# CISC 322

**Group23: Assignment3**

# The Enhancement of Apollo Architecture

**April 6, 2022**

**Authors:**

Yuxuan Yang 18yy82@queensu.ca
Jialing Gong 18jg39@queensu.ca
Muyun Sai 18ms78@queensu.ca
Yuxin Huang 18yh98@queensu.ca
Hairuo Li 19hl12@queensu.ca
Songyu Yang 18sy36@queensu.ca

## Abstract

This report will propose a specific feature or enhancement to the current concrete architecture. The impact that this feature or enhancement can have on the non-functional requirements and stakeholders of the system will be explored based on the existing architecture and component interactions.

In this regard, the entire team worked to discuss two specific approaches that could be implemented and compared the two approaches through SAAM analysis to determine the best one. As for further findings and evaluation of the functionality, the report summarizes six main sections: understanding the current concrete architecture and analyzing the benefits of adding functionality; discussing the evolution result of the two approaches under SAAM analysis; sequential diagrams of use cases; impact on existing subsystems, directories, and component interactions; potential risks to NFRs; and testing plans.

## Introduction and Overview

As is well known, autonomous cars are vehicles that are capable of sensing their environment autonomously through unmanned control. It relies on sensors, actuators, complex algorithms, machine learning systems, and powerful processors to execute software. Although there are still many possibilities to enhance its performance, these plans should be carefully considered (and may affect the interaction of components in the previous structure). In this study, we focused on making the vehicle capable of determining whether the passenger volume is within the specified range when carrying people. We will discuss the impact and the risks of this enhancement on the original apollo system, and analyze the interaction between components.

The main function of this enhancement is to be able to determine the impact of passenger weight on the safety index (whether it is overloaded or not, and whether it affects driving safety). CANBus uses CANBus/drivers driven sensors to get an approximate passenger weight, and this feature calls Monitor/common/math to calculate and compare with the safety values, and then feeds back to control. In addition to this feature, we also need this feature to call the CANBus/drivers/camera to detect the number of passengers in the car to ensure that the number and weight of passengers do not affect the safety of the car.

To achieve this improvement we have a main solution and an alternative solution. The two approaches could not be confirmed to have a more positive impact on the system's non-functional requirements and interactions between components.

Therefore using SAMM analysis will more logically organize and compare the two approaches to show which one should be the primary choice.

In fact, the two solutions vary in feasibility and complexity, and we analyze and compare them to explore the risks of improvement and the impact on other components on this basis. It is worth mentioning that two use cases will be used as visualizations to illustrate the improvements between enhancement we have made and the original features.

Finally, We will conclude the enhancement with a summary and discussion of what we have learned from this analysis.

## Benefits of the Proposed Enhancement

Traffic safety is always a challenge to autonomous vehicles, any behaviors that may cause damages to roads or the vehicle in autonomous driving are hidden safety hazards. Among all the behaviors, vehicle overload is one of the fatal accidents that poses a potential threat to the traffic system. That is also why the government pushes several regulation violations on this trouble. Therefore, if there is a functionality that is able to check if there is a vehicle overload in a particular car and can be implemented in the autonomous vehicles would be a huge enhancement of Baidu Apollo autonomous vehicles.

After discussing how to implement this functionality, there are two major benefits from the implementation of it. Firstly, the most obvious advantage is the decrease of the traffic risk. By avoiding it, there will be no excessive number of passengers and the weight which may cause problems like steering gets thrown off balance. When applying a break, the autonomous cars can properly triggles it instead of misjudging the stopping distance because the vehicle overloads. Secondly, the functionality highly reduces the possibility of getting punished by the regulation violation. With the high tech of the sensor on the car, it can automatically check the weight of the car and calculate the possible number of passengers in the car. Also, there are facial recognition systems on the car that can visualize the passengers in the car and give the warning or how to stop the car from moving.

There is no doubt that this functionality can highly improve the traffic safety when there is an autonomous car and users can also rest assured their safety when taking the autonomous cars.

## Used Architecture Styles

The Apollo self-driving car uses the pub-sub architecture style. We need to register overcrowding or overloading as a new event to the system. When the system starts, it will start with other systems to detect and view whether the car at this time has the problem of facing overloading or overcrowding, if so, the car is prohibited from starting. The use of this architectural style ensures timely detection of vehicle problems without delaying route planning.

## Current Concrete Architecture

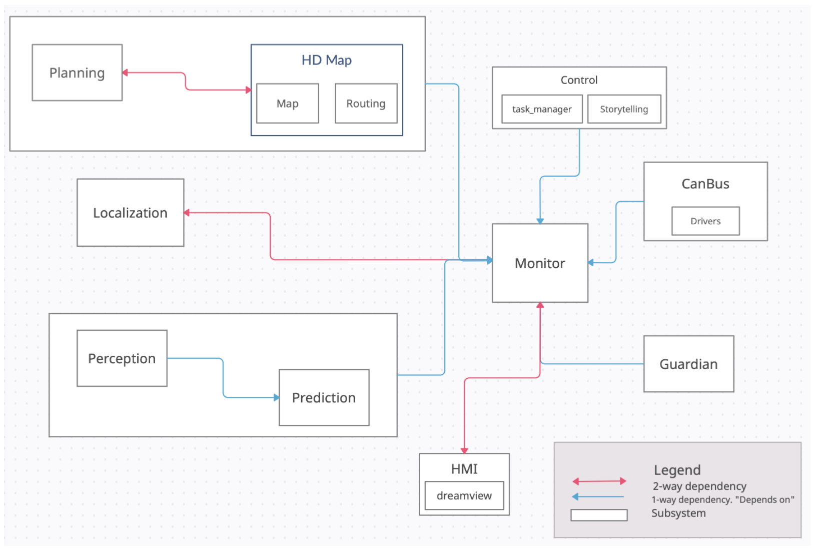

**Figure 1: Concrete architecture analyzed in A2**

Our derived architecture for Apollo Auto consists of 8 interacting components: Monitor, Localization, Perception, Prediction, Planning, Control, Guardian and CANBus. The main subsystems that would be affected by the addition of our Overcrowding/Overloading system are the Monitor, Guardian, and CANBus components. Below is a review of each component's functionality.

### Monitor

The Monitor component is one of the central components of our architecture that is responsible for system level software such as code to check hardware status and monitor system health. In particular, the Monitor is in charge of checking：

1.Status of running modules

2.Monitoring data integrity

3.Monitoring data frequency

4.Monitoring system health (e.g. CPU, memory, disk usage, etc)

5.Generating end-to-end latency stats report

### Guardian

The Guardian is the portion of Apollo that would intervene to monitor detected failures and the corresponding functions of the action center, and to perform the functions of the action center and intervene should monitor detected failures. Guardian has two main functions, which are making sure all modules work properly, and monitoring detects a module crash.

### CANBus

The CanBus is the interface for passing control commands to the vehicle hardware. It also passes chassis information to the software system. It has two data interfaces. The first data interface is a timer-based publisher with the callback function "OnTimer". If enabled, this data interface will periodically publish chassis information. The second data interface is an event-based publisher with the callback function "OnControlCommand", which is triggered when the CanBus module receives a control command. Abnormal load information is also translated into a command to stop the car in an emergency, which is transmitted to the hardware via the CANBus.

## Monitor (Alternative) Implementation

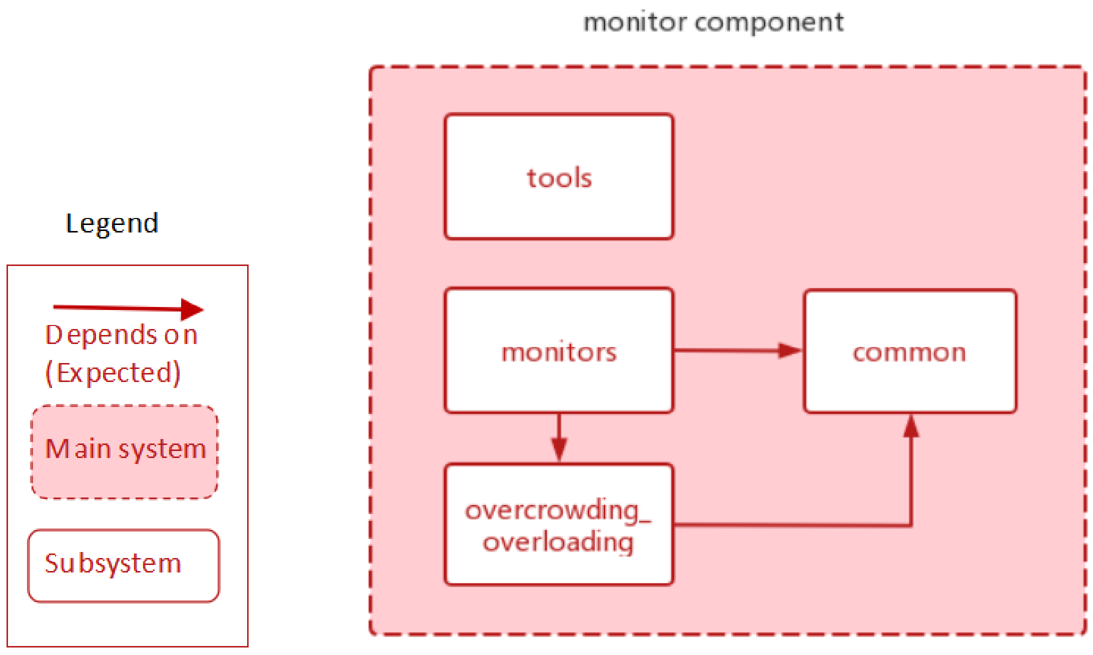

**Figure 2: The interaction within monitor subsystem**

As an alternative to implementing the overcrowding_overloading function, we have decided to add this module to the Monitor. We will add new data to the vehicle state in common/vehicle_state, such as the upper limit criteria for vehicle weight and people, and then the new overcrowding_overloading function will get the current vehicle weight and people data from the sensors (face recognition, and vehicle weight) and use the The common_math is then compared to identify whether the vehicle is overweight or overloading, and if so the monitor gets the exception information and passes it to the HMI for display to the user.

However, using this method to implement Overcrowding/overloading slows down the overall process of the program, as the car has to be maneuvered after the overcrowding_overloading has been identified. This requires Overcrowding/overloading to send the required movement of the car to the control, which in turn sends the specific operation to the CANBus for execution, leading to additional dependencies and more system operations, all of which result in a longer time between the identification of the overcrowding and the response of the car, as well as this can lead to a waste of system resources (the car is planning its route at the same time as it reacts) and thus to possible unsafe problems such as starting the car with an overload.

In order to avoid this waste of system resources and longer reaction times, we have chosen to abandon this approach for our new function. Furthermore, it is difficult to solve the problem of detecting the vehicle again while it is in motion, e.g. after stopping to pick up a new passenger or load a new load, because the function to obtain the real-time status of the vehicle itself is in the CANBus and the need to pull the information away would lead to an additional operation, which would not be possible to determine in time after the vehicle has been started.

## CANBus  (Actual) Implementation

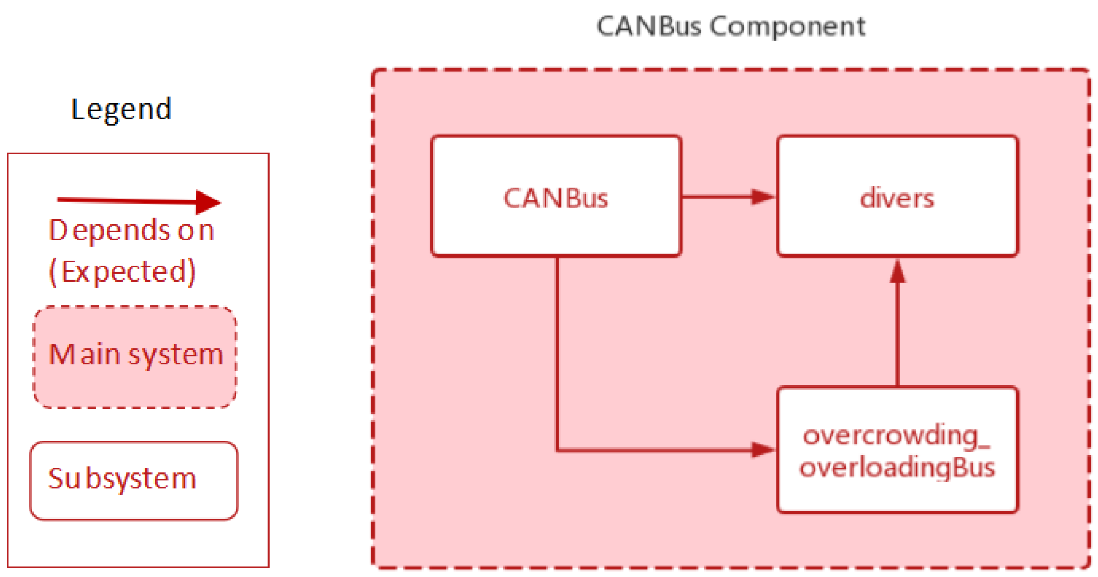

**Figure 3: The interaction within CANBus subsystem**

The actual implementation of overcrowding/overloading is to add this function to CANBus. CANBus_drivers drives the sensing system inside the vehicle and uses this module to provide real-time information about the occupants and load inside the vehicle for overcrowding/overloading, and then from CANBus it then tells the vehicle whether it is overcrowded and overloaded by comparing the standard weight and occupant limit from monitor_common, and then returns the overcrowding information to the control as status information. The CANBus also outputs this exception information to Guardian, then to Monitor, and finally to the user from the HMI.

In this case there are no additional dependencies, the workflow is already in place and is determined before the CANBus executes the command from the control, so that the detection can be done completely before the car is started, thus avoiding the situation where the car is started and not yet judged. Furthermore, the CANBus itself is a module that periodically returns the vehicle status to control, so that it can also make a determination while the vehicle is in motion and only needs to notify control if it detects a significant weight change or an increase in personnel. This ensures that the vehicle is safer to drive and does not consume so many system resources that it slows down the process.

With this approach, we also need to add a new code in the control to determine the status of the overweight and overload of the vehicle, as well as a proxy to output an overweight and overload alert in the HMI, and add the vehicle's original weight limit and occupant limit information to monitor_common_vehicle_state, which is not available in the original code.

## SAAM Analysis

After proposing the enhancement, we should consider how the new enhanced component should interact with other components within the system, what interfaces it should be connected to and what kind of changes it will make to the system interaction at different interfaces.

We explored two implementable options for various variable factors. One is to put it in the monitor to implement the feature, and the other is to put it in the CANBus to monitor the in-vehicle situation. It is undeniable that both approaches can achieve the purpose of our idea, but we prefer to go for a more optimized result in terms of NFRs and beneficiaries.

### Approach1(Monitor)

Our first approach is to add this functionality directly in the Monitor component. This function enables the vehicle system to directly monitor the interior environment to confirm whether there is overcrowding and overloading.

We will add new vehicle state data in common/vehicle_state, such as vehicle weight and the upper limit of the number of people, then the new overcrowding_overloading function will take from the sensor (face recognition, vehicle weight) and use common_math to compare to identify whether the vehicle is overweight Or overloaded, if so, the monitor gets the exception information and passes it to the HMI for display to the user.

This design can use overcrowding and overloading as part of the Monitor subsystem. This means that Monitor can directly get the command returned after monitoring overcrowding and overloading. It can then interact with the control system or the main system to determine if there is overcrowding and overloading and what the vehicle should do next.

### Approach2(CANBus)

Our second approach is to add functionality directly to the CANBus component to enable the vehicle system to monitor the environment inside the vehicle directly. This design treats Overcrowding/overloading as part of the CANBus subsystem, which means that the CANBus can directly get the command returned in Overcrowding/overloading and then interact with control to determine what the vehicle should do next. It is worth mentioning that the most important non-functional requirements (NFRs) regarding enhancement are considered to have a low running speed

In fact, Apollo has been so well conceived in terms of software that we can only consider further enhancements in terms of safety and driving. Previously, CANBus served as a bridge between the control and the guardian, communicating abnormal situations and returning the required action. But this decision was made after the route planning. If a vehicle is found to be overloaded/overweight after this planning, the process will stop. The route planning is done again until the next safety clearance is obtained. This can cause process waste to some extent.

By comparing the first implementation, we found that interacting as a subcomponent in the CANBus does not monitor the number of passengers in real time. In the real case, the load weight only changes every time you get on and off the vehicle. We prefer to make a determination of overcrowding or overload after planning each travel path. This effectively reduces the number of data streams in the system and gets permission for safety when the vehicle is ready to operate.

| NFR's Impacted | Approach1(Monitor) | Approach2(CANBus) |
|---|---|---|
| Performance | Plan the journey only after checking whether the car is overcrowding and overloading | Determination of overcrowding or overloading is made after planning a single path of travel |
| Maintainability | This can lead to longer times between identifying the system overcrowding and the car's response, and this can lead to a waste of system resources | No new dependencies appear between higher-level subsystems |

## Use Cases

**Use Case #1**:

The first use case is to check whether the vehicle is overloaded or overcrowded before starting. In order to ensure the safety of the vehicle, it is necessary to check whether the vehicle is overcrowded or overloaded before the vehicle starts. When the user sends the start command, the first task is sent from the HMI to the Monitor module, and then the Monitor component will send the monitoring data and status to the Guardian and CANBus components, and detect the vehicle overload or overcrowding status through the new component overcrowding_overloading. The overcrowding_overloading component gets the vehicle load and the number of passengers by calling the sensor and camera functions in CANBus_drivers, and then calculates and compares them with the standard values by calling the functions in common_math. If the measured load or the number of personnel is higher than the standard value, the CANBus will not send the start signal and return the information data to the Monitor and Control component and communicate the information to the user through the HMI.

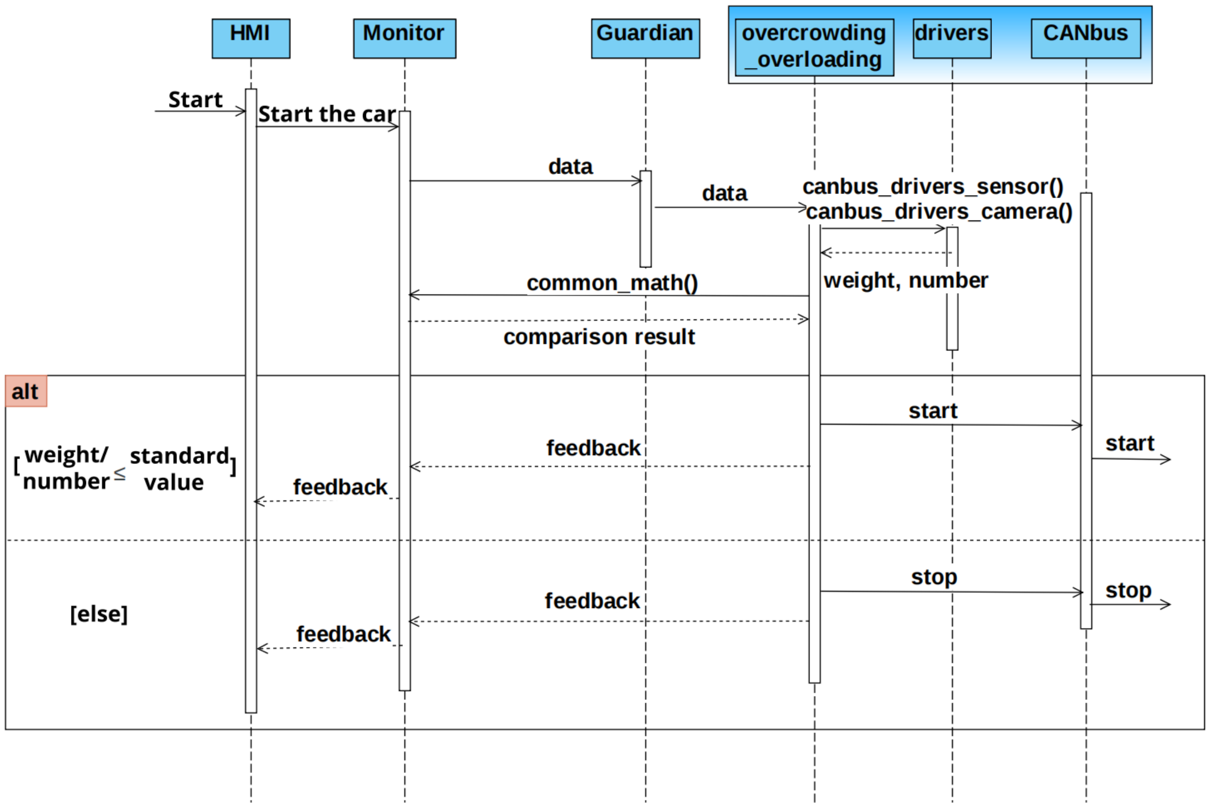

**Figure 4: The sequence diagram of use case #1**

### Use Case #2:

The second use case occurs when a vehicle is stopped and additional passengers are added while the vehicle is detected to be overloaded and cannot start. When the user issues a stop command, the Monitor module will continuously track and transmit data to other modules to ensure that a safe stopping route and location is found. Meanwhile, the Location, Perception and Prediction module is mainly responsible for detecting the surrounding environment, and then the Planning and Control module will develop the route. The Guardian module checks whether the current location is safe for parking and sends a signal to the CANBus module to stop if the safety conditions are met. When the vehicle stops and receives the stop status from HMI, USER can let the new passenger get in and send the vehicle start command. Before the vehicle starts again, the vehicle will recheck whether the vehicle load and the number of people exceed the standard value, if the standard value is exceeded, CANBus will not control the vehicle start and transmit the vehicle forbidden start status to HMI through Monitor.

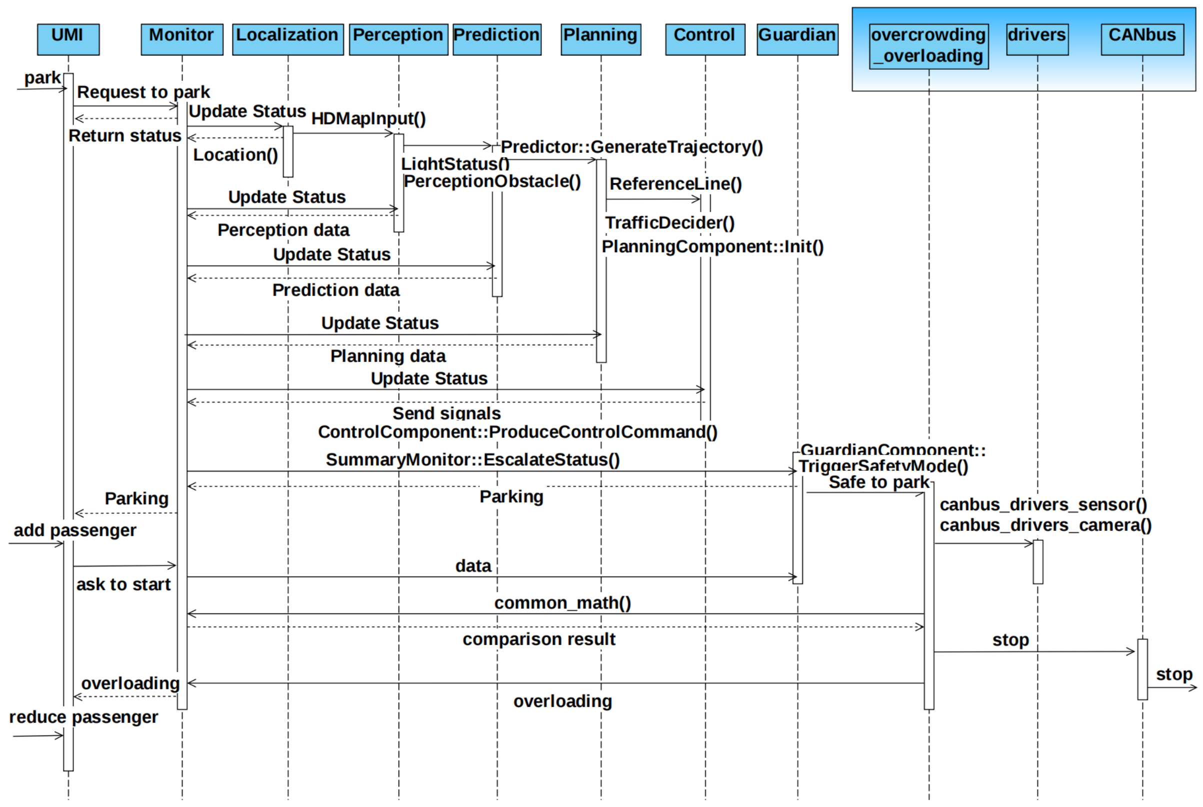

**Figure 5: The sequence diagram of use case #2**

## Impact of Enhancement on Existing Subsystems

Throughout the whole process of the enhancement, it is important to consider the invariability which means even if there are changes made for improving the system, there should be as best as small impact made on the existing subsystems. In this report, there will be two major impacts made to the whole subsystems: Canbus and Control.

### Canbus

Originally, module CANBus is mainly responsible for manipulating the car. Specifically, after receiving the instructions, Canbus will directly execute the instruction and manipulate the car like applying a break, easing up the gas pedal.

Meanwhile, it will also return the current status to the control module. However, as the enhancement is made, there will be a new module added so that Canbus has two more functionalities: facial recognition for detecting how many passengers are in the car and weight measurement for estimating the number of passengers. With the combination of the new module and the existing subsystem, the Canbus module becomes more essential since it can affect the whole decision made to the car.

### Control

Control module mainly transfers the computer language to the instructions that the car can understand. To be more specific, after each subsystem calculates the planned route in the system, it will tell the car what it is supposed to do physically. By adding the enhancement, there is one more process where the Control module receives the message from Canbus module, Control module needs to judge whether information provided from Canbus is allowable and decide whether it approves the car to start to move.

## Impact of Enhancement on Directories

After implementing the enhancement, there are certain impacts on the directories. The most obvious one is the modules directory. Specifically, the Control module, Canbus module and Monitor module, are highly affected by adding the new code for the Apollo subsystem because the new functionality should be implemented based on the cooperation of each module. At the same time, there should also be updates due to this new functionality. For example, when there is an overload situation, it should give passengers a warning telling them overload is dangerous and stop starting the car.

## Interactions with Pre-Existing Features

The implementation of Overcrowding/overloading is to add this function in CANBus, CANBus_drivers is to drive the sensing system in the car, and use this module to provide Overcrowding/overloading with real-time information about the occupant and the load in the car. By comparing the criteria from monitor_common Weight and occupancy limits, it then informs CANBus if the vehicle is overcrowded and overloaded, then returns the overcrowding information to the Control as a vehicle status message. CANBus will also output this exception message to Guardian, then pass it to Monitor, and finally output to the user by HMI.

We also need to add new code to the control to determine the status of the vehicle overweight and overload, and an agent to output the overweight and overload alarm in the HMI, and add the vehicle's original weight limit and occupant limit information to monitor_common_vehicle_state, which is not available in the original code.

## Potential Risks to NFRs

### Maintability

The new modules added to the CANBus and the new functions added to the control run within the same flow as before. In this process, no additional control or data flow is created by the architecture. In other words, the logic of the new functions is basically the same as the existing functions in the control, which is to plan the route by the current state of the car and the state of the surrounding obstacles. The complexity of the system will not increase much, and the maintainability should be consistent with the past, and will not decrease.

### Security

The enhancement should not add any new security risks. As stated in maintainability, because no additional control or data flows are created, apollo car has its own module guardian to protect the software, and these processes are monitored by the guardian. Therefore, the guardian's protection of the system does not create new tasks, it should be able to do its job as it did in the past.

### Performance

Although the judgment is made before the execution of the action, it is after the route planning. Therefore, there is still a waste of the process of the system. If in the case of overweight and overload, the previous route planning will be invalidated and all the previously planned routes will be wasted.

## Testing Plans

The test plan focuses on the impact of communication between enhancements and other features. This testing is typically ongoing from pre-development to post-development and focuses on runnability, completeness, and importance. If the interaction between components during testing does not give the expected results, there may be something wrong in this. We call this runnability. Testing needs to ensure that there are no problems with the flow of data between components.

“Completeness" refers to all situations where problems occur. This includes problems with interactions within subsystems or between high-level system components, data transfer problems, and runtime etc. And importance helps set out how the software will be checked, what specifically will be tested, and who will be performing the test. However, because system components and their subsystems have their own complexity, detecting all cases is difficult to achieve. We prefer to test the functionality of each module according to its importance and focus on the cases that occur more often.

Because the method we decided to implement required re-qualifying the number of people and occupants on board once after each stop, the enhancement only needed to be tested after the system gave "stop" feedback. However, it is important to note that testing of the entire system should be ongoing and monitored in real time by the developer to ensure that other components are working properly as enhancements are added.

## Conclusion

To enhance the safety aspect of Apollo, we came up with an enhancement in the form of an add-on module (subsystem) that can detect if the car is overweight and overloaded while/before it is in motion. We performed a SAAM analysis of two different implementations of the enhancement and it became clear that implementing System Overcrowding/overloading in the CANBus subsystem would achieve the results we wanted with the fastest synchronization. By comparing the bulk of the sensors and formulas for detecting weight and number of people to the CANBus subsystem, we avoided a huge impact on the overall architecture while not introducing more complexity. Through derivation and analysis of the definition of the function itself, we concluded that the non-functional requirements (NFR) have a high maintainability and also low risk. Ultimately, after careful planning and analysis, we believe this is the best way to address safety issues such as overloading and overweight at this time.

## References

1. Synopsys, Inc (2022). “ What is an Autonomous Car ”. Retrieved from https://www.synopsys.com/automotive/what-is-autonomous-car.html
2. Kenan Li ( November 11 , 2018 ). “Baidu Apollo 3.0 software architecture details (2) - a brief introduction to each algorithm module” Retrieved from https://zhuanlan.zhihu.com/p/48654886