CISC 322

Assignment 3
Architectural Enhancement of Apollo
April 8, 2022

Group-14: Artemis

Adrian Chu	18anhc@queensu.ca
Alex Baldassare	18cab16@queensu.ca
Randy Bornstein	18rjb10@queensu.ca
Jake Halay	15jmh9@queensu.ca
Kashish Khandelwal	20kmk3@queensu.ca

Table of Contents

Abstract

This report gives an overview of Team Artemis's proposed enhancement of Apollo's self Autonomous Driving Solution. After some discussion and a lengthy brainstorming period, our team decided that driver monitoring would be a great addition. This addition would detect if the driver's hand is in contact with the wheel and take action if this is not the case. This could greatly benefit those with medical conditions and those at risk of developing such conditions. This will also affect the suppliers of the needed sensors. The enhancement would impact the Monitor, HMI, Guardian, Prediction, and Planning modules of the Apollo system. This report also discusses ways to test the newly proposed feature potentially. These tests include response time and maintenance.

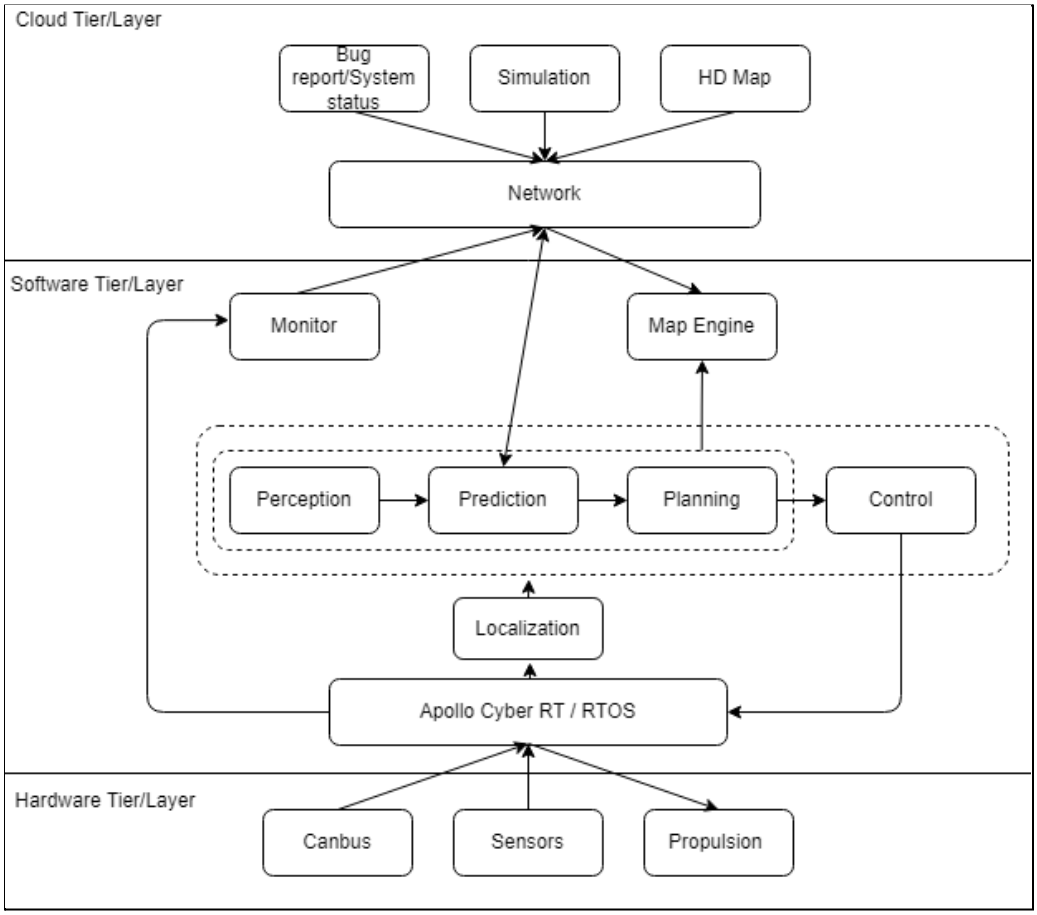
Introduction and Overview

An issue that our group found with Apollo's current architecture is that the vehicle has no way of detecting the driver becoming unresponsive due to a health emergency. In this scenario, the car would continue along its path without making the necessary changes required to aid the driver. We propose a new feature that allows the car to monitor the driver's health status to detect a potential emergency to solve this problem. In the case of an emergency, the vehicle will stop itself safely, and automatically contact emergency services if the driver is unresponsive. We implement this by modifying Apollo's existing architecture and adding new modules and dependencies where needed.

Recap of Conceptual Architecture

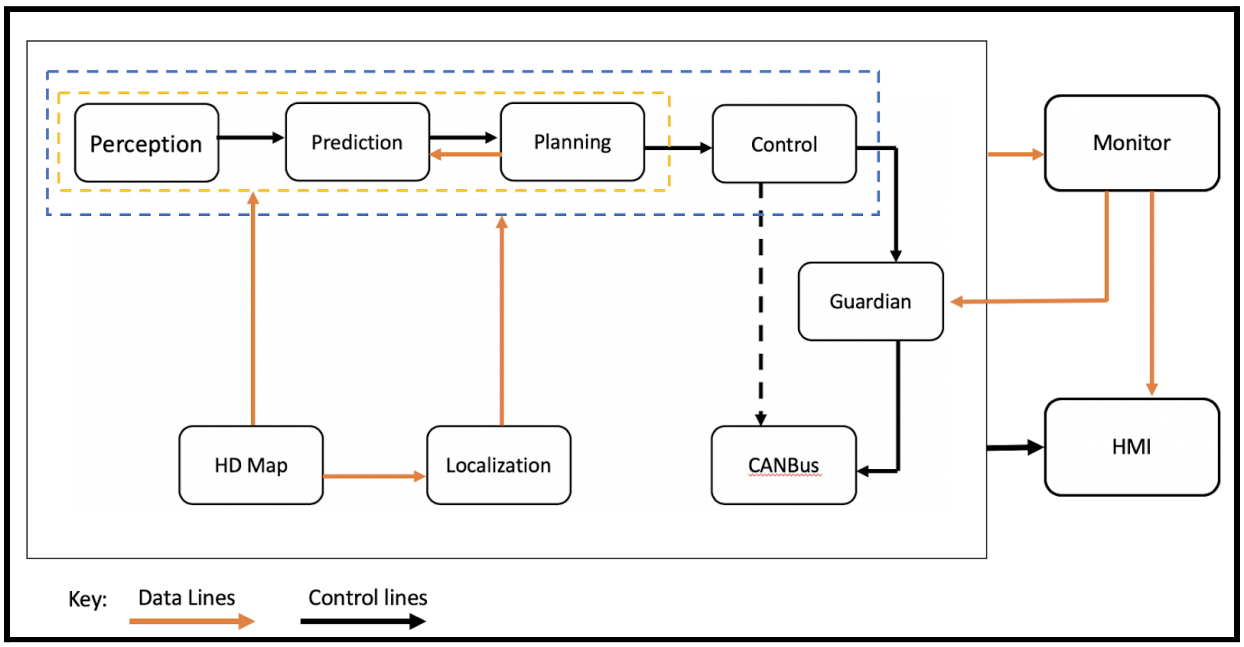
Our group initially proposed that Apollo's autonomous driving system could be represented by a layered/pipe and filter-style conceptual architecture. [Refer to figure 1]

Figure 1. Initially proposed layered-style architecture



After reconsideration, however, we found that the publish-subscribe architecture (or implicit invocation style) is more suitable due to its loosely-coupled collection of components. [Refer to figure 2]

Figure 2. Proposed publish-subscribe conceptual architecture



In Figure 2, the data and control lines represent relationships between the publishing modules and their subscribers. Modules that receive data lines are subscribed to the publishing subsystem's data output, while modules that receive control lines are subscribed to specific published commands. The perception subsystem performs tasks such as object detection and classification, based on the data published by the HD map module. The prediction module, subscribed to the commands given by perception, estimates how the car will interact with the world soon, and vice versa. The planning module interacts with the prediction module by finding practical solutions to these interactions.

Outside of the group of modules that are subscribed to the HD map data, the control module takes the plans published by the Planning subsystem and relays them to the guardian and CANBus modules as commands to be executed. The guardian makes sure that these commands are safe to execute, and will override them to the CANBus if they are not (eg. in case of an emergency). The CANBus module is the final step in executing the commands given to it by either the control or guardian subsystems.

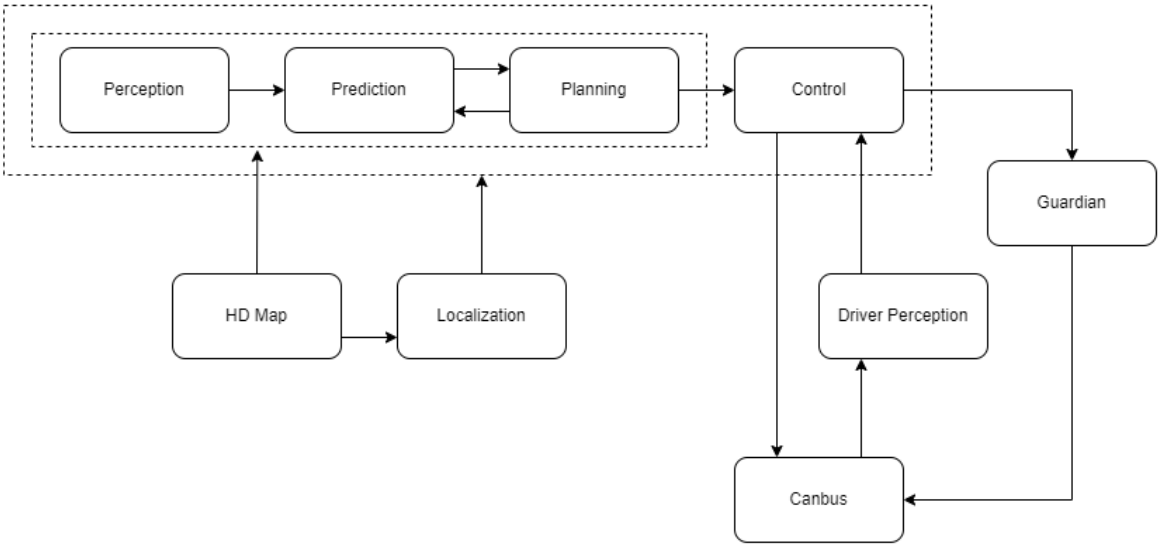
The monitor subsystem is subscribed to data published by all previously described modules for the sake of tracking the performance and status of the individual components within the system. It then publishes this information to the HMI, or human-machine interface, which relays it to the vehicle operator.

Proposed Enhancement

Our proposal for a new enhancement to integrate into the Apollo system is a driver monitor module. Not only will the vehicle be monitoring itself, but also the driver. So in the event of a driver having a medical emergency when driving alone, the vehicle can properly respond to the situation. For example, the driver could suffer from a heart attack while in the vehicle alone and be incapacitated. If the driver momentarily lost the ability to be able to stop the vehicle or contact emergency services, the vehicle would continue to drive until reaching its destination.

This new enhancement would require not only some new software modules but also new sensors integrated into the vehicle. Some of the required sensors would need to monitor the driver's health, which could be done in a few ways. Heart rate monitors could be used to monitor the driver, as well as other sensors such as cameras. This however does raise the issues around the ethics of putting an imaging sensor within the car.

Figure 4. Conceptual Architecture Proposal

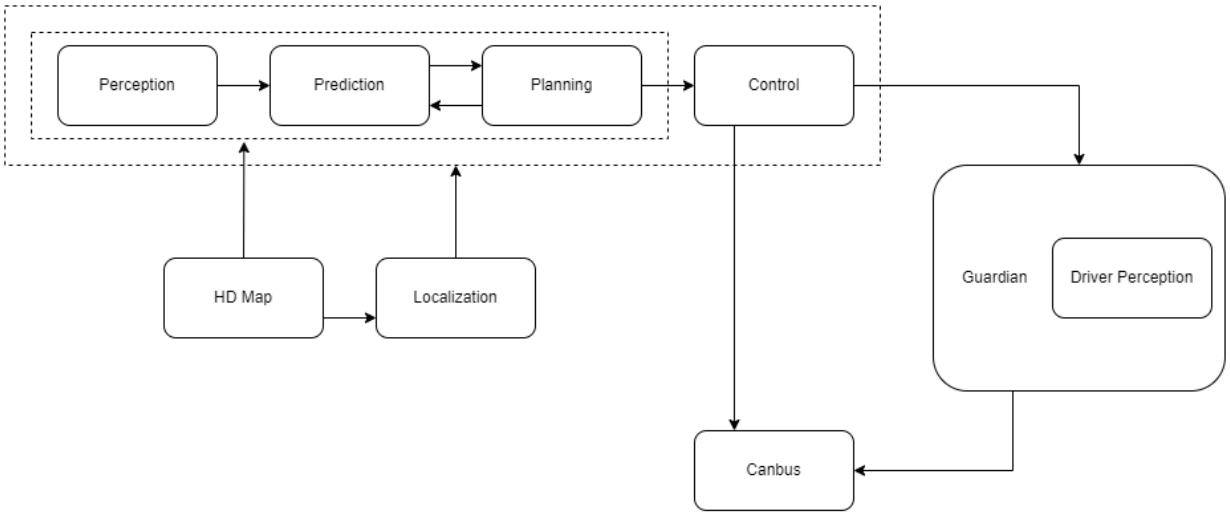


The new module (aptly named driver perception) in Apollo's architecture would need to interact with several other monitors. Firstly, the module would need to interact with the canbus module in order to receive sensor information. Since the vehicle already has so many sensors relating to the vehicle that uses the canbus as their means of communication to the rest of the system, it makes sense that the newly added sensors would use the canbus as well. After the driver perception module detects an issue with the driver and confirms there is an issue, the next module it would need to communicate

with would be the control module, where it would send a message telling the control module to pull over when safe. Finally, the module would then need to communicate with the monitor module to update the status of the vehicle and driver.

The effects of adding this module are quite minimal, as the changes needed for implementation are mostly sending information to the module. The only module the driver perception module would need to directly communicate with would be the control module, when there is a need to send a message to stop the vehicle.

Figure 5. Conceptual Architecture Proposal



Another method of implementing the driver perception module could be to interact with the guardian module. Since the guardian module is already part of the safety system of the vehicle, it would make sense to communicate directly with that system. The guardian system already can override other modules in the case of an emergency. The driver perception module could then be integrated as a submodule of the guardian system. The guardian module would also need to have access to the sensors necessary to monitor the driver. This would not be difficult to achieve, however, since the guardian module is already subscribed to the canbus module and is already receiving sensor information. The driver perception submodule could then identify when there is an emergency with the driver and take the appropriate course of action to mitigate the emergency.

Impacted Modules

The **Monitor** and **HMI** modules will be affected as these are where the interactions with the driver take place. The new hardware modules that monitor the driver's health condition will send this data to the Monitor, which redistributes it to the Guardian and HMI modules. If the Monitor receives data that implies that a medical emergency may be occurring, the first step the system will take is to use the HMI to attempt to confirm this with the driver/occupants. This will be achieved with an auditory warning, prompting the driver to verify that a medical emergency is *not* taking place. If this verification is not received within an acceptable time frame, the Guardian concludes that an emergency is indeed taking place, and sends this information to the Guardian.

The **Guardian** module will also be impacted. Since the Guardian is the system's current solution for detecting emergencies and overriding the current process, our new feature could be implemented here. As Guardian is currently only capable of detecting external emergencies, it will have to be updated to subscribe to the new hardware modules' data output from the Monitor. This will also involve having new procedures programmed that, when Guardian receives information from the Monitor that states a medical emergency is occurring, will override the commands sent by the Control module, to take the appropriate measures to save the driver.

The **Prediction** and **Planning** modules will also be affected since the system must now have a procedure in place to follow when the vehicle performs an emergency stop on the side of the road. Using the information provided by Perception, HD Map, and Localization, these modules must work together to find an appropriate location to stop, and execute the stop safely without putting the vehicle's occupants, other vehicles, or pedestrians at risk.

Non-Functional Requirements (NFRs)

In this report, the effectiveness of the proposed enhancements depends heavily on whether the Non-Functional Requirements (NFRs in short) are satisfied. These NFRs that we will focus on include reliability, availability, performance, and security. Reliability is undoubtedly the most important amongst these NFRs as it directly concerns whether the driver can request help in a possibly life-and-death situation and ensure their safety, whilst the other mentioned NFRs are factors that concern how testing is done and evaluating the implementation as a whole. The list of NFRs does not stop here, however, as some particular requirements are elaborated on in the SAAM analysis section which will be discussed later in the report.

SAAM Analysis

Stakeholders

There are a wide variety of drivers on the road at all times with varying conditions. Many of these people's conditions might hinder their ability to drive on a very minute level. For example, a driver with a heart condition may be able to drive without issue on a given day, but risk a heart attack happening on the road. For many others without previously diagnosed conditions, these types of events can still happen. Thus, the first major stakeholder would be Customers/Drivers. As with any change to a product, you must look at how it will affect your customers first. On the other side of the spectrum are those providing the equipment to "enhance the product." In our case, those supplying the new sensors, drivers, and software, AKA, the Suppliers. These are the people in charge of manufacturing and developing our newly added sensors and play a huge role in the project.

Important Non-Functional Requirements

As previously mentioned, the first major stakeholder of our enhancement are the Customers. Our group decided that the most important Non-Functional Requirements for the Customers are Accuracy, Maintainability, and Cost.

Accuracy: How accurately does the enhancement engage at the appropriate time. Will the system randomly start beeping at me as I am driving because I took my hands off the wheel for a split second to grab something?

Maintainability: What extra steps does the proposed enhancement have on the owner of the vehicle. Should the sensors be serviced every once in a while similar to an oil change?

Cost: The cost of deploying the system. If the enhancement is optional (like upgrade packages for cars), how much will it cost to integrate the feature later on?

As for the Suppliers, the most important Non-Functional Requirements for them are Maintainability, Reusability, and Manageability.

Maintainability: How easily maintained and tested are the sensors and drivers. Similar to certain engine types, some car systems require more maintenance than others.

Reusability: Can the manufacturer sell the sensors to other companies? Having more consumers for your product makes it more appealing to keep producing.

Manageability: The system should support system admin (dreamview) in troubleshooting problems. All issues with the sensors should be able to communicate with the dreamview module.

Impacts on Implementation

As both realizations of the enhancement are very similar, there is not a vast difference in how they impact each identified Non-Functional Requirement differently. In both cases, they both function in the same way, just from a different module of the architecture. For the first implementation, it is its own driver perception module, and for the second implementation, it serves the same functionality, except it is integrated with the Guardian module. They would both impact the Accuracy, Maintainability, Cost, and Reusability in the same way. The only place they stray apart is where they affect the Manageability of the system. In one way, all testing and troubleshooting can be done on the entire module and sensors whereas, in the other, testing must be conducted on a certain subset of the Guardian module. Apart from this, either implementation can be usable.

Comparison

Since, as previously stated, both implementations are almost identical with one having an advantage for Manageability, implementation 1 is the overall best way to deploy our enhancement. The separate module allows for a more easily implemented integration and testing environment within the Apollo architecture.

Sequence Diagrams

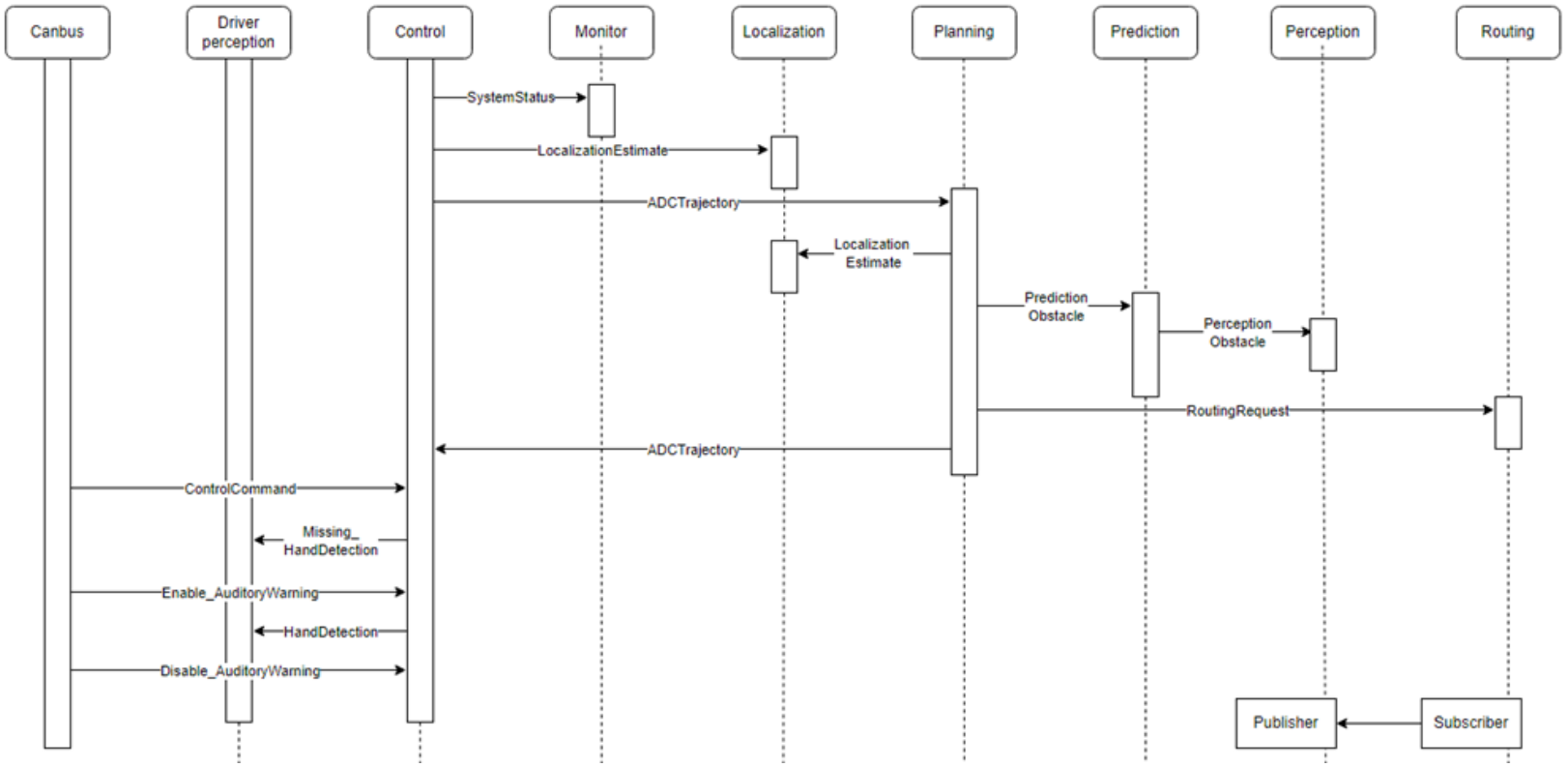


Figure 6- Auditory warning due to no hands on the wheel for an extended period

For our first sequence diagram shown in Figure 6, we can see what would happen if the driver were to let go of the wheel. This works very similar to our sequence diagrams created for A2 of this course as most of the functionality remains the same. The only change is where our 'driver perception' module is introduced. The car's system is initially doing its normal routine of receiving its SystemStatus, LocalizationEstimate, and ADCTrajectory messages for the Control module to function. Now, however, if at any point, the Missing_HandDetection message is ever published for the control to receive, a sequence of events occurs. This Missing_HandDetection message is given whenever the Driver has taken their hands off the wheel for an extended period. This is detected by the new sensors previously discussed. For whatever reason this may be, it is always best to err on the side of caution. When the control receives the message, it publishes a new message for the Canbus to enable the auditory warning system (Enable_AuditoryWarning) which could simply be a light beeping similar to the sound an airplane makes when it's time to put your seatbelt on. If the driver re-establishes contact with the wheel (HandDetection), the following Desable_AuditoryWarning message is published to stop the auditory warning and continue functioning as normal.

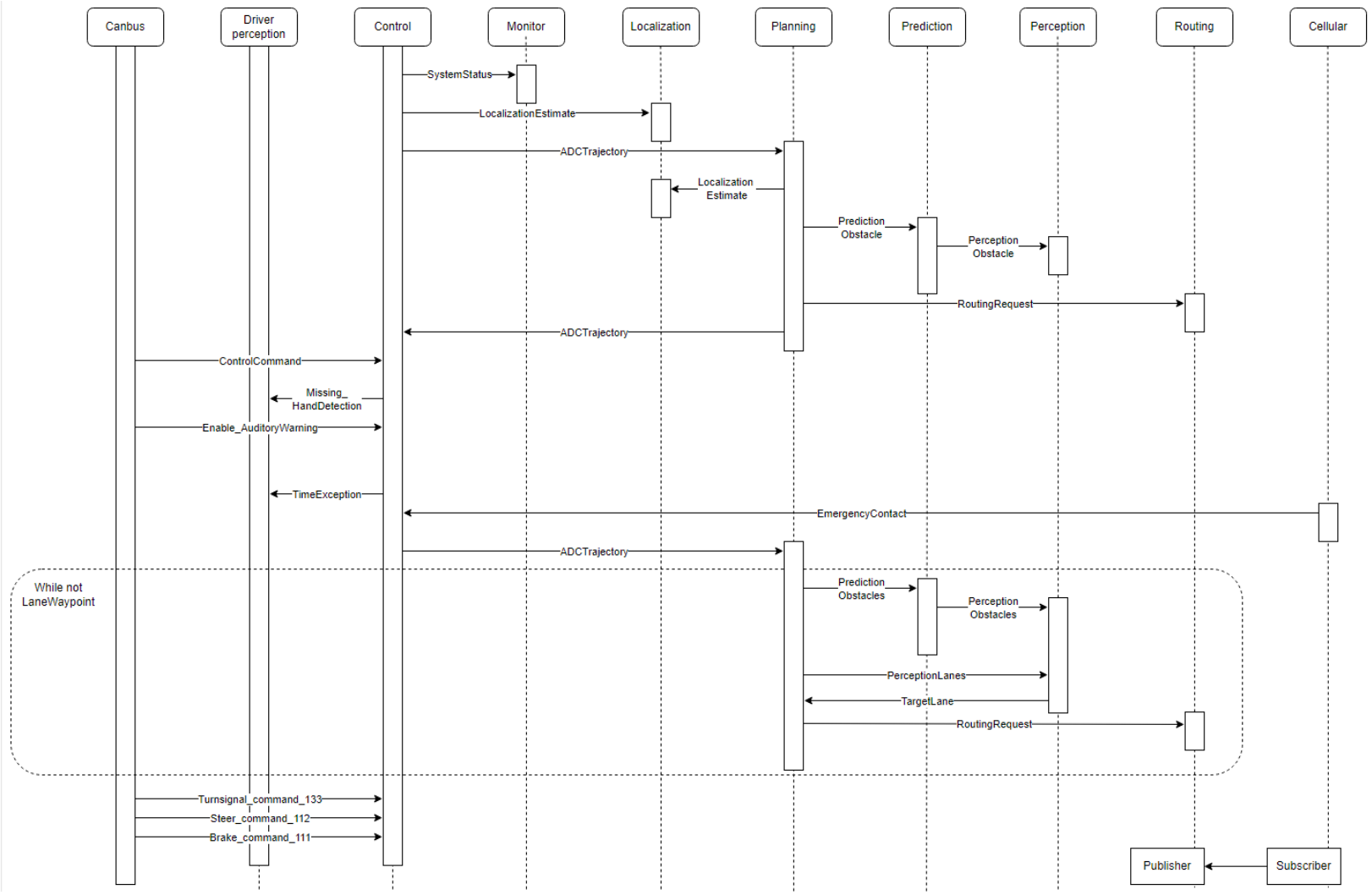


Figure 7 - The car moves to the shoulder of the road after no hands are on the wheel for an extended duration

The second sequence diagram, shown in Figure 7, shows an extension of the previous one regarding what would happen if the driver never re-grabbed the steering wheel. Similar to the first, the system starts out executing its normal routine and only differs when the Missing_HandDetection message is published. Unlike the previous, in this case, the user never re-establishes contact with the wheel. This could be due to a medical condition, fainting, or other reasons someone could lose consciousness. In this situation, after the auditory warning starts and there is no response for a considerable period, the system starts executing its emergency protocols. This involves publishing an EmergencyContact message to the cellular module to send messages to first responders and/or the desired contact as well as pulling the car safely onto the shoulder of the road. To do the second part, the system must find a series of LaneWayPoint's

(depending on the size of the road). In this example, only one is necessary. Once a safe ADCTrajectory is found with the necessary LaneWayPoint, the control can publish the Turnsignal_command_113, Steer_command_112, and Brake_command_111 messages for the Canbus to execute. Once the car is safely on the shoulder, all that is left is to wait for first responders.

Effects and Risks/Drawbacks of the Implementation

Although having such an implementation is for the benefit of the driver, such an open-ended idea can bring its own set of effects, risks, and drawbacks. In this section of the report, we list a few such examples.

This system requires many new hardware components such as heart rate sensors and a hand-detection capacitor system in the steering wheel. These hardware components may not always be reliable and are susceptible to failure. This can lead to the sensors not being able to sense properly, or making them completely nonfunctional altogether for the desired purpose.

There may also be scenarios when the driver / server-side may receive false alerts. For example, when the driver is fine and fit for driving the car but is getting caution messages from the car. There can be many reasons behind this, such as improper connectivity/connection drop with the server-side, which can lead to constant pings to the driver as well as the server without any success (*indirectly,* notification spams). This may lead to the car pulling up on the side of the road, thus stopping the whole journey and making it a cumbersome driving experience for the occupants of the car. This is one circumstance that may be an unavoidable drawback. A whole car's system reset *might* be needed, and will need to re-establish a proper connection to the server for a functional working of the implementation.

There is a slight chance of software failure when it comes to such implementations. The sensors might be sending correct signals out to the appropriate module(s) (eg. driver perception module), but that module may not properly interact with the other modules of the software, thus restricting the functioning of the whole system. This can occur for reasons such as the updates in the software modules which were not optimized properly, or issues that did not appear earlier. This addition of new sensors always has the potential to lead the entire system to this issue.

It is also to be noted that some external factors are unavoidable and can be a drawback to this implementation. One of the major factors is that the vehicle and traffic

norms and policies of each country are different, which can affect the efficacy of this new feature.

Such drawbacks and risks make it hard for this alerting system to be precisely reliable when needed, which in turn can bring down the overall performance level. Fortunately, there are not many potential security drawbacks in this implementation, as most of the transfers of data are happening between the car and the cloud - server-side. As long as the module files are properly encrypted, a security system that keeps the RTOS safe from invasion, and the only server access lies in the hand of the server administration, this would be a fit implementation. Such an implementation is also modifiable, evolvable, and testable, but any modifications will need time to be properly implemented. This can make it hard to maintain the corresponding modules with the constant upgrades and optimizations to the software.

Testing

To ensure that the NFRs are satisfied and maintain the full functionality of the new implementations, testing can be conducted in two ways: regular testing of general use cases and testing for particular risks. This can also be viewed as white-box testing and black-box testing respectively, as the outcomes are expected from general testing whilst developers are testing for unknown outputs when testing for those specific risks.

1. **General test cases (white-box testing)**

   To start with general testing, these test cases include:

   - Testing the response time for the system to activate emergency protocols and the time required to reach first responders (e.g. it takes 50ms for the message to be sent from the cellular module to the first responder unit)
   - Testing the time required for the system to detect that the driver has let go of the steering wheel (e.g. 5 seconds)
   - Testing the amount of force required to be put on the steering wheel to deactivate warning signals

As the outcomes are anticipated and known to developers, it would only be necessary to do regular testing to check if the values match the reference data. Moreover, it would ensure that requirements regarding reliability and performance are met.

2. **Testing for risks (black-box testing)**

Risks, however, are not as easily accounted for as they do not always occur and may cause the system to react differently than expected. Hence, the purpose of testing is to identify instances where system failure may occur and what the preconditions are, which relate to the reliability and security requirements.

- Testing for hardware conditions and whether maintenance is required

To test if the hardware components still function in some conditions, developers should consider the different conditions that may cause the hardware to malfunction. For example, driving under harsh weather may affect the communication between the car and the EmergencyContact module. Another example that also comes to mind is if under the unfortunate occurrence of an accident, say colliding with other cars or objects on the road, would the detachment of the airbag affect the palm detection conducted by sensors on the steering wheel and whether an SOS will still be transmitted to first responders.

- Software compatibility with hardware and checking for updates

Since only hardware may only work with specific releases or versions of software drivers and operating systems, it is mandatory to check if outdated drivers would still function as expected or if errors may occur under these circumstances. Should the latter be the case, the developers have to check dependencies between the software modules and their hardware counterparts, and also determine whether those drivers need to be uninstalled or have newer versions installed (similar to updating Android OS on smartphone devices).

- Antivirus and Security

Last but certainly not least, there is the issue with viruses and security. As a result of malicious attacks or security exploits in the software layer, the system can be susceptible to data theft or even remotely surveilled or grant control to an unwanted remote user. For example, in 2015, Miller and Valasek remotely hacked a traveling Jeep Cherokee to control the audio, windshield wipers, steering, and braking, revealing that an unprepared cybersecurity system can threaten driver safety. Furthermore, in 2016 and 2017, Keen Security Lab hacked a Tesla vehicle to demonstrate security threats and potential attacks related to connected vehicles.[1]

Hence, it is of utmost importance that the confidentiality of the driver's health condition and information is maintained when being constantly transmitted to the cloud server. Assuming that the transmission is protected using encryption protocols and antivirus software that is preinstalled in the system, it would be reasonable to test the

security using machine learning-based intrusion detection and other modules that detect external threats.

Limitations and Lessons Learned

After completing this project and its deliverables, we cannot downplay the challenges imposed by the implementations proposed, especially with risk assessment and evaluating the ramifications that the implementations may cause to the system and its stakeholders. Each consideration has to address the NFRs directly and make sure that the requirements have been met to precisely uphold the driver's safety and wellbeing when operating the autonomous vehicle. The use of SAAM and both white-box and black-box testing provided exceptional aid when determining the effectiveness of the implementations as well as overcoming identified drawbacks.

Conclusion

To wrap up, the proposed enhancement feature is a driver monitoring function that helps to prevent road accidents and ensure the safety of the driver, passengers, and other road users. It is evident from the SAAM analysis that the first implementation is a viable option with regards to meeting the NFRs by deploying a driver perception module instead of integrating it with the pre-installed Guardian module. The testing methods proposed also act as a good reference for assessing the functionality of the new implementation.

References

Park, S. & Choi, J.-Y. (2020). Malware detection in self-driving vehicles using machine learning algorithms. Journal of Advanced Transportation, 2020, 1–9. https://doi.org/10.1155/2020/3035741

--End of Report--