Video Presentation Link

https://mega.nz/file/WNgj0Byb#QY1V6QRWosUr_iB9QkRNcKILEzx7HAU9uxGBBExcrxI

Group Breakdown

Group Member	Contribution
Haadia Mufti (Group Leader)	Derivation Process, Lessons learned, Presentation
Emily Poon	Concrete Architectural style and Conclusion
Kevin Shroff (Presenter)	High-Level Concrete Subsystems, High level reflexion analysis
Oliver Cao	Abstract, Introduction
Gregory Secord	Use Case, Subsystem Reflexion Analysis
Connor Colwill (Presenter)	Use Case, Subsystem Concrete and Conceptual Architecture

Concrete Architecture:

Apollo v7

Group 9: Haadia Mufti, Emily Poon, Kevin Shroff, Oliver Cao, Gregory Secord, Connor Colwill

Overview

- Introduction
- Derivation Process
- Concrete Architecture
- Subsystem Architecture: CANBus
- High Level Reflexion Analysis
- Subsystem Reflexion Analysis
- System Diagram
- Lessons Learned
- Conclusion

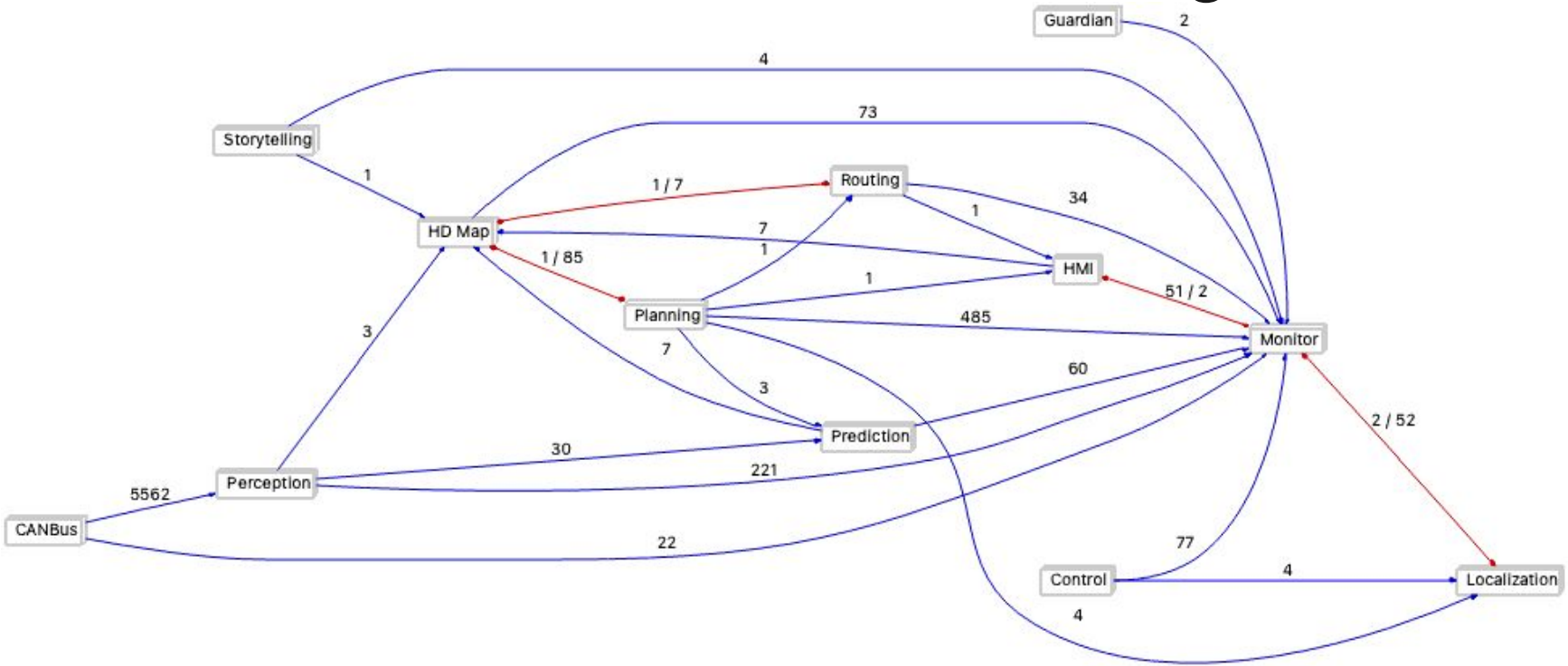
Introduction

- Derive the concrete architecture of Apollo v7 using Understand
- Found unexpected dependencies with the help of reflexion analysis
- Conceptual and Concrete architecture of subsystem CANBus was derived
- Reflexion Analysis was performed on CANBus
- Developed Use Case diagrams with concrete architecture

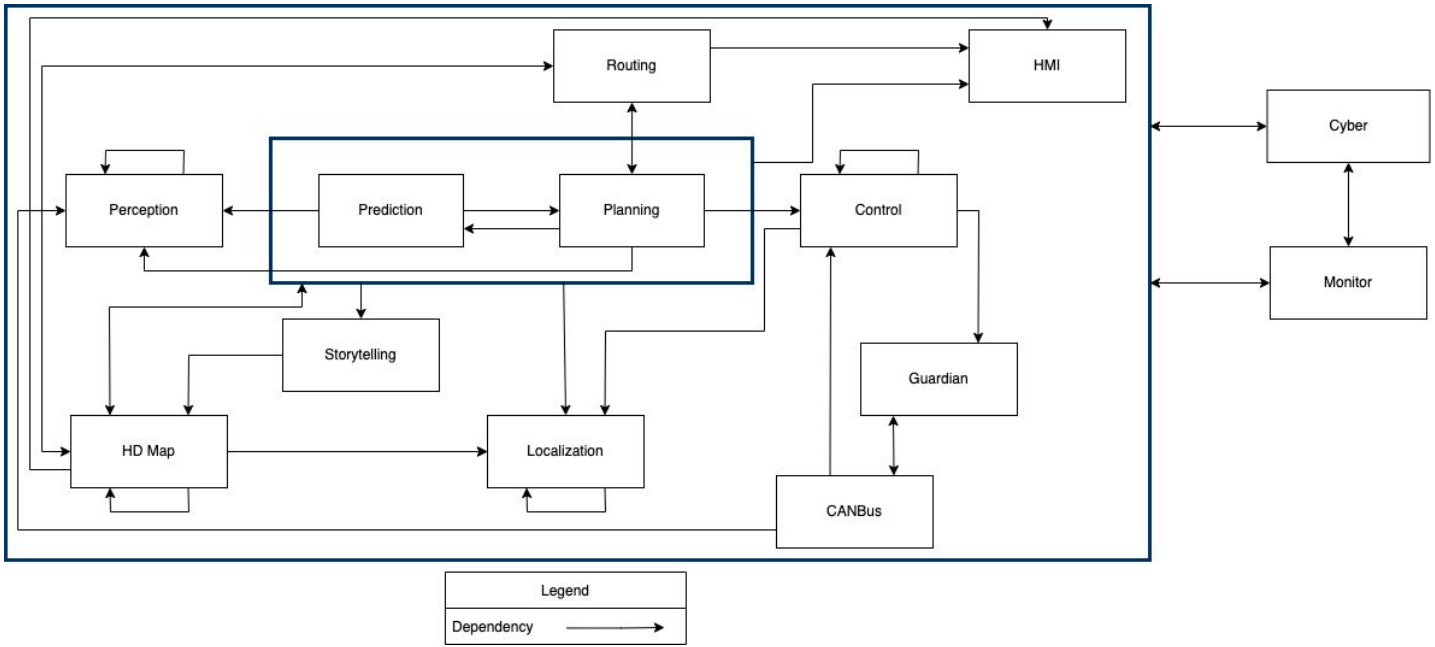
Derivation Process

- Step 1: Used Understand to get the base concrete architecture of Apollo v7
- Step 2: Used the publish-subscribe graph given by the instructor to get all the dependencies between modules
- Step 3: Put all our findings together to form the concrete architecture
- Steps 1-3 were repeated to get the architecture for CANBus

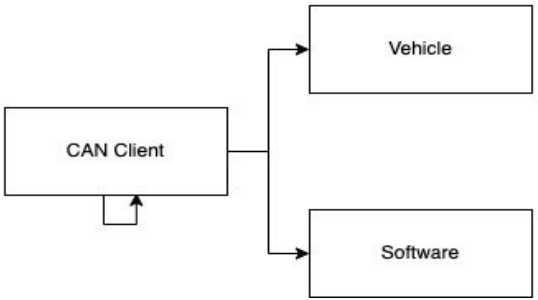
Derivation Process: Understand Diagram



Concrete Architecture



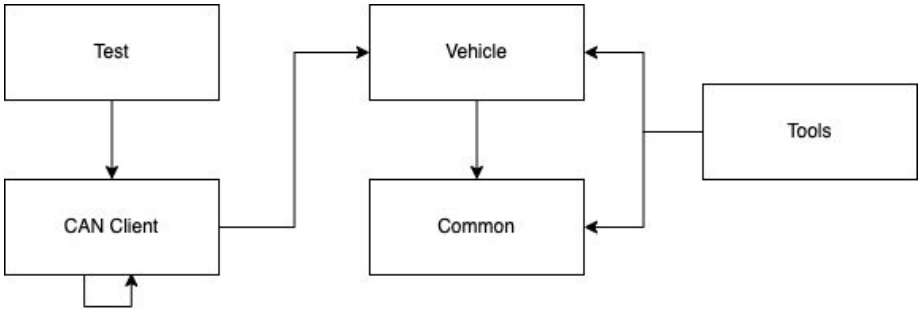
Subsystem Architecture: CANBus



CANBus Conceptual Architecture

- CANBus has client-server style architecture
- Responsible for sending information about the chassis to several different software components, including planning, HD map, and control.
- Used as the ‘bridge’ between the software and hardware components

Subsystem Architecture: CANBus



CANBus Concrete Architecture

Components Overview

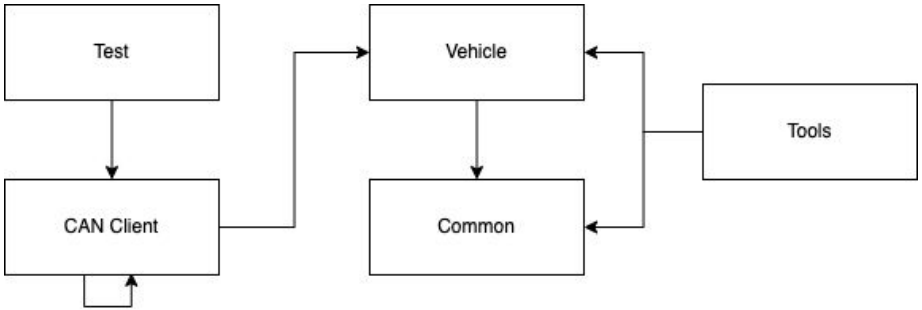
Test

- Holds tests which check if the CANBus system is working properly

Vehicle

- The physical vehicle
- CAN Client will send control commands to the car for driving

Subsystem Architecture: CANBus



CANBus Concrete Architecture

Components Overview

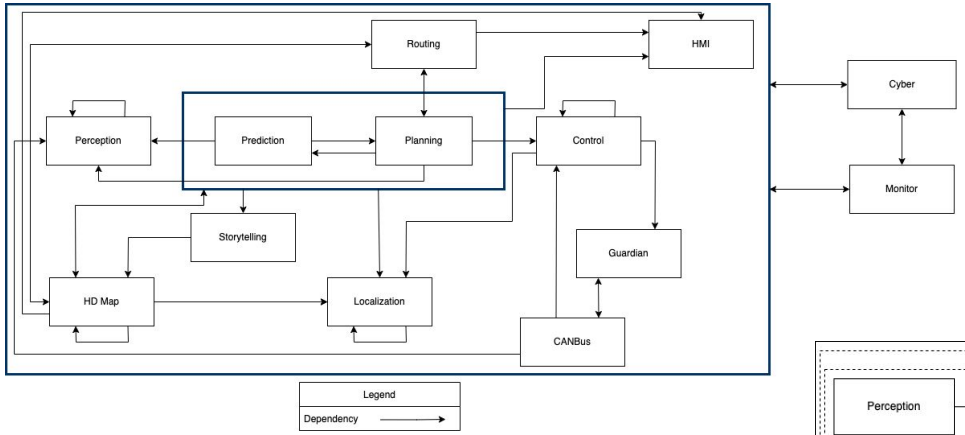
Common

- holds various flags and initialization data for CANBus
- Defines functions such as chassis message publishing, how to interpret received commands, location of test files, and more

Tools

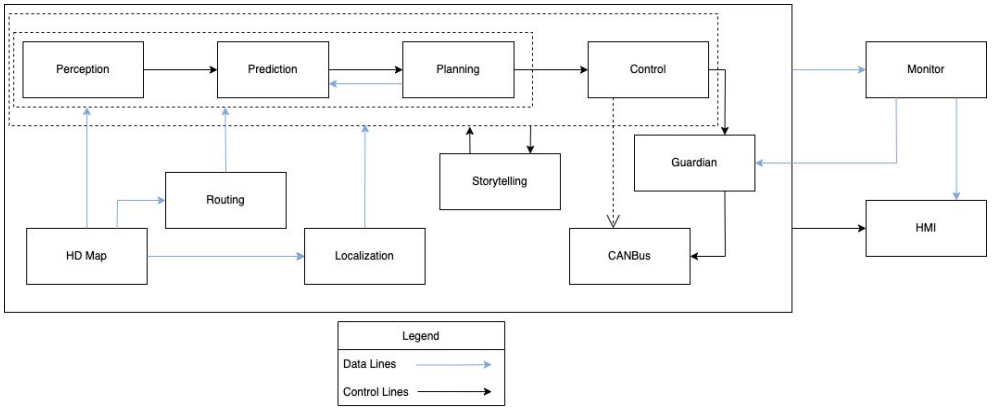
- Mainly responsible for defining driving motions such as throttle, acceleration, hand braking, and gear shifting

High Level Reflexion Analysis



Concrete Architecture

Conceptual Architecture



High Level Reflexion Analysis

Perception → Prediction

- Perception is also dependent on Prediction.
- Rationale for this is that the definition of perception of a vehicle's surroundings in the Apollo system includes the prediction of any obstacle behaviour

Prediction → Map

- Prediction is only dependent on Map in Concrete Architecture
- Discrepancy within our conceptual architecture, as well as even on Apollo's own documentation
- Due to Prediction communicating with the other subsystems through the Common subsystem.

Routing → Map

- Routing is dependent upon HD Map in Conceptual Architecture
- Also the case in the Concrete Architecture

High Level Reflexion Analysis

Planning

- Planning is dependent on Localization, Perception, HD Map, and Routing in Conceptual Architecture
- Also has dependencies on Dreamview (HMI), and Prediction
- dependency on Dreamview is unexpected as it is neither presented in Apollo's own documentation nor our Conceptual Architecture
- Possible Rationale
    - for Dreamview to carry out its graphical user-inclined representations of subsystem data, it needs to cross-communicate with the Planning subsystem
    - or it takes user inputs through the Dreamview module which may affect operation of the Planning subsystem

Control

- Control is dependent on Localization and Planning in Conceptual Architecture
- Unexpectedly not present in the Concrete Architecture

High Level Reflexion Analysis

CANBus → Guardian

- This dependency does not exist in the Concrete Architecture
- Control commands do not go through the Guardian to the Control subsystem, but rather go directly to the Control subsystem
- The Guardian subsystem may intervene in the scenario that it detects something wrong

Map

- Map is dependent on Planning and Routing subsystems in Concrete Architecture
- Map subsystem has a submodule called “Relative Map” which uses these dependencies to behave as a middle layer between modules

Localization

- Localization is not dependent on any subsystems in Concrete or Conceptual Architecture

High Level Reflexion Analysis

Dreamview/HMI

- In Conceptual Architecture, HMI is dependent on the output of all subsystems, however in the Concrete Architecture it is only dependent on Map and Common.
- Rationale for the missing dependencies is that communication with these subsystems must have gone through Common

Monitor

- Dependent on the output of all subsystems minus HMI, in Concrete Architecture however, it is only dependent on Dreamview
- Cross-communication between Dreamview and Monitor regarding the reporting of hardware and software status, and system health monitoring.
- Rest of the communication needed by Monitor is accomplished through Common

High Level Reflexion Analysis

Guardian

- Only dependent on Control and Monitor in Conceptual Architecture but in the Concrete Architecture, Guardian is only dependent on Dreamview.
- communicates with other subsystems through Common

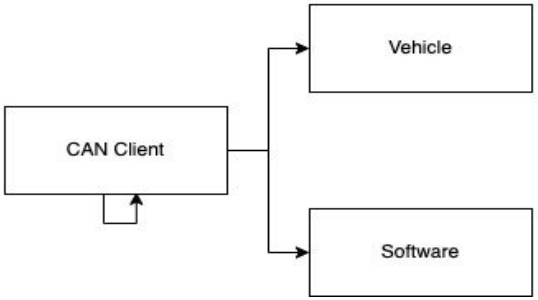
Storytelling

- In the Conceptual Architecture, Storytelling is not dependent on any subsystems but in the Concrete Architecture Storytelling has a dependency on Map
- Stories need to be mapped to a map in order to be executed

Common & Cyber (Missing Subsystem)

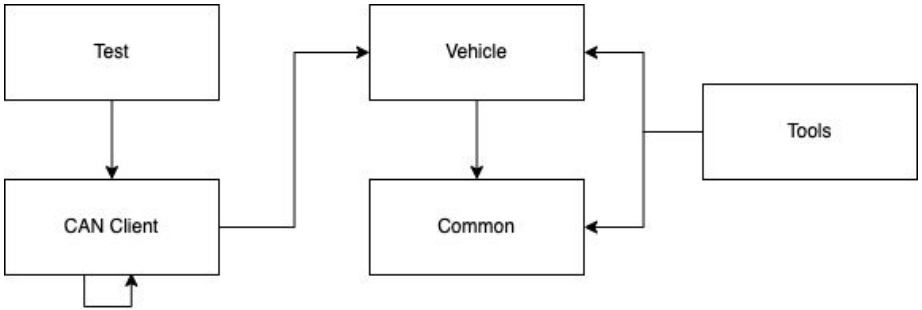
- Common & Cyber are subsystem that every aforementioned subsystem is dependent on
- Common is used for some common shared functionalities between subsystems
- Cyber represents the open-source “Apollo Cyber RT” runtime framework that all of the subsystems run on
- Common & Cyber were not explicitly referred to in the Apollo documentation

Subsystem Reflexion Analysis



Conceptual Architecture

Concrete Architecture



Subsystem Reflexion Analysis

Test
- Subsystem should interact with the control commands and the vehicle itself so it's constantly working as expected

Common
- More or less embodied by our Software component in conceptual architecture
- Software component would embody too many things

Tools
- Also categorized in the Software component
- Tools component deals with important driving motions, it too should be dealt with as an individual entity

Tools -> Common
- Tools component relies on test files

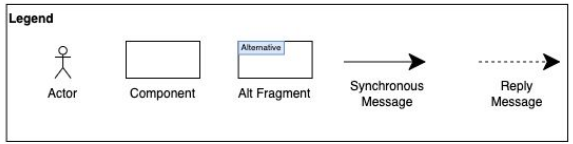
CAN Client -> Software
- CAN Client depended on the Software component in the conceptual architecture but CAN Client has no dependencies in concrete architecture
- CAN Client itself embodies the software that sends driving instructions, chassis status, etc

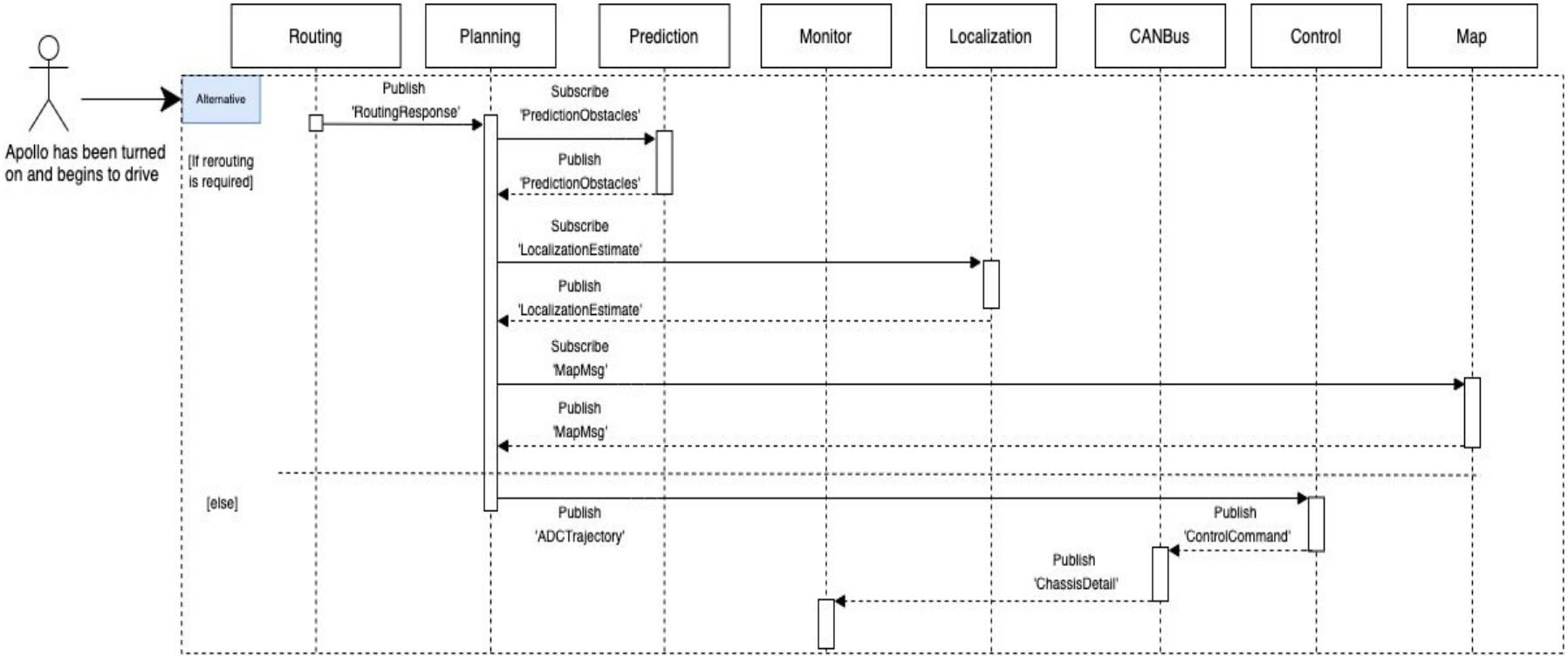
Test -> CAN Client
- Test depends on the CAN Client since that is the subsystem that it monitors

Vehicle -> Common
- Allows the controller to create and distribute messages for other modules to subscribe to

Tools -> Vehicle
- Allows cooperation between the vehicle and the code instructions of the throttle, acceleration, gear shifting, etc.

System Diagram





Limitations and Lessons Learned

Limitations:

- Too many dependencies to keep track of within Apollo's architecture
- Hard to find associated methods within the source code for the system diagrams

Lessons Learned:

- Concrete architecture gave us a deeper understanding of Apollo's architecture
- Understood the importance of doing the reflexion analysis

Conclusion

- Were able to visualize Concrete architecture with the help of Understand and the publish-sublish graph given to us
- Apollo is highly interdependent
- Found unexpected dependencies and new subsystem eg: Cyber
- CANBus has a client-server style architecture

References

[1] Apollo Module Breakdown:
https://github.com/ApolloAuto/apollo/tree/master/modules

[2] PubSub Dependency graph:
https://onq.queensu.ca/d2l/le/content/642417/viewContent/3865686/View

[3] Previous year projects:
https://research.cs.queensu.ca/home/ahmed/home/teaching/CISC322/F18/index.html

[4] Apollo Understand Diagram:
https://docs.google.com/document/d/1qcHmRh1gAGTZMorCl1ILjomHqvamk6pjCgH2GdOogFs/edit?usp=sharing

[5] Apollo Prediction Subsystem Documentation
https://github.com/ApolloAuto/apollo/tree/master/modules/prediction

[6] Apollo Planning Subsystem Documentation
https://github.com/ApolloAuto/apollo/tree/master/modules/planning

[7] Apollo Control Subsystem Documentation
https://github.com/ApolloAuto/apollo/tree/master/modules/control

[8] Apollo Relative Map Documentation
https://github.com/ApolloAuto/apollo/tree/master/modules/map/relative_map

[9] Apollo Dreamview Subsystem Documentation
https://github.com/ApolloAuto/apollo/tree/master/modules/dreamview