Conceptual Architecture of Apollo

CISC 322 Assignment 1
Date: Feb 20, 2022

Baik, Daniel
Chen, Oscar
Ma, Yanzhang
Pitcher, Tyler
Rutledge, Josh

Introduction

Apollo is a project developed to create software for driverless vehicles to be used in the commercial sector. In this report, we have included findings on the software's conceptual architecture as well as the background of the Apollo project. Our findings include technical factors such as development of the software such as figuring out control and data flow, as well as any concurrency present. Moreover, this report also describes the functionality of the software and how it was developed as an open-source project with the help of many individuals. With the research conducted on Apollo, the report will end with our conclusion that will consist of factors that we were able to learn while taking a deep look into Apollo's architecture and the relations modules have with each other.

Evolution

Baidu, the company behind Apollo, promises the software architecture is a flexible and efficient platform for developing, testing, and deploying fully autonomous vehicles (Xiao et al., 2022). Following that promise, Apollo's conceptual architecture has changed during each update to implement new functionalities to the system and to fix issues in older code. Initially, the system was capable of mimicking a driver in a closed venue testing facility; however, Apollo could not drive on public roads and relied heavily on global position services. Apollo 1.5 added the perception, prediction, planning, and map engine module to Apollo's open software platform. Starting in Apollo 2.0, the architecture was able to handle staying in lanes, changing lanes, stopping at traffic lights, and avoiding collisions with pedestrians or other vehicles. Apollo 2.0 provided a significant update to the platform. Then in Apollo 2.5, highway support was added without the need for high-definition maps. Cars could now drive on fenced-in highways without the need for pre-captured high-definition maps. L4 autonomous driving on enclosed test tracks was introduced in Apollo 3.0, and modifications to the perception module were made. Version 3.5 of Apollo can drive in residential or downtown areas. The system could now drive in residential or downtown regions thanks to its new perception algorithms and introduction of scenario-based planning. New scenario handlers were added in Apollo 5.0 to support pulling over and crossing intersections. These new scenarios were in addition to the previous scenarios added in Apollo 5.0. Apollo 5.5 vastly improved driving safety, and further planning scenarios were included. Apollo 6.0 saw data pipelines improve across the platform in order to better serve modules with the data they need. In Apollo 7.0, three new deep learning models were added to support the perception and prediction modules. Apollo's advanced platform is on display with Baidu and King Long's minibus system and is capable of L4 autonomous driving. This means the fully electric minibuses are capable of driving without human intervention, even in the event of a potential crash (Choksey, 2021). Since its release in 2017, Apollo has seen continuous support and improvement in autonomous driving, and its highly flexible architecture allows new modules to be added to the system (Xiao et al., 2022). Baidu intends to continue evolving the open platform and hopes to deploy robotaxis in one hundred different cities by 2030 (Kharpal, 2021). Apollo has grown drastically since 2017, and its flexible architecture is the reason for the exciting progress from the onset.

Functionality

Apollo’s seventh version of its open software platform for autonomous driving provides seven major subsystems to analyze. Apollo’s software developers call these subsystems modules. The map engine, localization, perception, prediction, planning, control, and human-machine interface modules are subsystems of Apollo’s architecture. Every module operates on the data provided and outputs its results for other modules to utilize. Each of the seven subsystems can be further divided into smaller subsystems.

Apollo’s control module receives the planned trajectory from the planning module and executes the planned trajectory. Trajectories provided are safe and comfortable routes for the control module to follow. In addition, car status and location are also passed to the control module. The control module's mission is to implement the calculated trajectory by controlling the movement of the vehicle through precise control of the car's hardware components. Apollo’s HMI links to the control module in order to send and receive messages. Messages sent by the HMI to the control module allow the autonomous driving system to make adjustments on the fly.

Autonomous automobiles require highly accurate maps containing information not ordinarily present in other maps. This information includes road markings, traffic signs, barriers, and other data necessary to autonomous systems. Baidu provides pre-captured high-definition maps through its cloud service platform. Apollo’s map engine utilizes these maps to supply other modules. These maps are provided to both the localization and planning modules.

Localization of the car within its environment is handled by the localization module. Localization is an essential ability for every autonomous vehicle. While global positioning systems have dramatically increased their accuracy in recent years, they are still not precise enough to rely on for driverless vehicles solely. Apollo’s localization module fixes this issue by utilizing the car’s GPS, IMU, and LiDAR sensors along with the map engine to accurately place itself on the map to within centimetres of accuracy. IMU stands for inertial measurement unit and provides rotational parameters such as pitch, roll, and heading; furthermore, LiDAR sensors create real-time 3D maps by bouncing light off surrounding objects. Both IMU and LiDAR sensors are outside the scope of this report. The localization module provides valuable data to the control and planning modules.

The perception module produces a real-time 3D obstacle track with obstacle classification by processing input from connected sensors. These sensors include LiDAR, radar, cameras, and a rotational sensor to calculate velocity. After observing the provided data, the perception module leverages its deep-learning neural networks to classify objects such as traffic lights, pedestrians, bicyclists, and other classifications. The primary purpose of the perception module is to detect and observe the car’s surroundings in order to pass this information to other modules.

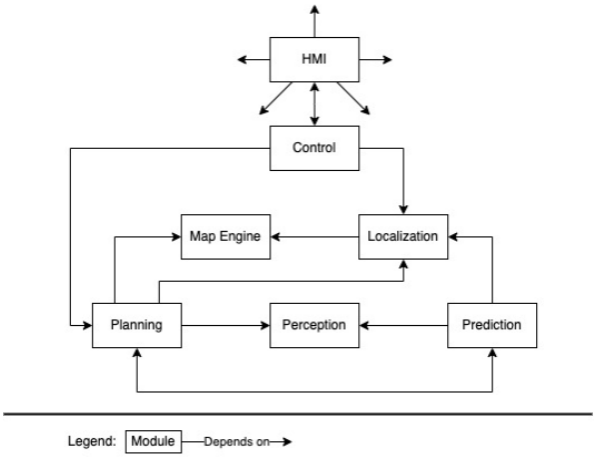
Predicting the movement and priority of objects is the sole responsibility of the prediction module. This module receives the obstacles detected by the perception module, localization data, and the previously planned trajectory for the vehicle. Each obstacle is annotated with its predicted course as well as the priority of the obstacle. There are three priorities to choose from, the ignore priority is given to objects that will not impede the vehicle, while

objects that will block vehicle flow are given a priority of caution; furthermore, any objects that do not meet the requirements for the previous two priorities are given the default priority.

Apollo's planning module creates a new and safe route in real-time in order to provide the control module with an up-to-date route trajectory. Localization, perception, prediction, and HD map data are all piped to the planning module in order to create a course for the autonomous vehicle to follow. The planning module contains handlers for each feasible scenario a car will experience. These include handlers for stop signs, traffic lights, pulling over, and other scenarios. Utilizing each handler, the planning module produces a safe trajectory for the control module to execute.

The human-machine interface present in Apollo's architecture provides digestible information to users inside the car or outside the car. Baidu calls Apollo's human-machine interface, DreamView. Localization, planned trajectory, perceived obstacles, routing, and prediction data are passed to DreamView by other modules to be displayed to users. The one side of DreamView receives data from all modules, while the other side presents this data in a human-friendly manner. Apollo and other developers utilize this interface to understand how the different modules are operating at any given moment.

Each component or module within Apollo's architecture provides a critical functionality to the system. A portion of modules interact with hardware counterparts, but most solely interact with other software modules of Apollo's architecture. The entire architecture could not provide autonomous driving without the cooperation of individual modules. The figure below displays which components rely on one another.



Human interaction with the system is supplied through the use of Apollo's human-machine interface, called DreamView. The powerful perception module handles the detection of surroundings and significant obstacles through hardware sensors capturing data in real-time. Apollo receives high-definition maps through its map engine, and all maps contain a high level of detail about the current area. The localization module helps localize the car in its environment to within centimetres of accuracy by leveraging Apollo's high-definition maps provided by the map engine. Potential obstacle paths are predicted by the prediction module and passed to the planning module to be taken into account when planning routes. The planning module creates a safe trajectory for the vehicle to follow, and the control module executes the scheduled course.

Control and Data Flow

The control flow of a system is best described as the order in which its processes are executed. One task at a time, how the system functions, and how tasks are split up to be carried out simultaneously in turns. In the case of Apollo, the Perception module starts the flow of control. With the perceived environment, the Perception module makes use of the Prediction module to decide where the car is going and what obstructions are present. The Planning module derives a trajectory that safely avoids any obstructions, this trajectory is passed to the Control module which carries out appropriate commands to achieve the proposed trajectory.

To illustrate this flow of events clearly, imagine there is a stop sign on the road we are currently driving along, the perception module would perceive the sign and in turn, call the prediction module. Prediction would analyze that we are approaching the stop sign and will reach it soon. The planning module would be called next and would decide we must slow down and eventually stop as we reach the stop sign. The control module would carry out these actions while also relaying a message to the HMI.

Data flow deals with information being passed from module to module and is far less linear. Information tasks can be carried out simultaneously. Apollo's HD Map module passes data to the Localization module and both modules provide information to the Perception, Prediction, Planning control flow as described above. The Planning module passes information back to the Prediction module creating a loop for more accurate results. The whole control system provides information to the Monitor which provides information to the HMI and back to the system. The flow of these events, unlike those described in control, is carried out in no particular order and can be performed in unison.

Division of Development

Apollo is an open-source software, meaning it is publicly accessible for use, edits, and upgrades. Anyone interested may become involved as a developer. This development style means the distribution of development is vast and there are many individuals involved. There is still the main development team responsible for the majority of the development. This includes a Senior Director, Senior Engineering Manager, Chief Architect, Senior Architect, and a few credited Software Engineers.

Apollo's publicly available GitHub page features 5 main software engineers, most of which are featured on their website as instructors to help interested parties with the use of the Apollo platform and further development. Xiao Wei, Qi Luo, Diego Hu, Calvin Miao, and Pride Leong. These are the people responsible for most of the code present on GitHub and make up the primary software engineering team.

There is a contributor licence agreement you must agree to and sign before becoming involved with any development, however once that is done you are free to contribute freely. There are currently over a thousand open issues. There is a vast array of tags that describe what modules an issue is related to and what type of issues the tag is. New developers are encouraged to explore issues labelled with the "Help Wanted" tag. Apollo's GitHub features over 200 contributors. Who range from having a few commits to hundreds of commits.

This style of development allows for a general direction and a guarantee of program quality to be controlled by a select few. It also allows for an increase in development speed as smaller issues that may be out of the scope of the main developers can be worked on by contributors with less responsibility. The main engineering team will oversee any changes made and ensure stability of the overall system.

Concurrency

Concurrency in computer software architecture refers to collecting techniques and mechanisms that enable a computer program to perform several different tasks simultaneously. Both hardware and software level features need to be implemented to achieve concurrency.

Modern autonomous driving may encounter many difficulties. Optimizing concurrency is one of the central issues engineers need to address. In autonomous driving, the platform should have the ability of local debugging plus fast cloud verification. A complete set of independent driving algorithms is the fusion of multiple algorithms such as prediction, positioning, perception, decision-making, and control. Various functions lead to a high concurrency scenario.

Baidu's Apollo latest version, 7.0, has four layers. The first layer is the certification platform, The second layer is the hardware platform, including computing unit, sensors, V2X, etc. The third layer is the software platform, including the primary operating system. And the top layer is the cloud service, including a map, data, etc. The layer where Apollo optimizes concurrency is taken place in the third layer. The Open Software Platform has two significant features, Apollo Cyber RT and ROS (real-time operating system).

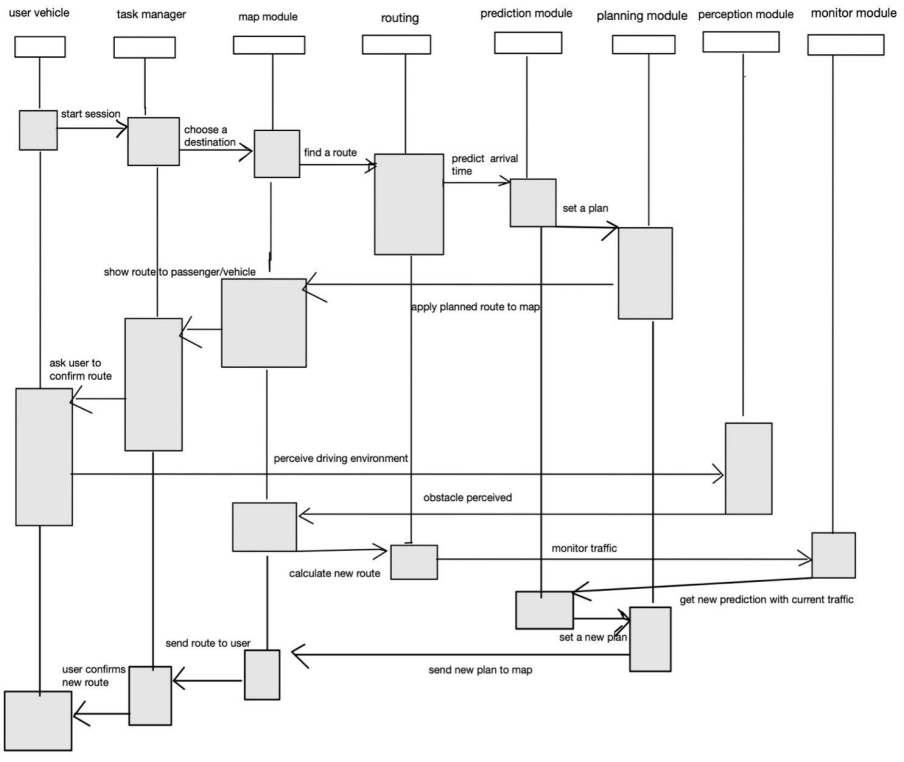
ROS is the first middleware that Apollo used, which has three parts: Communication System, Framework & Tools, and Ecosystem. The Communication System is a Loosely Coupled System (distributed system). It is a socket-based Publish-subscribe communication system, message sent and received between each algorithm module via Pub/Sub pattern. In Framework & Tools, developers can send and receive notifications based on the Client Library and communication layer; they will only deal with the message processing algorithms and leave the rest to the framework. The ecosystem will provide references in the community. ROS generally accepts and processes tasks fast enough, but it still faces challenges. Such as, the system encourages a proliferation of nodes and notelets, while under load, the operating system will have tons of threads, but there are few CPUs. Further, the ROS is too dynamic for hard real-time.

Apollo Cyber RT framework is the first open-source, high-performance runtime framework designed for autonomous driving. Since developers need a robust framework with high performance to support unexpected and challenging scenarios in autonomous driving. The framework's architecture lists components with predefined inputs and outputs. Each piece has a specific algorithm module to process multiple inputs and generate results. The Cyber RT framework is built based on the algorithms within the details. Those algorithm modules will use the DAG (Directed Acyclic Graph) to configure logic between tasks. The framework

combines the predefined components with fused sensor data at runtime to create lightweight user-level jobs. Then each task will be scheduled based on resource availability and task priorities and executed as optimized threads. The process that the scheduler puts tasks into each processor is a coroutine. The Apollo Cyber RT is designed as a centralized and parallel computing model, enabling high concurrency of task execution, low latency, and high throughput. In addition, the framework offers optimized data processing and abstract communication. Easy-to-use task interface and efficient data fusion mechanism allow developers to build their solutions. The framework also has an adaptive design for more straightforward deployment. Its adaptive communication capabilities allow for high data throughput. The user-level scheduler with resource awareness offers the highest performance to the centralized computing model. Advantages of the Apollo Cyber RT to solve ROS's challenges are that Cyber RT moves the scheduling and tasks from kernel space to user space. Therefore, the native threads of the operating system can be equivalent to the actual physical CPUs. In general, the scheduled coroutine will process on the native thread in Cyber RT.

Use Cases

Use case 1: Set/change routine

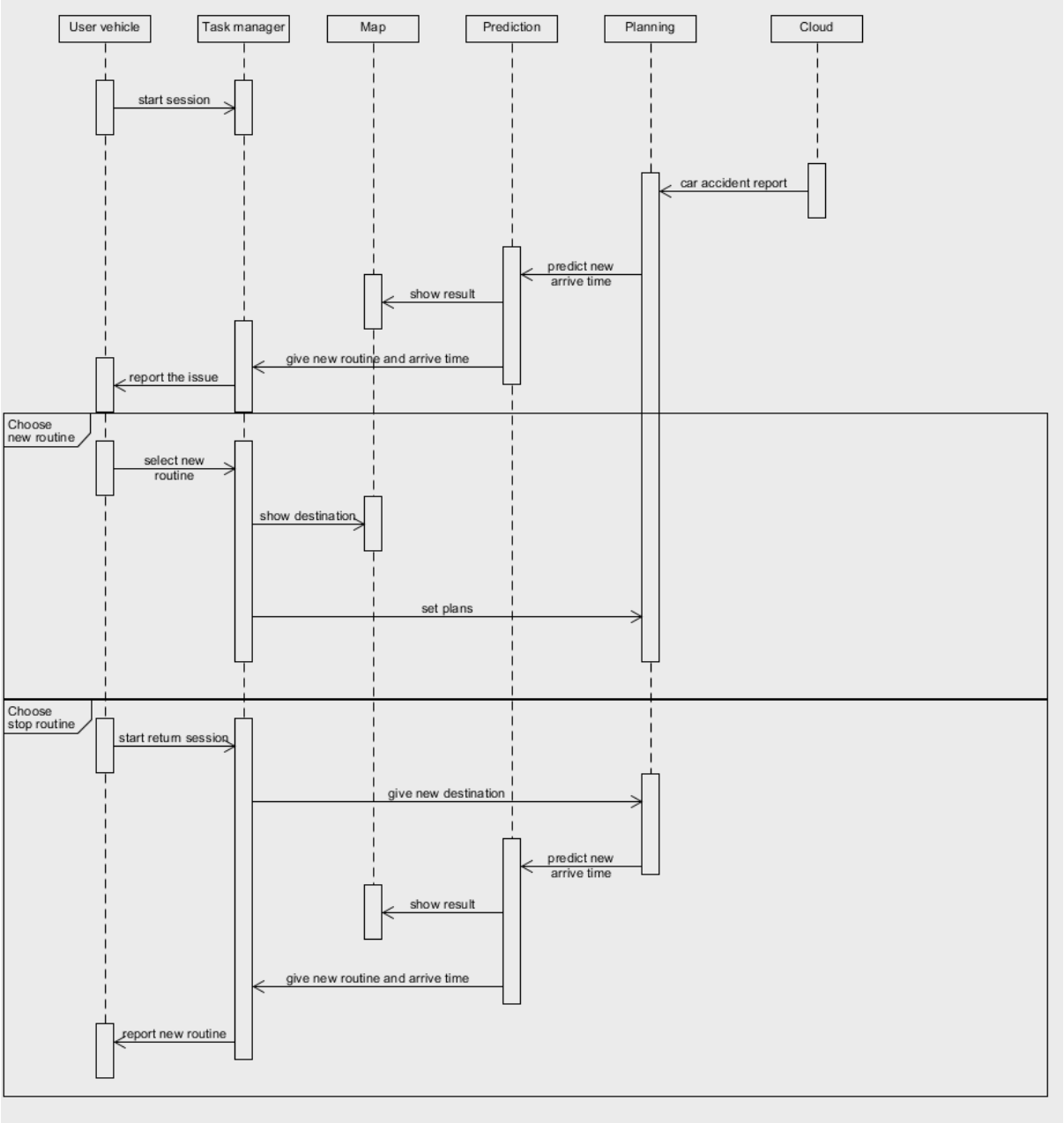


Sequence Diagram Of The User/Vehicle Choosing The Best Route To A Destination, Changing Routes To Update Best Route If Necessary

The above use case is for when the vehicle needs to find the best route to a destination and altering the found route if necessary depending on traffic conditions. First, it goes to the task manager to route the command to the map module where the user can set the destination. Then, the map module sends the command to the routing module in order to find a possible route to the destination to be passed on to the prediction module that predicts the arrival time. Next comes the planning module which can set up a plan for the trip. The command

then goes back to the map module to apply the planned route to the map. Finally, it is sent back to the task manager, then to the vehicle to start the trip. While travelling, the perception module is utilised to achieve a precise driving environment and it lets the map module know if there are some obstacles in the originally planned route. Then, the command goes to the routing module to recalculate the newly optimised route which will then go to the monitor module to monitor the current traffic. Here, it goes to the prediction module again to get the new prediction for the new route and the planning module is followed to set up the new plan. The new plan is sent to the map module to update the planned trip which is finally shown to the vehicle through the task manager.

Use case 2: Car accident



When a car accident happens, the accident location is right on the planned routine. And first, the cloud platform will find the accident location and report the detailed information to the planning module. The planning module receives the data and generates the information for

the prediction module. The prediction module will calculate the result location distance and more information and send it to the map and the task manager. The task manager will report the issue to the device on the user's vehicle. Then the user will have two major choices, to choose a new routine or choose to stop the planned routine and return where they started. If the user selects a new routine, the vehicle device will send the new destination to the task manager; the task manager will send the new destination on the map and the planning module. The planning module will process the data to other modules the same as User case 1. If the user selects stop routine, the task manager will give the original start point location to the planning module.

Conclusion

Apollo, the open-source software which the code is posted on the GitHub for autopilot, provides a flexible and efficient platform to develop, test, and deploy fully autonomous vehicles. After 7 version development, the system already includes the ability of local debugging plus fast cloud verification and fusion of multiple independent driving algorithms, which represent 7 functions-map engine, localization, perception, prediction, planning, control, and human-machine interface. In the 7 version, it is capable of L4 autonomous driving which means the fully electric minibuses are capable of driving without humans by this system. On the control and data level, the processes are executed once of time in the system and data is dealt in far less linear modules, so all the parts of the system will work efficiently and safely.

References

Choksey, J., & Wardlaw, C. (2022, May 5). Levels of autonomous driving, explained. Retrieved February 17, 2022, from https://www.jdpower.com/cars/shopping-guides/levels-of-autonomous-driving-explained

Kharpal, A. (2021, November 18). China's Baidu wants to launch its driverless robotaxi service in 100 cities by 2030. CNBC. Retrieved February 17, 2022, from https://www.cnbc.com/2021/11/18/chinas-baidu-wants-to-launch-robotaxi-service-in-100-cities-by-2030.html

Xiao, X et al. (2022). ApolloAuto/apollo: An open autonomous driving platform. Retrieved February 17, 2022, from https://github.com/ApolloAuto/apollo