Developing an Autonomous Vehicle

A Use-Case in Software Engineering

Dr. Michael Aeberhard

February 7, 2022

Outline

Outline

About Me
Education

- Georgia Institute of Technology (2002-2009)
  - B.S. in Computer Engineering (2007)
  - M.S. in Electrical Engineering (2009)
  - Formula Student, BMW Motorsport, BMW Plant Spartanburg
- Centrale Supélec, Master Professionelle (2009)
  - Master Thesis @ Daimler AG - *Processing of Out-of-Sequence Measurements in Tracking for an Automotive Pre-Crash Application*
- Technische Universität Dortmund, Dr.-Ing. (2010-2017)
  - Dissertation: *Object-level fusion for surround environment perception in automated driving applications* [1]
  - In cooperation with BMW AG

About Me

Career

- BMW AG (2010-2018)
  - Research in sensor data fusion
  - Involved in German/EU funded research projects: SmartSenior, UR:BAN, AdaptIVe
  - Developer, Product Owner, Project Lead for autonomous driving
- Autonomous Intelligent Driving GmbH (2018-2020)
  - Developer and Tech Lead for L4 "robot taxi" application
  - Software development for localization and perception
- Apex.AI (2020-)
  - Head of Application Engineering
  - Bringing a safety-certified version of ROS 2 to series production

History of Autonomous Vehicles Development

The Early Years (1990s)

Prometheus Project

[2] VaMP

- European R&D project (1985 - 1995)
- Prof. Ernst Dickmann's research vehicles demonstrated 1,000+ km automated highway drives

[1] https://path.berkeley.edu/research/connected-and-automated-vehicles/national-automated-highway-systems-consortium

History of Autonomous Vehicles Development

The Early Years (1990s)

Prometheus Project

[2] VaMP

- European R&D project (1985 - 1995)
- Prof. Ernst Dickmann's research vehicles demonstrated 1,000+ km automated highway drives

Demo '97

Platoon demo[1]

- US DOT sponsored research to demonstrate the Automated Highway System (AHS)
- A 12 km stretch of an HOV lane was used for the demonstration

[1] https://path.berkeley.edu/research/connected-and-automated-vehicles/national-automated-highway-systems-consortium

History of Autonomous Vehicles Development

The Early Years (1990s)

Prometheus Project

[2] VaMP

- European R&D project (1985 - 1995)
- Prof. Ernst Dickmann's research vehicles demonstrated 1,000+ km automated highway drives

Demo '97

Platoon demo[1]

- US DOT sponsored research to demonstrate the Automated Highway System (AHS)
- A 12 km stretch of an HOV lane was used for the demonstration

These projects set the foundation for OEM ADAS development in the 2000s

[1] https://path.berkeley.edu/research/connected-and-automated-vehicles/national-automated-highway-systems-consortium

History of Autonomous Vehicles Development

The DARPA Challenges (2004-2008)

DARPA Grand Challenges[1]

- In the 2004 competition, 15 teams competed for a $1 million prize on a 240 km desert route
- The prize went unclaimed; the best team was able to complete 11.78 km
- In 2005, a $2 million prize was up for grabs on a 212 km route
- The team from Stanford University with their vehicle *Stanley* completed the course the fastest (in total 5 teams completed the course)

[3]

[1] https://www.darpa.mil/about-us/timeline/-grand-challenge-for-autonomous-vehicles

History of Autonomous Vehicles Development

The DARPA Challenges (2004-2008)

DARPA Urban Challenge[1]

- In 2007, a third competition for a $2 million prize was held on a 96 km urban route
- A total of 11 teams competed in the final competition
- A team from Carnegie Mellon University won with their vehicle *Boss*

[4]

[1] https://www.darpa.mil/about-us/timeline/darpa-urban-challenge

History of Autonomous Vehicles Development

The DARPA Challenges (2004-2008)

DARPA Urban Challenge[1]

- In 2007, a third competition for a $2 million prize was held on a 96 km urban route
- A total of 11 teams competed in the final competition
- A team from Carnegie Mellon University won with their vehicle *Boss*

[4]

The student, engineers, and companies of the DARPA challenges laid the foundations for today's autonomous driving development and are leaders in the industry.

[1] https://www.darpa.mil/about-us/timeline/darpa-urban-challenge

History of Autonomous Vehicles Development

The Industry Ramp-Up Phase (2010-2020)

OEMs and Tier 1s

BMW [5]

Mercedes-Benz [6]

Bosch [7]

Continental [8]

New Players

Waymo [9]

Cruise [10]

Zoox [11]

TuSimple [12]

History of Autonomous Vehicles Development

Going to Production (2020 - ???)

Only since 2020 have the first true driveless vehicles been allowed to operate on public roads, albeit in limited numbers and locations.

Waymo One[1]

Mercedes-Benz DRIVE PILOT [13]

The next decade will be the beginning of the commercialization of autonomous vehicles. Many challenges remain for widespread adoption...

[1] https://waymo.com/waymo-one/

The Software Challenge

Modern vehicles are smartphones on wheels and the next generation vehicles will automated the driving task.
$\Rightarrow$ Software is taking over the automotive industry

- Dozens of ECUs in a modern vehicle
- High-resolution displays for the drive and passenger
- Sensors detect the vehicle's environment
- Actuators automatically control many vehicle functions
- Thousands of software developers from many companies collaborating
- Safety of utmost importance

The Software Challenge

Autonomous Driving

For an autonomous vehicle system, a very high-dimensional input space from environment sensing sensors and vehicle sensors must be translated, **with software**, into steering, brake and motor controls.

Creating an autonomous driving system is an incredibly complex task that even challenges humans.

The Phases of Development

Autonomous vehicles and ADAS development has traditionally followed three phases of development.

- **Research**: Develop prototypes to understand the use-cases, algorithms, hardware, etc. $\rightarrow$ is the product feasible?
- **Pre-development**: Iron out the details, requirements, commercial viability, etc. $\rightarrow$ develop a solid blue print for production
- **Production**: Software and hardware is hardened and optimized for production quality and cost, meeting any required safety, automotive or product standards

The Phases of Development

Autonomous vehicles and ADAS development has traditionally followed three phases of development.

- **Research**: Develop prototypes to understand the use-cases, algorithms, hardware, etc. $\rightarrow$ is the product feasible?
- **Pre-development**: Iron out the details, requirements, commercial viability, etc. $\rightarrow$ develop a solid blue print for production
- **Production**: Software and hardware is hardened and optimized for production quality and cost, meeting any required safety, automotive or product standards

The autonomous driving industry today is making the transition from pre-development to production.

Architecture

The architecture of a system can be broken down into different levels of detail.

- **Functional Architecture**: Logical components and sub-systems, along with a high-level definition of interfaces, are identified
- **Software Architecture**: Implementation of logical components into concrete software components with concrete interface implementations
- **Hardware Architecture**: Partitioning of the software components onto specific hardware and defining the low-level communication mechanisms of the components

The different phases of development will focus on different levels of the architecture and hence specific details of the system.

Outline

Functional Architecture

The functional architecture is initially developed during the research phase and refined as development matures to production.

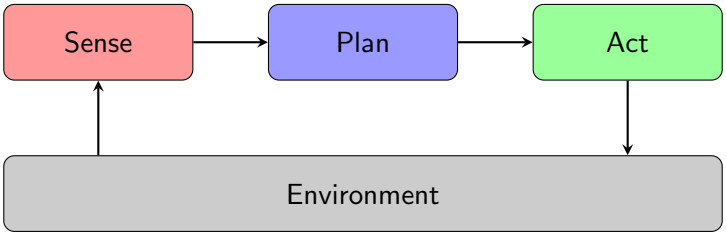
- The main components of a system are identified
- Specific algorithms which realize the components are developed
- High-level interfaces between components are created

The functional architecture for an autonomous driving vehicle has evolved over the past decade and is converging towards an industry standard.

Sense - Plan - Act

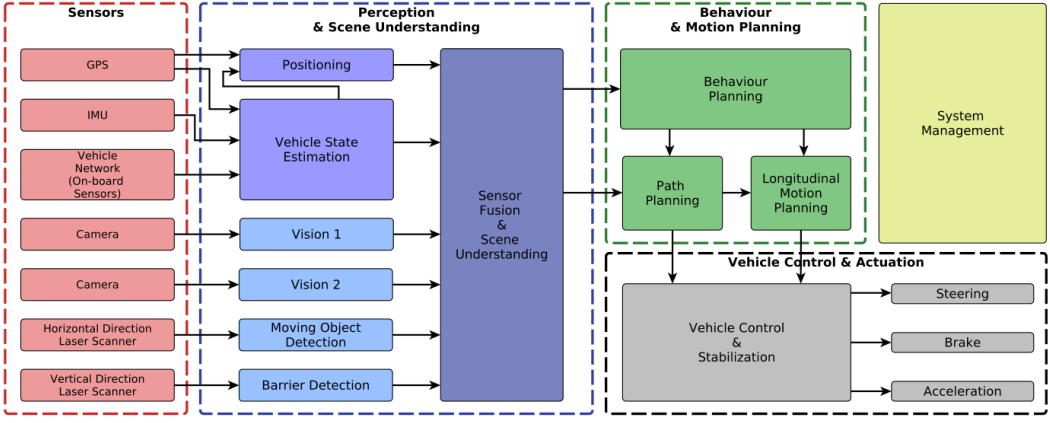
Common Autonomous Vehicle Functional Architecture

The *Sense* - *Plan* - *Act* architecture is still predominant in modern autonomous vehicle architectures.



Sense - Plan - Act

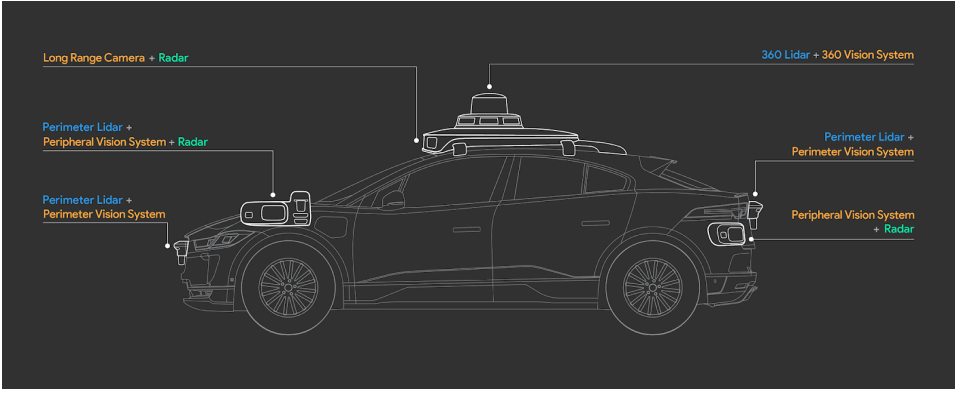
Common Autonomous Vehicle Functional Architecture



Architecture image from [14] based on the vehicle that won the Korean Autonomous Vehicle Competition [15].

Sensors

Autonomous vehicles use a combination of cameras, radar and lidar sensors to sense and perceive the environment.



Waymo sensor suite[1]

[1] https://blog.waymo.com/2020/03/introducing-5th-generation-waymo-driver.html

Sensors
Camera

Source: Daimler[1]

Tesla camera system, Source: YouTube (greentheonly)[2]

Vision sensors are the basis for an autonomous driving system, providing important perception input for many different purposes.

Capabilities:

- Dynamic object detection
- Static object detection
- Lane and road detection
- Classification
- Traffic sign/light detection

Important properties:

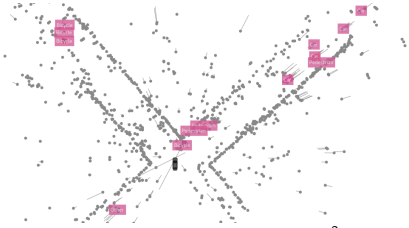
- High dynamic range
- 360 deg field of view
- Global shutter
- Low cost
- No distance or velocity measurements

[1] https://www.daimler.com/magazine/technology-innovation/automation-daimler-immendingen-camera-radar-lidar.html

[2] https://www.youtube.com/watch?v=rACZACXgreQ

Sensors
Radar

Source: Continental[1]



Source: Daimler, RadarScenes dataset[2]

Radar has been the workhorse of ADAS for over two decades, being the main sensor for ACC and continues its importance in autonomous driving systems.

Capabilities:

- Dynamic object detection
- Static object detection
- Road boundary detection

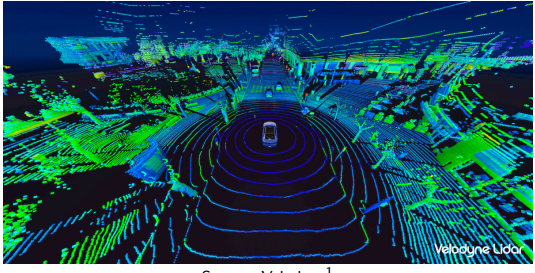
Important properties:

- Robust in weather, e.g. rain, fog, etc.
- Direct measurement of speed
- Reasonable cost
- Low resolution, poor vertical separation

[1] https://www.continental-automotive.com/en-gl/Passenger-Cars/Autonomous-Mobility/Enablers/Radars/Long-Range-Radar/ARS540

[2] https://radar-scenes.com/

Sensors
Lidar



Source: Velodyne[1]

[1] https://velodynelidar.com/media-kit/

Only very recently has lidar been introduced in production vehicles, but it has been a core sensor of autonomous driving research since the DARPA challenges.

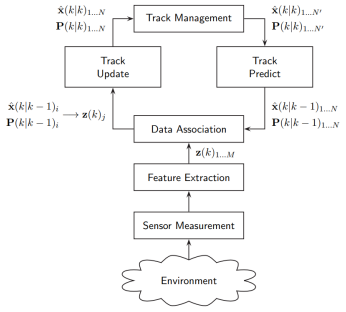
Capabilities:

- Dynamic and static object detection
- Road boundary and curb detection
- Lane detection

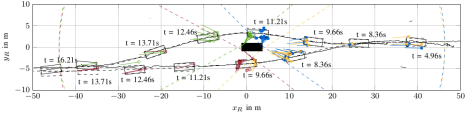
Important properties:

- High resolution
- Large field of view
- Measure intensity
- Poor weather robustness
- No velocity measurement

Perception
Object Detection and Tracking



Common tracking architecture [1]

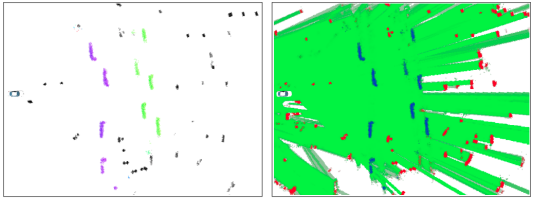


Tracking two objects with radar measurements [16]

Detection of dynamic objects is one of the central components in an AV system, with a very long history.

- Core algorithm is the Kalman filter [17] (and many variations therefore of) to estimate an object's state over time
- Bounding boxes are the most common representation form
- Pre-processing algorithms usually detect objects in a single data frame (clustering, ML-based detection, etc.)

Perception
Occupancy Grids



(a) Particle map. (b) DST map.

(c) Bayes map from DST map. (d) Standard occupancy grid. (e) Camera image.

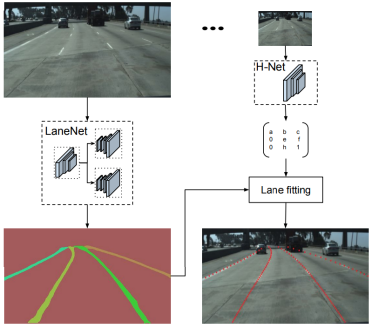
Dynamic occupancy grids [18]

Occupancy grids have been a core algorithm in robotics since the mid-80s [19] and have advanced ever since.

- Environment is rasterized into cells and a probability of occupancy is calculated
- Modern versions also calculate if a cell is dynamic or static or even the class (semantic)
- Free space (cells with no occupancy)
- Can be extended to include height information (elevation maps)
- Base environment representation which can be used by downstream component, e.g. object detection/tracking, localization, free space detection, etc.

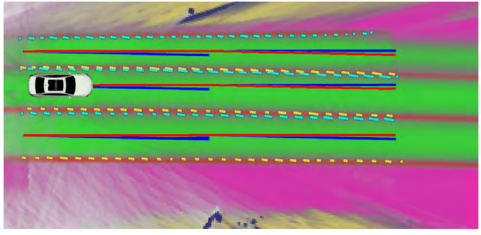
Perception
Lane / Road Detection

Camera based lane detection is the mainstream method, but difficult to obtain correct 3D projection.



DNN approach to lane detection in images [20]

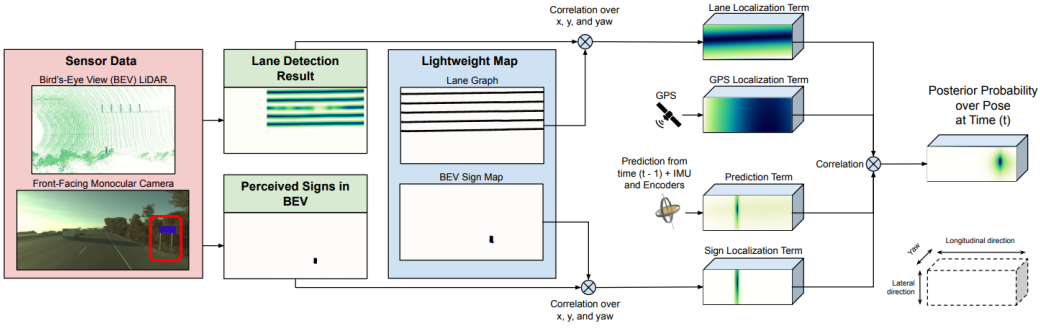
Semantic occupancy grid based approaches enable lane/road detection with several sensors directly in the 3D space.



Grid-based lane detection [21]

Localization

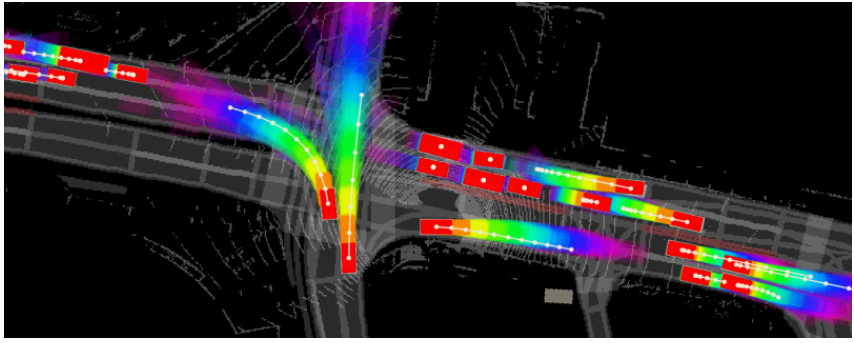
Highly detailed maps for autonomous driving provide vital prior information for navigating complex driving scenarios. Localization uses the sensor data to localize the vehicle within such maps.



Localization on a sparse lane and traffic sign map on highways [22]

Prediction

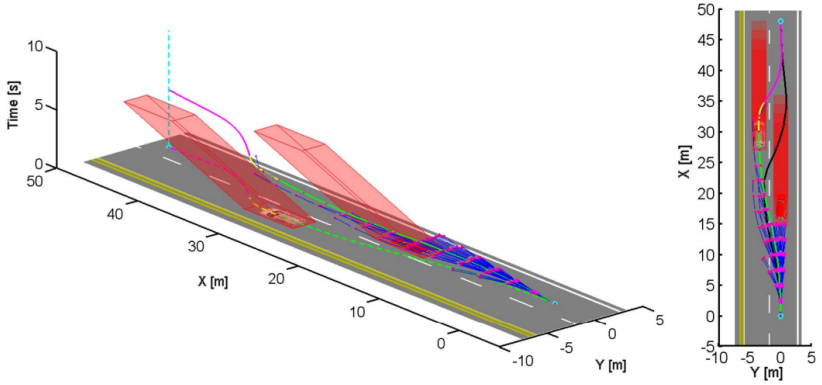
Predicting the future trajectory of traffic is one of the core challenges in advancing the performance of autonomous vehicles.



Heat map of future locations of traffic over time (white line is actual trajectory) [23]

Motion Planning

Based on the perception, localization and prediction input, motion planning plans a safe trajectory for the vehicle to follow. Sampling-based and optimization methods are the most common approaches.

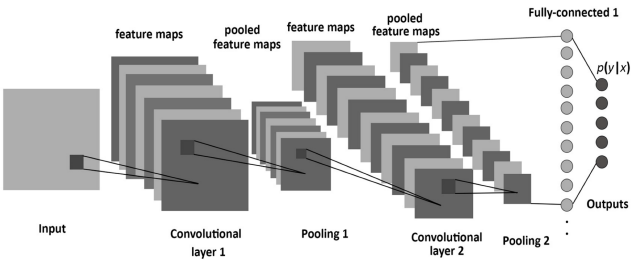


Motion planning in a highway scenario [24]

Machine Learning in Autonomous Driving

Recent advances in machine learning, in particular deep neural networks, have affected the algorithmic approach in almost all components in an AV stack.

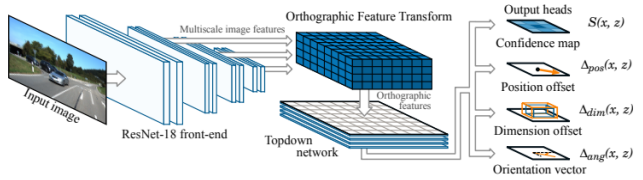
- Algorithms are not designed, but models are rather learned from (labeled) data
- Performance of certain tasks, in particular with computer vision, have made major leaps forward
- Requires large amount of data
- Requires specialized hardware for efficient, real-time processing

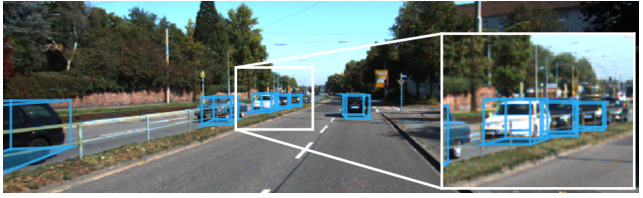


Convolutional neural network architecture for image classification [25]

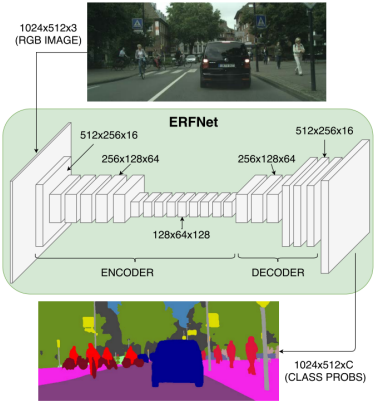
Machine Learning in Autonomous Driving

Examples for camera sensors





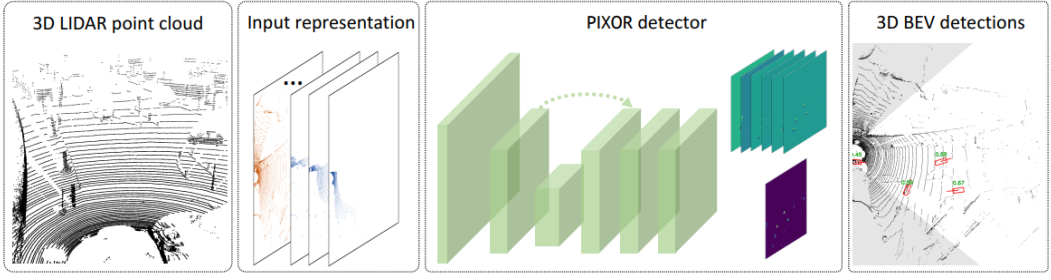
3D camera object detection [26]



Semantic segmentation [27]

Machine Learning in Autonomous Driving

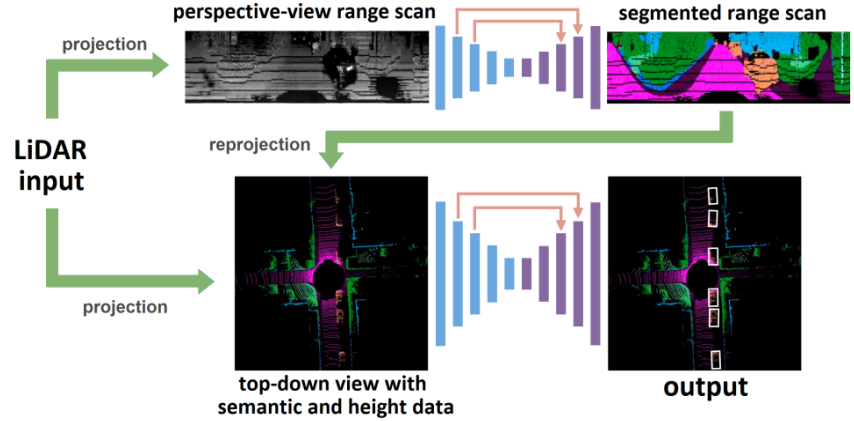
Examples for lidar sensors



3D lidar object detection [28]

Machine Learning in Autonomous Driving

Examples for lidar sensors

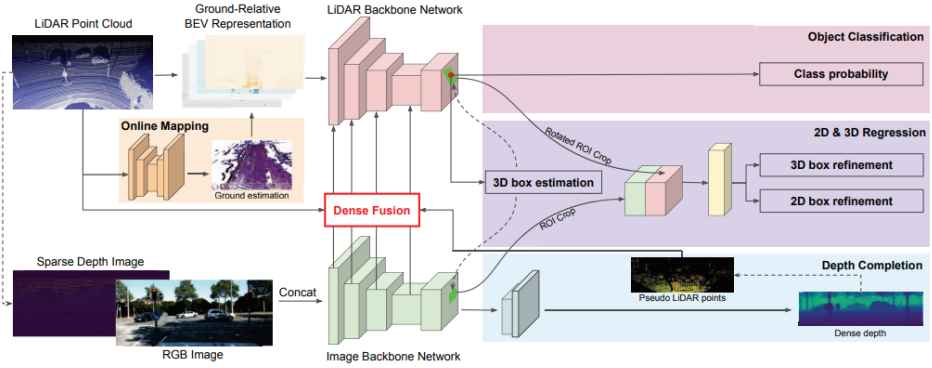


Lidar semantic segmentation [29]

Machine Learning in Autonomous Driving

Examples for sensor data fusion

Deep neural networks can take several sensors as input to realize a learned approach to sensor data fusion.

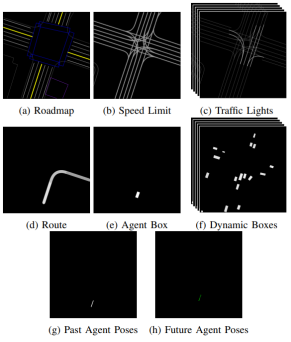


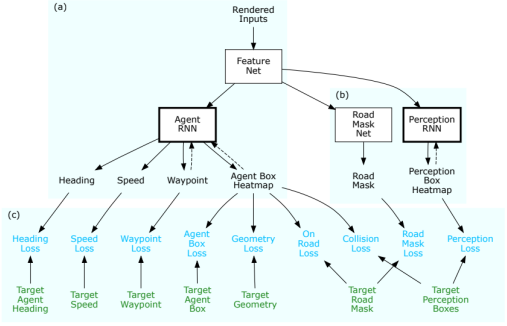
DNN with combined training of lidar and camera [30]

Machine Learning in Autonomous Driving

Example for planning components

ChauffeurNet is an example of deep neural networks directly driving the vehicle and learning the tasks of prediction and planning.

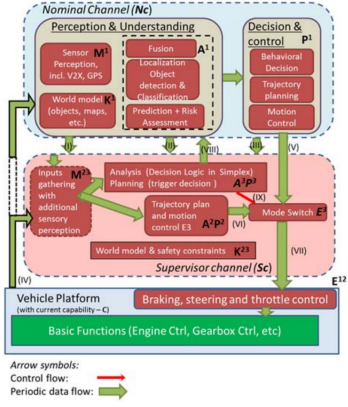




ChauffeurNet input images and training architecture [31]

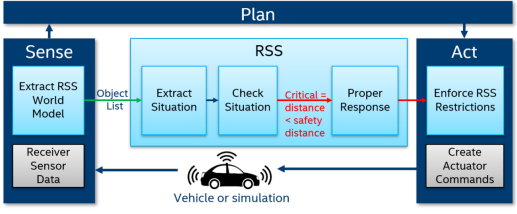
Architecture for Safety

A key challenge in autonomous driving is designing a functional architecture which is robust to failures.



Redundant supervisor channel [32]

- Some form of functional redundancy is required
- The redundant system is usually simpler and focuses only on safety goals, e.g. collision avoidance
- Overrides main system or applies constraints



Responsibility Sensitive Safety from Intel/MobilEye [33]

Outline

Software Architecture

The implementation of a functional architecture must consider several layers of software architecture, where many implementation details affect the performance of the system.

Applications	Implementations of the functional application code into individual components *Examples: Object detection, Localization, Planning, etc.*
Framework	Base software with common functions and utilities for system integration *Examples: ROS, AUTOSAR Adaptive, Qt, NodeJS, etc.*
Data Transport	Communication mechanism between processes, ECUs, cloud, etc. *Examples: DDS, SHMEM, SOME/IP, MQTT, Websockets, Protobuf, etc.*
Operating System	Kernel, scheduler, drivers, etc. to manage/interface hardware with software *Examples: Linux, QNX, INTEGRITY, PikeOS, FreeRTOS, etc.*
Hardware	Physical compute board with SOCs, CPUs, MCUs, GPUs, DSPs, etc. *Examples: Intel, NVIDIA, Renesas, TI, Qualcomm, etc.*

Software Frameworks for Autonomous Driving

Framework requirements

What are some of the functions, utilities and APIs that an application requires?

- Concept of partitioning a larger system into components
- Means of receiving and transmitting data between components
- Configuration / parametrization of components
- Management of component execution und underlying CPU resources, e.g. threads
- Tools for development, debugging, and deployment
- Compatible with different programming languages and platforms

Software Frameworks for Autonomous Driving

Today's most popular solutions

Robot Operating System[1]

MATLAB/Simulink[2]

EB Assist ADTF[3]

AUTOSAR[4]

NVIDIA DriveWorks[5]

TTTech Auto MotionWise[6]

[1] https://ros.org/
[2] https://www.mathworks.com/products/automated-driving.html
[3] https://www.elektrobit.com/products/automated-driving/eb-assist/adtf/
[4] https://www.autosar.org/
[5] https://developer.nvidia.com/drive/driveworks
[6] https://www.tttech-auto.com/products/safety-software-platform/motionwise/

Software Frameworks for Autonomous Driving

Today's most popular solutions

Apex.AI

Apex.AI[1] has developed Apex.OS[2] and Apex.Middleware[3], a software framework and middleware based on ROS and designed for safety-certified automotive applications.

[1] https://www.apex.ai/
[2] https://www.apex.ai/apex-os
[3] https://www.apex.ai/apex-middleware

Robot Operating System (ROS)

Overview

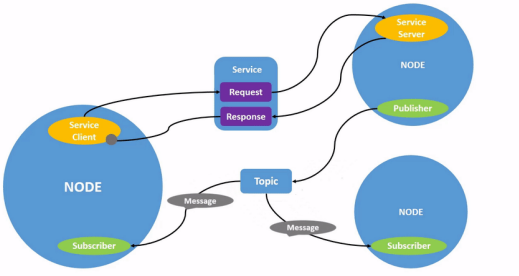
- Most popular framework for developing robotics (and other) applications
- Open source with thousands of contributors world wide
- Widely adopted by universities, research institutes and and companies
- Thousands of packages[1] (applications) for sensors, perception, SLAM, motion planning, etc.
- Powerful suite of tools (visualization, introspection, debugging, etc.)
- Wide support for different programming languages
- Hundreds of autonomous vehicles in R&D run on ROS

[1] https://index.ros.org/packages/

Robot Operating System (ROS)

The basics

- **Nodes**: A single processing entity with inputs and outputs
- **Topics**: Data channels between nodes
- **Messages**: The data type which is transmitted on a topic
- **Publisher**: Within a node, the entity that transmits data on a topic
- **Subscriber**: Within a node, the entity that receives data on a topic
- **Service**: Request/response mechanism between nodes



Source: ROS 2 Tutorials - Understanding ROS 2 nodes[1]

[1] https://docs.ros.org/en/galactic/Tutorials/Understanding-ROS2-Nodes.html

Robot Operating System (ROS)

Example ROS 2 node in C++

```cpp
// From https://docs.ros.org/en/galactic/Tutorials/Writing-A-Simple-Cpp-Publisher-And-Subscriber.html
class MinimalPublisher : public rclcpp::Node {
    public:
        MinimalPublisher() : Node("minimal_publisher"), count_(0) {
            publisher_ = this->create_publisher<std_msgs::msg::String>("topic", 10);
            timer_ = this->create_wall_timer(500ms, std::bind(&MinimalPublisher::timer_callback, this));
        }
    private:
        void timer_callback() {
            auto message = std_msgs::msg::String();
            message.data = "Hello, world! " + std::to_string(count_++);
            RCLCPP_INFO(this->get_logger(), "Publishing: '%s'", message.data.c_str());
            publisher_->publish(message);
        }
        rclcpp::TimerBase::SharedPtr timer_;
        rclcpp::Publisher<std_msgs::msg::String>::SharedPtr publisher_;
        size_t count_;
};

int main(int argc, char * argv[]) {
    rclcpp::init(argc, argv);
    rclcpp::spin(std::make_shared<MinimalPublisher>());
    rclcpp::shutdown();
    return 0;
}
```

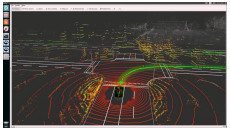
Robot Operating System (ROS)

Tools

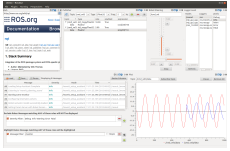
ROS has a rich suite of tools, both included in the standard installation as well as tools developed by the community.

- Command line tools for system introspection, configuration, and launching
- rosbag[1] recording and replay tool
- RViz[2]: 3D Visualizer
- rqt[3]: Qt based GUI with customizable plug-ins
- Robot Web Tools[4]: web based interfacing with ROS
- PlotJuggler[5]: plotting of ROS message data

T



RViz[2] 3D Visualizer
Source: YouTube (Autoware)[6]



rqt[3] GUI

[1] https://github.com/ros2/rosbag2
[2] https://github.com/ros2/rviz
[3] http://wiki.ros.org/rqt
[4] http://robotwebtools.org/
[5] https://www.plotjuggler.io/
[6] https://www.youtube.com/watch?v=zujGfJcZCpQ

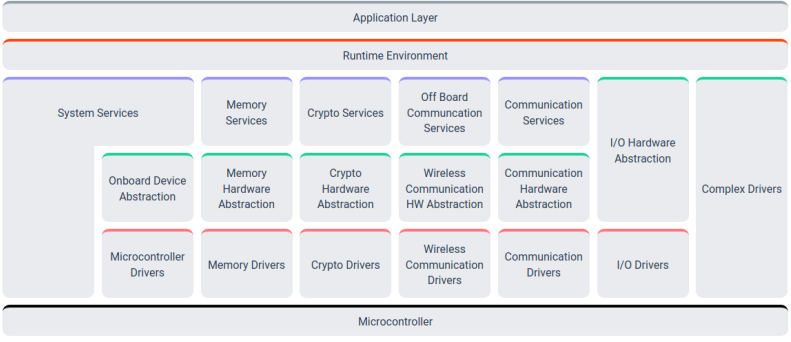
Production Software Frameworks

A software framework and middleware for a production vehicle has many additional requirements that need to be met.

- Safety requirements and standards must be met
- Software must run on embedded hardware with limited resources
- Automotive interfaces must be supported (CAN, LIN, automotive Ethernet, etc.)
- Security requirements must be met, e.g. end-to-end protection, identity access management, etc.
- Automotive interfaces such as Unified Diagnostics Service (UDS) must be implemented

AUTOSAR Classic

AUTOSAR was formed in 2003 to standardize automotive E/E architecture. Applications can be developed against a standard Runtime Environment for automotive *microcontrollers*.

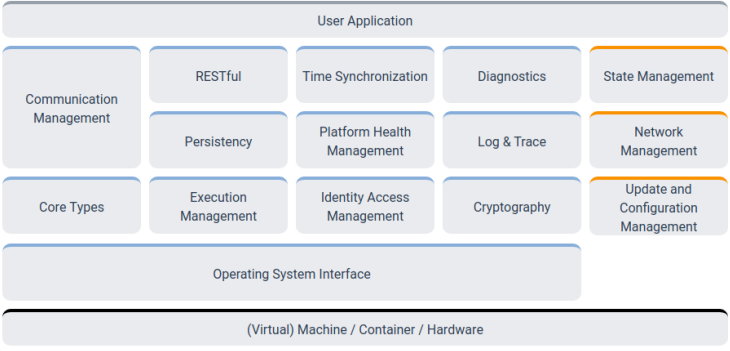


Source: AUTOSAR Classic Platform[1]

[1] https://www.autosar.org/standards/classic-platform/

AUTOSAR Adaptive

Next-generation of AUTOSAR designed for high performance computing hardware running with a POSIX based operating system. Uses a service oriented architecture.

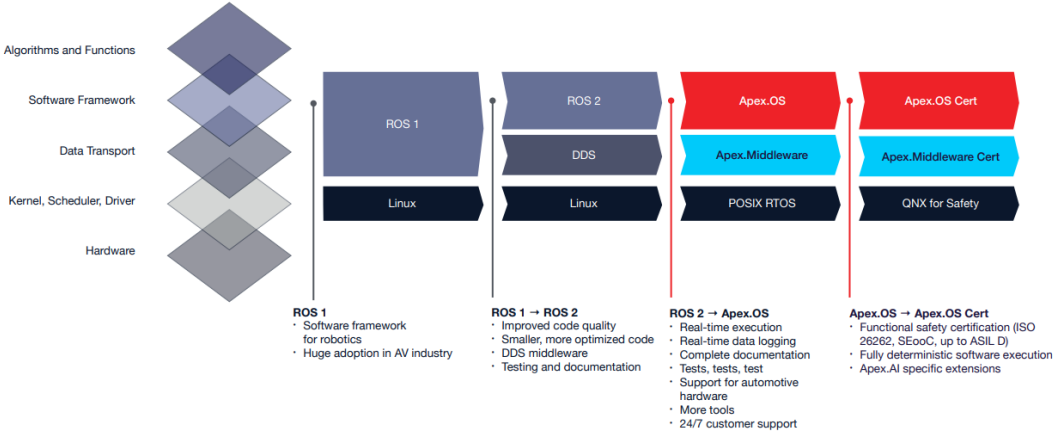


Source: AUTOSAR Classic Platform[1]

[1] https://www.autosar.org/standards/adaptive-platform/

Apex.OS

Apex.OS is a production framework and SDK based on ROS 2.



Source: Apex.AI

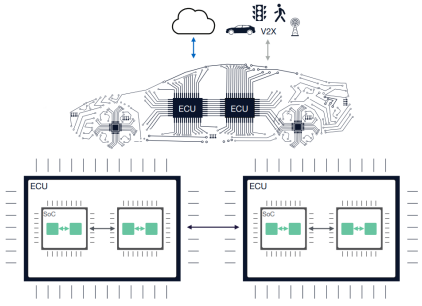
Data Transport Middlewares

Overview

A complete vehicle system requires many different mechanism for transporting data from one component to another.

Middleware requirements:

- Handle GByte/s data transfer (e.g. camera images)
- Short latencies and low run-time resource consumption
- Many-to-many communication
- Interfacing with diverse set of protocols



Source: Apex.AI

Data Transport Middlewares

Automotive communication buses

Automotive communication mechanisms are designed to be robust and reliable where several ECUs can exchange data.

- CAN: Reliable bus system to exchange small bits of data across many ECUs
- FlexRay: Improved performance for safety-critical systems
- LIN: Integration of simple sensors and actuators
- MOST: Designed for the transmission of multimedia data, e.g. video, audio, etc.

For each of these technologies, a middleware software stack is required to make the data on the bus system available to an application. In the automotive industry, this is often accomplished with the AUTOSAR Classic stack.

Data Transport Middlewares

Ethernet

In recent years, automotive Ethernet has made it into production vehicles, enabling the high-bandwidth data rates of Ethernet with the well-known TCP and UDP networking protocols. Two popular software middlewares are built on Ethernet.

- DDS[1] is an OMG standard
- Publisher/subscriber on topics
- Standardized interface, OMG IDL[2]
- Rich set quality of service settings
- Used in many industries: automotive, robotics, banking, medical, aerospace

SOME/IP[3]

- Standardized within the scope of AUTOSAR [34]
- Service oriented architecture
- Interface definition with ARXML (from AUTOSAR)
- No quality of service
- Mainly used in automotive

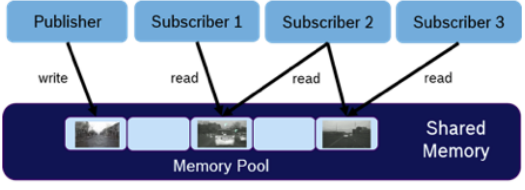
[1] https://www.dds-foundation.org/
[2] https://www.omg.org/spec/IDL/4.2/About-IDL/
[3] https://some-ip.com/

Data Transport Middlewares

Shared memory

For data transmission of high-bandwidth data (e.g. camera data) on the same ECU, the copying, serialization, and deserialization of the data consumes significant CPU resources.

An efficient shared memory data transport is required to reduce the CPU overhead. The Eclipse iceoryx^{TM1} shared memory middleware enables constant-time true zero-copy data transport.



Source: Eclipse Foundation[1]

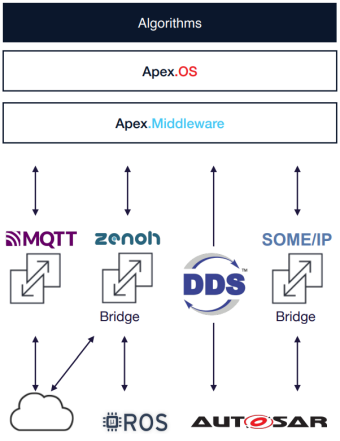
[1] https://www.eclipse.org/community/eclipse_newsletter/2019/december/4.php

Data Transport Middlewares

Apex.Middleware

A single middleware designed to communicate using different protocols in the context of automotive and robotics applications.

- Shared memory backbone with Eclipse iceoryx™[1]
- Ethernet communication with Eclipse Cyclone DDS™[2] and automotive SOME/IP
- Support for PDU communication, e.g. CAN
- MQTT for cloud connectivity



Source: Apex.AI

[1] https://iceoryx.io/

[2] https://cyclonedds.io/

Operating Systems

The operating system is responsible for managing the interface between software applications and the underlying hardware: memory management, process/task management, input/output communications, etc.

- Modern high-performance computers run a POSIX compliant (real-time) operating system (Linux, QNX®, INTEGRITY, PikeOS, etc.)
- AUTOSAR Classic OSEK [35] dominates the automotive microcontrollers
- A Hypervisor can be used to manage several guest operating systems sharing the resources of a single CPU
  - Partitioning of safety and non-safety applications
- Development is typically on Linux or Windows

Source: Wikipedia[1]

Source: Blackberry QNX®[2]

[1] https://en.wikipedia.org/wiki/Linux

[2] https://blackberry.qnx.com/en/products/safety-certified/qnx-os-for-safety

Outline

Hardware Architecture

The hardware used in an autonomous vehicle project evolves with the different stages of development.

- **Research**: Off-the-shelf hardware (x86 PCs, network switches, etc.) are used in early stages of development in order to focus on the functional software.
- **Pre-development**: Evaluation boards from chip vendors, or ruggedized version from third parties, are often used in pre-development stages to begin the work of optimizing the software for an embedded environment.
- **Production**: Hardware is produced in cooperation with a Tier 1 supplier (e.g. Continental, ZF, Bosch, etc.), where several samples (A-sample to D-sample) are produced before the final version is made for start of production (SOP).

Hardware Architecture
Early prototyping

Example of an early prototype with off-the-shelf hardware, but also prototype hardware from automotive vendors (VIGEM, Elektrobit seen here).

Source: BMW[1]

[1] https://www.press.bmwgroup.com/global/article/detail/T0320230EN/nextgen-2020

Hardware Architecture

First embedded hardware

Developer kits from chip vendors serve as a good platform for doing an initial integration of a system to an embedded environment and optimizing the application.

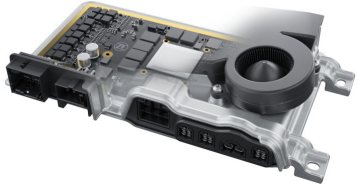
Source: NVIDIA[1]

[1] https://developer.nvidia.com/drive/drive-agx

Hardware Architecture
Production hardware

Tier 1 suppliers develop rugged and automotive grade ECUs for mass production, often customized for an OEM's specific requirements.

Continental ADAS/AD ECU[1]



ZF ProAI[2]

[1] https://press.zf.com/press/en/releases/release_28483.html

[2] https://www.continental-automotive.com/en-gl/Passenger-Cars/Autonomous-Mobility/Enablers/Control-Units/Assisted-Automated-Driving-Control-Unit

Hardware Architecture

Designing for safety and performance

A single, modern ECU designed to be a central domain controller for a vehicle may contain several different processing units to achieve the overall goals of safety and performance.

- CPUs at the core of the computing, mostly ARM-based architectures
- Hardware accelerators for specific tasks: GPUs, DSPs, NPUs, etc.
- Customized computing: ASICs, FPGAs
- Safety-critical microcontrollers
- SPI, Ethernet, PCI Express, hardware shared memory, etc. for communication between computing cores
- SOCs may combines several types of computing cores onto a single chip

Outline

Software Engineering

A core challenge for bringing autonomous driving to market is the sheer scale of the engineering effort required.

- Software must be developed that will scale in terms of complexity and the number of developers working on it
- Integration, testing and deployment must be efficient, fast and reliable
- A large number of engineers must be coordinated to work towards a single, common goal
- Product and development process must comply to a wide range of industry standards

Software Engineering

A core challenge for bringing autonomous driving to market is the sheer scale of the engineering effort required.

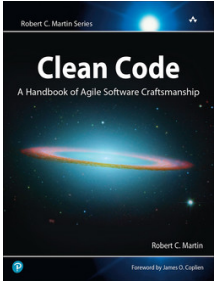
- Software must be developed that will scale in terms of complexity and the number of developers working on it
- Integration, testing and deployment must be efficient, fast and reliable
- A large number of engineers must be coordinated to work towards a single, common goal
- Product and development process must comply to a wide range of industry standards

These aspects of software engineering are often the majority of the effort required for developing a product $\rightarrow$ the functional application may be simple in comparison.

Software Best Practices

It is important that the code you write can be easily understood by someone you may never meet, several years later, long after you've left the project or company.

- Practice clean code
  - Use human-understandable and meaningful variable/function/class names
  - Keep method/function size small
  - The code *is* the documentation
- Follow the SOLID and DRY principles
- Minimize technical debt
- Code reviews are a must
- Use coding guidelines and enforce them with linters



Clean Code by Robert C. Martin[1]

[1] https://www.oreilly.com/library/view/clean-code-a/9780136083238/

Continuous Integration / Delivery / Deployment

Modern software engineering is not feasible without continuous integration and continuous deployment.

Continuous Integration: Automated building, testing, and integration with every change in the software.

Continuous Delivery: Release artifacts are created and automatically tested in pre-production environments. Deployment into the field is done manually.

Continuous Deployment: Ability to deploy software into the field automatically.

Continuous Integration / Delivery / Deployment

Modern software engineering is not feasible without continuous integration and continuous deployment.

Continuous Integration: Automated building, testing, and integration with every change in the software.

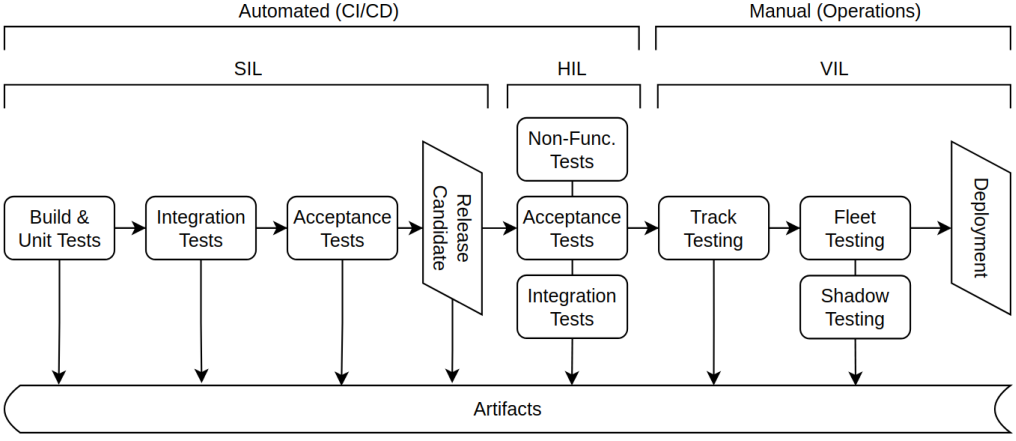
Continuous Delivery: Release artifacts are created and automatically tested in pre-production environments. Deployment into the field is done manually.

Continuous Deployment: Ability to deploy software into the field automatically.

What does CI/CD looks like in practice for an autonomous vehicles?

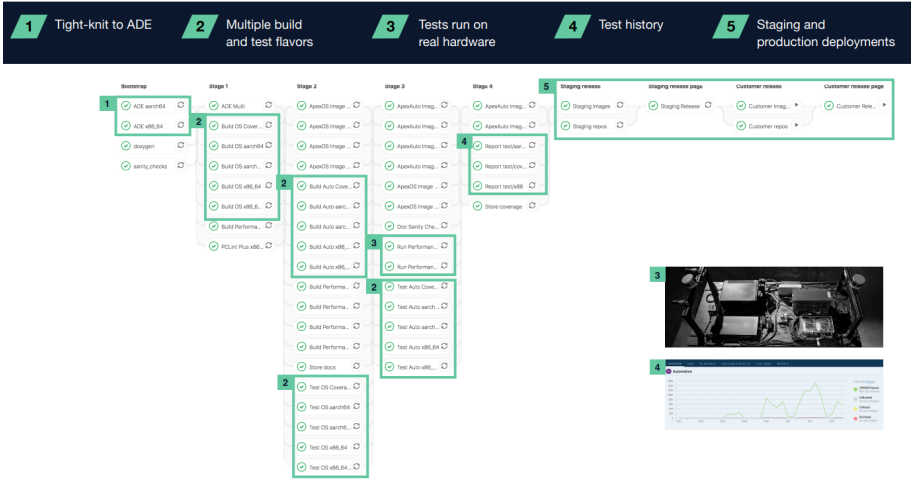
Continuous Integration / Deployment

Pipeline for a fleet of autonomous vehicles



Continuous Integration / Deployment

Example CI/CD pipeline from Apex.AI



Source: Apex.AI

Testing

Software testing is a vital aspect of successful software development, in particular for a safety-relevant application such as autonomous vehicles.

- Treat test code to the same standards as production code
- The amount of test code will likely be multiple factors larger than the actual production application code
- Tests should be automated, repeatable and independent
- Test the behavior, not the implementation
- Follow Arrange - Act - Assert pattern
- The scope of the system under test should be increased with every stage of a CI/CD pipeline

Unit Testing

Testing of the smallest, simplest entities, e.g. classes, functions, etc.

- Tests are lightning fast
- Number of tests is incredibly large
- Tests have no external dependencies (e.g. to file system, etc.)
- Test expected failure cases, e.g. invalid input
- GoogleTest[1] for C++

```
TEST(VehicleSpeedChecks, IsVehicleOverSpeedLimit)
{
    // Arrange
    constexpr float vehicle_speed_kmh = 120;
    constexpr float speed_limit_kmh = 100;

    // Act
    auto speeding =
        isVehicleSpeeding(vehicle_speed_kmh, speed_limit_kmh);

    // Assert
    EXPECT_TRUE(speeding);
}
```

Example for autonomous vehicles

Individual functions from a Kalman filter library, e.g. predict(), are tested.

[1] https://github.com/google/googletest

Integration Testing

Multiple components of a system are tested together.

- Black-box testing (implementation details of components are not known)
- Tests interfaces and interactions between components
- Input is more complex
- Tests may run longer and take more resources
- May interact with external dependencies, e.g. file system
- In ROS, the launch_testing[1] framework can be used to do integration testing on a system of nodes

Example for autonomous vehicles

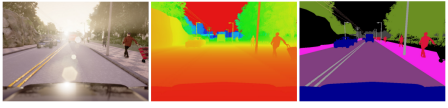
A lidar perception pipeline is tested, e.g. the components of clustering and tracking are tested together.

[1] https://github.com/ros2/launch/tree/master/launch_testing

Acceptance Testing

Testing the overall product behavior based on functional and business requirements.

- Validates the product requirements
- Requires interaction between product/business and engineering
- Simulations or recorded data can be used as input
- Given - When - Then pattern
- Cucumber[1] framework
- Often called *Scenario-based Testing* in the autonomous vehicles space



CARLA open source simulator for autonomous driving [36]

Example for autonomous vehicles

The vehicle behaves correctly when approaching a red light.

[1] https://cucumber.io/

Manual Acceptance Testing

Some acceptance testing is impossible, or very difficult, to automate, which still requires some testing to be done manually.

- Interaction with users is required, e.g. user acceptance testing
- System under test is increased to include artifacts of the physical, which cannot be automated in computer systems

Mcity Test Facility at University of Michigan[1]

Example for autonomous vehicles

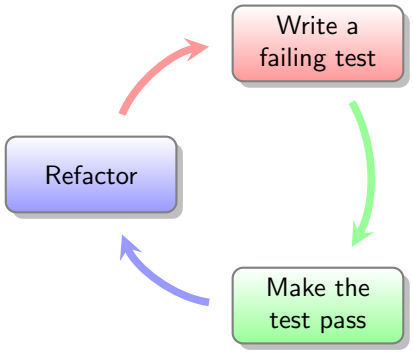
In-vehicle testing with a safety-driver, either on the test track or with a fleet on public roads.

[1] https://mcity.umich.edu/our-work/mcity-test-facility/

Test-Driven Development

Methodology for ensuring all written code is thoroughly tested.

- Tests are written first!
- Forces developer to think about the behavior of the code they are about to write
- Improved API design, since tests are written from the perspective of the user
- Leads to improved software architecture with reduced dependencies
- CI can be utilized from the beginning to protect the code



Development Process

An overview

It is inevitable that once a project becomes larger than a few core developers, a development process will emerge such that everyone involved is able move towards a common goal and "speak the same language".

A development process divides the work into separate steps or processes which are to be followed and which have defined artifacts, outcomes, expectations, etc.

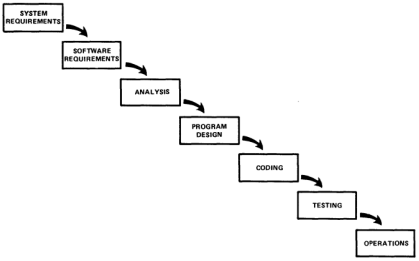
Let's review three common development processes in the automotive industry:

- Waterfall
- V-Model
- Agile

Waterfall

The waterfall approach is the most extreme interpretation of an ideal development process, where each phase of development follows a strict linear progression.

- Assumes that requirements, architecture design, etc. are correct and fully known before development begins
- Testing is the last step in the process where major issues could be uncovered
- The automotive industry, as a traditional mechanical engineering discipline, has had a tendency to apply this methodology, despite its pitfalls

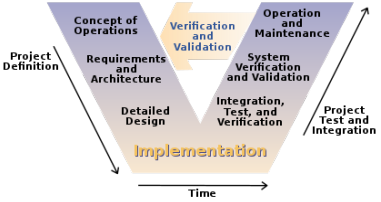


First figure of Royce's 1970 paper [37] (which is not what the paper proposes)

V-Model

The V-Model development process has been the mainstream process in the automotive industry, in particular in Germany. The process is often mandated by standards, contract obligations, etc.

- Originally developed in the defense industry
- Validation and Verification must prove the requirements and design
- Often interpreted and therefore implemented with a time axis at the bottom of the V



Source: Wikipedia[1]

Similar issues as in the waterfall process may raise: project definition assumptions and design may be invalidated during testing and risk that the project does not deliver on time.

[1] https://en.wikipedia.org/wiki/V-Model

Agile

In 2001, several a group of software developers formulated their frustrations with current state of software development, and came up with the *Agile Manifesto*[1].

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

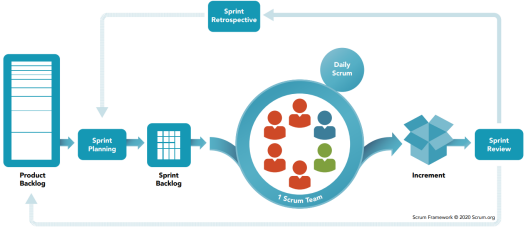
The 12 principles they formulated form the basis of agile and the many related methodologies and processes that are practiced today.

[1] https://agilemanifesto.org/

Scrum

A development process designed with the agile manifesto in mind, created by one of its authors, Jeff Sutherland.

- Short iterations (2-4 weeks) of focused and committed development to finish new features
- Work is visible and prioritized via the product backlog
- Well-defined roles and "rituals" within a Scrum team

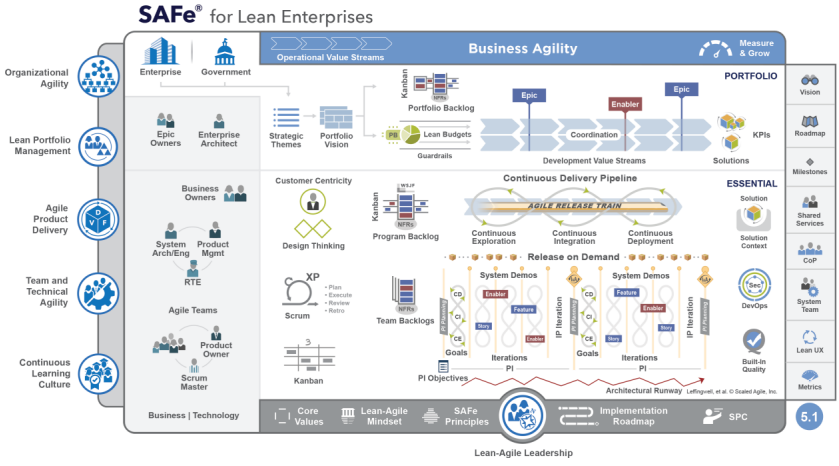


Source: Scrum.org[1]

This work great for one team, but what if you have an organization of hundreds, or thousands of developers?

[1] https://www.scrum.org/

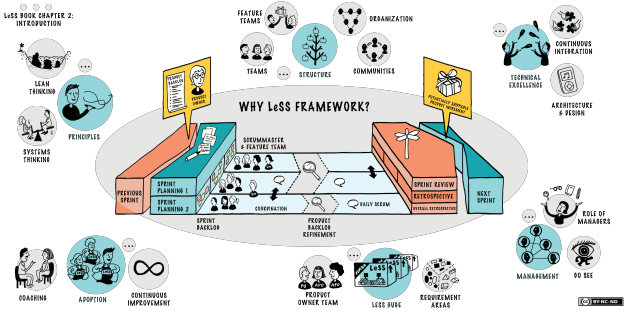
SAFe - Scaled Agile Framework



Source: Scaled Agile Framework[1]

[1] https://www.scaledagileframework.com/

LeSS - Large-Scale Scrum



Source: Large-Scale Scrum (LeSS)[1]

LeSS Huge was adopted by BMW Group's Autonomous Driving division.[2]

[1] https://less.works/

[2] https://less.works/case-studies/bmw-group-autonomous-driving

Agile
Some thoughts...

Despite the craze/hype with agile in today's software engineering industry, one should step back and reflect...

- Not everything has to be black-and-white and follow a known process → each company (and its people) and product are different, so do what works for you
- Most development processes have the same intention: deliver high-quality, well-tested, and on-time software that bring value to the customer
- Often the same technical development practices are encouraged → best to get those right
- Don't forget the V-Model → why not combine it with agile?

Standards

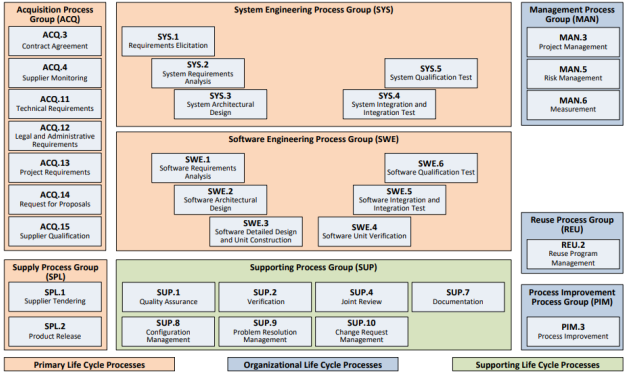
Developing a commercial product, in particular a safety-critical product such as autonomous driving, requires the compliance to industry best-practices, often defined by industry standards and guidelines.

The automotive industry is no exception, where many standards influence the daily work of a software developer.

ASPICE

Defines a software development process to be followed in the automotive industry.

- Based on the V-Model
- OEMs often require compliance for their suppliers
- Applies also to non-safety software
- Defines non-technical work, such as supplier contracting



Source: Automotive SPICE® Process Reference and Assessment Model[1]

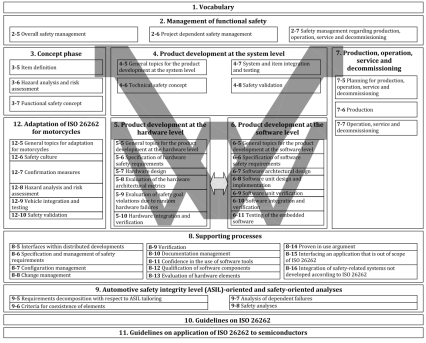
[1] https://automotivespice.com/

Standards for Safety

ISO 26262 - Functional Safety for Road Vehicles

ISO 26262 defines a development standard and practices for functional safety applications implemented in hardware (electrical) and software.

- Designed to protect against faults in E/E system
- Analysis of a system from a safety perspective
- Derive risks, safety goals, safety mechanisms
- Safety integrity levels (ASIL) are defined, which impose requirements on the process and implementation
- Example result: monitoring mechanisms, testing, system redundancy, etc.



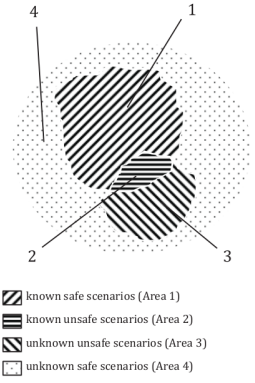
Source: ISO 26262-1:2018 [38]

Standards for Safety

ISO 21448 - Safety of the Intended Functionality (SOTIF)

Supplemental standard to ISO 26262 designed to protect against performance limitations of the system or intentional misuse.

- Definition of use-cases and scenarios to guide the development of relevant testing procedures
- Goal is to reduce the space of known/unknown unsafe scenarios
- Proposes methods for verification of SOTIF
- Annex B shows an example for mitigating false alarm rate in AEB systems
- Example result: scenarios for acceptance testing (automated and manual) are defined, found, and verified



Source: ISO/PAS 21448:2019 [39]

Coding Guidelines

Coding guidelines are used to impose restrictions on programming languages such that common errors or language misuse risks are mitigated.

- Static analysis tools like axivion[1] are available to automatically validate code against such guidelines
- AUTOSAR C++ 14 [40] and MISRA C++ [41] are common guidelines used for the C++ language

Examples from AUTOSAR C++ 14 [40]:

Rule M0-1-3: A project shall not contain unused variables.

Rule A12-6-1: All class data members that are initialized by the constructor shall be initialized using member initializers.

Rule A18-0-2: The library functions atof, atoi and atol from library ¡cstdlib¿ shall not be used.

Rule A18-5-2: Operators new and delete shall not be called explicitly.

[1] https://www.axivion.com/en/products/coding-guidelines/autosar-c14-check/

Autonomous Vehicle Ecosystem Landscape

Source: Orsay Consulting[1]

Autonomous vehicles and software engineering are taking over the automotive industry.

- Many companies, big and small, are providing solutions across the whole stack
- In-vehicle, cloud, and supporting infrastructure software need to seamlessly work together
- Efficient development tools are required

Many opportunities for aspiring software engineers!

[1] https://www.orsayconsulting.net/mobility-resources-en

How to Get Started on Your Own

It is easy to get started with autonomous driving! Check out some of the resources below.

- ROS 2 Tutorials: `https://docs.ros.org/en/galactic/Tutorials.html`
- Autoware.Auto: `https://gitlab.com/autowarefoundation/autoware.auto`
- CARLA Simulator: `https://carla.org/`
- Datasets:
  - NuScenes: `https://www.nuscenes.org/`
  - Waymo Open Dataset: `https://waymo.com/open/`
  - KITTI: `http://www.cvlibs.net/datasets/kitti/`
- NVIDIA Jetson Nano: `https://developer.nvidia.com/embedded/jetson-nano-developer-kit`
- RC Cars:
  - F1TENTH RC Car: `https://f1tenth.org/`
  - TUM RC Car: `https://www.mos.ed.tum.de/en/ftm/main-research/intelligent-vehicle-systems/autonomous-rc-cars/`

References I

[1] M. Aeberhard, "Object-Level fusion for surround environment perception in automated driving applications," PhD thesis, Technical University of Dortmund, May 2017.

[2] E. Dickmanns *et al.*, "The Seeing Passenger Car VaMoRs-P," in *IEEE Intelligent Vehicles Symposium*, Paris, France, Oct. 1994, pp. 68–73.

[3] S. Thrun *et al.*, "Stanley: The Robot that Won the DARPA Grand Challenge," *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[4] C. Urmson *et al.*, "Autonomous Driving in Urban Environments: Boss and the Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, Aug. 2008.

[5] M. Aeberhard *et al.*, "Experience, Results and Lessons Learned from Automated Driving on Germany's Highways," *IEEE Intelligent Transportation Systems Magazine*, vol. 7, no. 1, pp. 42–57, Jan. 2015.

[6] J. Ziegler *et al.*, "Making Bertha Drive - An Autonomous Journey on a Historic Route," *IEEE Intelligent Transportation Systems Magazine*, vol. 6, no. 2, pp. 8–20, 2014.

[7] Bosch Group. (Sep. 2015), Cars that Drive Themselves - Highway Pilot Technically Viable in Five Years, [Online]. Available: `http://www.bosch-presse.de/`.

[8] Continental. (Apr. 2018), Continental Expands Automated Driving Tests on the Autobahn, [Online]. Available: `https://www.continental.com/en/press/press-releases/cruisingchauffeur/`.

References II

[9] Waymo. (2021), Waymo Press - Images, [Online]. Available: https://waymo.com/press/.

[10] Cruise. (2021), Cruise Newsroom - Images, [Online]. Available: https://www.getcruise.com/news.

[11] Zoox. (2021), Zoox Press Room, [Online]. Available: https://zoox.com/press/.

[12] TuSimple. (2021), TuSimple Press Kit, [Online]. Available: https://www.tusimple.com/media/.

[13] Mercedes-Benz Group. (Dec. 2021), First internationally valid system approval for conditionally automated driving, [Online]. Available: https://group.mercedes-benz.com/innovation/product-innovation/autonomous-driving/system-approval-for-conditionally-automated-driving.html.

[14] Ö. Ş. Taş *et al.*, "Functional system architectures towards fully automated driving," in *2016 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2016, pp. 304–309.

[15] K. Jo *et al.*, "Development of autonomous Car—Part i: Distributed system architecture and development process," *IEEE Trans. Ind. Electron.*, vol. 61, no. 12, pp. 7131–7140, Dec. 2014.

[16] A. Scheel and K. Dietmayer, "Tracking multiple vehicles using a variational radar model," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 10, pp. 3721–3736, Oct. 2019.

[17] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

References III

[18] G. Tanzmeister, "Grid-based environment estimation for local autonomous vehicle navigation," PhD thesis, Technische Universität München, 2016.

[19] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, Mar. 1985, pp. 116–121.

[20] D. Neven *et al.*, "Towards End-to-End lane detection: An instance segmentation approach," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2018, pp. 286–291.

[21] J. Thomas, "Erstellung eines sensorbasierten straßenmodells für das automatisierte fahren," PhD thesis, Freie Universität Berlin, Mar. 2021.

[22] W. Ma *et al.*, "Exploiting sparse semantic HD maps for Self-Driving vehicle localization," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov. 2019, pp. 5304–5311.

[23] S. Casas *et al.*, "The importance of prior knowledge in precise multimodal prediction," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020, pp. 2295–2302.

[24] L. Ma *et al.*, "Efficient Sampling-Based motion planning for On-Road autonomous driving," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 4, pp. 1961–1976, Aug. 2015.

[25] S. Albelwi and A. Mahmood, "A framework for designing the architectures of deep convolutional neural networks," en, *Entropy*, vol. 19, no. 6, p. 242, May 2017.

References IV

[26] T. Roddick, A. Kendall, and R. Cipolla, "Orthographic feature transform for monocular 3D object detection," in *30th British Machine Vision Conference*, Sep. 2019.

[27] E. Romera *et al.*, "ERFNet: Efficient residual factorized ConvNet for Real-Time semantic segmentation," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 1, pp. 263–272, Jan. 2018.

[28] B. Yang, W. Luo, and R. Urtasun, "PIXOR: Real-time 3D object detection from point clouds," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2018, pp. 7652–7660.

[29] K. Chen *et al.*, "MVLidarNet: Real-Time Multi-Class scene understanding for autonomous driving using multiple views," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020, pp. 2288–2294.

[30] M. Liang *et al.*, "Multi-Task Multi-Sensor fusion for 3D object detection," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019, pp. 7337–7345.

[31] M. Bansal, A. Krizhevsky, and A. Ogale, "ChauffeurNet: Learning to drive by imitating the best and synthesizing the worst," in *Robotics: Science and System XV*, Freiburg im Breisgau, Germany, Jun. 2019.

[32] M. Törngren *et al.*, "Architecting safety supervisors for high levels of automated driving," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Nov. 2018, pp. 1721–1728.

References V

[33] F. Oboril and K.-U. Scholl, "Risk-Aware safety layer for AV behavior planning," in *IEEE Intelligent Vehicles Symposium*, Oct. 2020, pp. 1651–1657.

[34] *SOME/IP Protocol Specification*, 696, AUTOSAR, Nov. 2021. [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/foundation/21-11/AUTOSAR_PRS_SOMEIPProtocol.pdf.

[35] I. O. for Standardization, *Road vehicles - Open interface for embedded automotive applications - Part 3: OSEK/VDX Operating System (OS)*, ISO 17356-3:2005. Vernier, Geneva, Switzerland: International Organization for Standardization, 2005. [Online]. Available: https://www.iso.org/standard/40079.html.

[36] A. Dosovitskiy *et al.*, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[37] W. W. Royce, "Managing the development of large software systems," in *Technical Papers of Western Electronic Show and Convention*, ser. WesCon, Aug. 1970, pp. 1–9.

[38] I. O. for Standardization, *Road vehicles - Functional safety - Part 1: Vocabulary*, ISO 26262-1:2018. Vernier, Geneva, Switzerland: International Organization for Standardization, 2018. [Online]. Available: https://www.iso.org/standard/68383.html.

References VI

[39] ——, *Road vehicles - Safety of the intended functionality*, ISO/PAS 21448:2019. Vernier, Geneva, Switzerland: International Organization for Standardization, 2019. [Online]. Available: https://www.iso.org/standard/70939.html.

[40] *Guidelines for the use of the C++14 language in critical and safety-related systems*, 839, AUTOSAR, Mar. 2019. [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/adaptive/21-11/AUTOSAR_RS_CPP14Guidelines.pdf.

[41] *MISRA C++:2008 Guidelines for the use of the C++ language in critical systems*, MISRA, Jun. 2008. [Online]. Available: https://www.misra.org.uk/product/misra-c2008/.