Apollo

https://youtu.be/Cjmn6EQjRHs

Names and Roles

- Group Leader - Ryan Saweczko

  Report/Presentation: Abstract, Top Level Conceptual & Concrete Description, Lessons learned, Miscellaneous help to others

- Presenter 1 - Briggs Fisher

  Report: Introduction, proofread

  Presentation: Introduction, Conclusion

- Presenter 2 - Itay Ben Shabat

  Report/Presentation: Sequence Diagrams

- Tom Lin

  Report/Presentation: Detailed Explanation of Perception Module, Creating Conceptual and Concrete Architectures for it

- Douglas Chafee

  Report/Presentation: Reflexion Analysis of Top-level and Perception Module

- Karl Dorogy

  Report: Conceptual/Concrete Top-Level Architecture, Derivation Process

  Presentation: Derivation Process

Introduction

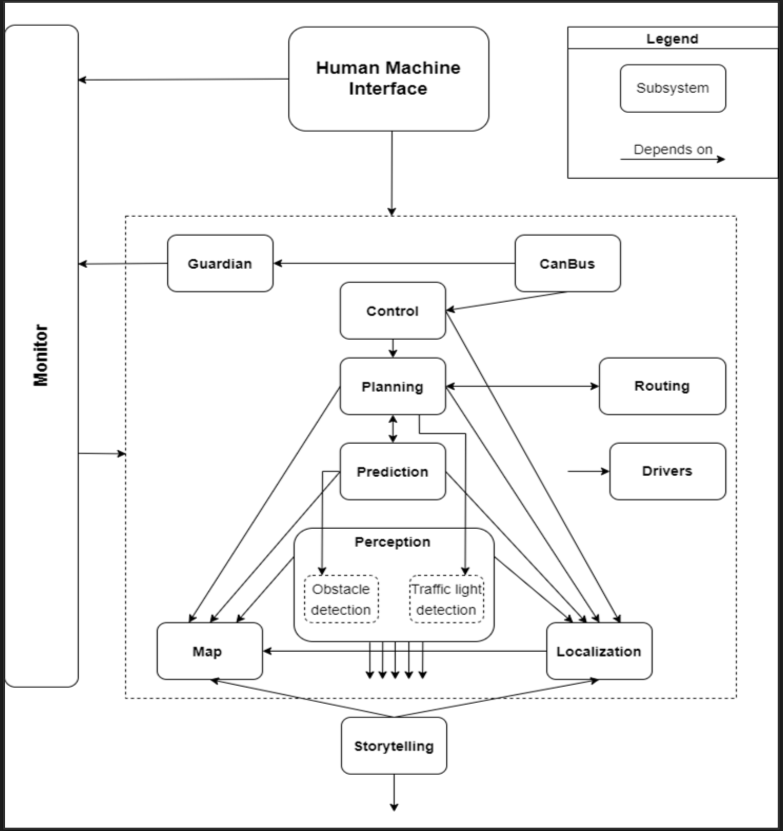
- This report is an analysis of the Apollo autonomous driving open software's concrete architecture. We then do a reflexion analysis of the conceptual and concrete architectures to determine the reason for divergences.

- The report is split into several sections including the Derivation process of the concrete architecture, a description of the top-level subsystems and their interactions followed by a reflexion analysis of the architectures. We then investigate the perception module, creating a conceptual and concrete architecture for this system followed by a reflexion analysis of that. We created sequence diagrams for two use cases in our conceptual architecture. We provide a data dictionary of key terms and unabbreviate all reported acronyms in naming conventions. Finally, we recall the lessons we learned.

Conceptual Architecture (Updated)

Figure 1: Updated Conceptual Architecture of Apollo from Assignment 1

The original conceptual architecture from Assignment 1 didn’t include the Storytelling module as the main difference. When we were looking at other group’s conceptual architectures, and re-referenced Apollo’s documentation, we found we had missed this as it’s own module, and so added it.



Derivation Process - Part 1

Understand™ by SciTools

An initial group meeting, where after importing the provided apollo.udp file into the Understand software tool as a group & we:

- Created a new architecture, creating corresponding components for each subsystem that was derived from our updated conceptual architecture from A1.
- Analyzed the Apollo system's source code (directories, folders, files), mapping each folder or file of source code to its corresponding subsystem within the newly created architecture.
  - For example the directory/folder labeled "planning" was mapped to the Planning subsystem.
  - For the specific cases where source code wasn't easily mappable, looking at documentation, opening individual classes, reading developer comments, and inspecting the names/invocation of methods were used to gain further understanding.
- This process was then repeated a couple more iterations within a couple more group meetings until finally all the source code was mapped to a desired subsystem within the new architecture and a graph visualization of Apollo source dependencies could be generated.

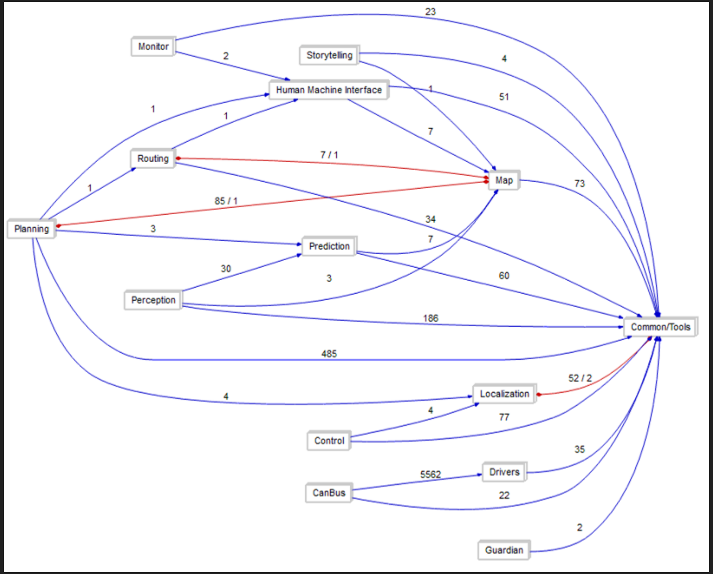


Figure 1: Graph visualization of Apollo source code dependencies from Understand tool.

Derivation Process - Part 2

Then combining both the graph visualization of Apollo’s source dependencies and the provided publish subscribe (pub-sub) communication graph shown in Figure 1, we then ultimately concluded Apollo’s concrete architecture.

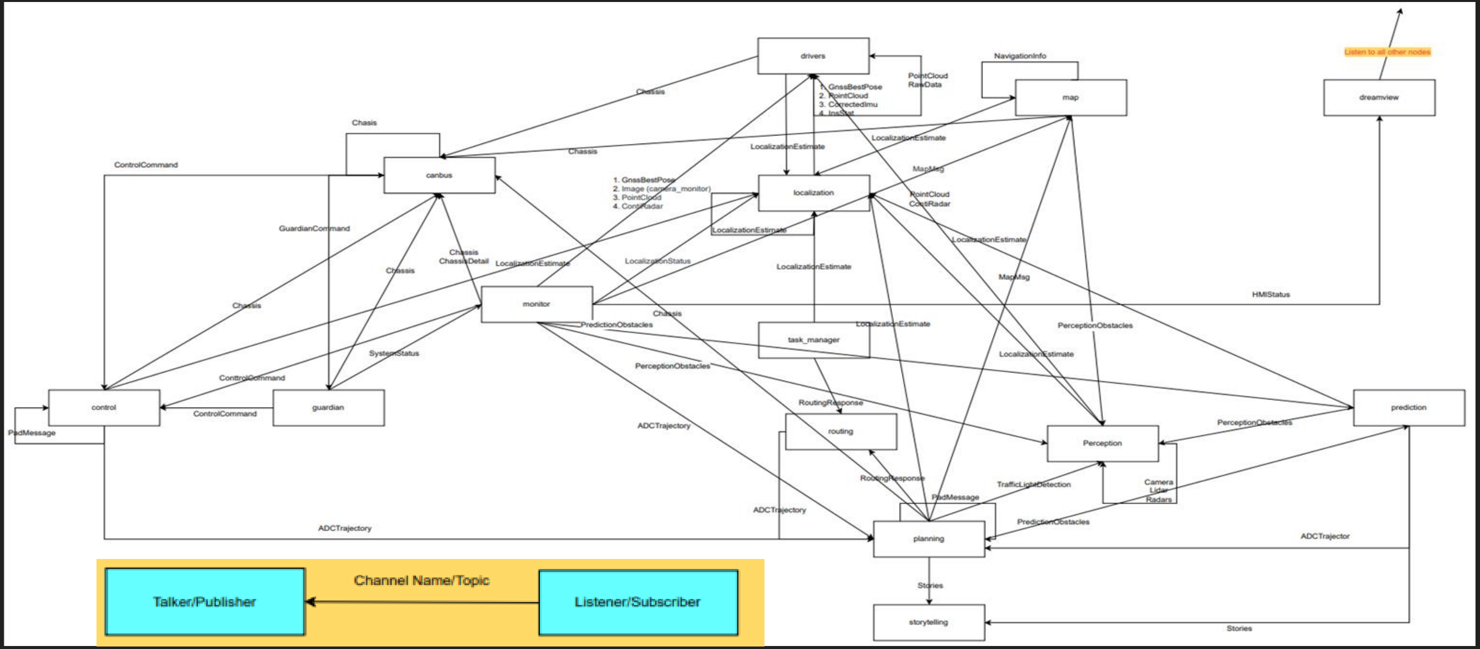


Figure 1: Graph visualization of Apollo system pub-sub message traffic.

Concrete Architecture

We found the concrete architecture to have several new dependencies creating discrepancies from the conceptual architecture.
These discrepancies include the monitor depending on different modules that we had assumed, the modules depending on drivers, more dependencies for the prediction module, and more.

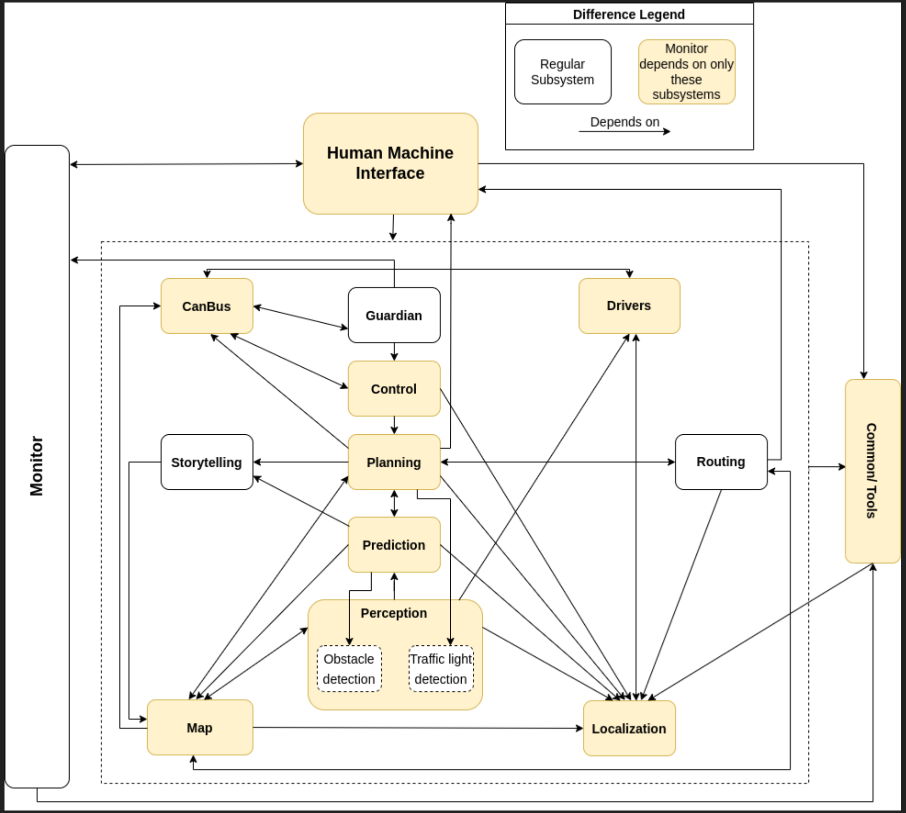


Figure 1: Concrete Architecture of Apollo

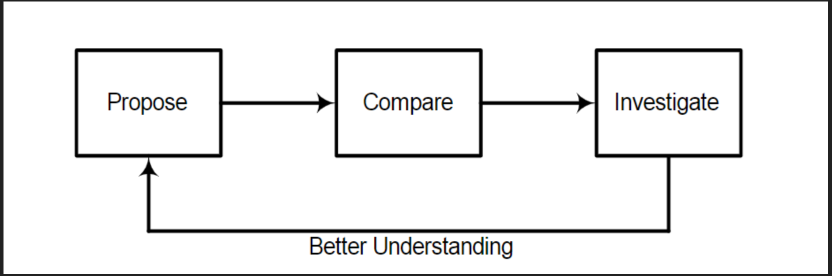
Reflexion Analysis Of Top Level - Some Major Divergences

1. New Module: Common/Tools:

Important general utility module used by other modules to obtain system information about the vehicle, and publish buffered information to monitor.

- 2. Common/Tools depends on Localization:
- Added in November of 2017; Common's vehicle_state_provider and vehicle_state_provider_test use localization to find the vehicle's current position. This is a required dependency as vehicle_state_provider is used by other modules to access general information about the vehicle such as dimensions, center of mass, and position.

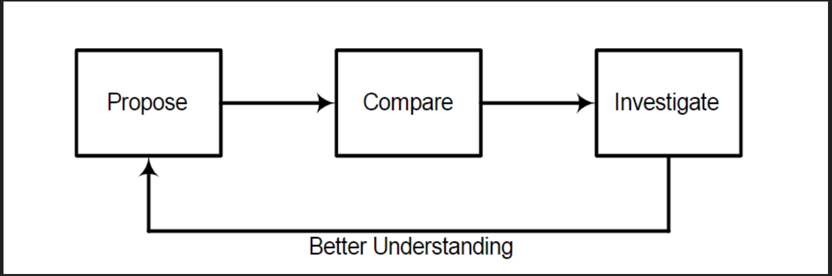
- 2. Every module depends on Common/Tools:
- There are five frequently used files; monitor_log_buffer.h, map_util.h, status.h, vehicle_config_helper.h, adapter_gflags.h.
- monitor_log_buffer allows MonitorBuffer objects to publish their logs to monitor topic.
- Map.util is frequently used to help generate maps.
- Status.h is used for evaluations/comparisons and generating readable error messages for the user.
- Vehicle_config_helper is used to access vehicle information such as length, width, height, center mass, and position on map / route.
- adapter_gflags is used to monitor sensor calibration process.



Reflexion Analysis Of Top Level - Major Absence?

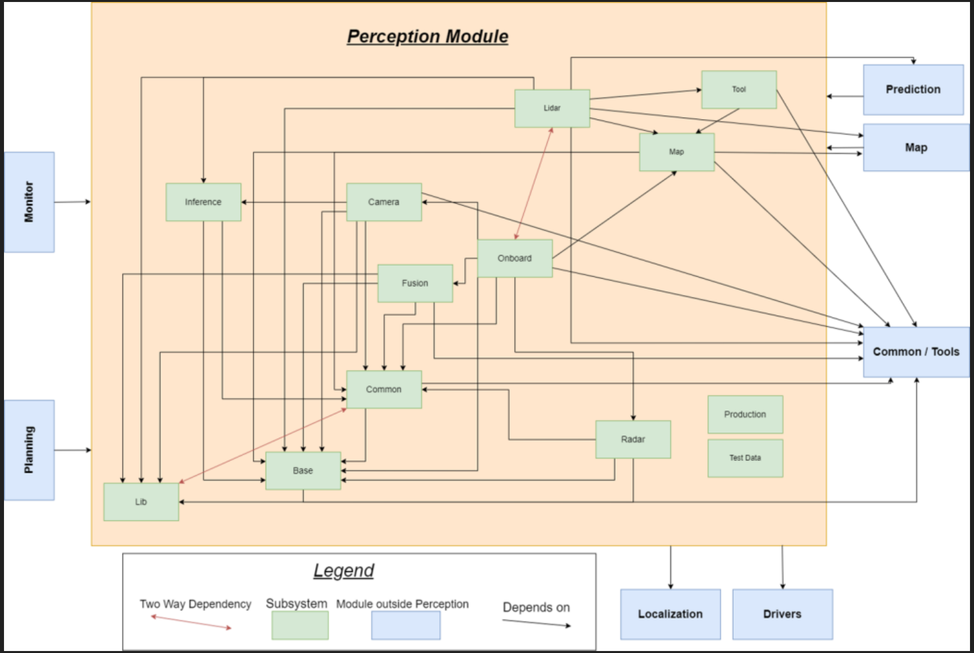
3. Routing depends on Common/Tools

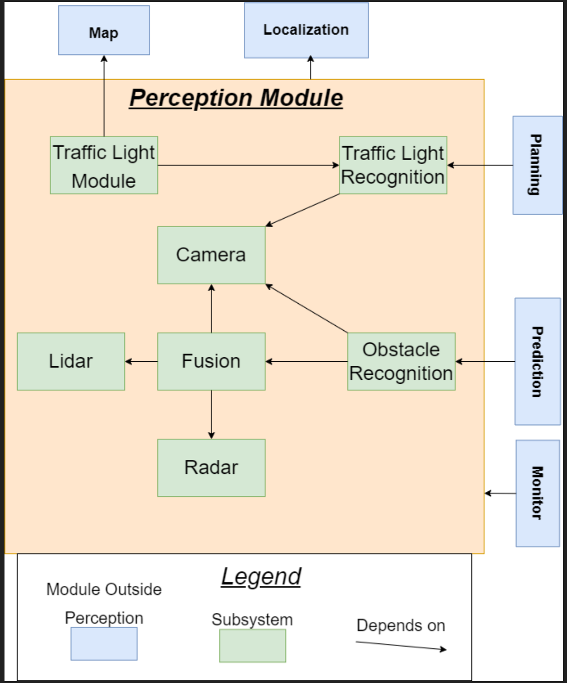
There are scenarios such as parking_routing_manager where routing needs the vehicle dimensions provided by vehicle_config_helper found in Common to verify parking locations large enough for the vehicle. This dependence was added during July of 2021



The Perception Module

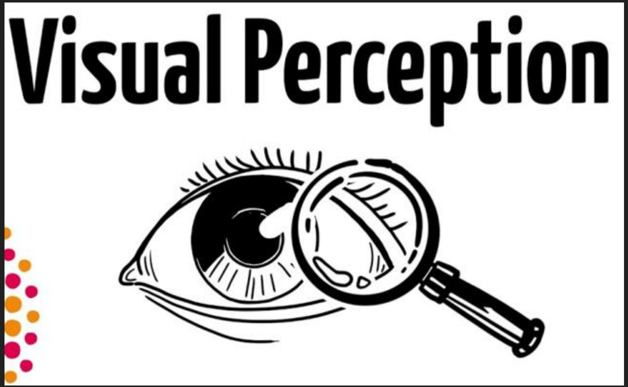
From observing the concrete and conceptual architectures, we can deduce the perception module heavily utilizes images, models, and image processors in order to compute incoming obstacles. If we look at the concrete architecture, almost all submodules has a dependency into Common or Base, which contain the aforementioned images, models, and image processors.

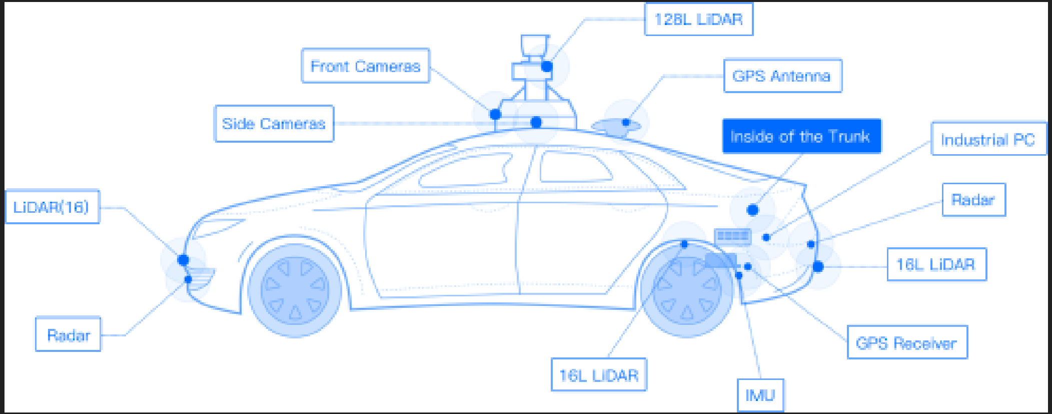




The Perception Module

The most notable submodules would be Lidar, Radar, Camera, and Onboard. We can infer this from looking at both the Concrete and Conceptual Architectures. In the Conceptual Architecture, Lidar, Radar, and Camera are core submodules in the process of recognizing an obstacle. In the Concrete Architecture, Lidar, Radar, Camera, and Onboard all have little incoming dependencies and many outgoing dependencies. We can deduce from this that these four are core submodules in creating the obstacle avoidance / traffic light routine.

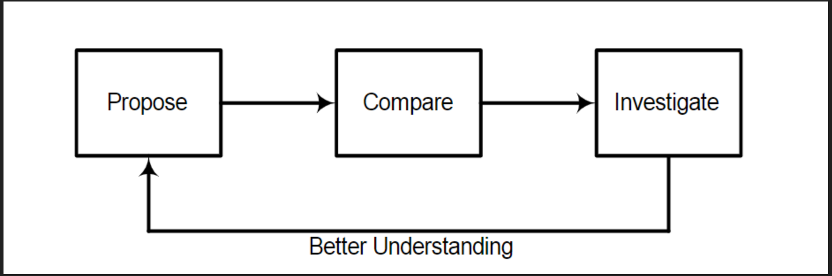




Reflexion Analysis Of Perception Module

- Perception/Map depends on Map as there is a “#include” for the Map modules hdmap_util.h file. However, this include is not used and there is a comment block from 2018 describing how its use as a xml map pointer is one of two options for retrieving a map. It appears option two was chosen making this dependency irrelevant.

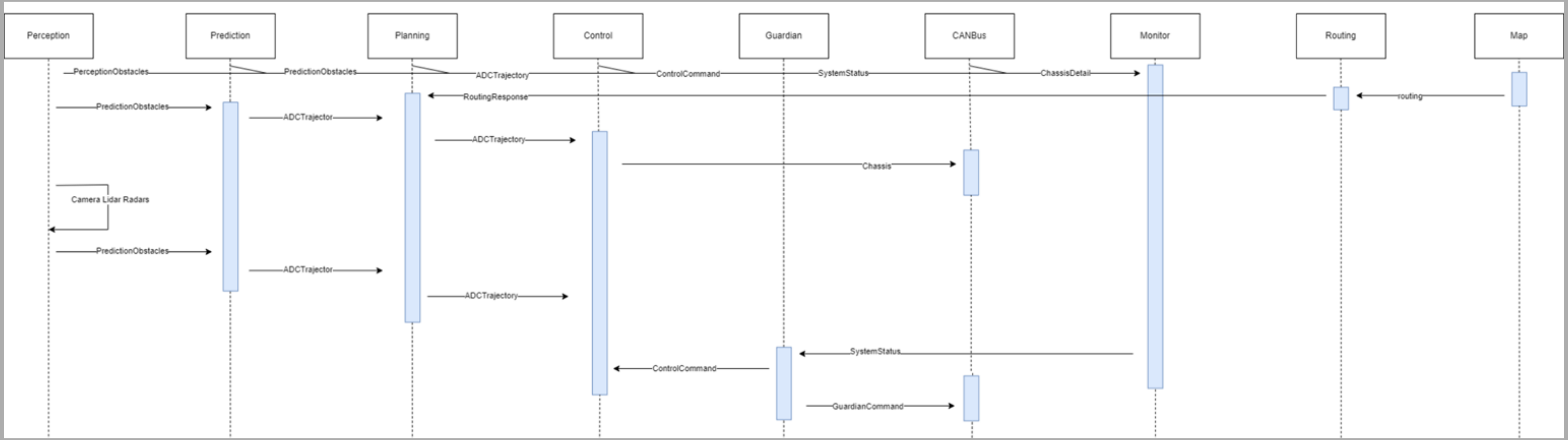
- Perception/Lidar depends on Prediction for feature_output.h for offline obstacle detection in surrounding lanes as well as the vehicle’s lane. This dependency was added January of 2019 and needed because perception also tracks the vehicle’s location using semantic_map from predictions. These imports allow perception to map the position of the vehicle and surrounding objects, information requested by other modules.



- Perception/Camera’s cipv_camera.h depends on Base’s lane_struct.h to create new lane objects as required by the perceived surrounds through the 360 degree camera. This dependency was added during April of 2019 and required as perception needs to identify all the surroundings of the vehicle, and lanes are an essential part of urban driving.

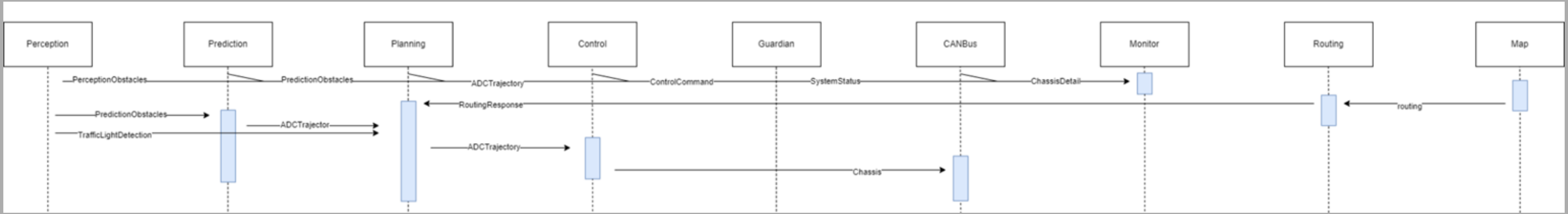
Sequence Diagram

The car is on the road and the car in front slows down but something breaks in the vehicle as it is slowing down



Sequence Diagram

The car is on the road and the traffic light turns red



Lessons Learned

Through creating the concrete architecture, we learnt a lot about how to read code to understand what is happening, and find the dependencies between modules. Understand and the professors graph of pub-sub messages helped us find which dependencies existed, but when determining the rationale for divergences, we needed to learn how to find the proper git commit with the changes, and see what was used where.
We learnt how to use Understand with entity names to create conceptual and concrete architectures for the perception module, which went into code

Conclusion

- Learned how to use Understand to create a concrete architecture for Apollo
- Compared the concrete architecture to the conceptual architecture for a reflexion analysis
  - Found one module absence, one module divergence and twelves divergences between the dependencies when comparing the two architecture types
- Analyzed the perception module to create both a conceptual and concrete architecture
  - Did a reflexion analysis over these architectures, finding several divergences
- Created two sequence diagrams to show use cases using the concrete architecture found

- Learnt a lot in the process!