

## Inhaltsverzeichnis

1. Einführung und Ziele .....	4
1.1. Aufgabenstellung .....	4
1.2. Qualitätsziele .....	5
1.3. Stakeholder .....	5
2. Randbedingungen .....	6
2.1. Technische Randbedingungen .....	6
2.2. Organisatorische Randbedingungen .....	7
2.3. Konventionen .....	8
3. Kontextabgrenzung .....	9
3.1. Fachlicher Kontext .....	9
3.2. Technischer Kontext .....	10
4. Lösungsstrategie .....	11
4.1. Einstieg .....	11
4.2. Modularer Aufbau .....	11
5. Bausteinsicht .....	14
5.1. Ebene 1 .....	14
5.2. Ebene 2: Lokalisierung und Standortverarbeitung .....	15
6. Laufzeitsicht .....	21
6.1. Annäherung an ein Stoppschild .....	21
7. Verteilungssicht .....	22
7.1. Infrastruktur .....	22
8. Querschnittliche Konzepte .....	23
8.1. Common .....	23
8.2. Guardian .....	23
8.3. Drivers .....	24
8.4. Protocol Buffers .....	24
8.5. Funktionale Abgrenzung .....	24
9. Entwurfsentscheidungen .....	26
9.1. ROS vs CyberRT .....	26
10. Qualitätsanforderungen .....	27
10.1. Qualitätsbaum .....	27
10.2. Qualitätsszenarien .....	27
11. Risiken und technische Schulden .....	29
11.1. Aufwand der Implementierung .....	29

11.2. Komplexes Gesamtsystem .....	29
11.3. Ausstehende Implementierung vieler Verkehrssituationen .....	29
12. Glossar .....	30
12.1. Einstieg .....	30
12.2. Begriffe .....	30
13. Anhang .....	31
13.1. Erfahrungen mit Apollo .....	31
13.2. Erfahrungen mit Arc42 .....	31
13.3. Erfahrungen mit docToolchain .....	31
13.4. Sonstige Anmerkungen .....	31

## Über arc42

arc42, das Template zur Dokumentation von Software- und Systemarchitekturen.

Erstellt von Dr. Gernot Starke, Dr. Peter Hruschka und Mitwirkenden.

Template Revision: 7.0 DE (asciidoc-based), January 2017

© We acknowledge that this document uses material from the arc42 architecture template, <http://www.arc42.de>. Created by Dr. Peter Hruschka & Dr. Gernot Starke.

---

# 1. Einführung und Ziele

Dieser Abschnitt beschreibt die wesentlichen Anforderungen und treibenden Kräfte, die bei der Umsetzung der Softwarearchitektur und Entwicklung des Systems berücksichtigt werden müssen.

Dazu gehören:

- zugrunde liegende Geschäftsziele,
- wesentliche Aufgabenstellungen und
- essenzielle fachliche Anforderungen an das System sowie
- Qualitätsziele für die Architektur und
- relevante Stakeholder und deren Erwartungshaltung.

## 1.1. Aufgabenstellung

*Was ist ApolloAuto?*

Apollo ist eine leistungsstarke und flexible Architektur für autonome Fahrzeuge, die eine schnelle Entwicklung und Tests von diesen ermöglicht. Dessen API ist dabei gänzlich öffentlich einsehbar. Die Ziele der einzelnen Apollo Releases seit dem Start im April können der folgenden Grafik entnommen werden.



Mit diesem Ansatz soll die Grundlage für Implementierungen des autonomen Fahrens geschaffen werden. Langfristig wird Level 5 autonomes Fahren verfolgt.

Die vorliegende Architektur soll verschiedene Anwendungen ermöglichen. Einige mit entscheidenden Kernfunktionen werden in folgender Tabelle aufgelistet.

Use Case	Erläuterung
Apollo Go Robotaxi	Ein autonom fahrender Taxiservice, welcher Level 4 autonomes Fahren beherrschen soll
Apollo V2X	Ein intelligentes System für die Kommunikation zwischen Autos und allen anderen Elementen im Straßenverkehr
Valet Parking	Das Fahrzeug übernimmt die Parkplatzsuche und den eigentlichen Einparkvorgang, beispielsweise in Parkhäusern

## 1.2. Qualitätsziele

Die folgende Tabelle zeigt zentrale Qualitätsziele von ApolloAuto auf.

Qualitätsziel	Erläuterung
<i>Vereinfachte Entwicklung autonomer Fahrzeuge (Änderbarkeit)</i>	<i>Entwickler haben die Möglichkeit, funktionsfähige technische Anwendungen mit verhältnismäßig geringem Aufwand zu implementieren.</i>
<i>Schnelle Tests von autonomen Fahrzeugen (Analysierbarkeit)</i>	<i>Neue Implementierungen müssen leicht testbar sein.</i>
<i>Flexible Architektur (Anpassbarkeit)</i>	<i>Durch die schnellen technischen Fortschritte im autonomen Fahren muss die bisherige Architektur leicht angepasst werden können.</i>
<i>Verständliche Darstellung (Nutzerfreundlichkeit)</i>	<i>Entwickler, die nicht bei Baidu arbeiten, müssen mit dem vorliegenden Code und den gegebenen Beschreibungen des Systems zurechtkommen.</i>
<i>Hohe Fehlertoleranz</i>	<i>Im Straßenverkehr sind Systemausfälle nicht zulässig</i>

## 1.3. Stakeholder

In dieser Tabelle werden Stakeholder und ihre Erwartungen an das Projekt dargestellt.

Rolle	Erwartungshaltung
<i>Baidu Product Owner</i>	<i>Weiterentwicklung des autonomen Fahrens und von V2X-Anwendungen</i>
<i>Baidu Entwicklungsteam</i>	<i>Aufbau einer Struktur, welche für eine Weiterentwicklung bei zukünftigen Aufgaben geeignet ist</i>
<i>Unabhängige Entwickler</i>	<i>Ziel ist die Entwicklung von Tools, um Produkte für das autonome Fahren zu entwickeln. Die Einarbeitung in das System muss dafür ermöglicht werden.</i>

## 2. Randbedingungen

Vor dem Lösungsentwurf waren verschiedene Randbedingungen zu beachten, welche die Architektur beeinflussen. Diese werden im folgenden Kapitel anschaulich dargestellt.

### 2.1. Technische Randbedingungen

Die technischen Randbedingungen beziehen sich auf die aktuelle Version Apollo 6.0. Es ist sinnvoll, zu unterscheiden, welche Anforderungen an die CPU für die Installation und welche an das Auto gestellt werden.

Tabelle 1. Installationsanforderungen

Randbedingung	Erläuterung
Prozessor und RAM	8-Kern Prozessor mit mindestens 16GB Arbeitsspeicher
GPU	Eine NVIDIA Turing GPU wird empfohlen
Betriebssystem	Ubuntu 18.04
GPU-Treiber	NVIDIA 455.32.00 Treiber sowie alle nachfolgenden Versionen ( <a href="#">Nvidia Treiber</a> )
Docker	Docker-CE 19.03 sowie alle nachfolgenden Versionen ( <a href="#">Installation von Docker auf Ubuntu</a> )
NVIDIA Container Toolkit	NVIDIA 455.32.00 Treiber sowie alle nachfolgenden Versionen ( <a href="#">NVIDIA-Docker Github Link</a> )

Tabelle 2. Anforderungen an das korrespondierende Fahrzeug

Randbedingung	Erläuterung
Drive-by-Wire	Fahren und Steuern der Fahrzeuge muss ohne mechanische Kraftübertragung möglich sein. Dazu gehören Lenkung, Bremsen, Gas und Schaltvorgänge.
Sensorik	Mehrere LiDAR- und Radar-Sensoren zur besseren Auffassung der Umgebung
Kameras	Seitliche und Frontkameras zur Umgebungsaufnahme
GPS	GPS-Antenne und Receiver zur Lokalisierung des Fahrzeugs
IMU	Inertiale Messeinheit zur Aufnahme verschiedener Sensordaten zum Fahrzeug

Der Aufbau eines solchen Fahrzeugs kann dem nachfolgenden Bild entnommen werden.

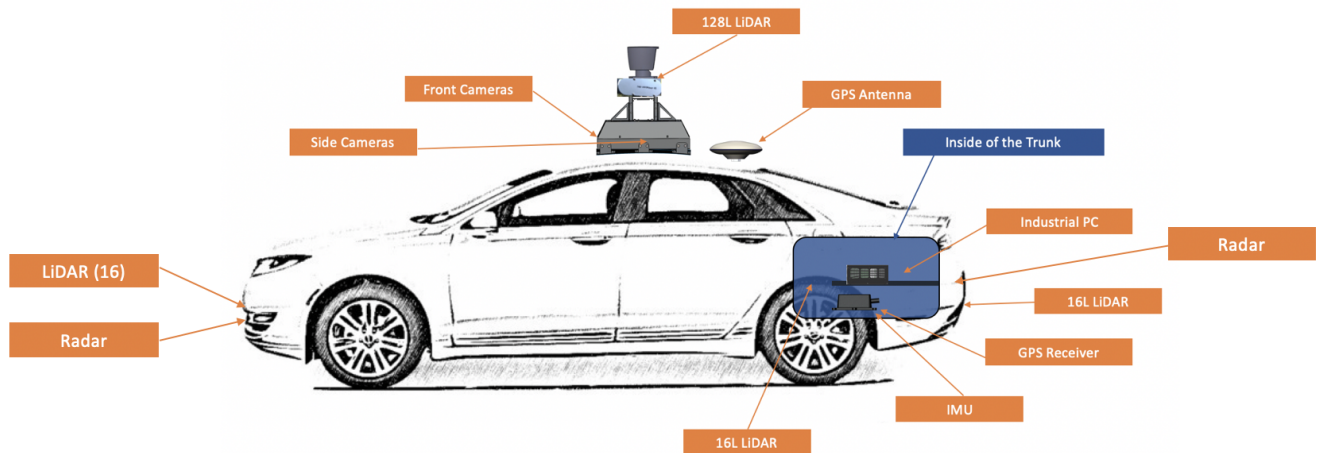


Abbildung 1. Fahrzeuganforderungen ApolloAuto

Die Spezifikationen anderer Versionen können der [Github Seite](#) entnommen werden.

## 2.2. Organisatorische Randbedingungen

Randbedingung	Erläuterung
Team	Entwicklerteam von Baidu, ggfs. stark auf den chinesischen Markt spezialisiert
Zeitplan	Updates mit Abständen von Monaten bis zu einem Jahr
Vorgehensmodell	Mit den Updates verbunden sind Erweiterungen der Funktionalität der Software und Integration von zusätzlicher Hardware
Konfigurations- und Versionsverwaltung	Der Repository von ApolloAuto ist frei zugänglich auf GitHub einsehbar.
Veröffentlichung als Open Source	<b>Lizenzbedingungen hier eintragen</b>

## 2.3. Konventionen

Konvention	Erläuterung
Dokumentation	Aktuelle Dokumentation erfolgt über Readme-Dateien innerhalb des Github-Repositories.
Sprache	Da sowohl eine internationale Veröffentlichung vorgesehen ist, als auch der heimische Markt bedient werden soll, werden Dokumente in Englisch und Chinesisch aufgeführt.
Datenübertragung	Zur Übertragung von Informationen werden Protobuf-Dateien oder der installierte CAN-Bus verwendet.
Kodierung	Es wird hauptsächlich C++ verwendet (siehe untenstehende Abbildung). Zu Python existiert eine designierte Schnittstelle.

### Languages

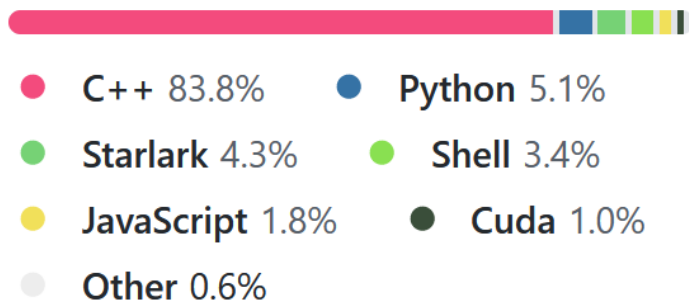


Abbildung 2. Verwendete Programmiersprachen



# 3. Kontextabgrenzung

Dieser Abschnitt beschreibt die Schnittstellen von ApolloAuto. Welche Personen und Systeme interagieren miteinander?

## 3.1. Fachlicher Kontext

Interaktion der Software mit den anderen Verkehrsteilnehmern (Sensordaten), den Kartendaten (Fremdsystem) und der Fahrzeugaktorik (Sensordaten).

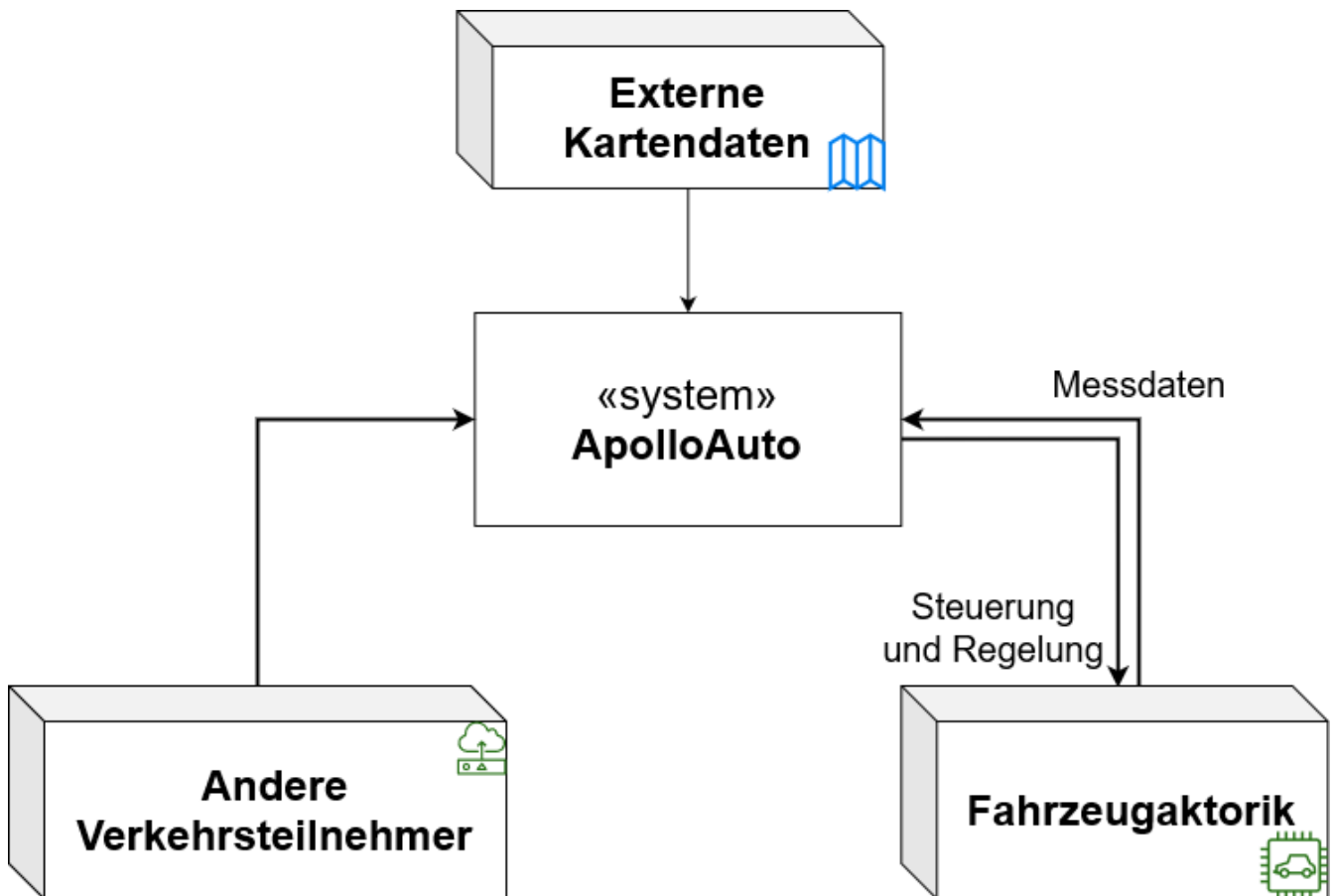


Abbildung 3. Fachlicher Kontext ApolloAuto

### Externe Kartendaten

ApolloAuto benötigt Zugang zu einer Bibliothek mit Kartendaten. Nur so kann sichergestellt werden, dass das Fahrzeug die Verkehrssituation auf Basis seiner Position sinnvoll bewertet.

### Fahrzeugsensorik

Die Sensorik, welche das Umfeld des Fahrzeugs aufnimmt, wird von ApolloAuto verwertet. Dazu gehören beispielsweise das GPS und LiDAR-Sensoren.

### Fahrzeugaktorik

Dem System ist es möglich, Vorgänge wie die Lenkung, Beschleunigung und Bremsvorgänge zu steuern. Gleichzeitig erhält es vom Fahrzeug Zustandsinformationen.

## 3.2. Technischer Kontext

Technische Interaktion des Systems mit externen Beteiligten.

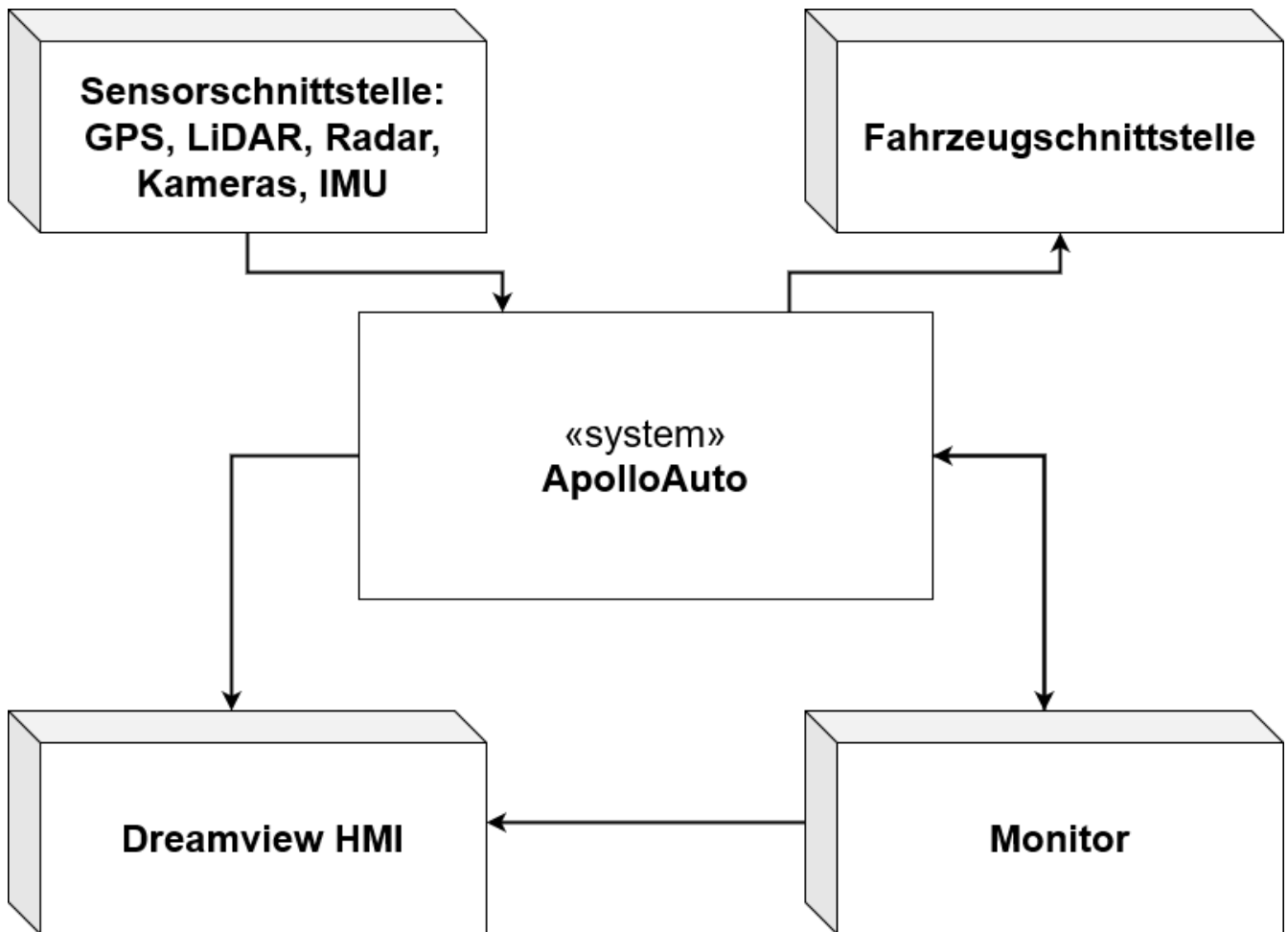


Abbildung 4. Technischer Kontext ApolloAuto

### Sensorschnittstelle

Das System verwertet die empfangenen Daten der Fahrzeugsensorik, wie GPS, LiDAR, Radar, Kamerasysteme und eine inertielle Messeinheit. Damit empfängt es sowohl die Informationen zum Fahrzeug, als auch zur umgebenden Verkehrssituation.

### Monitor

Der Monitor ermöglicht die externe Überwachung von Fahrzeugzuständen. Bei bestimmten, festgelegten Zuständen wird in das System eingegriffen. Außerdem ermöglicht es diese Zustände an die Dreamview HMI weiterleiten.

### Dreamview HMI

Im Dreamview Human Machine Interface können Ausgabewerte des Fahrzeugs visualisiert werden. Dieses bildet somit die Schnittstelle zum Menschen, über welche eine Überwachung des Fahrzeugzustands möglich ist.

### Fahrzeugschnittstelle

Steuerungs- und Regelungsbefehle müssen an die Fahrzeugaktuatorik weitergeleitet werden.

## 4. Lösungsstrategie


In diesem Kapitel wird ein Überblick über die Architektur geboten. Dabei stehen die Ziele und Lösungsansätze im Fokus.

### 4.1. Einstieg

Qualitätsziel	Dem zuträgliche Ansätze in der Architektur
<i>Vereinfachte Entwicklung autonomer Fahrzeuge</i>	<ul style="list-style-type: none"><li>• <i>Cyber RT Framework, Laufzeitumgebung die extra für das Autonome Fahren entwickelt wurde</i></li><li>• <i>Modularer Aufbau der Softwarepakete</i></li><li>• <i>Python API ermöglicht die Programmierung durch Entwickler, die mit C++ nicht vertraut sind</i></li></ul>
<i>Schnelle Tests von autonomen Fahrzeugen</i>	<ul style="list-style-type: none"><li>• <i>Dreamview Human Machine Interface zur Überwachung des Fahrzeugzustands</i></li></ul>
<i>Flexible Architektur</i>	<ul style="list-style-type: none"><li>• <i>Continuous Integration Ansatz, bei welchem in Updatezyklen die bestehenden Funktionen erweitert werden</i></li><li>• <i>Ablage des Repositories auf Github</i></li><li>• <i>Modularer Ansatz garantiert Erweiterbarkeit</i></li></ul>
<i>Verständliche Darstellung</i>	<ul style="list-style-type: none"><li>• <i>Implementationsbeschreibung über Readme-Dateien in der Repository</i></li><li>• <i>Kurzbeschreibungen der Ordnerinhalte</i></li><li>• <i>Grafische Darstellungen von einigen übergeordneten Ansätzen, wie bspw. der Hardware- und Softwareübersicht</i></li></ul>
<i>Hohe Fehlertoleranz</i>	<ul style="list-style-type: none"><li>• <i>Guardian, welcher bei kritischen Input-Werten von Steuergerät oder Monitor direkt auf den CANBus zugreifen kann</i></li><li>• <i>Verwendung vieler verschiedener Sensoren</i></li></ul>

### 4.2. Modularer Aufbau


Das Gesamtsystem ist grundsätzlich modular aufgebaut. Im Unterordner /modules können die einzelnen Bestandteile eingesehen werden.

 r6.0.0
 apollo / modules /
 

Go to file
 Add file
 ...

This branch is 49 commits ahead, 553 commits behind master.
 

Contribute


 SeasoulChris seperate vehicle config files (#13707)
 564e485 on 19 Apr
 History

..		
audio	Format: use scripts/clang_format.sh to format all proto directories	9 months ago
bridge	Format: apollo.sh format modules/bridge	9 months ago
calibration/data	seperate vehicle config files (#13707)	2 months ago
canbus	canbus: update the devkit battery soc and ch battery info (#13120)	7 months ago
common	monitor: add pointcloud_16_topic to monitor module (#13184)	6 months ago
contrib	Docs&Scripts: updated scripts/bridge.sh and updated README.md for cyb...	10 months ago
control	Control: fix some bugs in C matrix setting of MPC controller	9 months ago
data	cyber_recorder: add support for black listing topic names in record l...	8 months ago
dreamview	monitor: add pointcloud_16_topic to monitor module (#13184)	6 months ago
drivers	Drivers: rearrange driver files and configurations	7 months ago
guardian	MockTime support: realtime for guardian; clock mode for dreamview	10 months ago
localization	D-Kit : Add localization timer (#12796) (#12834)	8 months ago
map	Map: Update Borregas map for LGSVL 2020.06	5 months ago
monitor	monitor: add pointcloud_16_topic to monitor module (#13184)	6 months ago
perception	Perception: fix perception to v2x fusion error (#12632)	9 months ago
planning	Planning: only relax path bounds for lane boundaries	9 months ago
prediction	Format: use scripts/clang_format.sh to format all proto directories	9 months ago
routing	routing: (1) fixed an issue where end_s is shorter than start_s in so...	10 months ago
storytelling	Format: use scripts/clang_format.sh to format all proto directories	9 months ago
third_party_perception	Format: use scripts/clang_format.sh to format all proto directories	9 months ago
tools	add default main_sensor in data extract tool (#12941) (#12942)	7 months ago
transform	Drivers: rearrange driver files and configurations	7 months ago
v2x	Perception: fix perception to v2x fusion error (#12632)	9 months ago

Abbildung 5. Module in der ApolloAuto Repository

In diesen Ordnern sind teilweise weitere Informationen zu den Modulen hinterlegt. Im Kontext dieser Dokumentation werden für das Grundverständnis der Funktionsweise von ApolloAuto folgende Module genauer betrachtet.

- Perception
- Prediction
- Planning
- Control
- Map
- Localization
- CANBus
- Guardian
- Monitor
- Dreamview HMI

- Common

Der Monitor und die Dreamview HMI wurden bereits in Kapitel 3 erläutert, da diese im Zusammenspiel eine Interaktion von außen ermöglichen. In Kapitel 5 werden auch die übrigen Module in die Funktionalität des Gesamtsystems eingeordnet. Grundsätzlich lassen sich die Funktionalitäten dabei in folgende Gruppen unterteilen:

- Regelungs- und Steuerungseinheit
  - Perception
  - Prediction
  - Planning
  - Control
- Lokalisierung und Standortverarbeitung
  - Map
  - Localization
- Überwachungs- und Kontrolleinheit
  - In die interne Steuerung integriert
    - CANBus
    - Guardian
  - Funktion als externe Schnittstelle
    - Monitor
    - Dreamview HMI
- Modulübergreifender Code
  - Common

In Common beinhaltet dabei alles, was für mehrere Funktionen von Relevanz ist. In Kapitel 8 wird explizit auf dieses Modul als Konzept eingegangen. Bei der Überwachungs- und Kontrolleinheit kann außerdem zwischen dem CANBus und Guardian sowie dem Monitor und der Dreamview HMI unterschieden werden, da diese direkten Zugriff auf die Regelung und Steuerung des Fahrzeugs haben.

# 5. Bausteinsicht

Diese Sicht zeigt die statische Zerlegung des Systems in Bausteine sowie deren Beziehungen. Bausteine der ersten Zerlegungsebene werden in den nachfolgenden Kapiteln als Subsysteme bezeichnet.

## 5.1. Ebene 1

In Kapitel 4 wurden bereits verschiedene Module thematisch unterteilt. Diese Gruppierungen bieten sich auch für die Beschreibung und Interaktion zwischen einzelner Subsysteme an.

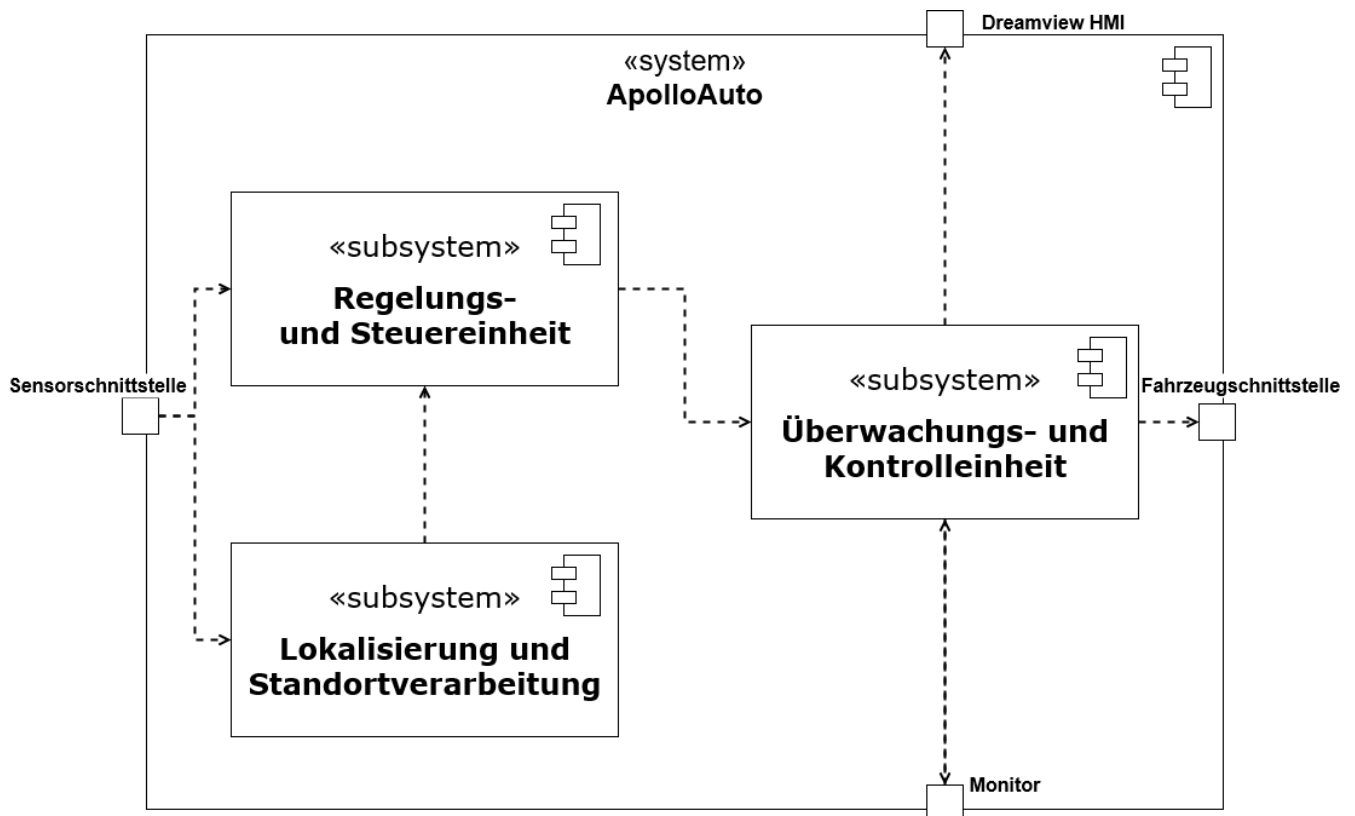


Abbildung 6. Bausteinsicht Ebene 1

Subsystem	Kurzbeschreibung
Regelungs- und Steuereinheit	Übernimmt die Berechnungsaufgaben des autonomen Fahrens, welche letztendlich das Fahrzeugverhalten vorgeben. Das Subsystem empfängt Sensordaten, um bspw. Kollisionen zu verhindern.
Überwachungs- und Kontrolleinheit	Verantwortlich für die Übermittlung und Überwachung der Datensätze, die von der Regelungs- und Steuereinheit empfangen werden.
Lokalisierung und Standortverarbeitung	Empfängt Sensordaten, auf deren Basis die Lokalisierung und Standortverarbeitung verwaltet wird.

Da die genaue Interaktion zwischen den einzelnen Subsystemen gänzlich durch die Module definiert wird, ist die Betrachtung der Schnittstellen innerhalb des Systems auch erst ab der zweiten Ebene sinnvoll, folglich erfolgt diese dort.

## 5.2. Ebene 2: Lokalisierung und Standortverarbeitung

Im folgenden Kapitel wird die Lokalisierung und Standortverarbeitung genauer betrachtet.

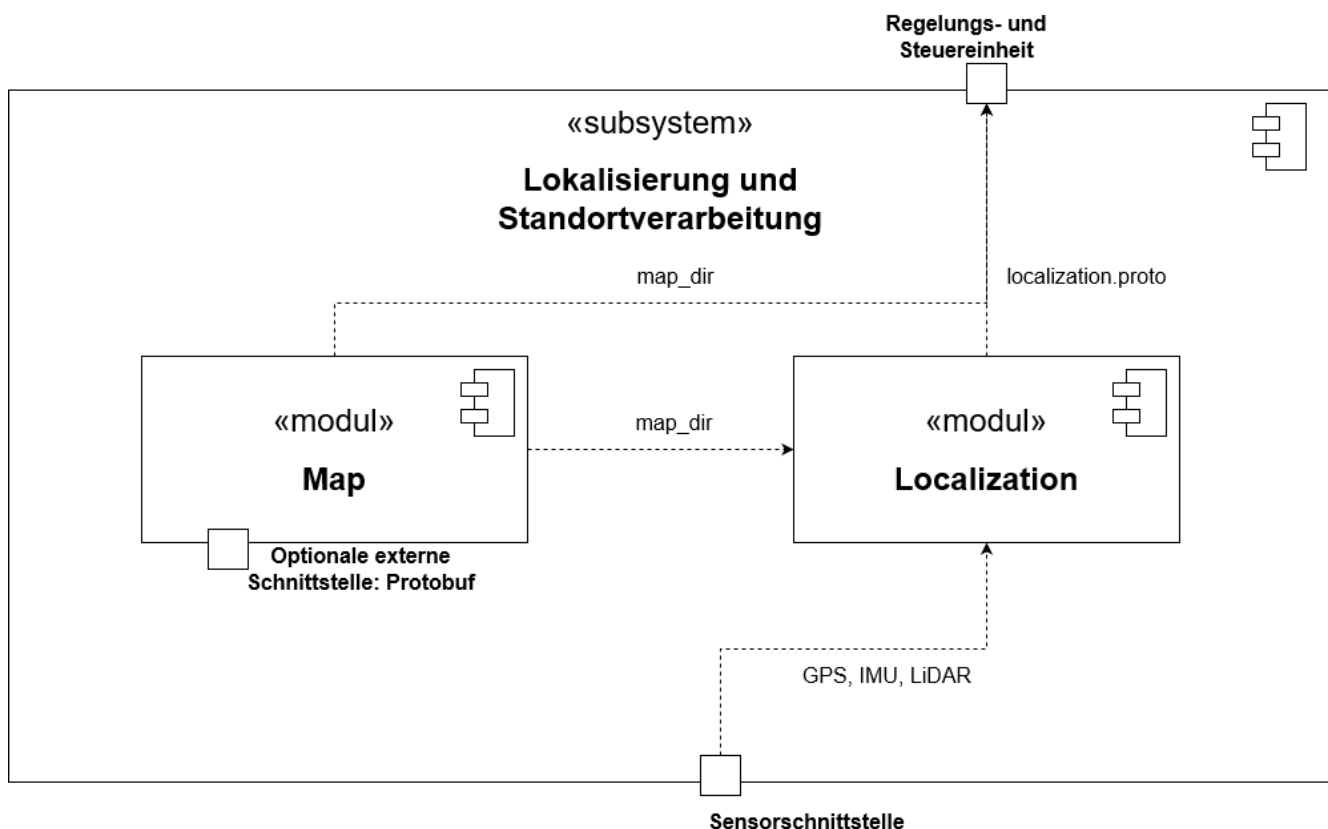


Abbildung 7. Bausteinsicht Ebene 2

Modul	Kurzbeschreibung
Map	Lädt Kartendaten ein, verwaltet sie und leitet diese an die anderen Teilnehmer im System.
Localization	Bestimmt mithilfe der Sensordaten die Position des Fahrzeugs und leitet diese an die Regelungs- und Steuereinheit.

### 5.2.1. Map

Kartendaten werden in Verzeichnissen abgelegt, in welchen zusammenhängende Kartendaten abgespeichert werden. Sie folgen dabei der folgenden Struktur:

```

<map_dir>                                # Defined by FLAGS_map_dir
|-- base_map.xml                          # Defined by FLAGS_base_map_filename
|-- routing_map.bin                       # Defined by FLAGS_routing_map_filename
|-- sim_map.bin                           # Defined by FLAGS_sim_map_filename
|-- default_end_way_point.txt             # Defined by FLAGS_end_way_point_filename
    
```

Abbildung 8. Datenstruktur einer Map

Es wird dabei zwischen der *base\_map*, *routing\_map* und *sim\_map* unterschieden.

*basemap*

Die *base\_map* ist dabei die kompletteste Darstellung mit allen Straßen, Spurgeometrien und weiteren Kennzeichen. Aus dieser werden die anderen beiden Karten generiert.

#### *routing\_map*

In dieser wird die Topologie der Fahrbahnen dargestellt.

#### *sim\_map*

Zur Visualisierung vereinfachte Darstellung von *base\_map*. Diese kann anschließend von dem Dreamview HMI verwendet werden.

Die Kartendaten werden dafür zunächst als HDMAP durch *hdmmap.cc* geladen.

```
#include "modules/map/hdmap/hdmap.h"

#include "modules/map/hdmap/hdmap_util.h"

namespace apollo {
namespace hdmap {

int HDMAP::LoadMapFromFile(const std::string& map_filename) {
  AINFO << "Loading HDMAP: " << map_filename << " ...";
  return impl_.LoadMapFromFile(map_filename);
}

int HDMAP::LoadMapFromProto(const Map& map_proto) {
  ADEBUG << "Loading HDMAP with header: "
    << map_proto.header().ShortDebugString();
  return impl_.LoadMapFromProto(map_proto);
}
```

#### Abbildung 9. *hdmmap.cc*

Dies kann sowohl über eine abgelegte Datei, als auch über Protocol Buffers bewerkstelligt werden. Die abgerufene *map.proto* unterteilt sich in eine Header- und eine Map-message.



```

message Header {
  optional bytes version = 1;
  optional bytes date = 2;
  optional Projection projection = 3;
  optional bytes district = 4;
  optional bytes generation = 5;
  optional bytes rev_major = 6;
  optional bytes rev_minor = 7;
  optional double left = 8;
  optional double top = 9;
  optional double right = 10;
  optional double bottom = 11;
  optional bytes vendor = 12;
}

message Map {
  optional Header header = 1;

  repeated Crosswalk crosswalk = 2;
  repeated Junction junction = 3;
  repeated Lane lane = 4;
  repeated StopSign stop_sign = 5;
  repeated Signal signal = 6;
  repeated YieldSign yield = 7;
  repeated Overlap overlap = 8;
  repeated ClearArea clear_area = 9;
  repeated SpeedBump speed_bump = 10;
  repeated Road road = 11;
  repeated ParkingSpace parking_space = 12;
  repeated PNCJunction pnc_junction = 13;
  repeated RSU rsu = 14;
}

```

*Abbildung 10. map.proto*

Die Inhalte in Map sind dabei von besonderem Interesse. Diese setzen sich aus wiederholenden .proto-Nachrichten zusammen, welche in den weiteren .proto-files des Ordners definiert werden.

This branch is 49 commits ahead, 553 commits behind master.

storypku and changsh726 map: imported proto unused fix

..	
BUILD	add rsu element for hmap (#12476)
map.proto	add rsu element for hmap (#12476)
map_clear_area.proto	add hmap feature
map_crosswalk.proto	add hmap feature
map_geometry.proto	All: fixed proto format.
map_id.proto	bytes id -> string id
map_junction.proto	minor fixes.
map_lane.proto	HDMa:: Add overlap region between lane and crosswalk (#1265)
map_overlap.proto	Format: use scripts/clang_format.sh to format all proto directories
map_parking_space.proto	add parking space for hmap
map_pnc_junction.proto	Localization/Map: clang-format protos.
map_road.proto	add road type for hmap
map_rsu.proto	map: imported proto unused fix
map_signal.proto	add no right turn on red for traffic light
map_speed_bump.proto	All: fixed proto format.
map_speed_control.proto	Localization/Map: clang-format protos.
map_stop_sign.proto	Localization/Map: clang-format protos.
map_yield_sign.proto	All: fixed proto format.

Abbildung 11. Protobuf-Dateien der Map

In diesen werden alle einzelne Informationen zum mapping abgebildet. Beispielsweise beinhaltet `map_lane.proto` den genauen Straßentypen, wozu unter anderem dessen Fahrspurmarkierungsart, ob es sich um eine Kurve oder Gerade handelt und die dazugehörige Länge gehören.

```

message LaneBoundaryType {
  enum Type {
    UNKNOWN = 0;
    DOTTED_YELLOW = 1;
    DOTTED_WHITE = 2;
    SOLID_YELLOW = 3;
    SOLID_WHITE = 4;
    DOUBLE_YELLOW = 5;
    CURB = 6;
  };
  // Offset relative to the starting point of boundary
  optional double s = 1;
  // support multiple types
  repeated Type types = 2;
}

message LaneBoundary {
  optional Curve curve = 1;

  optional double length = 2;
  // indicate whether the lane boundary exists in real world
  optional bool virtual = 3;
  // in ascending order of s
  repeated LaneBoundaryType boundary_type = 4;
}

```

Abbildung 12. Beispiel einer spezifischen Information in den .proto-files

Die geladenen Kartendaten können an die Lokalisierung und Regelungs- und Steuereinheit übertragen werden.

### 5.2.2. Lokalization

Zur Lokalisierung werden zwei verschiedene Methoden verwendet:

#### RTK

Real Time Kinetic Localization verwendet zwei Sensorinputs

- GPS
- IMU

#### Multi-Sensor Fusion

Diese Lokalisierungsmethode ergänzt die bisherige Sensorik um einen LiDAR-Sensor

- GPS
- IMU
- LiDAR

Es wird eine Protobuf-Datei ausgegeben, welche an die Regelungs- und Steuereinheit übertragen werden kann. Diese beinhaltet die geschätzte Position des Fahrzeugs und die damit verbundenen Standardabweichungen, um Ungenauigkeiten zu berücksichtigen.

```

message Uncertainty {
  // Standard deviation of position, east/north/up in meters.
  optional apollo.common.Point3D position_std_dev = 1;

  // Standard deviation of quaternion qx/qy/qz, unitless.
  optional apollo.common.Point3D orientation_std_dev = 2;

  // Standard deviation of linear velocity, east/north/up in meters per second.
  optional apollo.common.Point3D linear_velocity_std_dev = 3;

  // Standard deviation of linear acceleration, right/forward/up in meters per
  // square second.
  optional apollo.common.Point3D linear_acceleration_std_dev = 4;

  // Standard deviation of angular velocity, right/forward/up in radians per
  // second.
  optional apollo.common.Point3D angular_velocity_std_dev = 5;

  // TODO: Define covariance items when needed.
}

message LocalizationEstimate {
  optional apollo.common.Header header = 1;
  optional apollo.localization.Pose pose = 2;
  optional Uncertainty uncertainty = 3;

  // The time of pose measurement, seconds since 1970-1-1 (UNIX time).
  optional double measurement_time = 4; // In seconds.

  // Future trajectory actually driven by the drivers
  repeated apollo.common.TrajectoryPoint trajectory_point = 5;

  // msf status
  optional MsfStatus msf_status = 6;
  // msf quality
  optional MsfSensorMsgStatus sensor_status = 7;
}

```

Abbildung 13. Protobuf-Datei der Lokalisierung

## 6. Laufzeitsicht

Diese Sicht erklärt konkrete Abläufe und Beziehungen zwischen Bausteinen in Form von Szenarien. Dabei werden im Gegensatz zur Bausteinsicht dynamische Aspekte visualisiert.

### 6.1. Annäherung an ein Stoppschild

Die Fahrwegermittlung ist komplex, grundlegende Zusammenhänge sollen jedoch anhand eines Beispiels beschrieben werden. ApolloAuto verwendet für die Beschreibung komplexer Fahrsituationen Szenarien. Anhand dieser Szenarien können Fahreigenschaften dynamisch an bestimmte Situationen angepasst werden. Die vollumfänglichen Möglichkeiten, die in ApolloAuto für diese Szenarien vorgesehen sind, können im Modul Storytelling eingesehen werden. So wird hier die Grundlage geschaffen, um Verkehrssituationen mit den verschiedensten Eigenschaften zu beschreiben. Das umfasst beispielsweise Stoppschilder, Ampeln und Fußgängerüberwege.

In dieser Komplexität ist jedoch die Betrachtung für eine vereinfachte Laufzeitsicht nicht notwendig. Stattdessen betrachten wir Teile des im Planning integrierte Szenarios `stop_sign` um Interaktionen zwischen Modulen zu verdeutlichen. Das stark vereinfachte Beispiel für die Visualisierung umfasst daher, dass sich unser Fahrzeug einer Kreuzung mit einem Stoppschild annähert, ohne dass andere Verkehrsteilnehmer involviert sind.

[Laufzeitdiagramm] | *Laufzeitdiagramm.png*

*Abbildung 14. Sequenzdiagramm für die Annäherung an eine Kreuzung mit Stoppschild*

LiDAR-, Kamera- und Radardaten werden durchgehend an das Perception-Modul weitergeleitet, worin diese ausgewertet werden. Parallel dazu erhält auch die Localization Daten von GPS, IMU und LiDAR-Sensoren.

Wird nun ein Stoppschild wahrgenommen, muss das Fahrzeug dafür das Fahrverhalten ändern. Dies geschieht in Planning. Das Modul verwendet dafür die Kartendaten aus Map, die Umgebungsaufnahme aus Perception und die festgestellte Fahrzeugposition aus Localization. In Planning wird das Fahrverhalten für bestimmte Fahrsituationen angepasst. Integriert sind in ApolloAuto derzeit die Szenarien für das Folgen der Fahrspur, Kreuzungen, Fahrspurwechsel, Park-and-go und Notfallsituationen. Im vorliegenden Fall wird das Fahrzeugverhalten vom Folgen der Fahrspur auf das Kreuzungsverhalten geändert.

Die daraus resultierende geplante Route wird an das Control-Modul weitergegeben. Dieses gibt die Regelungsvorgaben für das Fahrzeug wiederum an CANBus weiter. Dort können diese Daten zur Ausführung an das Fahrzeug übermittelt werden. Des Weiteren können auch Monitor und das Dreamview HMI diese Daten empfangen. In letzterem kann ein Mode Change Request gestellt werden, welcher direkt zur Control zurückführt.

# 7. Verteilungssicht

In der Verteilungssicht wird auf den Betrieb von ApolloAuto eingegangen. Was ist die Umgebung, in der die Software ausgeführt wird?

## 7.1. Infrastruktur

Im Verteilungsdiagramm wird die Implementierung von ApolloAuto auf einem Ubuntu-PC aufgezeigt. Innerhalb dieses devices kommuniziert ApolloAuto mit keinen weiteren Programmen.

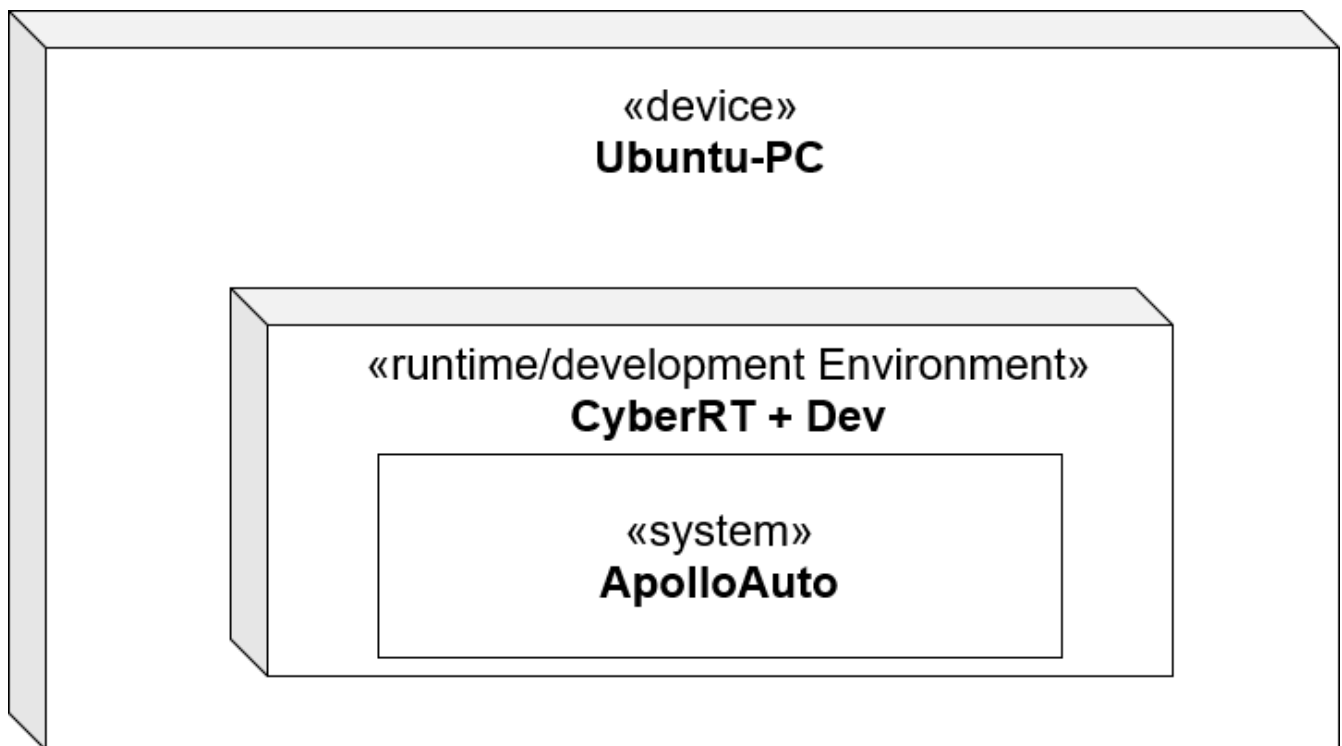


Abbildung 15. Deployment von ApolloAuto auf Ubuntu

Auf einem Ubuntu-PC ist zunächst die Laufzeit bzw. Entwicklungsumgebung zu installieren. Seit Apollo 3.5 wird dafür die speziell für ApolloAuto entwickelte Umgebung CyberRT verwendet, welche ROS abgelöst hat. Parallel zu CyberRT existiert auch noch eine vollumfänglichere Dev-Umgebung.

Mithilfe dieser Frameworks kann das System ApolloAuto ausgeführt werden.

## 8. Querschnittliche Konzepte

In diesem Kapitel werden Konzepte aufgeführt, die systemweite Bedeutung haben.

### 8.1. Common

Dateien, die in common-Ordnern aufgeführt werden, sind für mehrere Funktionen im übergeordneten Ordner von Bedeutung. Anschaulich kann dies am modules-Ordner aufgezeigt werden.


master

apollo / modules / common /

Go to file

Add file


...

 shr-project and changsh726 Release Build: Installation ready for modules/common/vehicle\_model

6ed96a1 16 days ago


History

..

 adapters


Dkit: Update perception configurations cherry pick from r5.5.0 (#13354)

4 months ago

 configs


Dreamview loop routing function

7 months ago

 data


Release Build: make scripts/bootstrap.sh work

4 months ago

 filters


Bazel: buildifier --lint=fix all BUILD files

13 months ago

 kv\_db


Build: explicit sqlite3 dependency for modules/common/kv\_db

11 months ago

 latency\_recorder


modules/common: apollo::common::time::Clock retired in favor to cyber...

10 months ago

 math


modules.common: fix eigen align issues of kalman\_filter\_test

7 months ago

 monitor\_log


Revert "Dreamview:Open space planner (#13709)" (#13717)

2 months ago

 proto


Common: add direction proto

9 months ago

 status


Bazel: buildifier --lint=fix all BUILD files

13 months ago

 util


MockTime support: realtime for guardian; clock mode for dreamview

10 months ago

 vehicle\_model


Release Build: Installation ready for modules/common/vehicle\_model

16 days ago

 vehicle\_state


control: fix vehicle\_state pitch error

2 months ago

 BUILD


Release Build: Installation ready for modules/common/vehicle\_model

16 days ago

 README.md

Docs: update README for modules/common

10 months ago

 README\_cn.md

Docs: update README for modules/common

10 months ago

Abbildung 16. common-Ordner bei den modules

Die Inhalte in diesem Ordner zeichnen sich dadurch aus, dass sie für mehrere andere Module auch von Bedeutung sind. Anstatt sie daher für alle Funktionen einzeln festzulegen, wird ein allgemeingültiger Standard gesetzt, der von allen anderen Modulen implementiert wird. So wird beispielsweise die Protobuf-Datei geometry.proto unter anderem von den Modulen Prediction und Planning verwendet. Dem liegt zugrunde, dass Geometriebeschreibungen sowohl für die Vorhersage vom Verhalten anderer Verkehrsteilnehmer, als auch für die allgemeine Routenplanung von Bedeutung sind.

### 8.2. Guardian

Autonomes Fahren bedarf hohen Sicherheitsanforderungen. Um zusätzliche Sicherheiten zu gewähren, ist es sinnvoll, ein Modul zu verwenden, welches eine Schutzfunktion einnimmt und bei kritischen Fehlern oder Zuständen das Fahrzeug stoppt. Solche Zustände können beispielsweise Fehlfunktionen von Sensoren oder erkannte Objekte im Fahrweg sein. Es greift dafür auf die Regelungsdaten des Control-Moduls, den Systemstatus vom Monitor und den Fahrzeugzustand zu.

```

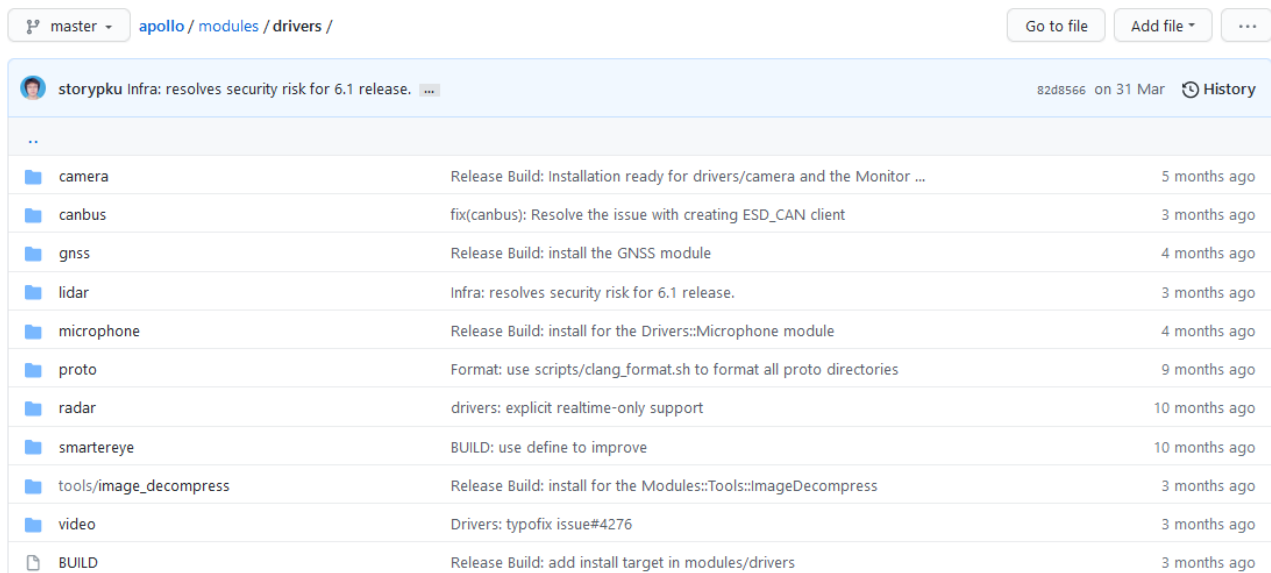
for (int i = 0; i < chassis_.surround().sonar_range_size(); ++i) {
    if ((chassis_.surround().sonar_range(i) > 0.0 &&
        chassis_.surround().sonar_range(i) < 2.5) ||
        chassis_.surround().sonar_range(i) > 30) {
        INFO << "Object detected or ultrasonic sensor fault output, will do "
            "emergency stop!";
        obstacle_detected = true;
    }
}

```

Abbildung 17. Beispiel für Fehler, der einen Eingriff durch den Guardian verursacht

## 8.3. Drivers

Die Implementierung eines Systems mit so vielen unterschiedlichen Schnittstellen kann für den Anwender sehr unübersichtlich werden. Um die Ankopplung zur externen Schnittstellen wie Sensoren und Kamerasysteme von der übrigen Softwareentwicklung zu trennen, wird das Modul Drivers verwendet.



Module	Commit Message	Time
camera	Release Build: Installation ready for drivers/camera and the Monitor ...	5 months ago
canbus	fix(canbus): Resolve the issue with creating ESD_CAN client	3 months ago
gnss	Release Build: install the GNSS module	4 months ago
lidar	Infra: resolves security risk for 6.1 release.	3 months ago
microphone	Release Build: install for the Drivers::Microphone module	4 months ago
proto	Format: use scripts/clang_format.sh to format all proto directories	9 months ago
radar	drivers: explicit realtime-only support	10 months ago
smartereye	BUILD: use define to improve	10 months ago
tools/image_decompress	Release Build: install for the Modules::Tools::ImageDecompress	3 months ago
video	Drivers: typofix issue#4276	3 months ago
BUILD	Release Build: add install target in modules/drivers	3 months ago

Abbildung 18. Drivers Modul

Die hier vorliegenden Dateien dienen als Hardwaretreiber, die das Einlesen und Verwalten von deren Datensätzen bewerkstelligen.

## 8.4. Protocol Buffers

Zur Informationsübertragung werden an vielen Schnittstellen Protocol Buffers verwendet. Damit wird garantiert, dass ein leicht lesbares Datenformat verwendet wird, welches gleichzeitig effizient und gut erweiterbar ist. Es kann außerdem damit berücksichtigt werden, dass verschiedene Programmiersprachen (C++ und Python) zur Entwicklung verwendet werden können.

## 8.5. Funktionale Abgrenzung

Im Laufe dieser Dokumentation wurde bereits viel auf die einzelnen Module eingegangen, in welche die Funktionen des Systems eingeteilt wurden.



master
apollo / modules /
Go to file
Add file
...

changsh726 and AndrewXWei Transform: adapts to simulation mode
✓ ec30ae6 3 days ago
History

..		
audio	Release Build: apply install rule to the audio module	4 months ago
bridge	Infra: resolves security risk for 6.1 release.	3 months ago
calibration	Revert "Dreamview:Open space planner (#13709)" (#13717)	2 months ago
canbus	Canbus: bug fix (#13760)	last month
common	Release Build: Installation ready for modules/common/vehicle_model	16 days ago
contrib	Release Build: Installation ready for modules/contrib/cyber_bridge	3 months ago
control	control: delete redundant judgment conditions in set_gear_location	27 days ago
data	cyber_recorder: add support for black listing topic names in record l...	8 months ago
dreamview	dreamview: fix bug at sim control (#13727)	last month
drivers	Infra: resolves security risk for 6.1 release.	3 months ago
guardian	Release Build: install for the Guardian module	4 months ago
localization	localization: cyber_record_parser keep the original interface	16 days ago
map	updated runtime standalone functions	16 days ago
monitor	fix(canbus): Resolve the issue with creating ESD_CAN client	3 months ago
perception	Transform: adapts to simulation mode	3 days ago
planning	Revert "Dreamview:Open space planner (#13709)" (#13717)	2 months ago
prediction	Bazel: fix visibility bug	5 months ago
routing	Release Build: Installation ready for modules/routing/topo_creator	2 months ago
storytelling	Release Build: install for storytelling	4 months ago
task_manager	Revert "Dreamview:Open space planner (#13709)" (#13717)	2 months ago
third_party_perception	Format: use scripts/clang_format.sh to format all proto directories	9 months ago
tools	updated runtime standalone functions	16 days ago
transform	Transform: adapts to simulation mode	3 days ago
v2x	Infra: resolves security risk for 6.1 release.	3 months ago

Abbildung 19. Einteilung von Funktionalitäten in Module

Mithilfe von diesen wird die Abgrenzung der einzelnen Funktionalitäten bewerkstelligt. Damit folgt die Architektur den Prinzipien Separation of Concerns und Lose Kopplung, da einzelne Aufgaben als Teillösungen umgesetzt wurden, deren Veränderungen keine unvorhersagbaren Einflüsse auf das Gesamtsystem haben.

# 9. Entwurfsentscheidungen

Im folgenden Kapitel wird eine interessante Entscheidung im Entwicklungsverlauf des Projektes genauer betrachtet

## 9.1. ROS vs CyberRT

### *Fragestellung*

Zur Entwicklung eines autonom fahrenden Fahrzeugs muss festgelegt werden, mithilfe welches Frameworks dies umgesetzt wird. Dieses muss verschiedene Funktionen umfassen und eine Programmbibliothek liefern, welche für diesen Anwendungsfall geeignet ist.

### *Relevante Einflussfaktoren*

Funktionsumfang Programmbibliothek Anforderungen des Zielsystems

### *Alternative: ROS*

Zum Beginn der Erarbeitung autonomer Fahrzeuge wurde als Framework ROS verwendet. Dieses, ursprünglich für Roboter entwickelte Framework (Robot Operating System) bietet grundlegende Funktionen und Bibliotheken, die für die Entwicklung technischer Anwendungen notwendig waren. Es zeichnet sich durch hohe Flexibilität aus, kommt jedoch bei Echtzeit-Anforderungen an seine Grenzen. Es ermöglicht eine Programmierung in Python und C++ und stellt somit eine gute Grundlage für die Entwicklung eigener Systeme dar.

Bei der Zielsetzung, Level 4 autonomes Fahren umzusetzen, hat man sich jedoch aufgrund der fehlenden Berücksichtigung von Echtzeitanforderungen und einer Bibliothek, die nicht dediziert für autonomes Fahren entwickelt wurde, gegen eine Weiterverwendung von ROS entschieden.

### *Entscheidung: Entwicklung des eigenen Frameworks CyberRT*

Mit Apollo 3.5 wurde das eigens von Baidu entwickelte Framework CyberRT eingeführt. Diese auf hohe Performance ausgerichtete Umgebung, die explizit für das autonome Fahren entwickelt wurde, zeichnet sich durch hohe Nebenläufigkeit, geringe Latenz und hohen Datendurchsatz aus. Dies soll mehrere Vorteile mit sich bringen:

- Beschleunigte Entwicklung
  - Vielzahl an Entwicklungstools
  - Große Anzahl an Sensortreibern
- Einfacher Einsatz
  - Effiziente und effektive Nachrichtenkommunikation
  - Portable und weniger Abhängigkeiten
- Vereinfachte Implementierung in eigene Fahrzeuge
  - Plug-and-Play-System
  - ApolloAuto verwendet das default-open-source runtime framework

# 10. Qualitätsanforderungen

Dieser Abschnitt enthält die Qualitätsanforderungen als Qualitätsbaum mit Szenarien.

## *Inhalt*

Dieser Abschnitt enthält möglichst alle Qualitätsanforderungen als Qualitätsbaum mit Szenarien. Die wichtigsten davon haben Sie bereits in Abschnitt 1.2 (Qualitätsziele) hervorgehoben.

Nehmen Sie hier auch Qualitätsanforderungen geringerer Priorität auf, deren Nichteinhaltung oder -erreichnung geringe Risiken birgt.

## *Motivation*

Weil Qualitätsanforderungen die Architekturentscheidungen oft maßgeblich beeinflussen, sollten Sie die für Ihre Stakeholder relevanten Qualitätsanforderungen kennen, möglichst konkret und operationalisiert.

## 10.1. Qualitätsbaum

### *Inhalt*

Der Qualitätsbaum (à la ATAM) mit Qualitätsszenarien an den Blättern.

### *Motivation*

Die mit Prioritäten versehene Baumstruktur gibt Überblick über die — oftmals zahlreichen — Qualitätsanforderungen.

### *Form*

- Baumartige Verfeinerung des Begriffes „Qualität“, mit „Qualität“ oder „Nützlichkeit“ als Wurzel.
- Mindmap mit Qualitätsoberbegriffen als Hauptzweige

In jedem Fall sollten Sie hier Verweise auf die Qualitätsszenarien des folgenden Abschnittes aufnehmen.

## 10.2. Qualitätsszenarien

### *Inhalt*

Konkretisierung der (in der Praxis oftmals vagen oder impliziten) Qualitätsanforderungen durch (Qualitäts-)Szenarien.

Diese Szenarien beschreiben, was beim Eintreffen eines Stimulus auf ein System in bestimmten Situationen geschieht.

Wesentlich sind zwei Arten von Szenarien:

- Nutzungsszenarien (auch bekannt als Anwendungs- oder Anwendungsfallszenarien) beschreiben, wie das System zur Laufzeit auf einen bestimmten Auslöser reagieren soll. Hierunter fallen auch Szenarien zur Beschreibung von Effizienz oder Performance. Beispiel: Das System beantwortet eine Benutzeranfrage innerhalb einer Sekunde.

- Änderungsszenarien beschreiben eine Modifikation des Systems oder seiner unmittelbaren Umgebung. Beispiel: Eine zusätzliche Funktionalität wird implementiert oder die Anforderung an ein Qualitätsmerkmal ändert sich.

#### *Motivation*

Szenarien operationalisieren Qualitätsanforderungen und machen deren Erfüllung mess- oder entscheidbar.

Insbesondere wenn Sie die Qualität Ihrer Architektur mit Methoden wie ATAM überprüfen wollen, bedürfen die in Abschnitt 1.2 genannten Qualitätsziele einer weiteren Präzisierung bis auf die Ebene von diskutierbaren und nachprüfbaren Szenarien.

#### *Form*

Entweder tabellarisch oder als Freitext.

# **11. Risiken und technische Schulden**

Eine nach Prioritäten geordnete Liste der erkannten Architekturrisiken und/oder technischen Schulden.

## **11.1. Aufwand der Implementierung**

Es müssen hardwareseitig alle Voraussetzungen genau erfüllt werden, damit das Setup funktionieren kann.

## **11.2. Komplexes Gesamtsystem**

(?-→) Für die Entwicklung des Gesamtsystems ist es von großer Bedeutung, sicherzustellen, dass durch Weiterentwicklungen dieses komplexe System nicht funktionsunfähig wird.

## **11.3. Ausstehende Implementierung vieler Verkehrssituationen**

# 12. Glossar

Die wesentlichen fachlichen und technischen Begriffe, die Stakeholder im Zusammenhang mit dem System verwenden.

## 12.1. Einstieg

\*Was ist autonomes Fahren?

## 12.2. Begriffe

\*Die folgenden Begriffe werden im Kontext dieser Dokumentation verwendet.

Begriff	Definition
Protobuf	<Definition-1>
CAN-Bus	<Definition-2>
LIdAR	<Definition-2>

### *Inhalt*

Die wesentlichen fachlichen und technischen Begriffe, die Stakeholder im Zusammenhang mit dem System verwenden.

Nutzen Sie das Glossar ebenfalls als Übersetzungsreferenz, falls Sie in mehrsprachigen Teams arbeiten.

### *Motivation*

Sie sollten relevante Begriffe klar definieren, so dass alle Beteiligten

- diese Begriffe identisch verstehen, und
- vermeiden, mehrere Begriffe für die gleiche Sache zu haben.

### *Form*

- Zweispaltige Tabelle mit <Begriff> und <Definition>
- Eventuell weitere Spalten mit Übersetzungen, falls notwendig.

# 13. Anhang

## 13.1. Erfahrungen mit Apollo

ApolloAuto ist als Anwendungsfall sehr interessant aber auch gleichermaßen umfangreich. Während wir uns mittlerweile sicher sind, grundlegende Konzepte und Strukturen verstanden zu haben, sind wir uns gleichermaßen sicher, dass viele Inhalte bei tieferem Einblick auch anders dargestellt werden sollten. Fünf iterative Durchgänge durch unsere Dokumentation, bei denen wir kritisch Punkte diskutieren, würden daher wahrscheinlich fünf mal dazu führen, dass wir noch einige Sachen anders beschreiben.

## 13.2. Erfahrungen mit Arc42

Arc42 hat bei der Aufarbeitung von ApolloAuto sehr geholfen, da es eine Struktur bietet, nach welcher man sich durch das Gesamtsystem arbeiten kann. Die gestellten Fragestellungen sind sinnvoll angeordnet, um sich nach und nach tiefer einzuarbeiten.

Bei der Beschreibung eines Gesamtsystems, an welchem man weder Erfahrungen sammeln konnte noch selbst mitgearbeitet hat, wie in der vorliegenden Aufgabenstellung, kommen jedoch mehrfach Probleme auf. Die Arbeitsweise ist grundsätzlich iterativ, da mit der Zeit viele Zusammenhänge immer besser verstanden werden und Fehler oder Unklarheiten auffallen. Dadurch, dass viele Kapitel inhaltlich direkt aufeinander aufbauen, bewirken späte Abänderungen große Überarbeitungen an der gesamten Dokumentation. So zieht sich beispielsweise eine falsch verstandene Schnittstelle in der Kontextabgrenzung eventuell durch alle Abbildungen und Beschreibungen der nächsten drei Kapitel.

## 13.3. Erfahrungen mit docToolchain

docToolchain empfanden wir als sehr interessant, da es nach anfänglicher Implementierung eine sehr übersichtliche Bearbeitung der Dokumentation in der Entwicklungsumgebung unserer Wahl ermöglichte - ganz unabhängig davon, ob das Eclipse oder Visual Studio ist. Die Einbindung von PlantUML-Diagrammen hielt uns leider etwas länger auf, als sie es wahrscheinlich sollte, da wir einige Kleinigkeiten übersehen haben. Aber auch hier gilt - nachdem man es einmal gemacht hat, war es eigentlich ganz einfach.

## 13.4. Sonstige Anmerkungen

Insgesamt sind wir zwiegespaltener Meinung zu dieser Aufgabe. Der kombinierte Aufwand aus einer neuen Implementierung, einem sehr komplexen, zu beschreibenden Architektur und dem Umfang von Arc42 bringt uns zeitlich an die Grenzen.

Gleichzeitig ist es schwer zu sagen, worauf man hätte verzichten sollen. Dass die Inhalte nicht nur kurz in der Vorlesung angesprochen wurden, sondern inklusive der Implementierung auf dem eigenen Computer bearbeitet werden mussten, hat für den Lerneffekt einen enormen Mehrwert. Auch dass in einem solchen Projekt ein sehr interessantes Anwendungsbeispiel integriert wird ist auch aus überfachlichem Interesse heraus sehr spannend. In der grundsätzlichen Konzeption ist

das Projekt daher definitiv sinnvoll. Leider waren wir jedoch zeitlich zu oft gezwungen, Inhalte nicht in der Tiefe zu betrachten, wie wir es gerne gemacht hätten. Wenn es daher eine Möglichkeit gäbe, den Umfang etwas zu verkleinern oder die Abgabe außerhalb der Klausurenphase zu legen, dann würden sich die Probleme erübrigen.