GROUP 24 Diana Candy: Enhancement Proposal

Linus Zuo (zuo.xinghao@queensu.ca)

Peiyang Huang (18ph18@queensu.ca)

Shengqi Yuan (18sy52@queensu.ca)

Alex Wang (18tw7@queensu.ca)

Yifan Zhu (18yz153@queensu.ca)

Shen Ye (18ys121@queensu.ca)

CISC 322

Software Architecture

April 10, 2022

Contents

1 Introduction

In this report, we will present a specific feature, Driver Mode. This mode allows the driver to manually take over the vehicle while the automatic driving is running. At the same time, during the driving process, the eye camera sensor is used to determine whether the driver is fatigued. If the driver enters fatigued driving, the automatic driving system is activated to take over the vehicle and alert the driver. This report will explain in detail which components and interfaces will change due to the new functionality and which should be changed at a high level. At the same time, we also use sequence diagrams to analyze and explain the possible impact of architectural changes caused by new features and the risks of these changes. At the end of the report, we conduct an SEI SAAM architectural analysis of two different ways to implement enhancements. Such analysis includes: 1.Identifying key stakeholders for the proposed improvements. 2.Identify the most important non-functional requirements (NFRs) related to enhancements for each such stakeholder. 3.Evaluate the two proposed approaches to implementing the new functionality and how they impact each identified NFR and stakeholder. 4.According to the analysis in the previous step, determine which way to achieve the enhancement is the best overall.

2 Benefits of the Proposed Enhancement

Currently, in many scenarios, the autopilot system cannot satisfy the requirement of driving in complex road conditions. In this way human drivers will need to take over the vehicle, however the Apollo system doesn't currently have this function.

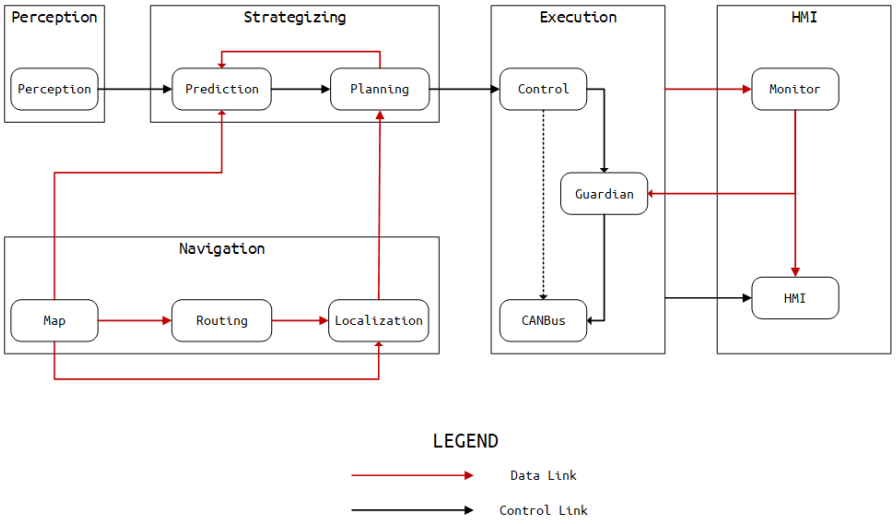
Our goal of enhancement, is to provide such a function, that allow human driver to take over the vehicle from autopiloting. On the other hand, it will constantly detect the current status of the driver through the eye camera. When driver fatigue is detected, the autopilot mode takes over again and alerts the driver.

This enhancement implementation will contain most functionalities within a single new inner subsystem called "Driver Mode", it will interact with the current existed subsystems without changing their current code to achieve the goal.

3 Used Architecture Styles

The current Apollo system is using a Sub-Pub style architecture, which is great for our new features. The new subsystem "Driver Mode" will run based on the Sub-Pub system, gathering information from it, and send the command that interact with others.

4 Current Conceptual Architecture

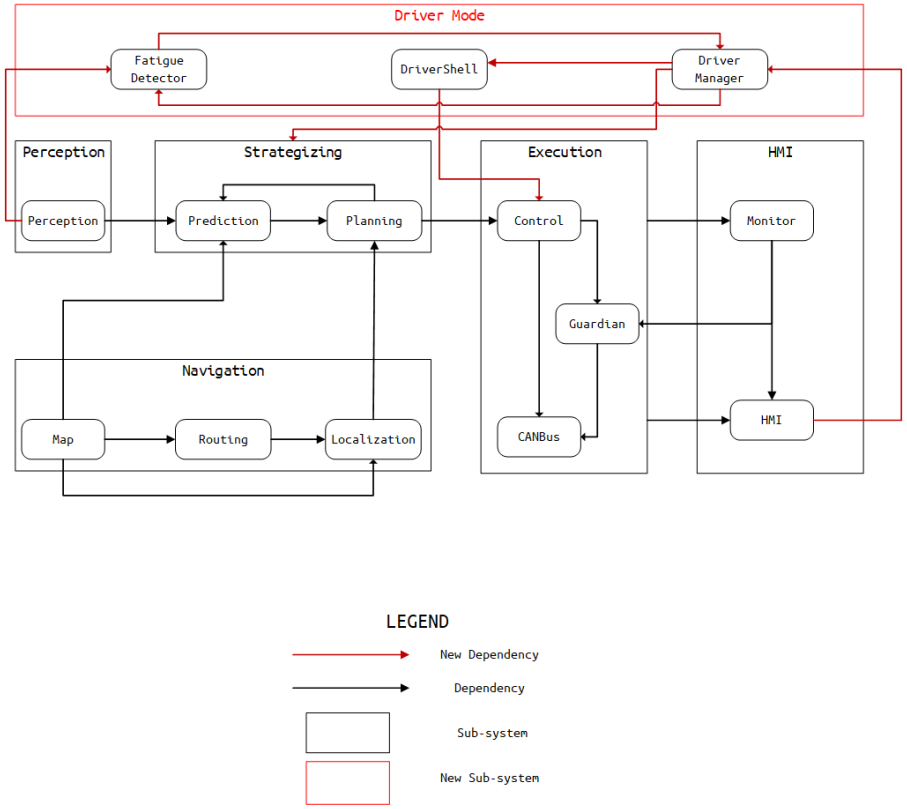


Our derived architecture for Apollo consists 5 interacting subsystems: Perception, Strategizing, Execution, HMI, Navigation. Below is a review of each subsystem's functionality.

1. ***Perception***: Perception subsystem contains 1 module:
   (a) Perception: The perception module identifies the world surrounding the autonomous vehicle.
2. ***Strategizing***: Strategizing contains 2 main modules: Prediction and Planning
   (a) Prediction: The prediction module predicts the future motion trajectories of the perceived obstacles.
   (b) Planning: The planning module will provide a collision-free and comfortable trajectory.
3. ***Excecution***: Excecution Sub-system contains 3 main modules: control, guardian, CanBus.
4. ***Human-Computer interaction***: Human-Computer interaction Subsystem contains 1 module: HMI
   Human Machine Interface (H.M.I.) is a web application. It visualizes all the information from each module and allows drivers to check the status of all the modules.
5. ***Navigation***: Navigation Sub-system contains 3 main modules: Map, Localization, and Routing
   (a) The map engine queries the map data from the cloud service and provides those data to Localization and other modules. The map engine is also responsible for collecting data from the environment.
   (b) The localization module provides positioning service.
   (c) The routing module provide the function as a navigation router.

5 Driver Mode Implementation

5.1 Approach: New Subsystem



Build a new drive mode subsystem. It interacts with perception subsystem. The data flow from perception subsystem to this new subsystem. The drive mode subsystem will send request to strategizing subsystem to take over the car when driver is fatigued. Also, the drive mode subsystem will communicate with HMI. If the driver wants to take over the control of the car. Drive mode will send command to execution subsystem and prevent the command from control module.

There are 3 modules in this subsystem:

1. ***DriverManager***
   The module DriverManager in DriverMode subsystem is used to switch the driving mode between autopiloting and artificial piloting. It will receive commands from HMI and the module FatigueDetector. The HMI module will send it with the driver's command of turning on autopiloting or take over by artificial piloting, the module FatigueDetector will only send the command of autopiloting when it detect fatigue driving.

2. ***DriverShell*** The module DriverShell in DriverMode subsystem is used to send the control information of artificial driving to the Execution subsystem, which will give the human driver the ability to control the car when they choose to take over autopiloting by themselves. It will receive the

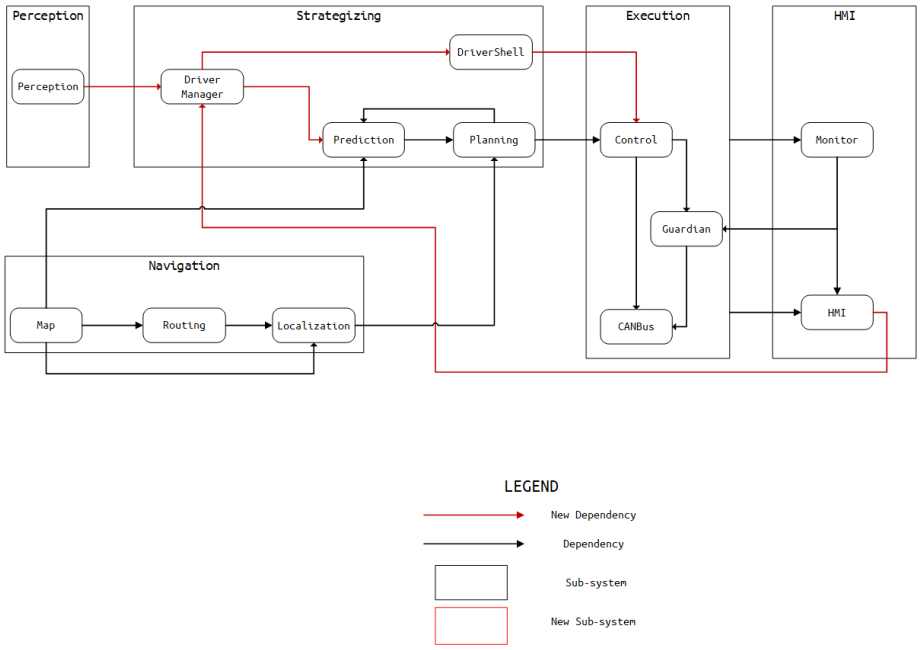
command from DriverManager when user ask to take over the autopiloting and the command of stop artificial piloting when need to stop.

3. ***FatigueDetector*** The module FatigueDetector in DriverMode subsystem is used to to detect fatigue driving during the artificial piloting. It will receive the command from DriverManager when need to start detect the fatigue driving and will gather data from perception subsystem. After it detected, the information will be sent to DriverManager.

   We also need to modify the Strategizing subsystem, to give it ability to listen the command from DriverManager in order to stop autopiloting.

5.2 Alternative: Only Modify Strategizing Subsystem

Implementing the enhancement in the strategizing is an option. It will receive the data from perception system. If the driver is fatigued driving, the module will ask the strategizing subsystem to take over the driving. The data will flow to predication to take over the car. If the driver takes over the car again, data will flow from HMI to driver mode module. This module will communicate with prediction to stop the auto driving.



In this scenario the FatigueDetector module will be integrated into the DriverManager module, and the DriverManager and DriverShell Module will be integrated into the Strategizing subsystem. Here are the details:

1. ***DriverManager***
   The module DriverManager is used to switch the driving mode between autopiloting and artificial piloting. It will receive commands from HMI when need to switch driving mode. And when it is in the autopiloting mode, the information gathered from Perception Subsystem will be passed to the Prediction for autopiloting. When user ask to switch to artificial

driving. It will ask Prediction to stop and activate DriverShell module. And during the artificial driving it will gather sensor information from perception inoder to detect if the driver is now fatigue.

2. ***DriverShell*** The module DriverShell is used to send the control information of artificial driving to the Execution subsystem, which will give the human driver the ability to control the car when they choose to take over autopiloting by themselves. It will receive the command from DriverManager when user ask to take over the autopiloting and the command of stop artificial piloting when need to stop.

6 S.A.A.M. Analysis

We discussed many implementations of out feature. This feature needs to interact with HMI, Exaction, Strategizing and perception subsystems. Those subsystems were constantly changing during this assignment. The interaction with other subsystems is a big concern.

We come up with two implementations. One is implanting the feature as a module in the Strategizing subsystem. Another implementation is we can have a new subsystem to interact with those subsystems in order to implement features.

6.1 Approach 1: Strategizing subsystem implementation

The first approach is adding the feature as module into Strategizing subsystem directly. Compared to new subsystem implementation, this will have much less code changes. In this approach, the enhancement will become part of the Strategizing subsystem.

This was appealing to us when we design the feature. This implementation will have lower impact on the system. The module will be relatively easy compared to build a subsystem. The data will be easier to flow from this module to execution subsystem. Also receive data from perception will be easier.

However, in our further discussion, we think stargazing subsystem is highly related to safety and AI. Adding a module may increase the difficulty of maintenance and developing cost. Also, with single module it is hard to deal with the conflict between perception data and HMI data. We need to create a communication between HMI and this module, they are not connected in the original architecture.

For future development, this enhancement can contain more features on drive mode. The module implementation will have less flexibility to upgrade and add more features in this module. The increase demand of those features will not be met in the future

6.2 Approach 2: The new subsystem drive mode Implementation

We think the new subsystem will be hard to implement and maintain. However, we think new subsystem implementation has its own advantage. It is

By adding new subsystem, the stargazing subsystem will have less changes. Stargazing subsystem is a complex subsystem compared to the new drive mode

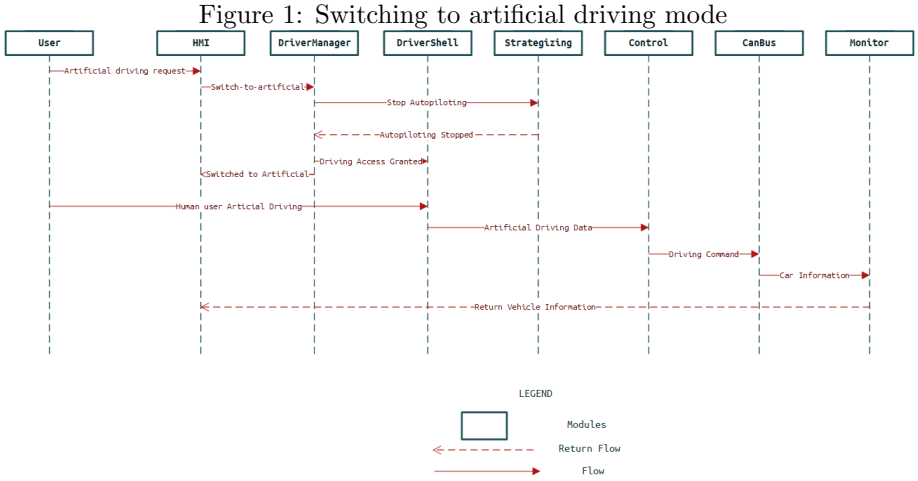
subsystem. Also, we can avoid create new connection between HMI and Stargazing subsystem. It will be required to develop a new subsystem. But the result and user experience won't have much difference.

Also, by adding a new subsystem, it has more flexibility to add new features about the driving safety. The future development of other safety features and upgrade will be easier. It can avoid changing stargazing subsystem when adding other enhancement. We think the modification to the stargazing subsystem is much more difficult.

7 Use Cases

7.1 Use Case 1:

Figure 1: Switching to artificial driving mode

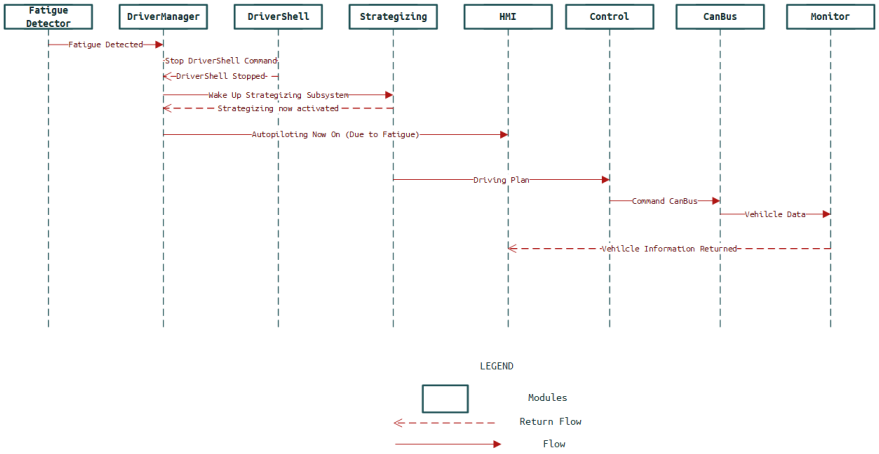


The first use case is the user choose to switch to the artificial driving mode in order to drive manually. Firstly, the user will use HMI to ask taking over autopiloting. Then the command will be sent to DriverManager. The DriverManager will then ask the Strategizing subsystem to stop autopiloting immediately. When autopiloting stopped, the Strategizing subsystem will return a information to DriverManager to inform that.

Now the system has successfully switched to the artificial driving mode. Now the user will be able to drive the car manually. In this case the user will now directly control the DriverShell module by steering wheels and other car controls, DriverShell will send these information to the Control subsystem. The Control subsystem will translate it as command and send to CANBUS. This is how the user control the car directly.

7.2 Use Case 2:

Figure 2: Switching to autopiloting mode (Fatigue)



The second use case is when the system detect fatigue driving and switched to autopiloting. The FatigueDetector will constantly detect the status of the human driver during the artificial driving mode. When it detect that the driver is in fatigue it will send this information to DriverManager. The DriverManager module will now ask DriverShell stop and go to sleep mode. When DriverShell stopped, it will tell DriverManager, the DriverManager will now wake Strategizing subsystem to activate autopiloting. When Stargazing activated it will tell the DriverManager and begin autopiloting.

Then the DriverManager will send this information to HMI with a stronger notification to try to sober up the fatigue driver, and tell the driver the autopiloting is now on.

8 Impact on current system

In-vehicle driver fatigue monitoring technology has evolved in two directions:

1. Based on image signals that directly characterize the degree of driver fatigue, such as driver's face, eye and head movements.

2. Based on numerous indirect signals that characterize the degree of driver fatigue, such as driver behavior, vehicle status and trajectory..

The former generally requires additional sensors such as cameras and infrared sensors on the vehicle, which increases the cost of the vehicle. Still, the recognition accuracy is higher and is suitable for high-end models or models matching ADAS, which can make use of the hardware of ADAS.

The latter does not need to add any hardware equipment, will not bring a cost increase, is suitable for low-end and middle-end models, based on vehicle Canbus signal, GPS signal, etc., can realize driver fatigue status monitoring. However, the accuracy is lower than the former. According to the technical route, the development trend of driver fatigue monitoring is described in detail below.

These detections will be closely linked to Perception, Prediction and Monitor modules, but the impact of the new modules may be reflected in deviations

from the ideal structure and reality. The existing preconceived situation encounters are significantly different from the natural traffic environment, which can be used to explore fatigue monitoring methods but lack realism if used for fatigue monitoring system design. Therefore, these effects can also be reflected in that the Perception and Monitor modules tend to ignore some special working conditions such as lane change, side wind, uneven road surface and emergency avoidance, thus providing wrong information to the prediction module and consequently generating wrong judgment on the driver's fatigue status. Compared with the traditional method, the machine learning algorithm can improve the accuracy of driver fatigue recognition. However, it takes up more memory when computing. It is difficult to be integrated into the controller of the existing electronic control system of the vehicle, and the real-time performance is poor.

On the other hand, we also need to modify the current Stargazing subsystem, the current subsystem doesn't have the interfaces for user to take over the driving process. All the modules are currently designed only to adapt the autopiloting while cannot adapt artificial piloting. Therefore we need to change the code of those modules to allow them stop the autopiloting process temporarily for human driver.

The Common and StoryTelling subsystem (in current Concrete Architecture) will also need to modify and provide interfaces for all 3 new modules DriverManager, DriverShell, and FatigueDetector to transfer information from each other and join the Pub-Sub data stream.

9 Testing Plan

We believe that the focus of the test plan will be on implementing enhanced tests. In the early test, we only have a small team for testing, and in the enhanced test, we will increase the number of people into two groups for testing. The first is to test the code to check whether the results of the independent operation are excellent. If the code testing team finds a bug during the testing process, it needs to describe the problem in detail to facilitate the development, reproduce, and modify it. At the same time, the bug is accurately graded, the progress is tracked in real-time, and the project is completed on schedule. At each stage of the development, the code testing team needs to pay attention to whether the "driver mode" module can cooperate well with other modules. In the enhanced test, it is necessary to focus on the safety test because the driver mode is a module with extremely high safety requirements related to the safety of the driver's life and whether it can help them get out of danger in time. The test team needs to consider common and critical situations and allocate more resources to critical and common situations to ensure that accidents are reduced.

Several important points need to be focused on. First of all, whether the "driver mode" can allow the driver to take over the vehicle manually during the automatic driving operation. Manual operation is required when the automatic driving mode is out of control. In a high-speed vehicle, the response speed is very critical. Code testing teams need to pay special attention to this point. The safety test team needs to consider various situations, such as whether the driving state of the car will not be affected by the manual switching when the driver wants to switch to manual operation (for example, during the process of

turning, whether the driver switches to manual mode will cause the vehicle to fail. direction out of control)

Secondly, the test of accuracy is also very important. Another important function of "driver mode" is to judge whether the driver is fatigued and can freely switch between manual driving and automatic driving. Determining fatigue by eyeballing still requires consideration of many contingencies and surprises. (For example, consider the case where the driver's eyeballs cannot be observed)

In addition to testing, we also need to achieve excellent code management because it is an additional module, which is different from the requirements added from the beginning; we need to ensure that the development process conforms to the old system, and code development needs to attach great importance to management and prioritize important features and new requirements in each development step.

10 Potential Risks to NFRs

1. Performance:
   The potential problem in the performance of enhancement is basically regarding the reaction speed of switching mode(automatic driving / manual driving). Fatigue driving is quite dangerous for the users therefore it is needed to change mode quickly once the eye camera sensors detect driver fatigue. However, it is not difficult for the system to use the control module to transfer the right of control from the driver to the system and the speed of it would be quick to enough to handle any situation.

2. Security:
   Allowing the driver to manually take over the vehicle while it is running on autopilot has some risks. Some situation when the vehicle is running on autopilot, is risky for the driver to switch to manual driving, such as sharp bend, emergency brake, changing lanes in the highway. It is hard for the driver to react immediately in those situations. In this way, the system would detect whether the current situation is safe to switch mode then giving the permission to the drivers.

3. Maintainability:
   The risk related to the maintainability of the enhancement is quite low. Whether giving the permission to the drivers to switch to manual driving, or switch to automatic mode once detecting driver fatigue, are both function that added in the system that would not affect the previous function. The enhanced part is not a significant part in the system therefore once the system previous existing part stays stable, the potential problem in the maintainability would have a super low risk. Furthermore, due to the interaction of each module in the system(ex. perception, prediction, control), it would be a simple task for the enhancement to execute.

11 Conclusion

In order to provide Apollo the ability to switching driving modes between artificial and auto pioliting, we designed to bring a new subsystem: DriverMode into the system for implementation. We did a S.A.A.M. analysis on two different enhancement realizations, and it was clear that implement a new subsystem will achieve our goal. Although the change might be simple, but the whole system will require update in order to fit the new function. We also define the test we need for the system to lower the NFR risk. After carefully planning and analysis, we believe this is the optimal way to provide the driver mode switcher to the Apollo System.

12 Reference

1. Apollo Development Team. 2021. Baidu Apollo System. [Source Code] https://github.com/ApolloAuto/apollo
2. Apollo Development Team. 2021. Baidu Apollo System. [Changelogs] https://github.com/ApolloAuto/apollo
3. Apollo Development Team. 2021. Baidu Apollo System. [Documentations] https://github.com/ApolloAuto/apollo