PG iTraffic with TurtleBots: Preliminary Project Report

All members

October 9, 2023

Contents

This documentation serves as the very first submission, providing information of a project still in development. It has been refined to its current state, representing the progress achieved thus far. Further iterations and enhancements to this documentation are expected and planned, but for now, this submission stands as a reflection of our ongoing efforts.

1 Product Vision

The goal of the project group iTraffic with TurtleBots is to develop autonomous driving functions with TurtleBots to solve scenarios of varying complexity based on a German Autobahn. To assure that these driving functions meet their specifications, test-based validation will be used.

The project group will use the TurtleBot platform as a basis. This makes it possible to experimentally validate autonomous driving functions in different simulated traffic scenarios with the use of mostly low-cost sensor technology. The goal is to present and tackle the challenges of autonomous driving in a way that is cost-effective and low-risk.

In order to achieve this, a platform for creating autonomous driving functions based on the TurtleBot will be developed. It will enable members of the project group as well as future developers to implement controllers for vehicles on different autonomy levels and simulating those vehicles on a TurtleBot. This platform is called TurtleCar.

Additionally, TurtleCar will provide capabilities to validate the controllers in a simulated as well as a real life environment. For this, a language to define test cases for the controllers will be developed. The simulation suite „Gazebo“ will be used for testing in a simulated environment. Using the simulation, it will be possible to automatically execute test cases, gather the results and determine whether the test conditions where met. TurtleCar will ensure that both environments behave similarly with regard to the inputs and outputs of the controller.

Using the TurtleCar platform, several scenarios of various complexities will be developed and provided for the creation of test cases for the testbed. This will enable a developing cycle that is closely related to the DevOps method: The development of the testbed follows the requirements posed by the scenarios, and can be adjusted as needed.

1.1 Autonomous Driving Functions

The scenarios developed as part of this project group will all be based on a highway with the properties of a German Autobahn. Controllers with three different levels of autonomy will be built:

- no autonomy
- partially automated
- highly automated

The functions will be implemented using suitable, robust control strategies. They will be based on the current state of the art in the control engineering domain.

1.2 Functions of the Non-Autonomous Vehicle

The non-autonomous vehicle has no assistance systems. In the non-autonomous vehicle, a human driver controls the vehicle completely. They can define the speed and the steering angle, and the bot moves according to the vehicle's dynamics.

Functions of the Partially Automated Vehicle The partially automated vehicle can perform certain functions autonomously within bounded conditions. It may call for the driver's intervention if needed.

Lane Keeping Assistant The driver will determine the speed of the vehicle. As long as the Lane Keeping Assistant is activated, the vehicle will keep to the center of its current lane without the need of the driver to control the steering angle.

Adaptive Cruise Control The speed of the vehicle will be partially determined by the driver. When Adaptive Cruise Control is activated, and another, slower vehicle is driving in the front, the speed will be adjusted so that a safety margin will be kept.

Lane Changing When the Lane Changing function is engaged, if conditions permit, the vehicle will execute lane changes, while maintaining appropriate spacing from neighboring vehicles.

Collision Avoidance System The vehicle will avoid static obstacles like road works by changing lanes or stopping safely before the obstacle until a safe lane changing is possible.

Overtaking The vehicle will avoid obstacles moving in the same lane, like a slower car ahead, by changing lanes or reducing speed until a safe lane changing is possible.

Functions of the Highly Automated Vehicle The highly automated vehicle is able to drive on the highway without needing intervention from the driver. All actions will be self-initiated. Using only the aforementioned driving functions, it will move the vehicle forward as safely as possible without any input from the driver while adhering to the German traffic regulations in terms of safety margins. Also, it will adhere to speed limits and „no overtaking“ road signs.

Malicious Agent Avoidance The vehicle will avoid collisions with cars which are moving in defiance of traffic rules by choosing a safe driving strategy.‘

Platooning In platooning mode, the vehicle will join a closely coordinated group of vehicles traveling in a convoy-like formation. The system will automatically control the vehicle's speed, following distance, and positioning within the platoon. The platooning system will continuously communicate with other vehicles in the group, ensuring safe and efficient travel.

2 State of the Art

This section introduces the autonomy levels according to SAE, shows the group's ongoing research on driving functions, and outlines past projects working on similar topics.

2.1 Levels of Automotive Autonomy

The Society of Automotive Engineers (SAE) defines levels of autonomy in on-road automated driving vehicles. The SAE standard J3016_202104 [26] outlines the six levels of driving automation, ranging from Level 0 (no automation) to Level 5 (full automation) depicted in Table 1 as follows.

Level	Description
Level 0	No Driving Automation
Level 1	Driver Assistance
Level 2	Partial Driving Automation
Level 3	Conditional Driving Automation
Level 4	High Driving Automation
Level 5	Full Driving Automation

Table 1: Levels of driving automation according to the SAE standard J3016_202104 [26]

While level 1–2 use „driver support“ features, level 3–5 use „automated driving“ features. The level of driving automation of a vehicle is determined by a combination of factors: the extent of required human involvement in driving tasks, the vehicle's capability to perform driving functions, and the operational design domain under which a feature is designed to function (e.g. environmental restrictions). The standard also differentiates between three types of actors: the (human) user, the driving automation system, and other vehicle systems and components.

Because of this, systems that provide alerts about driving hazards are excluded from this classification as they neither automate driving tasks nor change the driver's role in performing them. Additionally, the lane keeping assistant, the electronic stability control or other certain types of driver assistance systems are not covered by this driving automation classification. This is because it provides momentary intervention rather than sustained automation of driving tasks.

2.2 Assisted Driving Functions

The following contains the initial research done before implementing driving functions.

2.2.1 Lane Departure Warning System [19]

The Lane Departure Warning System is a feature designed to alert the driver when their vehicle unintentionally drifts from its lane without using a turn signal. There are different types of such a system:

- Lane departure warning (LDW)
- Lane keeping assist (LKA)
- Lane centering assist (LCA)
- Automated lane keeping systems (ALKS).

While the LDW only warns the driver, the LKA ensures that the vehicle stays in its lane. Furthermore, the LKA makes sure that the car stays centered in its lane. The ALKS is a combination of LKA and ACC.

There are several vehicles in which a LDWS is integrated dating back to 2001. Generally, they are based on video sensors mounted behind the windshield, laser sensors and infrared sensors.

The LDW observes the TurtleBot's movements and its position within the lane. It can recognize the TurtleBot leaving its lane without using a turn signal and gives an alert. The LDW can be implemented with the LIDAR Sensor by orienting the lanes along a wall and / or with the camera

When the lanes can be perceived, the TurtleBot leaving its lane or starting to leave its lane has to be recognized. This can be achieved by observing the displacement of the TurtleBot in its lane. To later control the TurtleBot to stay in its lane, the direction in which the TurtleBot deflects should be identified as well.

2.3 Related Projects

In the past, there were several projects from the Carl von Ossietzky University of Oldenburg who dealt with implementing driving functions on hardware representing a vehicle. In the following, these will be described and distinguished from the project group.

„Realtime Controlled Cooperative Autonomous Racing System“ (RCCARS) has undertaken the task to develop a safety-critical system using the racetrack Mini-Z Grand Prix Circuit 30 and RC-Cars from Kyosho. This system is responsible for observing and controlling autonomously operating vehicles on a racetrack. In their „collision-free“ scenario, a single car is supposed to autonomously complete five laps on the racetrack at a minimum average speed of 1.5 m/s without colliding with the track's boundaries. [6]

„Realtime Controlled Cooperative Autonomous Racing System Next Generation“ (RCCARSng) builds upon the work of RCCARS. It extends the project by adding a second car and several static obstacles. Both cars are supposed to complete a minimum of ten collision-free laps. During this, both vehicles have the opportunity to overtake each other and should avoid obstacles while doing so. This group divides their scenario „collision-free overtaking“ in three variants:

- One vehicle following the other.
- One vehicle overtaking the other.
- Following and overtaking while avoiding obstacles. [5]

RCCARS and RCCARSng both use global knowledge and external calculations. A camera situated above the racetrack perceives the track and the vehicles on the track. There exists an external component responsible for location determination and for controlling the vehicles. For the overtaking function, they use a preceding trajectory calculation implemented in Matlab.

„Emergency Braking Assistant for fully Autonomous Cars“ (EmBrAAC) has undertaken the task to develop a real-time vehicle assistant. Depending on the situation, it should be capable of calculating an evasive strategy or performing emergency braking. They use a remotely-controlled vehicle from Traxxas in combination with a predefined and self-build course. Their focus lies on real-time capabilities and contract-based design. [13]

Within the context of the university course „Forschendes Lernen - Mobiles Multiagenten-Robotersystem“ eight students investigated and practically implemented method-oriented topics in the field of mobile robotic systems using a TurtleBot. They familiarized themselves with the simulation software Gazebo and used it to validate initial prototypes before transferring them into real hardware. After doing some fundamental work with the TurtleBot and Gazebo software, the students were split into two groups.

One group focused on using Simulink to address the question „How can an autonomous driving function for obstacle avoidance be developed?“. As part of this, they developed control algorithms that enable the robot to follow the desired path, navigate around obstacles, and perform precise navigation.

The other group, using Python, explored the question „How is realistic driving behavior simulated?“. In doing so, they researched vehicle models and implemented a suitable one. This included considering factors such as friction, inertia, road conditions, and other physical properties.

During the course, Simulink and Python were compared for the implementation of driving functions on a TurtleBot. The course was meant as a preliminary project for the „iTraffic with TurtleBots“ project group. The project group adopted the vehicle model and knowledge about the differences between reality and Gazebo simulation.

The project group „iTraffic with TurtleBots“ enables the utilization and implementation of driving functions on a TurtleBot based on local knowledge.

The implemented functions use a camera und a LIDAR sensor on the TurtleBot. These sensors can be combined freely. The environment in which the TurtleBot operates and the TurtleBot itself closely resembles reality: The TurtleBot is located on a three-lane highway and behaves like a specific car. The goal is to develop a modular development platform. That means vehicle models, environments and driving functions can be added and are interchangeable. Alongside the creation of the development platform, an automated testing platform is created. This allows experimentally validating the driving functions.

3 Environment

To develop a testing framework based on the German Autobahn, it is necessary to create a representative environment. Such an environment has to be defined in a way that can be used in real-life and in a Gazebo simulation.

3.1 Model

A German Autobahn generally has the following measurements [25]:

- Lane width: 2,75m - 3,75m
- Dash mark width: min. 15cm
- Dash mark length: 6m
- Dash mark spacing: 12m

The real-life and Gazebo models of a three-lane Autobahn use the dimensions depicted in figures Figure 1 and Figure 2.

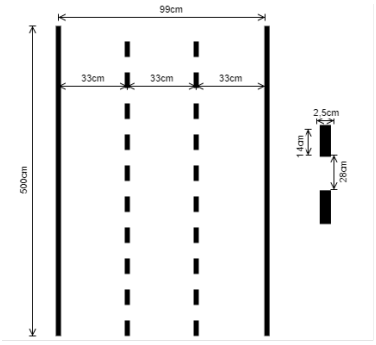

Figure 1: Specifications of the straight Autobahn environment

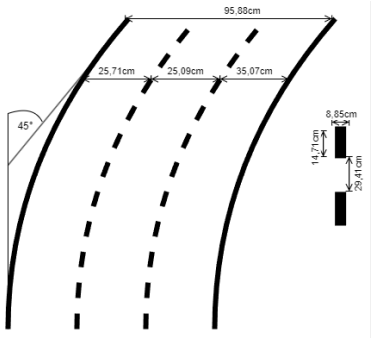

Figure 2: Specifications of the curved Autobahn environment

This is done because the TurtleBot is supposed to represent a real car, but its dimensions are much smaller than a car's. Thus, the road dimensions need to be scaled down to correspond to the TurtleBot's. Figure 3 shows the specifications of the updated road dimensions.

Curve		
Radius	280	m
Circuference	1759,291886	m
90 deg. Circuference	439,8229715	m
45 deg. Circuference	219,9114858	m

Scale Length		
Length TB	0,1	m
Length Golf	4,284	m
Scale Length	0,02334267	

Length of 45 Grad Turn	5,133321329	m

Scale Speed		
Pretend Speed	100	km/h
Pretend Speed	27,77777778	m/s
Real TB Speed	0,15	m/s
Scale Speed	0,0054	

Time to drive 45 Grad Turn	34,2221422	s

Scale Width		
Width TB	0,178	m
Width Golf	2,05	m
Scale Width	0,08682927	

Scale diff Length vs Speed
4,3227167410174

Figure 3: Scales of the TurtleBot in relation to real cars

The environments are depicted as in Figure 4 and Figure 5.

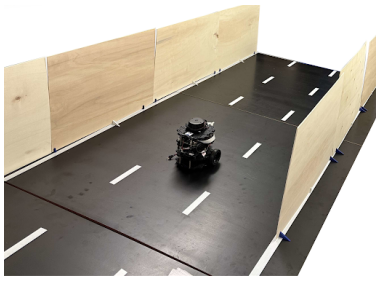

Figure 4: Road in real-life

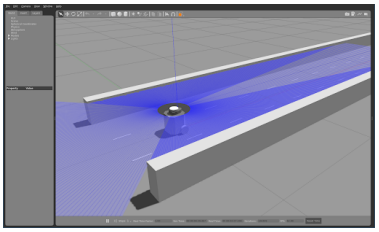

Figure 5: Road in the simulated environment (Gazebo)

3.2 Differences between Gazebo and Real-Life

Various aspects of the Gazebo simulation lead to inevitable differences between it and the actual real-world setup. The following differences between the two have been identified:

- Gazebo has a continuous guardrail. In the real-life model, this is approximated with smaller straight segments, which can lead to inaccuracies especially in curves. If the segments are placed too far apart, it can happen that a hole in the wall is detected, resulting in calculating wrong lanes.
- The small stands for the guardrail segments in real-life currently reach

into the lanes. This provides a potential hazard to the TurtleBot at the moment.

- The obstacle in Gazebo is 33cm x 100cm while in real-life it is 32cm x 44cm. This means that the real-life obstacle doesn't completely fill a lane.
- The positions of lanes and lane markings in Gazebo are according to the measurements in Figure 1 and Figure 2. In real-life, they might be different depending on how precisely they are set up. The ground segments in real life are not perfectly flat, resulting in small inaccuracies in the lane widths.

4 Vehicle Emulation

This page describes the vehicle model and the configuration files used to emulate a specific vehicle with the TurtleBot.

4.1 Vehicle Model

The vehicle emulation is based on the Kinematic Bicycle Model. The Bicycle Model is a simplified representation of a car's dynamics that is often used in the field of vehicle dynamics and autonomous driving for simulation and control purposes. This model captures the essential dynamics of a vehicle while being computationally efficient. It is called the „bicycle model“ as it consolidates the dynamics of a car into a two-wheeled model, where the two front wheels and the two rear wheels are each represented as a single wheel.

In the Bicycle Model, the following variables are defined:

- X and Y: The longitudinal and lateral positions of the vehicle, respectively.
- θ: The heading angle of the vehicle.
- v: The speed of the vehicle.
- α: The steering angle.
- l: The wheelbase, or the distance between the front and rear axles.

The equations provided describe the kinematic relationships of the model [11]:

$$\dot{x_1} = \dot{X} = v \cdot \cos(x_3)$$
$$\dot{x_2} = \dot{Y} = v \cdot \sin(x_3)$$
$$\dot{x_3} = \dot{\theta} = \frac{v}{l} \cdot \tan(\alpha)$$

In the context of simulating TurtleBot behavior, the Bicycle Model provides a framework to emulate vehicular motion. The TurtleBot's linear and angular velocities can be controlled to emulate the motion of a vehicle as described by the Bicycle Model. The model's parameters are mapped to TurtleBot controls as follows:

- The longitudinal velocity v of the model corresponds to the linear velocity of the TurtleBot.
- The steering angle α of the model is used to control the angular velocity of the TurtleBot, this needs to take into account that the TurtleBot only has a single wheel axis.

Additionally, a motor model is used and integrated into the Bicycle Model. Thus, the calculated speed of the vehicle is influenced by the simulated engine, which in turn is dependent on the current gear and transmission rates. The motor model plays an important role in determining the vehicle's speed and acceleration. The values used by the motor model depend on the vehicle being emulated.

4.1.1 Motor Model

The motor model is implemented through a series of calculations and functions which account for various factors including the engine's rotational speed (RPM), torque and gear ratios. The following sections provide an explanation of how the motor is modeled.

Engine RPM Calculation: The engine's rotational speed (RPM) is computed based on the vehicle's current speed, the wheel circumference, and the current gear and final drive ratios.

1. Convert the speed from meters per second (`speed_m_s`) to meters per minute by multiplying with 60:

$$\text{speed\_m\_min} = \text{speed\_m\_s} \cdot 60$$

2. Calculate the wheel rotations per minute (RPM) by dividing the speed in meters per minute by the wheel circumference (`wheel_circumference_m`):

$$\text{wheel\_rpm} = \frac{\text{speed\_m\_min}}{\text{wheel\_circumference\_m}}$$

3. Finally, calculate the engine RPM by multiplying the wheel RPM with the current gear ratio (`current_gear_ratio`) and the final drive ratio (`final_drive_ratio`):

$$\text{engine\_rpm} = \text{wheel\_rpm} \cdot \text{current\_gear\_ratio} \cdot \text{final\_drive\_ratio}$$

Torque Calculation: The current torque is calculated based on the engine's RPM. A linear interpolation function, `interp1d`, is employed to interpolate the torque values from a predefined set of engine speed and torque points.

Gear Shift Handling: The motor model checks whether a gear shift is available or necessary based on the current RPM and the specified RPM ranges for each gear. If a gear shift is required, the current gear is updated, and the time of the last gear switch is recorded.

Engine Acceleration Force Calculation: The engine acceleration force is the total force provided by the engine and is calculated using the current torque, gear ratio, final drive ratio, and the wheel radius. This calculation accounts for transmission losses.

1. Compute the engine torque after transmission by multiplying the average engine torque (`avg_engine_torque_nm`) with the gear ratio (`gear_ratio`) and the final drive ratio (`final_drive_ratio`):

$$\begin{aligned}\text{engine\_torque\_after\_transmission} = \text{avg\_engine\_torque\_nm} \\ \cdot \text{gear\_ratio} \cdot \text{final\_drive\_ratio}\end{aligned}$$

2. Compute the engine torque after accounting for engine losses by multiplying the engine torque after transmission with the engine loss factor (`engine_loss`):

$$\text{engine\_torque\_after\_losses} = \text{engine\_torque\_after\_transmission} \cdot \text{engine\_loss}$$

3. Finally, calculate the engine acceleration force by dividing the engine torque after losses by the wheel radius (`wheel_radius_m`):

$$\text{engine\_acceleration\_force} = \frac{\text{engine\_torque\_after\_losses}}{\text{wheel\_radius\_m}}$$

The Bicycle Model, paired with the simulation of a motor, enables realistic emulation of a variety of vehicles in a robust and simple way, aiding in the development and testing process.

4.2 Vehicle Configuration

Vehicle configuration files, which contain all parameters necessary to simulate a realistic vehicle, are used. These can be switched out depending on the simulated scenario. Each configuration file is written in the YAML language and contains the parameters for a specific vehicle model. Furthermore, the configuration of each vehicle includes models for single vehicle parts such as the Motor or the Transmission. This modularity enables constructing configurations using various vehicle part models that have already been defined. Currently two different configurations are used, one for simulating a sports car (Jaguar F-Type) and one for a more casual car (VW Golf VII). The modules describing vehicle parameters are divided into five categories.

4.3 Engine

The engine is a component that is used by all vehicles and usually varies from vehicle to vehicle, therefore, its parameters need to be specified separately. Performance diagrams might need to be evaluated to acquire some of the engine parameters.

Table 2: Engine Specifications

Parameter	**Unit**
Max torque	Newton Meter (Nm)
Speed at maximum torque	Revolutions per minute (RPM)
Maximum power	HP
Speed at maximum power	RPM
Average torque	Nm
Average loss	Percentage
Speed points full load	RPM
Static torque points full load	Nm

4.4 Transmission

Similar to the engine, transmissions are usually unique across different vehicle models.

Table 3: Transmission Specifications

Parameter	**Unit**
Gear switch time	Seconds
Start speed	Revolutions per minute (RPM)
End speed	Revolutions per minute (RPM)
Gear Ratio	Multiplier value
Final drive ratio	Multiplier value

4.5 Wheels

The wheels are a highly variable component when comparing different vehicles. The configuration of wheels uses one of the basic wheel size specified by the manufacturer.

Table 4: Wheel Characteristics

Parameter	**Unit**
Wheel radius	Meter
Wheel circumference	Meter
Friction	Newton

4.6 Vehicle Dynamics

The remaining parameters are dependent on the whole car.

Table 5: Vehicle Dynamics and Performance

Parameter	**Unit**
Maximum steering angle	Radians
Wheelbase	Meters
Braking force	Newton
Mass	Kilogram
Air resistance	Newton
Aerodynamic drag	Newton
Frontal area	Square meter
Maximum speed	KPH
Acceleration time 0 to 100 KPH	Seconds

4.7 VW Golf VII

The VW Golf VII was selected to represent a casual everyday car compared to the rather sporty Jaguar F-Type. The file `VW-Golf-7_2-0-TDI_DSG.yml` describes a Volkswagen Golf MK7 with an 2.0 litre diesel engine which provides 150 HP. The used parameters correspond to models built from 12/2016 to 05/2020, and mainly influence the motor and transmission type. There has been a facelift in 2017, which slightly changed the exterieur und interieur design, but has no impact on technical parameters.

The transmission type is a DSG, which is an automatic transmission in the Volkswagen Group, and has seven gears. The specific transmission type is called „DQ381“. The default wheel and tires suggested by the manufacturer are of the dimensions 205/55 R16.

Some of those specifications are not strictly bound to the car model itself, e. g. the transmission „DQ381“ is used in many other vehicles. The information regarding the specifications of this vehicle was gathered via several internet sites and is linked in the subsection below.

Important note:
In comparison to the Jaguar F-Type configuration file, the golf has two different `final gear ratio` for different sets of gears. Due to this fact, the Jaguar F-Type configuration was adapted to represent this structure.

4.7.1 Model information

Table 6: VW Golf VII Model Details

Parameter	**Details**
Model	VW Golf VII 2.0 TDI with DSG
Build Duration	12/2016 - 05/2020
Remarks	The Golf VII had a facelift in 2017 (no impact on specifications)
Engine Type	Diesel
Engine Series	VW EA288
Engine Code Letters	CRMB, DCYA, DEJA, CRLB
Displacement	1968 cm^3
Max. HP @ RPM	150 @ 3500 - 4000
Max. Torque @ RPM	340 @ 1750 - 3000
Used Wheel Size	205/55 R16
Transmission Type	DQ381
Remarks on Transmission	DSG with 7 gears (From 12/2026, 6 gears previously)
Drive Type	Front wheel drive

4.7.2 Notes for parameters

Mass

- The mass is calculated by adding 100 kg to the curb weight (Leergewicht) of the car.
- Curb weight is 1316 kg.
- 100 kg is split into:
  - avg. of 80 kg for one person.
  - roughly 20 kg of fuel (diesel mass = 0.820g per litre times half tank volumes = 25 litres).

Final Drive Ratio

- The used transmission has two values instead of a single global one for each gear.
- The final drive ratio is assigned to each corresponding gear.

Steering Angle

- *Auto Motor und Sport* specifies a „40°steering angle for conventional vehicles“ [2].
- 40°converts to 0.6981 radians.

Braking Force

- In Newton, given by weight times deceleration.
- Deceleration depends on how hard the brake is applied.
  - Emergency braking equals about $10.6m/s^2$ for the Golf MK7 [8].
  - The Minimum required by law is $2.5m/s^2$ [7].
  - For calculation $7m/s^2$ is used, which represents medium braking.
- 1416 kg times $7m/s^2$ equals 9912N.

Aerodynamic drag, Frontal area and Air resistance

- The values were taken form the collection of *Rüdiger Cordes* [10].

Gear Switch Time

- The values were taken from *VWVortex* [12].

Gears

- The following links contain information about the gear ratios:
- `https://www.golfmk7.com/forums/index.php?threads/dq381-dsg-gear-ratios.360005/`
- `https://forums.tdiclub.com/index.php?threads/shift-points-on-mk7-tdi-manual.431653/`
  - Even though internally manual gears are used in the model, the shift points should be the same as in the acquired data.

Engine

- The dyno chart was taken from *More BHP* [3].
  - It shows two graphs, the important one is the thick line representing the stock engine.
  - The chart was manually evaluated using *Engauge Digitizer* [20].

4.7.3 Further Sources

- Car:
  - https://de.wikipedia.org/wiki/VW_Golf_VII#Dieselmotoren
  - https://carwiki.de/vw-golf-7-technische-daten/(mustbemanuallysettoDiesel/150PS/2.0TDI(150PS)DSG)
  - https://www.auto-data.net/de/volkswagen-golf-vii-facelift-2017-2.0-tdi-150hp-dsg-27831
- Wheels:
  - https://www.1010tires.com/Tools/Tire-Size-Calculator/205-55R16?active=0&ismetric=true

4.8 Jaguar F-Type

The configuration of this vehicle model originates from the pre-project and contains the specifications of a Jaguar F-Type. The Jaguar F-Type was selected to represent a sports car amongst the vehicles that will be simulated.

4.8.1 Model information

Table 7: Jaguar F-Type Specifications

Parameter	**Specification**
Model	Jaguar F-Type
Engine type	3-litre V6 DOHC V6
Max. HP @ RPM	340 @ 6500
Max. torque @ RPM	450 @ 3500
Used wheel size	295/30 R20
Transmission type	Automatic, ZF8HP, RWD
Drive type	Rear-wheel drive

4.8.2 Notes for parameters

Mass

- The mass is calculated by adding the driver's weight to the curb weight of the car.
- Curb weight is 1741 kg.
- Driver's weight is 80 kg.

Engine

- Engine speed for maximum torque: 3500 rpm
- Engine speed for maximum power: 6500 rpm
- Maximum engine speed: 6500 rpm
- Minimum engine speed: 1000 rpm

Transmission

- Highest gear: 8
- Final drive ratio (differential): 3.31
- Driveline efficiency: 0.85

Tires

- Tire width: 0.295 m
- Rim diameter (converted to meters): 0.508 m
- Wheel (tire) friction coefficient: 1.1
- Rear axle load coefficient: 0.65

Vehicle

- Drag coefficient: 0.36
- Frontal area: 2.42 m^2

The parameters were taken from *X-engineer.org* [27]. Additionally, this site provides a single `final_drive_ratio` parameter that applies to all gears and therefore was initially being set only once. Due to the introduction of the VW Golf configuration (subsection 4.7) which has two different `final_drive_ratio`, this parameter was copied and is the same for both final drive ratios.

5 Sensor Augmentations

This section describes various sensor augmentations made during the project group. The augmentations are ranging from hardware modifications to additional ROStopics.

5.1 Camera

The camera of the TurtleBot streams the image data in front of the TurtleBot into the ROSnetwork. The camera data can then be used to perform lane and object detection in the frames sent by the camera. Lane boundaries and road participants are examples of objects to be detected.

5.1.1 Camera Service

The camera service is used to stream image data into the ROSnetwork. It can be used in two ways:

- With a Server
- Headless

With a Server This variant uses a web server to show the image stream sent by the TurtleBot camera on a webpage. The server is hosted on the TurtleBot's network address and is running on port 5000. To see the images, the webpage has to be refreshed once after starting the camera service. This variant is more suitable for troubleshooting.

Headless The second variant needs no user interaction for sending messages. If you call the ROSservice

```
/camera_serice
```

it will either start or stop sending images.

Parameters In the service request, different values are used for parameters as „Opcodes“ to customize how the node should behave. These are listed below:

Table 8: Parameters for the camera service request

Parameter	Description	Default Value
request	The request opcode	0
frame_width	The requested frame width of images	640
frame_height	The requested frame height of images	480
frame_rate	The framerate to capture images with	10

Table 9: Table for Opcodes

Opcode	Description
0	Toggels camera server

Table 10: Parameters for the camera service response

Parameter	Description	Default value
status	The response Code	0
message	A message with status information	n.a

Table 11: Table for status codes

Opcode	Description
0	Success

5.1.2 Camera Mount

The TurtleBot already had a static mount for the camera attached. To be able to dynamically change the viewport of the camera, the simple mount was replaced by a more advanced mount. This new mount uses a pan-tilt design to make the camera angle adjustable on two axles. The new mount was designed for and printed using a 3D-printer. For the assembly, small screws were used and the servos were put in place, even though they are not connected or controlled yet. It can be seen in Figure 6.

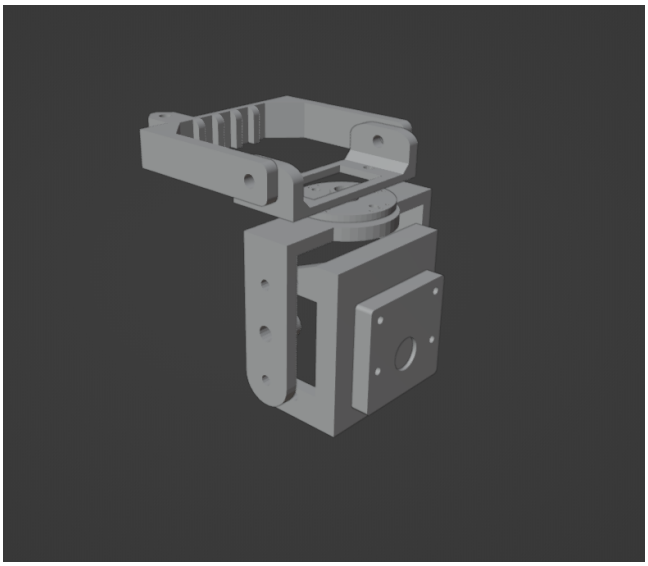

Figure 6: New camera mount with movable joints to control the camera with servo motors.

5.2 Odometry

The TurtleBot uses the Odometry topic `/odom` to publish information about the TurtleBot's position and movement. This data is however quite nosiy. In order to acquire smoother data, an Extended Kalman Filter is employed to combine Odometry and IMU data. The IMU sensor yields data about the TurtleBot's orientation. The topic `/odometry/filtered` is used to publish the filtered data.

The filter was implemented using an online guide by „Automatic Addi-

son“ [1]. The following dependency is required for the filter node:
`ros-humble-robot-localization`

6 TurtleCar-Core

There are multiple ROS Nodes running on the TurtleBot which together form TurtleCar-Core. The architecture of each node is described here. In order to run these nodes, a specific setup of the image running on the TurtleBot is required, which is explained here as well.

6.1 TurtleCar Node

In the module called `TurtleCar-Core`, the main parts of the software controlling the TurtleBot are implemented. Its tasks are to gather sensor data, define a control action according to its current scenario and goal, and publish that action to the relevant actuators.

6.1.1 Architecture

The diagram in Figure 7 shows the basic building blocks of the code. It is simplified in the way that the *TurtleCarNode* class is the root class and consists of all other classes. In order not to clutter the diagram, these compositions are not drawn.

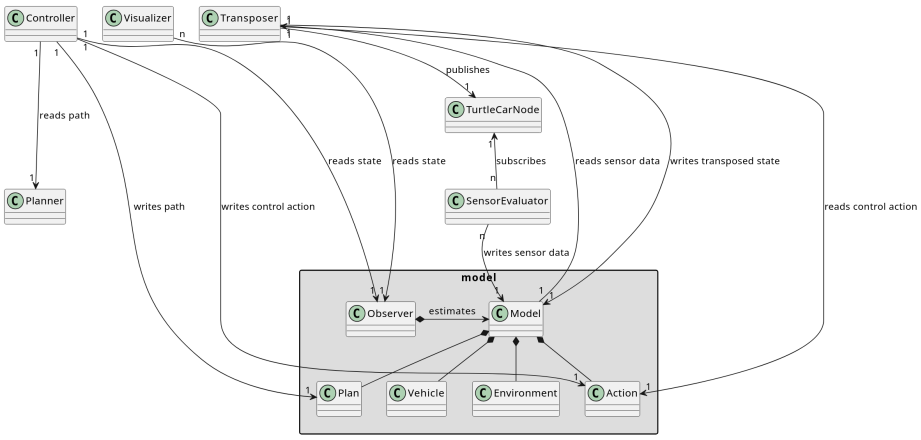

Figure 7: Static view of the architecture of the *TurtleCarNode*

The architecture is modular and can be separated into these classes:

- **TurtleCarNode:** The *TurtleCarNode* class is the root class. It provides the ROS interface which is used by other parts of the software to subscribe or publish to ROS topics. It is also the root for the tree of dependent classes.

- **Model:** The model represents the observable state of the robot. It contains information on the state of the vehicle as well as the environment and the control actions taken. It is filled by the *SensorEvaluator* classes and read out by the *Observer*.
- **Sensors Evaluators:** These classes read out sensor values by subscribing to their ROS topics and processing the information gathered to create meaningful information from them, e.g. detecting obstacles or lanes. The processed information is added to the *Model*. To gain information from a sensor and put it into the model, this class needs to be inherited from.
- **Observer:** This class acts as an Observer in the context of control design. The data in the model only represents the observable state, which may not be the complete state information needed to control the system. The observer estimates the actual state from the observable model. The *Controller* and the *Visualizer* read from this observer instead from the *Model* directly. If the model is amended, the observer probably has to be altered as well.
- **Controller:** Reads the state information provided by the *Observer* and decides on a control action depending on that state. Writes the control action back into the *Model*. Adaptation of the robots actions is done here. Third parties are able to write their own controllers, in order to implement driving functions.
- **Visualizer:** Reads the state information provided by the *Observer* and visualizes it through a GUI. You may add additional visualizers.
- **Transposer:** The goal is to simulate a car which has a different behaviour than the Turtlebot. The Transposer reads the control actions from the *Model* and maps them to the behaviour of the car model. It then publishes messages via the *TurtleCarNode* to the bot's actuators so that the robot shows that behaviour. Since it simulates the car, it also writes the information about the car's new state - like the current gear - back into the *Model*.

6.1.2 TurtleCarNode Core Loop

The core loop of the *TurtleCarNode* on a high level is shown in figure Figure 8.

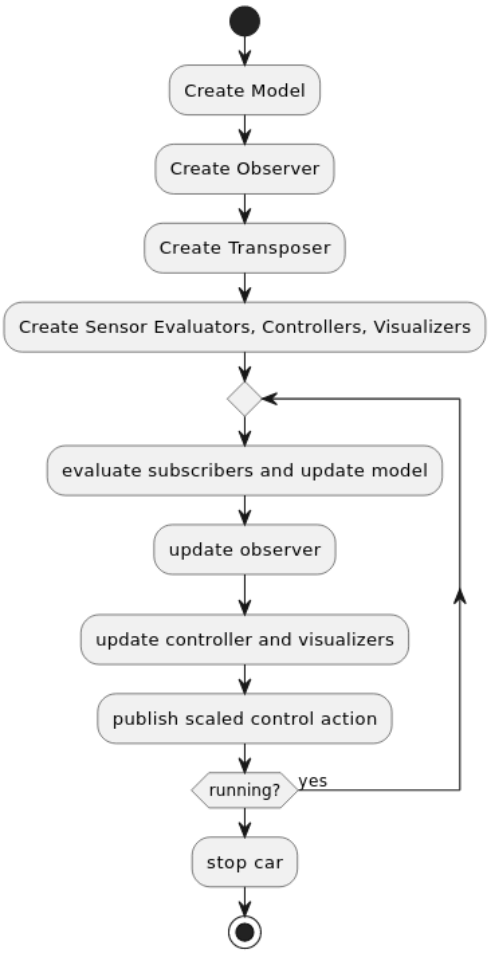

Figure 8: Start and core loop of *TurtleCarNode*

6.1.3 Filtering Sensor Values

The information gathered from the sensors via the ROS interface may need to be filtered to be usable by the modules interpreting that sensor data. In the context of the project group, two variants of filtering are defined, which are reflected in different aspects of the architecture:

Technically Necessary Filtering There exist technical reasons for filtering values directly when they are retrieved from a ROS message. One example is the LIDAR: It has a varying resolution which must be upscaled to a fixed resolution by interpolating missing values. This is done directly when retrieving the values. Sensor Evaluators using the standard `lidar.subscribe_lidar()` function implicitly receive the fixed-resolution values. When necessary, a similar standard filtering behaviour may be implemented for other sensors as well.

Task Specific Filtering Some Sensor Evaluators may have requirements for filtering the sensor values that are not necessary for processing the values, but are functional requirements related to their task. These filters are implemented in the context of the Sensor Evaluator and only used to fulfill its task, but do not influence the input to other Sensor Evaluators. Each Sensor Evaluator has to explicitly implement the filters it needs or explicitly use a filter function shared between evaluators.

6.1.4 Unit Testing

pytest [18] is used to perform unit tests. For mocking, *mockito-python* [21] is used. When writing unit tests, the following criteria should be met:

- Test one specific aspect of the code under test
- Mock the complete environment of the function. Everything that is not part of the code under test should not be executed.
- If the tests or the mocking effort is high, consider refactoring the tested code to enable smaller tests.

All tests are located in the `tests` directory.

6.2 TurtleBot ROS2 Image

It is possible to automatically build a minimal, customized image, which is real-time capable, for the TurtleBot. The repository which can be used for this can be found in the project groups GitLab [17].

Dependencies are customizable by opening

```
image_builder/data/jammy-rt-hubmle/scripts/
```

and adapting the file `phase1-target`. Under the comment „user-specific dependencies“ it is possible to add desired dependencies via apt.

To build the image, change to the top folder and run

```
make jammy-rt-ros2
```

After that, a fully bundled `.img` file is generated, which can be burned to an sd card. Detailed documentation can be found in the repository. Please note this is an updated version of another public project called *Raspberry Pi image with ROS 2 and the real-time kernel* [15].

7 Code Quality

In this part of the documentation, elaboration on the decisions regarding Coding Style and Static Code Analysis (SCA) can be found. In the context of the project group, Coding Style and SCA are differentiated. Coding Style includes the ruleset and principles which influence what code is produced. SCA consists of development tools and CI/ CD methods, which allows maintaining parity to the set Code Quality. CI stands for Continuous Integration, and is a typically automatically triggered process that performs tasks such as checking the source code, running tests and ensuring that the source code is compilable [14]. With this process, the goal of having a stable code repository in terms of Code Quality is supported, since automatic Code Quality checks are possible. This is explained in more detail in subsection 7.3.

7.1 SCA

There are two main parts of SCA used in the project group: formatting and linting. The coding style is provided by the tools used.

Formatting is the way how the code is formatted: which indentation size is used, how long lines should be and where newlines are located. Formatting ensures that each line of code that is written is in a format that is comprehensible by each member of the project group. The code formats automatically and no further manual intervention is required.

Linting on the other hand makes sure that the code that is written is error-free and adheres to a certain code style: Here, checks against unused variables, long lines and unnecessary complexity are employed. Some linting errors are also fixed by formatting, e.g. long lines. But because fixing most linting errors is a non-trivial task, oftentimes manual intervention is required.

7.2 Development Tools

In order to ensure that the code is in the correct format and to reduce its errors, tools are employed. For formatting, use *black* [4] is used. For linting, use *ruff* [23] is used.

Both tools were chosen for the following reasons:

- Opinionated
  - Being opinionated allows for adhering to community rules forged by years of development time.
  - At the beginning, there is no desire to employ custom, project group specific rules. Everything is changing constantly - there is no need for complex configurations, but for quick usage.
- Modern
  - Modern tools allow for staying cutting-edge.
  - They improve the readability of the codebase.
- Fast
  - Being fast means every machine can run the tools, even if one developer happens to have a slow machine (by modern standards).
  - There is no need to worry about the code base growing so large that the SCA tools will take an unreasonable amount of time to run.
  - Limited numbers of job runners are available in GitLab, as the instance is self-hosted. Therefore, being fast reduces the occupation on those limited resources.

7.3 Continuous Integration

In this section, the ways CI is used in the project group are described. Also, the configuration of the employed pipelines is explained. Pipelines are essentially a set of steps the source code has to pass in order to be valid.

7.3.1 Integration in the workflow with CI

In order to allow constant integration, *black* and *ruff* are used not only locally, but also in the GitLab projects pipelines. This ensures that every commit and merge request is checked.

If *black* detects that the format is not correct or *ruff* finds any linting errors, merging the respecting merge request is disallowed. Also, reviewers will immediately take notice of this and will ask the developer to fix this.

This ensures that the code in the stable branches of the projects remains protected and in a valid state. Additionally, this provides fast feedback for developers whether their code contains errors. This makes locating and fixing errors faster.

7.3.2 Pipeline

In the pipeline, ruff and black are executed. In the following, the mechanics of the pipeline are documented. This part explains the following:

- Elaboration on the pipeline concepts, not the details
- Explanation of the most important caveats, like caching and sometimes allowing pipes to fail
- Starting point for getting to know the pipeline

How the pipeline works The following will explain the structure & concepts of the pipeline in use. For a better understanding, please take a look at the *.gitlab-ci.yml*. It is included at the root of the *turtlebot* project.

In the implementation of the pipeline configuration, the official Python docker image is used, so that some configurations for the executing runner are already present.

Pipeline Building Blocks

- `before_script` Block
  - Ensures that a virtualenv is used
  - Debugs the Python version
  - Executes before each job
- `build-job`
  - Currently only a stub
  - Might be used later, when actual building of the ros packages is required
- `format-test-job`
  - Runs black and checks for formatting errors
  - Prints encountered errors to ‘stdout‘ for debugging purposes
- `lint-test-job`
  - Runs ruff
  - Looks at the .pyproject.toml file in order to configure ruff
  - Generates a codequality artifact .json, which is used by GitLab to measure code quality
  - Also prints all encountered errors and warnings to `stdout`
  - By using *dependencies*, this job only runs after `format-test-job`

Important note: The `test` in the jobs name refers to the task of testing if the source code is in a conforming state. This does not mean that the jobs are only 'test' versions.

Caching the installed pip packages The `cache` is used in order to let the runner cache installed packages, so that ruff und black are not reinstalled in every run of the pipeline.

By configuring `PIP_CACHE_DIR`, pip is told to cache its dependencies and installed packages in the directory provided - which are defined as a pipeline cache directory as well. Therefore, the cache directory gets cached in between job runs and reused.

Conclusion: Working with the pipeline Now that the pipeline is configured, it is possible to review Merge Requests based on their generated code quality report. Also, this makes sure that every line of code is formatted in a consistent way. When committing to a custom branch, or merging to ‘main‘, the pipeline is evaluated and run. Project members are required to provide conforming source code, and get hints to why their changes might not be of the desired quality.

8 Lane Detection

For a TurtleBot that is used in the context of autonomous driving, the ability to perceive and understand the road environment is of paramount importance. One crucial aspect of this perception is the lane detection, which involves identifying and tracking the lanes on the road. Accurate lane detection is a fundamental building block for many autonomous driving functions, from simple lane-keeping assistance to complex path planning and decision-making algorithms.

8.1 LIDAR-Based Lane Detection

LIDAR technology plays a pivotal role in the current state of the project's lane detection implementation. The LIDAR provides essential data about the robot's proximity to surrounding objects. The concept here is to utilize this data to calculate and represent lane boundaries accurately. Visual lanes as indicated by lane markings are therefore not directly detected but are rather extrapolated based on a given road configuration and a rightmost boundary that is detectable by the LIDAR. These desired preconditions are described below, followed by the used concepts and calculations of the current implementation.

8.1.1 Preconditions

The calculation of the lane boundaries using LIDAR assumes that a certain structure for the lanes is always present. One particular assumption is that there always exists a wall that is detectable by the LIDAR sensor on the right side of the road. Furthermore, the first lane always has a distance of w_s to this wall, forming a road shoulder with constant width. Additionally, every lane has the exact same constant width, noted as w_l in the following.

8.1.2 Coordinate Transformation

The process of the LIDAR-based lane detection begins with transforming polar coordinates into a more intuitive Cartesian coordinate system. This conversion simplifies subsequent processing steps and provides a clear representation of the environment. The Cartesian coordinate system uses the TurtleBot itself as origin $(0, 0)$. Based on a respective angle α_i and a distance value d_i of each LIDAR measuring point i, Cartesian coordinates x_i and y_i for such point can be created using the common formulas $x = d * \cos(\alpha)$ and $y = d * \sin(\alpha)$.

8.1.3 Boundary Detection and Lane Projection

Once in Cartesian coordinates, the system calculates the lane boundaries based on the distance of the robot from a surrounding wall at specific angles. In particular, this calculation uses the coordinates of a potential wall that is detected between 230° and 300°. This is effectively any wall that is to the right of the TurtleBots facing direction. A B-spline is then fitted to these data points, ensuring smooth and continuous representation of the lane-defining wall. Using B-splines offers the possibility to control the degree of the lane boundaries, which is useful to extend the lane projections from straight to curved roads. For individual points of the given B-spline, normalized orthogonal vectors are then calculated. This is done by first calculating a tangential vector of a given point on the B-spline, normalizing that vector and then rotating it by 90° into the correct direction. For a given lane $n \geq 0$, these normalized vectors determine the position of the lane's right $(j = n)$ and left $(j = n + 1)$ boundary, if multiplied with the factor $(w_s + j * w_l)$.

8.1.4 Current Lane Determination

Identifying the current lane is the next critical aspect of the lane detection. This is accomplished by evaluating the closest measured distance to the wall that is used as a basis for the lane projection, as introduced above. For example, if the robot's facing direction is parallel to the wall, the angle for the closest distance is typically at 270°. Given that this minimum distance to the boundary wall is d_w, the current lane number n_{TB} can then be calculated as follows:

$$n_{TB} = \left\lfloor \frac{d_w - w_s}{w_l} \right\rfloor$$

8.2 Camera-Based Lane Detection

The camera-based lane detection that uses image recognition concepts to directly recognize the lanes based on their markings instead of projecting them, is currently work-in-progress is roughly based on the process depicted in Figure 9.

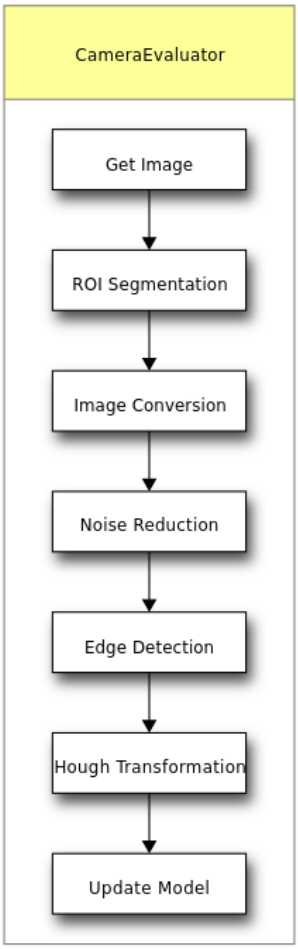

Figure 9: Image transformations for detecting lanes from the camera images.

9 Path Planning

In this section, the planning of paths for a vehicle is described. First, Path Planning and Trajectory Planning are defined, and context to these topics within the project group is given. Next, the implementation of Path Planning is explained. Also, the end of the section contains guidance for building and integrating a custom path planning module.

9.1 Definitions and Context

First, Path Planning is differentiated from Trajectory Planning, and an introduction to what both of these terms mean in the context of the project group is given. For further references and mathematical function definitions, see *Trajectory Planning* [22].

Path Planning A path P is a continuous function which connects a start q_{start} and a goal q_{goal} in a coordinate system. Therefore, the domain of P is $[0, 1]$ and its co-domain is $\mathcal{C}$, i.e. the coordinate space that is used. P is devoid of any time information, and only resembles the geometric component. When enriching it with time information, it becomes a trajectory [22]. For us, planning a path means to plan out a geometric ordered list of points that the robot should follow, disregarding any time information.

Trajectory Planning A trajectory Π is a path P endowed with a time parameterization s. s is a strictly increasing function, which gives the position on the path for each time instant t. Thus, the same path P can give rise to many different trajectories Π [22]. For us, planning a trajectory means to take into account time information to the planned path.

At the current state of the project group, trajectories are not planned, only paths. Planning trajectories would involve many more considerations, which have not been prioritized as of now.

9.2 Implementation

The architectural overview of the path planning implementation can be seen in Figure 10. It closely resembles Figure 7, but elaborates more on the path planning part.

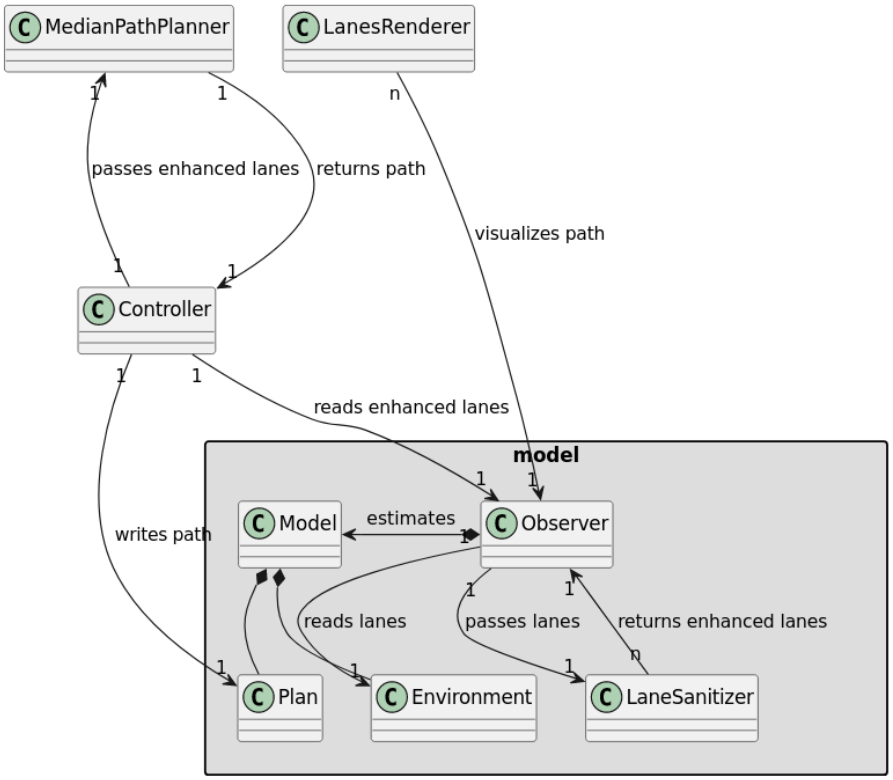

Figure 10: Architectural overview with the path planner.

9.2.1 Processing the Lane Data

Since the original sensor readings contain incomplete and not ready-to-use data, the data is processed before usage. For us, **sanitizing** lane data refers to the process of converting it into a different format, that is easier to use than the original sensor data. Whereas **enhancing** the lanes means using the format from sanitizing the lanes, but enhancing its data - e.g. with interpolation.

Sanitizing is done right after the sensor readings are completed, whereas enhancing is done by the `Observer`.

9.2.2 Sanitizing the Lanes

Where the old format specified start and end points for each pair of sensor angle readings, the new format is much simpler. It is called `BorderList` and is a two-dimensional list of points. The first dimension contains all borders of lanes, the second dimension contains a border itself, containing all points that belong to that border. The result is a `BorderList` containing each border of the given lanes exactly once.

9.2.3 Enhancing the Lanes

Using the new format, the border points are interpolated using a Euclidean distance formula. The formula is the following using points p and q:

$$Distance(p, q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}$$

When the distance of two consecutive points in the border list exceeds a parameterized threshold, new points are interpolated in between them, also using this distance formula. There is no real formalized threshold defined, the results of different parameters are tested empirically.

9.2.4 Planning the Path

Now that the lanes are in a usable format and contain enough data points to use, a plan for the path for the robot to take can be created. In order to do that, the middle of the border points from the enhanced lane data is calculated and thus creates a path along the center of a lane. The path is visualized via the `LanesRenderer`.

9.2.5 Example images

In this section, example images for the path planning module are demonstrated. The snapshots are taken directly from the debugging tool, where orange points visualize paths and yellow points indicate lanes.

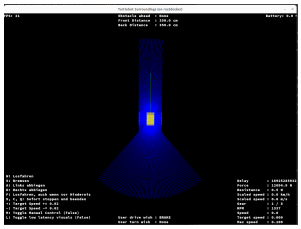

Figure 11: An ordinary path

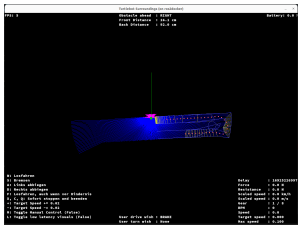

Figure 12: A more complex path

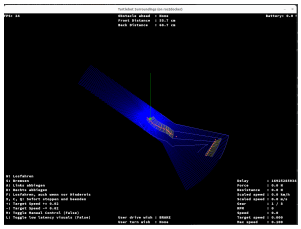

Figure 13: Lower border interpolation resolution

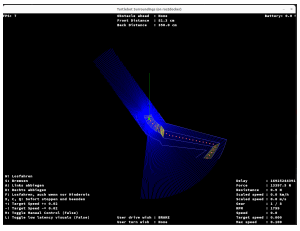

Figure 14: Higher interpolation resolution for the border

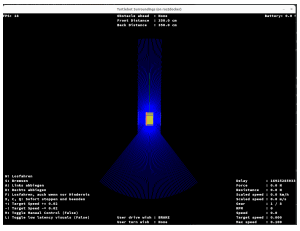

Figure 15: Higher sample rate and offset to the left of the border

9.3 Writing a custom Path Planner

In order to write a custom Path Planner, the programmer would create a class in `controllers/path/`, which extends `AbstractPathPlanner`. The planner needs access to `Observer`, because the lane information is present there. Then, implement the `plan_path()` method in the new class. It must return a `BorderList`. Afterward, an instance of the new class must be created in the controller. `Controller.planner` must be assigned to that instance. The `LanesVisualizer` will now render the calculated path of the custom Planner.

10 TurtleCar-Test

This section describes the usage of TurtleCar-Test with test cases. This includes integration of the tests with Gazebo and creation of tests in a predefined file layout.

TurtleCar-Test is a test framework developed by the project group. This framework includes an extensive test setup, allowing for interactive testing of both the Gazebo simulation and the controls of the TurtleCar framework. Additionally, it supports headless testing, where no user interface is needed.

10.1 Structure

TurtleCar-Test will select the Controller according to the Scenario and initialize the Robot within the Gazebo Simulator. After the test has completed, it will return a pass/fail result of the test. The interaction between TurtleCar-Test and TurtleCar-Core can be seen in Figure 16.

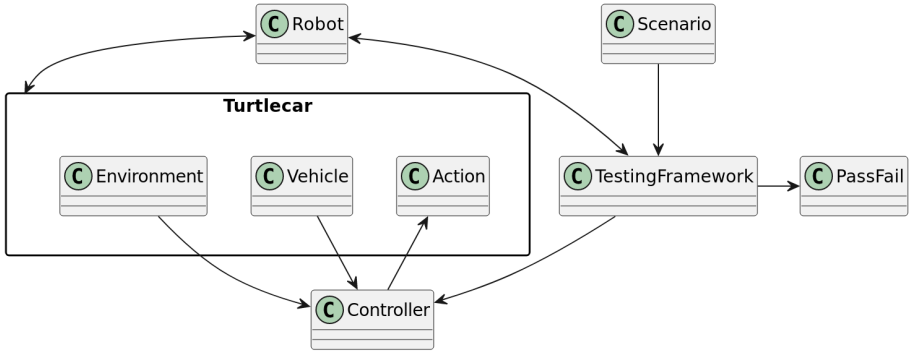

Figure 16: The static view of the interaction of TurtleCar-Test and TurtleCar-Core

10.2 Requirements

For each test case, there are corresponding requirements that must be met in order to define a correct test. This is described in the following list.

- Robot Model
  - Sensors
  - Type
- Gazebo Map
- Starting Position, Rotation on Map
- Turtlecar ROS Parameters
  - for example: Level of autonomy
  - for example: Type of controller
- Emulate user input (via ROS Parameters (special ROS Topics with status requests))
- Acceptance criteria
  - Timeout: (Unit: seconds) If specified, overstepping this timeout passes the test
  - Final Position: (Unit: Gazebo-Coordinates, see sample config for formatting) If specified, entering this area passes the test
  - Area Allowlist: (Unit: List of Gazebo-Coordinates (meters), see sample config for formatting) If specified, the bot has to stay in one of these areas for the whole test, otherwise it fails
- Failure Criteria
  - Timeout: (Unit: seconds) If specified, overstepping this timeout fails the test
  - Area Denylist: (Unit: List of Gazebo-Coordinates (meters), see sample config for formatting) If specified, entering this area fails the test

10.3 Test Case Configuration and Requirements Capturing

In order to write test cases as dynamically as possible, a file layout offering a variety of configurations for individual test cases was created. These files define formal test cases, in the sense that the necessary criteria in these files must be met. The file format is written in the markup language YAML.

```yaml
requirements:
robot_model_path: "models/robot_with_camera.model"
gazebo_map_path: "maps/curve.map"
robot_start: {x: 2, y: 4, rot: 3, vel: 0.1 }
# alternative
robot_start_area: {x_min: 2, x_max: 4, y_min: 10, y_max: 20,
        rot_min: 3, rot_max: 3, vel_min: 0.1, vel_max: 0.2 }
turtlecar_main_path: "turtlecar/main.py"
user_input:
- {time: 0.0, name: "autonomy_level", value: 1}
- {time: 0.0, name: "controller", value: "fast_lidar_controller"}
- {time: 0.5, name: "drive_wish", value: "drive"}
- {time: 0.501, name: "drive_wish", value: "brake"}
- {time: 10.0, name: "turn_wish", value: "left"}
- {time: 10.1, name: "drive_wish", value: "drive"}
acceptance_cirteria:
timeout_sec: 30
final_positions:
- {x_min: 2, x_max: 4, y_min: 10, y_max: 20, rot_min: 3,
        rot_max: 3, vel_min: 0.1, vel_max: 0.2 }
area_allowlist:
- {x_min: 2, x_max: 4, y_min: 10, y_max: 20, rot_min: 3,
        rot_max: 3, vel_min: 0.1, vel_max: 0.2 }
- {x_min: 2, x_max: 4, y_min: 10, y_max: 20, rot_min: 3,
        rot_max: 3, vel_min: 0.1, vel_max: 0.2 }
failure_criteria:
timeout_sec: 60
area_allowlist:
- {x_min: 2, x_max: 4, y_min: 10, y_max: 20, rot_min: 3,
        rot_max: 3, vel_min: 0.1, vel_max: 0.2 }
- {x_min: 2, x_max: 4, y_min: 10, y_max: 20, rot_min: 3,
        rot_max: 3, vel_min: 0.1, vel_max: 0.2 }
```

10.4 Execution

The basic structure of TurtleCar-Test consists of the following steps:

1. Read and check configuration
2. Start Gazebo reliably with map, model and start position from config
3. Start testing target (TurtleCar)
4. Wait for end criteria
5. Check pass/fail criteria
6. Stop Gazebo and testing target

7. Return Pass/Fail

10.5 Implementation Roadmap

1. Implement Gazebo launch with custom parameters ✓
   - Map
   - Robot Model
   - Start Position
   - Start Rotation
   - Start Velocity
2. Implement program that checks if Gazebo start parameter are met (check if start worked) ✓
3. Implement testing config parser ✓
4. Implement testing config checker for conflicting requirements and valid YAML schema ✓
5. Implement visualizer for allow specified areas ✓
6. (optional) Implement visualizer that allows placing areas ✓
7. Implement core loop that checks if criteria are met ✓

11 Driving Functions

In this section, the driving functions implemented by the project group are documented. For each driving function, the definition of the requirements, the controllers used to implement the driving function and the validation is described.

11.1 Lane Keeping Assistant

The requirements of the Lane Keeping Assistant are based on the ISO standard 11270 [16], but do not yet cover all aspects. These requirements are subject to rework.

11.1.1 General Requirements

- The LKA must be able to be switched on or off
  - The LKA can be toggled by user input
  - The LKA can be switched on by startup flag
- The robot must identify the lane its on and its center

- The LKA must be disabled when the lane change disabling condition according to the ALKS regulation is fulfilled
- The LKA enabled vehicle should never cross lane borders
- The robot should follow the lane's center
- The controller should be based on model prediction

11.1.2 Functional Requirements

Requirement LKA.1

GIVEN

- The driving function for the lane keeping is started

WHEN

- The driving function was started with the flag „lka-initially-enabled“ or the „K“ key is pressed after the driving function was started

THEN

- The Lane Keeping Assistant is enabled

Requirement LKA.2

GIVEN

- The Robot starts inside of lane boundaries
- The initial velocity is 0, the initial acceleration is 0, the initial steering angle is 0
- The Robot heading fulfills the following criteria:
  - If the robot is left of the center of the lane, it faces in the direction it will drive, oriented within 0 and 40.2°to the right of the lane direction.
  - If the robot is right of the center of the lane, it faces in the direction it will drive, oriented within 0 and 40.2°to the left of the lane direction.

WHEN

- A target velocity is defined by a human driver
- The lane keeping assistant is activated

THEN

- The Robot identifies the lane it is on
- The Robot accelerates to the speed defined by the human driver and maintains this speed
- The Robot follows the center of the lane
- The Robot never crosses lane borders
- The steering angle is always within the vehicle's specifications

Requirement LKA.3

GIVEN

- The robot is driving with arbitrary speed, arbitrary acceleration
- The lane keeping assistant is active
- The steering angle is arbitrary within the vehicle's specification
- There is no steering input from the user

WHEN

- The lane change disabling condition according to the ALKS regulation is fulfilled

THEN

- The lane change assistant is disabled

Requirement LKA.4

GIVEN

- The LKA is enabled

WHEN

- The relevant user input is received

THEN

- The Lane Keeping Assistant is disabled

11.1.3 Non-Functional Requirements

LKA.A

The controller for the lane keeping assistant is based on model prediction.

11.1.4 Additional Information

The maximum possible orientation is based on the most narrow curve that a car with the wheel base length and the maximum steering angle of a VW Golf. When in the center of the lane, the greatest angle it can recover from is 31°. This can be computed as follows:

a: maximum steering angle (40°for VW Golf)
w: wheelbase length (2.6365m for VW Golf)
r: radius of turning circle
$r = \frac{w}{\tan(a)}$
Then half of the golf's width is added to the to the radius:
$r_{golf} = r + 0.9$
This is shown graphically in Figure 17.

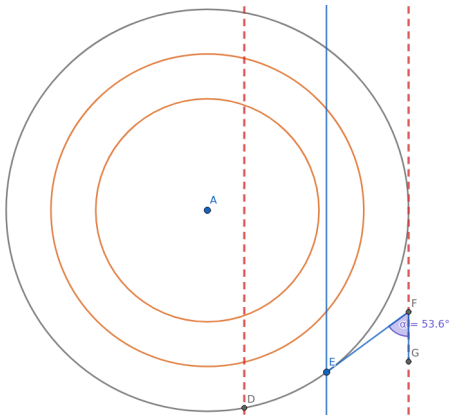

Figure 17: Graphical representation of the maximum heading pointing out of the lane that a car can recover from

11.1.5 Implementation

For the implementation, a controller based on the bicycle model is used. Since the bicycle model is a nonlinear differential equation, it is linearized in order to obtain a linear controller.

Controller Model The Bicycle Model is defined as follows:

$$\dot{x_1} = \dot{X} = v \cdot \cos(x_3)$$
$$\dot{x_2} = \dot{Y} = v \cdot \sin(x_3)$$
$$\dot{x_3} = \dot{\theta} = \frac{v}{l} \cdot \tan(u_1)$$

where X is the position in the linear direction of the car, Y the lateral position, and θ the heading. These are all relative to the next point that the

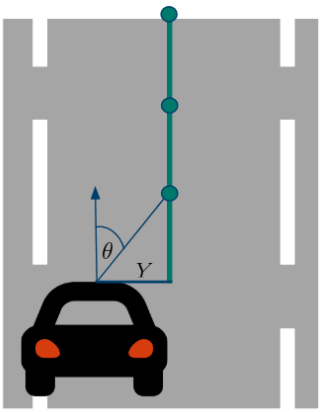

Figure 18: A graphical representation of the state variables used for the controller of the lane keeping assistant. Y and θ are always relative to the next point provided by the path planner.

path planner provides. Since the Lane Keeping Assistant isn't able to influence the speed forwards and only the lateral deviation is relevant $\dot{x_1}$ can be removed to simplify the model:

$$\dot{x_1} = \dot{Y} = v \cdot \sin(x_2)$$
$$\dot{x_2} = \dot{\theta} = \frac{v}{l} \cdot \tan(u_1)$$

A depiction of the meaning of Y and θ can be seen in Figure 18.

This allows for a simpler linearization. The operating point to linearize around is $x = 0$ and $u = 0$. This represents the state where the vehicle is exactly on the line that has to be followed, and assumes that the controller only needs to make small corrections.

With that the system is linearized:

$$\begin{aligned}
\dot{x}(t) = f(x,u) = Ax + Bu &\approx f(0,0) + \frac{\partial f}{\partial x}\Big|_{\substack{u=0\\x=0}} \cdot x + \frac{\partial f}{\partial u}\Big|_{\substack{u=0\\x=0}} \cdot u \\
&= \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2 s} \end{bmatrix}\Bigg|_{\substack{u=0\\x=0}} \cdot x + \begin{bmatrix} \frac{\partial f_1}{\partial u_1} \\ \frac{\partial f_2}{\partial u_1} \end{bmatrix}\Bigg|_{\substack{u=0\\x=0}} \cdot u \\
&= \begin{bmatrix} 0 & v\cos(x_2) \\ 0 & 0 \end{bmatrix}\Bigg|_{x=0} \cdot x + \begin{bmatrix} 0 \\ \frac{v}{l \cdot \cos^2(u)} \end{bmatrix}\Bigg|_{u=0} \cdot u \\
&= \begin{bmatrix} 0 & v \\ 0 & 0 \end{bmatrix} \cdot x + \begin{bmatrix} 0 \\ \frac{v}{l} \end{bmatrix} \cdot u
\end{aligned}$$

The state feedback control law $u = -[k_1\ k_2] \cdot x$ is used to design the controller.

This results in the closed-loop function:

$$f_{cl} = \begin{bmatrix} 0 & v \\ 0 & 0 \end{bmatrix} \cdot x + \begin{bmatrix} 0 \\ \frac{v}{l} \end{bmatrix} \cdot (-[k_1\ k_2] \cdot x) = \begin{bmatrix} 0 & v \\ 0 & -\frac{v}{l} * k_2 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

From the closed loop function it can be seen that k_1 can remain undetermined, as the lateral position has no direct influence on the control input. Now the operating domain for the velocity the Lane Keeping Assistant should be stable in has to be chosen. For this $v \in (0, 34]$ was chosen, which is about 0 to 120 km/h.

Using Matlab the characteristic polynomial for which all eigenvalues have real parts strictly less than 2 was determined (using an example from previous exercises). With the coefficients, it is possible to solve for values of k_2 which hold the closed loop system in a stable domain. For this, the parameter space was sampled with a step size of 0.5. Since the system is uncontrollable for $v = 0$, sampling started at 0.5. This resulted in values for $k_2 \in [0.05, 1.4]$.

This enabled us to build a stable controller for the Lane Keeping Assistant. From the domain of stable values for k_2, choosing $k_2 = \frac{l}{v}$ has been determined experimentally to yield the best results for any speed v.

11.2 Adaptive Cruise Control

11.2.1 General Requirements

- The vehicle must be able to activate/deactivate Adaptive Cruise control depending on the drivers wishes
  - The ACC must be deactivated if the driver takes manual control
- The vehicle must keep at least a minimum safe distance including a suitable error margin
  - The vehicle should keep the same speed as the front vehicle
  - The vehicle must be able to reduce its speed to keep the minimum safe distance
  - The vehicle should be able to increase its speed to keep the distance to front vehicle
- The driver can issue a command to drive at a certain speed overriding the ACC
- The ACC must be disabled if the driver issues a brake command
- The ACC controller should be build using model prediction

Definition: Minimum Safe Distance To be defined according to relevant regulations: Minimum safe distance which a following vehicle needs to maintain in order to be able to decelerate if the leading vehicle brakes (with bounds for deceleration)

In the German traffic regulations, a rule of thumb to determine the distance between two vehicles considered safe is described: The vehicle needs to keep a distance of at least half of the current speed (kilometers per hour) in meters [9, §2 Abs. 3a S. 2a].

11.2.2 Functional Requirements

Requirement ACC.1

GIVEN

- The ego vehicle is driving behind another vehicle. Both vehicles have arbitrary speed and the ego vehicle maintains at least minimum safe distance to the leading vehicle.

WHEN

- The driver triggers the switch for the adaptive cruise control

THEN

- The Adaptive Cruise Control is enabled

Requirement ACC.2

GIVEN

- The vehicle is driving behind another vehicle with arbitrary speed v_i with at least minimum safe distance including error margin

WHEN

- The adaptive cruise control is enabled

THEN

- The ego vehicle drives at most with velocity v_i
- The ego vehicle accelerates or brakes within the velocity bounds so that it maintains at least a minimum safe distance including error margin to the leading vehicle

Requirement ACC.3

GIVEN

- The adaptive cruise control is enabled
- The vehicle is driving behind another vehicle with arbitrary speed v_i and has at least minimum safe distance including error margin

WHEN

- The other vehicle brakes to velocity v_b

THEN

- The ego vehicle drives at most with velocity v_b
- The ego vehicle decelerates to velocity v_b and maintains at least a minimum safe distance without error margin at all times

Requirement ACC.4

GIVEN

- The adaptive cruise control is enabled
- The leading vehicle is driving with velocity v_i
- The vehicle is driving behind another vehicle with arbitrary speed v_i and has a a distance to the leading vehicle that is smaller than the minimum safe distance including error margin
- The driver does not give a command to accelerate to a speed greater than v_i

WHEN

- No Action

THEN

- The ego vehicle drives at most with velocity v_i
- The ego vehicle decelerates to until it maintains at least a minimum safe distance including error margin

Requirement ACC.5

GIVEN

- The adaptive cruise control is enabled
- The vehicle is driving behind another vehicle with arbitrary speed v_i and has at least minimum safe distance including error margin

WHEN

- The other vehicle accelerates to velocity v_a

THEN

- The ego vehicle drives at most with the minimum v_m of velocities v_i and v_a
- The ego vehicle accelerates to velocity v_m and maintains at least a minimum safe distance with error margin at all times

Requirement ACC.6

GIVEN

- The adaptive cruise control is enabled
- The vehicle is driving behind another vehicle with arbitrary speed v_i and has at least minimum safe distance including error margin

WHEN

- The driver continuously issues a command to drive with velocity v_t

THEN

- The ego vehicle accelerates to velocity v_t without regard for the minimal safe distance

Requirement ACC.7

GIVEN

- The adaptive cruise control is enabled

WHEN

- The relevant user input is received

THEN

- The adaptive cruise control is disabled
- The vehicle drives according to the driver's commands only

Requirement ACC.8

GIVEN

- The adaptive cruise control is enabled

WHEN

- The driver issues a braking command

THEN

- The adaptive cruise control is disabled
- The vehicle drives according to the driver's commands only

11.2.3 Non-Functional Requirements

ACC.A

The controller for the lane keeping assistant is based on model prediction.

12 Organization

This section describes everything related to the internal organization of the project group. A more detailed description on how the product vision will be achieved is provided here.

12.1 Milestones and Timeline

In order to reach the project group's goal, the following four milestones as listed in the table below were defined.

Milestone	Start date	End date
MS 1: Lane keeping assistant and fundamental architecture	05.05.2023	08.09.2023
MS 2: Adaptive cruise control and basic testbed features	09.09.2023	28.09.2023
MS 3: Autonomy Features, Robot Vision	29.09.2023	22.12.2023
MS 4: Rogue actor and platooning	23.12.2023	07.03.2023

Table 12: Planned milestones

Furthermore, a more detailed time schedule depicted in Figure 19 is offered. The thick vertical lines depict the end of a milestone. Additionally, the epic can be grouped together as follows: driving functions (green), test framework (orange), obstacle avoidance (purple), platooning (blue), documentation (yellow), and higher-level (grey).

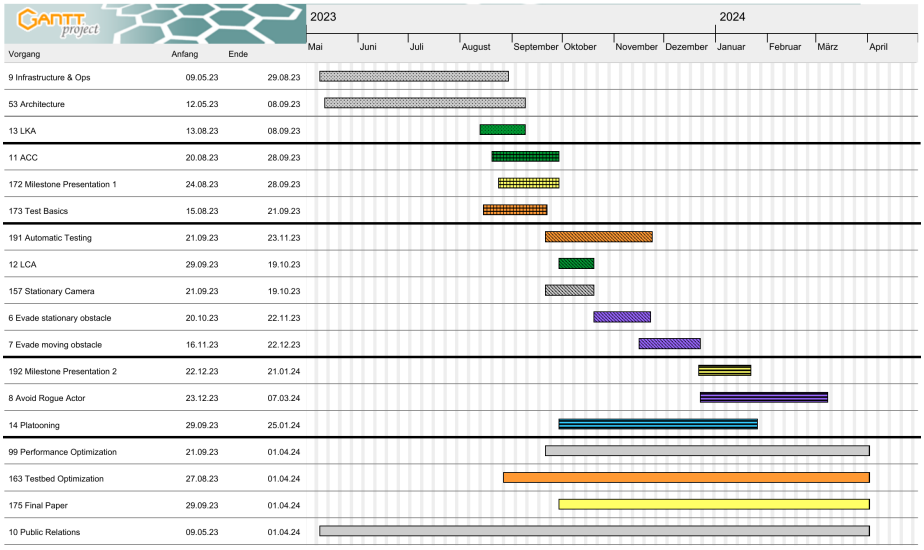

Figure 19: Gantt chart of epics

12.2 Sprint-flow

The previously defined milestones will be archived in an agile way using the scrum process [24]. A sprint lasts three weeks and consists of the following aspects:

- **Feature-Planning (FP)**
  In this phase Product Owner (PO) and Business Engineer (BE) and all interested parties consider which features should be developed in the future to reach the milestones. The features are recorded in Jira. Tickets are created that contain the needed requirements.

- **Implementation**
  During this period, the tickets are processed, documentation is written, and reviews are performed by others so that they can finally be merged.

- **Review**
  The goal of the review is to bring all stakeholders up to date. It should be mentioned which goals have been achieved and the progress should be presented.

- **Retrospective**
  The team sits down internally at the retrospective at the end of the sprint and draws a summary. The focus is on filtering out problems, exploring possible solutions and citing positive aspects.

During the sprint, a weekly serves as an exchange with the stakeholders by giving a quick presentation of last week's progress. Internally, meetings

are scheduled twice a week. Once every sprint, a refinement of the backlog is planned which is done to facilitate the feature planning and refine Jira tickets to enable faster sprint plannings.

12.3 Sprint Workflow

To assure that all members follow the same workflow regarding the arising tasks during a sprint, the following well-defined workflows have been agreed upon. For example, every sprint follows a specific workflow. An overview is given in Figure 20, where every colored step (except the „Sprint planning“) resembles one column in a Jira Sprint Board, as it is shown in Figure 21. First, the sprint has to be planned. Every ticket in the sprint is then assigned to one or more people. When they have finished processing the ticket, it goes into review and finally into acceptance by the PO or BE.

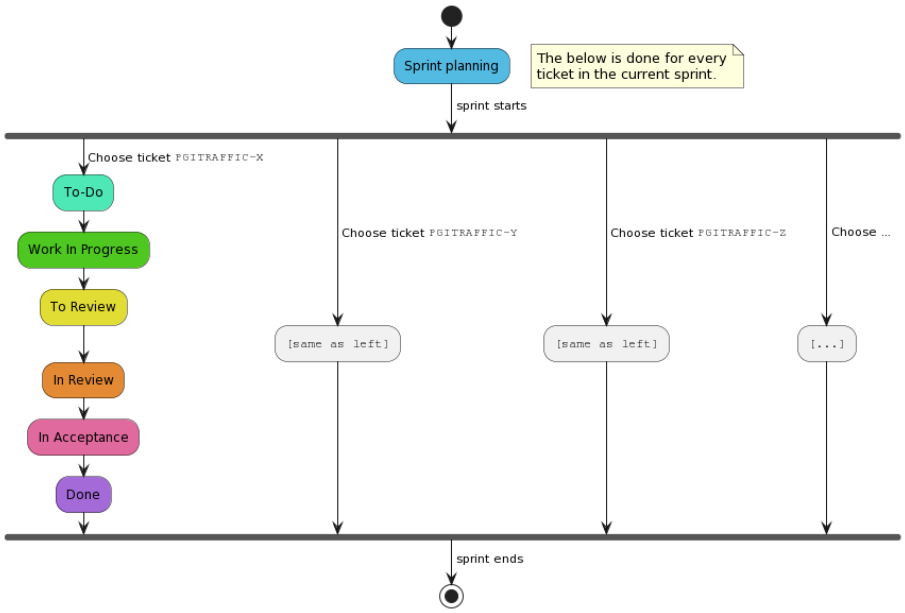

Figure 20: Overview of how the work on items in a sprint is done.

Since some of these steps are complex in nature, it is important to clearly define their respective workflows. This is done by the following diagrams, with continued usage of the color coding as seen above. The „Sprint planning“ workflow is described in Figure 22. The activity „To-Do“ is empty, as this step only consists of waiting for any ticket-related work to start, thus requiring no well-defined workflow. The workflow described in Figure 23 shows how tickets that are in progress should be worked on. The workflow for „To Review“ and „In Review“ is shown in Figure 24 and the workflow for finalizing a ticket is described in Figure 25.

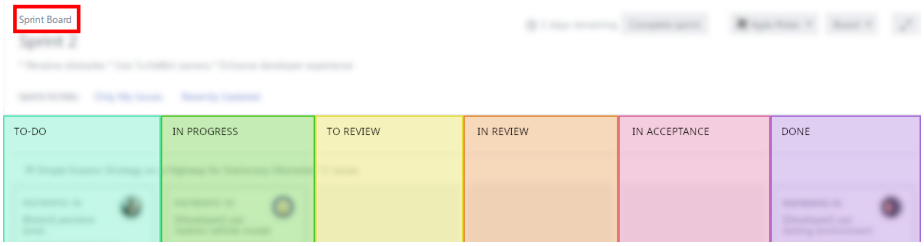

Figure 21: The states of an issue as represented in the Jira board.

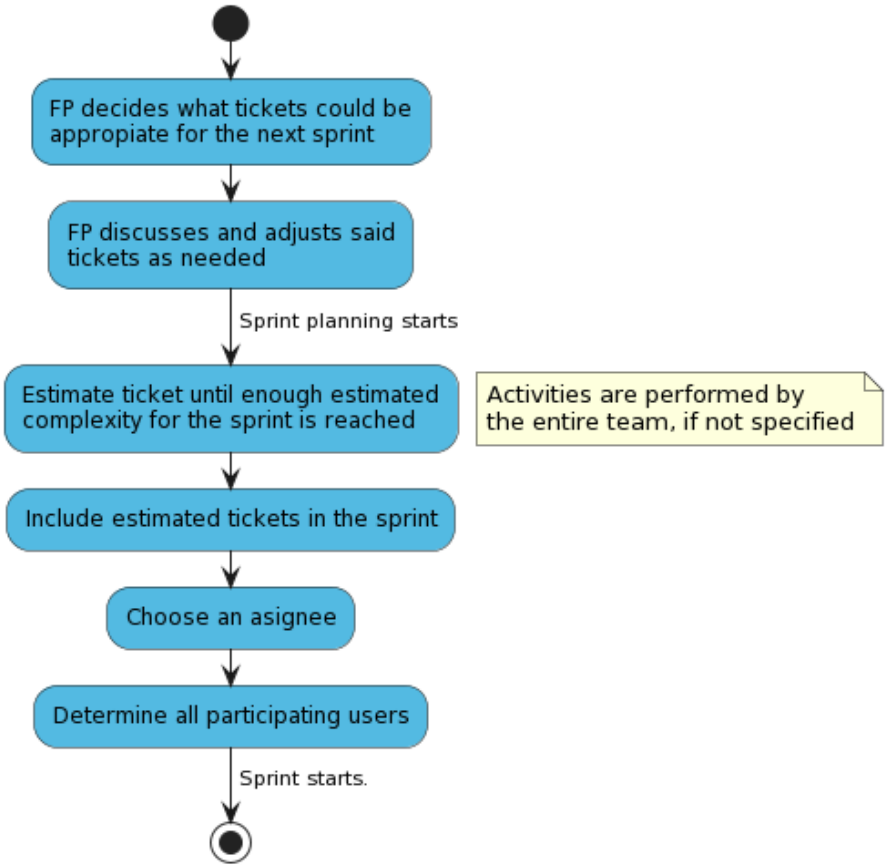

Figure 22: Sprint planning workflow

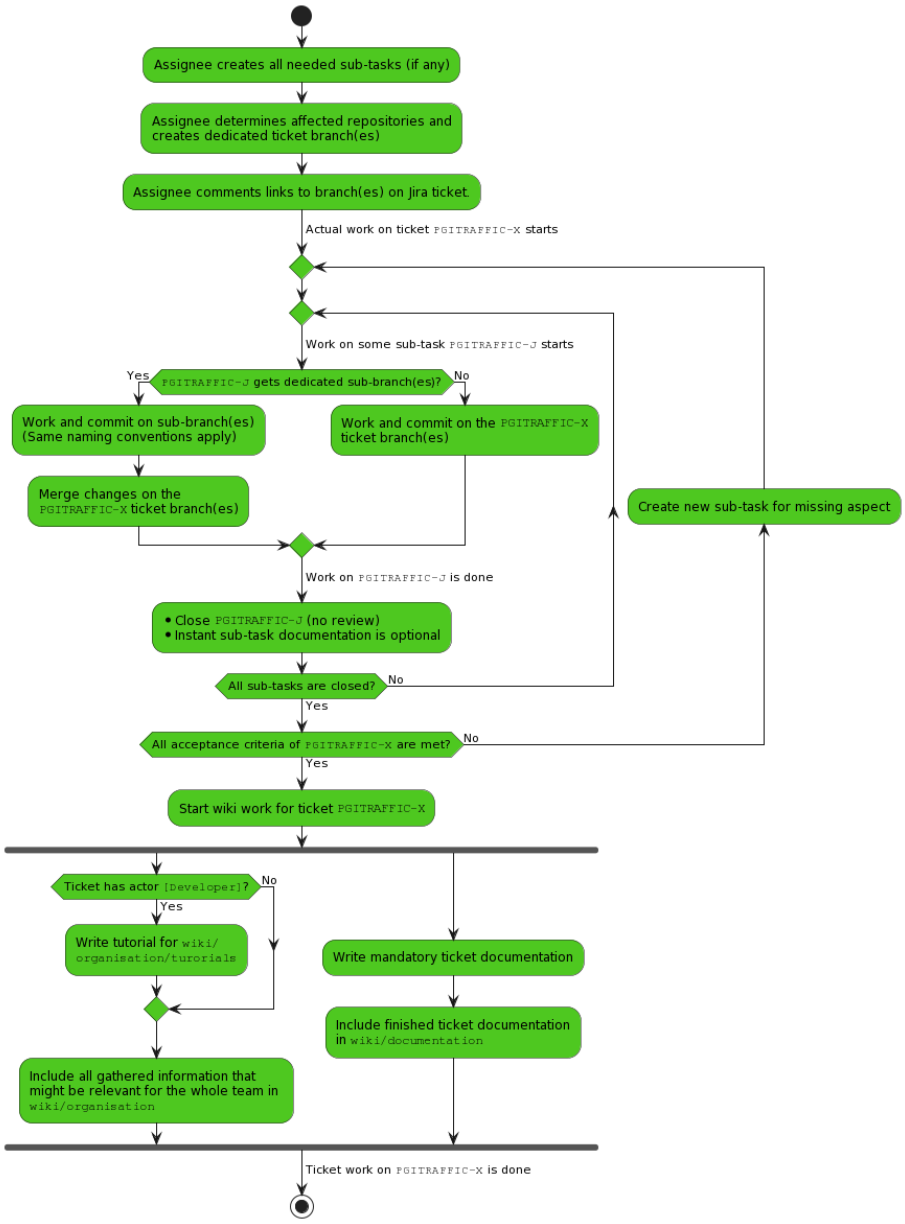

Figure 23: Work in Progress workflow

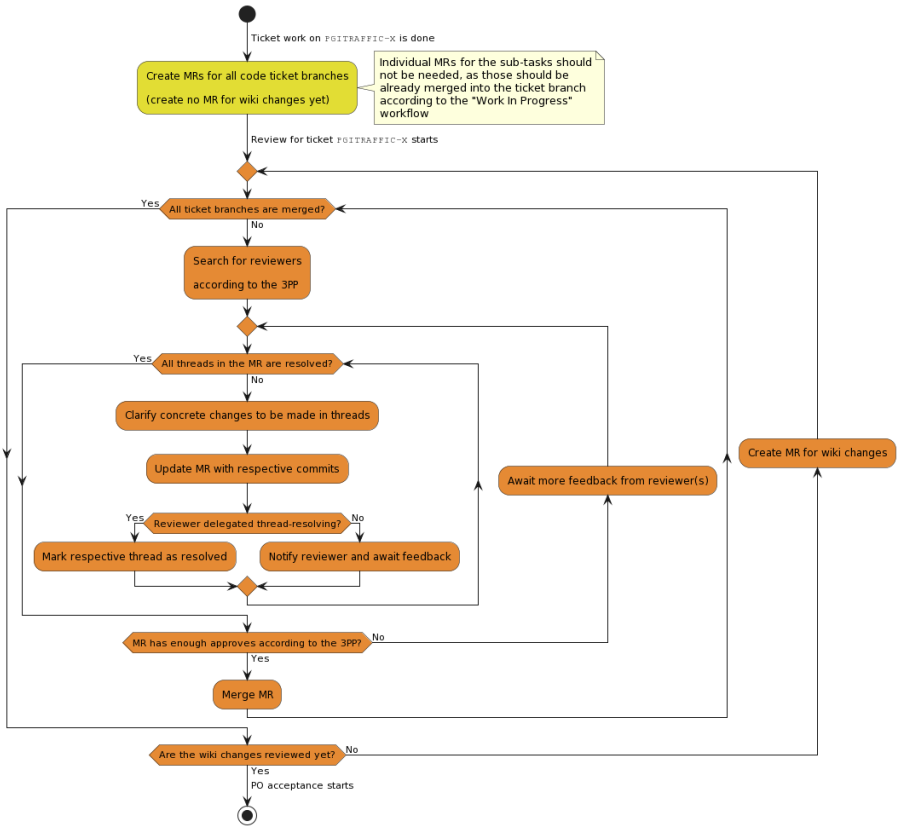

Figure 24: To Review and In Review workflow

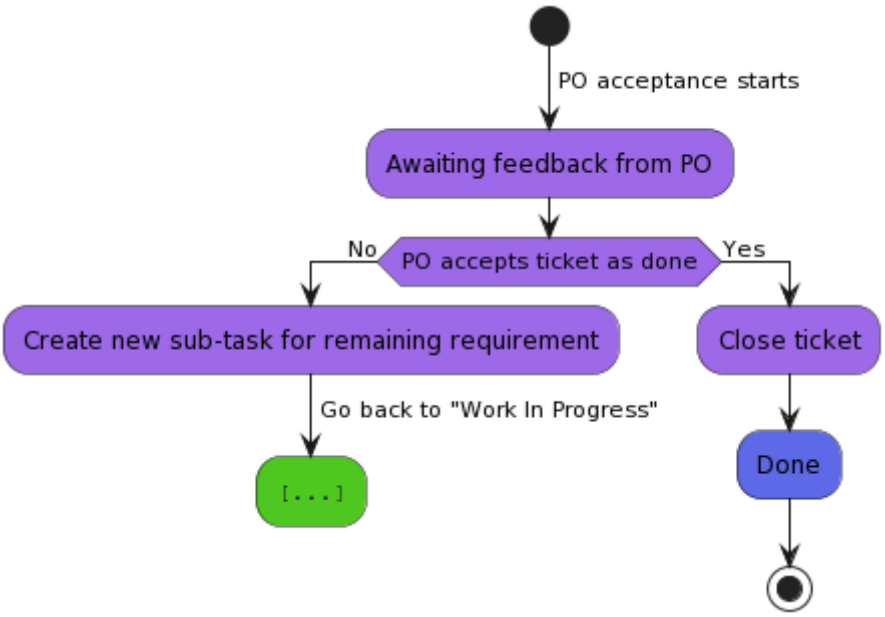

Figure 25: In Acceptance workflow

12.4 Defintion of Done

A ticket is considered done if the following requirements are fulfilled:

- Functionality implemented
- Reproducibly tested
- Documented
- At least three persons were involved in implementation and review, at least one of them is only a reviewer
- All acceptance criteria are met
- Accepted by PO or BE

12.5 Roles

The project group consists of eleven students. Each member is a developer, but some also fulfill different roles or focus on certain topics. These roles with the member's names inside this project are listed below.

- Scrum Master (Carl Schneiders)

  - Ensures conformity to scrum practices
  - Maintains the team's processes and takes care of removing obstacles in the process
  - Organizes retrospectives.
- Product Owner (Marie Marken)
  - Creates and maintains the product vision in consultation with the group and other interested parties.
  - Communicates with BTC ES, Foundations and Applications of Systems of Cyber-Physical-Systems and Distributed Control in Interconnected Systems
  - Maintains the backlog and organizes feature planning
  - Leads sprint planning and sprint review
- Business Engineer (Lasse Heckelmann)
  - Supports the Product Owner in her tasks
  - Keeps track of the project
  - Provides a point of contact for specialized questions
- Documentation Steward (Nellson Eilers)
  - Keeps track of the internal wiki
  - Makes sure that everyone documents their work
  - Ensures that conventions regarding the documentation are adhered to
- Infrastructure (Malte Grave)
  - Maintains the server infrastructure
  - Makes sure that everyone can work
  - Also offers technical support
- Code Steward (Jan-Magnus Monenschein)
  - Ensures that the code quality is of the desired level
  - Specifies rules and principles for working on the code base
  - Helps with all things CI/CD
  - Helps with configuring and working with development tools
- Technical Lead (Simon Struck)
  - Has an oversight over the whole system
  - Responsible for creation/management of datatransfer protocols

  - Evaluates technical feasibility
- Quality Analysis (Filip Wojciak)
  - Ensures product functionality
  - Tasked with testing of the product
  - Ensures fulfillment of requirement
- Developer (Julia Debkowski)
  - Assists with software development
  - Keeps track of the project's progress
- Software Architect (Stefan Gerber)
  - Maintains the architecture
  - Is a contact for architectural questions
- PR work (Paulina Kowalska)
  - Responsible for public work
  - Responsible for planning events

12.6 Tools

For easier collaboration, using a few tools proved to be essential. Below, some of these tools are presented.

12.6.1 Jira

Jira is a popular project management and issue tracking tool developed by Atlassian. Jira helps to manage tasks efficiently, maintain transparency, adapt to different project methodologies and collaborate effectively.
Jira allows teams to create, track, and manage issues, tasks, bugs, and user stories. This helps in maintaining a clear and organized list of work items, making it easier to prioritize and address them. Also, Jira supports agile methodologies like Scrum. It provides features such as sprint planning, backlog management, and burndown charts to facilitate agile processes.
Furthermore, Jira is highly customizable. This enables the ability to have custom workflows, issue types, and fields to tailor it to a project's specific needs.
Another important aspect is that Jira can integrate with a wide range of tools, including source code repositories (e.g. Gitlab), CI/CD pipelines and more.

12.6.2 Discord

Discord is used for communication within the team. A custom bot called Hugo is used, which partially automates processes. Particularly, he reminds the group of the internal weekly deadline, helps with the estimation process of user stories and can be used to list current merge requests and their review status.

12.6.3 Gitlab

Versioning is essential. Gitlab is used for this purpose. Repositories for the following projects exist.

- the internal wiki
- the website
- everything related to public relations
- the project report
- the server configuration
- and of course the TurtleCar implementation itself

12.6.4 Google Calendar

To keep track of important dates Google Calendar is used. Here all appointments as well as vacations are entered.

12.6.5 Etherpad

Etherpad is used to share notes and to keep the agenda for meetings.

13 Public Relations

In this section, the presentation to the public will be addressed. This will cover tasks performed during participation in events like the FleiWa, as well as the management of the project group's online presence, including a website and Instagram account.

13.1 Quartierstag

Figure 26: Presentation at the Quartierstag

At the „Alte Fleiwa“ neighborhood, as part of its 100th-anniversary celebration the „Quartierstag“ was held. Here a first major milestone, the Lane Keeping Assistant, was presented. This can be seen in Figure 26. During this event, local businesses, research institutions, organizations, and municipal offices provided insights into their work. More information can be found on the following link: `https://quartierstag.de/` On behalf of the University and BTC-ES, current findings were presented, a live demonstration was offered, a poster as seen in Figure 27 was created and the opportunity to examine hardware and software, including the Visualizer was given.

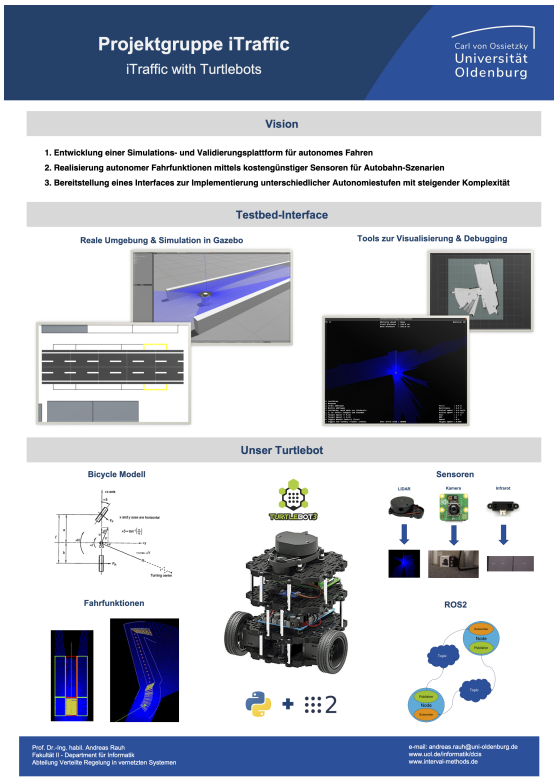

Figure 27: Overview of the poster for the Quartierstag.

13.2 Website

In today's world, it is of paramount importance to establish an online presence. To this end, a digital presence was created. First, an Instagram account exists, which will be actively curated in the near future. What is already accessible by the public is the website. The project's website displays the most important information for the public. Its public domain is `https://itraffic-uol.de/`.

The website is a good way to document the progress being made over time and to also show it to stakeholders. The content should primarily address the goals of the project group. Progress should be documented as well as challenges to avoid or not to repeat possible mistakes. The team represents the basic building block of the project group and is therefore presented. This way, even strangers who have nothing to do with the project group can build a good understanding of it.

13.2.1 Dependencies

The following tools are used to create the website:

- Jekyll (Static Site Generator)
- Minimal Mistakes Theme for Jekyll
- Ruby Bundler in order to manage the projects dependencies
- Gitlab CI/CD for building and deploying the site automatically

13.2.2 Content Review Policy

Since the content affects everyone and appears online, changes should be approved by everyone in advance. Joint reviews are mandatory.

13.3 Email

The teams public email address is: team@itraffic-uol.de

13.4 Instagram

The project group's Instagram channel can be found here: `https://www.instagram.com/pg_itraffic/`

The Instagram channel is a bit more informal and is intended to represent the project group away from the achievement of goals. For this purpose, insights into meetings but also social events can be shared. Regularity to post is secondary.

References

[1] Author automaticaddison. *Sensor fusion using the robot localization package - Ros 2*. Dec. 2021. URL: `https://automaticaddison.com/sensor-fusion-using-the-robot-localization-package-ros-2/` (visited on 10/08/2023).

[2] Uli Baumann. *Die Räder stehen fast quer*. July 2022. URL: `https://www.auto-motor-und-sport.de/tech-zukunft/zf-easyturn-achse-extrem-lenkung/` (visited on 10/08/2023).

[3] More BHP. *VW MK7 Golf GT 2.0TDI 150 ECU Remap*. URL: `https://www.more-bhp.com/volkswagen-golf-remapping/vw-mk7-golf-gt-20tdi-150-ecu-remap.html` (visited on 10/08/2023).

[4] *Black Python Formatter GitHub Repository*. Oct. 2023. URL: `https://github.com/psf/black` (visited on 10/07/2023).

[5] Philipp Borchers et al. *Realtime Controlled Cooperative Autonomous Racing System next generation*. Checked 2023-10-05. Apr. 2018. URL: `https://uol.de/f/2/dept/informatik/download/lehre/PGs/PG-RCCARS.pdf` (visited on 10/08/2023).

[6] Nikolai Bräuer et al. *Realtime Controlled Cooperative Autonomous Racing System*. Nov. 2016. URL: `https://uol.de/f/2/dept/informatik/download/studium/pg/PG_RCCARS.pdf` (visited on 10/05/2023).

[7] *Bremsen*. URL: https://vorschriften.bgn-branchenwissen.de/daten/dguv/70/19.htm (visited on 10/08/2023).

[8] *Bremswege im Vergleich*. Oct. 2019. URL: https://www.adac.de/rund-ums-fahrzeug/autokatalog/autotest/bremswege-vergleich/ (visited on 10/08/2023).

[9] Bundesrepublik Deutschland. *Straßenverkehrsordnung*. 2013. URL: https://www.gesetze-im-internet.de/stvo_2013/ (visited on 10/08/2023).

[10] Rüdiger Cordes. *cw-Werte*. 2022. URL: http://rc.opelgt.org/indexcw.php (visited on 10/08/2023).

[11] Yan Ding. *Simple Understanding of Kinematic Bicycle Model*. Nov. 2021. URL: https://www.shuffleai.blog/blog/Simple_Understanding_of_Kinematic_Bicycle_Model.html (visited on 10/08/2023).

[12] *DSG Shift Time*. June 2007. URL: https://www.vwvortex.com/threads/dsg-shift-time.3311040/ (visited on 10/08/2023).

[13] PG EmBrAAC. *Projektgruppe Emergency Braking Assistant for fully Autonomous Cars*. Sept. 2019.

[14] Brian Fitzgerald and Klaas-Jan Stol. "Continuous software engineering: A roadmap and agenda". en. In: *Journal of Systems and Software* 123 (Jan. 2017), pp. 176–189. ISSN: 01641212. DOI: 10.1016/j.jss.2015.06.063. URL: https://linkinghub.elsevier.com/retrieve/pii/S0164121215001430 (visited on 10/07/2023).

[15] ROS 2 Real-Time Working Group. *Raspberry Pi image with ROS 2 and the real-time kernel*. 2023. URL: https://github.com/ros-realtime/ros-realtime-rpi4-image (visited on 10/04/2023).

[16] International Organization for Standardization. *Intelligent transport systems — Lanekeeping assistance systems (LKAS) —Performance requirements and testprocedures*. Tech. rep. 2014. URL: https://www.iso.org/obp/ui/en/#iso:std:iso:11270:ed-1:v1:en (visited on 10/08/2023).

[17] PG iTraffic. *TurtleBot 3 Image Builder*. 2023. URL: https://gitlab.itraffic-uol.de/itraffic/TurtleBot3-image-builder (visited on 10/04/2023).

[18] Holger Krekel. *pytest Documentation*. docs.pytest.org, Oct. 2023. URL: https://buildmedia.readthedocs.org/media/pdf/pytest/latest/pytest.pdf (visited on 10/04/2023).

[19] *Lane departure warning system*. URL: https://en.wikipedia.org/wiki/Lane_departure_warning_system (visited on 10/05/2023).

[20] Baurzhan Muftakhidinov Mark Mitchell and Tobias Winchen et al. *Engauge Digitizer Software*. URL: http://markummitchell.github.io/engauge-digitizer (visited on 10/08/2023).

[21] *mockito-python GitHub Repository*. Oct. 2023. URL: https://github.com/kaste/mockito-python (visited on 10/04/2023).

[22] Quang-Cuong Pham. "Trajectory Planning". en. In: *Handbook of Manufacturing Engineering and Technology*. Ed. by Andrew Y. C. Nee. London: Springer London, 2015, pp. 1873–1887. ISBN: 978-1-4471-4669-8 978-1-4471-4670-4. DOI: 10.1007/978-1-4471-4670-4_92. URL: https://link.springer.com/10.1007/978-1-4471-4670-4_92 (visited on 10/04/2023).

[23] *Ruff Python Linter GitHub Repository*. Oct. 2023. URL: https://github.com/astral-sh/ruff (visited on 10/07/2023).

[24] Scrum.org. *What is Scrum?* 2023. URL: https://www.scrum.org/learning-series/what-is-scrum (visited on 10/08/2023).

[25] Forschungsgesellschaft für Straßen- und Verkehrswesen, ed. *Richtlinien für die Markierung von Straßen. Teil A: Autobahnen*. ger. Ausgabe 2019. FGSV 330A. Cologne: Forschungsgesellschaft für Straßen- und Verkehrswesen e.V, 2019. ISBN: 978-3-86446-251-1.

[26] *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. Apr. 2021. URL: https://www.sae.org/standards/content/j3016_202104/ (visited on 10/05/2023).

[27] *Vehicle acceleration and maximum speed modeling and simulation*. URL: https://x-engineer.org/vehicle-acceleration-maximum-speed-modeling-simulation/ (visited on 10/08/2023).