CISC 322

Assignment 1
Conceptual Architecture of Apollo
January 28, 2022

Zephyrus

Yucan Li 18yl259@queensu.ca
Yuzhe He 18yh46@queensu.ca
Xuchuan Mu 18xm24@queensu.ca
Yiming Zheng 19yz38@queensu.ca
Wenran Hou 18wh10@queensu.ca
Mukun Liu 19ml13@queensu.ca

Table of Contents

Abstract

This semester our group is going to explore the conceptual architecture design of Apollo from Baidu. Apollo is an open-source platform that enables autonomous driving. We are focusing on several functional modules inside Open Software Platform including Map Engine, Localization, Perception, Prediction, Planning, Control, and HMI. Other operating systems (Cyper, RTOS) and V2X adapters are excluded in this research report. After analyzing the Apollo documentation and further research on autonomous driving, we conclude that Apollo is based on a pipe-and-filter architecture style.

Introduction

Due to the high accident rate on the road, autonomous driving can yet be regarded as an intuitive approach to overcome this problem. Research reports that autonomous vehicles have the potential benefit to increase drivers' safety on the road and reduce their stress and tedium in the meantime.[1] In addition, autonomous driving can also be applied in circumstances like cargo transporting and public traffic infrastructure. With the growing information density and diversity of the geographic labor force, the cost of maintaining such modules becomes higher and higher. As a representative of decentralized local units, autonomous driving is gaining a significant position.[2]

As one of the rising autonomous driving industries, Apollo, is an open-source autopilot platform belonging to the Chinese biggest search engine "Baidu". Autonomous driving technology is achieved under several modules. The application of Localization helps vehicles acquire a general view of where it's located and what it's surrounded by. The Perception and Prediction modules give vehicles intelligence. Vehicles can not only follow road lines but also deal with unintended events like the sudden appearance of pedestrians and car accidents. In terms of Planning modules, it acts like navigation software but with much less response time to a route change. The Control module follows the route that was planned by the Planning module and gets ready to activate or stop the vehicle as soon as possible. Upon these modules, the Human Machine Interface (HMI) gives drivers a clear and laconic interface to make the interaction between Apollo and driver as simple as possible.

Apollo leaves drivers under a safe and undemanding circumstance. A reliable platform is based on a user-friendly interface and solidary software structure. The following report will give out a general view of how Apollo's conceptual architecture is implemented. The relation between each module and its dependency will be pointed out and will explain why this platform is well-structured and organized.

Conceptual Architecture

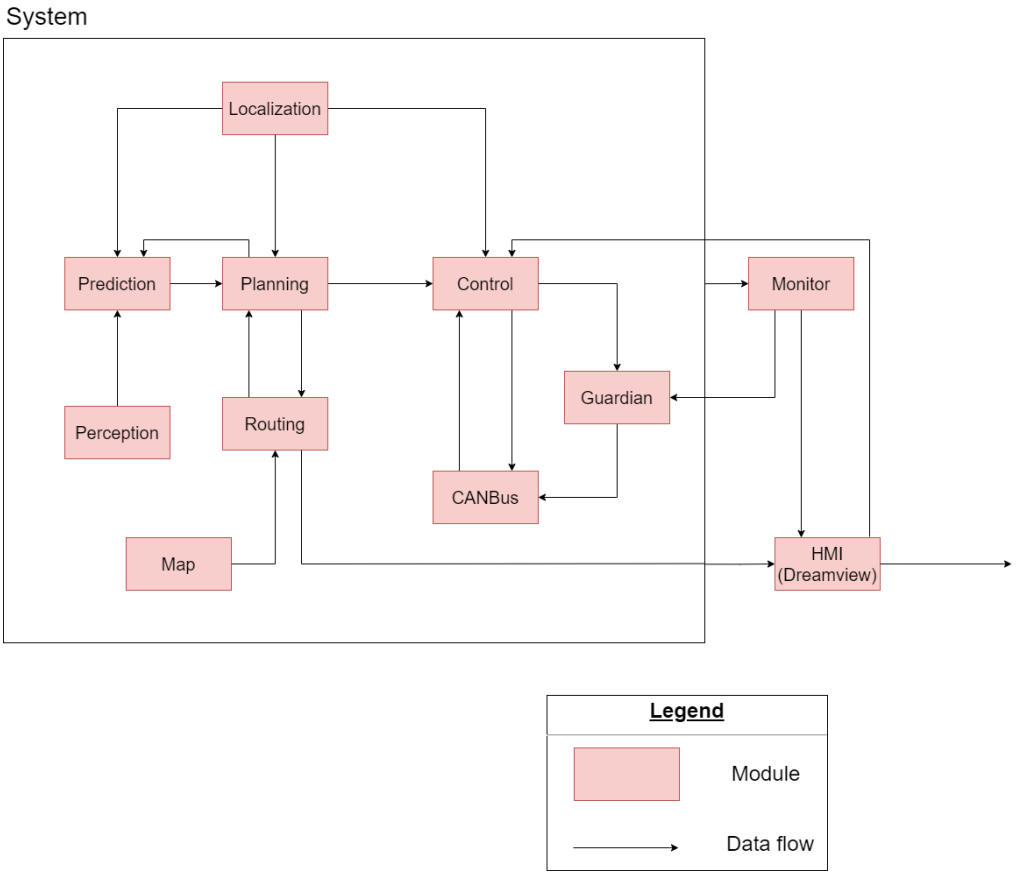

Figure 1: The conceptual Architecture of Apollo Open Software Platform

Our research result is a pipe-and-filter architecture style. The above red boxes and arrows indicate the sub-module relationships and dependency inside the Apollo Open Software Platform. According to what we learned in the lectures, the pipe and filter architecture style defines the structure where the components (filter) read the stream of data as input and produce streams of data as output. It is suitable for applications that need a defined series of independent computations to be performed on data, and Apollo is one of those applications while it reads the input from the sensors and maps and produces the stream of control commands as output. Each module acts as a filter that computes the data and their data flow acts as pipes that work as the conduits for streams.

In addition, the interaction between when Map and Localization modules communicate with Perception, Prediction, and Planning modules follows the client-server architecture style.

HMI is the main interface users use to interact and get feedback from the modules. The planning module acts as the main core of the whole system which receives the current location, long-term routing, and prediction path and turns this information into a short-term procedure. Then the Control module will make the planning come into action.

Map

In Apollo 7.0, the main role of the map module was to load the map and provide a series of APIs for routing, perception, prediction, planning modules to get the identifying information of traffic elements to use. The HD map mainly stores static map information collected in advance from offline. It has the advantage of being highly accurate and not dependent on road conditions. The role of the relative map is to connect the HD map to the perception and control modules. It generates a map based on the body coordinate system in real-time by combining the navigation information provided by the dreamview module, the lane marker provided by the perception module, and the localization information provided by the localization module. (Input information is divided into offline and online components. The offline information comes from the navigation line and HD map information, and the online part comes from the traffic marker information provided by the perception module)

Apollo also developed a navigation mode based on the relative map. The navigation mode generates a reference line by recording the trajectory with a human driver and editing it with Apollo tools. It is used to replace the navigation path output with the routing module and give reference to the planning module. It also works with the visual perception system to generate basic map information and is the baseline for generating relative maps.

To import the reference line, Apollo adds a navigator module. Navigator is a cloud service that stores guideline data, HD map module‘s data, and a series of modules that process the data. Navigator connects HD maps to relative maps. It provides guidance information in the relative map and can be used either alone or in conjunction with the HD map.

Routing

The routing module only depends on the map module, and its function is to find the shortest route between the departure and destination. It has two inputs, which are the request of routing and map information of HD map. The routing module makes a graph according to the map information and stores the completed graph in the routing map. Finally, the trajectory of the shortest route of the routing map is passed to the planning module. However, when the planning module rejects the route given by the routing module based on the real-time information, the routing module recalculates the route and passes the new route back to the planning module.

Localization

The localization module is used to locate the temporal position of the vehicle and output information like the vehicle's status and velocity. This module mainly acts as an external device that collects data from surrounding and transfers them to inner modules, just like a pair of eyes for the Apollo. This purpose is achieved through the three different locating methods: Real-Time Kinematic (RTK) based method. NDT-based Lidar Localization (NDT) and Multi-sensor Fusion Localization method(MSF).

RTK applies GPS/GNSS and Inertial Measurement Unit (IMU) to achieve global localization. Although GPS localization is accurate, it is difficult to achieve real-time positioning due to its low update frequency. IMU has the advantage of high-frequency updates, but it can produce

kinematic errors at any time. The two functions complement each other to establish the RTK module.

NDT determines the specific information of four dimensions (X, Y, Z, and Yaw) through point cloud technology. According to this information to determine the specific location of the vehicle to achieve the accuracy of positioning. If this information is passed to the MSF algorithm, the location will be more accurate.

MSF outputs high-precision vehicle position and attitude through data interaction between two modules and a framework. The GNSS module provides accurate RTK positioning results according to GPS signals. LiDAR outputs high precision lidar positioning results from online lidar scanning data and lidar positioning maps. The SINS module in the fusion framework uses IMU data for inertial navigation. The framework uses the error-state Kalman filter at the back end. “It estimates the error of the SINS and uses the estimated error state to correct the SINS.” [3]

These three methods will create a “Protobuf” type message which will proceed to the following modules. At the same time, the localization module will provide the calculated high-precision vehicle location information to the three modules (Prediction, Planning, and Control) in real-time.

Perception

This module is used to recognize surrounding obstacles, incoming traffic lights, and lanes to follow. It comprises front cameras, front and rear radars, and four LiDARs around the vehicle. The cameras are not only able to determine the current status of traffic lights, but also implemented with the latest SMOKE technology. Radars at the front and rear of the vehicles send and collect radar waves to surrounding environments. The four LiDARs also collect obstacle information from the surroundings which use similar technology as the Localization module does.

After receiving information from the cameras and LiDARs, the Perception module is going to pre-process the images from cameras and point cloud data from LiDARs. After data is fully pre-processed, they are put into deep networks for learning. The algorithms are based on CNN, which takes imported images lists as inputs and goes through a deep calculation, and has an activation function to provide the result of the recognition of the images. In the perception module, cameras are responsible for recognizing traffic lights. Due to the several cameras and the neural networks cannot reach 100% accuracy, there would be multiple results for traffic lights. A vote will be thrown and the final traffic light result will follow the highest vote result. For object recognition, it requires cameras, LiDARs, and Radars to work together and a final result could be conducted. The pretreated obstacle information that is collected from the camera, radar, and LiDAR will be combined through “Configurable Sensor Fusion” and turned into 3D obstacle tracking information including heading, velocity, and classification. [3] The traffic light results and obstacle tracking information will be transmitted to the Prediction module.

Prediction

The prediction module is used to predict the movement of obstacles and provide each trajectory with a probability value. There are four different functionalities inside of Prediction: Container, Scenario, Evaluator, Predictor. Besides, obstacles are defined as variables that have a feature called priority categorized into ignoring, Caution, and Normal. The container acts as a storage database. All the input data from Perception, Localization, and Planning is imported into the Container. The scenario module analyzes the current scenario including Cruise and Junction. The evaluator module first makes a prediction about path and speed to input obstacles from Container and provides output probabilities of paths. The predictor uses the possibility from the previous module to calculate the predicted trajectories for obstacles. Finally, each obstacle is marked with specific trajectories and priorities, then they are packed up and sent to the Planning Module.

Planning

Planning is the central module of the Apollo system. It requires inputs from multiple modules, including information of the vehicle from Localization, traffic light recognition from Perception, obstacle trajectories from Prediction, localization from Map, routine from Routing, and chassis information from Canbus. Compared with the Routing module, the Planning module tends to provide the vehicle with a short-term trajectory. The planning module first generates a navigation routing, then the vehicle follows the short-term trajectory to the destination which depends on navigation track and road conditions. As soon as a suitable trajectory of the vehicle is planned, it will form an execution command and transits to the Control Module.

Control

The control module of Apollo is the module that commands the vehicle, gives orders including acceleration, speed, and steering based on the vehicle's Planning trajectory and car's status to make the autopilot journey safe and efficient. The control module will take inputs from the planning module and localization module by CANBus to get Planning trajectory, Car status, localization, and Dreamview AUTO mode change request.[5] The module will process these data in 3 steps: preprocessing, controller and postprocessing.

For preprocessing: the inputs will be checked by the module and then reject the bad input including wrong or abnormal information, then it will handle the emergency status, and finally, it will filter the inputs for signal smoothing, Lag compensation, loop shaping, etc.[6]

After preprocessing, the inputs will be handled by the controller where they will be used to control the vehicle's movement. In reality, the vehicle's planning trajectory and its actual trajectory and position might not be identical, so the controller has to modify the vehicle's driving trajectory in order to achieve the following goals:

- Accuracy: the controller has to avoid departure from the planning trajectory. It also provides safety to the journey since in some situations moving off the Planning trajectory might result in a dangerous state for the vehicle.

- Possibility: the controller has to make a possible command. Reality is not made of code, so the control command must obey the physical law.
- Stability: to make the journey comfortable and safe.

There are three 3 types of controllers widely used in the Apollo control module: PID controller, LQR controller, and MPC controller.

For post-processing, the signal is filtered again, and the control commands will be sent to the chassis through the CANBus.

CANBus

CANBus refers to the Controller Area Network (CAN). First, a bus is a specialized internal communication network between components inside a vehicle. A Controller Area Network is a robust vehicle bus that can make different microcontrollers or devices communicate with each other without a host computer which has better extensibility and convenience.[9]

Here CANBus is taking command from the control module and collects and sends the status of the vehicle's chassis as feedback to control. It mainly acts as a connecting bridge between the vehicle and the autonomous driving software. The outputs are chassis status and chassis detailed status. CANBus is implemented by two major components: Vehicle and CAN Client. The vehicle is the vehicle itself since the Apollo system might adapt to different cars (GitHub uses the Lincoln mkz as an example). It includes the vehicle's controller and message manager of the details and status of the chassis. CAN Client is shared by different sensors so it moves to the hardware part, which is not in our scope.

Guardian

Just like its name, the purpose of the guardian module is to protect the vehicle and the driver. It will stop the vehicle when the system faces any emergency. The guardian module sits between the monitor, control modules, and the Canbus. It receives the inputs of alerts from the monitor module when an emergency happens.

In the regular situation, the guardian stays “asleep”, and allows the control module to send commands to the CANBus.

When the guardian is alarmed by the monitor module, the guardian module will take over the control of the Canbus from the control module. The guardian will have 3 different ways to stop the vehicle depending on the state of the ultrasonic sensor. (1)If the state of the ultrasonic sensor is normal and no danger is detected, the guardian will stop the vehicle with a safe halt. (2)If the ultrasonic sensor has malfunctioned, the guardian will execute a hard break. (this module constantly checks the state of the ultrasonic sensor) (3) If the driver did not react to the HMI or vehicle’s alarm in 10 seconds while the danger was detected. (Ex. The collision is expected to happen) The Guardian will stop the vehicle with a hard break.[7]

Monitor

The monitor module is a system-level software that its main job is to monitor the entire system including detecting hardware status and system health. It performs the following checks: Detecting the status of previous models, Monitoring unusual data includes checking

the data frequency to see if there's too much error data, Monitoring the system's effect on the launching PC. For example, the use ratio of CPU and latency between processes of modules.[8]

In general, the Monitor module needs to detect a lot of things. These things include both hardware parts and software parts.

The working flow of the Monitor module is basically getting information from HMI, generating system status, and giving it back to HMI and Guardian. The input data of the Monitor module comes from the HMI or Dreamview module. HMI generates a special output HMI status and HMI mode and converts to Monitor by serializing data through protocol buffers. The output of Monitor is mainly system status and provides it to the HMI and Guardian modules.

The implementation of the Monitor module is basically through two parts. Hardware Monitors and Software Monitors.

Hardware Monitors: This part includes GPS, Resource, ESD-CAN, SOCKET-CAN. The monitor checks these statuses and reports back to HMI.[9]

The software monitors can be divided into three parts, process monitor, topic monitor, and summary monitor. The process monitor is to check if a process is running or not and the topic monitor is to check if a given topic is updated normally. The summary monitor is to summarize the results from other specific monitors into simple conclusions such as OK, WARN, ERROR, or FATAL. Some other monitors like channel monitor and latency monitor, their job is to monitor the data in all the modules to check data frequency and calculate the latency between each module. Basically, each monitor's output inside the software file will be collected and stored in the system status.[9]

HMI

HMI or Apollo's Dreamview module is an application that can visualize the current output of the autonomous driving modules. HMI refers to the human-machine interface. More precisely, the Dreamview module provides a human-machine interface in which you can select different views of hardware and check the output from the Apollo modules like planning trajectory, car chassis status, etc[8]. All are presented in an intuitive way for developers to have a better understanding.

Since HMI provides all the relevant autonomous driving module outputs, the input of HMI must be the input from all the previous modules. Data flows to Dreamview through protobuf. Currently, it monitors the following messages:

- Localization, including visualized information of car localization. Eg. Visualized MSF.
- Chassis and chassis status, defined by CANBus module.
- Planning, eg: planning trajectory.
- Monitor, which mainly distributes data of system status.
- Perception. Inputting perception obstacle to the Dreamview.
- Prediction obstacles.
- Routing module's output through protocol buffer.

The output of HMI is a web-based dynamic 3D rendering of the monitored messages in a simulated world[8]. This is implemented through the front end and backend. All of the visualized outputs are shown on the UI of HMI.

Concurrency

In the Apollo system, concurrency is reflected in the processing and transmission of data by the different modules.

When the user sends a navigation request, the routing module obtains the exact origin and destination information from both the localization module and the map module, respectively. The MFS method does not work until RTK and NDT have completed their respective calculations. When the routing module finishes its work, the system will derive the shortest path from the origin to the destination.

At this point, the planning module of the Apollo system will start working in conjunction with the perception module and the prediction module. The perception receives road information in real-time through multiple sensors, and the prediction module continuously calculates the received information (paths, obstacles). module in the prediction module of many types of predicator at the same time for different obstacles for the calculation of the movement of the trajectory, the final calculation of the data will be encapsulated and transmitted to the planning module, the planning module planning to adjust the navigation trajectory, and constantly transmit the instructions to the control module, the control module set up three controllers, three controllers must work simultaneously to complete the instructions from the planning module

The control module will execute these commands constantly. During this time, the operating status of each module is monitored by the monitor module, and the status is returned to the HMI in real-time so that the user can confirm that the system is functioning properly. When the monitor monitors a fault, the module will also pass the information to the guardian module. When the guardian module is working, the control module will not work with it at the same time to avoid the conflict of commands.

Evolution

Apollo was released in 2017 and now it has been through 8 versions. The first version released is Apollo 1.0. It was also called Automatic GPS Waypoint Following and can only work in enclosed venues. In this first-generation, only Localization, Control and HMI are included in the Open Software Platform. Later in Apollo 1.5, LiDAR was added to perform better and it mixed lane cruising. 4 modules were added: Map engine, perception, planning, End-to-end. Urban roads autonomous driving was first introduced in Apollo 2.0. Apollo Team deletes end-to-end modules and makes updates in Map engine, Perception, and Planning. Later in Apollo 2.5, it allowed vehicles to run on geo-fenced highways and added cameras to transfer images into the Perception module for obstacle detection. The major updates are in Perception and Planning. Apollo 3.0 provided an open platform for developers. Vehicles at this time were able to maintain lane control, cruise, and avoid collisions with vehicles ahead of them.[5] Apollo 3.5 was capable of navigating through more complex driving scenarios in downtown areas. 360-degree visibility was introduced. The Localization, Perception, and Planning were updated. Apollo 5.0 improved performance for Geo-Fenced Autonomous Driving and added scenarios. For example, unanticipated cars pull over and passengers walk through crossroads. A new module is added: Prediction. For Apollo 5.5, the

teams enhanced autonomous driving capabilities, especially in urban areas, and added curb-to-curb driving support. Major update in modules: Perception Prediction, Planning, and Control. In Apollo 6.0 and Apollo 7.0, the aim was to add a better deep network for the Perception module to make autonomous driving perform better.

Sequential Diagram

According to the conceptual architecture of our article, we intend to use sequence diagrams to illustrate the two vehicle driving patterns. The first is an urban model that works in a multi-barrier area. The second is the cruise mode that can maintain the car in motion.

Urban Mode

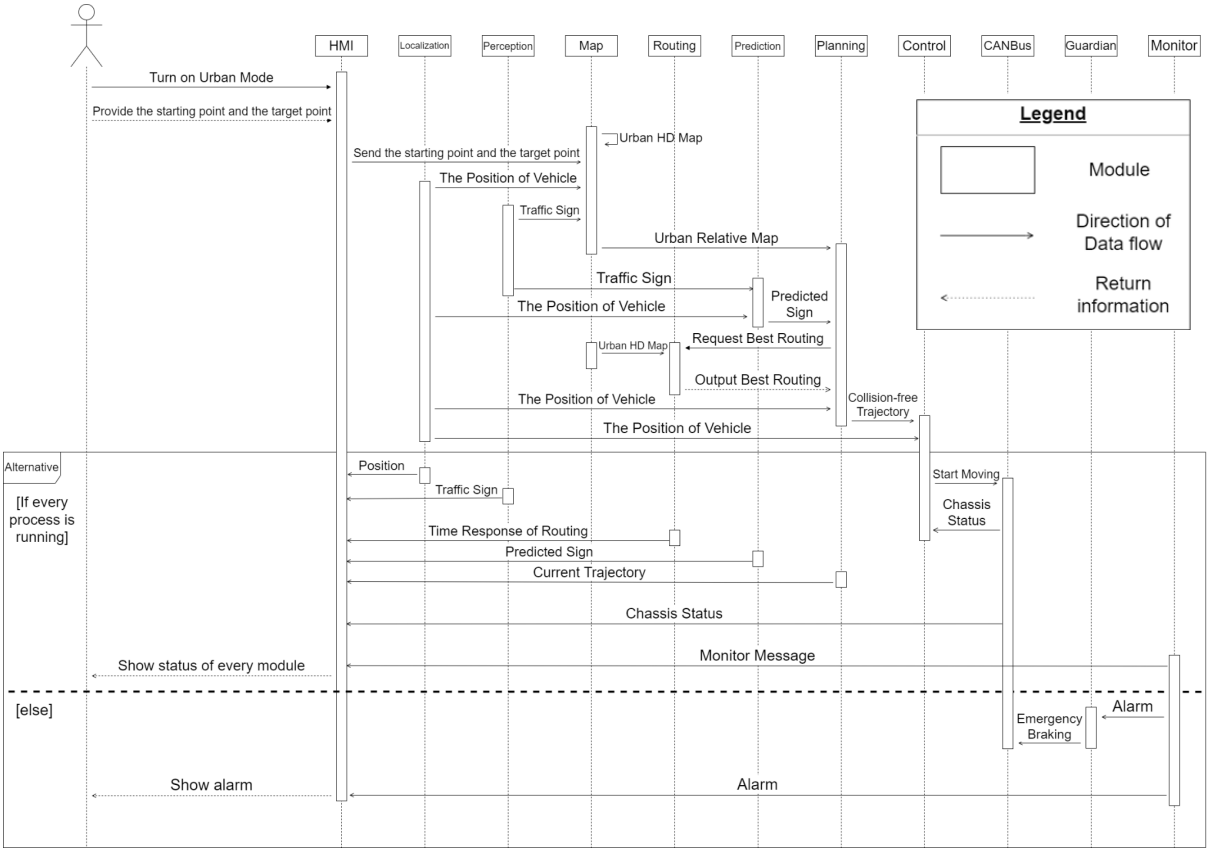

Figure 2: Apollo System in Urban Mode

After the urban mode is enabled, the driver needs to provide information about the starting point and destination for HMI. The Map module will generate a relative map of the urban area based on the HD map of the city, destination, current vehicle location, and traffic signs. At the same time, the prediction module will use the vehicle location and traffic signals to calculate the predicted traffic signal. The Routing module receives a request from the planning module to create the best path using the HD map. The planning module will use these outputs and vehicle location information to generate a collision-free trajectory. If every module is running, the control module will send commands to CANBus based on collision-free trajectory, vehicle position, and chassis status. HMI also summarizes the status of the vehicle and sends it to the driver. However, if there are some errors, the monitor alerts both the guardian and the HMI. The CANBus module will receive instructions from the guardian to execute the command of emergency braking. The HMI will show the alarm to the driver. It's up to the driver to decide what to do next.

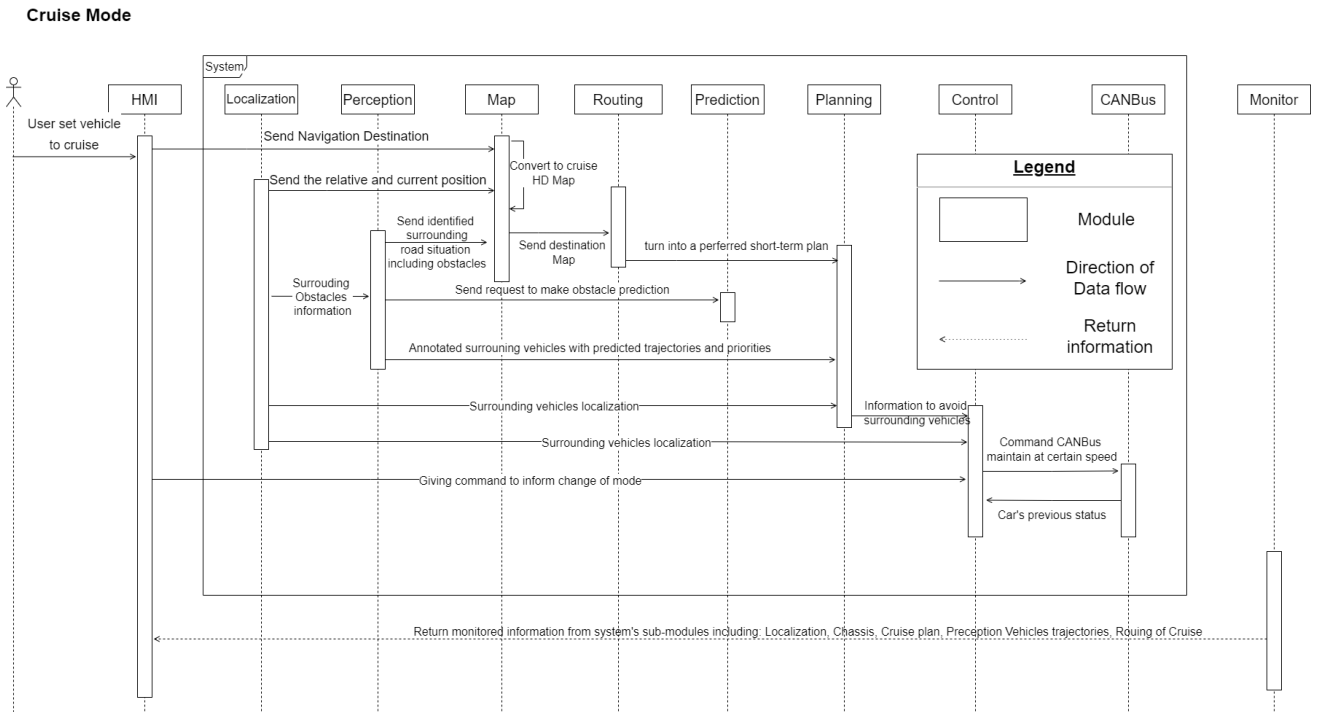

Figure 3: Apollo System in Cruise Mode

When the user sets the mode of Apollo to cruise mode through HMI, a signal about the navigation destination will be sent to Map. After receiving current localization information, the perception module will articulate trajectories related to surrounding obstacles like traffic lights and vehicles. This data is transferred to Routing to make a long-term estimate of navigation. In the planning module, it receives incoming information about how to plan cruise cars from localization, prediction, and routing. After that, Planning posts a preferred plan to the Control module and orders the car to activate cruise mode through CANBus. All the monitor information within the system will be returned back to HMI and reviewed by the user.

Division of Responsibility

The Apollo autonomous driving system is an international project which was created by developers in China and America divided by modules and functions. This kind of division of responsibilities among participating developers brings both advantages and risks.

There is no wonder that this kind of division benefits the development process. The Apollo system highly depends on Artificial neural networks and AI algorithms since it was widely used in prediction and planning modules. These kinds of techniques are completely different from traditional software engineering techniques (Ex. Controllers from control modules). Dividing Responsibility and assigning them to different expert developers can accelerate the development process. However, this kind of division also brings safety risks to the system. Autonomous driving is an Engineering project which is a safe-critical system, unlike normal software (web, social media, game, etc.) any small bugs or defects could lead to dangerous accidents. Division by modules could lead to defects in Data Synchronization. For example, if developers update a new and more time-consuming trajectory-generating algorithm, but forget to update the input filter and latency elimination code in the control module. It leads to delay of control commands which are very dangerous to the journey.

Derivation Process

In our early research about autonomous driving, we have built up a basic view of the functional module architecture.

During the discussion phase of the report, we thought that the architecture style of Apollo might be object-oriented because the Apollo system works by concurrency of different modules doing different calculations and processing. Each module can be seen as a different object which is connected by functions and methods that process the different inputs such as the sensor's result or map.....However, one of the most important features of the object-oriented style is that the first object must know the identity of the second object while this kind of feature does not exist in the apollo system between its modules.

Our group also has concerns when reading some files saying that the Perception module reads input from Localization and HD Map. However, these files were last updated in 2018 and we are not sure if we should keep following its idea. In order to solve this ambiguity, we tried to find useful information from the Perception module. As shown in Perception documentation, there is no need for input data from Map and Localization since Perception could collect the required information on its own. As a result, we decided to follow recent documents whose map modules did not contain input from localization and HD Map. The Perception module had the most updates, especially for Apollo 6.0 and Apollo 7.0, the major updates were for the Perception module and in Apollo 5.5, they added Prediction to predict the obstacles' trajectories which made the Perception module focus on recognitions.

Limitation and Lesson Learned

During the conceptual architecture research of the Apollo project, we not only have made progress but also met unexpected challenges and learned valuable lessons.

The Apollo project experienced development through several generations and we are focusing on the 7th generation of Apollo. However, our group is having trouble integrating sub-modules into modules. During the development of Apollo, the open software platform also experienced iterations. There are only three modules in the first generation (Localization, Control, and HMI) and modules increase to 7 with Map, Perception, Planning, End-to-End are introduced. Until Apollo 5.0, the modules finally became similar as we did research today with the introduction of the Prediction module and the deletion of the End-to-end module. There are sub-modules under these modules or sub-modules that act as bridges or extend modules during development including CANBus, Monitor, Guardian, and Routing. They are not categorized by Apollo officials into the open software platform, but they still play important roles in integrating the whole system. Some of the documents in GitHub are not keeping up-to-date, so we have to find out the latest relevant information to help us understand modules.

After the analysis of the whole Apollo project, we have built up a basic understanding of how software is built up piece by piece. By connecting the relations between modules and modules, we understand how every single module plays its role in the system. They either control the data flow synchronously or asynchronously. In order to deal with accidental events or accidents, modules will implement deep learning to vote out the most suitable solution.

Conclusion

In Conclusion, the Apollo autonomous driving system uses a pipe-and-filter architecture style. By entering certain commands, the whole sub-modules in Apollo are able to cooperate with each other and perform autonomous driving. Having HD Map and Localization to provide a real-time location for vehicle, Perception, Prediction, Planning, Routing to recognize the road situation and make a calculation for best control of the vehicle and finally Control and CANbus to execute the control information, users only have to input their command in HMI and let the Apollo deal with the remaining. The concurrency within modules is highly automatic and has quick responses. This report allows readers to get a clear and brief overview of the whole Apollo software.

Glossary

Dreamview: a web application visualizes the output of relevant autonomous driving modules
Chassis: The supporting frame of the vehicle, in this report, could be considered as an interface to control the whole vehicle.
CANbus: Control Area Network, a bridge lies between vehicle and autonomous driving which receives control commands and re-transmit to chassis.
SMOKE: A single-stage monocular 3D object detection model, specifically, it helps vehicles to perceive surrounding objects, identify their category and even track the flow of lanes on the road.
CNN: convolutional neural network
Protobuf: (Protocol Buffer) a type of message format that serializes data into one specific binary format that could be transferred between multiple modules. A very efficient way with high compatibility to transfer data inside a program.
Geo-Fence: a virtual perimeter for a real-world geographic area [10]

Reference

[1]Litman, T. (2022, February 3). *Autonomous Vehicle Implementation Predictions - vtpi.org*. Victoria transport policy institute. Retrieved February 5, 2022, from https://vtpi.org/avip.pdf

[2]Flämig, H. (2016, May 22). *Autonomous vehicles and autonomous driving ... - springerlink*. Springer Link. Retrieved February 5, 2022, from https://link.springer.com/chapter/10.1007/978-3-662-48847-8_18

[3]Wan, G., Yang, X., Cai, R., Li, H., Zhou, Y., Wang, H., & Song, S. (2018, May). Robust and precise vehicle localization based on multi-sensor fusion in diverse city scenes. In 2018 IEEE international conference on robotics and automation (ICRA) (pp. 4670-4677). IEEE. Retrieved February 10, 2022, from https://arxiv.org/pdf/1711.05805.pdf.

[4]Feng, M., & Zhang, H. (2022, January 30). *Application of Baidu Apollo Open Platform in a course of control simulation experiments*. Wiley Online Library. Retrieved February 5, 2022, from https://onlinelibrary.wiley.com/doi/full/10.1002/cae.22492?saml_referrer

[5] Xiao, X., & Zhag, L. (n.d.). *Apolloauto/Apollo at R7.0.0*. GitHub. Retrieved February 16, 2022, from https://github.com/ApolloAuto/apollo/tree/r7.0.0

[6]Apollo. (n.d.). Retrieved February 16, 2022, from https://apollo.auto/devcenter/coursetable_cn.html?target=2

[7]Apollo Auto. (2018, July 18). *Apollo 3.0: Entering the New Era of autonomous driving*. Medium. Retrieved February 16, 2022, from https://medium.com/apollo-auto/apollo-3-0-entering-the-new-era-of-autonomous-driving-d781bc769cef

[8]ApolloAuto. (n.d.). *Apollo/modules at R7.0.0 · Apolloauto/apollo*. GitHub. Retrieved February 18, 2022, from https://github.com/ApolloAuto/apollo/tree/r7.0.0/modules

[9]Wikimedia Foundation. (2022, February 14). *Canbus*. Wikipedia. Retrieved February 18, 2022, from https://en.wikipedia.org/wiki/CAN_bus

[10]Wikimedia Foundation. (2022, January 17). *Geo-fence*. Wikipedia. Retrieved February 18, 2022, from https://en.wikipedia.org/wiki/Geo-fence