus is on the scheduling framework. The scheduling framework shields the underlying details of the operating system and no longer reflects the concept of threads to the application layer but encapsulates it as a Processor and combines the use of coroutines to implement task scheduling and switch in user space.

Cyber RT provides a specialized scheduling system to fulfill real-time performance requirements. The scheduling system extracts the scheduling logic from kernel space to user space, and coroutines are used as the fundamental scheduling unit. In the classic multithreading approach, each thread corresponds to a CPU core, corresponding to the user's calculation. The scheduling and switching off the thread depend on the scheduling algorithm of the Kernel, and the controllability is low. During the scheduling of user space, each coroutine corresponds to the corresponding task. At the same time, coroutine corresponds to the Processor and switching of a task by the Processor happens entirely in the user space and is controlled by a specific scheduler.

For each task, Cyber RT is abstracted into individual tasks that are created corresponding to a coroutine, and each coroutine has a set of independent context information, including stack and register information. After each task is created, it is transferred to the scheduler and assigned to the corresponding Processor. Moreover, each Processor corresponds to an actual thread of the operating system. Usually, processors and the logical cores of the CPU are in one-to-one correspondence to avoid frequent migration of tasks between different CPUs during running, which will not cause excessive context switching and Cache Miss. Finally, the scheduling strategy will decide the underlying Processor division and which task should be assigned.

There are two main strategies that Cyber RT can use to schedule the tasks, which are orchestration mode and the classic mode. Compared to standard server-side apps, autonomous driving tasks have a distinct feature that most tasks run periodically. In this situation, the scheduler will be aware of which jobs must be accomplished at specific times and in what sequence. This feature is the foundation of the orchestration approach. It is pre-configured which CPU each job executes on and the sequential relationship between processes through analyzing the specific business of the system. For example, tasks that are not interdependent are distributed over different CPUs, and tasks with upstream and downstream interactions are distributed over the same CPU to improve parallelism. Once the task scheduling is complete, it simply has to be done in order, which reduces runtime dynamics. Under the orchestration technique, each task is connected to a fixed Processor that binds to the CPU logic core, reducing the complexity of thread switching, the cost of Cache Miss during runtime, and the system's unpredictability. Sometimes, some high-priority tasks may be blocked in the system. Cyber RT sets aside specific reserved Processors that are scheduled when a temporary high-priority task appears to address this issue. Due to the bottom layer of these Processors being Real-time threads, user space task preemption can be achieved to ensure the execution of real-time tasks. That is the primary strategy that Cyber Rt uses.



Another strategy is the classic strategy. Compared with the orchestration strategy, the logic of the classic strategy is more like the traditional thread pool mode in that each Processor does not have a separate task queue, and tasks are not fixed on a specific Processor. The classic strategy applies the first-in-first-out mechanism. Under this strategy, Cyber RT implements a multi-priority queue mechanism to support the priority distinction of tasks whose work is assigned to a particular queue based on its importance. In this way, it can reduce the concurrency bottleneck caused by a single queue, ensuring the priority execution of high-priority tasks.

This is how concurrency applies to the Cyber RT, which is the Apollo framework. During the process, we can see different strategies track the execution of a program. As mentioned, there are two main strategies that Cyber RT can use to schedule the tasks and how do these two strategies allow the program to run rationally. Although these strategies are well-used in the Apollo framework, latent strategies are still more optimal than existing ones. The enhancement of these strategies will hopefully appear in the future.

4.5 Division of Responsibilities

The key to division responsibilities is to ensure that the critical stakeholders within the business will be involved in the software solution delivery. Since there are many stakeholders involved, we will discuss the most significant ones in this paper. As a meritocratic community project, Apollo has four roles that operate together to design and decision-making. Each of them shares responsibility for the project developments.

- **Users**

Users are a significant part that generate needs and provide feedback for the project development. In Apollo's projects, user acceptance testers can suggest changes to make it easier for users to operate. Their role is critical because developers develop based on their understanding, and by doing so, user acceptance testers reduce the friction that can exist when end-users move away from an existing software system that they know, love, and use daily.

- **Contributors**

Everyone that contributes to the projects in a certain way are considered as contributors. Among all kinds of contributors, the software developers are in charge of estimating costs and timelines based on the technical requirements provided by the technical lead. They're also in charge of creating deliverables and keeping track of the software project's progress. If they fail to do their tasks, they may put the project at risk.

- **Committers**

Committers are contributors who have shown their commitments to the continued development of the project. Among all the ongoing engagement with the community, they mainly participate in design discussions and control the overall code quality of the projects.

- **PMC (Project Management Committee)**

The Project Management Committee oversees the Apollo community as the core management team. They oversee making decisions when community consensus cannot be met. The PMCs have additional responsibilities over and above those of Committers, which ensure the project runs smoothly. Handling the reported security issues like CVE is part of their jobs. They also deal with major changes such as feature proposals and organization or process changes. As the top level in the hierarchy of stakeholders, the project manager is responsible for knowing the "who, what, where, when, and why" of a software project, which means knowing the stakeholders of the project and communicating effectively with them. If the project manager fails to do so in a specific Apollo project, project members may not know their specific tasks and the project's completion is delayed.

4.6 Sequence Diagram Explanation

Based on the presented conceptual architecture, our team has developed two sequence diagrams. The first one is the general case of Apollo smart transportation, and the second is Apollo's traffic light perception.

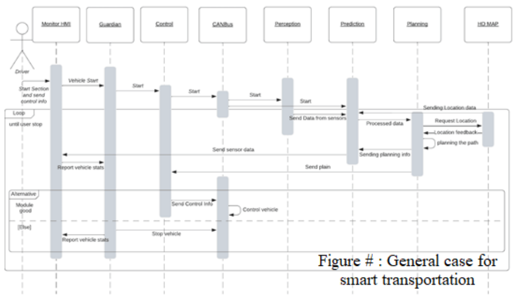


Figure # : General case for smart transportation

After the user starts the system and inputs destination information, the system will be activated. The first that starts to work is the perception module which contains all the sensors and predicted obstacle motion and position information. Then the perception module's output will be taken into the prediction module, which will subscribe to the location and planning information. Prediction can process the data and send it to the planning module as a package. Then the planning module needs the location and vehicle information to plan a safe and collision-free trajectory. The control module will take the and generate it as commands to pass to CANbus. During the process, the Monitor will be gathering data from prediction since it holds the data. Then the data will be sent to the Guardian. Control can usually be sent to CANbus if all the modules are working correctly. Otherwise, Guardian will prevent Control signals from reaching CANbus and bring the car to a stop.

The use case for traffic lights will start when the car is in operation, and the planning module will subscribe to this whole module. The two onboard cameras will be taking pictures of the current traffic light and sending pictures from both cameras simultaneously to the pre-process, which will select the picture that matches the criteria for the next module. The processing module will take in the pre-processed data and push out the result for the planning module. Following that, the planning module will communicate the plan to the control module, which will run the CANbus depending on the data.

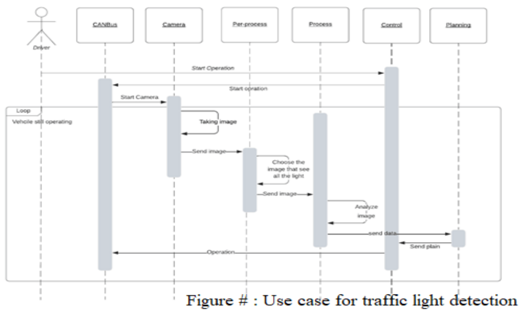


Figure # : Use case for traffic light detection

5. Data dictionary

TensorRT: an SDK for high-performance deep learning inference

6. Naming convention

CANbus: Controller Area Network bus
HMI: Human machine interface
AV performance: Anti-virus performance
LIDAR: Light detection and ranging
CIPV detection: Closest in-path vehicle detection
AEB: Automatic emergency brake
RPC: Remote procedure call
ROS: Robot operating system
ADS: Autonomous driving system
RTPS: Real-time service publish-subscribe

7. Conclusion

Our research believes that Apollo's conceptual architecture is layered at a high level. Furthermore, the processing system uses the publish and subscribe style to transfer the processed data. This design allows Apollo to add more modules and reuse them within the processing system. The overall layered style combining with other styles makes Apollo easy to implement, simple, flexible scalability and more secure.

After it was announced in 2017, Apollo has released seven versions, with the newest being able to face complex urban road conditions. In 2020 "Apollo" will open its self-driving taxi in Beijing, where the operation range is over 700km. It is incredible how much Apollo has evolved in such a short time, from 1.0 being only able to be operated in a parking lot or test track to a public road. Our group is looking forward to seeing what it can achieve in the future.

8. Lessons learned

We learned that the system needs to be integrated with modern technology and continuously improved and that Apollo itself has been iterated for eleven versions. Along the way, the Apollo open platform for autonomous driving has overcome many technical problems of autonomous driving, expanded the application scenarios from narrow roads to urban roads, and gradually approached the mass production of autonomous driving on the ground. We should also keep improving ourselves and evolve a better version when studying the problems. At the same time, we know the importance of each stakeholder. For autonomous driving technology, user experience is the most important, by determining the most critical part first can let us design the software better. At the same time, in the latest speech, Baidu founder Robin Li believes that driverless will give rise to a new industry of shared mobility and that "the self-driving car industry will enter a full commercial phase in 2025". He also predicted that China's first-tier cities would no longer need purchase and traffic restrictions within five years, relying on improving traffic efficiency; within ten years, the congestion problem can be basically solved. Furthermore, we should think about how software should be designed for the future society and what conditions and user needs should be met to be ahead of the times.

9. References

Leonard, J., Fiore, G., Epstein. (2007, December 14). *Team MIT Urban Challenge Technical Report*
https://www.researchgate.net/publication/38000178_Team_MIT_Urban_Challenge_Technical_Report

Peng, Z., Yang, J., Chen, T.-H. (2020, November 8). *A first look at the integration of machine* https://petertsehsun.github.io/papers/fse2020_ADS.pdf

ApolloAuto. (n.d.). *Apollo/how_to_understand_architecture_and_workflow.MD at master ·*
https://github.com/ApolloAuto/apollo/blob/master/docs/howto/how_to_understand_architecture_and_workflow.md

Apollo Auto (2019) Medium
https://medium.com/apollo-auto/apollo-cyber-rt-the-runtime-framework-youve-been-waiting-for-8b2e89206368

Apollo Auto (ND) Apollo.auto https://apollo.auto/cyber.html

HaoBBNuanMM (2019) CSDN
https://blog.csdn.net/HaoBBNuanMM/article/details/86634814

The concurrency viewpoint (ND)
https://www.viewpoints-and-perspectives.info/home/viewpoints/concurrency/

Apollo 3.0 entering the new era of autonomous driving
https://medium.com/apollo-auto/apollo-3-0-entering-the-new-era-of-autonomous-driving-d781bc769cef

Apollo 5.0 Technical Deep Dive
Apollo 5.0 Technical Deep Dive. This is the part two of the two-part… | by Apollo Auto | Apollo Auto | Medium

Inside Apollo 6.0: A road Towards Fully Driverless Technology
Inside Apollo 6.0: A Road Towards Fully Driverless Technology | Apollo Auto (medium.com)

(The dropped member should be in charge of part3.5 and discussing the architecture style)