ApolloAuto

Inhaltsverzeichnis

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

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

1.3. Stakeholder

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

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

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.

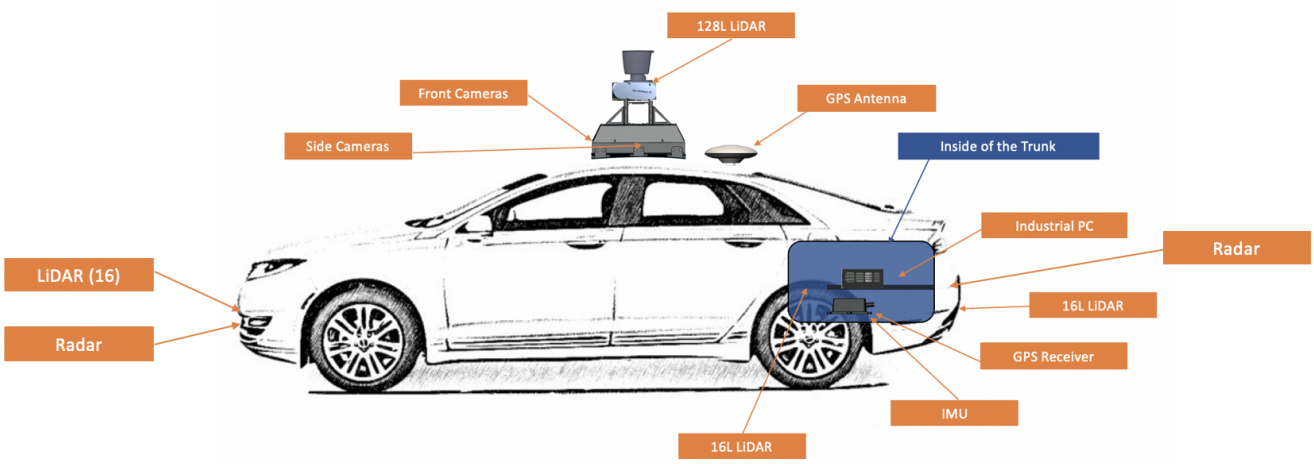
Tabelle 1. Installationsanforderungen

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* (Nvidia Treiber)
Docker	*Docker-CE 19.03 sowie alle nachfolgenden Versionen* (Installation von Docker auf Ubuntu)
NVIDIA Container Toolkit	*NVIDIA 455.32.00 Treiber sowie alle nachfolgenden Versionen* (NVIDIA-Docker Github Link)

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.



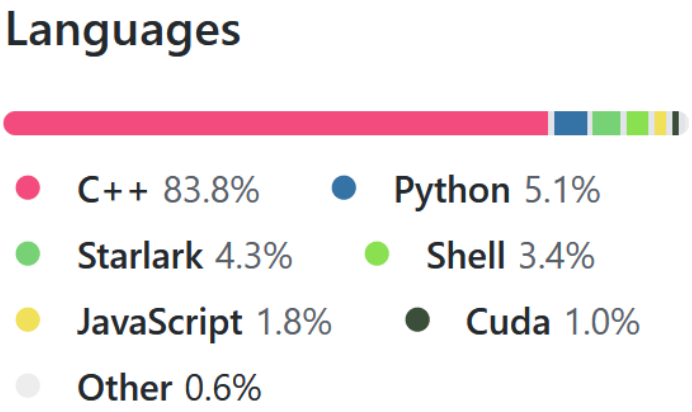
Die Spezifikationen anderer Versionen können der Github Seite entnommen werden.

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	***Lizenzbedingungen hier eintragen***

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.*

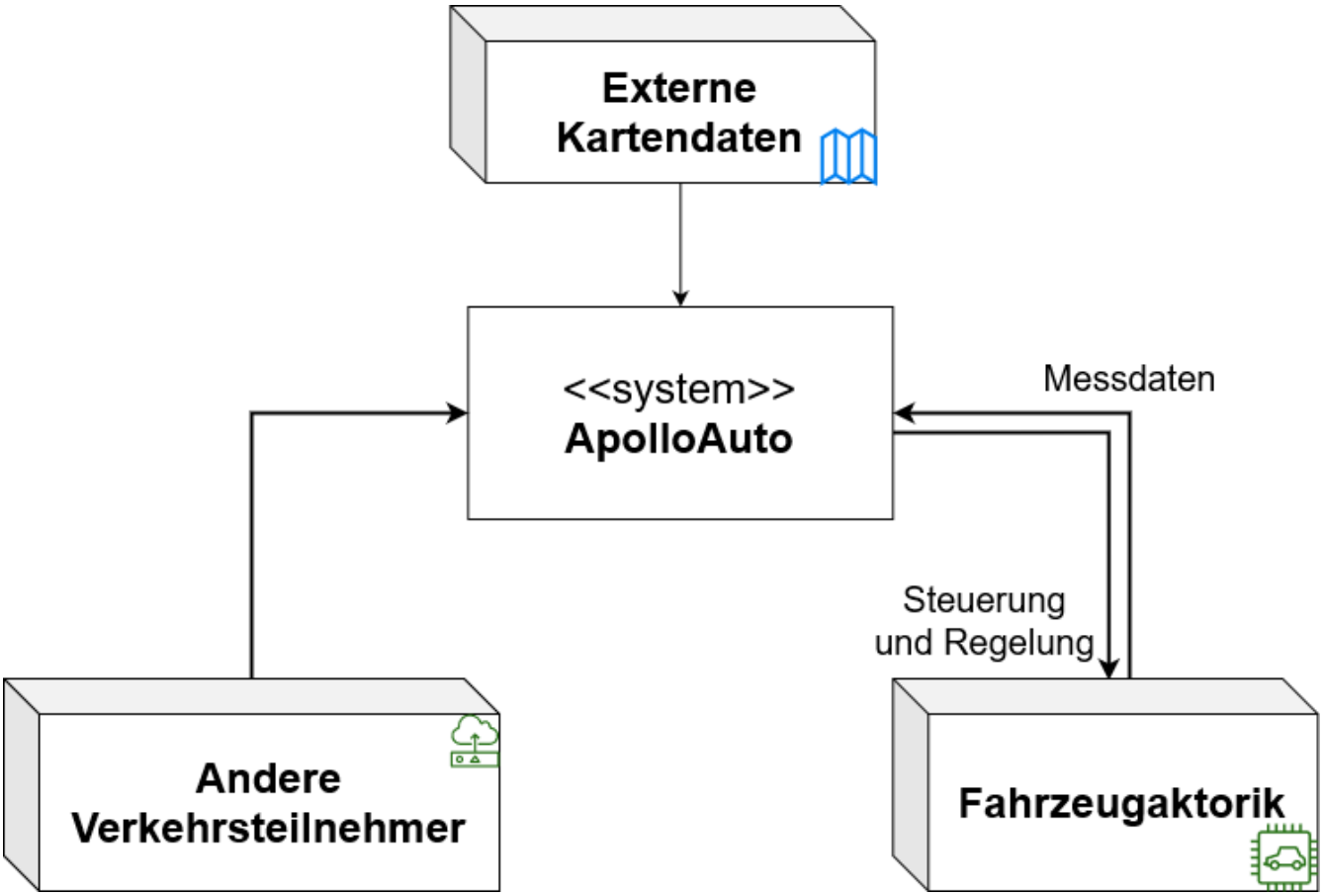


3. Kontextabgrenzung

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).



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

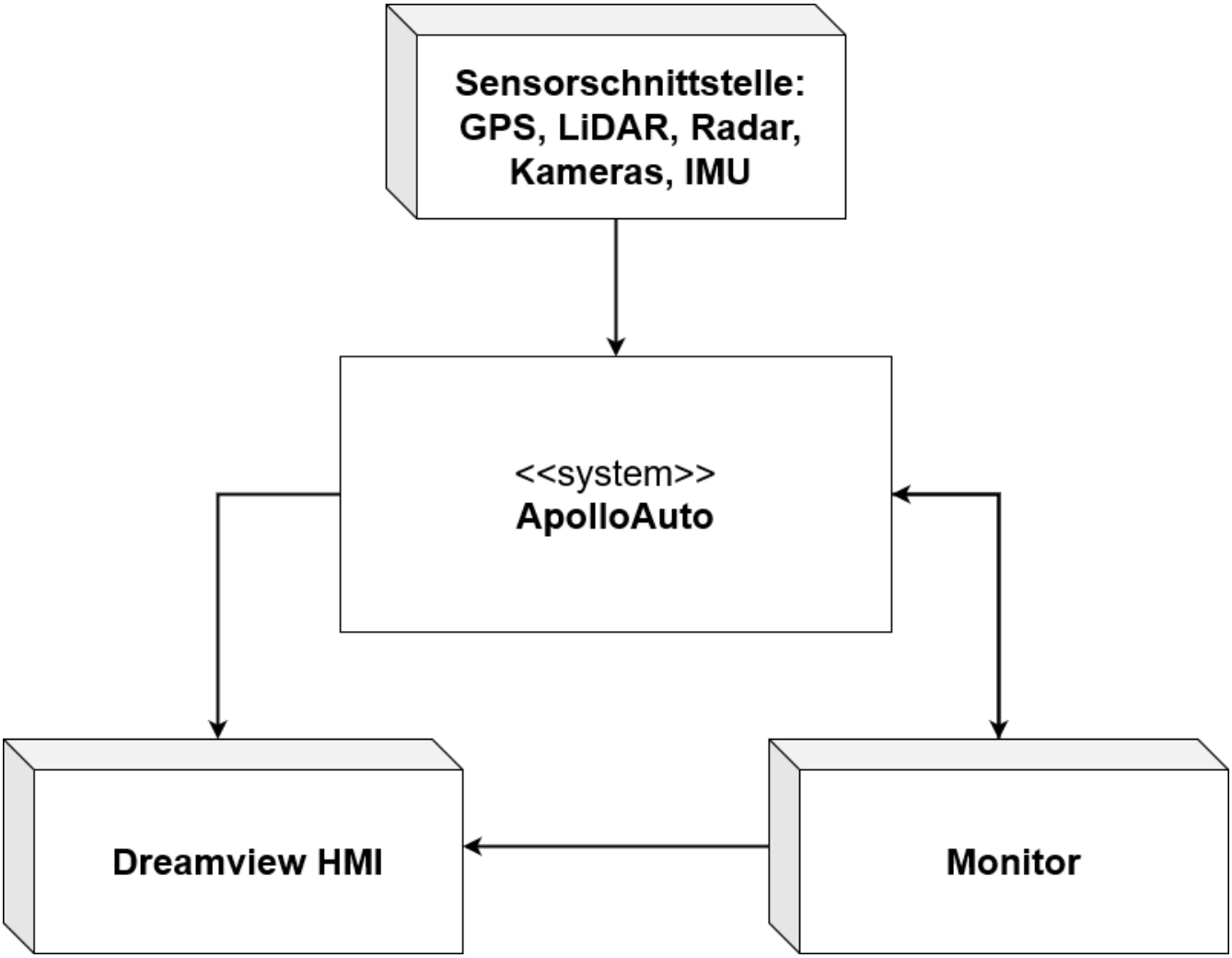
Die Sensorik, welche das Umfeld des Fahrzeugs aufnimmt, wird von ApolloAuto verwertet. Dazu gehören beispielsweise das GPS und LiDAR-Sensoren.

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.



Sensorschnittstelle

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

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.

4. Lösungsstrategie

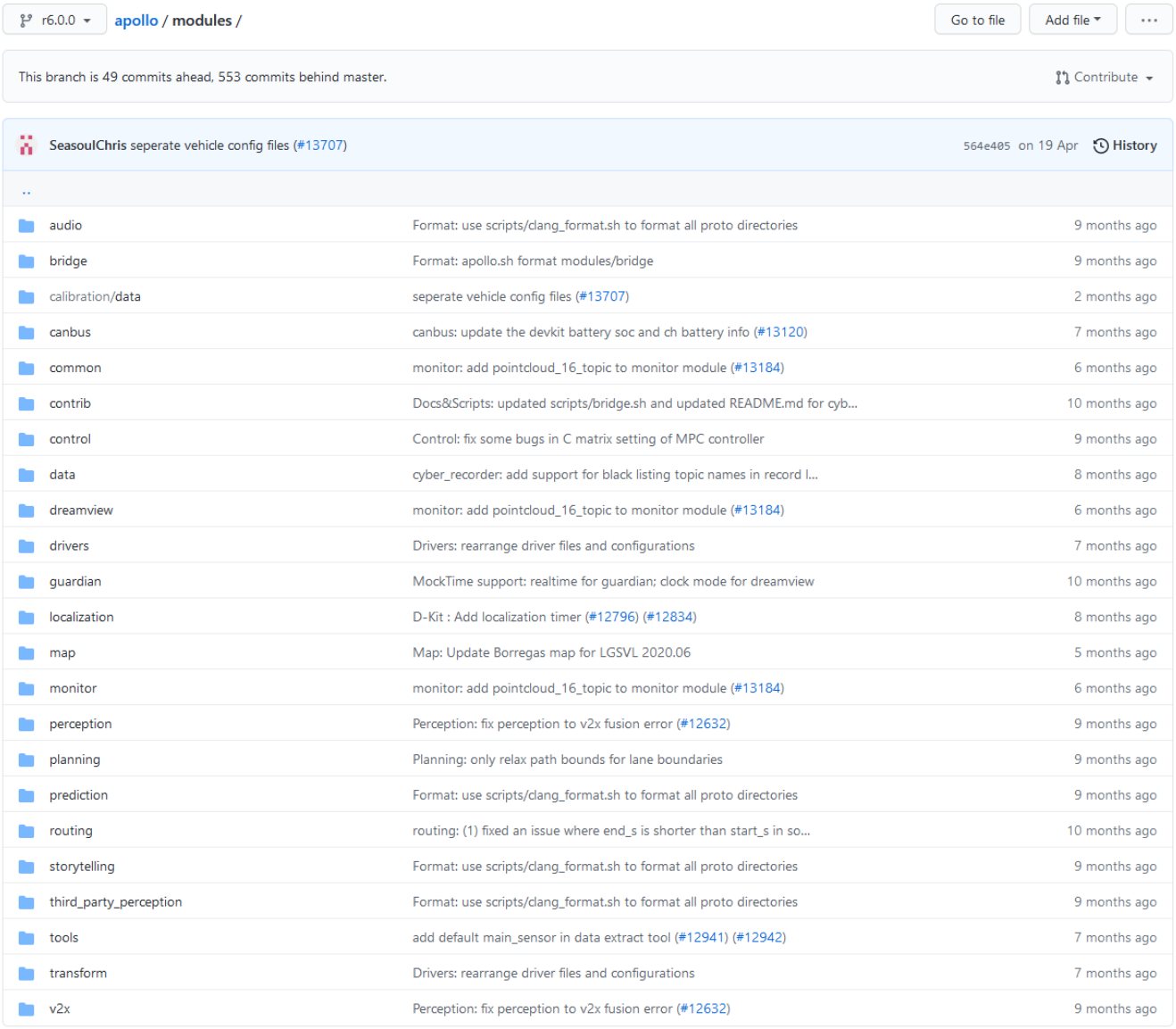
In diesem Kapitel wird ein Überblick über die Architektur geboten. Dabei stehen die Ziele und Lösungsansätze im Fokus.

4.1. Einstieg

Qualitätsziel	Dem zuträgliche Ansätze in der Architektur
Vereinfachte Entwicklung autonomer Fahrzeuge	• *Cyber RT Framework, Laufzeitumgebung die extra für das Autonome Fahren entwickelt wurde* • *Modularer Aufbau der Softwarepakete* • *Python API ermöglicht die Programmierung durch Entwickler, die mit C++ nicht vertraut sind*
Schnelle Tests von autonomen Fahrzeugen	• *Dreamview Human Machine Interface zur Überwachung des Fahrzeugzustands*
Flexible Architektur	• *Continious Integration Ansatz, bei welchem in Updatezyklen die bestehenden Funktionen erweitert werden* • *Ablage des Repositories auf Github* • *Modularer Ansatz garantiert Erweiterbarkeit*
Verständliche Darstellung	• *Implementationsbeschreibung über Readme-Dateien in der Repository* • *Kurzbeschreibungen der Ordnerinhalte* • *Grafische Darstellungen von einigen übergeordneten Ansätzen, wie bspw. der Hardware- und Softwareübersicht*
Hohe Fehlertoleranz	• *Guardian, welcher bei kritischen Input-Werten von Steuergerät oder Monitor direkt auf den CANBus zugreifen kann* • *Verwendung vieler verschiedener Sensoren*

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) 564e405 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

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
  - CANBus
  - Guardian
  - 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 Konzept eingegangen.

5. Bausteinsicht

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

5.1. Whitebox Gesamtsystem

Name	Verantwortung
<Blackbox 1>	*<Text>*
<Blackbox 2>	*<Text>*

5.1.1. <Name Blackbox 1>

Beschreiben Sie die <Blackbox 1> anhand des folgenden Blackbox-Templates:

- Zweck/Verantwortung
- Schnittstelle(n), sofern diese nicht als eigenständige Beschreibungen herausgezogen sind. Hierzu gehören eventuell auch Qualitäts- und Leistungsmerkmale dieser Schnittstelle.
- (Optional) Qualitäts-/Leistungsmerkmale der Blackbox, beispielsweise Verfügbarkeit, Laufzeitverhalten o. Ä.
- (Optional) Ablageort/Datei(en)
- (Optional) Erfüllte Anforderungen, falls Sie Traceability zu Anforderungen benötigen.
- (Optional) Offene Punkte/Probleme/Risiken

<Zweck/Verantwortung>

<Schnittstelle(n)>

<(Optional) Qualitäts-/Leistungsmerkmale>

<(Optional) Ablageort/Datei(en)>

<(Optional) Erfüllte Anforderungen>

<(optional) Offene Punkte/Probleme/Risiken>

5.1.2. <Name Blackbox 2>

<Blackbox-Template>

5.1.3. <Name Blackbox n>

<Blackbox-Template>

5.1.4. <Name Schnittstelle 1>

…

5.1.5. <Name Schnittstelle m>

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. Fahrwegermittlung

Sequenzdiagramm Zeitliche Abfolge der Kumminikation zwischen Subsystemen in der 1. Ebene von oben nach unten

- <hier Laufzeitdiagramm oder Ablaufbeschreibung einfügen>
- <hier Besonderheiten bei dem Zusammenspiel der Bausteine in diesem Szenario erläutern>

[Laufzeitdiagramm] | *Laufzeitdiagramm.png*

6.2. *<Bezeichnung Laufzeitszenario 2>*

...

6.3. *<Bezeichnung Laufzeitszenario n>*

...

Inhalt

Diese Sicht erklärt konkrete Abläufe und Beziehungen zwischen Bausteinen in Form von Szenarien aus den folgenden Bereichen:

- Wichtige Abläufe oder *Features*: Wie führen die Bausteine der Architektur die wichtigsten Abläufe durch?
- Interaktionen an kritischen externen Schnittstellen: Wie arbeiten Bausteine mit Nutzern und Nachbarsystemen zusammen?
- Betrieb und Administration: Inbetriebnahme, Start, Stop.
- Fehler- und Ausnahmeszenarien

Anmerkung: Das Kriterium für die Auswahl der möglichen Szenarien (d.h. Abläufe) des Systems ist deren Architekturrelevanz. Es geht nicht darum, möglichst viele Abläufe darzustellen, sondern eine angemessene Auswahl zu dokumentieren.

Motivation

Sie sollten verstehen, wie (Instanzen von) Bausteine(n) Ihres Systems ihre jeweiligen Aufgaben erfüllen und zur Laufzeit miteinander kommunizieren.

Nutzen Sie diese Szenarien in der Dokumentation hauptsächlich für eine verständlichere Kommunikation mit denjenigen Stakeholdern, die die statischen Modelle (z.B. Bausteinsicht, Verteilungssicht) weniger verständlich finden.

Form

Für die Beschreibung von Szenarien gibt es zahlreiche Ausdrucksmöglichkeiten. Nutzen Sie beispielsweise:

- Nummerierte Schrittfolgen oder Aufzählungen in Umgangssprache
- Aktivitäts- oder Flussdiagramme
- Sequenzdiagramme
- BPMN (Geschäftsprozessmodell und -notation) oder EPKs (Ereignis-Prozessketten)
- Zustandsautomaten
- …

7. Verteilungssicht

Inhalt

Die Verteilungssicht beschreibt:

1. die technische Infrastruktur, auf der Ihr System ausgeführt wird, mit Infrastrukturelementen wie Standorten, Umgebungen, Rechnern, Prozessoren, Kanälen und Netztopologien sowie sonstigen Bestandteilen, und
2. die Abbildung von (Software-)Bausteinen auf dieser Infrastruktur.

Häufig laufen Systeme in unterschiedlichen Umgebungen, beispielsweise Entwicklung-/Test- oder Produktionsumgebungen. In solchen Fällen sollten Sie alle relevanten Umgebungen aufzeigen.

Nutzen Sie die Verteilungssicht insbesondere dann, wenn Ihre Software auf mehr als einem Rechner, Prozessor, Server oder Container abläuft oder Sie Ihre Hardware sogar selbst konstruieren.

Aus Softwaresicht genügt es, auf die Aspekte zu achten, die für die Softwareverteilung relevant sind. Insbesondere bei der Hardwareentwicklung kann es notwendig sein, die Infrastruktur mit beliebigen Details zu beschreiben.

Motivation

Software läuft nicht ohne Infrastruktur. Diese zugrundeliegende Infrastruktur beeinflusst Ihr System und/oder querschnittliche Lösungskonzepte, daher müssen Sie diese Infrastruktur kennen.

Form

Das oberste Verteilungsdiagramm könnte bereits in Ihrem technischen Kontext enthalten sein, mit Ihrer Infrastruktur als EINE Blackbox. Jetzt zoomen Sie in diese Infrastruktur mit weiteren Verteilungsdiagrammen hinein:

- Die UML stellt mit Verteilungsdiagrammen (Deployment diagrams) eine Diagrammart zur Verfügung, um diese Sicht auszudrücken. Nutzen Sie diese, evtl. auch geschachtelt, wenn Ihre Verteilungsstruktur es verlangt.
- Falls Ihre Infrastruktur-Stakeholder andere Diagrammarten bevorzugen, die beispielsweise Prozessoren und Kanäle zeigen, sind diese hier ebenfalls einsetzbar.

7.1. Infrastruktur Ebene 1

An dieser Stelle beschreiben Sie (als Kombination von Diagrammen mit Tabellen oder Texten):

- die Verteilung des Gesamtsystems auf mehrere Standorte, Umgebungen, Rechner, Prozessoren o. Ä., sowie die physischen Verbindungskanäle zwischen diesen,
- wichtige Begründungen für diese Verteilungsstruktur,
- Qualitäts- und/oder Leistungsmerkmale dieser Infrastruktur,
- Zuordnung von Softwareartefakten zu Bestandteilen der Infrastruktur

Für mehrere Umgebungen oder alternative Deployments kopieren Sie diesen Teil von arc42 für alle wichtigen Umgebungen/Varianten.

*<**Übersichtsdiagramm**>*

Begründung

<Erläuternder Text>

Qualitäts- und/oder Leistungsmerkmale

<Erläuternder Text>

Zuordnung von Bausteinen zu Infrastruktur

<Beschreibung der Zuordnung>

7.2. Infrastruktur Ebene 2

An dieser Stelle können Sie den inneren Aufbau (einiger) Infrastrukturelemente aus Ebene 1 beschreiben.

Für jedes Infrastrukturelement kopieren Sie die Struktur aus Ebene 1.

7.2.1. *<Infrastrukturelement 1>*

<Diagramm + Erläuterungen>

7.2.2. *<Infrastrukturelement 2>*

<Diagramm + Erläuterungen>

...

7.2.3. *<Infrastrukturelement n>*

<Diagramm + Erläuterungen>

Unabhängig von der Installation, welche implementiert werden soll, muss stets zunächst die Apollo

Version 1.0 installiert werden.

8. Querschnittliche Konzepte

8.1. Common

Dieser Abschnitt beschreibt übergreifende, prinzipielle Regelungen und Lösungsansätze, die an mehreren Stellen relevant sind.

Inhalt

Dieser Abschnitt beschreibt übergreifende, prinzipielle Regelungen und Lösungsansätze, die an mehreren Stellen (=*querschnittlich*) relevant sind.

Solche Konzepte betreffen oft mehrere Bausteine. Dazu können vielerlei Themen gehören, beispielsweise:

- fachliche Modelle,
- eingesetzte Architektur- oder Entwurfsmuster,
- Regeln für den konkreten Einsatz von Technologien,
- prinzipielle — meist technische — Festlegungen übergreifender Art,
- Implementierungsregeln

Motivation

Konzepte bilden die Grundlage für *konzeptionelle Integrität* (Konsistenz, Homogenität) der Architektur und damit eine wesentliche Grundlage für die innere Qualität Ihrer Systeme.

Manche dieser Themen lassen sich nur schwer als Baustein in der Architektur unterbringen (z.B. das Thema „Sicherheit“). Hier ist der Platz im Template, wo Sie derartige Themen geschlossen behandeln können.

Form

Kann vielfältig sein:

- Konzeptpapiere mit beliebiger Gliederung,
- übergreifende Modelle/Szenarien mit Notationen, die Sie auch in den Architektursichten nutzen,
- beispielhafte Implementierung speziell für technische Konzepte,
- Verweise auf „übliche“ Nutzung von Standard-Frameworks (beispielsweise die Nutzung von Hibernate als Object/Relational Mapper).

Struktur

Eine mögliche (nicht aber notwendige!) Untergliederung dieses Abschnittes könnte wie folgt aussehen (wobei die Zuordnung von Themen zu den Gruppen nicht immer eindeutig ist):

- Fachliche Konzepte
- User Experience (UX)
- Sicherheitskonzepte (Safety und Security)
- Architektur- und Entwurfsmuster
- Unter-der-Haube
- Entwicklungskonzepte
- Betriebskonzepte

8.2. *Protobuf-Schnittstelle*

<Erklärung>

8.3. *Funktionale Abgrenzung*

<Erklärung>

8.4. *Sicherheit, Ausnahme- und Fehlerbehandlung*

8.5. *common*

8.6. *guardian*

8.7. *drive*

9. Entwurfsentscheidungen

Ausführung von interessanten Architektur- oder Entwurfsentscheidungen inklusive der jeweiligen Begründung.

9.1. ROS vs CyberRT

Inhalt

Wichtige, teure, große oder riskante Architektur- oder Entwurfsentscheidungen inklusive der jeweiligen Begründungen. Mit "Entscheidungen" meinen wir hier die Auswahl einer von mehreren Alternativen unter vorgegebenen Kriterien.

Wägen Sie ab, inwiefern Sie Entscheidungen hier zentral beschreiben, oder wo eine lokale Beschreibung (z.B. in der Whitebox-Sicht von Bausteinen) sinnvoller ist. Vermeiden Sie Redundanz. Verweisen Sie evtl. auf Abschnitt 4, wo schon grundlegende strategische Entscheidungen beschrieben wurden.

Motivation

Stakeholder des Systems sollten wichtige Entscheidungen verstehen und nachvollziehen können.

Form

Verschiedene Möglichkeiten:

- Liste oder Tabelle, nach Wichtigkeit und Tragweite der Entscheidungen geordnet
- ausführlicher in Form einzelner Unterkapitel je Entscheidung
- ADR (Architecture Decision Record) für jede wichtige Entscheidung

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 -erreichung 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.

12. Glossar

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

12.1. Eistieg

*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.