Conceptual architecture analysis of OpenPilot

Jerry Wu (jerrywu0@my.yorku.ca), **Joseph Spagnuolo** (joe13@my.yorku.ca),
Tarun Bhardwaj (tarunb4@my.yorku.ca), **Krishna Raju** (krishnar@my.yorku.ca),
Irsa Nasir (inasi022@my.yorku.ca), **Aish Singh** (aish19@my.yorku.ca),
Stanley Ihesiulo (ihesiulo@my.yorku.ca),
Connor Francis McGrath (connorm3@my.yorku.ca),

Due Feb 14 2024

Contents

1 Abstract

This paper will outline the overarching architecture of the open source system "OpenPilot" 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 braking 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]. This report will go into detail about the larger systems, their subsystems, the architectural styles that can be derived from their interactions with each other.

In order for the car to actually be controlled, openpilot takes advantage of the automobile's CAN bus. The panda device allows the software to send the information collected from the peripherals to the mechanisms of the car [1]. This report will also describe the processes which convert the inputs from peripherals into actionable commands that can physically affect the vehicle.

Important to mention that this report will not just describe the software within the system, but the external interfaces the system retrieves information from, and the interfaces it outputs information to, such as the panda interface mentioned above, as well as devices such as the comma 3, which communicates with the panda interface [4], and is the hardware device that is physically mounted to the car, which the driver interacts with.

In order to demonstrate the capabilities of the openpilot system in the real world, this report will describe several concrete use cases of openpilot in action. The first describes how openpilot determines if it is safe to engage the system, describing how the software components are able to detect the condition of the driver and whether or not the system is safe enough to be engaged. The second use case demonstrates how the software components are able to provide tangible instructions to the CAN bus. This includes longitudinal information, that controls brakes and acceleration, and lateral information that controls steering.

However, it is not important to merely mention the current state of the system, but how comma.ai has set up a system that facilitates the evolution of future iterations of the openpilot system, through logging sensor data and compressed video to train and improve the neural network [5], as well as the open source github repository that welcomes contribution from all users [3].

In order to make the report more readable, naming conventions of the architecture, as well as key terms will be described.

To conclude, the report will also document the lessons learned from the group, as well as proposals for features that openpilot could incorporate into future iterations, or improvements to the existing systems, as well as what the group could have done differently in our exploration of the openpilot software system.

Figure 1: The most pretty panda yet. 9 Oct. 2019. Comma-Ai.medium.com, Medium, `https://comma-ai.medium.com/our-hardware-future-eea980d8c3bd.Accessed9Feb.2024.`

This is the panda interface system that allows openpilot to send the information collected from the peripherals to be transferred to the car. It also allows the comma device to connect to the vehicle [1].

2.2 Components and architecture

Give the overall structure of the studied system, with descriptions of each major component and the interactions among them. These components are to be primarily subsystems or modules, but may also include threads or processes, files, and databases. In your descriptions, concentrate on goals, requirements, evolvability, testability, etc., rather than on lower-level concepts such as classes, variables and control flow. Discuss any parts of the system that are performance critical, i.e., which might not run fast enough. Discuss how the architecture supports future changes in the system. You should clarify global control flow, such as units of concurrency and method of passing control from one component to another. Your system's architecture should be easy to understand, with simple interfaces, and modest interactions among subsystems and modules. Clarify the architectural style (in the sense of Garlan and Shaw) that characterizes the overall system and its various parts. You are not to concentrate on minor components, such as classes and procedures, which are smaller than packages or modules. However, you may wish to discuss important abstractions, patterns, classes, data structures or algorithms that are critical to the success of the architecture.

- **Architectures used**
  - 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 (modeId, dmonitoringmodeId, dmonitoringd, etc.)
- Localization + calibration (locationd, calibrationd, etc.)
- Controls (radard, planners, controlsd, etc.
- System logging/misc (manager, thermald, etc.)
- Hardware components (panda module)

2.3 Architecture in depth

- **General OpenPilot Architecture**:
  In general, OpenPilot can be depicted as a layered style architecture which contains 8 layers.
  - Layer 1: Vehicle Interface
  - Layer 2: Communication and Data Interpretation
  - Layer 3: Sensing and Actuation
  - Layer 4: Core Neural Network Runners
  - Layer 5: Localization and Calibration
  - Layer 6: Control Algorithms
  - Layer 7: System Management and Logging
  - Layer 8: User Interface and Experience

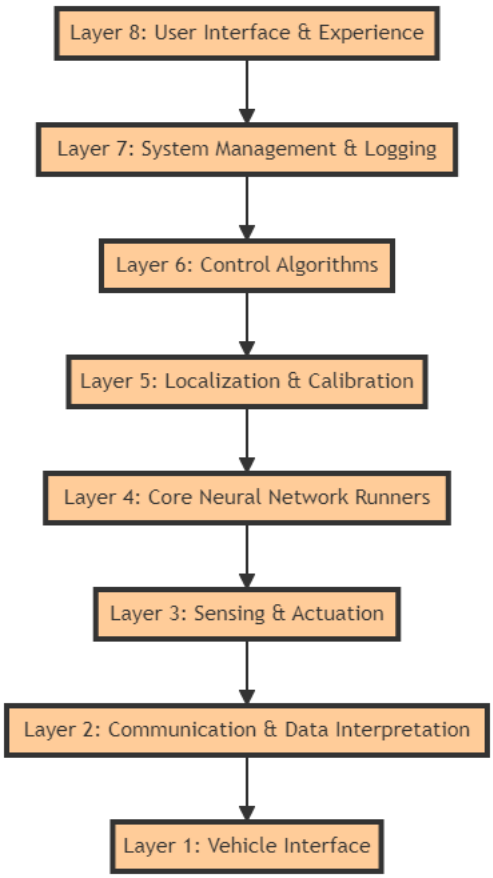

Figure 2: A bottom up diagram detailing the general architecture of openpilot

- The **vehicle interface layer** includes the car's hardware and peripherals that interface with OpenPilot. This includes **CAN** (Controller Area Network) interface which enables communication with the car's internal network, **steering wheel angle** sensor measures angle of steering wheel, **camera** provides visual input around the vehicle, **IMU** (Inertial Measurement Unit) also provides movement and orientation information, **GNSS Receiver** provides global positioning data, and along with other vehicle sensors that are used to ensure the vehicle is functioning properly.
- The **communication and data interpretation layer** consists of libraries like opendbc which interprets CAN bus data based on **DBC** files, **panda** which is hardware that is used to read-/write from and to the CAN bus, **laika** which processes **GPS** data for more precise positioning, and **cereal** which is the message protocol for inter-process communication.
- The **sensing and actuation layer** includes **boardd** which manages communication between the panda and the openpilot software, **camerad** which processes image data from the car's cameras, and **sensord** which handles IMU and additional sensor data reading. All together this layer manages the direct interaction with the car's hardware and the data from the sensors.
- The **core neural network runners layer** includes **modeld** which runs the driving policy in order for the neural network to make driving decisions, **dmonitoringmodeld** which runs the driver monitoring neural network to ensure that the driver is paying attention, **dmonitoringd** which contains logic to see whether the driver can take over the wheel if necessary. Overall, this layer runs the neural networks which are responsible for driving and driver monitoring.
- The **localization and calibration layer** includes **ubloxd** which parses GNSS data for localization, **locationd** which uses a Kalman filter service to provide precise vehicle localization, **calibrationd** which calibrates the camera to align it with the vehicle's frame, and **paramsd** which adjusts the vehicle parameters for more accurate control.
- The **control algorithms layer** uses **radard** which processes radar data for detecting objects and tracking them, **plannerd** which laterally and longitudinally calculates the path planning, and **controlsd** which generates and sends the control commands to the actuators of the vehicle.
- The **system management and logging layer** contains **manager** which oversees the lifecycle of openpilot services, **thermald** which monitors the thermal state of the device and ensures it does not overheat, **loggerd** which records driving and analyses that data for improvements, **logcatd** which logs system messages and errors, **proclogd** which logs detailed process data, and **athenad** which manages the connection to common.ai services for updates and any remote assistance.
- Lastly, **the user interface and experience layer** has a **UI** which is in charge of displaying the interface for the user, including a live camera feed, system alerts and status.

- **Global control flow**
- **Concurrency**
  - **Concurrent testing**: Utilises the **pytest** module to achieve parallelization and streamline test suite execution. This allows for concurrent testing, maximising resource use and minimising idle time.
  - **Driving controls system**: The control system handled by the **controlsd** module is responsible for managing autonomous vehicle operations, it relies on concurrent data from other modules and is responsible for executing crucial driving functions including acceleration, braking, and steering.

– **Driver monitoring system**: The driver monitoring system handled by the **dmonitoringd** is responsible to ensure the driver remains alert, It ensures driver attentiveness and readiness to take control when necessary. It concurrently processes data provided by a machine learning model, and video data to decide if the driver needs to be alerted.

2.4 How do these components interact?

3 Diagrams

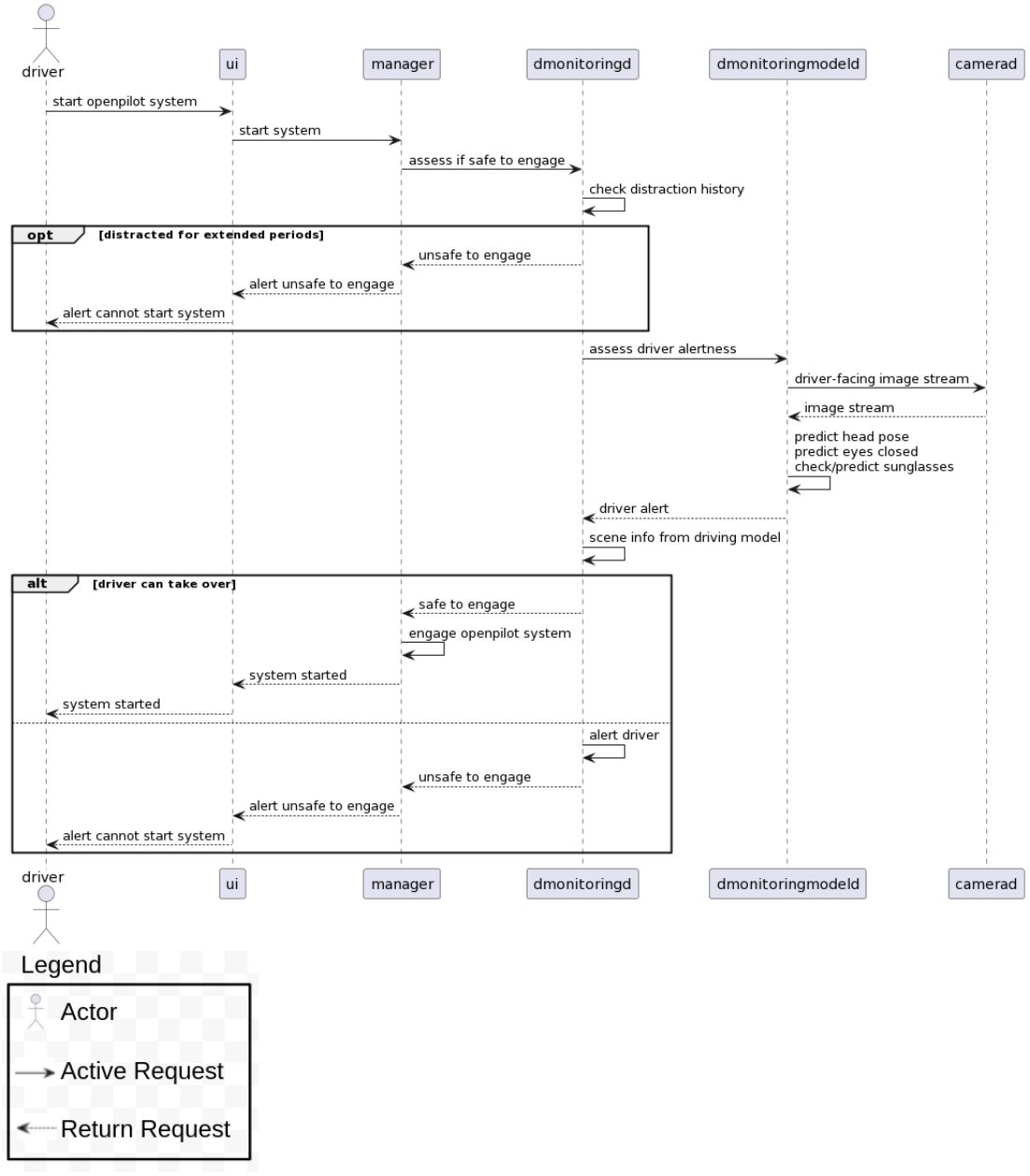

Figure 3: A sequence diagram detailing a case where a driver becomes distracted while openpilot is active

4 External Interfaces

5 Use Cases

6 Data Dictionary

7 Naming Conventions

8 Conclusion

9 Lessons learned and Alternatives Presented

References

[1] S. Anumakonda, "The state of comma.ai," Medium.com, https://srianumakonda.medium.com/the-state-of-comma-ai-2140aabc6f52 (accessed Feb. 13, 2024).

[2] "openpilot 0.9.5," comma.ai, https://blog.comma.ai/095release/ (accessed Feb. 13, 2024).

[3] "Openpilot - Open source advanced driver assistance system," comma.ai, https://comma.ai/openpilot (accessed Feb. 13, 2024).

[4] "From vision to architecture: How to use openpilot and live," From Vision To Architecture: How to use openpilot and live - DESOSA 2020, https://desosa.nl/projects/openpilot/2020/03/11/from-vision-to-architecture (accessed Feb. 13, 2024).

[5] "How openpilot works in 2021," comma.ai, https://blog.comma.ai/openpilot-in-2021/ (accessed Feb. 13, 2024).

[6] "Comma 3x - make driving chill," comma.ai, https://comma.ai/shop/comma-3x (accessed Feb. 13, 2024).