Conceptual architecture analysis of OpenPilot

Jerry Wu (217545898), **Connor Francis McGrath** (216869133),
Joseph Spagnuolo (216279184), **Tarun Bhardwaj** (217889742),
Krishna Raju (218199497), **Irsa Nasir** (218069567),
Aish Singh (217521147), **Stanley Ihesiulo** (216985236)

Due Feb 14 2024

Contents

1 Abstract

This paper will outline the overarching architecture of the open source system "Open-Pilot" by comma.ai.

2 Introduction and Overview

2.1 The big picture

Designed by comma.ai with the intent of increasing road safety, openpilot is an Advanced Driver Assistance System. It offers many driving safety features such as: Adaptive Cruise Control, Automatic Lane Centering, the ability to observe the attentiveness of the driver, and pathfinding capabilities that allow for automated steering and breaking for other vehicles on the road [3].

The purpose of this report is to provide an analysis of the conceptual architecture of the openpilot. As will be discussed later, openpilot is the amalgamation of many different architecture systems, who in turn are composed of many different subsystems that take raw data from peripherals and hardware attached to the car, process the data, sends their output to a neural network which outputs commands that the car can execute [5].

In order for the car to actually be controlled [1]

2.2 Components and architecture

This section will outline the various architectures and components involved in making the system work. They are:

- **Conceptual architectures**
  - The software utilizes an implicit invocation architecture which can be found in their use of cereal; comma.ai's own open source library for robotic message communication.
  - Implicit invocation is also used in the boardd class which acts as the "publisher". The class manages messages sent/received to/from the panda module.
  - A pipe and filter architecture is exhibited when data from sensors and actuators in the car itself are fed through openpilot's logic and processed by the neural networks working behind the scenes.

  - This may be unrelated, but the open source codebase is stored on a GitHub repository, so this could also be a repository style architecture for when developers are working on the project.

- **Components**
  - Sensors + actuators (boardd, camerad, sensord, etc.)
  - Neural network runners (modeld, dmonitoringmodeld, dmonitoringd, etc.)
  - Localization + calibration (locationd, calibrationd, etc.)
  - Controls (radard, planners, controlsd, etc.
  - System logging/misc (manager, thermald, etc.)
  - Hardware components (panda module)

2.3 How do these components interact?

3 Diagrams

4 External Interfaces

5 Use Cases

6 Data Dictionary

7 Naming Conventions

8 Conclusion

9 Lessons learned and Alternatives Presented

10 References