EECS 4314

Assignment 1: Conceptual Architecture Of OpenPilot

AUTHORS:

JOSHUA GENAT	joshgena@my.yorku.ca
NOBAIHA ZAMAN RAYTA	nobaiha@my.yorku.ca
ABHIRAMI VENUGOPAL	abhirami@my.yorku.ca
SAYED MOHAMMED	sayedmohammed1965@gmail.com
AHMED EL-DIB	eldib@my.yorku.ca
JETHRO EMILRAJ	jethro.emilraj@gmail.com
DEXTER EROMOSELE	dexter25@my.yorku.ca
VIVAAN RAJPUT	vivaanxr@protonmail.com

Contents

Abstract

The report on OpenPilot, an advanced driving tool by comma.ai, offers a thorough exploration of its architecture, functionality, collaboration implications for developers, use cases, conclusions, and lessons learned. OpenPilot integrates cutting-edge technologies like machine learning, sensor fusion, and real-time control systems to automate various driving tasks, ensuring safety and enhancing the driving experience. Through detailed use cases, the report illustrates how OpenPilot leverages its architecture to perform functions such as automated steering, cruise control, and driver monitoring. It emphasizes the importance of effective collaboration among developers, modular architecture, and well-defined external interfaces for the successful development of OpenPilot. The report concludes with valuable lessons learned and references for further exploration, highlighting OpenPilot's potential as a promising solution for autonomous driving applications.

Introduction

This report offers an introduction to OpenPilot, a cutting-edge tool developed by comma.ai designed to make driving safer and more enjoyable. OpenPilot is a companion for drivers, assisting with tasks like staying cantered in a lane, adjusting the car's speed based on traffic, and alerting drivers if they get distracted. It works by gathering information from the car's sensors, including cameras and GPS, to make smart decisions that help control the car smoothly during drives. At its core, OpenPilot uses smart technology, including machine learning, to understand the road and how the car is moving. This allows it to offer features like Adaptive Cruise Control (ACC) and Automated Lane Centering (ALC), which are available for certain cars.

The goal of this report is to give the reader a clear picture of OpenPilot's architecture and its use cases. We aim to make this complex system understandable, showing how it evolves, manages data, and ensures everything runs smoothly and safely. The discussed architecture for Openpilot entails a layered approach, comprising hardware interaction, neural networks, localization and calibration, controls, and system management / UI layers. Each layer serves specific functions in managing the complex system. Additionally, the concept of "points" is introduced, offering encapsulated design elements for control and monitoring. Event-based implicit invocation, employing a publisher-subscriber model, facilitates flexible and modular communication between components, enhancing adaptability, fault tolerance, and ease of system evolution. This architecture enables Openpilot to handle real-time data effectively, ensuring prompt responses crucial for advanced driver-assisted systems, while also promoting component reuse and reducing overall system complexity.

This is a starting point for anyone new to OpenPilot, offering the foundational knowledge needed to dive deeper into the project.

Architecture

Functionality and interacting parts

The system achieves this through a combination of machine learning, sensor fusion, and real-time control systems.

Functionality:

- *Automated Steering and Speed Control:* Openpilot autonomously manages steering to maintain the vehicle's position within its lane and adjusts speed according to traffic conditions using adaptive cruise control. Its decision making is informed by machine learning trained on real world driving data to determine the safest route.
- *Driver Monitoring:* Openpilot continuously monitors the driver's attentiveness by analyzing head pose, eye openness, and sunglasses usage. If the driver is distracted for more than six seconds, Openpilot initiates a gradual deceleration, eventually bringing the vehicle to a stop while audibly alerting the user.
- *Geo Positioning and Calibration*: Openpilot accurately determines the vehicle's location and orientation in the surrounding environment, ensuring precise navigation and decision making capabilities.
- *Strategizing:* Openpilot strategically plans the vehicle's trajectory, including lane changes and speed adjustments, based on its comprehensive understanding of the road environment.
- *Assisted lane change*: With user-initiated turn signals, Openpilot facilitates lane changes by utilizing its model-based decision-making. Optional confirmation via a steering wheel nudge may be required. Additionally, on certain vehicle models, Openpilot coordinates with blind spot monitors to prevent unsafe lane changes.
- *Software updates:* Openpilot seamlessly receives over-the-air software updates via WiFi or cellular networks, ensuring continuous improvement and functionality enhancements. (OTA updates).

Openpilot Interacting Parts:

Core Components:

- **OpenPilot System:** Integrates various services like camera, modelD, sensorD, controlD, panda, and the car for autonomous driving capabilities.
- **Sensors and Actuators:** Utilizes both vehicle's built-in sensors and additional OpenPilot sensors for environmental awareness.

Services:

- **boardd:** Enables communication with vehicle and peripherals.
- **camerad:** Manages camera inputs for road and driver vision.
- **sensord:** Collects data from physical sensors for motion and environmental conditions.

Neural Network Processing:

- **modeld:** Processes image data for driving path prediction.
- **dmonitoringmodeld and Dmonitoringd:** Assess and ensure driver's attention.

Localization and Calibration:

- **ubloxd/locationd:** Provides vehicle localization.
- **calibrationd/paramsd:** Calibrates neural network inputs and estimates vehicle parameters.

Vehicle Controls:

- **radard/plannerd/controlsd:** Process data for vehicle steering, acceleration, and braking through CAN bus.

System Management:

- **manager/thermald:** Manages processes and monitors system health.
- **loggerd/logcatd/proclogd:** Handles logging.
- **athenad:** Manages device requests.
- **ui:** Displays system status and driving visuals to the user.

Openpilot’s Interaction Flow:

Sensors and cameras gather information on the vehicle's environment and the condition of the driver. This information is then analyzed by neural networks to understand the surroundings and predict the optimal travel path and driver focus. Utilizing the analyzed data, planning services make immediate decisions regarding steering, acceleration, and lane switching. The system persistently evaluates the results of its actions using sensors and cameras, fine-tuning its models and decisions on the fly.

Architectural styles

Layered Architecture

Layered architecture could be used in Openpilot for organizing and managing the complex system. In this architectural style, the architecture is separated into ordered layers. Each layer is tailored to a specific process and management functions. A program in one layer may obtain services from the layer below it. This design is based on increasing levels of abstraction. Referring back to the main components of Openpilot, we could assume that the layers might be some or all of the following:

- Hardware Interaction Layer: A layer of programming that allows a computer OS to interact with a hardware device at a general or abstract level rather than at a detailed hardware level. We know for sure that Openpilot has this layer from their documentation.[2] It is used to enable quality ports for other hardware.
- Neural Network Layer: Usually there are three parts in a neural network: an input layer, with units representing the input fields; one or more hidden layers; and an output layer, with a unit or units representing the target field(s). It could be running neural network models to understand vehicle's surroundings and driver's state.
- Localization and Calibration Layer: Detection or localization is a task that finds an object from image input and localizes the object with a bounding box. This can be used to localize the vehicle. The calibration layer could be used to calculate the confidence intervals and prediction probabilities of a given model.
- Controls Layer: Controls layer is the conductor of operations for a request. It controls the transaction scope and manages the session related information for the request. The controller first dispatches to a command and then calls the appropriate view processing logic to render the response. This layer could make real-time decisions about vehicle control including steering, speed and lane changes.
- System management and UI layer: System management layer provides a centralized solution control and monitoring, displaying the real-time status of every configured Solution object, and activating and deactivating solutions and single applications including user-defined solutions. [3] This layer would ensure that Openpilot runs smoothly.

An interesting way of constructing the software would be to use a concept called "points" for control and monitoring functions, where each point acts as an encapsulated object-oriented design element. These points contain detailed information on operating parameters, tuning parameters, configuration parameters and control strategies. The design allows for modular and flexible configuration of control strategies, enhancing the system's adaptability to process control scenarios. [4]

Implicit Invocation – Publisher Subscriber

Openpilot's architecture employs an event-based implicit invocation system, drawing on concepts from actors, constraint satisfaction, daemons, and packet-switched networks, making it well-suited for its advanced driver-assisted systems (ADAS) due to the high modularity and flexibility it provides. This architecture facilitates real-time data processing and continuous evolution, managing complex interactions among system components efficiently. In Openpilot, components are loosely coupled, communicating through events rather than direct calls, akin to a Pub-Sub architecture where modules function as either publishers or subscribers. For instance, perception, planning, and control modules operate independently, broadcasting control instructions and subscribing to sensor data without needing to understand other modules' inner workings. This event-driven design is crucial for ADAS, ensuring rapid response to sensor data and enabling quick, informed decisions critical for safety and performance in dynamic driving environments. Moreover, the decoupling of components enhances fault tolerance, system reusability, and ease of evolution. New functionalities can be seamlessly integrated by subscribing to existing events or publishing new ones, thereby reducing system complexity and supporting intricate interactions through focused, task-specific component design.

Process-Control

Openpilot could feasibly adopt a process-control architecture to enhance its capabilities as an advanced driver-assistance system (ADAS), leveraging real-time sensor data processing and control loop mechanisms to dynamically adjust vehicle controls based on continuous feedback. This architecture would not only improve Openpilot's real-time decision-making and safety features by allowing for swift, accurate adjustments in steering, braking, and acceleration, but also offer scalability for integrating new technologies. However, implementing such an architecture presents challenges, including managing increased system complexity, ensuring high reliability and fault tolerance, and minimizing latency to meet the rigorous demands of vehicle control. Successfully addressing these challenges would be crucial to fully realizing the benefits of a process-control architecture for enhancing vehicle safety and performance.

Dynamic Evolution of OpenPilot

The evolution of OpenPilot's driver assistance system over time is largely driven by iterative development, community contributions, and data-driven enhancements.

Firstly, Community Contributions. Being open-source, OpenPilot benefits from the contributions of developers and researchers around the world. Improvements can include everything from bug fixes and performance enhancements to the integration of new features and the expansion of vehicle compatibility. Secondly, OpenPilot uses data collected from users (with their consent) to improve its algorithms. This data is crucial for training machine learning models, especially for tasks like lane detection, object detection, and decision making. Thirdly, machine learning and AI. The core of OpenPilot's capabilities comes from deep learning models that are continuously trained on vast amounts of driving data. As more data is collected, these models are retrained to improve their accuracy. Finally, Software Updates. OpenPilot releases regular software updates that users can download and install on their Comma.ai devices. These updates include improvements to existing features and new functionalities.

Concurrency in OpenPilot

Concurrency is essential for handling the real-time nature of autonomous driving tasks, such as perception, decision-making, control, and communication. Here are some key aspects of concurrency present in OpenPilot:

1. **Parallel Sensor Data Processing:** OpenPilot typically receives data from multiple sensors simultaneously, such as cameras, LiDAR, radar, and GNSS receivers. Concurrency is employed to process data from these sensors in parallel, allowing for efficient utilization of system resources and timely responses to changes in the vehicle's environment. For example, while one thread may be processing camera images for object detection and tracking, another thread may be processing LiDAR data for obstacle detection and localization.
2. **Computational Efficiency:** The perception and localization modules of OpenPilot often involve computationally intensive tasks, such as image processing and feature extraction in the visionpc. Concurrency is utilized to distribute these tasks across multiple threads or processes, enabling efficient utilization of multi-core processors and improving overall system performance.
3. **Asynchronous Control and Decision-Making:** OpenPilot's decision-making and control modules continuously analyze sensor data and make real-time decisions to control the vehicle's motion. Concurrency is employed to handle asynchronous events and responses, such as detecting and reacting to sudden changes in the environment or the vehicle's state.
4. **Inter-Process Communication:** OpenPilot consists of multiple interconnected modules or processes, each responsible for different tasks. Concurrency is utilized to enable communication and data exchange between these modules, often through mechanisms such as message passing or shared memory.

Implications for Participating Developers

Overall

The division of responsibilities is crucial for collaboration and the successful development of systems, involving hardware developers who design and integrate sensors, actuators, and components, alongside software developers responsible for implementing algorithms and control logic to ensure compatibility and safety. Sensor specialists focus on calibrating sensors to guarantee accurate data collection, while machine learning experts work on developing models for object detection, path planning, and more. Collaboration and coordination are essential, requiring effective communication to align responsibilities, share expertise, and tackle challenges, ensuring seamless integration and optimal performance across all components.

Layered

Specialization enhances code quality by enabling developers to concentrate on their areas of expertise, while modularity promotes reusability and simplifies updates and maintenance. However, the addition of more layers can lead to increased latency, a factor that is particularly critical in the context of automotive safety. Despite these performance concerns, the layered architecture offers scalability, allowing for the easy addition of resources or functionalities within specific layers without necessitating a complete system overhaul, striking a balance between flexibility and performance efficiency.

Implicit Invocation - PubSub

In event-driven architecture, domain specialization allows developers to concentrate on either producing or consuming events within specific domains, fostering expertise in distinct areas. This specialization enables more independence in development, leading to faster development cycles and the ability to update modules independently. However, effective event management is crucial, necessitating clear definitions and coordination for event contracts to ensure seamless communication between components. Additionally, a coordinated approach is essential for integration and testing to guarantee that the system functions as a cohesive whole when all components interact. To enhance system resilience, modules must implement fallback mechanisms to maintain functionality despite the failure of other components, ensuring the system's robustness and reliability.

Process-Control

Adopting a process-control architecture for Openpilot demands specialized efforts from software developers, including mastering real-time systems programming, emphasizing safety through fail-safe mechanisms and redundancy, and managing feedback loops for dynamic adjustment of vehicle controls. They must focus on optimizing the system for minimal latency, ensuring scalability for future expansions, and rigorous integration and testing to guarantee seamless operation of all components. Developers also need to document processes thoroughly for team alignment and comply with regulatory standards specific to automotive safety. This multi-faceted approach ensures Openpilot's reliability and performance in executing complex driving tasks, highlighting the necessity for a comprehensive understanding of both the technical and regulatory landscapes in autonomous vehicle development.

Diagrams

State Diagram1: Assisted Lane change

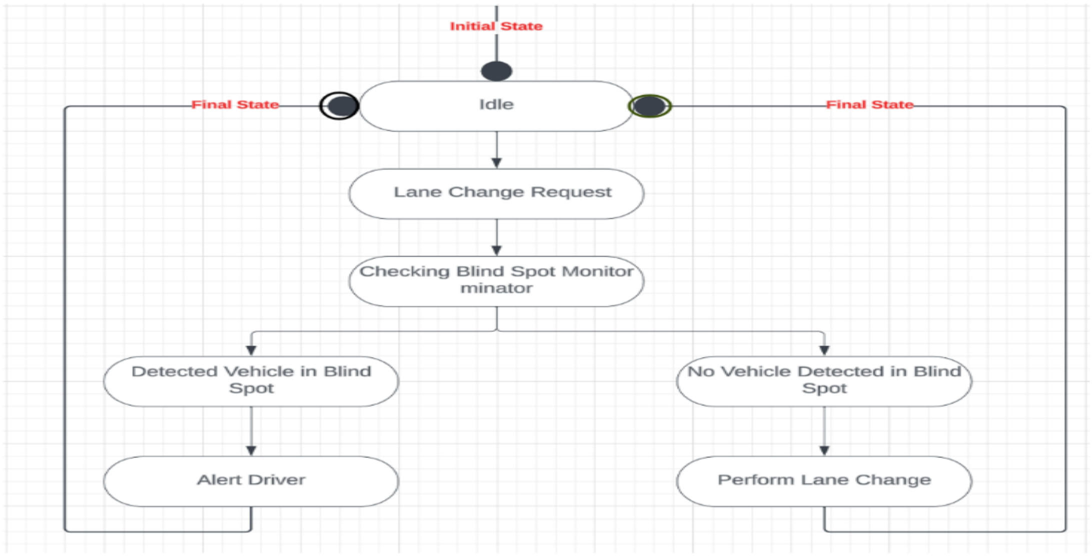
Initial State
Idle
Final State
Final State
Lane Change Request
Checking Blind Spot Monitor minator
Detected Vehicle in Blind Spot
No Vehicle Detected in Blind Spot
Alert Driver
Perform Lane Change

Figure 1: State Diagram 1- Assisted Lane Change

State Diagram2: Navigate on OpenPilot

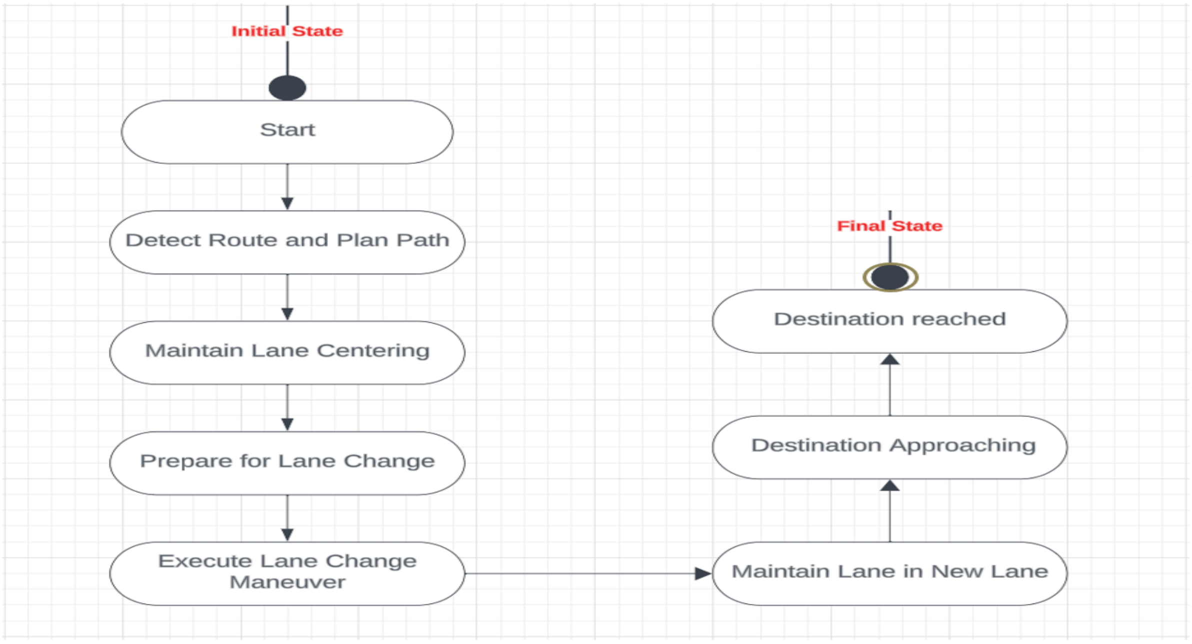

Figure 2: State Diagram 2- Navigate on Openpilot

Use Cases

Show or explain how each Use Case activates or uses the various major parts of the current or proposed architecture.

UC1: Automated Steering

System will automatically adjust the vehicle’s steering to keep it centered within its lane.

Camera: Captures real-time video of the road to detect lane markings.
Perception Module: Analyzes the camera feed using machine learning algorithms to identify lane boundaries.
Control Module: Calculates the necessary steering adjustments to keep the vehicle centered based on the data from the Perception Module.
Steering System: Receives commands from the Control Module and adjusts the steering angle accordingly.

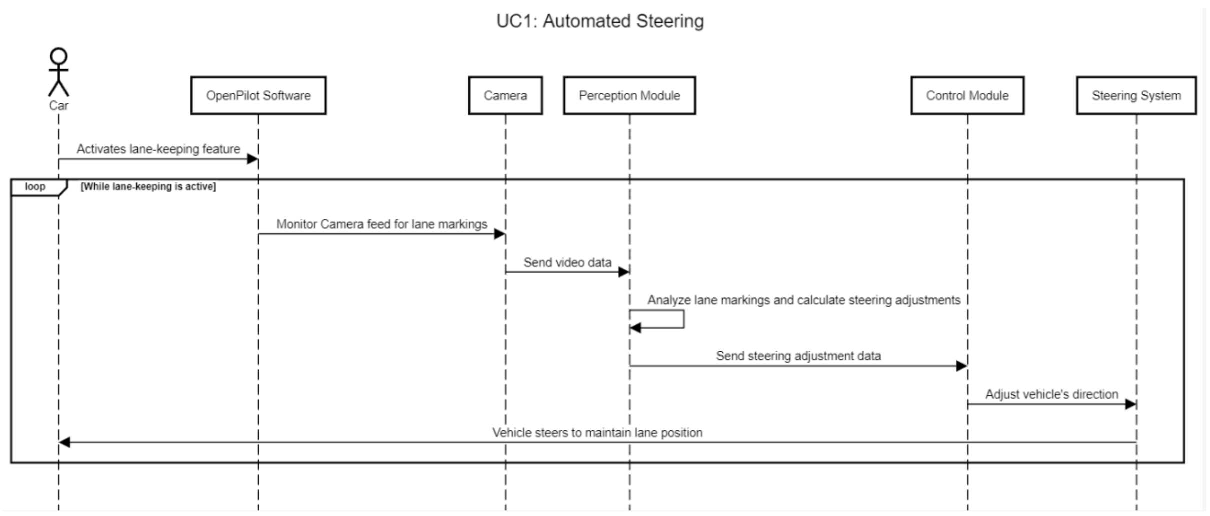
UC1: Automated Steering
Car
OpenPilot Software
Camera
Perception Module
Control Module
Steering System
Activates lane-keeping feature
loop
[While lane-keeping is active]
Monitor Camera feed for lane markings
Send video data
Analyze lane markings and calculate steering adjustments
Send steering adjustment data
Adjust vehicle's direction
Vehicle steers to maintain lane position

Figure 3: UC1- Automated Steering

UC2: Cruise Control

System modulates the vehicle's speed to maintain a safe following distance from the vehicle ahead.

Radar and Camera: Collect data on the distance and relative speed of the vehicle ahead.

Perception Module: Processes this data to assess the current traffic situation and to predict the future position of other vehicles.
Control Module: Determines the required acceleration or deceleration needed to maintain a safe following distance.
Acceleration System and Braking System: Execute the commands from the Control Module to adjust the vehicle's speed.

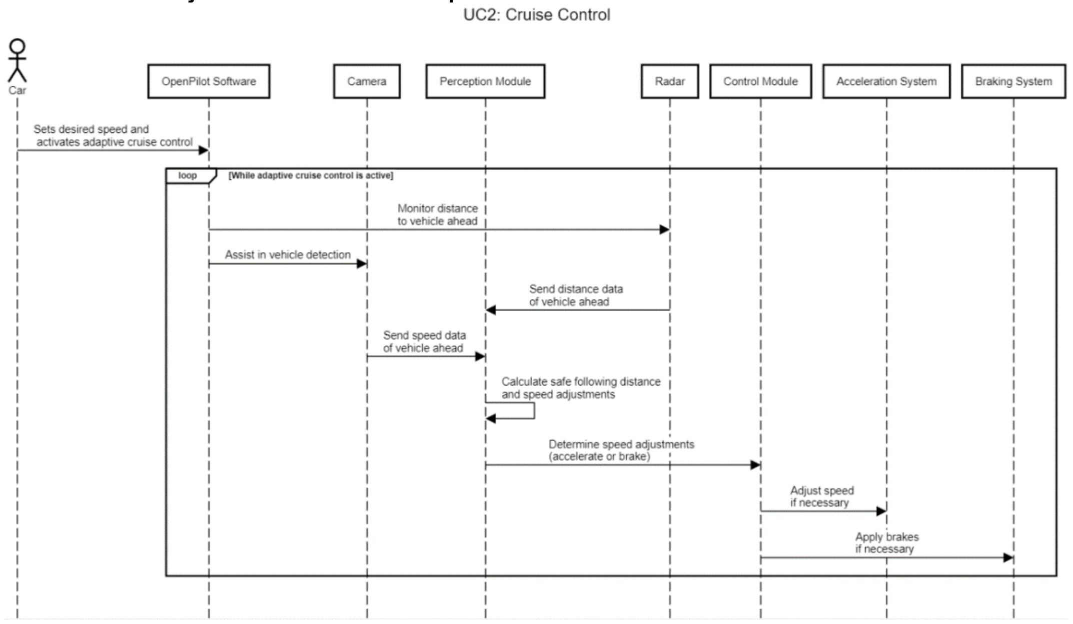
UC2: Cruise Control
Car
OpenPilot Software
Camera
Perception Module
Radar
Control Module
Acceleration System
Braking System
Sets desired speed and activates adaptive cruise control
loop
[While adaptive cruise control is active]
Monitor distance to vehicle ahead
Assist in vehicle detection
Send distance data of vehicle ahead
Send speed data of vehicle ahead
Calculate safe following distance and speed adjustments
Determine speed adjustments (accelerate or brake)
Adjust speed if necessary
Apply brakes if necessary

Figure 4: UC2 – Cruise Control

UC3: Driver Monitoring

Monitors the driver to ensure they remain attentive and ready to take control of the vehicle.

Driver Monitoring Camera: Captures video of the driver to assess their state of attention, including head pose, eye movement, and blink rate.
Driver Monitoring Module: Analyzes the video feed using machine learning algorithms to detect signs of inattention or drowsiness.
Alert System: If the driver appears distracted or drowsy, the system triggers an alert to encourage the driver to refocus on the driving task or to take a break if necessary.

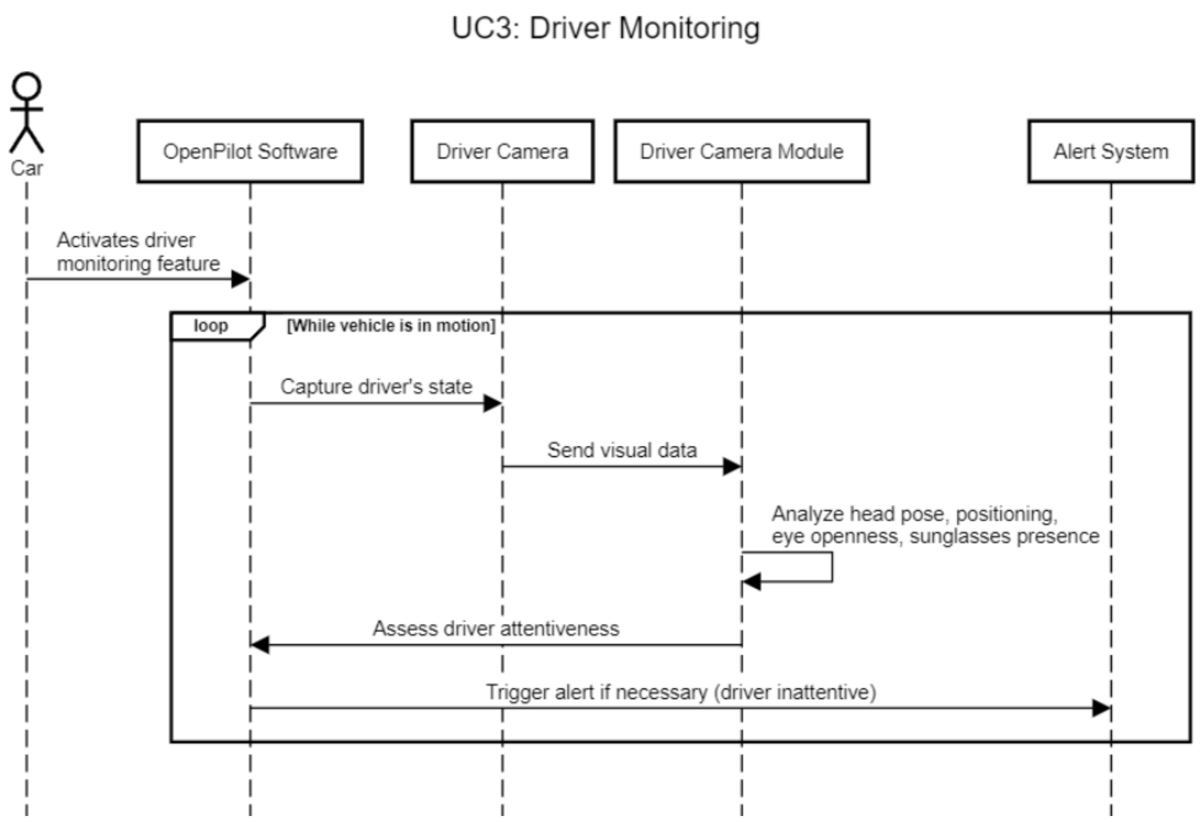
UC3: Driver Monitoring
Car
OpenPilot Software
Driver Camera
Driver Camera Module
Alert System
Activates driver monitoring feature
loop
[While vehicle is in motion]
Capture driver's state
Send visual data
Analyze head pose, positioning, eye openness, sunglasses presence
Assess driver attentiveness
Trigger alert if necessary (driver inattentive)

Figure 5: UC3 – Driver Monitoring

Data Dictionary

Include a glossary that briefly defines all the key terms used in your architecture, giving when appropriate, the "type" of the item being explained.

Key Words	Definition
Daemon	In computing, a daemon (pronounced DEE-muhn) is a program that runs continuously as a background process and wakes up to handle periodic service requests, which often come from remote processes.
Adaptive Cruise Control (ACC)	A feature of OpenPilot that adjusts the vehicle's speed according to traffic conditions to maintain a safe distance from other vehicles
Automated Lane Centering (ALC)	A feature of OpenPilot that autonomously manages steering to keep the vehicle centered within its lane.
Over-the-Air (OTA) Updates	Updates received by OpenPilot via WiFi or cellular networks to enhance functionality and performance continuously.

Concurrency	The ability of OpenPilot to handle multiple tasks simultaneously, such as sensor data processing, decision-making, and communication, essential for real-time autonomous driving tasks.
Deadlock Freedom and Synchronization	Ensuring that OpenPilot avoids deadlock situations where multiple tasks are indefinitely blocked, ensuring system responsiveness and reliability

Conclusion

OpenPilot could be termed a synthesis of state-of-the-art technologies, namely machine learning, sensor fusion, and real-time control systems that together could lead to improved vehicular safety and driving experience. OpenPilot is a technological upgrade to the market for the vehicle systems to automate steering, control speed, and monitor drivers, as well as geo-locate, calibrate, and chart strategic paths in traffic.
What is different about the OpenPilot architecture is that it is a composition of different services or "demons" with all of them interlinked and coordinating among themselves for smooth operation. The architecture of concurrent processing further emphasizes the ability of OpenPilot to respond in real-time to environmental conditions and operations by drivers.

OpenPilot makes use of this uniformity as it uses a generic codebase to build up on an abstraction of hardware differences. Secondly, OpenPilot has been developed with a modular architecture, in order to present opportunities for the development of additional features and abilities that may further its functionality beyond just highway driving to urban and varied driving conditions.

Lessons Learned

Choosing Architectural Models Tailored to Specific Needs

After careful consideration not to use an object-oriented model as OpenPilot's architecture emphasizes how crucial it is to choose architectural models that complement the particular needs and limitations of a project. While object-oriented design offers benefits such as encapsulation, inheritance, and polymorphism, it may not always be the most suitable choice for complex, real-time systems like autonomous driving platforms. In the case of OpenPilot, the layered architecture and event-driven design were preferred due to their ability to manage concurrency, facilitate modular development, and support real-time data processing effectively. By prioritizing architectural models tailored to the system's needs, developers can optimize performance, scalability, and maintainability, ultimately enhancing the success and reliability of the project.

References:

List any documents that your reader may wish to or need to read in conjunction with your report. Since the report is to be web-readable, include links to references when appropriate

https://en.wikipedia.org/wiki/Openpilot

[1] "Openpilot documentation," docs.comma.ai, https://docs.comma.ai/LIMITATIONS.html (accessed Feb. 13, 2024).

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

[3] "Openpilot documentation," docs.comma.ai, https://docs.comma.ai/SAFETY.html (accessed Feb. 13, 2024).

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

[5] "Management layer," Documentation:FR:Dep:MgmtLayer:Current - Genesys Documentation, https://docs.genesys.com/Documentation/FR/latest/Dep/MgmtLayer#:~:text=The%20Management%20 Layer%20provides%3A,that%20records%20applications%20maintenance%20events. (accessed Feb. 13, 2024).

[6] An introduction to software architecture, https://userweb.cs.txstate.edu/~rp31/papers/intro_softarch.pdf (accessed Feb. 13, 2024).