CISC 322
Assignment 1 Report
Apollo: Autonomous Driving Vehicles
Friday, February 18, 2022

Group 1: Hextech
Benjamin Hui (Ben) (He/him) (18bh13@queensu.ca #20148554)
Fuwei Zhuang (Elina) (She/her) (19fz2@queensu.ca #20189056)
E Ching Kho (Noon) (He/him) (17eck3@queensu.ca #20118077)
Zewen Zheng (Zelvin) (He/him) (18zz114@queensu.ca #20150673)
Yixin Su (Allen) (He/him) (17ys114@queensu.ca #20108833)
Ruiyang Su (Amelia) (She/her) (18rs60@queensu.ca #20151860)

Abstract

The conceptual architecture of a system depicts how the system can function through the interactions between its respective components and subsystems. The chosen architecture system is the Apollo autonomous driving system by Baidu, otherwise known as Baidu Apollo. The driverless vehicle system has been deployed since 2017 and is on an open-source operating platform with Apache 2.0 License. The system ranges from smart traffic signals, visual detection, perception, autonomous taxis and buses, and even valet parking. Apollo's evolution, concurrency, subsystems, interactions, and potential use cases are discussed in the following document. Our proposed architecture styles will also be mentioned as part of the report and will provide the derivation process throughout the document.

Introduction and Overview

Currently, autonomous driving is taking the world by storm with its cutting-edge research. Emerging AI technology and revolutionary technology have been pioneered and developed throughout the years. With Tesla leading at the forefront, Apollo has been testing their system in China with great success. Their Robo-taxis and minibus accurately navigate through the roads and are expected to bring three million users to its system by 2023. With its recent expansion of a national-level pilot demonstration zone for their intelligent vehicle data gathering, its partnership with over seventy-plus car companies in intelligent in-car systems has catapulted its progress within recent years. The system not only features autonomous driving systems, but also navigation systems, traffic light optimizations, and visual perception and mapping systems. Apollo has now reached its latest version at version 7.0 and incorporates numerous new deep learning models that enhance the capabilities for its perception and prediction models. As its vehicles constantly compile and pool data together, the systems are constantly learning and improving as the data pool exponentially increases. The report will touch upon the system's conceptual architecture, subsystems, concurrency, control, data flow, use cases, diagrams that model the system, and a detailed analysis of the overall system.

Derivation Process

The Making of Components

When thinking about autonomous driving, the immediate response would be AI doing the work. What is AI? AI (Artificial Intelligence) is the process of maximizing your expected utility and a basic AI agent model is shown below:

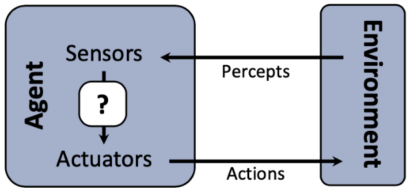

Figure 1. AI Agent Model

An agent is an entity that perceives and acts in the environment it is placed in. If the vehicle is the agent and our utility is to drive automatically after deciding the starting point and destination safety on roads, it must have a perception component and a control component of the vehicle to perform what we have said. How can a vehicle recognize its current position? There should be at least 3 components: perception - knowing what is surrounding through sensors, localization - knowing where the vehicle is on a wide scale, like GPS, and a map such that it lists all the road characteristics like speed limit or traffic lights or the number of lanes. For Actions, there should be 2 components, one is to interpret/compile the software language to a lower level machine language for the hardware to understand and one acts as an interface to transfer codes to hardware. But how do we decide what should be coded? There should be a scheduler or a planner to decide which paths to take and in that case, the planner requires more information like a prediction of the objects ahead and all the possible roads/paths the vehicle can go from starting position to the destination. To make sure the whole system is working healthy, we need to have a monitor to look at the health of every component and make sure they are doing well. If the monitor detects a failure of the component, it will send a signal to a handling center or emergency center. Finally, there should be a user interface such that users can view the status of each component and enter the location of the starting point and destination.
Thus we have derived 11 components: Perception, Localization, Map, Control, CANBus, Planning, Prediction, Routing, Monitor, Guardian, and HMI.

Architecture Styles

Overall the architecture is a pipe and filter style. Though it is specialization such that filters do know the identity of their upstream or downstream filters. It precepts information from the environment and processed by different modules and outputs different actions for the vehicle to drive autonomously. One thing to notice is that every module is a hybrid style that has a publication-subscription style embedded with other styles since each module node publishes and subscribes to certain topics. The subscribed topics serve as a data input while the published topics serve as data output. Perception, Localization, HD-Map is Repository style which stores a

central body of information of the environment and the rest of modules get and operate on the central data. Then, the data is passed on to Routing and Prediction. Planning will receive outputs from prediction, routing, localization and plan a safe and collision-free trajectory. The Planning output will be sent to Control, in which Control is an interpreter that generates the control command and passes onto CAN Bus where CAN Bus is the interconnection of software commands to hardware controls and having a Process-Control style will help maintain the specified properties of the vehicle that the Planning has decided. The monitor is somewhat a peer-to-peer style that connects to all the modules to receive their status and data to ensure all the modules are working without any issue. The monitor is also a client/server style which has the role of a server and sends data to HMI which is the client. If Monitor receives any module or hardware failure, it will send an alert to the Guardian so that Guardian can prevent Control signals from reaching CAN Bus and bring the vehicle to a stop. In that case, Guardian is a Process-Control style.

Conceptual Architecture

Overview

The conceptual architecture of the Apollo system utilizes a growing library of subsystems that include multiple major functionalities that are required for the system to function. By working together in conjunction with each module, Apollo can construct its open software platform for different developers to use and customize.

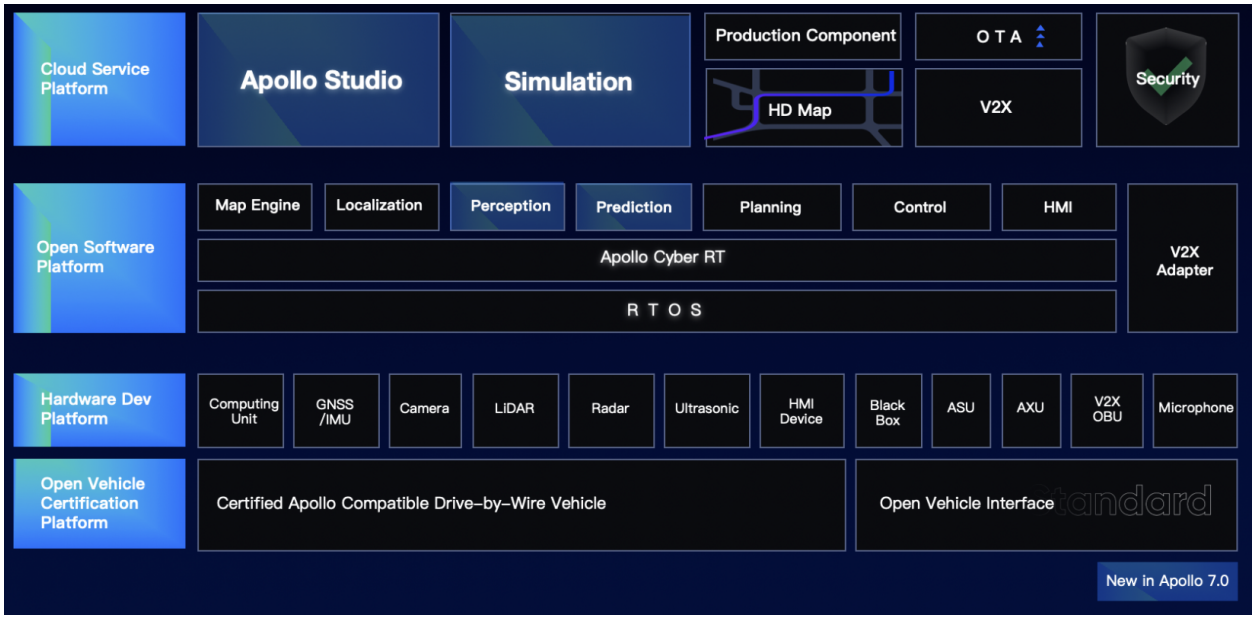

Figure 2. Apollo 7.0 Architecture Overview

Subsystems Breakdown and Interactions

Perception

There are two essential submodules inside perception: obstacle detection and traffic light detection. Two critical functions depended on multiple cameras, front and rear radars and LiDARs, to recognize obstacles and fuse their tracks to obtain a final tracklist. The obstacle sub-module detects, classifies, and tracks obstacles. This sub-module also predicts obstacle motion and position information. We construct lane instances for lane lines by postprocessing lane parsing pixels and calculating the relative lane location to the ego-vehicle. In perception, there are seven inputs: 128 channel LiDAR data,16 channel LiDAR data, Radar data, Image data, Extrinsic parameters of radar sensor calibration, Extrinsic and Intrinsic parameters of front camera calibration, Velocity, and Angular Velocity of the host vehicle. And the perception module outputs are the 3D obstacle tracks with the heading, velocity, and classification information, The output of traffic light detection and recognition. The data obtained by these modules will be passed to Prediction so that the vehicle can make a judgment.

HD Map

The most significant characteristic of an HD map (High definition map) is its accurate and comprehensive representation of road characteristics, which requires higher real-time performance. The difference lies between its usages. HD maps are used for machine processing while people use traditional maps. This allows the system to understand the environment around it with specific road information, such as crosswalks, traffic lights, speed limit signs, lanes, left and right turns, etc. The system can sense the external environment through interaction with the Perception module, collect raw reflection data, label road graphs and overlay intensive maps, and send perceptual processing results to the control module. HD Map also has some human-computer interaction interfaces (HMI).

Localization

This module provides localization services. The essence of localization is the perception of the surrounding environment, feature extraction, feature matching, and finally, getting an accurate positioning result. There are two ways to help the vehicle locate, one is RTK, and the other is multi-sensor fusion. In the RTK localization method, GPS and IMU(Inertial Measurement Unit) are the two primary location-based devices. The GPS provides a rough map of the vehicle related to the vehicle's MAP Engine, while the IMU is a sensor that can accurately locate the vehicle in bad road conditions. Multi-sensor fusion has three inputs, two of which are the same as RTK inputs, and the other input is LiDAR. LiDAR is a more accurate sensor of the vehicle's surroundings, allowing the vehicle to make the best judgment and implementation when driverless. The only drawback is that the accuracy of the range is very low in extreme environments. So these three sensors have to work together. These three sensors pick up the

signal, confirm the vehicle's location and the surrounding road conditions and send the digital information to perception and planning.

Prediction

The behavior of all obstacles detected by the perception module is studied and predicted by the prediction module. Prediction receives obstacle data and basic perception information, including positions, headings, velocities, accelerations, and then generates expected trajectories with probabilities for those obstacles. A significant point that needs to be mentioned here is that the Prediction module only predicts the behavior of obstacles and not the EGO vehicle. On the other hand, the Planning module plans the trajectory of the EGO vehicle. The prediction module comprises four main functionalities: Container, Scenario, Evaluator, and predictor.

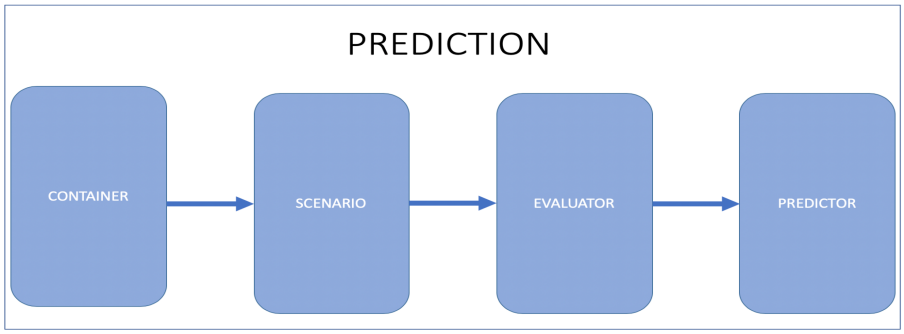

Figure 3. Functionalities of Prediction Module

Container - stores input data from subscribed channels. Currently supported inputs are *perception obstacles*, *vehicle localization* and *vehicle planning*.

Scenario - analyzes the scenarios that include the ego vehicle. There are two defined scenarios: cruise and junction. The cruise includes vehicles that remain in the lane and the vehicle continues at a constant speed at a set speed. Junction is the scene at the intersection is accompanied by traffic lights.

Evaluator **-** predicts path and speed separately for any given obstacle. An evaluator evaluates a path by outputting a probability for it (lane sequence) using the given model stored in. It includes a list of evaluators: Cost evaluator, MLP evaluator, RNN evaluator, Cruise MLP + CNN-1d evaluator, Junction MLP evaluator, Junction Map evaluator, Social Interaction evaluator, Semantic LSTM evaluator, Vectornet LSTM evaluator, Jointly prediction planning evaluator.

Predictor - generates predicted trajectories for obstacles. It includes a list of supported predictors: Empty, Single lane, Lane sequence, Move sequence, Free movement, Regional movement, Junction, Interaction predictor, Extrapolation predictor.

Routing

The routing module is utilized by the planning module and is what generates high-level navigation information based on the requests it receives. Using the control commands it receives, it combines it with a routing topology file and generates a map based on the information it

receives. It outputs the desired map that is then passed onto the planning module which then executes a series of actions.

Planning

The planning module receives data from the perception, prediction, map engines, routing, and localization modules. Its goal is to provide a "collision-free" trajectory for the vehicle to navigate through when it encounters various situations. Examples of the potential situations the system could face are when a stop sign is detected, the vehicle should reduce its velocity and prepare to stop, or if the traffic light is turning yellow, based on its speed and distance to the intersection, should it cross or come to a stop. The planner uses the information passed to it and calculates the system's execution with precision. As it adapts to real-time traffic conditions, the projected trajectories create a seamless transition between the planning and execution. Its extensive database can perform simulations that improve the system's planning process and attempt to make the best decisions with speed and precision.

Control

The handling of the vehicle including acceleration, steering, deceleration, parking, turning, and even pulling over is what the control module is responsible for.

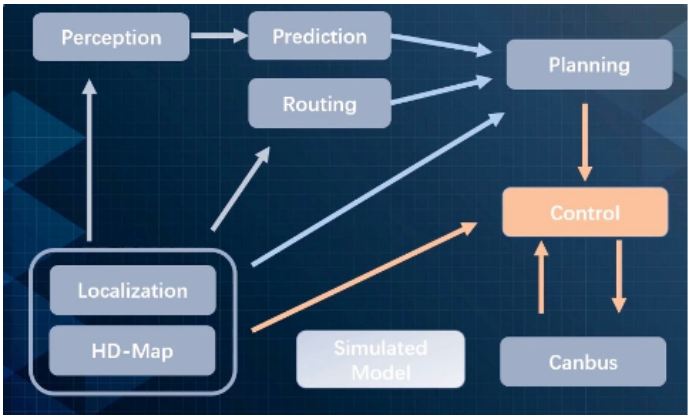

Figure 4. Interactions with Control Module

There are two inputs for the control module: One comes from Planning. In the Planning stage, the intention is what track we want the vehicle to run along. The other comes from Localization and HD Map modules. In the feedback stage, we can get the vehicle's current information under the world coordinate system and vehicle coordinate system, generally including vehicle position, orientation, speed, terrain, and geometry information. The control module mainly outputs control instructions. For example, it interacts with CAN Bus, the lower module, to get feedback on vehicle speed, rotational speed, and health information, such as errors in chassis and fault handling.

The Simulated Model is a vehicle dynamics and kinematics model associated with the control module. If the Control module uses a Model-based algorithm, the system vehicle needs to be modeled first.

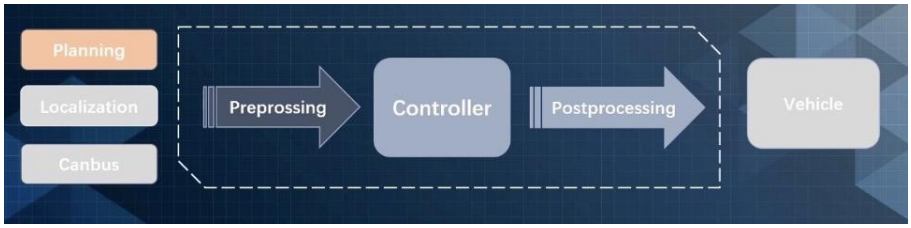

Figure 5. Control Module Breakdowns

The control module includes preprocessing, controller, and postprocessing.

Preprocessing:

1. Bad inputs rejection and fall back
2. Emergency handling
3. Filters/Observers for
    a. Signal smoothing
    b. Lag compensation
    c. Loop shaping

Controller:

1. Modeling
2. System Identification and analysis
3. Controller/Observer Design
4. Parameter tuning

Post Processing:

1. Saturation/Limitation handling
2. Signal Filtering
    a. Signal Smoothing
    b. Gain attenuation

CAN Bus

CAN Bus is the module that takes in commands and tasks from the control module and is responsible for executing them. Working hand in hand with the control module, the CAN Bus monitors whether or not the actions are being executed. Movements of the vehicle chassis are recorded as inputs and provide feedback to make adjustments based on the commands it is given.

Monitor

The monitor receives data from Perception, Prediction, Planning Control, HD map, Localization, and other various modules, then transfers the data flows to the HMI for the driver to view. The function of the monitor is to supervise the hardware and software components to ensure that all the modules are working without any issue. If it detects a problem, the monitor will send an alert to Guardian for further steps to prevent a crash.

Guardian

Guardian is the safety center of the whole system. Guardian receives data flows from Monitor to ensure all other subsystems are working well. If the data from the monitor is detected failure, Guardian will prevent the CAN Bus from executing further actions and cut down the connection between Control and CAN Bus, and then Guardian will take actions based on the response of the Ultrasonic sensor and the HMI to choose one of the three preset executing modes.

HMI

HMI is the human-machine interaction that connects the user with the system. The GUI is what bridges the connection and provides a graphical representation of the system that allows the user to utilize its functions. HMI is a python application based on Flask and uses web sockets to query ROS modules and applications. It executes a subprocess to execute the corresponding binary when it receives a command or task.

Use Cases

The following are some use cases scenarios to see how each component communicates with each other.

Use Case 1: Automatic Rerouting (e.g. encounter a road closure)

For use case one, the process begins when the Perception module detects an obstacle in front of the vehicle, then the Prediction module makes predictions about the obstacle's movement and determines that this obstacle remains in its position (it is a road closure block). HD Map shows that there is a road sign advising vehicles to divert. The Localization module then sends the current location to both Routing and Planning modules. According to the original designated destination and the vehicle's current location, the Routing module computes the passage lanes and roads, while the Planning module decides which path to go. After that, the Planning module sends the trajectory to the Control module to process and give command signals. Before the control signals reach the CAN Bus, Monitor will keep track of all the modules and ensure they are working without any issue. In a module or hardware failure, the monitor alerts Guardian, preventing Control signals from reaching CAN Bus and bringing the vehicle to a stop. CAN Bus will execute the commands if no failure occurs and the vehicle finishes the rerouting process.

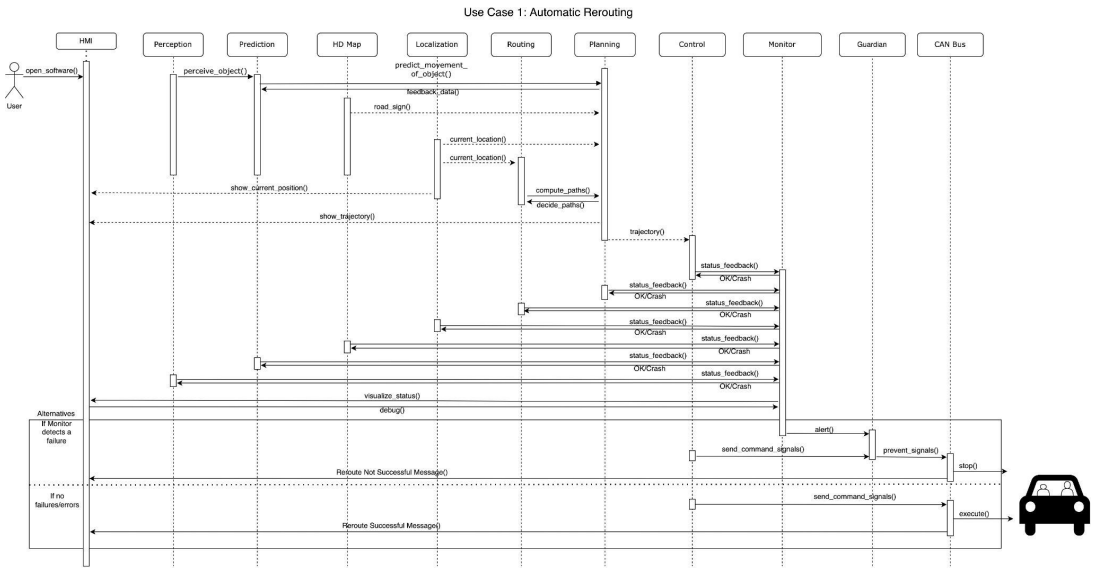

Figure 6. Sequence Diagram of Use Case 1: Automatic Rerouting

Use Case 2: Automated Valet Parking

The second use case is Valet Parking. Since there are pain points like manual parking is difficult, extended parking waiting time, etc. Apollo provides Automated Valet Parking which allows the user to give a command of parking on his/her phone to start. Localization gives the environmental information and the current position of the vehicle in order to find a parking space. Then Perception detects obstacles including a look around perception (UPA Ultrasonic Radar, APA Ultrasonic Radar, fisheye camera), forward perception (Monocular camera), backward perception (Monocular camera). Prediction receives obstacle data and basic perception information and generates predicted trajectories with probabilities for those obstacles. Planning modules use the information collected from perception, prediction, and localization modules to provide a "collision-free" trajectory for the vehicle to navigate through when it encounters various situations. Planning will receive parking space information (length and width of the space) from the perception module and decide if the available parking space is big enough, if it is not, the car will leave to find another parking space. Control modules get the input information from planning, then give tasks to routing to output the desired map. Before the control signals reach the CAN Bus, the monitor will keep track of all the modules and ensure they are working without any issue. In a module or hardware failure, the monitor alerts Guardian, preventing Control signals from reaching CAN Bus and bringing the vehicle to a stop. CAN Bus will execute the commands if no failure occurs and the vehicle finishes the rerouting process.

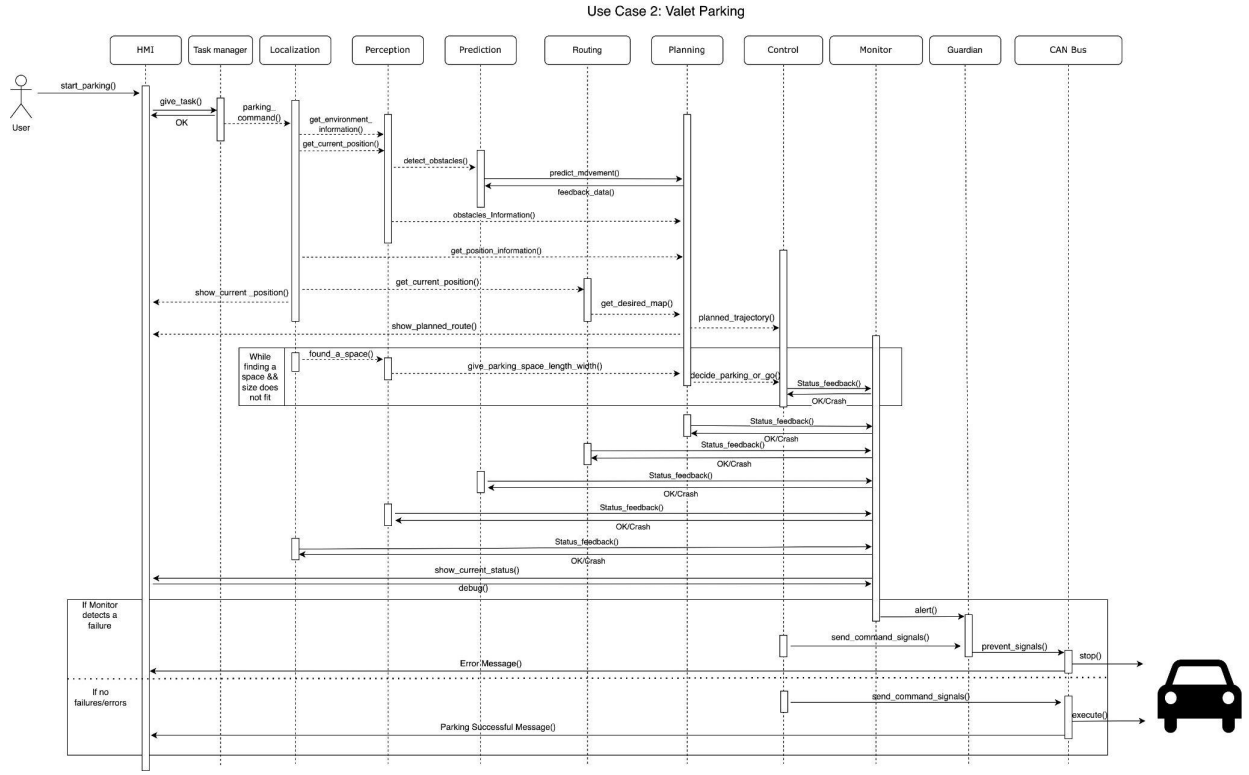

Figure 7. Sequence Diagram of Use Case 2: Valet Parking

Version Control (Evolution)

Apollo-v1.0-v2.0

These are the initial two versions that were developed and the difference between them is the implementation of its autonomous driving capabilities. Version 1 did not include any autonomous driving capabilities, instead, it was known as the Automatic GPS Waypoint Following, due to its only functionality of tracking the vehicle's path and the speed at which it travels. The improvement for version 2 was that it allowed the vehicle to cruise and drive on roads using its integrated HMI design, camera, radar, and drivers/modules.

Apollo-v3.0-v5.0

This is the stage at which the software transitions from a product to a platform, allowing its contracted partners to be flexible and adapt the system to their hardware. Safety and surveillance modules were adjusted and improved as well as additional visibility cameras, enabling the system to have a full 360-degree visibility spectrum which increases the perception of the deep learning model to adapt and learn according to various environments and weather conditions. New pipeline data services were also added as calibrations and weights in the deep learning model were tweaked.

Apollo-v6.0-v7.0

Various new deep-learning models enabled the system to enhance its abilities in obstacle detection, semantic map building, trajectory planning which allowed for an increase in smart training and evaluation. Reinforcement learning was also introduced in the most recent version and the large influx in data collection allows the new deep-learning models to improve and build a steady foundation for the system.

Data & Control Flow

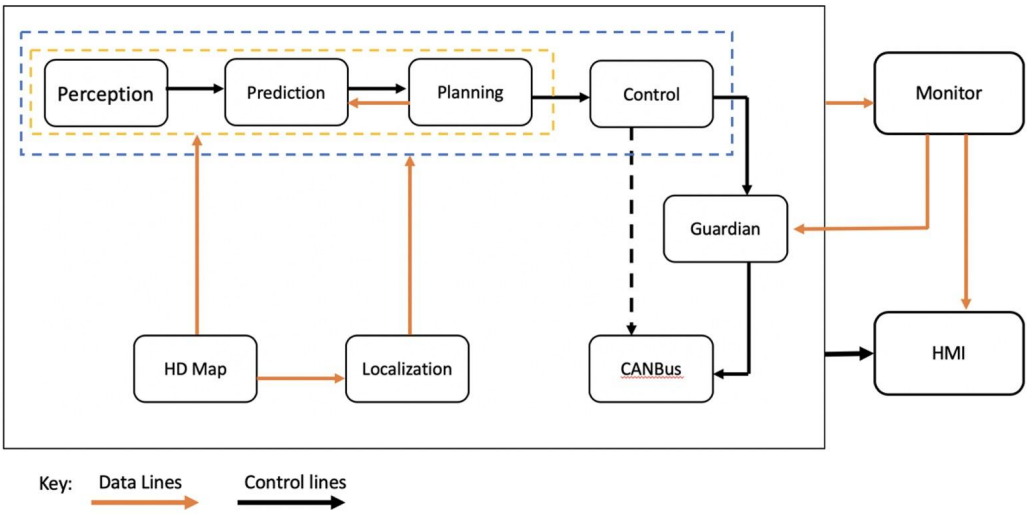

Figure 8. Data & Control Flow

Data Flows: When an autonomous vehicle traveled on the road, HD Map collected real-time road and traffic information and provided it to Perception, Prediction, Planning, and Localization Modules. The Localization Module locates the vehicle's location on the earth and provides the data to the Perception, Prediction, Planning, Localization, and Control modules. The Planning Module plans routes for the vehicles and provides feedback data to the Prediction Module. Monitor receives data from different modules and passes them on to HMI for the driver to view and ensure that all the modules are working without any issue. In a module or hardware failure, the monitor sends an alert to Guardian (new Action Center Module), which then decides what needs to take action to prevent a crash.

Control Flows: The Perception Module controls the Prediction Module to predict the movement of the perceived obstacles after obtaining the vehicle and road information provided by HD Map and Localization. After making projections, the Prediction Module will control the Planning Module to plan the vehicle's route. The Planning Module will then provide the starting and ending information of the path to the Routing Module and hand it over to the Control Module to process, which controls the CAN Bus to execute. If the data from the monitor is detected failure, Guardian will prevent the CAN Bus from further actions and cut down the connection between Control and CAN Bus, and then Guardian will take actions based on the response of the Ultrasonic sensor and the HMI to choose one of the three preset executing modes.

Responsibilities among participating developers

Since Apollo is committed to providing an open platform for all the stakeholders who are interested in a vibrant autonomous driving ecosystem, developers from all aspects are playing an essential role in building such grand progress. Developers need to take different responsibilities based on their roles to help themselves and other developers focus on their specific areas of expertise, and also in case of wasting both time and resources. Among all the key stakeholders, there are three key roles in software development and their corresponding responsibilities.

- The Project Manager is responsible for effectively communicating with all the other members and also scheduling for the project (around six month/update). Communication is one of the key elements in collaboration, Project Manager needs to have good management and help other project members to overcome risks and issues.
- Software Developers are responsible to meet business requirements using technical requirements. Therefore, for all Software Developers, coding with several or specific programming languages is required. Besides, Apollo has also provided a series of lessons on autonomous vehicles. Software Developers also need basic knowledge of autonomous vehicles.
- Software Testers are responsible to ensure that the software solution meets the business requirements and that it is free of bugs, errors, and defects. Software Testers are mainly involved in the testing phase. They need to set up different environments to evaluate the results and document problems found.

After the final testing phase, developers should also be responsible for the deployment of their progress, and the Apollo requires maintenance from fibre attacks.
Precise division of responsibilities is the key to completing the project both efficiently and effectively.

Data Dictionary

- **CAN Bus (Controller Area Network bus)**: Cassis, a vehicle communication protocol.
- **Evaluators:** probabilities calculated using the specified models
- **Cruise MLP + CNN-1d evaluator**: probability is calculated using a mix of MLP and CNN-1d models for the cruise scenario.
- **Junction MLP evaluator**: probability is calculated using an MLP model for junction scenario.
- **Junction Map evaluator**: probability is calculated using a semantic map-based CNN model for junction scenarios. This evaluator was created for caution-level obstacles.
- **Social Interaction evaluator**: this model is used for pedestrians, for short-term trajectory prediction. It uses social LSTM. This evaluator was created for caution-level obstacles.
- **Semantic LSTM evaluator**: this evaluator is used in the new Caution Obstacle model to generate short-term trajectory points which are calculated using CNN and LSTM. Both vehicles and pedestrians are using this same model, but with different parameters.
- **Vector net LSTM evaluator**: this evaluator is used in place of a Semantic LSTM evaluator to generate short-term trajectory points for "Caution" tagged obstacles. More detail is in vector net lstm evaluator readme.
- **Jointly prediction planning evaluator**: this evaluator is used in the new Interactive Obstacle(vehicle-type) model to generate short-term trajectory points which are calculated using Vectornet and LSTM. By considering ADC's trajectory info, the obstacle trajectory prediction can be more accurate under interaction scenarios.
- **Lane sequence**: obstacle moves along the lanes
- **Move sequence**: obstacle moves along the lanes by following its kinetic pattern
- **Regional movement**: obstacle moves in a possible region
- **Junction**: Obstacles move toward junction exits with high probabilities
- **Interaction predictor**: compute the likelihood to create posterior prediction results after all evaluators have run. This predictor was created for caution level obstacles
- **Extrapolation predictor**: extends the Semantic LSTM evaluator's results to create an 8-second trajectory.

Lessons Learned

In this assignment, we need to analyze Apollo and its software related to unmanned driving. Analyzing the software and stripping the framework from the overall structure, we can accurately pinpoint the conceptual architecture of the system. The interaction and data flow

between various components in the system were taken apart and dissected. Most of the information we extracted was from the official Apollo GitHub and its surrounding subsidiaries and informational websites. Throughout the assignment, multiple obstacles impeded the collection of information but were resolved when having meetings and working together to find a solution together. Scholarly articles and presentations held by Apollo executives were external resources that we ended up citing and referencing. The complexity of the system proved to be difficult to properly break apart and synthesizing our analysis of an existing system was a task that needed to be overcome. Although weekly meetings and emergency meetings were held, there were still some communication issues as we weren't all very familiar with each other. In this assignment, we learned the importance of cooperation, brainstorming can achieve twice the result with half the effort, and rationally utilizing the knowledge learned in class.

Conclusion

In conclusion, the Apollo system was something we needed to tackle and draw different perspectives to accurately describe and analyze. The complexity of the autonomous driving system consisted of numerous modules and the way data flows within its system. The evolution of the system has shown how its modules and subsystems have improved over the years. Its approach in making its system open source and modifiable allows its partners to utilize and customize the systems according to their own needs. The use of multiple architecture styles allows the system to be efficient and stable while integrating high-end technology into its systems and modules.

References

Apollo Auto. "Apollo 5.0 Technical Deep Dive." Medium, Apollo Auto, 3 July 2019, https://medium.com/apollo-auto/apollo-5-0-technical-deep-dive-d41ac74a23f9.

"Apollo Governance." Apollo, https://apollo.auto/docs/manifesto.html.

ApolloAuto. "ApolloAuto/Apollo: An Open Autonomous Driving Platform." GitHub, https://github.com/ApolloAuto/apollo.

Ma, Changjie. "Baidu ApolloHD Map - UN-GGIM." Intelligent Transportation and Autonomous Vehicles, https://ggim.un.org/unwgic/presentations/2.2_Ma_Changjie.pdf.

Swords, Simon. "Software Development Team Roles and Responsibilities: Atlas." Atlas Computer Systems Ltd, 19 Jan. 2020, https://www.atlascode.com/blog/software-development-project-roles-and-responsibilities/#SOFTWARE_DEVELOPERS.

WEI, Hao. "How Baidu Apollo Builds HD (High-Definition) Maps for Autonomous Vehicles." Medium, Towards Data Science, 27 Oct. 2021, https://towardsdatascience.com/how-baidu-apollo-builds-hd-high-definition-maps-for-autonomous-vehicles-167af3a3fea3.