OpenPilot

Discrepancy Analysis

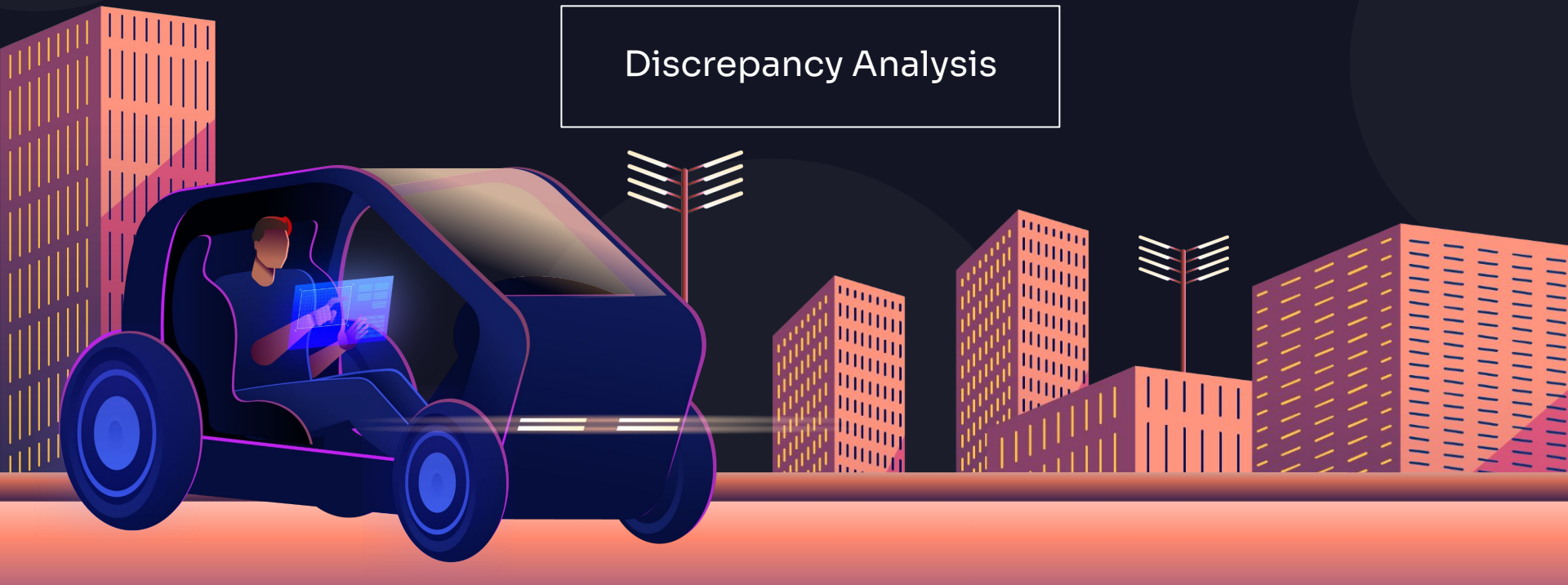
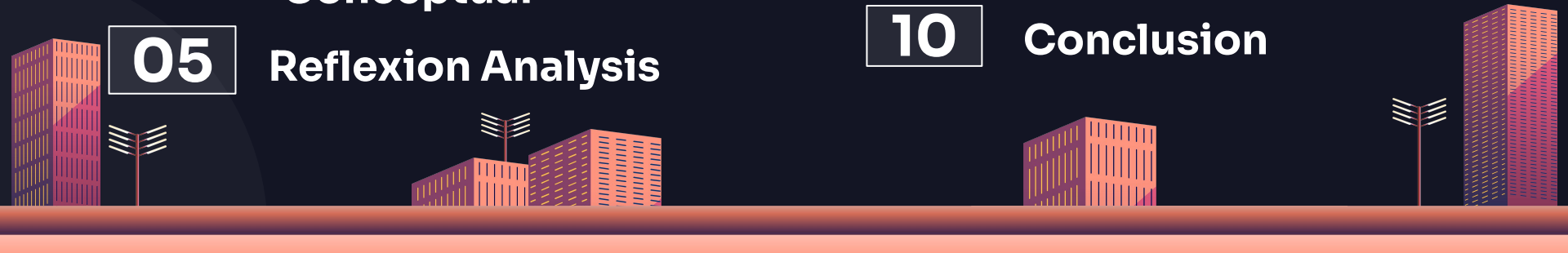


Table of contents



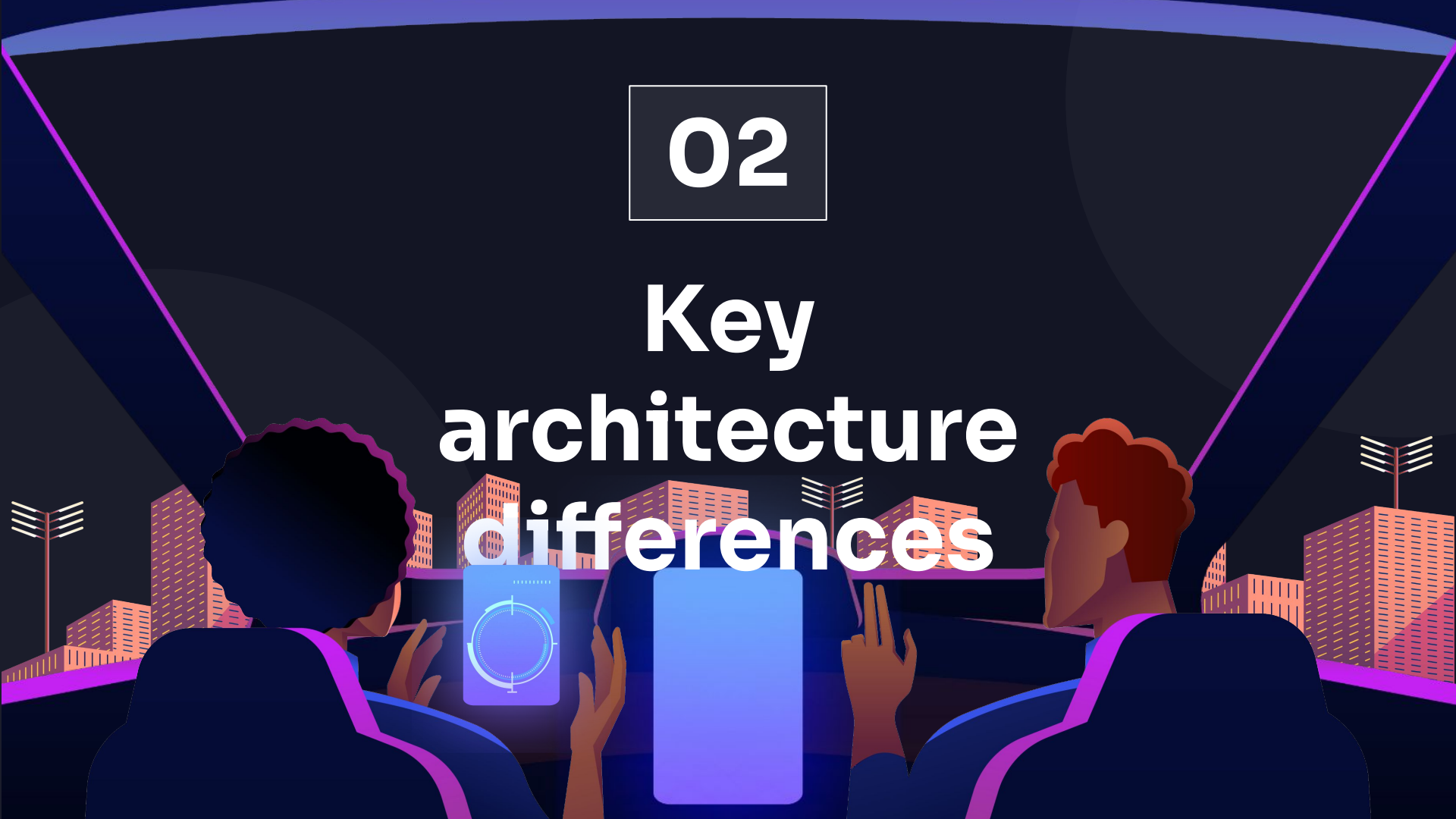
01

Introduction

Introduction

Background: OpenPilot, an advanced open-source driver assistance system, stands at the forefront of autonomous vehicle technology. Its architecture, a testament to modern engineering, facilitates critical functionalities ranging from sensor data processing to dynamic vehicle control.

Objective: This presentation aims to dissect the key architectural differences between OpenPilot conceptual design and its concrete implementation. Through a meticulous discrepancy analysis, we endeavor to shed light on the system's complexity, efficiency, and robust real-world applicability.


02
Key architecture differences

Critical Role of Cereal Module:

- Acts as the messaging specification framework for inter-process communication in OpenPilot.
- Essential for efficient and reliable communication among various services and components.

Structured Communication with Publisher/Subscriber Model:

- Facilitates structured publishing and subscribing to messages within OpenPilot.
- Ensures availability of published messages to subscribed processes, crucial for coordination tasks.

Coordination in OpenPilot:

- Supports coordination among tasks like sensor input processing, driving decisions, and vehicle control.
- Enables seamless communication between different processes for efficient functioning.

Example Use Cases:

- camerad service publishing image data for processing by modeld service.
- controlsd subscribing to messages for translating desired driving path into control signals.

Efficiency and Reliability:

- Messages passed efficiently and in a well-defined manner, enhancing system reliability and performance.
- Ensures smooth operation of OpenPilot's complex architecture.

Expansion in Concrete Architecture:

- Introduction of additional subsystems like Localization and Neural Network systems.
- Enhances the concrete architecture with components not explicitly mentioned in the conceptual architecture.


03
Absences and Divergences

Absences

1. **Messaging Framework Oversight:**

   Doesn't highlight the messaging system (cereal module) critical role in facilitating inter-process communication as revealed in the concrete architecture.

2. **Dynamic Dependency Absent:**

   Runtime Determined Dependencies: Concrete architecture implementation introduces dynamic dependencies not addressed in the conceptual design.

Divergences

1. **Modularity and Interdependencies Contrast:**

   Concrete Depth: Concrete architecture illustrates the depth of interdependencies, particularly emphasizing reliance on the cereal messaging system.

2. **Hardware Integration Differences:**

   Conceptual Oversight: Fails to fully articulate integration between software and major hardware components.


04
Concrete Vs
Conceptual

Concrete Vs Conceptual

1. Inter-Process Communication

Heavy reliance on cereal messaging system for efficient, reliable communication among services.

Analysis Approach:

We Investigated the flow of messages between services in the concrete architecture. We compared it the flow we see from the conceptual architecture at https://blog.comma.ai

Concrete Vs Conceptual

2. Hardware Integration

Detailed integration between software services and hardware components addresses practical needs of interfacing with vehicle systems and sensors.

Analysis Approach:

We analyzed the roles of services that communicate with hardware components in both the conceptual and concrete architecture.

Concrete Vs Conceptual

3. Dynamic Dependencies and Runtime Behavior

Conceptual architecture lacked detailed analysis as opposed to the concrete architecture which reflects a design that accommodates runtime variability and complexity.

Analysis Approach:

We examined the dependencies when components communicated and noticed dependencies would change during runtime.


05
Reflexion
Analysis

Investigating Dependencies using 4 “W”s

1. Camera → Sensor

Which?	Camera (Openpilot/System/camerd/cameras/camera_qcom2.cc) depends on sensors(Openpilot/System/camerad/sensors/ar0231_registers.h)
Who?	Third-party developer(GitHub user: deanlee)
When?	Dec 7, 2023
Why?	This dependency was introduced to help improve code organization and clarity within the camerad module. The content of sensor2_i2c.h was moved to ar0231_registers.h reflecting the specific purpose of the file, which is to define register settings for the ar0231 sensor. Moving the file to the system/camerad/sensors directory aligns with a more logical and structured organization of files within the module. It separates sensor-specific configurations from other camera-related functionalities, making the codebase more understandable and maintainable.

Investigating Dependencies using 4 “W”s

2. Camera → python VisionBuf

Which?	Camera (Openpilot/System/camerd/snapshot/snapshot.py) depends on Python VisionBuf class(Openpilot/cereal/visionipc)
Who?	Third-party developer(GitHub user: mitchellgoffpc)
When?	Jul 25, 2023
Why?	This dependency was introduced to help improve code readability and maintainability, and to align with best practices by utilizing attributes of the buf object instead of passing individual parameters. This change reduces the number of arguments the function requires and enhances its flexibility.

Investigating Dependencies using 4 “W”s

3. Car → carcontroller and carstate

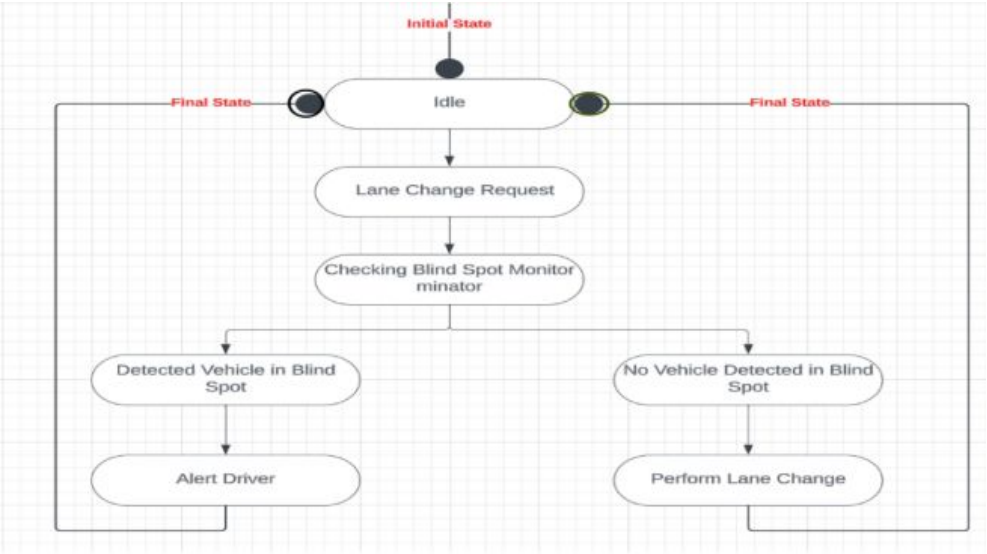
Which?	Car (Openpilot/selfdrive/car) depends on carcontroller and carstate (Openpilot/selfdrive/car/mock)
Who?	Third-party developer(GitHub user: adeebshihadeh)
When?	March 17th , 2024
Why?	This dependency was introduced to simplify and optimize the loading process. It aims to enhance code maintainability, optimize initialization processes, and facilitate testing by introducing mock implementations of car state and controller interfaces.

06

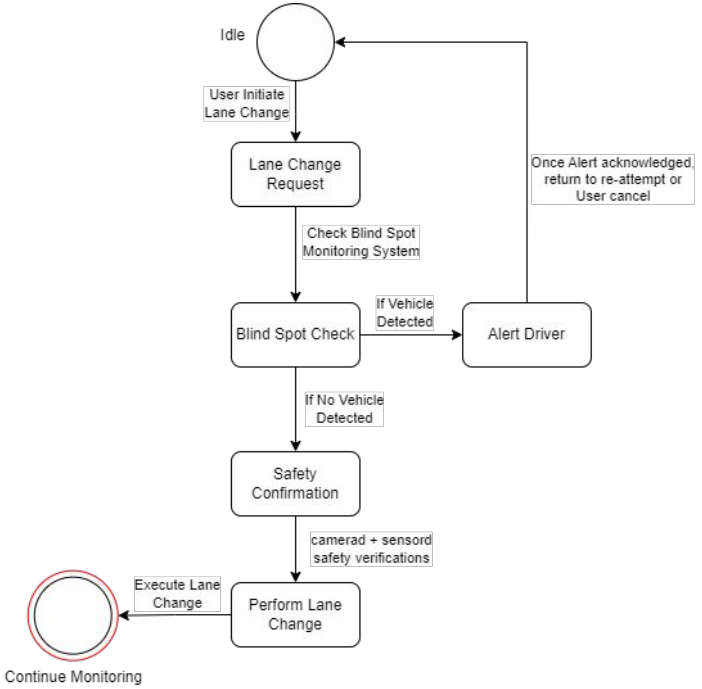
Diagrams

State Diagram: Assisted Lane Change

Conceptual

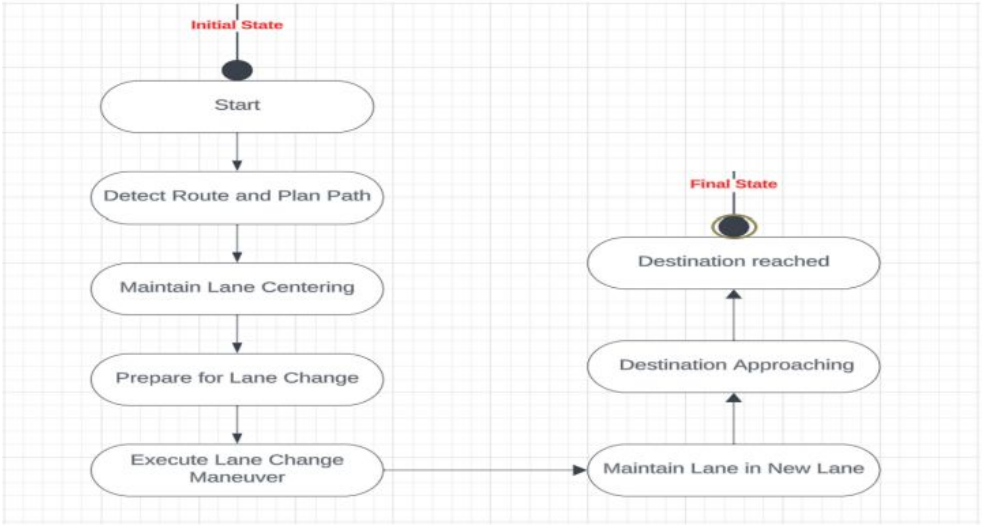


Updated

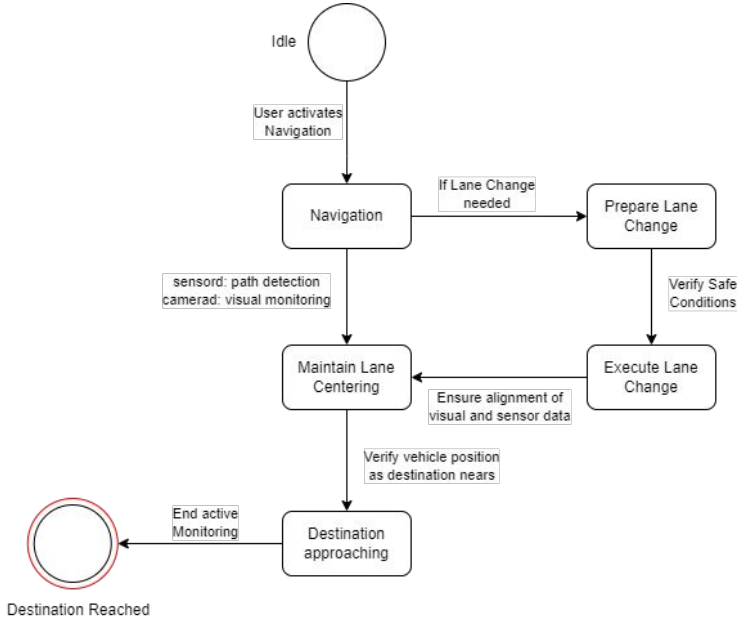


State Diagram: Navigation

Conceptual

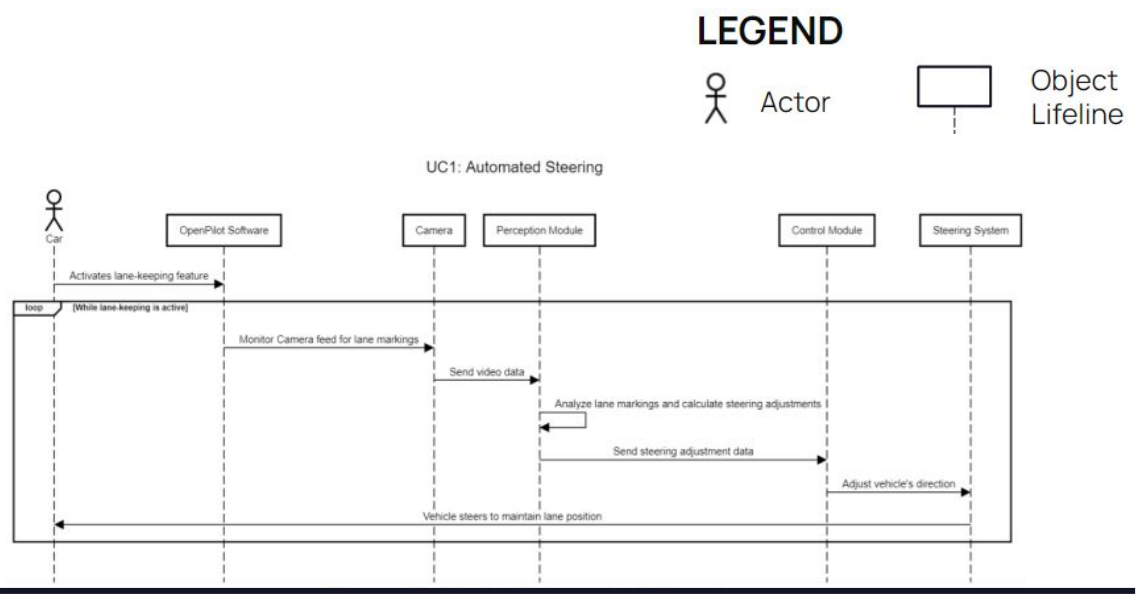


Updated

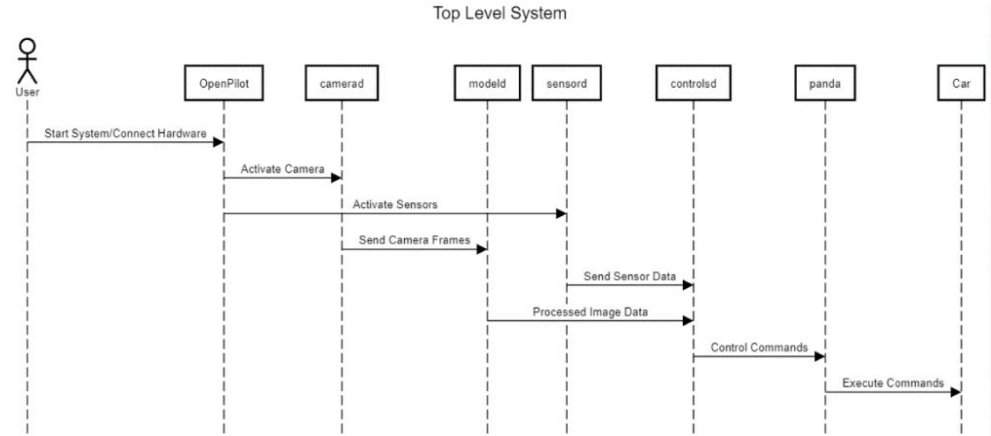


Sequence Diagram

Conceptual



Updated

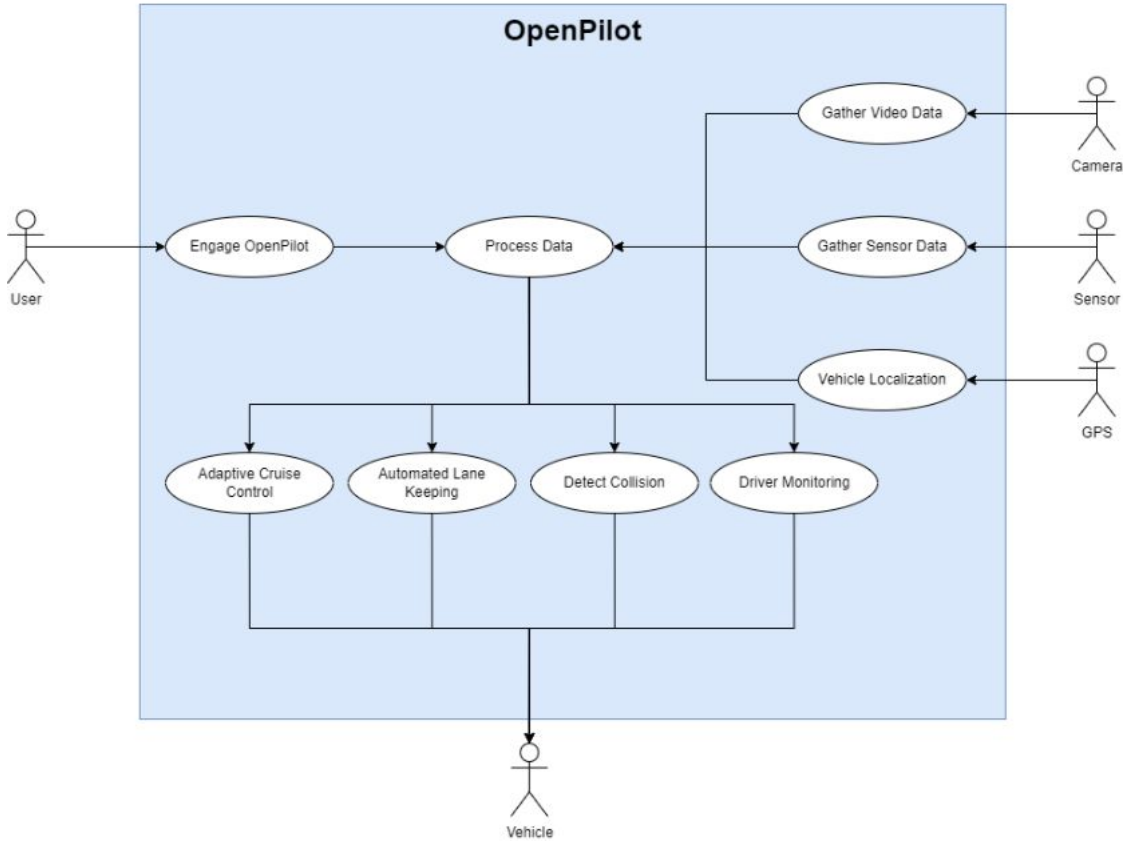


Use Case Diagram

LEGEND

Actor

Use Case



07

Concurrency

Concurrency in Camerad & Sensord

Camerad :

1. Processing data from cameras on the vehicle.
2. Handle multiple streams of camera data simultaneously.
3. Processes data from multiple cameras in real-time which is essential for tasks such as object detection, lane detection etc.

Sensord :

1. Managing data from various sensors
2. Handling data from these sensors concurrently.
3. Gathering and processing information from multiple sensors, enhancing the system's perception capabilities and providing more accurate situational awareness.

Effects of Concurrency

- Improved Performance
- Enhanced Robustness
- Increased Scalability
- Real-time Responsiveness

08

Limitations

Limitations of Discrepancy Analysis

Inherent Complexity: OpenPilot's architecture has very complicated directory structures → difficult to parse/read

Documentation Gaps: Pull requests not always documented correctly → i.e., introduced a dependency but did not mention, confusing pull request conversation between contributors and reviewers

Rapid Evolution: Concrete architecture can quickly diverge from documented conceptual models, making ongoing discrepancy analysis necessary.


09
Lessons
Learned

Lessons Learned

Runtime concrete architecture diverged a lot from the conceptual

Divergences were much more common than absences

Reading the conversations in commit logs and PRs as well as tracing the associated codes helps us understand design decisions

Good documentation is crucial for easier onboarding and collaboration.

10

Conclusion

Conclusion

The discrepancy analysis of OpenPilot's architecture not only reveals the intricacies of transitioning from conceptual to concrete implementations but also illuminates the path forward for enhancing autonomous driving systems.

By embracing lessons learned and anticipating future challenges, we can continue to push the boundaries of what's possible in autonomous vehicle technology.