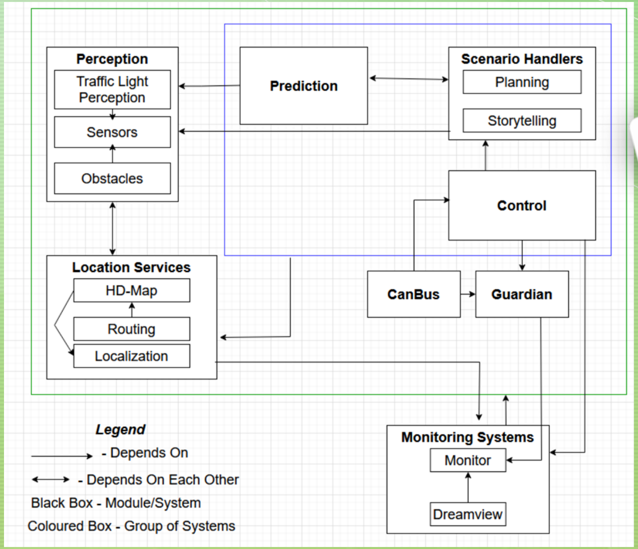
Presentation video URL: https://youtu.be/Zp_uBJyTO2w

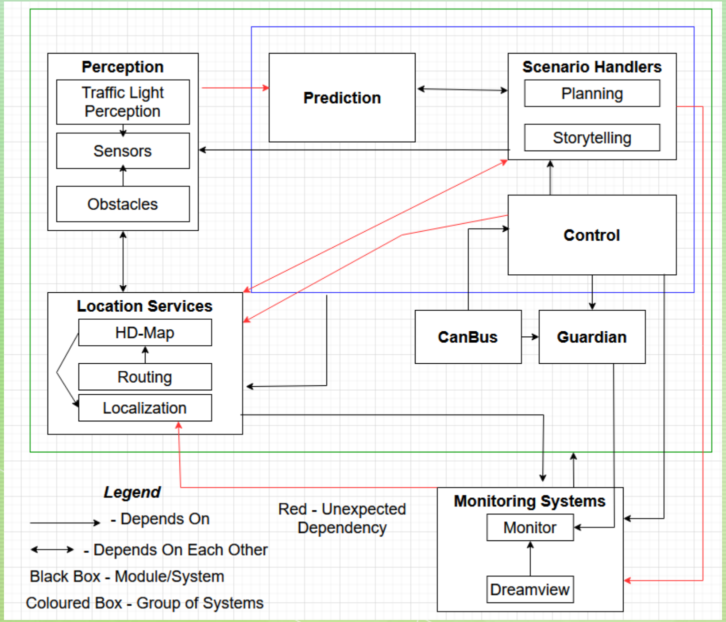
Apollo Concrete Architecture Presented by - 40 Below

- Team Leader - Joshua Kouri
  - Intro, Derivation process, Subsystem breakdown
- Presenter - Alex Nechyporenko
  - Reflexion Analysis, Presentation
- Josh Tremblay
  - Abstract, Second-level subsystem architecture
- Kennan Bays
  - Sequence diagrams, Lessons learned
- Mason Choi
  - Updated Conceptual Architecture, Concrete Architecture, Conclusion

Derivation Process: Concrete Architecture

Inserted dependencies generated by the Understand tool onto the Conceptual Architecture Graph



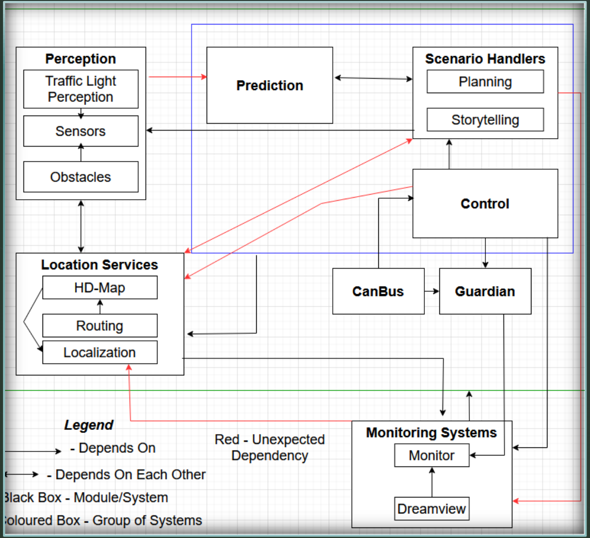


Updated Conceptual

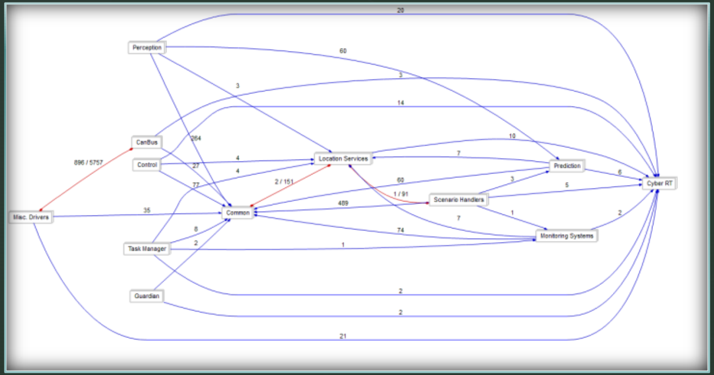
- We now know that the architectural style of Apollo is publish-subscribe, rather than our original proposal of the process-control style.
- After reviewing the publish-subscribe architectural style, we see that it is more fitting for the overall software, while process-control only better fits the Control subsystem.
- The software of Apollo consists of multiple modules, which each have their own operations to carry out and end up enabling other operations in the process. When an event is triggered by a module, it implicitly invokes procedures within the other modules in the system through a broadcasting system, which in this case is the Cyber RT component.

Comparison to Concrete

- The concrete architecture has several modules that were not included in the conceptual, such as Common, Misc. Drivers, and Task Manager
- The major subsystems such as Perception, Control, and Prediction remain largely the same
- Some of the dependencies are different from what we had thought
- Concrete architecture shows the actual number of dependencies
- The concrete architecture contains the Cyber RT module



Concrete Architecture



- From the extracted dependency file of Apollo's software system, we were able to create this concrete architecture. This was done by creating the subsystems based on our conceptual architecture, then sorting the modules as best as we could into them.
- Some general modules, such as "common", became subsystems of their own, as well as anything that could not be sorted into a specific subsystem using the information we currently have.

Reflection Analysis - Unexpected Dependencies

- Missing Component:

- Cyber RT - Connects to all other components and acts as a central framework which enables high concurrency communication and efficient data interpolation

Notable Dependencies:

- Control -> Location Services
  - Adjust vehicle output based on location information
- Perception -> Prediction
  - Inverse relationship to the one mapped out in conceptual architecture. Calls on Prediction to analyse information based on sensor data

Summary of New Dependencies:

- Control -> Location Services
- Perception -> Prediction
- Scenario Handlers -> Location Services
- Scenario Handlers -> Monitor Systems
- Monitor Systems -> Localization
- All subsystems -> Cyber RT

Control - Conceptual Architecture

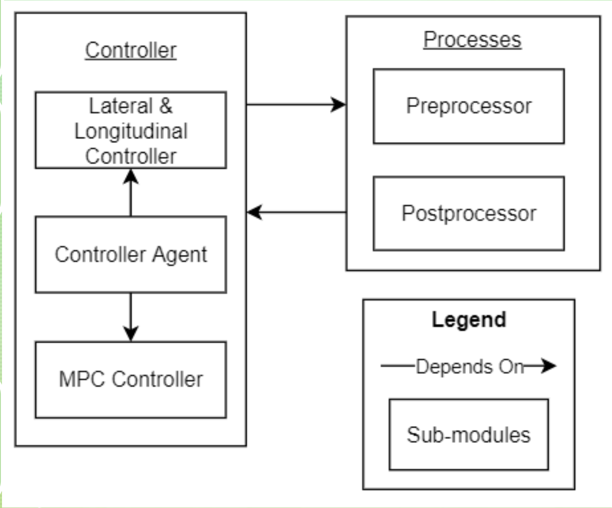
The control module can be represented by the process-control style

Process component: manipulates variables from other modules that are passed to the control module

- Preprocessor - processes input data from modules to be prepared for the controller
- Postprocessor - processes data after being passed through the controller to ensure proper output

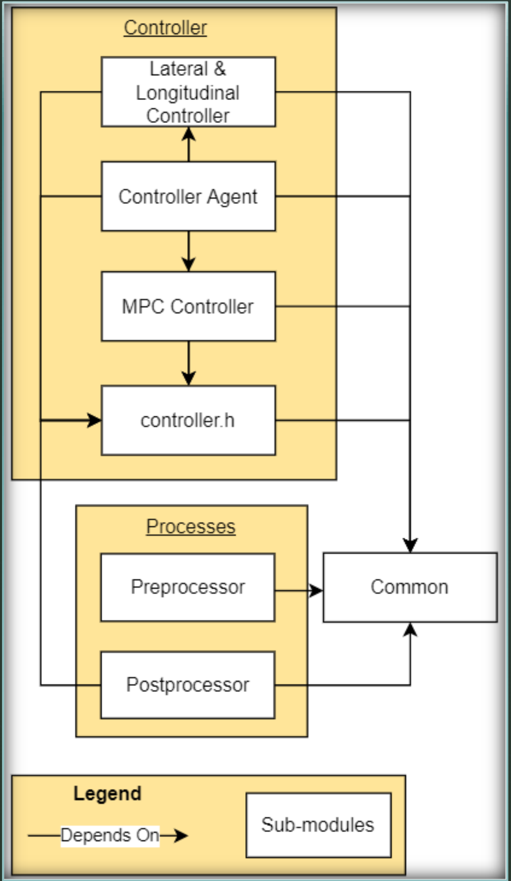
Controller component: determines how the variables are manipulated

- Lateral & longitudinal controller - manages values relating to vehicle controls steering, brake, & throttle
- MPC controller - help predict control commands that are optimal to execute
- Controller agent - manages other 2 controllers



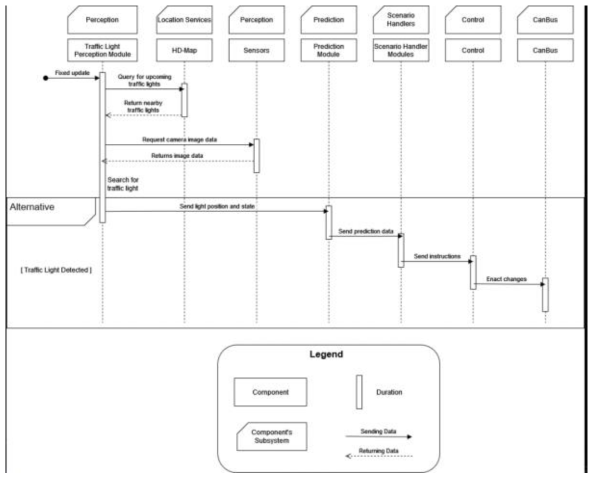
Control - Concrete Architecture & Reflexion Analysis

- A few components were missing in the conceptual architecture when creating this concrete architecture.
- controller.h
  - Acts as a base class for all controller components. Postprocessor uses this file also to get data of variables passed in the controller
- Common
  - Module that contains files important for the general functioning of the control module
  - control_glfags.h → defines flags used in control
  - mrac_controller.h → more accurate and faster control commands
  - Trajectory_analyzer.h → obtains & manipulates current location point for a trajectory



Sequence Diagram - Traffic Light Detection

- Traffic Light Perception Module
  - Queries the HD-Map for any upcoming traffic lights
  - If any found, then requests image data from the camera sensor
  - Searches image for traffic lights and reports their state and position to Prediction module
- Prediction & Scenario Handling
  - Prediction module sends prediction data to Scenario Handler modules
  - Scenario Handling module sends instructions to Control module
- Control & CanBus
  - Control module enacts required changes through CanBus module (eg. slowing down for amber or red light)



Limitations and Lessons Learned

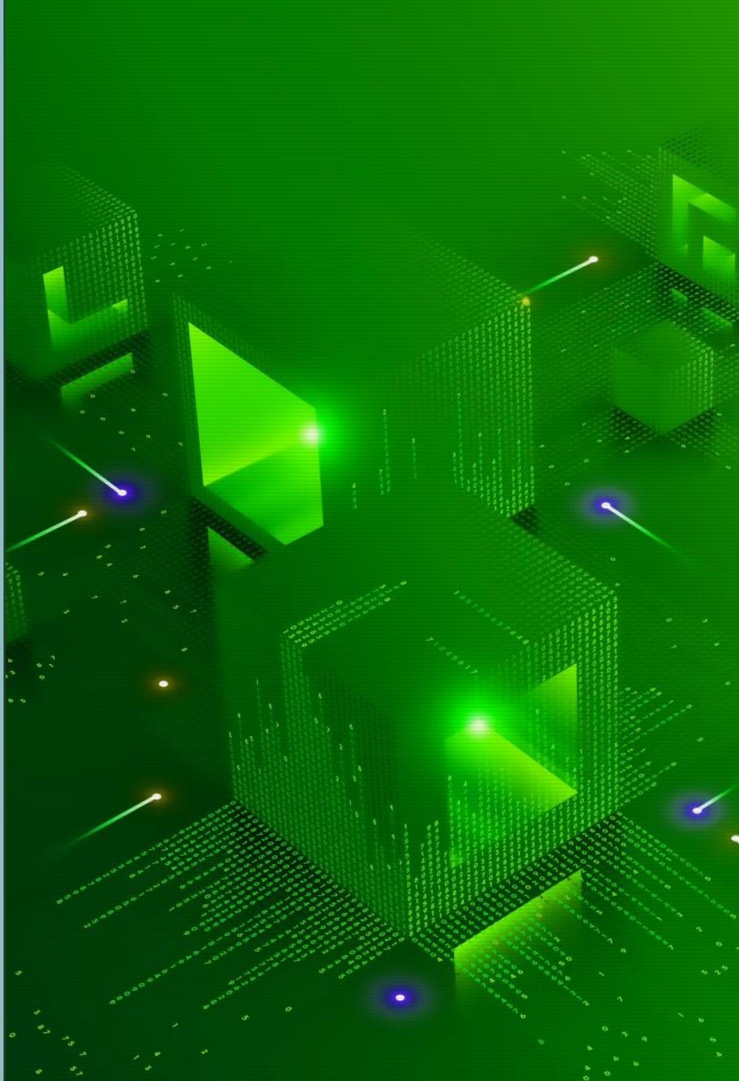
- Trying to create a conceptual architecture on a 2nd-level subsystem was more difficult than the creation of the top-level system as there wasn’t as much documentation detailing the interactions within the subsystem
- Given that Apollo is a large system it is expected that there will be many different interactions at different levels, however, the limitation of the Understand tool underpins the difficulty of being able to cohesively map out those interactions in a digestible fashion
- Generating sequence diagrams which align with the nuances of a concrete architecture can be deceivingly difficult. To avoid potential errors, these diagrams should be verified by other members of the team and iterated upon
- Ratifying our vision with the actuality of an architecture is a difficult task. In a professional development setting, these difference could lead to delays in production time or a worse product

Team Issues

- Given the extremely large nature of the system, it is difficult to map out with how different sections interact with each other. Being able to expand on current features of different subsystems would require extensive knowledge of the interactivity and system call impacts.
- Not all sections of the code have comprehensive commenting, therefore raising difficulty in comprehension of the system and its interactivity
- Limited documentation from a coding perspective of certain modules, such as dreamview backend and how it operates, along with how to build it locally for a coding environment. This also extends to the concern to keeping standard coding practices for maintaining the submodules properly for future developers working on this project.

Concurrency

- Concurrency is built in the system on Cyber RT, which acts as the centralizer framework that enables the modules for efficient parallel processing and communication
- Given the publish and subscribe nature of the architecture, it allows developers to add new modules which will be supported by the Cyber RT framework
- With low-latency communication, along with parallel computation and addition of new modules, this system allows for comprehensive concurrency communication between all needed dependencies of the subsystems



Conclusion

- In conclusion, while the concrete architecture for Apollo was not too unexpected compared to our conceptual, there were several dependencies that we had not accounted for, as well as a couple of new modules. The most notable one was Cyber RT, which acts as a connector for all the modules. We also analyzed the subsystem architecture for the Control module and found some divergences due to the controller.h file and the common submodule. Overall, we saw the ways in which a system’s concrete architecture is not a complete match to its conceptual architecture, and learned how to analyze these differences.

References

- [1] *Understanding model predictive control*. MATLAB & Simulink. (n.d.). Retrieved March 19, 2022, from https://www.mathworks.com/videos/series/understanding-model-predictive-control.html

- [2] *Apollo/modules/common at master · Apolloauto/apollo*. GitHub. (n.d.). Retrieved March 19, 2022, from https://github.com/ApolloAuto/apollo/tree/master/modules/common

- [3] *Apollo/modules/control at master · Apolloauto/apollo*. GitHub. (n.d.). Retrieved March 19, 2022, from https://github.com/ApolloAuto/apollo/tree/master/modules/control

- [4] Marshall, D., & Graff, E. (2021, December 14). *GFlags - Windows Drivers*. Windows drivers | Microsoft Docs. Retrieved March 19, 2022, from https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/gflags