CISC 322/326
Assignment 1: Conceptual Architecture of Apollo

February 18, 2022

Sebastian Wakefield	19sslw@queensu.ca
Dominic Guzzo	18dpg3@queensu.ca
Oscar Wojtal	18ow2@queensu.ca
Alice Cehic	17ac121@queensu.ca
Cooper Lloyd	17cel4@queensu.ca
Daniil Pavlov	17dp15@queensu.ca

Table of Contents

Abstract

This report will illustrate our conceptual architecture findings of ApolloAuto's open-source driving platform. It provides a comprehensive, safe, secure, and reliable solution that supports all major features and functions of an autonomous vehicle. From our research, we have determined that ApolloAuto uses an amalgamation of process control and publish-subscribe architectural styles. This is based on the knowledge that Apollo's functionality is implemented through modules interacting with each other using the CyberRT system. We also look more into the different components that make up the conceptual architecture including the Perception, Prediction, Planning, HD-MAP, Localization, Control, Human-Machine interface, Routing, and Guardian modules.

Introduction

Announced in April 2017, Apollo is an open-source high-performance architecture with the primary purpose of becoming a vibrant self-driving driving ecosystem. ApolloAuto is able to autonomously control an automobile in real-time. The system recognizes fixed and moving obstacles surrounding the vehicle, traffic signals, and predicts the movement of obstacles. It plans and issues vehicle control commands based on these predictions. This report specifically looks at the software components of Apollo 7.0, which was released in December 2021. The major updates for 7.0 include brand new deep learning models, Apollo Studio Services, PnC reinforcement learning services and an upgraded Perception module code structure. The conceptual architecture found in this report is a result of the documentation mainly found on the ApolloAuto Github repository and website, with reference to Behere's paper, A functional reference architecture for autonomous driving (see references).

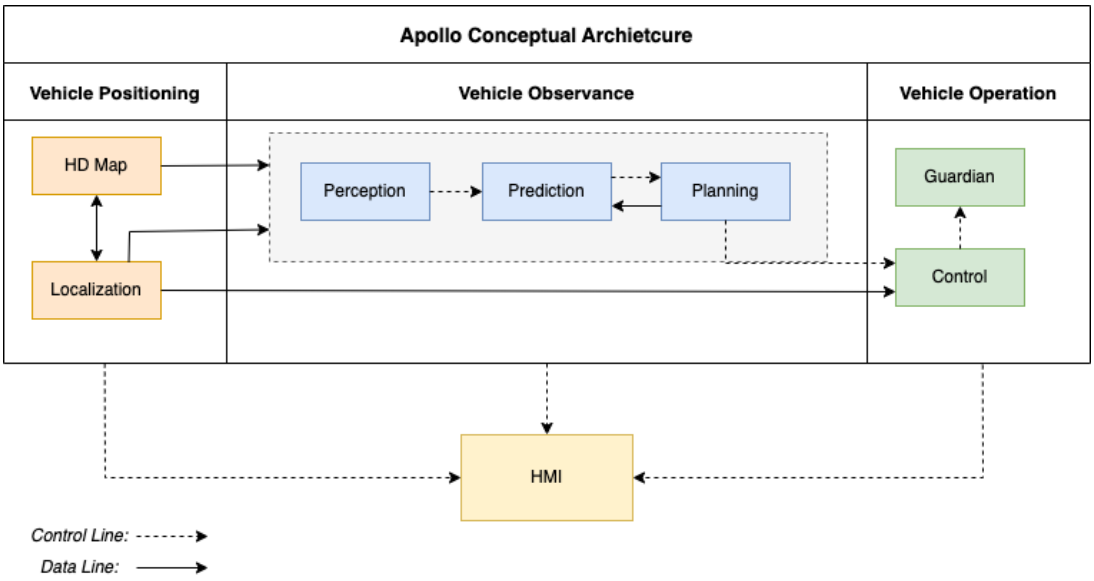
The group began its work on the report by individually reviewing Apollo and its architecture. We decided this approach would allow us to broaden our ideas as each group member could gather info that the others might have missed. We converged our thinking in the end by discussing what architectural styles we gathered and deliberated until a final consensus was reached. The rest of the workload was split based on the main written sections, that is the introduction/conclusion, conceptual architecture, evolution, diagrams and use cases. We collectively worked in the same Google Documents folder to share research and see the progress of one another. Initially, meetings were held once a week, although the frequency increased as we approached the due date. At the meetings, the group would discuss any questions about their individual tasks and review each other's parts. In the final stretch of the project, the focus of meetings was shifted to involve everyone in the proofreading process to ensure that the final product was perfect and reflected the group's comprehension of the system.

The report will go into depth analyzing the conceptual architecture styles, main modules and use cases of ApolloAuto. The structure of the system is found to utilize a mix of process-control and publish-subscribe architectural styles. We believe this is the case as process-control provides the ability to provide real-time control, decompose the system into subsystems and its capability to perform feedback loops. By incorporating the publish-subscribe style to allow submodules to receive and communicate information, it allows each part to grab the information relevant to them, along with the ability to enable and trigger components. This report will also focus on the main modules that construct the system, which are the Perception, Prediction, Planning, HD-MAP, Localization, Control, HMI (Human Machine Interface), Routing, and Guardian modules. The interaction between the modules is demonstrated by diagrams and reviewing two use cases in which the autonomous

vehicle finds itself driving on an expressway and valet parking. Finally, the report will discuss the lessons learned by the group during the research and development of the conceptual architecture.

Conceptual Architecture

The interaction of components within Apollo's architecture can be best illustrated by the Conceptual Architecture below,



Since the interaction between the main components of the system is organized using the CyberRT system (in previous versions of ApolloAuto - ROS), each module of the system has an independent implementation and can be registered in CyberRT as a publisher or subscriber of messages. It can also exchange data with other modules using the CyberRT parameter service. Functionally, ApolloAuto controls the vehicle in the following sequence:
1) Determine the current position of the automobile relative to the environment as in obstacles, other vehicles, the road, and traffic lights.
2) Predict the behaviour of obstacles and vehicles on the road based on the computation of their current positions, possible speeds, and continuous re-evaluation of their movement.
3) Plan the required changes in the route and speed of the automobile in accordance with the following exigencies: endpoint of the route, current state of the vehicle, the position on the 3D map, and the GPS position.
4) Formation of vehicle and vehicle device control commands and their transmission via CANBus.
5) Continuous monitoring of the state of the automobile and the forced cessation of movement in case of malfunctions.

ApolloAuto consists of modules that interact with each other through the CyberRT system. Each module interacts with external sensors and/or shares data with other modules. The sensors of the system are camera, lidar, radar, GPS, microphone, compass, gyroscope. Data from the lidar and cameras are processed using Deep Neural Network technology, supported by hardware such as NVIDIA CUDA, which allows for high-performance computation. By processing the results, the system is able to determine the positions of objects in the image (lidar distance map) and their preliminary classification (whether it's a traffic light, car, person, road, etc.). Next, the movement of objects relative to the automobile is predicted and the control subsystem comes in to handle braking

and steering. The required changes of the route are planned and commands are issued to the vehicle's devices.

Architectural Style

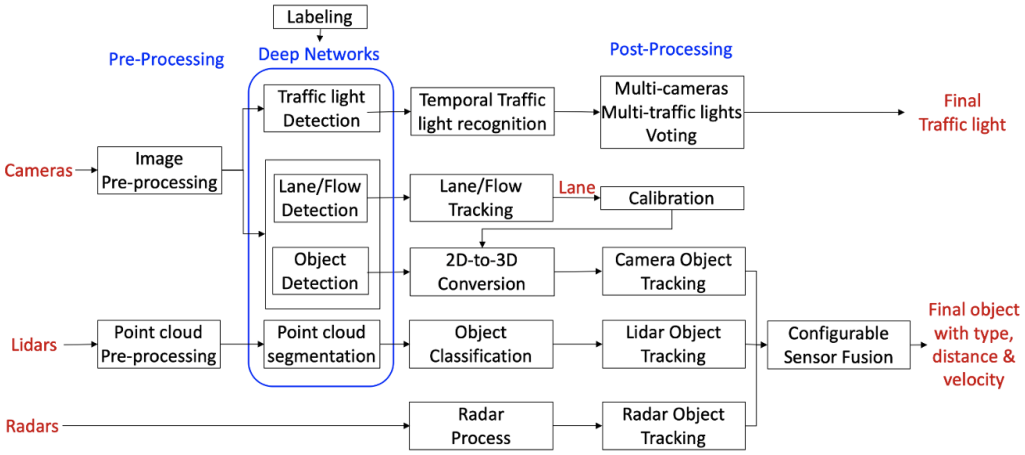
The conceptual architectural style that we believe best suits Apollo's software architecture is a mix of process-control and publish-subscribe. We chose process-control as a plausible style because many hardware components that the software must control are real-time systems and provide highly critical functions. For example, process-control architectural style makes the most sense for the cars real-time steering and braking. Moreover, Apollo needs an architecture that is able to decompose the entire system into subsystems or modules and connect them, and process-control provides great infrastructure for this. Each subsystem will also benefit from using process-control feedback loops, where there are sensors which monitor gathered information, a controller which handles logic, and an actuator which manipulates the hardware or physical component. These major components of a process-control feedback loop will give constant adjustment of process variables and keep the car ready to react at any moment. We additionally chose a publish-subscribe architectural style because of the system's capability to constantly communicate with many components which have to enable and trigger other components. The publish-subscribe architectural style provides great infrastructure for the perception system to trigger the control system as well as the planning system. If an obstacle appears on any side of the car, the control system must be notified of the event and triggered to manoeuvre out of the way after consulting the planning system which received the predicted obstacle's movement. Therefore, using this architecture, these systems can register an interest in the perception component and be invoked whenever there is a threat to the car's safety. This implicit invocation allows for speedy interactions between components and allows constant communication of the software architecture subsystems.

Perception Module

The perception module incorporates the capability of using 5 cameras (2 front, 2 on either side and 1 rear) and 2 radars (front and rear) along with 3 16-line LiDARs (2 rear and 1 front) and 1 128-line LiDAR to recognize obstacles and fuse their individual tracks to obtain a final tracklist. The obstacle module detects, classifies and tracks obstacles, in addition to predicting the obstacles motion and position information (e.g., heading and velocity).

This module is of the layered architectural style since it implements data transformation within several layers of abstractions. At the same time, each layer interacts only with the neighbouring one, sequentially receiving, transforming, and transmitting data to the next layer of abstractions. The following layers are present: Data Pre-Processing, Labeling, and Data Analysis (Post-Processing). Data Pre-Processing receives data from sensors (such as cameras, lidar, radars) in digital form, preliminarily processes them (with procedures such as position alignment, colour calibration, contrast enhancement) for their standardization (calibration), bringing them to a form suitable for further processing. The next layer is Labeling, which is implemented using Deep Neural Networks and is used to segment, classify, and label the data received from the previous layer. Thus, we get the positions and types of objects on the image or on the lidar distance map. The Deep Neural Network technology provides segmentation, classification, and labelling of data in a fixed time, unlike algorithmic methods, although it requires significant computing resources and large amounts of data. Computing resources are provided by the use of hardware accelerators such as NVIDIA CUDA. Deep Neural Networks training data is provided by Baidu. The third layer, Data Pre-Processing, performs further analysis of segmented and classified data by algorithmic methods

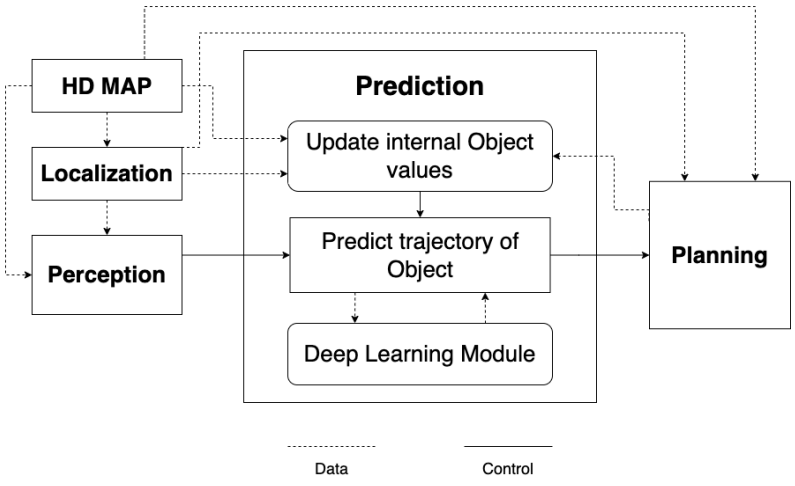
in order to obtain information about traffic lights, and moving and stationary objects in the field of view of vehicle sensors.



Additionally, the Perception module provides sensor calibration. Each sensor has its own production tolerances and installation features. Therefore, they can produce different information in the same situations. Thus, the sensor calibration possibility has been introduced, and at the Post-Processing stage, individual sensor features can be taken into account.

One of the main functions of Perception is Closest In-Path Object (CIPO) detection, vanishing point detection, and recognition of traffic signals. As an example, we will analyze the operation of the traffic signal recognition function. The recognition of traffic signals begins with obtaining a pre-processed image with information on object frames and their preliminary classification. Next, normalizing the positions of objects recognized as traffic lights vertically and horizontally. Finally, recognition of traffic signals taking into account previous states (a flashing signal).

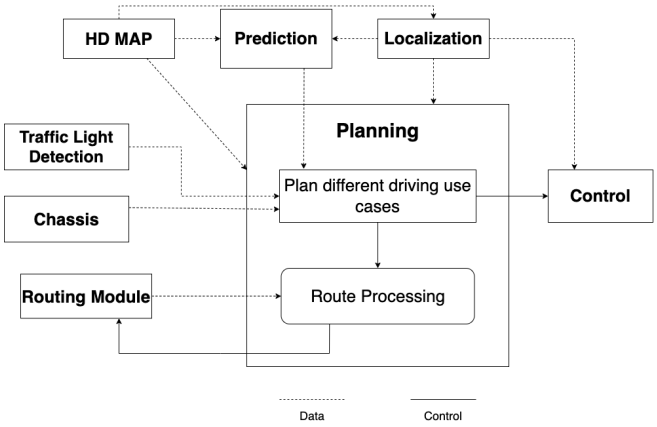
Prediction Module



The Prediction module follows a publish-subscribe style of architecture where it subscribes to the data published by the Perception, HD Map, Localization, and Planning modules. The Prediction subsystem uses this information in order to process possible velocities and directions of the objects currently in it's memory and sent by the Perception module. More specifically, it constantly updates it's saved objects trajectories with the information published by the aforementioned modules, although the prediction only triggers when it is sent an object by the Perception module. New to Apollo 7.0, the Prediction now incorporates a Deep Learning module that helps make more reliable predictions

regarding the possible trajectories. Once the prediction information is processed, it is sent down the process-control chain to the planning module.

Planning

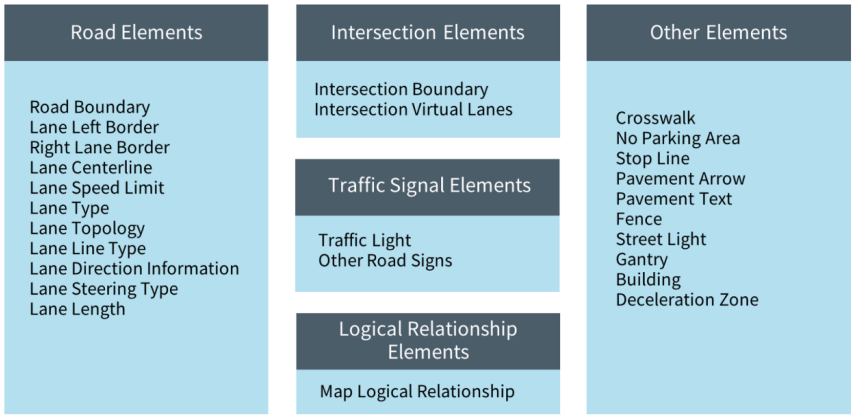


The Planning module follows a similar publish-subscribe architecture as the previous Prediction component. This system takes the bulk of it's info from the Prediction module along with the information published by the HD MAP, Localization, and chassis modules, in order to plan different driving use case scenarios. An improvement over earlier versions of Apollo, the Planning unit now considers multiple driving scenarios as opposed to a single one in order to improve on the flexibility of it's options; if a problem arises with a specific scenario, it can now be fixed without affecting the other possible plans. Since the Prediction module wraps the information passed by the Perception module, it truncates the instructions of traffic lights. Therefore, the Planning unit must also subscribe to the data published by the traffic light detection module in order to incorporate the information in its calculations. The planning module additionally takes the data from the Routing module and analyzes whether the route is possible. If not, it triggers a new routing request back to the routing module.The planning module uses the information passed by these modules in order to create various different driving scenarios. These scenarios are updated, fixed, and erased depending on the continuous information it receives from each of these sources. This information is then sent to the Control Module.

HD-Map

This module is similar to a library. Instead of publishing and subscribing messages, it frequently functions as query engine support to provide ad-hoc structured information regarding the roads. Apollo HD maps employ the OpenDRIVE format, an industry-wide mapping standard. It functions as an API that makes it easy for everyone to read the same map data.

OpenDRIVE API readable data:



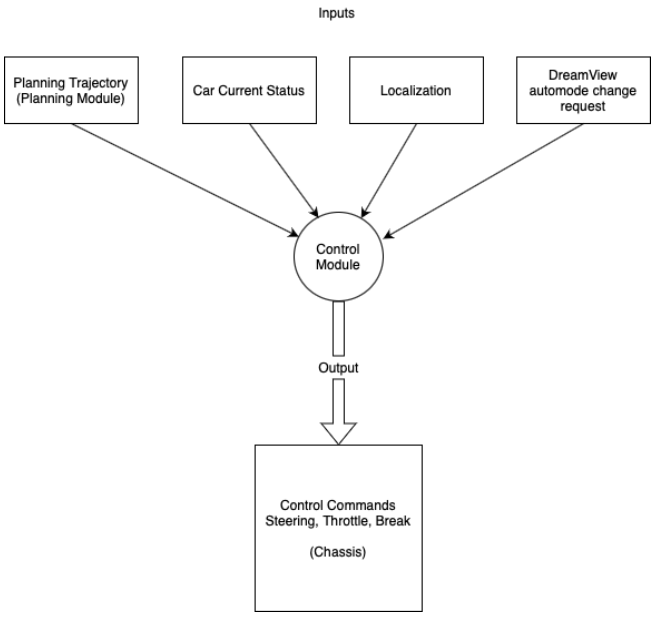
While the overall structure is similar, Apollo's modified OpenDRIVE is very different from the normal OpenDRIVE standard, as they have changed several fundamental details, for instance,

how roads are represented. Apollo uses this classification API along with GPS data and their perception module to configure the HD map so that it can navigate the road. A high-definition map helps the vehicle find suitable driving space. For example, a high-definition map helps the vehicle identify the exact centerline of a lane on the road, that way, the vehicle can drive as close as possible to the centre as well as help the vehicle narrow its options so that it can select the best manoeuvre.

Localization

The localization module aggregates various data to locate the autonomous vehicle using callback functions and timers to attain the car's location in real-time. The benefit of creating this module is that the car's position relative to exterior objects can be computed so that the car is always at a safe distance from these obstacles. The localization of the car must have single-digit centimetre accuracy and is vital for many of the car's main utilities such as object avoidance and parking. This module leverages various information sources such as GPS, IMU, and LiDAR to estimate where the autonomous vehicle is located. The GNSS, which is a kind of GPS, refers to a constellation of satellites providing signals from space that transmit positioning and timing data to GNSS receivers. The receivers then use this data to determine location. An inertial measurement unit (IMU) is an electronic device that measures and reports a body's specific force, angular rate, and sometimes the orientation of the body, using a combination of accelerometers, gyroscopes, and sometimes magnetometers. We can then combine GPS and IMU data to localize the car. On one hand, IMU compensates for the low update frequency of GPS. On the other hand, GPS corrects the IMU's motion errors. With LiDAR, we can localize a car by means of point cloud matching. This method continuously matches the detected data from LiDAR sensors with the pre-existing HD map. This requires constantly updating the HD map which is extremely difficult.

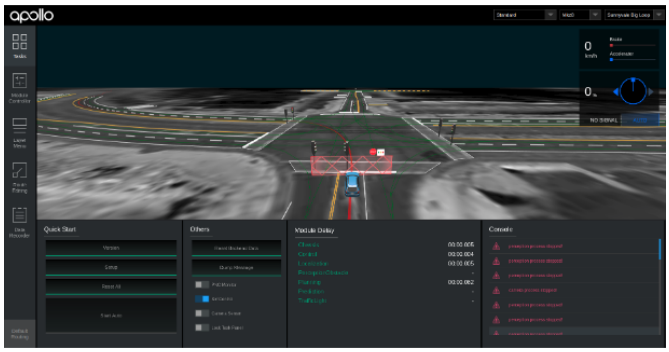
Control



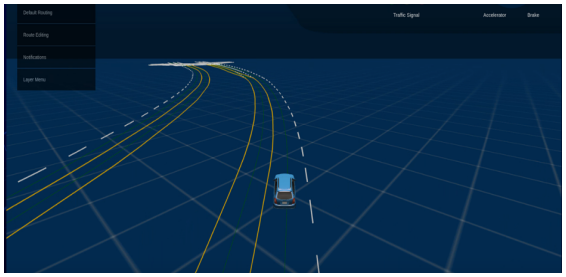
The Control component uses various related algorithms to generate a realistic driving experience. It accepts information like planning trajectory, car status, localization, and Dreamview AUTO mode change requests to generate control commands such as steering, throttle, and braking to then deliver the output to the CANBus component. CANBus is an interface that sends chassis information to the software system and controls the vehicle hardware through the Guardian module. The multiple interfaces in the Apollo software control component combine to produce a controlled autonomous vehicle simulation, based on the planned trajectory and the vehicle's status.

HMI (Human Machine Interface)

The HMI, also known as Dreamview in Apollo, is a web application used for checking vehicle status, controlling vehicle operation in real-time, and testing other modules. It provides a visual of the current output of relevant autonomous driving modules, such as planning trajectory, car localization and chassis status. It provides control options that allow the user to drive the autonomous vehicle and includes a debugging setting which tracks issues in the modules.



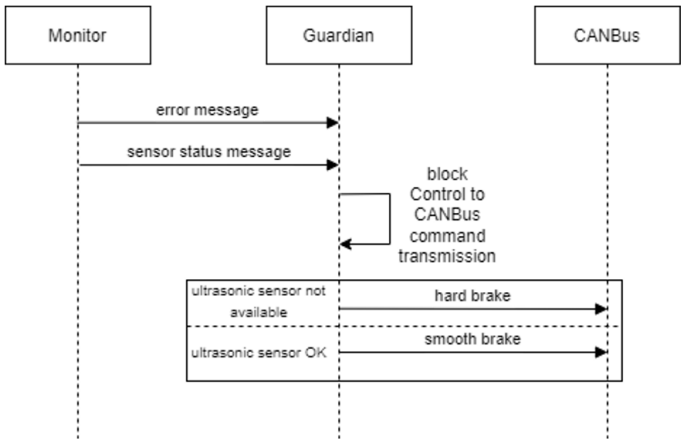
Routing



The Routing module conducts the autonomous vehicle along roads using the routing topology file. It creates the driving instructions using the collected map data and start/endpoints. The output of this module is taken by the Planning component, which could initiate a new routing request if the route assigned to it is unfulfillable.

Guardian module

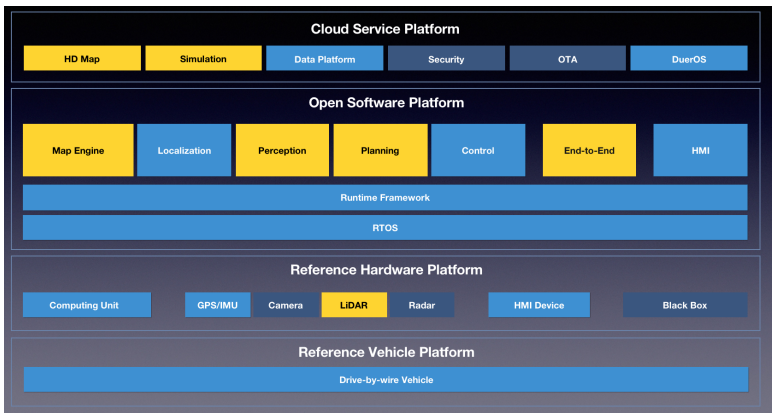
This is a safety unit that ensures a safe stop of the automobile if messages (events) on problems (errors, malfunctions) are received from the Monitor module. If a failure is detected, Guardian will ensure that control signals to the CANBus are terminated and the vehicle will be brought to a safe stop. The automobile can be stopped smoothly or abruptly depending on the current state of the ultrasonic sensor and the driver's influence on the control elements (steering wheel, pedals) in the last 10 seconds.



Evolution of Apollo

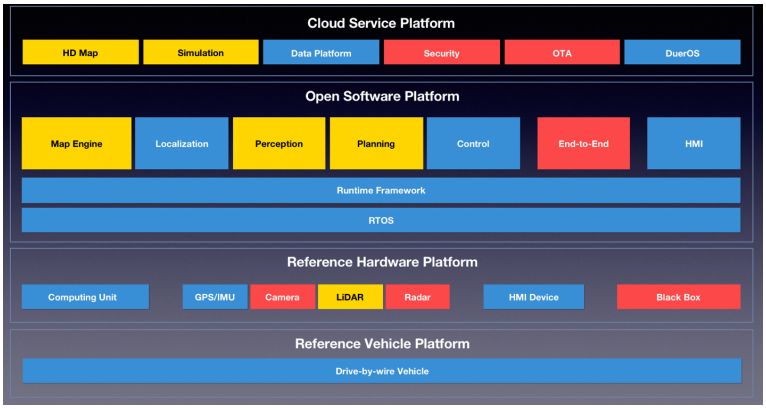
Apollo 1.0/1.5:

Referred to as the Automatic GPS Waypoint Following, Apollo 1.0 Closed Venue Autonomous Driving, introduced software that allowed the autonomous vehicle to work in enclosed spaces like parking lots or test tracks. They used this stage to ensure the software worked



seamlessly before entering public areas. In Apollo 1.5 Fixed Lane Autonomous Driving, the addition of software components like Map Engine, Perception, Planning, and End-to-End, gave the vehicle a better perception of its surroundings. Allowing it to plan a safe trajectory for manoeuvring in its own lane on simple city roads.

Apollo 2.0/2.5:



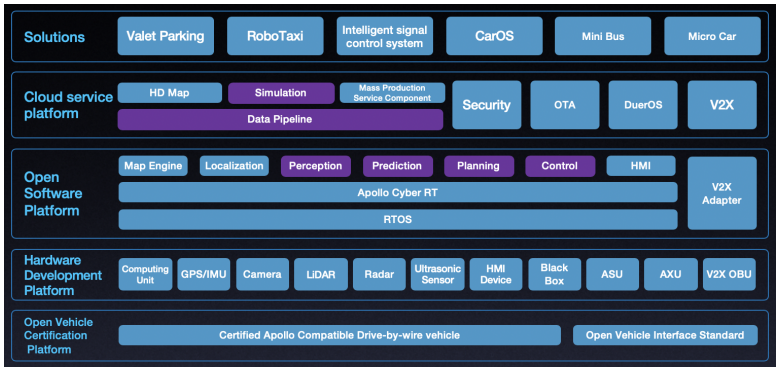
In Apollo 2.0 Autonomous Driving on Simple Urban Road, vehicles can safely ride on roads, avoid collisions, and navigate the roads by changing lanes and stopping at traffic lights. In this version, the End-to-End software component was updated, in addition to the camera, radar, and block box hardware to accommodate for the new vehicle abilities. Updates on Perception and Planning components lead to Apollo 2.5 Geo-Fenced Highway Autonomous Driving, where the vehicle is now able to navigate geo-fenced highways, while maintaining lane control and collision avoidance using the updated camera for obstacle detection.

Apollo 3.0/3.5:

In Apollo 3.0: Production-Level Closed Venue Autonomous Driving, the main focus was to build upon the successful closed-venue low-speed environment testing by creating a mass production solution. No major changes or updates were done to the software in this version. For Apollo 3.5 City Urban Road Autonomous Driving, major updates were made to the Apollo Cyber RT Framework, V2X Adapter, Localization, Perception, and Planning. This software upgrade allowed the vehicle to navigate through complex driving scenarios like residential and downtown areas. In this version, the vehicle now has a 360-degree visibility range and has upgraded awareness and security to handle changing road conditions.

Cloud Service Platform	HD Map	Simulation	Data Platform	Security	OTA	DuerOS	Volume Production Service Components	V2X Roadside Service	
Open Software Platform	Map Engine	Localization	Perception	Planning	Control	End-to-End	HMI	V2X Adapter	
	Apollo Cyber RT Framework								
	RTOS								
Hardware Development Platform	Computing Unit	GPS/IMU	Camera	LiDAR	Radar	Ultrasonic Sensor	HMI Device	Black Box	Apollo Sensor Unit	Apollo Extension Unit	V2X OBU
Open Vehicle Certificate Platform	Certified Apollo Compatible Drive-by-wire Vehicle								Open Vehicle Interface Standard		

Apollo 5.0/5.5:

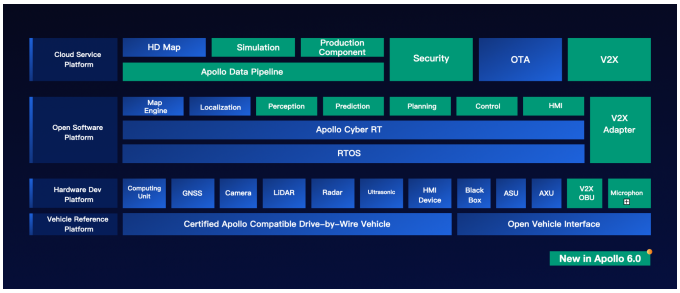


Apollo 5.0: Autonomous Driving Empowering Production, was created to support volume production of Geo-Fenced Autonomous Vehicles. It also brought enhanced scenario-based planning like pull over and crossing bare intersections. Updates

were done to most software components in this version. Apollo 5.5: Curb-to-Curb Urban Road Autonomous Driving, introduces enhanced complex urban road capabilities by presenting curb-to-curb driving support. This version brings Apollo much closer to a fully autonomous urban road driving solution, with an updated deep learning model in the Perception component and a new prediction model that handles complex roads and junction scenarios.

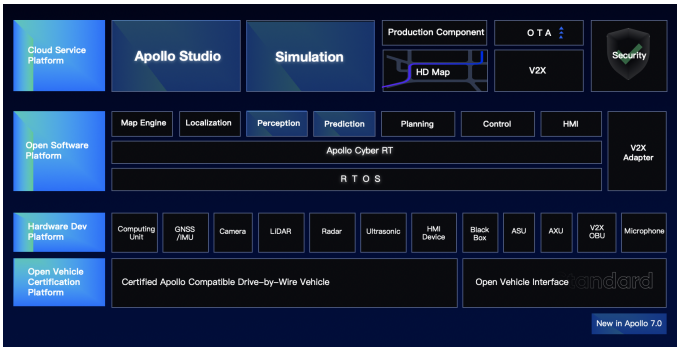
Apollo 6.0:

Apollo 6.0: Towards Driverless Driving, includes new deep learning models to enhance the capabilities of certain modules and introduces new data pipeline services to support Apollo developers. This version is the first version to integrate features that demonstrate Apollo's exploration efforts towards driverless technology.



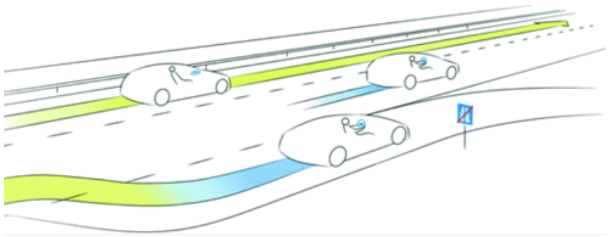
Apollo 7.0:

Apollo 7.0 features three new deep learning models to enhance the Perception and Planning module capabilities. In addition, Apollo Studio, a one-stop online development platform is introduced to better support Apollo developers. Apollo 7.0 includes the PnC reinforcement learning model training and simulation evaluation services based on previous simulations. This is the latest version of Apollo software and could be the version that transforms the autonomous driving industry.



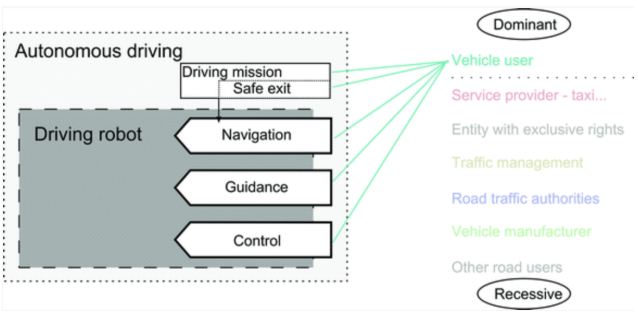
Use Cases

Interstate Pilot Using Driver for Extended Availability

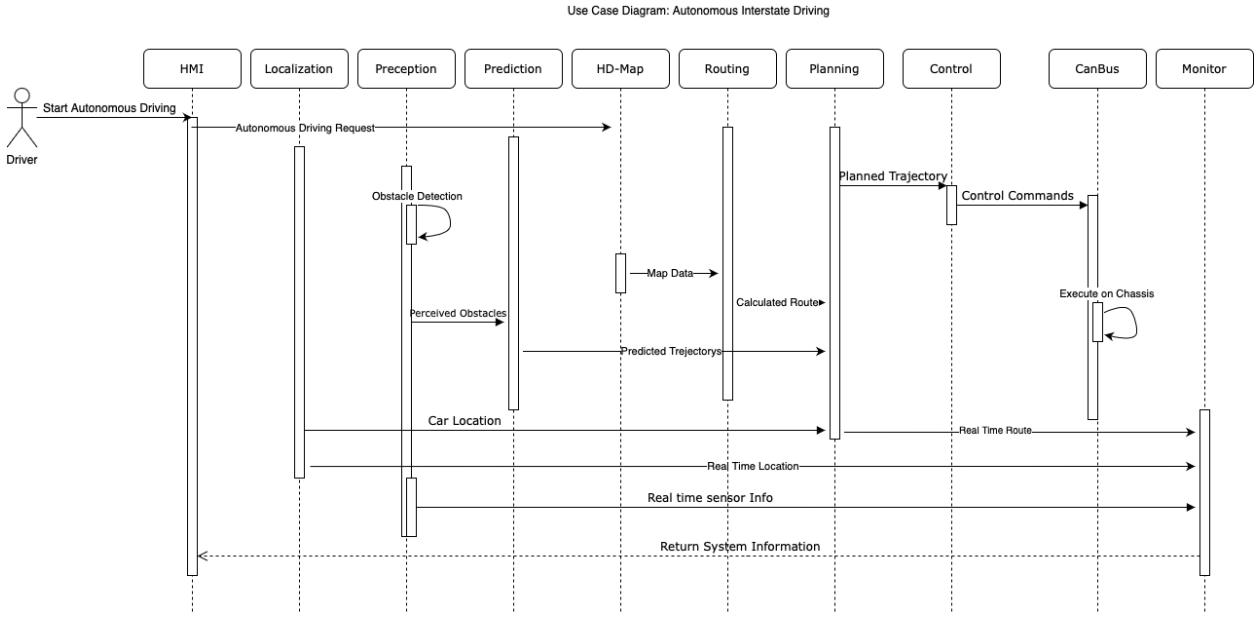


Description: As soon as the driver has entered the interstate, if desired, they can activate the autonomous driving software. The map engine component would come into play and would take note of all major highways, the HD Map and GPS would be then used in conjunction to route to the driver's specified

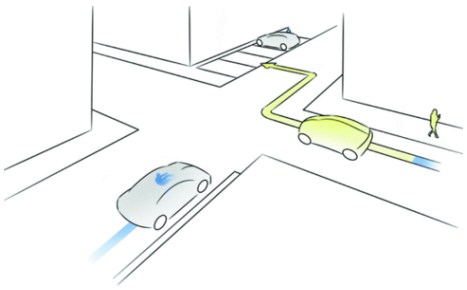
Benefit: The driving robot takes over the driving task of the driver exclusively on interstates or interstate-like expressways. The driver becomes just a passenger during the autonomous journey and can take his or her hands off of the steering wheel and pedals, and can pursue other activities.



destination. The driving robot takes over navigation, guidance, and control until the exit of the interstate is reached. The driving robot safely coordinates the handover to the driver. The driving software would need to use the control subsystem here to understand how to steer, throttle, and brake while taking the driver to their destination. If the driver does not meet the requirements for safe handover after getting off of the highway, e.g. because they are asleep or appears to have no situational awareness, the driving robot transfers the vehicle to the risk-minimal state on the emergency lane or shortly after exiting the interstate. There would also be a significant role of the planning subsystem to make sure that the vehicle can be parked safely and without risk of danger. During the autonomous journey, no awareness is required from the occupant. Because of simple scenery and limited dynamic objects, this use case is considered as an introductory scenario, even if the comparatively high vehicle velocity exacerbates accomplishing the risk-minimal state considerably.



Autonomous Valet Parking

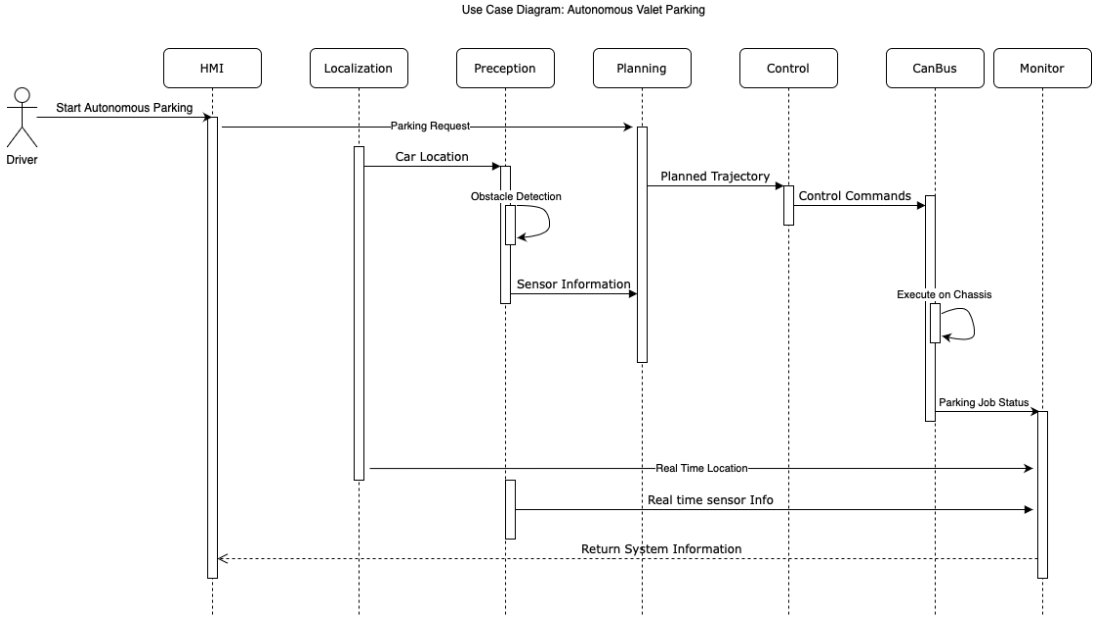


Benefit: The autonomous driving software parks the vehicle at a remote location after the passengers have exited and cargo has been unloaded. It also drives the vehicle from the parking location to the desired destination and can then re-park the vehicle. The driver saves the time of finding a parking spot as well as of walking to/from a remote parking spot. In addition, access to the vehicle is eased (spatially and temporally). Additional parking space is used more efficiently and searches for parking is arranged more efficiently.

Description: Once a driver has reached their destination, they stop the vehicle, exit, and order the driving robot to park the vehicle. The vehicle can be privately owned but might also be owned by a carsharing provider or similar business model. Therefore, the driving robot may now drive the vehicle to a private, public, or service-provider-owned parking lot. It is important to assign a parking lot to the driving robot because the search for the respective parking lot by the driving robot is not taken into consideration for this use case and so, a defined destination for the driving robot is always

given. The localization subsystem will be vital for parking the car perfectly, as the car needs to have centimetre-level accuracy of where the car is at all times to navigate between cars and into the parking space. Moreover, the planning and perception subsystem will also play a big part in getting the car in the proper position to start parking as well as keep track of any incoming objects. Because of the low velocity and the light traffic situation, the deployment of autonomous valet parking is limited to the immediate vicinity of the location where the driver left the vehicle.

An authorized user in the vicinity of the vehicle can also indicate a pick-up location for the driving robot, where it drives the vehicle to the target destination and stops so that the driver can enter and take over the driving task. And, if desired by the parking lot administration, the driving robot will additionally be able to re-park the vehicle.



Data Dictionary

Chassis	An artificial object's load-bearing framework that structurally supports the object's construction and function.
Geo-Fence	A virtual parameter around a physical location
Publish-Sub scribe Architecture	An architectural style that allows independently implemented system components to register themselves as a publisher or subscriber for a particular event or message channel. That is, the publisher produces an event or message to the channel, and the subscriber receives that information.
Process Control Architecture	Represents a real-time process that controls external variables (for example, sensor values) and issues control commands in accordance with these values. For example, a thermostat controls the temperature and gives a signal to start heating if the temperature drops to a certain value.
Conceptual	Describes the system in a general way, without going into implementation

Architecture	details, but describing the requirements for the system and approaches to solving them. For example, a description of a set of system modules and the nature of their interaction without a specific description of the implementation of the modules and details of their interaction, such as data structures.

Naming Conventions

GPS	Global Positioning System
HMI	Human Machine Interface
LiDAR	Light Detection and Ranging
GNSS	Global Navigation Satellite System
API	Application Program Interface
CIPO	Closest In-Path Object
CANBus	Controller Area Network Bus
CUDA	Compute Unified Device Architecture

Limitations & Lessons Learned

Throughout the duration of this assignment, our group encountered multiple limitations that presented the following challenges:

- Apollo lacks documentation on more recent versions of their software (Apollo versions > 5.5) so we had to use some of the older documentation to give a better analysis for some older components. The documentation that they provided did not give much of a description of how components interact and instead talked mostly about how each individual subsystem functioned. This made it increasingly difficult to figure out every concurrent interaction in the architecture
- Apollo doesn't have a very detailed description of how their HD Map is implemented, so we had to use exterior resources such as various research articles. These articles provided us with information about Apollo's modified version of OpenDRIVE
- Some of Baidu's and Apollo's research documentation was written in Mandarin and were difficult to translate directly using Google Translate.

That said, we also made several mistakes and learned valuable lessons while working on this project:

- Our group was able to attain a more in-depth analysis about architectural styles and how it can be applied in real-life scenarios
- None of us knew anything about how autonomous vehicles worked other than that they used cameras to monitor obstacles on either side of the car. This assignment really opened our eyes to how many interactions between components are needed to truly achieve autonomy in a vehicle.

- This assignment also taught us how to represent seemingly complicated architecture with elegant diagrams. It was difficult to visualize the interactions of so many subsystems in the Apollo software architecture, but completing them really helped us understand the architectural style

Conclusion

After careful analysis and review of ApolloAuto documentation, it can be concluded that Apollo employs a combination of process control style and publish-subscribe style for it's conceptual architecture. Process control style is used by the system for the style's ability to provide real-time control, breaking down the system into subsystems and feedback loops. Publish-subscribe is used for its unique capability of broadcasting events to the control and planning systems. In the future, we will look into the concrete architecture of ApolloAuto from the source code. By analyzing how and why it is different to the defined conceptual architecture. This will allow for a greater understanding of how the system was implemented compared to the original plan and if the ideal conceptual architecture was plausible

References

Apollo Documentation:
https://github.com/ApolloAuto/apollo

Use Cases:
https://link.springer.com/chapter/10.1007/978-3-662-48847-8_2

HD-Map:
http://road2ai.info/2018/08/11/Apollo_02/

https://www.ntnu.edu/documents/1284037699/1285579906/Gran-ChristofferWilhelm_2019_Master_NAP_HDMaps.pdf/79ef2eec-c9e2-454b-bf14-08d585cf8826

Architecture Styles:
https://www.tutorialspoint.com/software_architecture_design/data_flow_architecture.htm#:~:text=Process%20Control%20Architecture,or%20modules%20and%20connects%20them.

Architecture Analysis:
https://onq.queensu.ca/d2l/le/content/642417/viewContent/3814366/View