Apollo

https://youtu.be/hAqB9MjMmms

Names and Roles

- Group Leader - Ryan Saweczko

  Report/Presentation: Concurrency, Abstract, Limitations

- Presenter 1 - Briggs Fisher

  Report: Introduction

  Presentation: Introduction, Subsystems

- Presenter 2 - Itay Ben Shabat

  Report/Presentation: Sequence Diagrams

- Tom Lin

  Report/Presentation: Developer Responsibilities and Data Flow

- Douglas Chafee

  Report/Presentation: System Evolution, Conclusion

- Karl Dorogy

  Report: Subsystems, Derivation Process

  Presentation: Derivation Process

Conceptual Architecture - Ryan, Karl, Tom

Introduction

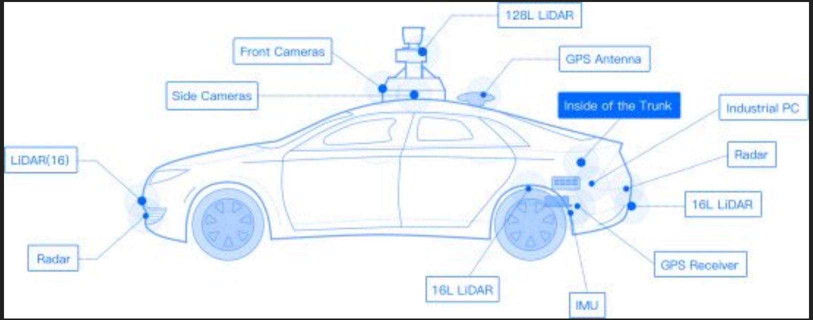
- This report is an analysis of the Apollo autonomous driving open software's conceptual architecture. We will discover how Apollo competes in the ever expanding autonomous driving market through an analysis of their high level architecture.

- The report is split into seven sections including the Derivation Process, Conceptual Architecture (and its eleven subsystems), System Evolution, Developer Responsibilities and Data Flow, Concurrency, Sequence Diagrams, and a batch of the limitations, conclusions, and lessons we learned.

- The report will ultimately define Apollo's high level architecture as implicit invocation with insights on Layered, Pipe and Filter, and Repository styles found within the submodules.

Derivation Process

In documenting and formulating Apollo’s conceptual architecture, the derivation process had three major steps, the first being:



Background Knowledge & Understanding

- Each group member was assigned to read through "A functional reference architecture for autonomous driving" (Behere et al.)
- Individually research through the documentation of Apollo (GitHub & Apollo Website)
- Drafting a preliminary version of the conceptual architecture based on their understanding of the information they had read in addition to the provided lecture material within CISC 322/326.

Derivation Process

In documenting and formulating Apollo’s conceptual architecture, the derivation process had three major steps, the second being:

Initial Group Meeting & Discussion

- Once each member had developed a possible conceptual architecture type, a group meeting was set and held.
- The entire group as a whole discussed the possible alternative architectures that each group member found and or created as well as their respective non-functional pros, cons, and invariants in regards to an autonomous driving software system.
- After a long discussion we picked which modules/components should be included and excluded from the conceptual architecture as well as its overall type of being implicit invocation.
- Ended the meeting by dividing up all the major points we wanted our documentation to cover and each group members' respective designated tasks that individuals were responsible for.

Alternative Architectures

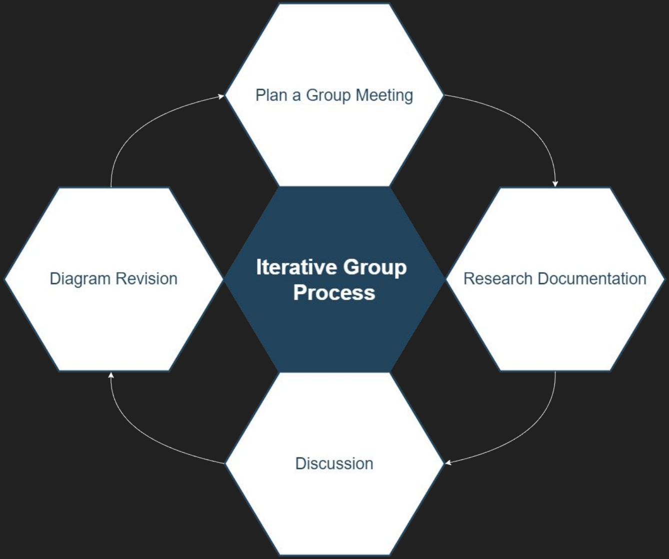
- Layered
- Pipe and Filter
- Implicit Invocation
- Repository Style

Derivation Process

In documenting and formulating Apollo’s conceptual architecture, the derivation process had three major steps, the third & final being:

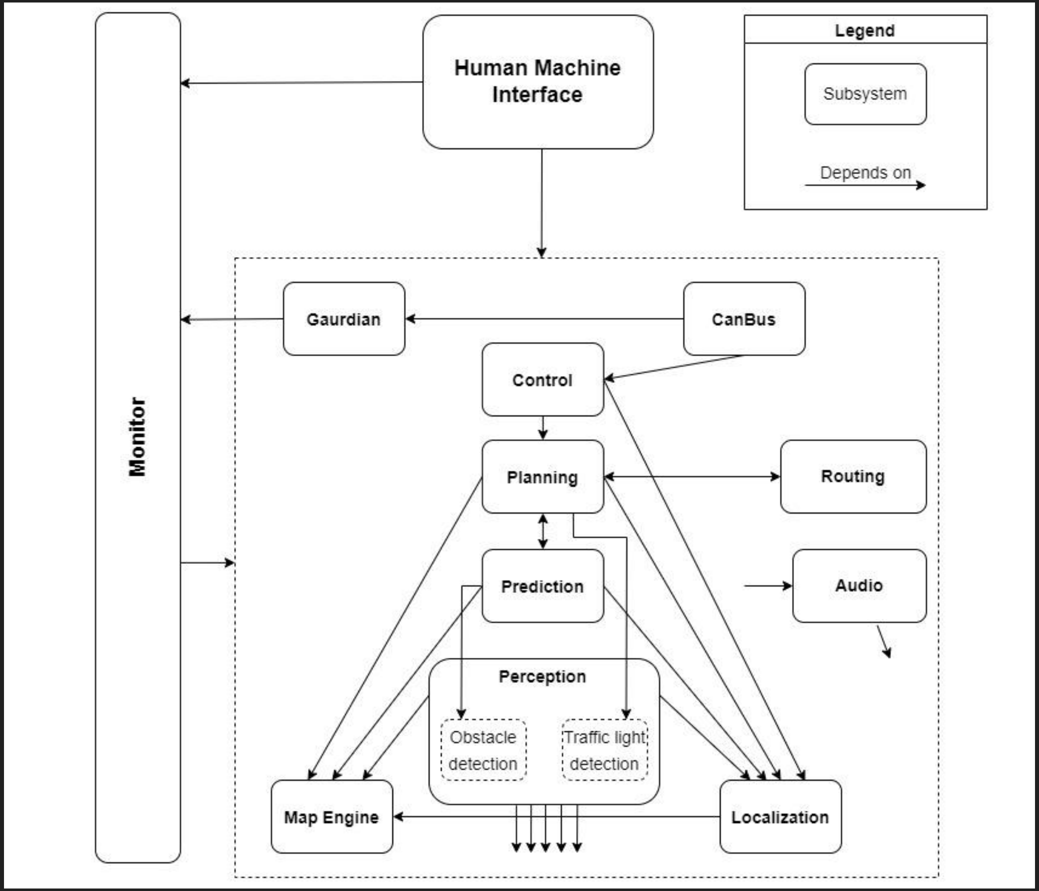
Iterative Group Meeting & Finalization

- After individuals have done some work on the documentation of their tasks and upon new understandings/findings within the documentation for apollo’s open-source platform for autonomous driving cars, additional meetings are set and held weekly.
- Our group iteratively improved on our initial conceptual architecture and sequence diagram user cases, adding additional modules such as routing and audio in addition to discussing their involvement within the system and possible new dependencies that have arisen between components.
- Ultimitally finalizing our conceptual architecture of Apollo



Conceptual Architecture

Figure 1: Conceptual Architecture of Apollo



Implicit Invocation

- Using real-time operating system CyberRT with cyber channels as an eventing system to pass data between modules
- Each dependency arrow indicates a module is subscribed to an event the preceding module emits.

Subsystems

1. Perception

Outputs a 3D obstacle track with the heading, velocity, and classification information for any nearby obstacles or traffic lights recognized by the perception module's two submodules.

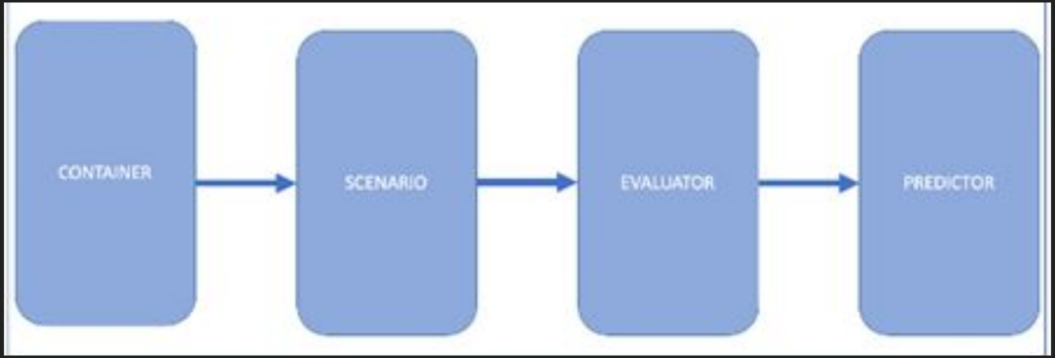
Obstacle Perception Submodule

- Detects, classifies, and tracks obstacles in the ROI that is defined by the high-resolution map.
- Includes LiDAR and RADAR based obstacle perception

Traffic Light Perception Submodule

- Obtains the coordinates of traffic lights in front of the car by querying HD-Map/MapEngine
- A larger ROI is created where the traffic lights are detected with a surround bounding box and recognized as different color states.

Subsystems



2. Prediction

Studies and predicts behaviors of obstacles detected by the perception module.

3. Planning

Plans a collision-free and safe trajectory for the control module to execute by interacting with other modules.

4. Control

Manages vehicle control commands (steering, throttle, brake) based on the planned trajectory and the car's current status.

5. Canbus

Interface that accepts and executes control module commands to the vehicle's hardware.

6. Human Machine Interface

Provides an interface to enable/disable modules, view hardware status, start the autonomous driving car as well as debugging tools.

Subsystems

7. Localization

Aggregates various data to locate the autonomous vehicle and provide localization services. Provides localization using GPS, IMU, and LiDAR information.

8. Monitor

Ensures all modules are working and passes its data on to the HMI for the driver to view.

9. Guardian

Intervenes if the Monitor module detects a failure and provides countermeasures

10. Audio

Detects siren sounds and displays an active alert on the Dream view web application (HMI) if an emergency vehicle is detected.

11. Routing

Tells the autonomous vehicle how to reach its destination via a series of lanes or roads depending on a topology file.

System Evolution

Apollo release 3.5 is the most important architectural change with the implementation of monitor, and guardian.

- Monitor takes input from every component and has guardian interrupt control commands so the CANBus can correct errors and fix failures.

Developer Responsibilities & Data Flow

From observing the software architecture, we can determine there are four major development groups in Apollo.

- Perception, Prediction, Planning → Group 1
- Control → Group 2
- HD Map, Localization, CANBus, Guardian → Group 3
- Monitor, HMI → Group 4

Developer Responsibilities & Data Flow

Group 1

Group 1’s main responsibility would be to make the “autonomous” of “autonomous driving car” work.

- Perception: ensures radars and cameras are functional
- Prediction: ensures A.I understands obstacles
- Planning: ensures A.I gives correct instructions to Control

Developer Responsibilities & Data Flow

Group 2

Group 2’s main responsibility would be making sure the Control module is functional.

- Control: ensures the instruction (braking, steering, throttling) gets passed to CANBus

Developer Responsibilities & Data Flow

Group 3

Group 3 would be responsible for ensuring the other parts of the group were functional.

- HD Map: ensures the map is giving an accurate representation of the road
- Localization: ensures the module is giving an accurate prediction of where the car is
- CANBus: ensures the module is executing the commands given from Control
- Guardian: ensures the module is intervening in cases where Monitor detects a failure

Developer Responsibilities & Data Flow

Group 4

Group 4 would be responsible for monitoring the other groups as well as provide an interface for viewing the status of the vehicle.

- Monitor: Ensures the module is monitoring all the other modules as well as feeding Guardian with adequate information
- HMI: Maintains the interface to ensure that it is providing the correct statuses of the vehicle

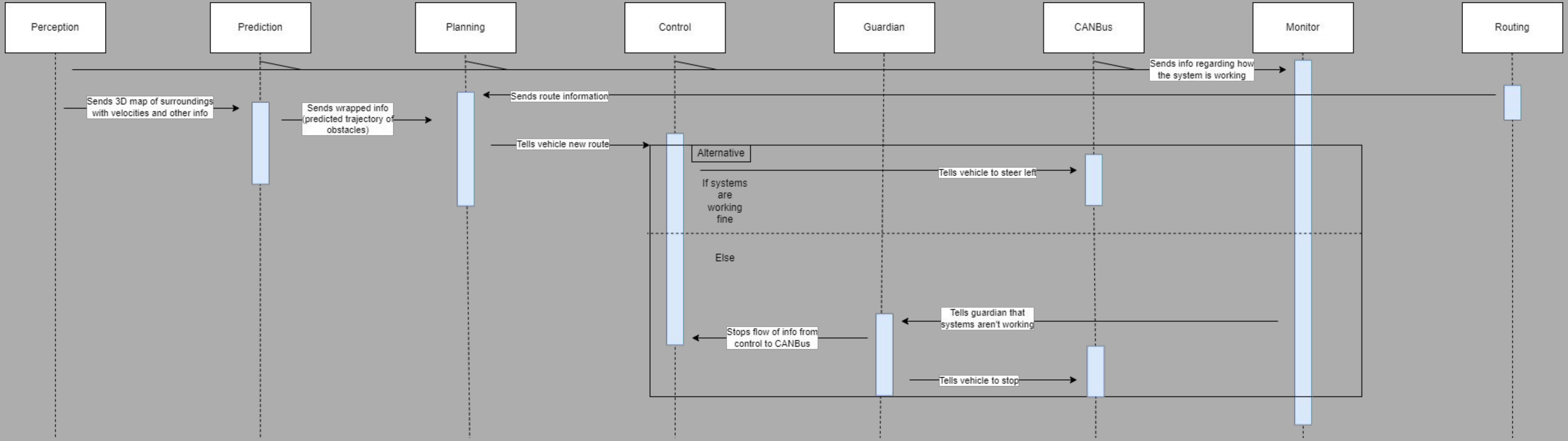
Concurrency

Most of the system is very concurrent. A few specific examples of concurrency within Apollo are:

- The monitor module, which runs concurrent to all other software to monitor the system’s health
- The camera systems that all have drivers running concurrently to everything else in the system to provide a continuous input field.
- The HMI (human-machine interface), which is a web server that is run beside the rest of the system, providing a visual of what the autonomous car is doing.

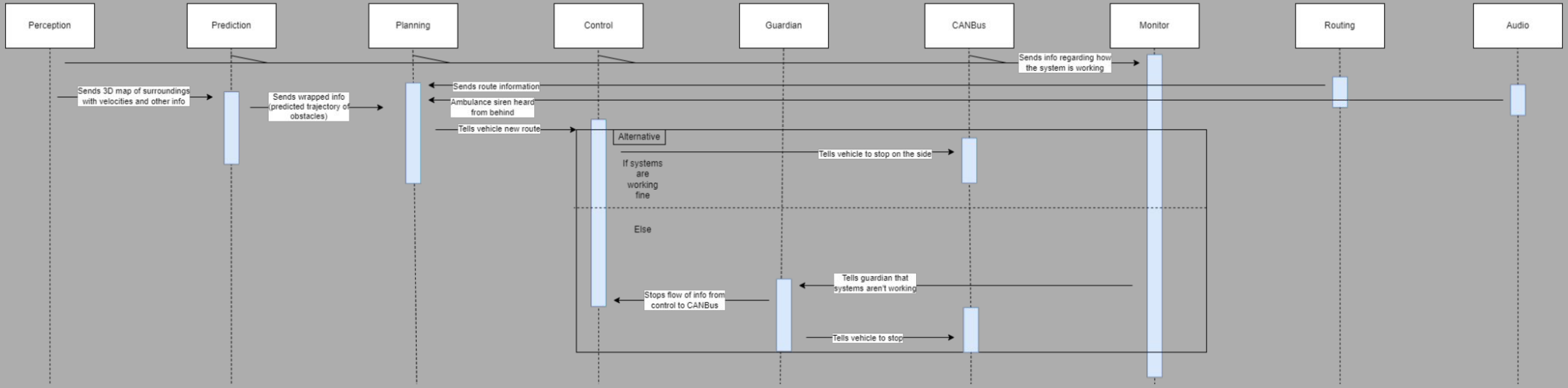
Sequence Diagram

Use case when the car is driving and the road twists left


Perception
Prediction
Planning
Control
Guardian
CANBus
Monitor
Routing
Sends info regarding how the system is working
Sends route information
Sends 3D map of surroundings with velocities and other info
Sends wrapped info (predicted trajectory of obstacles)
Tells vehicle new route
Alternative
Tells vehicle to steer left
If systems are working fine
Else
Tells guardian that systems aren't working
Stops flow of info from control to CANBus
Tells vehicle to stop

Sequence Diagram

Use case when an ambulance is heard


Perception
Prediction
Planning
Control
Guardian
CANBus
Monitor
Routing
Audio
Sends info regarding how the system is working
Sends route information
Sends 3D map of surroundings with velocities and other info
Sends wrapped info (predicted trajectory of obstacles)
Ambulance siren heard from behind
Tells vehicle new route
Alternative
Tells vehicle to stop on the side
If systems are working fine
Else
Tells guardian that systems aren't working
Stops flow of info from control to CANBus
Tells vehicle to stop

Limitations and Lessons Learned

Some limitations of the architecture being used are the sub-optimal calling of procedures that a broadcasting system may cause. This system has a lot of data being transferred from cameras, microphones, and other peripherals, and a pipe-and-filter architecture might be able to streamline this aspect of the software better.

We learnt a lot about the process of discovering the conceptual architecture of a system through the documentation. Learning how to read and interpret the documents, and finding different resources from the repository, website, and research papers.

Conclusion

The Apollo autonomous driving system uses a specialized implicit invocation architecture and the control component uses a process-control architecture.

- Implicit invocation is an event handling architecture essential for automated driving as the vehicle needs fast scenario based decision making to match the systems environment.
- Process-control architecture is also critical as the vehicle controls are always changing so the vehicle can reach its target destination while maintaining its target speed and safety protocols.