POLITECNICO DI TORINO

Master's Degree in Automotive Engineering

Master's Degree Thesis

Lane detection via Camera and Map Matching

Supervisors

Prof. Albertengo Guido

Prof. Reuss Hans-Christian (Uni. Stuttgart)

Candidate

Festa Andrea

March 2024

Abstract

The aim of this work is to improve an existing lane recognition system for autonomous driving. Such system lacks reliability when facing intersections, roundabouts or complex road segments. To overcome this failure, it is possible to make use of the concept of map-matching which exploits the GPS sensor data to provide a high-level guidance to the vehicle, enabling the existing method to cope with a large variety of road environments. Map-matching is required due to the poor precision and accuracy of the sole Global Positioning System location which is not sufficient for autonomous driving tasks. The challenge came from the research of a time and memory-efficient algorithm suitable for the project's requests.

The proposed method consists in determining a high-level trajectory between two geographical locations, complying to the criterion of shortest possible path, which is improved and refined by a series of further steps. The existing camera-based lane recognition system is in this way used at low level only. With this enhancement, the failures detected before in correspondence of complex road scenarios are no longer an issue since the vehicle is able to proceed relying on the reference trajectory only.

The backbone of this method is surely simplicity: it has been an essential criterion in the design of this algorithm enabling reliability and high-speed operation at the same time. Another key factor came from the availability of predefined online libraries used to ease and speed up the process of functions design.

After this first phase, the method has been redeployed to the Robotic Operating System environment to evaluate its performances in an autonomous-driving simulation scenario. Particular attention has been devoted to the communication between the various components of the algorithm, relying also in this case either to predefined and user-defined interfaces.

Acknowledgements

Questo lavoro è dedicato a chi, in un modo o nell'altro mi ha seguito nel mio percorso.

A chi purtroppo non ha potuto mantenere la promessa di esserci in questa giornata speciale, ma mi ha sempre assistito da Lassù.

A chi mi ha da sempre ispirato e successivamente dato la forza per non mollare mai.

A chi mi ha continuamente sostenuto ed appoggiato in qualsiasi scelta abbia preso.

A chi, con la sua presenza costante, mi ha dato motivo di credere nel futuro, non smettendo un solo istante di credere in me.

A chi, nel suo piccolo, mi ha riempito di gioia il cuore facendomi sentire un esempio da seguire.

A chi mi conosce letteralmente da una vita ed ha sempre fatto parte di questo percorso e a chi ho appena conosciuto, che mi ha dato più di quanto potessi mai immaginare.

Table of Contents

List of Tables

List of figures

Nomenclature

2D/3D:	Two/Three - dimensional
AV:	Autonomous Vehicle
CAN:	Controller Area Network
CNN:	Convolutional Neural Network
DLA:	Deep Layer Aggregation
ECU:	Electronic Control Unit
GNSS:	Global Navigation Satellite System
GPS:	Global Positioning System
IMU:	Inertial Measurement Unit
LIN:	Local Interconnect Network
$\mathbb{R}$	: Set of Real Numbers
RCNN:	Region based Convolutional Neural Network
RESA:	Recurrent Feature Shift-Aggregator
RF:	Reference Frame
SCNN:	Spatial Convolutional Neural Network
SLAM:	Simultaneous Localization and Mapping
UFLD:	Ultra-Fast Structure-aware Deep Lane Detection
VGG:	Visual Geometry Group
YOLO:	You Only Look Once

Introduction

The automotive sector plays a pivoting role in today's industry panorama due to its implication in many worldwide concerns, such as environmental protection, economics, and road safety. To tackle all these challenges, this field is in constant evolution and one of the central topics of research is for sure Autonomous Driving.

This technology envisions a world where vehicles are able to perceive the surrounding environment, make intelligent decisions, and navigate autonomously from point A to point B without the need for human intervention. This allows to eliminate the so-called "human error", consequently reducing the occurrence of road accidents, mainly caused by it. Furthermore, driverless vehicles increase the traffic flow capacity in congested areas and the pollutant emissions related to the transport sector.

Another often neglected application can be found in the development phase of new vehicles: driverless cars can perform any test on behalf of human drivers with a much higher level of consistency and in shorter times.

To achieve these promising results, cutting-edge technologies are needed in the field of perception sensors and processing units which must be well integrated and cooperated to accomplish all the subtasks leading to autonomous driving.

Among these subtasks, one of the most important is devoted to lane recognition. Its aim is to provide to the central unit the information related to the lane in which the vehicle is traveling (road markings, left and right boundaries, presence of obstacles, etc.) so that it can safely follow the defined path.

This task is relatively easy in highways or roads without intersections but when it comes to roundabouts or complex road geometries, the lane recognition system may encounter some difficulties.

One of the solutions of this problem is the fusion of camera and GPS signals to provide a robust perception knowledge to the algorithm and let it perform well in a large variety of environments.

Chapter 1

Autonomous driving evolution

In this first chapter, a brief overview of the development of autonomous vehicles is presented. A deep focus on the Advanced Driver Assistance Systems then follows, supported by the definitions and the regulations provided by the Society of Automotive Engineers. In the end, the state-of-the-art technological enablers are analyzed one by one.

1.1 History of Autonomous Vehicles

The concept of Autonomous Vehicle (AV) dates way back before the concept of the car itself. In fact, in the 16th century Leonardo Da Vinci designed a small, three-wheeled, self-propelled cart which is today referred not only as the first driverless vehicle but also the first robot of any kind.

It took a long time before the next example of a driverless car showed up, when in 1925 the inventor Francis P. Houdina proposed a radio-controlled vehicle riding in the streets of New York. Unfortunately, the car crashed into another vehicle and the project lost credibility soon after.

The Big Apple gave birth to another ambitious project in 1939 when the industrial designer Norman Bel Geddes showcased a mobility concept based on magnetized freeways. Semi-autonomous vehicles were supposed to travel on them, exploiting an electromagnetic field capable of controlling a current flow and act on the commands of the vehicle.

In the next years academic researches produced a series of concepts as the *Standford Cart* of 1961 (a small autonomous cart designed for the Moon capable to detect obstacles and follow a predefined path), the first autonomous passenger vehicle produced by the Japanese University of Tsukuba (the first employment of cameras in a driverless vehicle) and the *Eureka PROMETHEUS Project* of the early '90s (consortium of universities and car makers culminated in 1994 with a thousand kilometer ride on the Parisian highways in self-driving mode).

Through its Defense Advanced Research Projects Agency (DARPA), the U.S. Department of Defense promoted the research on AVs but the floodgates really opened when the Agency launched a series of competitions for driverless cars on desert roadways for 150 miles. In the first attempt of 2004, no one succeeded in completing the race, but the following year five contestants arrived at the end, led by the Standford University's car.

By the mid-2010s a new trend started to emerge, namely the cooperation between IT leaders like Apple or Google with automaker incumbers to tackle the challenges brought by AVs [1]. It yielded to the presentation of key products such as Waymo's autonomous taxi in San Francisco in 2016 [2]. Despite its initial hype, real autonomy proved to be more difficult to achieve than originally planned: as of 2022 leading companies in the driverless panorama such as Aurora and Intel with its Mobileye have lost up to 75 billion dollars, causing a chasm that has almost taken over the sector [3]. Nevertheless, the rush recently seems to be started over not only in the Silicon Valley but also in many R&D research centers all over the Europe. On the other hand, besides the technological challenges, customers psychology must surely be taken into account when considering the spreading of AV. Public acceptance and trust are key factors for a large-scale diffusion of driverless cars and they can be achieved only with honest transparency about the potential risks and the good outcomes that the adoption of such technology may bring with itself. Another remarkable aspect is represented by the blame in case of accident. Legal issues are involved in this topic and the recipient of the law implications deriving from an accident is still difficult to determine with clarity [4].

In the end, as of today, SAE level 3[1] vehicles are ready for commercialization, but it seems too early to speculate about the diffusion of cars with a higher level of automation [5].

1.2 SAE levels

The Society of Automotive Engineers (SAE) provides a comprehensive set of definitions and regulations that represent a pillar in the field of self-driving vehicles. It developed 5 levels of driving automation, with the number of tasks demanded to the vehicle rising from level 0 to 5.

1.2.1 Preliminary definitions

To understand the concepts presented in Table **1.1** and Table **1.2**, it is necessary to provide a few preliminary definitions, as reported in [6].

- *Automated Driving System* (*ADS*): "The hardware and software that are collectively capable of performing the entire DDT on a sustained basis, regardless of whether it is limited to a specific operational design domain (ODD); this term is used specifically to describe a Level 3, 4, or 5 driving automation system."
- *Dynamic Driving Task* (*DDT*): "All of the real-time operational and tactical functions required to operate a vehicle in on-road traffic [...] including, without limitation, the following subtasks:
  1. Lateral vehicle motion control via steering (operational).
  2. Longitudinal vehicle motion control via acceleration and deceleration (operational).
  3. Monitoring the driving environment via object and event detection, recognition, classification, and response preparation (operational and tactical).

[1] See the following chapter for more details

4. Object and event response execution (operational and tactical).[2]
5. Maneuver planning (tactical).
6. Enhancing conspicuity via lighting, sounding the horn, signaling, gesturing, etc. (tactical)."

- *DDT Fallback*: "The response by the user to either perform the DDT or achieve a minimal risk condition [...] after occurrence of a DDT performance-relevant system failure(s), or [...] upon operational design domain (ODD) exit, or the response by an ADS to achieve minimal risk condition, given the same circumstances."
- *Operational Design Domain* (*ODD*): "Operating conditions under which a given driving automation system or feature thereof is specifically designed to function, including, but not limited to, environmental, geographical, and time-of-day restrictions, and/or the requisite presence or absence of certain traffic or roadway characteristics."

[2] Both tasks (3) and (4) are referred to collectively as *object and event detection and response* (*OEDR*)

1.2.2 SAE levels tables

	Level	Name	Narrative definition	DDT		DDT Fallback	ODD
				Sustained Lateral and Longitudinal vehicle motion control	**OEDR**		
Driver Performs Part or All of the DDT							
	0	No Driving Automation	The performance by the driver of the entire DDT, even when enhanced by active safety systems	Driver	Driver	Driver	n/a
Driver Support	1	Driver Assistance	The sustained and ODD-specific execution by a driving automation system of either the lateral or the longitudinal vehicle motion control subtask of the DDT (but not both simultaneously) with the expectation that the driver performs the remainder of the DDT.	Driver and System	Driver	Driver	Limited

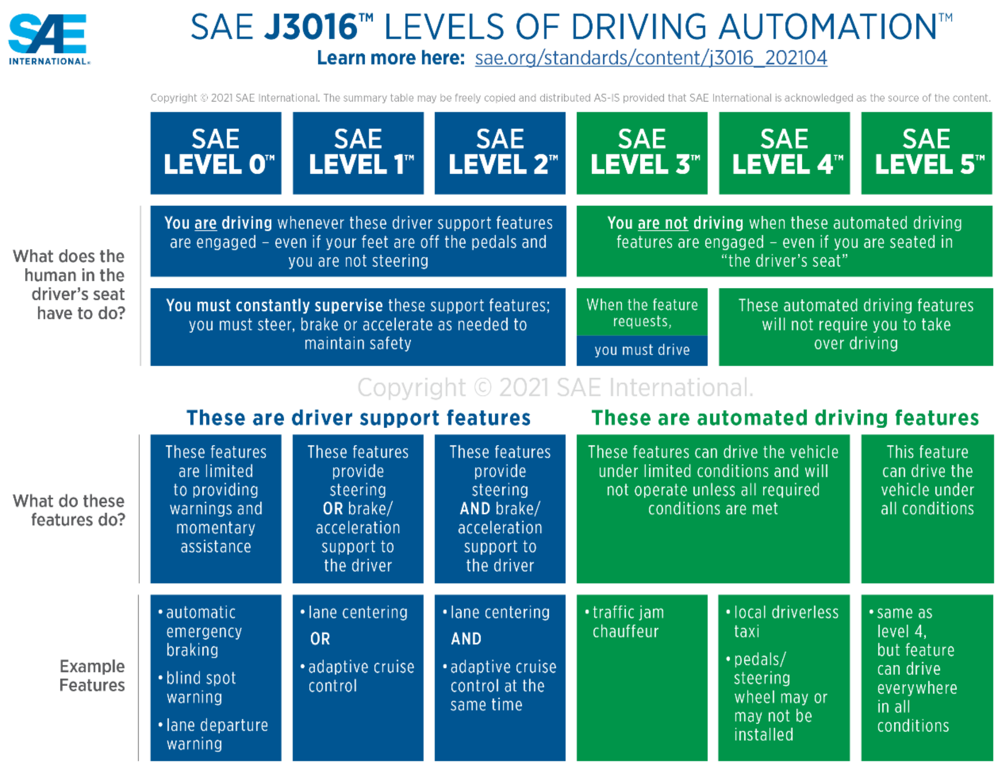
2	Partial Driving Automation	The sustained and ODD-specific execution by a driving automation system of both the lateral and longitudinal vehicle motion control subtasks of the DDT with the expectation that the driver completes the OEDR subtask and supervises the driving automation system.	System	Driver	Driver	Limited

Table 1.1: Summary of levels of driving automation – [6]

	Level	**Name**	**Narrative definition**	**DDT: Sustained Lateral and Longitudinal vehicle motion control**	**DDT: OEDR**	**DDT Fallback**	**ODD**
	ADS ("System") Performs the Entire DDT (While Engaged)						
Automated driving	3	Conditional Driving Automation	The sustained and ODD-specific performance by an ADS of the entire DDT with the expectation that the DDT fallback ready user is receptive to ADS issued requests to intervene, as well as to DDT performance relevant system failures in other vehicle systems, and will respond appropriately.	System	System	Fallback ready user (becomes the driver during fallback)	Limited

Automated driving	4	High Driving Automation	The sustained and ODD-specific performance by an ADS of the entire DDT and DDT fallback without any expectation that a user will need to intervene.	System	System	System	Limited
	5	Full Driving Automation	The sustained and unconditional (i.e., not ODD-specific) performance by an ADS of the entire DDT and DDT fallback without any expectation that a user will need to intervene.	System	System	System	Unlimited

Table 1.2: Summary of levels of driving automation (continued)



SAE J3016™ LEVELS OF DRIVING AUTOMATION™
Learn more here: sae.org/standards/content/j3016_202104

	SAE LEVEL 0™	SAE LEVEL 1™	SAE LEVEL 2™	SAE LEVEL 3™	SAE LEVEL 4™	SAE LEVEL 5™
What does the human in the driver's seat have to do?	**You <u>are</u> driving** whenever these driver support features are engaged – even if your feet are off the pedals and you are not steering			**You <u>are not</u> driving** when these automated driving features are engaged – even if you are seated in "the driver's seat"		
	You must constantly supervise these support features; you must steer, brake or accelerate as needed to maintain safety			When the feature requests, you must drive	These automated driving features will not require you to take over driving	
	These are driver support features			These are automated driving features		
What do these features do?	These features are limited to providing warnings and momentary assistance	These features provide steering **OR** brake/ acceleration support to the driver	These features provide steering **AND** brake/ acceleration support to the driver	These features can drive the vehicle under limited conditions and will not operate unless all required conditions are met		This feature can drive the vehicle under all conditions
Example Features	• automatic emergency braking • blind spot warning • lane departure warning	• lane centering **OR** • adaptive cruise control	• lane centering **AND** • adaptive cruise control at the same time	• traffic jam chauffeur	• local driverless taxi • pedals/ steering wheel may or may not be installed	• same as level 4, but feature can drive everywhere in all conditions

Figure 1.1: SAE levels of automation - Source [6]

1.3 ADAS

Advanced Driver Assistance Systems (ADAS) cooperate and help the drivers at the wheel, increasing their perception capability, reducing the reaction time, or detecting the onset of drowsiness. On the majority of Countries, the legislative framework allows SAE level 2 of automation only, but much more is yet to come, and these systems are paving the way for autonomous vehicles.

The following section will give an overview of the most popular systems.

1.3.1 Blind Spot Detection

This system increases the driver's perception capability to detect obstacles in the vehicle's blind spots (dark blue areas in Figure 1.2) in order to increase the safety during lane changes.

The basic variant of Blind Spot Detection can be implemented with ultrasonic sensors or one single corner radar sensor. They monitor the blind area and alert the driver in case of presence of an obstacle with a visual signal on the side mirror.

The advanced variant employs two corner radar sensors embedded in the rear bumper. The signals coming from them are merged to provide a complete picture of the traffic behind the car, alerting the driver even in case of fast-approaching vehicles.

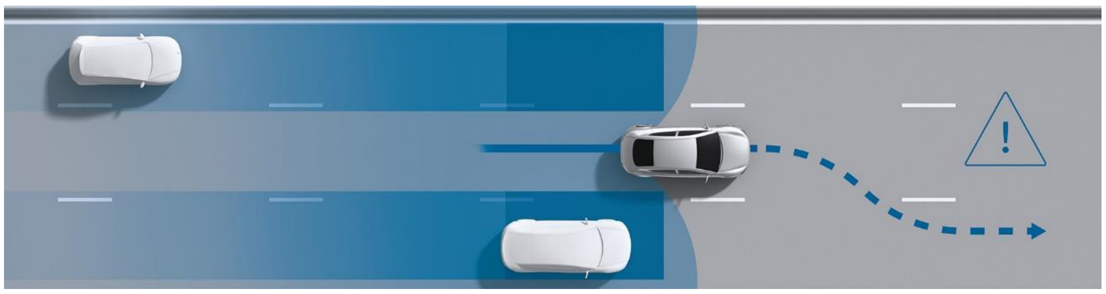


Figure 1.2: Blind Spot areas - Source [69]

1.3.2 Driver drowsiness detection

It can detect the driver's fatigue or microsleep from the steering wheel movements. Through a steering angle-sensor and a complex algorithm that takes into account many other parameters, the system is able to warn the driver that he or she is better have a break. A typical sign of fatigue is represented by a constant and slow steering, followed by a slight, yet quick and abrupt maneuver to keep the vehicle on track.

1.3.3 Traffic Sign Recognition

Traffic signs display useful information for the driver, but at the same time, they may be difficult to be properly detected. This system uses an all-purpose camera (usually installed on the windscreen of the vehicle) to recognize and classify round, rectangular or triangular road signs. Then, the corresponding signal is displayed on the cluster and easily read by the driver.

Traffic sign recognition proves particularly useful when detecting the start and the end of stretches where speed limits are in force.

1.3.4 Forward Collision Warning

This system is capable of warning the driver in case it is approaching a rear-end collision with an obstacle in its forward path. Camera and radar detect the possibility of a collision when the distance between the vehicle and the obstacle in front is closing too quickly and the relative speeds are too different from each other. The driver is alerted with acoustic, visual, or haptic signals and when possible, emergency braking is activated.

1.3.5 Adaptive Cruise Control

The aim of this driver assistance system is to maintain a minimum safe distance from the preceding vehicle by adjusting the speed set by the cruise control. A front radar monitors the traffic ahead of the vehicle and if no obstacles are detected, the system maintains the driver's desired speed. If otherwise, a slower preceding car is revealed, ACC slightly reduces the speed so as to keep a safe distance from it. In case the vehicle in front accelerates again, the ACC increases the speed until it reaches the one set by the driver. This system may be improved by implementing a multi-purpose camera that allows to detect with larger advance another vehicle entering in driver's own lane and act consequently.

1.3.6 Intersection Turn Assistance

Another system that leverages on the operation of a radar and a camera is the Intersection Turn Assistance. Its goal is to warn the driver (or when possible, execute an emergency braking maneuver) in case of danger when performing a left turn. It is demonstrated that this maneuver causes a large number of accidents because the drivers usually underestimate the speed of the vehicle coming from the opposite direction or are not aware of the presence of vulnerable road users[3] in the intersection.

1.3.7 Lane Keeping Assist

Tiredness or distraction may lead to unintentional lane departure with consequent severe danger for the driver and the other road users. Lane Keeping Assist (LKA) uses a video camera to detect the lane boundaries ahead (road markings, curbs, etc.) and acts on vehicle lateral motion control to keep it on track. In vehicles equipped with electronic power steering, LKA gently, but noticeably, countersteers to maintain the trajectory, otherwise, it intervenes through the Electronic Stability Program (ESP) by braking some wheels and achieving the same effect.

1.3.8 Emergency Braking

This system is reasonably listed as one of the most effective in reducing the number and the severity of road accidents. It employs a camera, a front radar and a corner radar, to achieve a wide-ranging awareness of the surroundings and detect the possible onset of a collision. In this case, the system first alerts the driver with acoustic and visual warnings while it prepares the braking system for an emergency operation. Then, if the driver does not react, it initiates a slight braking and as soon as the brake pedal is pressed, the system provides the correct pressure in the brake circuit to bring the vehicle to a standstill before a collision occurs.

This operation specifically refers to rear-end collision situation. Nevertheless, the Emergency Braking intervenes in many other scenarios, as the backing-up impact. Using corner radar data, it detects the vehicle driving past (possibly not noticed by the driver) and activates the brakes to avoid a collision.

[3] The term refers to pedestrians and cyclists.

1.3.9 Park Assist

The Park Assist system employs ultrasonic sensors installed in the corners of the front and rear bumpers to scan the environment along the side of the vehicle. When sufficient room for a parallel or perpendicular parking is found, the system takes control of the vehicle acting on the steering wheel and provides instructions to the driver who is usually still in control of the accelerator and the brake pedal.

1.4 Sensors

The enabling technology for the achievement of any level of driving automation includes state-of-the-art perception sensors. They consists in ultrasonic sensors, radars, lidars, cameras and inertial measurement units. When it comes to their choice, many parameters must be considered such as accuracy, sampling rate, range, field of view (FoV) and cost to reach the best trade-off in terms of performance and complexity.

In the following section a more detailed description of the aforementioned systems is provided.

1.4.1 Ultrasonic sensor

They exploit not-human-audible ultrasounds in the form of soundwaves with frequency of 40 kHz. These sensors have a range between 15 cm and 6 m and are usually installed inside the bumpers to detect low-profile obstacles or vulnerable road users in proximity of the vehicle.

Their working principle is based on the generation of an ultrasonic pulse which is reflected by the obstacle and then detected back by a piezoelectric receiver.

$$D = {}^{\tau_f}/_2 \cdot v_{sound}(T) \qquad [\ 1.1\ ]$$

The distance (D) from the obstacle is calculated by multiplying half the flight time (τ_f) by the speed of sound at given temperature ($v_{sound}(T)$).

Nevertheless, ultrasonic sensors lack accuracy because if the obstacle's surface is not perpendicular to the sensor itself, the time-to-flight information will record the distance from the nearest point within the emission cone and not from the actual target.

The strongest points of these systems are their low cost, low dimension and high level of integration with the vehicle body.

The main weak point is the sensitivity to dirt.

1.4.2 Radar

The first employment of this technology dates back to the 1940s for aircraft applications. The working principle is similar to the one of ultrasonic sensors, but the radio waves' frequency is much higher (20÷70 GHz), thus the flight time is too short to be simply detected by a microprocessor.

A transmitting module (TX) emits a radio wave that is reflected by the obstacle and then caught by the receiving module (RX). The process of distance evaluation is firstly carried out by the mixer, which evaluates the time delay between the two radio signals, and then the Low Pass Filter (LPF), which outputs a sinusoidal wave known as *IF tone*.

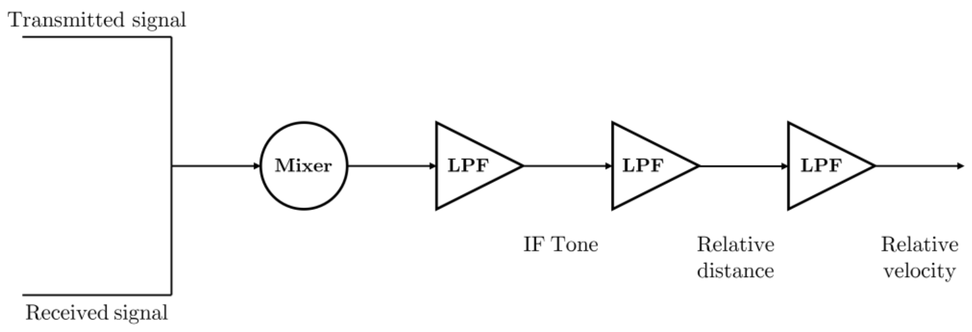


Figure 1.3: Schematic block of radar operation principle

Using a Fast Fourier Transform (FFT) it is possible to identify the different *IF tones* and calculate the related distances. The output of this first step is fed to another FFT block which allows to evaluate the relative velocity with respect to the target.

Nevertheless, relative distance and speed are not enough to properly detect obstacles. The last information that is needed is the angular direction of the target. This is achieved with two RX antennas installed in the same radar device that, exploiting the phase difference between the received radio waves, return the desired information. The accuracy of the measurement can be increased by using more antennas.

The range of radars can extend up to 300 m.

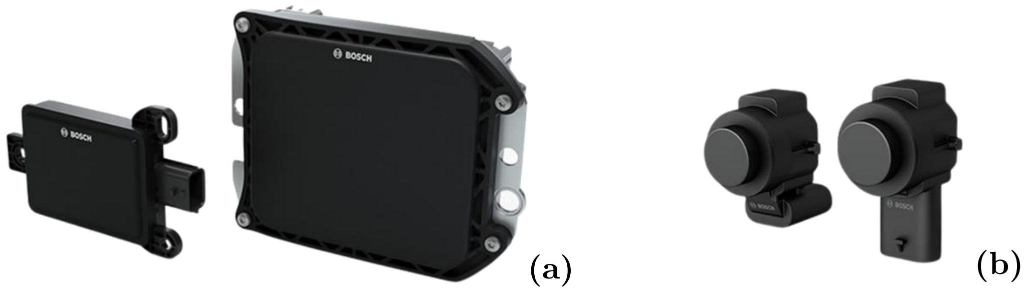


Figure 1.4: radar (a) and ultrasonic sensors (b) - Source [69]

1.4.3 LiDAR

Laser Imaging Detection and Ranging is a method to perceive the surroundings based on a laser pointing an object and measuring the time for the reflected light to return to the receiver. The distance from the reflecting target can be estimated via time-to-flight calculation (as for ultrasonic sensors) or through phase shift evaluation (as for radars).

This process is repeated with a certain frequency to create a detailed map of the surroundings, the so-called *point cloud*. Depending on the arrangement of the sensor, 3D or 2D maps can be generated. The processing unit uses then these data to guide the vehicle in the detected environment.

Taking a look inside the equipment, two main components can be found: the *Vertical Cavity Surface Emitting Laser* (VCSEL) and the *Single Photon*

Avalanche Diode (SPAD). Their tasks are respectively creating the laser arrays and generating an avalanche current when hit by the reflected light.

Lidars have evolved massively in the last years and represent now the major technological enabler for automated driving. Nevertheless, their high cost still represent a major constrain for a wide-scale adoption.

1.4.4 Camera

Cameras have found in the automotive field a large number of applications. They may be employed to support systems complying with *Automotive Safety Integrity Level* (ASIL)[4] requirements, or to simply assist the driver while maneuvering. Additionally, for the majority of the autonomous driving tasks, a proper perception of the surroundings is mandatory, and it is often performed by images. For this reason, cameras play such an important role in this field.

Their operating principle is based on the production of a bidimensional image of the surroundings by capturing the light reflected by 3D objects. Many views from different cameras are anyway needed to properly reconstruct a 3D scene of the environment in which the vehicle is operating. The principle at the basis is known as *stereo vision*. It is carried out by finding corresponding pixels on the images produced by the different devices and subsequently putting in relation the outputs through appropriate geometry calculations. More details about stereo vision are provided by [7].

In the end, it is worth remembering that the acquired image quality is always susceptible to environmental conditions, like weather and level of illumination. Therefore, the visual information retrieved from other sensors is required for robust environmental perception.

[4] They define a risk classification as specified in ISO26262 Standard for functional safety of road vehicles.

1.4.5 Inertial Measurement Unit

Unlike the previously listed items, IMUs are often neglected when considering sensor equipment inside a vehicle while their contribute is pivotal. They allow to measure accelerations and angular rotations up to six degrees of freedom along the three cartesian axes. They can be divided in two main categories: two-sensor-types equipped, and three-sensor-types equipped. The first kind of unit is made up of an accelerometer and a gyroscope for the evaluation of accelerations and rotations respectively. The latter additionally contains a magnetometer for the determination of the yaw angle "thus it can be calibrated to the gyroscope data to improve the big drift issue" [8]. Today's IMUs are capable of valuable performances in terms of precision and responsiveness, working up to 200 Hz all of it inside very small and light packages.

The operating principle differs for acceleration and rotation. In the first case, it relies upon micromechanical structures able to change their capacitance in relation to acceleration indeed. In the second case instead, it exploits the Coriolis principle, i.e. hinges on the inertia force of an oscillating mass placed in a rotating system. This ensures an accurate evaluation since mechanical interferences are greatly reduced by the high resonance frequency of the measuring element.

1.5 Actuators

The second major component of an autonomous driving vehicle technological asset is represented by the actuators. Their role is to translate the electric signals received from the ECU into physical commands to control the vehicle's longitudinal and lateral motion. To do so, the actuators leverage on the electronic nature of most of the commands (electronic power steering, shift by wire, electronically controlled throttle, etc.) which employ wires instead of steel cables, valves, or gears to transmit driver's input to the mechanical organs. Nevertheless, also bulky hundreds of meters of cables may represent a problem in modern vehicles where lightness is a key design criterion. Hence, in today's applications the communication among ECU and the actuators is performed through serial protocols as CAN and LIN buses which employ fewer and lighter electric cables as physical layer.

1.6 Software architecture

Along with hardware (sensors, actuators, mechanical components of the vehicle) the realization of a driverless car would not be possible without the software component. For organization and standardization purposes it has been modeled in different architectures like Stanley [9], Junior [10], Boss [11] and Tongji AC [12]. The five ingredients shared among all these schemes can be identified into *perception, localization and mapping, prediction, planning and control* whose interaction is showed in Figure 1.5.

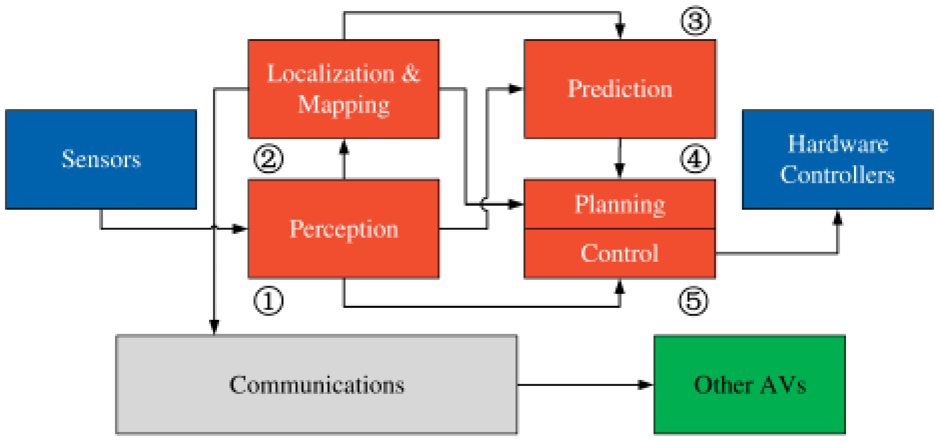


Figure 1.5: Tongji-like software architecture - Source [2]

The perception block oversees the analysis of sensors' output to produce a comprehensive understanding of the environment. Such process can be compared to human vision and usually includes the recognition of lane markings, vehicles or other road users as cyclists or pedestrians. The state-of-the-art technologies in this field can be broken into two main categories: computer vision-based and machine learning-based. The former address visual perception problems by employing geometrical models that show to be the best-fitting ones after an optimization process. The latter provide the solution of a perception problem exploiting data-driven classification or regression models like Convolutional Neural Networks (CNNs). The literature is plenty of examples of perception software based on CNNs such as SegNet [13] and U-Net [14].

The localization and mapping block handles instead the estimation of the position of the vehicle in the space and the recreation of the surroundings. The concept at the basis is the Simultaneous Localization and Mapping (SLAM). Introduced in 1986 [15], SLAM systems can be either filter-based and optimization-based. The first ones iteratively estimate the car pose by continuously integrating the sensor data, while the others tackle the problem by finding correspondences between observations and the map, updating the environment recreation accordingly.

The prediction block is responsible for the analysis of motion patterns to predict future trajectories. Its task involves either the AV itself or the other road users. The model-based prediction systems hinge on kinematic and dynamic mathematical models to predict the future state of a moving object but may fail in long-term prediction [16]. Data-driven-based methods perform better in this context relying upon Artificial Intelligence and High-Performance Computing [17, 18].

The planning module determines viable safe navigation routes based on the output of the previous blocks. Its operation can be broken down into three steps: path, maneuver, and trajectory. The first is a sequence of geometrical points in the space that the vehicle has to follow to avoid collisions and safely reach its destination; the second is a high-level motion characterization process that takes into account also traffic conditions and rules; the last is instead a sequence of AV states.

Finally, the control block sends commands to throttle, brakes and steering to let the vehicle follow the planned trajectory as closely as possible avoiding abrupt actions at the same time. The mostly employed controllers are Proportional Integral Derivative (PID) [19], Linear Quadratic Regulator (LQR) [20] and Model Predictive Control (MPC) [21]. All of them have in common the closed-loop structure and the minimization of the error function paradigm.

Chapter 2

Current state review

In this chapter a deep analysis of the state of the project where the new system has been built on is provided. First, it is thoroughly described, with particular attention to the installed components. Furthermore, the camera-based lane detection algorithm is studied along with its strongest and weakest points.

2.1 Project generalities

The project object of study has been launched by FKFS[5] with the aim of creating an autonomous vehicle capable of driving without human intervention in a controlled environment. Such project is acknowledged as *eWolf* and at the current level of development is meant to operate in the road network around and inside the Campus of the University of Stuttgart (DE). It achieved promising results so far, but its full operability has still to be met. In this context the developed project offers an opportunity to enhance the

[5] Forschungsinsitut fur Kraftfahrwesen und Fahrzeugmotoren Stuttgart

functionalities of the vehicle, hopefully enlarging the spectrum of conditions in which it can reach a satisfactory operation.

2.2 Vehicle overview

The hosting infrastructure for the eWolf project is represented by a minivan which offers enough room to fit all the needed hardware components. In the front part of the van, which usually hosts the driver and the passengers, are installed the processing units and the actuators controlling the longitudinal and lateral dynamics of the vehicle. In the rear portion instead, the seats have been removed to leave space to the RC control unit and the GPS receiver. Figure 2.1 and Figure 2.2 provide visual description of the outline and the internal layout of the eWolf respectively.

Figure 2.1: eWolf vehicle - Source FKFS

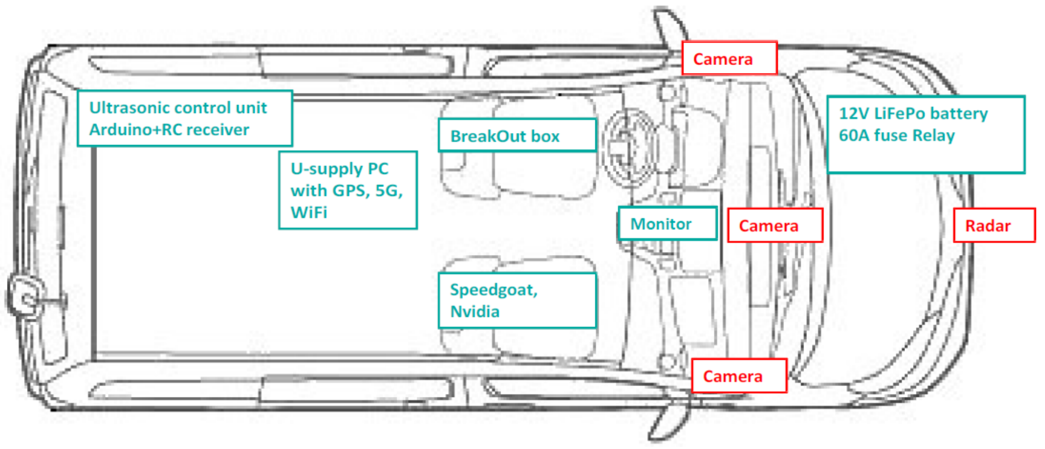


Figure 2.2: internal eWolf layout - Source FKFS

The three cameras installed on the dashboard enable stereovision, ensuring a wide-ranging awareness of the environment in front of the vehicle. This makes possible to detect the road markings essential for lane detection. The Nvidia® processing unit elaborates the images caught by the cameras and, after the implementation of the developed algorithm, builds the command signals provided to the different actuators. The BreakOut-Box serves as a diagnostic tool to find any anomalies in the serial communication among the installed components, enabling to test each communication channel and connect or disconnect devices. The U-Supply serves as an additional processing unit to manage the signal coming from the GPS sensor.

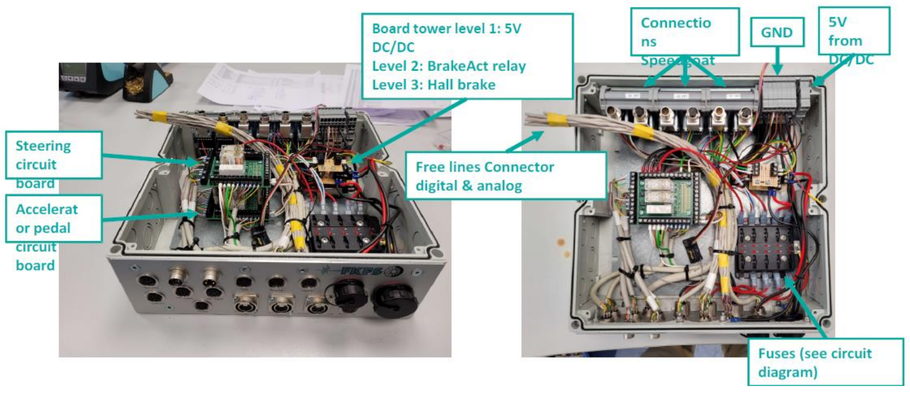

Board tower level 1: 5V DC/DC
Level 2: BrakeAct relay
Level 3: Hall brake
Connections Speedgoat
GND
5V from DC/DC
Steering circuit board
Accelerator pedal circuit board
Free lines Connector digital & analog
Fuses (see circuit diagram)

Figure 2.3: BreakOut box detail - Source FKFS

2.3 Lane detection algorithm

The algorithm used in the case-study and mentioned in the previous section is known as *LaneDet* [22]. It is an open-source lane detection toolbox with the aim of combining a comprehensive range of state-of-the-art lane recognition models. It supports many neural networks as backbones, such as ResNet, ERFNet, VGG, MobileNet and DLA (coming soon) and many detectors as well: SCNN [23], UFLD [24], RESA [25], LaneATT [26], CondLane [27] and CLRNet (coming soon). Included in the main framework, an object detection branch can be found as well. It is based on the YOLO network and makes the algorithm capable of recognizing in real time the surrounding objects like other cars, pedestrians or cyclists.

If the main branch of the code is analyzed the following sequence of steps can be outlined:

1. Implementation of the stereo vision process: it is necessary to combine the images from the three different cameras and provide tridimensionality to the recorded snapshot.
2. Retrieval of the frame: instant by instant the 2D frame must be extracted from the output of the previous step.
3. Denoising of the frame: with the utilization of a kernel, the picture noise is drastically reduced.
4. Grayscaling and edge detection: first converting the frame into a black-and-white image, the edges of the objects contained in it are outlined.
5. Determination of the region of interest: to increase the computational efficiency, the process of detection is carried out on a limited region of the frame only.
6. Perspective warping: to properly perform the further operations, the perspective of the image must be converted into a birds-eye-view (BEV) angle[6].

[6] This view emulates a top view, as a bird would have from above the vehicle

7. Lanes segmentation: this step plays a pivoting role in the process because allows to determine the position of the lane boundaries considering the distribution of the frame pixels.
8. Determination of the fitting model: a quadratic model is used to fit the detected lane boundaries in order to display them in vehicle's coordinate system.
9. Draw and display lines on the frame: this last passage accounts for visualization purposes.

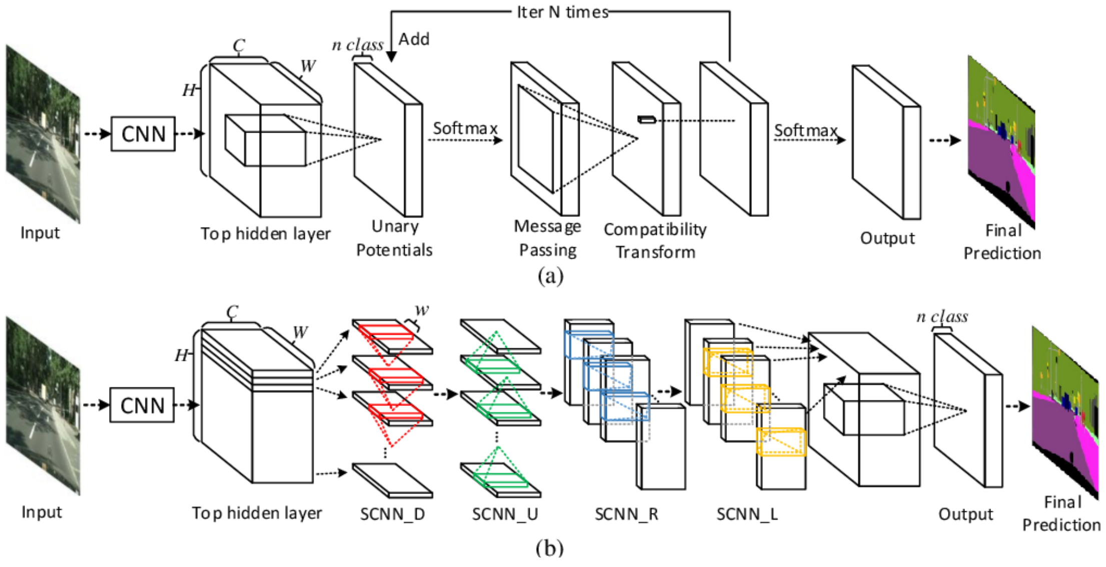


Figure 2.4: Schematics of SCNN architecture - Source [23]

2.4 Strong and Weak points

As mentioned in the introductive paragraph of this chapter, the system has remarkable capacities of autonomous driving in many different scenarios. The tests have demonstrated its capability to properly detect the lane boundaries and follow the shape of the road accordingly. On the other hand, when complex road geometries as roundabouts or intersections come up, the system is no longer capable of recognizing the lane boundaries and fails its task of guiding the vehicle. This happens because the camera alone does not provide sufficient information to the algorithm to distinguish the actual lane boundaries in case of missing or complex features as in those situations.

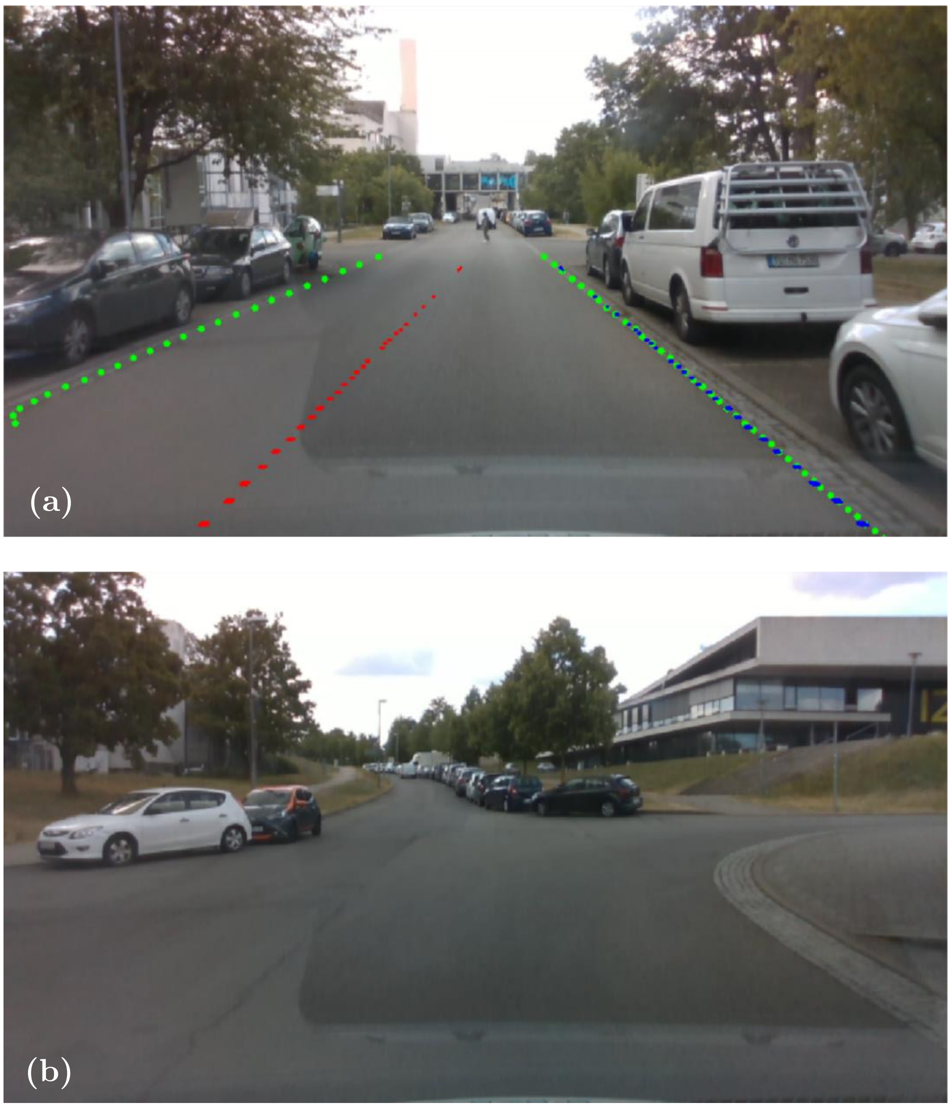


Figure 2.5: different behavior of the lane recognition algorithm in correspondence of a straight road (a) and intersection (b)

In Figure 2.5 (a) it is possible to understand how the algorithm works: once recognized the lane boundaries it fits them with the quadratic model mentioned in the previous section. At this point, the method compares the heading of the vehicle (represented by red and blue dots) to the direction of the route. The system then intervenes on the lateral dynamics control to close the gap between these two quantities. Figure 2.5 (b) represents instead a failure case in which the algorithm is not able to identify the road boundaries in correspondence of the intersection, thus the quadratic model parameters are not defined with the final consequence of no guidance to the eWolf.

2.5 Applicable enhancements

Acknowledging that there is this flaw in the system, the objective of this work is to provide an additional source of environmental perception to guide the vehicle in the challenging scenarios. Considered the unavailability of a Lidar, the only adoptable option was represented by the Global Positioning System. It enables *map-matching* which may be employed to guide the vehicle when the camera signal is not sufficient.

By generating a high-level reference route for the vehicle, the system would be capable to operate also when the existing method fails. This trajectory is the result of a refined path finding algorithm based on digital maps. The process is then completed by matching the GPS coordinates provided by the installed receiver to localize the vehicle on the route.

Chapter 3

Sensor fusion

In this chapter an introductory section is devoted to describe the technological enablers for the GPS-based method. Subsequently, the method employed to merge the information coming from the stereo camera and the GPS receiver is outlined. A very simple yet effective algorithm has been developed to achieve the capability to work in real time and build a successful guidance trajectory for the automated vehicle. The whole process has been subdivided into a number of simpler steps to ease the comprehension.

Note that the development of the algorithm has been carried out in a Python environment to cope with the existing code related to camera-based only lane detection.

3.1 Technological enablers

To adopt the GPS signal as the basis for a lane detection algorithm, it is necessary to understand how it works, the nature of its output and how it can be related to the road map of the surroundings. In the following paragraph the operation of the Global Positioning System is described, followed by a focus on the coordinates systems, the digital maps and the process of map-matching.

3.1.1 GPS

The Global Positioning System origins date back to the 1980s, when the Department of Defense of the United States of America started the first experiments. Originally conceived for military usage only, during the years its availability has grown up and it is currently widely employed for civil applications as well. The GPS consists of three segments:

- *Space*: the 24 satellites constellation surrounding the Earth
- *Control*: tracking, master and up-loading stations with data monitoring, processing, and transmitting tasks
- *Users*: wide set of different receivers determining their own position, velocity, and time

The determination of the position is based on a pulse exchange between the orbiting stations and the receivers which have the task of identifying the satellites in view and perform triangulation. This last process is essential since the location of the user is given by the intersection of the spheres generated by each satellite's signal on the Earth surface. It goes without saying that at least three satellites are needed to accomplish the task.

Performing a more accurate analysis of the phenomenon, everything starts with the distance estimation between the satellite and the user. It is obtained by multiplying the pulse speed (c, speed of light) by the delay time between its emission and its reception (τ):

$$D = c \cdot \tau \qquad [\ 3.1\ ]$$

The accuracy of the estimation depends on the accuracy in the estimation of τ. Nevertheless, this arises a problem since the perfect synchronization between emitter and receiver's clock is often not ensured. For this reason, the distance is affected by an error (δt_U) that takes this misalignment into account.

$$R = c \cdot \tau + c \cdot \delta t_U = D + c \cdot \delta t_U \qquad [\ 3.2\ ]$$

The result of [3.2] is referred to as the *pseudorange*.

To determine the user coordinates on the Earth's reference system it must be considered that the distance between the satellite and the receiver is given by [3.3]:

$$\rho = \sqrt{(x_s - x_r)^2 + (y_s - y_r)^2 + (z_s - z_r)^2} + c \cdot \delta t_U \qquad [\ 3.3\ ]$$

Where (x_s, y_s, z_s) are the known coordinates of the satellite and (x_r, y_r, z_r) are the unknown receiver coordinates. Considering that also the synchronization error is not known, and that triangulation must be performed, it comes up with four equations with four independent variables:

$$\begin{cases} \rho_1 = \sqrt{(x_{s1} - x_r)^2 + (y_{s1} - y_r)^2 + (z_{s1} - z_r)^2} + c \cdot \delta t_U \\ \rho_2 = \sqrt{(x_{s2} - x_r)^2 + (y_{s2} - y_r)^2 + (z_{s2} - z_r)^2} + c \cdot \delta t_U \\ \rho_3 = \sqrt{(x_{s3} - x_r)^2 + (y_{s3} - y_r)^2 + (z_{s3} - z_r)^2} + c \cdot \delta t_U \\ \rho_4 = \sqrt{(x_{s4} - x_r)^2 + (y_{s4} - y_r)^2 + (z_{s4} - z_r)^2} + c \cdot \delta t_U \end{cases} \qquad [\ 3.4\ ]$$

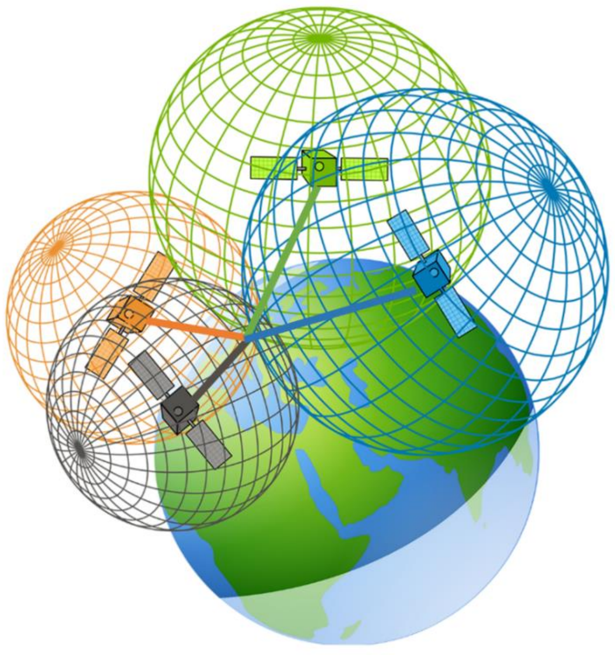


Figure 3.1: GPS coordinate determination - Source [70]

The result of [3.4] is a set of coordinates (x_r, y_r, z_r) which locates the user in a reference system belonging to the family of Conventional Terrestrial Reference Systems (CRTS). This category is characterized by the origin located into the center of mass of the Earth, the z-axis in correspondence of the rotation axis as it was at the beginning of the past century, the x-axis in correspondence of the plane defined by the Greenwich meridian and the y-axis that complies to the right-hand rule. The World Geodetic System 1984 (WGS-84) is a great example of a CRTS, but it actually locates the x-axis on a plane 5.3 arc seconds east of the Greenwich meridian and considers the datum surface as an oblate spheroid[7]. Nonetheless, to a large percentage of users, these systems are not as familiar as geographic coordinates like latitude (λ), longitude (φ), and altitude. Figure 3.2 shows how the coordinates are related each other.

In the end, it is worth mentioning that the accuracy of the estimated location, whichever is the reference system used, is still too poor for almost all technological applications. The solution to this problem comes from an augmentation system known as *differential GPS*. It relies on a series of stations to increase the reliability of the position information hinging on the evidence that the relative error between the location of two users that are not too far apart each other is theoretically null. Hence, a reference station calculates its position and compares it to its actual coordinates, then it broadcasts the resulting error (differential corrections) to all the receivers in the range of 100-150 km enabling them to correct their estimation.

[7] Ellipsoid of revolution obtained by rotating an ellipse around its minor axis.

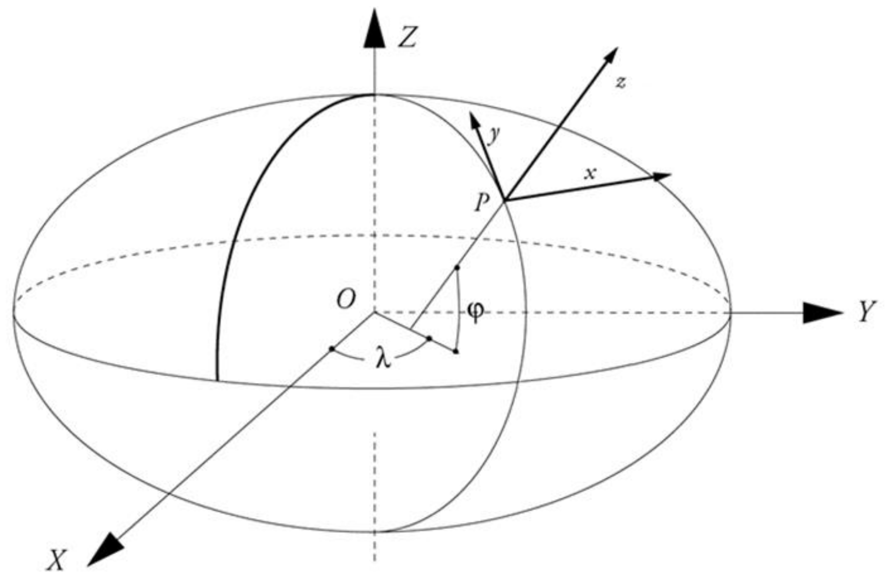


Figure 3.2: Relation between WGS84 and latitude and longitude coordinates - Source [71]

3.1.2 UTM coordinates system

When the geographical coordinates are used for navigation, their three-dimensional nature loses importance since the altitude information is usually not as fundamental as the other two. For this reason, bidimensional representations as maps are used. They constitute a projection of the geoid[8] for a delimited area, but it is clear that errors and distortions are prone to appear. For many centuries cartographers devised a large scale of projections, some of them fitting better for some purposes than the others. If navigation is concerned, the most valuable one is the Mercator Projection. It dates back to 1569 and "projects the Earth onto a concentric cylinder tangent to it along the equator. The meridians [...] are equally spaced and straight. The parallels [...] are also straight but unequally spaced, closest together at the equator and cutting the meridians at right angles." [28] From this representation, four

[8] The solid best approximating the Earth surface

centuries later the US army developed the Universal Transvers Mercator (UTM) reference system. As it happened with the GPS, it was initially conceived for military purposes, but it has been later widely adopted in civil applications as well. It divides the Earth into 60 zones each of which is small enough to neglect the distortion errors inside its borders. Each point is then defined with its X and Y coordinates inside each zone, often referred to as *eastings* and *northings* respectively. Their deployment opens the doors to algebraic and geometric calculations which would be otherwise not feasible with latitude and longitude coordinates, particularly when low distances are involved. A great example is provided by the possibility of building a spline to smooth a trajectory constituted by a set of coordinates, made only possible by the employment of the UTM system.

3.1.3 Digital maps

In modern times many applications rely on digital maps, from smartphone navigation apps and arcade videogames to driver assistance systems. They are 2D-graphs made up by a series of edges, representing road segments and a set of nodes, indicating road intersections or edges terminal points. These graphs may be correlated with databases that associate additional information to the road segments, such as width, number of lanes, restricted areas, etc.

The standardized description of the digital maps is given by the Geographic Data Files (ISO 14825:2004). GDF have no scale, can present the desired level of detail and are application independent, all features that make them very effective for the employments mentioned before.

A GDF may present three levels of detail:

- Level 0 (topology): includes low level description of features.
- Level 1 (features): contains information on road elements, river boundaries, etc.

- Level 2 (complex features): describes the elements in the most abstract way.

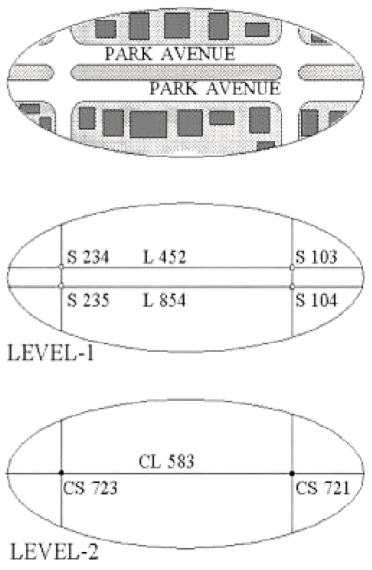


Figure 3.3: Three levels of abstraction in GDF - Source [72]

3.1.4 OpenStreetMap

Among the constellation of digital map providers available these days, OpenStreetMap has been chosen for this project. It is an open-source database created by volunteer users that contains road data collected around the Globe. OSM represents physical features on the ground with basic data structures referred to nodes and edges, and associating tags to them. All these data are represented in WGS-84 coordinate system, as GPS does.

- *Node*: it consists of a single point defined by its latitude, longitude, and node ID. It can be used to indicate point features but is more often employed in sequence with others to build the path of a way.
- *Way*: although it can be defined as a line, it technically consists in an ordered list of nodes representing linear features on the ground.
- *Relation*: it is an element containing a group of members, namely an ordered list of nodes, ways and/or relations. It defines logical or geographical relationships between these elements.

- *Tag*: it describes a geographical attribute of the feature it is attached to. There is a list of standard tags which include a comprehensive set of features, nevertheless users are free to create their own tag, if necessary. The full list of standard tags can be found at the organization's wiki website. [29]

To make use of the database offered by OpenStreetMap, it is sufficient to specify the desired bounding box on the source website [30] and download the resulting file. By default, it is set as an XML-like format which is manageable by the majority of applications and allows to easily implement the graph representation which characterizes the OSM data.

Digital maps play a pivoting role in the presented project, functioning as dynamic reference for the improved lane detection algorithm allowing it to operate in various road scenarios.

3.1.5 Map Matching

In the majority of applications, the sole GPS coordinates are quite useless due to their poor accuracy and large noise affection. To be employed in the field of autonomous driving, the GPS results must be matched with the digital map information, to properly locate the user on a defined path and perform the required operations. This activity is known as *map matching,* and it has been actively studied since the 1990s. Map matching is the result of two sub-activities: selection of candidate roads and identification of the best matching segment. For what concerns the first step, the state-of-the-art methods include geometrical [31, 32], topological [33, 34], and probabilistic [35 - 37] approaches. The first ones rely on the definition of a searching area in the neighborhood of the GPS point to determine the matching road, while topological approaches select candidates on the basis of the connections inside the road network. The latter ones instead, make use of static parameters to establish confidence intervals and determine the matching road. The outcome of this selection process may consist in one or more segments, according to the complexity of the road scenario. In case of multiple outputs, the identification of the best matching one becomes imperative. The methods found in literature can be categorized into four groups:

1. Direct projection algorithm, which directly projects the locating point on the nearest road.
2. Curve fitting algorithm, calculating the similarity between the driving path and the candidate road. [38, 39]
3. Advanced map matching algorithms that adopt intricate mathematical models to identify the best matching road. [40 - 44]
4. Weights-based algorithms, determining the right match through the association of weights to certain parameters. [37, 45 - 48]

3.2 Literature review

Before exposing the method developed for this project, an overview of the methodologies reported in the literature is provided. It serves as a background of the activities presented in this work and enables a better understanding of the steps taken.

The papers mentioned in the following review adopt the concept of sensor fusion mostly considering the combination between GPS signal and camera images. The only exception to this statement is represented by the second method of [49] which employs a LiDAR point cloud to perform map matching in conjunction with the geographical information. An interesting approach is presented in [50], leveraging on a High-Definition Map. Containing more detailed information about the road characteristics, traffic signs and street surroundings than a conventional map, it offers more cues to associate the recorder camera image with geographical data. The main drawback is represented by the availability of such source which in the majority of cases must be generated in advance with in-depth inspections of the driving scenario. More focused on the ego-lane detection in case of missing features, is instead [51]. This paper addresses the problem of identifying the lane in which the vehicle is driving when the road markings are not clearly visible or not present at all. This issue is also encountered in the case study, but the main focus of the work is on a wider range of circumstances. One of the most captivating methods is exposed in [52], which aims at creating a *Relational Local Dynamic*

Map to develop the concept of map-matching upon it. It represents a projection of the content of a digital map, so as to create *candidates* to compare with the actual recorded camera image and detect the position of the vehicle accordingly. In the end, the approach that mostly resembles the one adopted in this work is described by [53], that bases the operation of map-matching on a simple yet effective strategy: associate the coordinates provided by the GPS sensor to the closest point on the reference trajectory. The main difference stands in the generation of such trajectory: in the reported paper it is recorded with in-the-field acquisitions, while in this project it is generated with a path-finding algorithm.

3.3 Method abstract

At this point of the discussion, it is possible to deep dive into the main logic behind the implemented method. As explained before, the camera-based lane detection algorithm is not sufficient to provide guidance to the vehicle in every environment, thus the intervention of another system is required. The idea at the basis of the project is to find and draw a high-level trajectory that the vehicle must follow by default, which is improved and refined by the information coming from the camera-based lane detection system whenever it is available. Figure 3.4 depicts this logic in a flow chart.

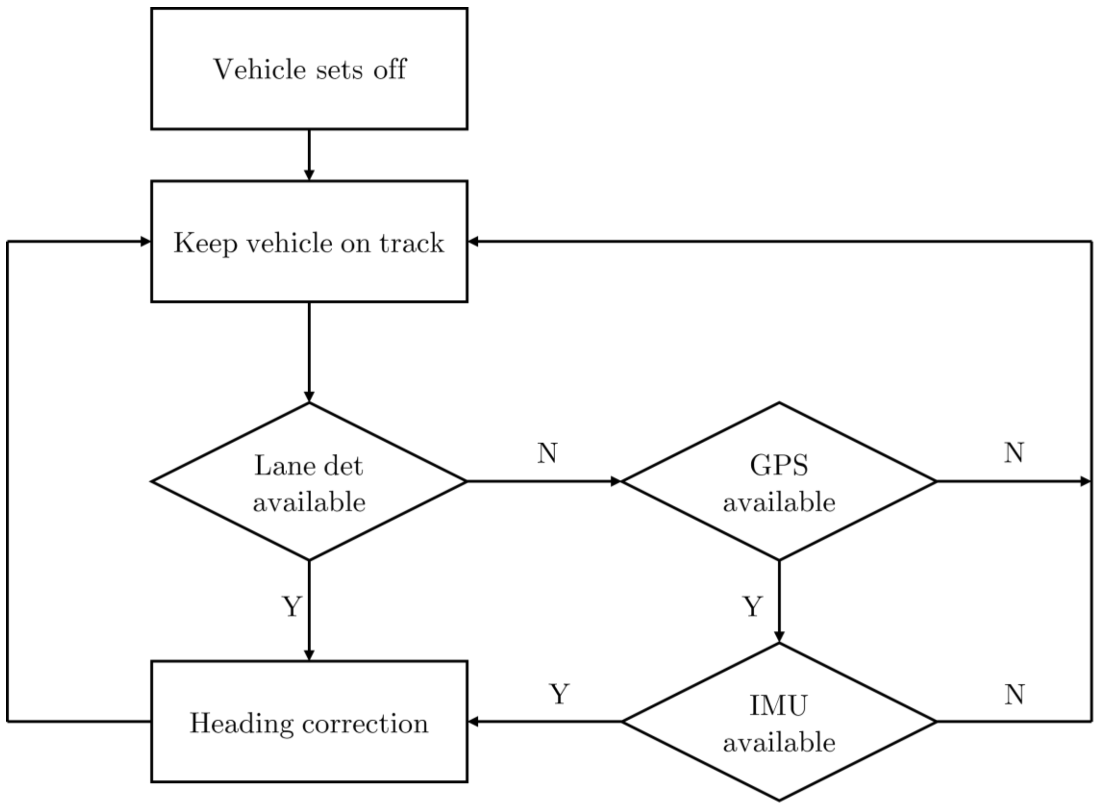


Figure 3.4: flow chart illustrating the logic of the project

At the beginning, the vehicle is considered to lay in correspondence of the first geographical location constituting the reference trajectory and not to be in the middle of an intersection, a junction, or a complex road scenario. These assumptions yield to the immediate availability of the lane detection algorithm. As soon as the vehicle sets off, the camera-based lane detection keeps it on track by comparing its heading with the detected road boundaries. When this method gets no longer available in presence of the conditions listed above, the GPS-based algorithm intervenes, providing guidance through the comparison between the vehicle's yaw angle and the direction imposed by the reference trajectory. It is worth mentioning that the second method keeps the vehicle in the correct lane identified before with the stereo camera since it does not force it to follow the trajectory virtually drawn in the middle of the road, but it is confined to keep the vehicle parallel to the direction indicated by the route.

In the end, whenever the GPS or the IMU data are not available, the algorithm preserves the direction imposed in the previous time instant and reattempts the whole process in the next step.

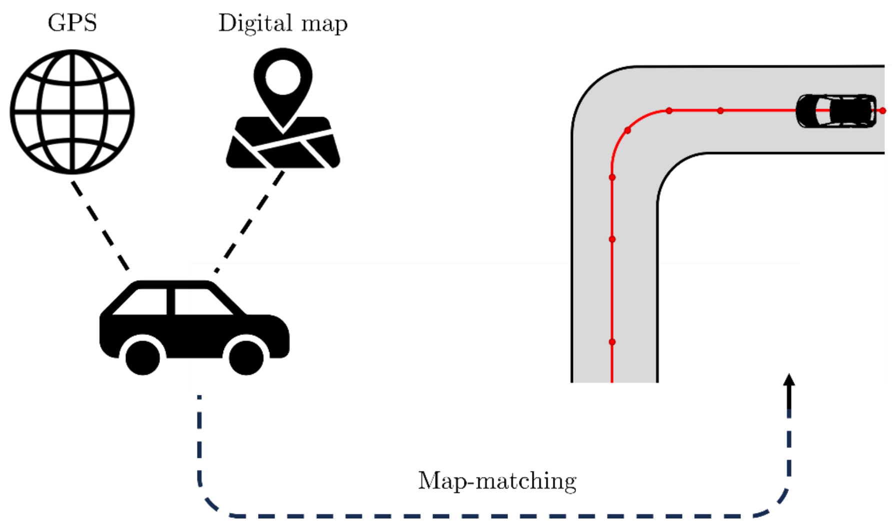


Figure 3.5: GPS-based lane detection process

Diving deeper into the GPS-based method, its nature can be inspected. The underlying idea is extremely simple but it showed promising results. Basically, a reference trajectory is drawn between two geographical points so as to respect the criterion of the shortest path and taking into account the preferred mean of transport (with the associated viable ways) as well. After this first step, map matching phase takes place. It consists in nothing but localizing the vehicle on the map, task that is accomplished through the application of a straightforward concept: being the vehicle forced to follow a predefined trajectory, its exact position will surely be in correspondence of one of the points constituting such trajectory. Of course, this idea cannot be implemented as-is and requires a sequence of refinement, which are listed in the following paragraphs.

3.4 Determination of the shortest path

The first thing to take into account when considering routing tasks is the representation of geospatial data which is crucial to design an efficient algorithm. Representing geospatial information has consistently presented a complex challenge, and as the quantity of location data expands, this intricacy is set to persist. When it comes to visualizing geospatial data, a couple of crucial aspects must be considered:

- When zoomed in at higher scales, it becomes imperative to present a condensed or restricted perspective of the data. Conversely, at lower scales, the majority of the data should be displayed in comprehensive detail.
- The capacity of the routing processor presents constraints concerning the efficient management of substantial data sets.

For those reasons the implementation of *map tiles* becomes imperative. Map tiles are conceived as square portions of a map (256 x 256 pixels), employed in map visualization browsers to drastically reduce the computational effort in visualizing geospatial data. Their concept can be extended to routing algorithm, for which just the map tiles involved in the pathfinding process are loaded with consequent advantages in terms of efficiency. More details about map tiles and their implementation can be found at [54].

Leveraging on the graph nature of the OpenStreetMap data, a routing method to find the shortest path between two locations can be effortlessly implemented with the python library `pyroutelib3` [55]. It first receives as input the source and destination coordinates, then finds the nearest map nodes to them, and finally employs the *A* search algorithm* to determine the shortest path. It is a widely used pathfinding and graph traversal method that finds the shortest path from a starting point to a goal node in a graph but differentiates from the others because "it incorporates an estimate of the cost of 'path-completion [56].

The outcome of this step is a list of nodes constituting the shortest route from the source to the destination location but if the relative coordinates are not inherently linked to them, their usefulness is diminished.

For this reason, this conversion is performed creating a reference trajectory in the so-called global map. For visualization purposes an example route is plotted in Figure 3.6.

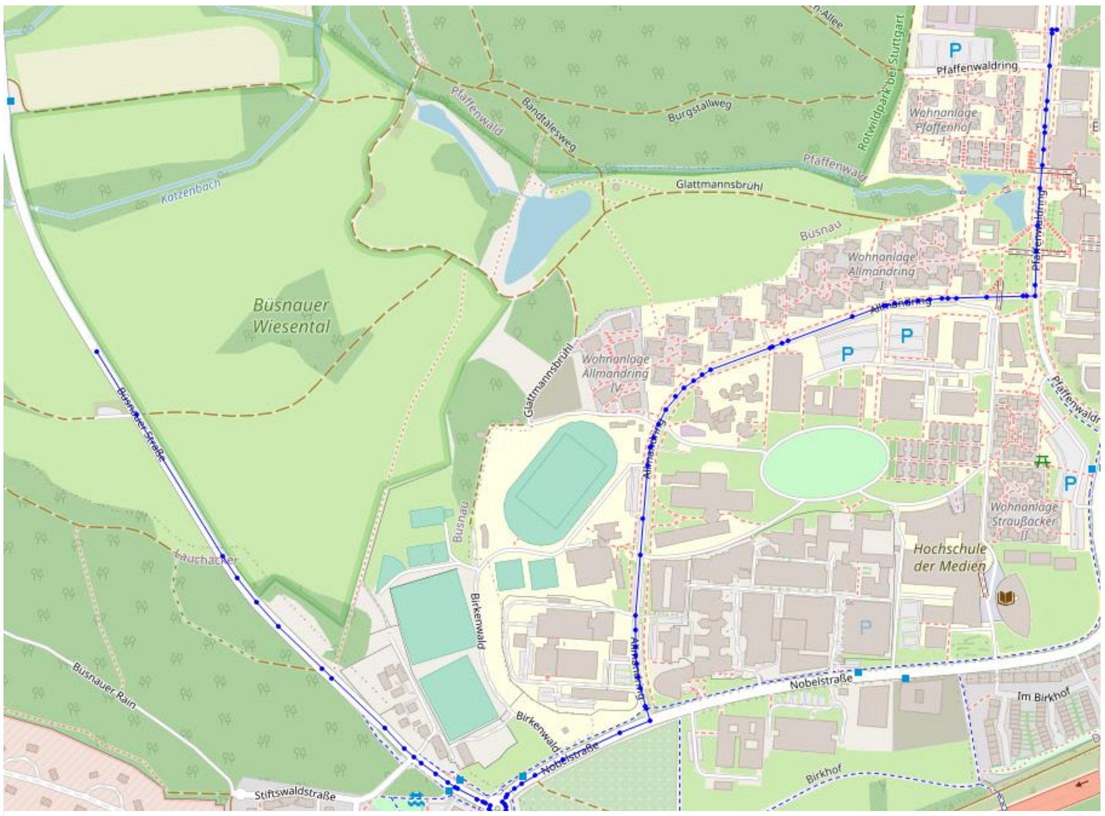


Figure 3.6: set of nodes (blue dots) constituting the shortest path between two points

3.5 Trajectory refinement

The set of points obtained from the previous procedure represents nothing more than just a reference for the vehicle guidance. The first problem is represented by the fact that it presents a non-constant distribution of points, which are much more dense in correspondence of turns or road intersections; additionally, the edgy sequence of segments seen in Figure 3.6 must be converted into a smooth trajectory, meant at avoiding abrupt changes in direction that would be critical for the vehicle dynamics control. Finally, the sole coordinates are lacking information for the task of automated driving, thus an augmentation process must be carried out. Specifically, the method is subdivided into three sub-steps:

- Point density increment
- Sharp corners rounding
- Data augmentation

3.5.1 Point density increment

This passage plays a role in the trajectory refinement process since it allows to increase the exploitability of the drawn path, incrementing the number of points it is made of. Precisely, it finds its application in the map-matching phase, in which the vehicle's coordinates system must be associated to a point of the trajectory it is following. It is easy to understand that the higher the number of points constituting the route, the higher the accuracy of the map matching process.

The designed algorithm firstly takes as input a set of coordinates, namely the trajectory returned by the router function, and calculates which is the minimum distance between two consecutive points that occur in the whole set. Once found this parameter, the number of items to be inserted between two consecutive points in the list is calculated as the closest integer to the ratio between the distance from a point to its subsequent one and the minimum gap in the list discovered before. The result of this operation can be visualized as a set of coordinates almost equally spaced along the trajectory.

Another version of the method, which may fit better in some cases, instead of calculating the minimum distance in the list, fixes the number of points to be inserted to a predefined value. Figure **3.7** shows the different outcomes of the two approaches.

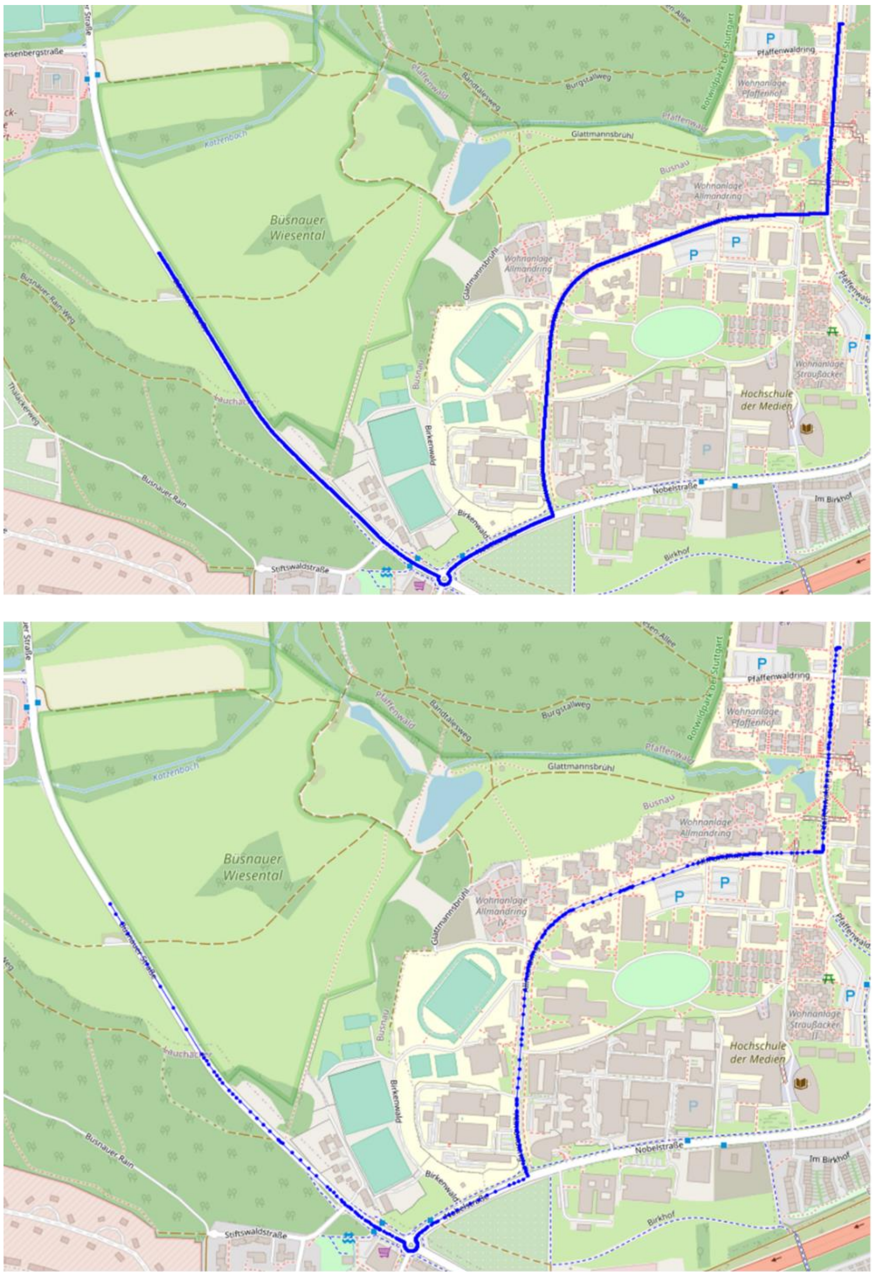


Figure 3.7: density incrementation with dynamic (a) and fixed parameter (b)

As clearly visible in the figure, the density of points is much higher in the first case, while in the second method it only increases in the contour of road direction changes. The reason lies in the nature of the OpenStreetMap graph that is characterized by denser segments (thus more points bounding them) in correspondence of bends, intersections, or roundabouts.

3.5.2 Sharp corners rounding

The subsequent step is represented by the creation of a smooth trajectory to replace the edgy sequence of segments returned by the previous passages. The reason why it is so necessary is closely related to the vehicle dynamics: since the drawn route must work as a reference to guide the vehicle through the driving environment, it must be as smooth as possible to avoid abrupt changes of direction. On the other hand, the rounding operation must not create a new trajectory that crosses non-drivable or potentially dangerous areas such as sidewalks, roadside barriers, or curbs. To fulfill these requirements, two state-of -the-art methods are available: the *Bézier Curve* and the *spline.*

A *Bézier curve* is a parametric curve used in computer graphics and related fields [57], that took the name from its inventor Dr. Peter Bézier. It has been originally developed in 1960 to provide a mathematical model to the curves that shape the body of motor cars, but soon its application extended in many other fields. The basic concept consists in creating a set of control points, representing themselves the vertices of a sharp corner, in order to define a curve that approximates the bend. On the other hand, the formal definition describes the *Bézier Curve* as a "mapping from $s \in [0,1]$ to convex combinations of points $v_0, v_1, \dots, v_n$ in some vector space [...]".[58]

$$B(s) = \sum_{j=0}^{n} \binom{n}{j} s^j (1-s)^{n-j} \cdot v_j \qquad [\ 3.5\ ]$$

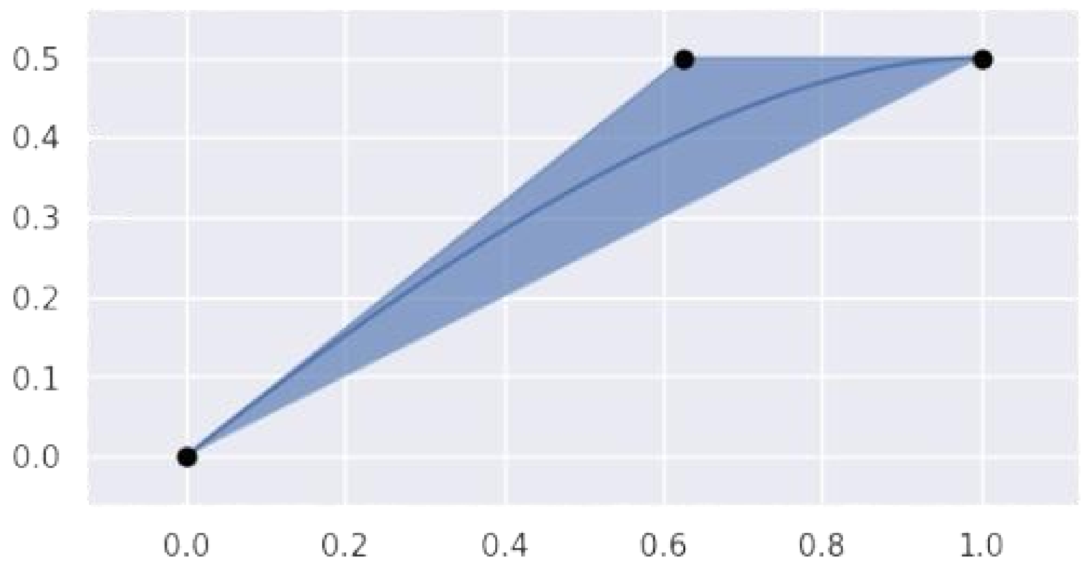
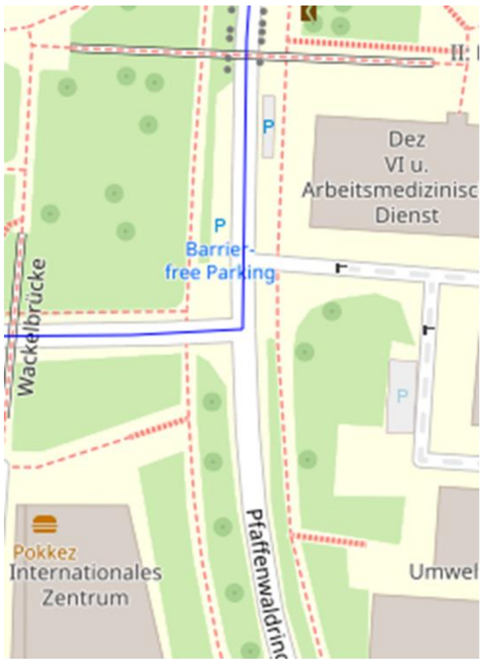
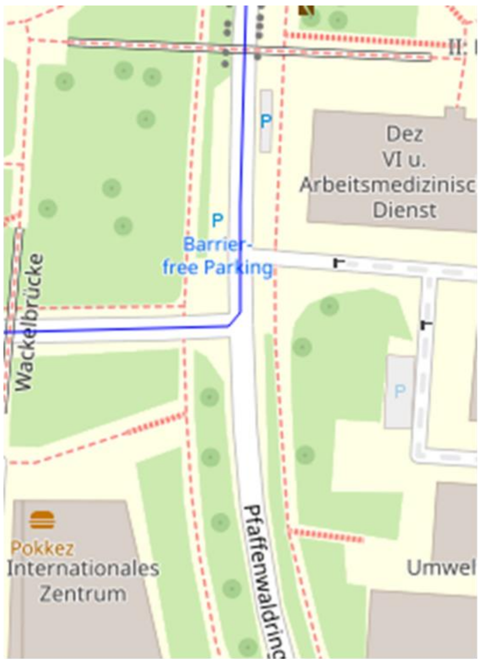


Figure 3.8: Bézier Curve and its control points - Source [58]

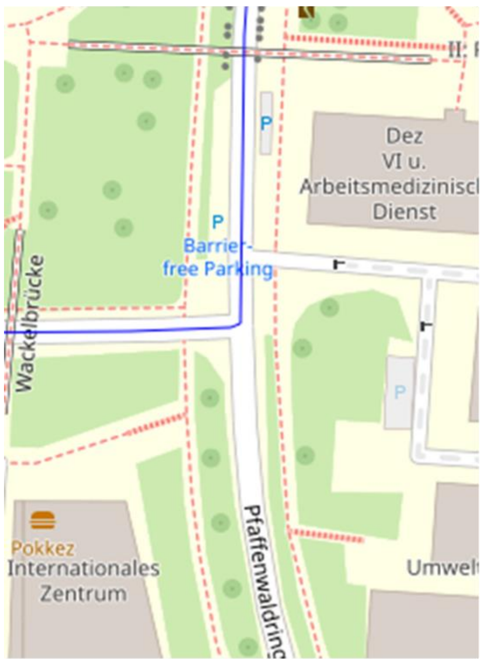
As visible in Figure 3.8, the Bézier Curve requires 3 control points to be built, thus the algorithm must take this number of items into consideration in every iteration on the list of coordinates. To do so, to every object in the list are associated two control points at a certain distance (dependent from the parameter *alpha*), aligned with the previous and subsequent points. Then, since the curve is defined in a vector space, it must be evaluated in real-world coordinates, with the desired accuracy for the independent variable. It is understandable that the higher the accuracy, the more precise the trajectory but the more computationally demandant the algorithm at the same time. It is also worth noticing that not only the accuracy and the distance at which the control points are drawn, but also the density of points represents a parameter to be fine-tuned to obtain the best outcome. The result of the whole process is a trajectory that shows smooth corners in correspondence of sharp changes of direction but preserving the straightness when not necessary at the same time.



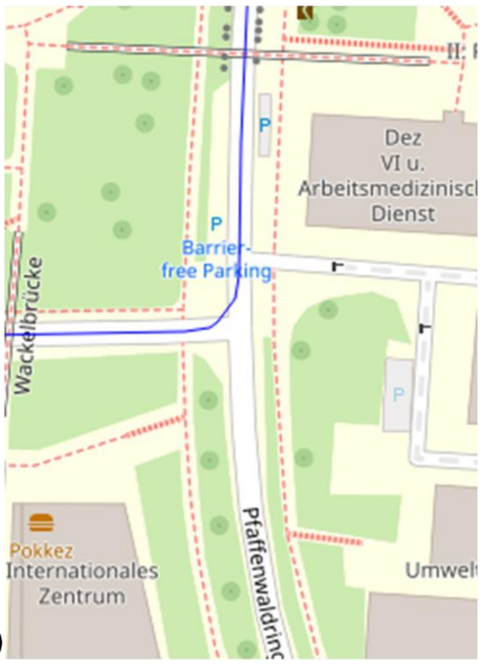
(a)



(b)



(c)



(d)

Figure 3.9: Fine-tuning outcomes

figure	alpha	accuracy	density
(a)	0	10	5
(b)	0.5	5	5
(c)	0.2	10	10
(d)	0.5	10	2

Table 3.1: Parameters associated to Figure 3.9: Fine-tuning outcomes

As clearly visible in Figure 3.9 the definition of the parameters has a strong impact on the final result. If alpha is too low, namely close to 0, the smoothing action is not appreciable, while if it is too large (maximum acceptable value is 0.5) and in combination to a low value of point density incrementation, the final path ends up lying on a non-drivable area. Depending on the final needs, the choice may fall on a medium-range value of alpha combined with a larger value of accuracy and density. For sake of comparison with the spline method, the parameters reported in example (c) have been used.

The alternative to this method is represented by *splines*. They are "piecewise polynomial curves that are differentiable up to a prescribed order" [59]. Their implementation ranges from data science to graphics and they are involved whenever it comes to interpolation or approximation of points. A particular case of spline, known as B-spline (abbreviation for *basic spline*) may be somehow associated to the Bézier curve since it can be described as an affine combination of control points c_i, namely:

$$s(u) = \sum c_i N_i^n(u) \qquad [\ 3.6\ ]$$

A more refined definition of B-splines and the way they are constructed involves complex mathematical notions, that are not worth to mention here. What is relevant is the outcome of the implementation of such a curve to approximate the trajectory returned by the router algorithm. Note that also this technique involves parameters to tune, among the others the smoothing factor *s*.

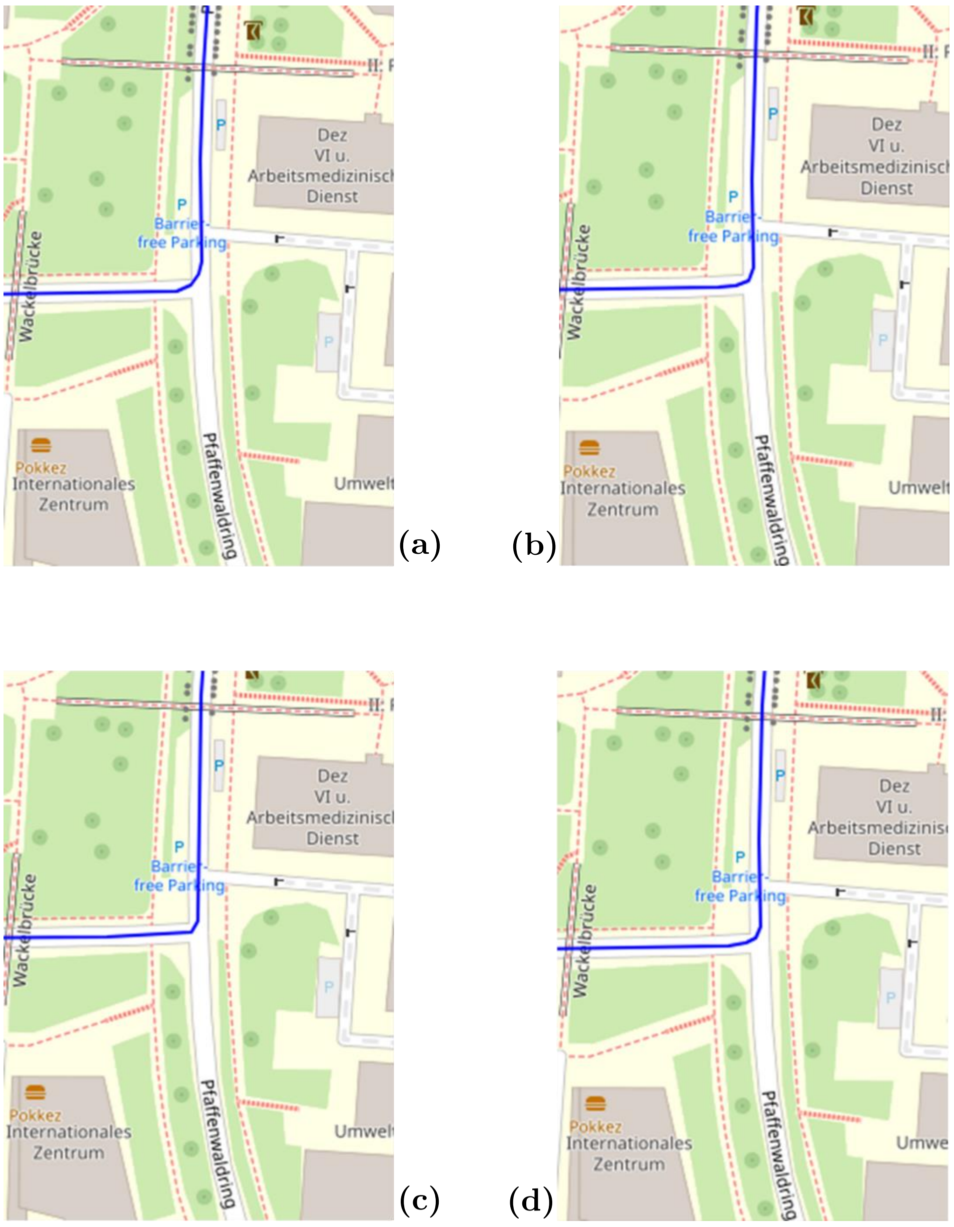


Figure 3.10: B-Spline approximation with factor s=2 (c), s=10(d), s=25(b), s=100(a)

After a deep analysis, the output of the spline turned out to be more effective, smoother and more efficient with respect to the Bézier method. For this reason, it has been chosen to proceed with the algorithm development.

3.5.3 Data Augmentation

As stated at the beginning of this chapter, the previous steps are not sufficient to provide a comprehensive set of information for the automated vehicle. Firstly, to create a smooth path using a B-spline the geographical coordinates (latitude and longitude) must be converted into UTM values which allows an easier manipulation. Secondly, beside the self-localization mission, the vehicle must be instructed about the direction it has to keep or follow, so as whenever the received GPS information is associated to a point on the trajectory, it is acquainted with the correction to apply to its yaw angle. To accomplish this task, each point of the reference route comes with an additional information: the direction towards the next point.

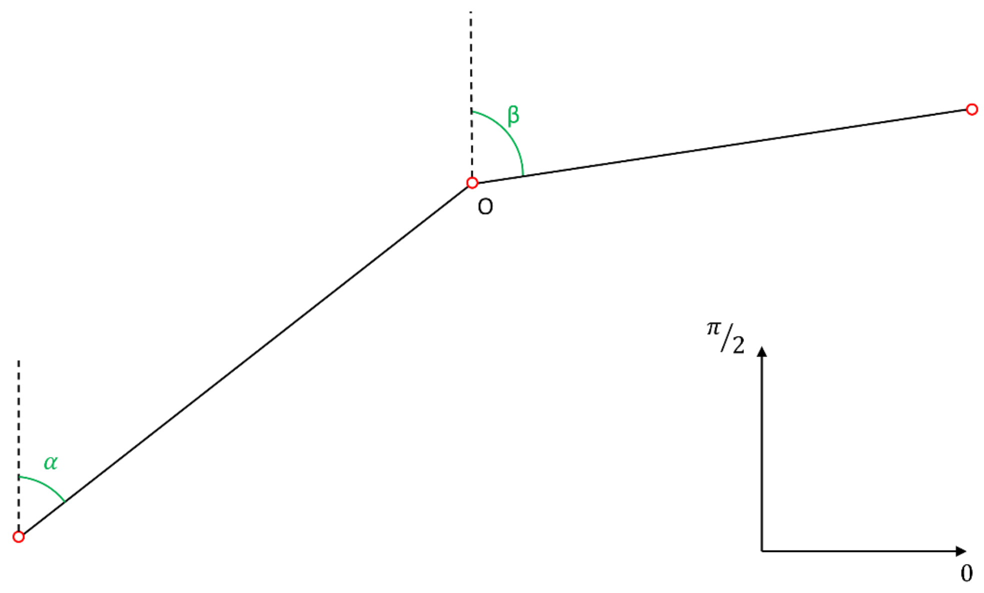


Figure 3.11: direction toward the next point for two subsequent segments

This quantity is calculated by a function that receives as input a point of the trajectory and once determined its position in the list (i.e. its index), memorizes the following one. At this stage, the x and y coordinates of the two points are considered to calculate their difference which is in turn fed to the `atan2()` function to determine the angle in radians between the two points.

$$\alpha = tan^{-1}\left(\frac{x_1 - x_0}{y_1 - y_0}\right) \qquad [\ 3.7\ ]$$

3.6 Closest point determination

In this section the process of map-matching is broken down into its details and exposed in all its simplicity. As reported in the previous paragraphs, the idea is to compare the received GPS signal to the coordinates constituting the reference trajectory built before and find the closest point on it. In this way the vehicle is localized on a specific point of the route. The adopted method is not far from the one described in [53]. At first, each point provided by the GPS sensor p_k is considered as a list of its three geographical coordinates, even though the altitude is not considered for simplicity.

$$p_k = (x_{pk}, y_{pk}, z_{pk}) \qquad [\ 3.8\ ]$$

After the previous tweaking to the points of the reference trajectory, they appear like a list of four elements: the three geographical coordinates and the reference direction.

$$r_j = (x_{rj}, y_{rj}, z_{rj}, \psi_r) \qquad [\ 3.9\ ]$$

Since the assumption of bidimensionality of the process always applies, in the next step only the first two elements of each list will be taken into consideration. At this point, around each GPS point a buffer of 10 meters of radius is built (green circles in **Figure 3.12**), which is supposed to enclose some coordinate couples belonging to the reference trajectory. They are identified as follows.

$$r_j^b = \left(x_{r_j^b}, y_{r_j^b}\right) \qquad [\ 3.10\ ]$$

Among them, the Euclidean distance is iteratively calculated with respect to GPS coordinates to find the minimum value, thus the closest point.

$$CP_{p_k}^b = r_j^b, where\ j = \operatorname{argmin}\left\{\sqrt{\sum_{t=1}^{2}\left(p_{kt} - r_{jt}\right)^2}\right\} \qquad [\ 3.11\ ]$$

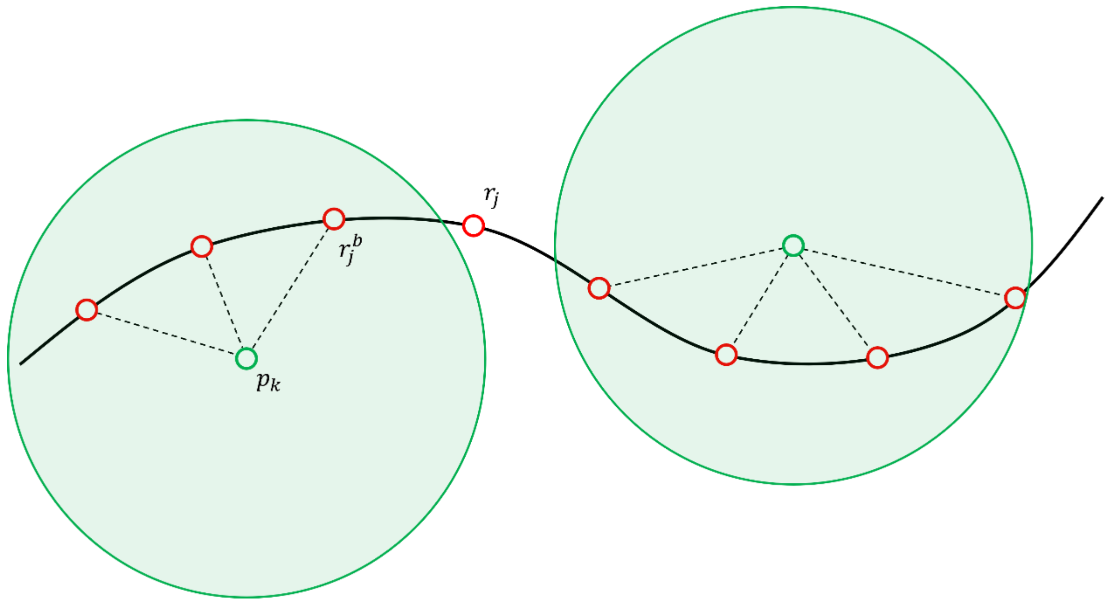


Figure 3.12: closest point determination

The result of these operations enables to localize the vehicle on a specific point of the trajectory with sufficient accuracy, considering the acquisition frequency of the GPS signal equal to 1 Hz and the relatively small velocity of the vehicle, especially in correspondence of junctions or roundabouts.

3.7 Coordinate conversion

Once the map-matching task is accomplished and consequently the reference direction is retrieved, it becomes imperative to put in relation the different reference systems involved. Either if the yaw angle of the vehicle is converted to global coordinates frame, or the reference direction is converted to vehicle's coordinates, this step cannot be missed.

Such problem may be solved borrowing some concepts from the robotics field, namely the Homogeneous Transform Matrix [60]. Nevertheless, before diving into this topic it is necessary to take a step behind. To perform a coordinate conversion from a fixed reference frame to a moving one (as in this case) it is first necessary to describe the *configuration* of the latter. This task responds to the questions "where is it located?" and "which is its orientation?", and to do so the couple of algebraic quantities (R, p) is needed. $R \in \mathbb{R}^{3,3}$ and it is the rotation matrix representing the orientation of the moving frame with respect to the fixed one, while $p \in \mathbb{R}^3$ and defines the position of the moving frame in fixed RF coordinates. Once R and p are stacked together in a single entity, the Homogeneous Transform Matrix T is obtained.

$$T = \begin{pmatrix} R & p \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & p_1 \\ r_{21} & r_{22} & r_{23} & p_2 \\ r_{31} & r_{32} & r_{33} & p_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad [\ 3.12\ ]$$

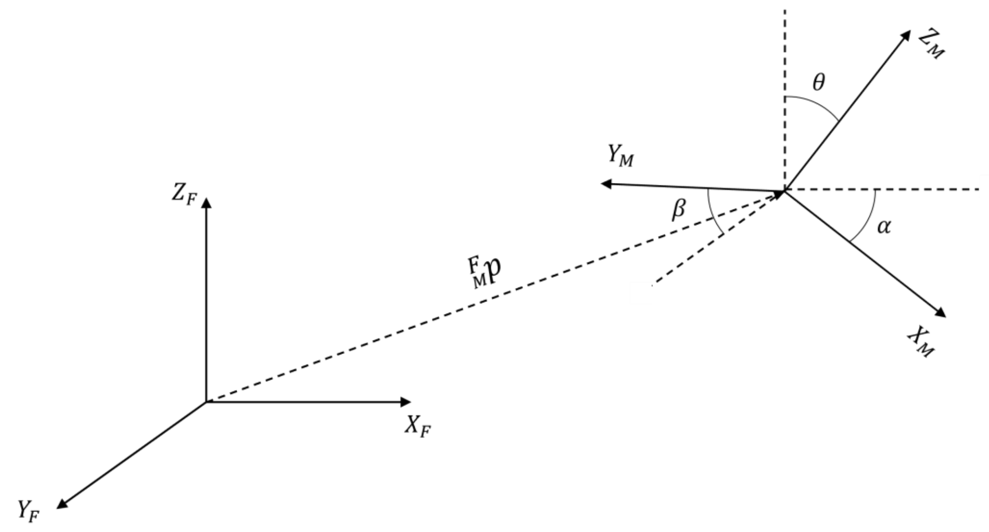


Figure 3.13: Fixed and moving reference frames

Figure 3.13 graphically shows the meaning of the *configuration* concept. The fixed frame is identified by the subscript F, while subscript M indicates the moving one. The angles θ, α, β are associated to the orientations of the three axes of the moving frame with respect to the fixed one. Anyway, this representation may be reduced to a two-dimensional space for the case study, decreasing the computational effort and increasing consequently the process efficiency. The Homogeneous Transform Matrix will be reduced, so $T \in \mathbb{R}^{3,3}$.

$$T = \begin{pmatrix} R & p \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & p_1 \\ r_{21} & r_{22} & p_2 \\ 0 & 0 & 1 \end{pmatrix} \qquad [\ 3.13\ ]$$

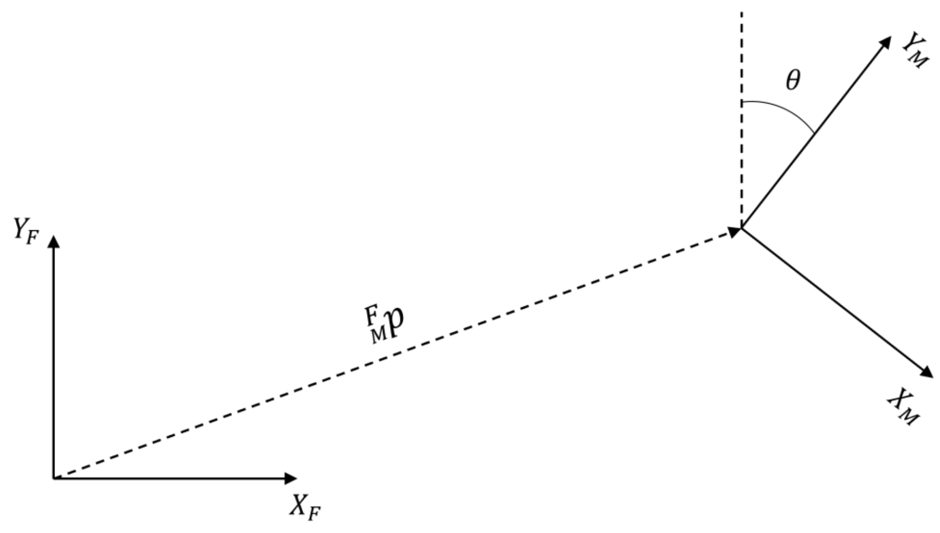


Figure 3.14: Fixed and moving reference frames in 2D

At this point it is possible to define with more accuracy the components of the rotation matrix $r_{11}, r_{12}, \ldots, r_{nn}$ since the angle θ is enough to define the orientation.

$$T = \begin{pmatrix} \cos\theta & -\sin\theta & p_1 \\ \sin\theta & \cos\theta & p_2 \\ 0 & 0 & 1 \end{pmatrix} \qquad [\ 3.14\ ]$$

Now that the transformation operator has been fully defined, the coordinate conversion can be performed straightforwardly.

Defining v_F and v_M as the vectors describing the position of point P in the fixed and moving reference frame respectively, from the application of the Homogeneous Transform Matrix, it comes that:

$$v_F = {}^F_M T\ v_M \qquad [\ 3.15\ ]$$

Note that [3.15] performs a conversion from moving to fixed RF, thus if the opposite operation is needed, it is sufficient to invert the equation by multiplying both sides for the inverse of T

$${}^F_M T^{-1}\ v_F = {}^F_M T\ v_M\ {}^F_M T^{-1} \qquad [\ 3.16\ ]$$

$${}^F_M T^{-1}\ v_F = v_M \qquad [\ 3.17\ ]$$

Where

$${}^F_M T^{-1} = {}^M_F T = \begin{pmatrix} R^T & -R^T p \\ 0 & 1 \end{pmatrix} \qquad [\ 3.18\ ]$$

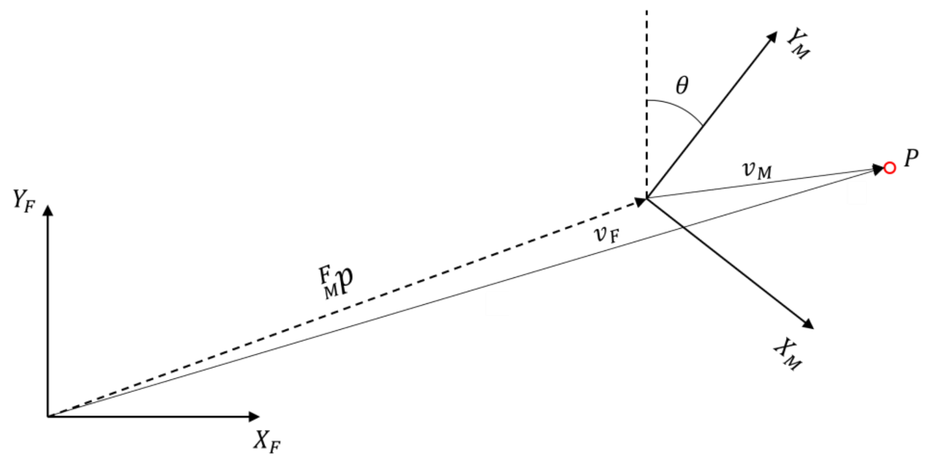


Figure 3.15: coordinate conversion process

3.8 Vehicle lateral dynamics control

To guide the vehicle and keep it on track, the lane detection algorithm leverages on the control of its lateral dynamics, thus a dissertation about this topic is well deserved.

The design of a proper control system is of paramount importance when it comes to keep a vehicle on track. It is based on a mathematical vehicle model which must describe the reality with a sufficient level of approximation without introducing useless and computationally-demandant complexity. The mostly-used model to fulfill this requirement is the so-called Dynamic Single-Track Model (DST). It is characterized by a single-track (two wheels, one in front and one in the rear axle), that is equivalent to a four-wheeled car symmetrical with respect to the longitudinal x axis.

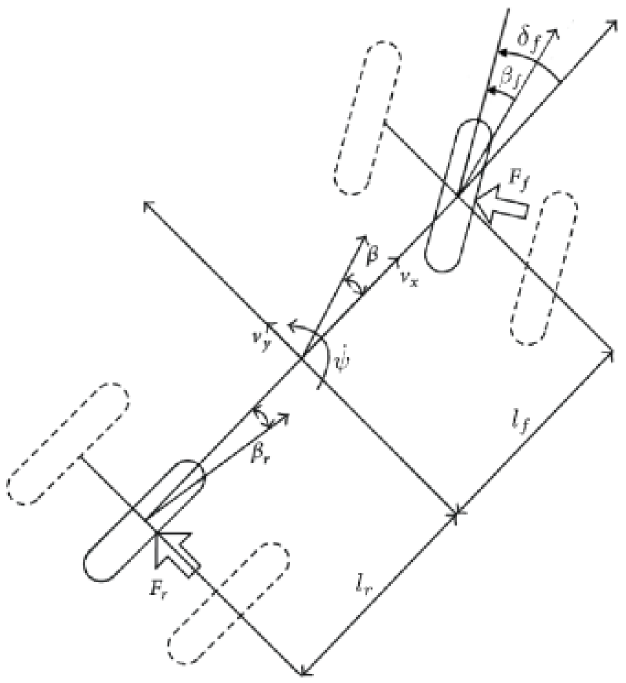


Figure 3.16: Dynamic Single Track Model - Source [73]

The vehicle variables visible in Figure 3.16 are:

- δ_f: steering angle
- β: vehicle side slip, i.e. angle between the vehicle's longitudinal axis and the vector of velocity
- β_f, β_r: tire side slip, i.e. the angle between the tire's longitudinal axis and the vector of velocity
- v_x, v_y: longitudinal and lateral velocity
- ψ: yaw angle

The vehicle parameters are instead identified as:

- l_f*:* distance between the center of gravity and the front axle
- l_r: distance between the center of gravity and the rear axle
- m, J: vehicle mass and moment of inertia

The resulting state equations of the DST model are shown in [3.19]

$$\begin{cases} \dot{X} = v_x \cos\psi - v_y \sin\psi \\ \dot{Y} = v_x \sin\psi + v_y \cos\psi \\ \dot{\psi} = \omega_\psi \\ \dot{v_x} = -v_x\omega_\psi + a_x \\ \dot{v_x} = -v_x\omega_\psi + \frac{2}{m}\left(F_{yf} + F_{yr}\right) \\ \dot{\omega_\psi} = \frac{2}{J}\left(l_f F_{yf} - l_r F_{yr}\right) \end{cases} \qquad [\ 3.19\ ]$$

Where a_x is the longitudinal acceleration, F_{yf} and F_{yr} are the lateral forces exchanged between the tires and the ground, calculated with linear relations from the side slip angle. If the latter is relatively small and v_x is constant, such definition is sufficient. For a more precise model the *Pacejka's magic formula* must be adopted [61]. The state of the system is instead represented by $\zeta = \left(X, Y, \psi, v_x, v_y, \omega_\psi\right)$ and the input by $u = \left(a_x, \delta_f\right)$.

The control quantity differs according to the algorithm that is considered. The camera-based lane detection leverages on the *cross-track error* e_{ct} while the GPS-based method exploits the *heading error* e_h. The first is defined as the orthogonal distance between the vehicle's left side and the

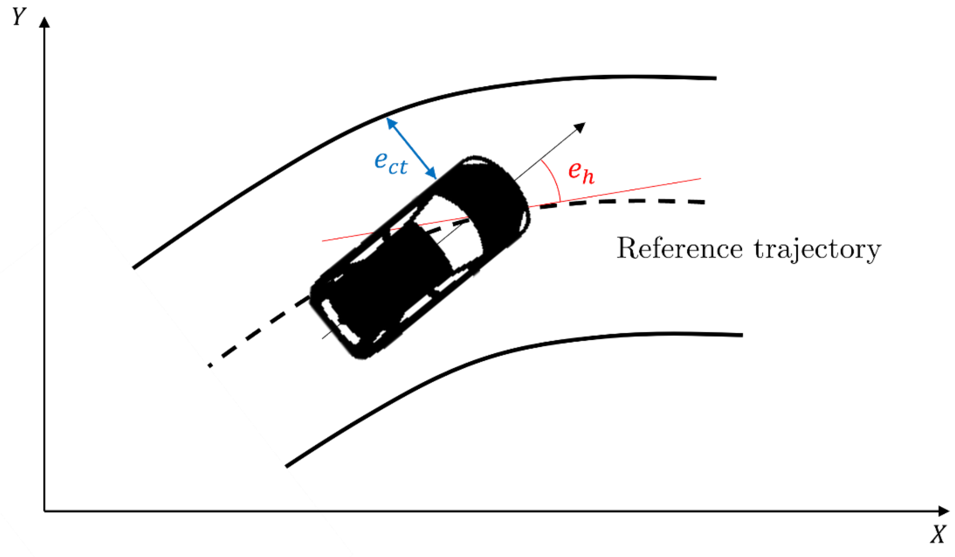


Figure 3.17: heading and cross-track error

detected left lane boundary (see Figure 2.5 for better understanding). The latter refers instead to the angle between the vehicle longitudinal axis and the reference trajectory tangent.

To properly compute this error, it is necessary to introduce the following quantities:

- $p_a \doteq (X_a, Y_a, \psi)$: vehicle front axle pose
- $P_r \doteq \{p_r^i, \dots, p_r^N\}$: reference trajectory
- $p_r^i \doteq (X_r^i, Y_r^i, \psi_r^i) \in P_r$
- $p_r^c = (X_c^c, Y_r^c, \psi_r^c),\ c = \text{argmin}_i \left\| (X_r^i, Y_r^i) - (X_a, Y_a) \right\|$

Finally, the heading error is defined by:

$$e_h \doteq \psi_r^c - \psi \qquad [\ 3.20\ ]$$

And the cross-track error by:

$$e_{ct} \doteq (Y_r^c - Y_a)\, cos\, \psi_r^c - (X_r^c - X_a)\, sin\, \psi_r^c \qquad [\ 3.21\ ]$$

At this point it is possible to implement a closed-loop control system able to provide the steering angle to the actuators controlling the lateral dynamics of the eWolf.

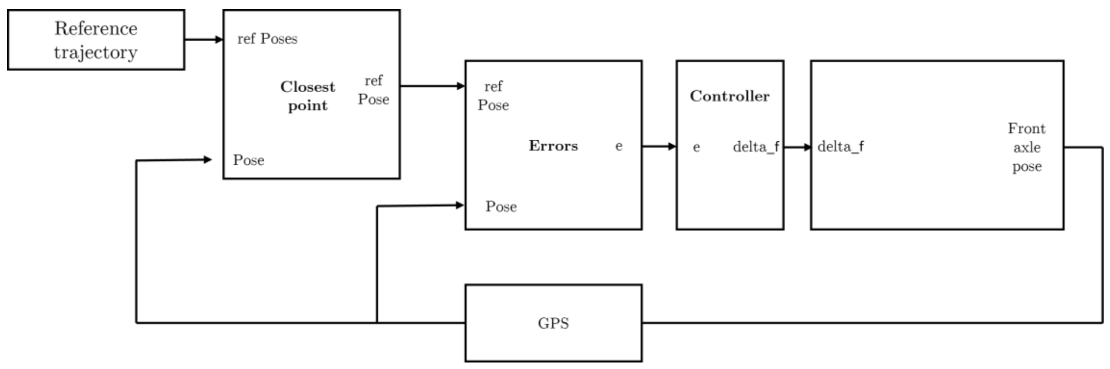


Figure 3.18: closed-loop control scheme

The reference trajectory calculated from the target geographical locations is fed into the closest point block which determines the reference pose with the procedure described in the previous sections. It is then compared to the actual pose of the vehicle to calculate the heading and/or the cross-track error, used as input by the controller. In the final block the front axle pose is eventually determined and augmented with the information coming from the GPS sensor. Such block scheme is tailored depending on the adopted algorithm.

The study of the controller is not the main scope of this work, but it can be of the Proportional Integrative Derivative (PID) or Stanley Kinematic kind. More details are available at [62, 63].

Chapter 4

ROS implementation

The operations listed in the previous pages serve as background framework for the final implementation of autonomous driving in the simulation environment adopted for the case study: ROS. The Robot Operating System has originally been conceived for the development of robot systems, as the name suggests, but with the proper tweaking it can be seamlessly used to test autonomous vehicles. In the next sections the components and the functionalities of the middleware will be listed and analyzed, followed by the actual implementation of the system.

4.1 ROS Basics

ROS is a free and open-source robotics middleware employed in many applications spanning from commercial to research fields. The ROS framework offers a list of robot-programming features listed in the following. [64]

- Process intercommunication: ROS provides a message-passing interface to allow communication between processes and programs, core feature for robot programming
- Operating system features: even though ROS does not properly work as an operating system, it provides many functionalities as

multithreading[9], low-level device control and hardware abstraction which allow the programmers to write a single source code, fitting at the same time for multiple environments
- High-level programming language support: the most popular programming languages as C++ or Python are supported in ROS with specifically designed libraries and functionalities enabling programmers to save time and enhance their productivity
- Availability of third-part libraries: beyond the aforementioned standard libraries, ROS accepts packages coming from different sources as well, enlarging its range of capabilities
- Easy prototyping: the availability of pre-defined robot models makes possible to adopt them on a variety of different projects with simple tweaking
- Community support: the ROS community is worldwide spread and along with its constant support, it maintains and develops packages and libraries to provide cutting-edge features every time

The first version of ROS was released in 2009, with the name of ROS 0.4. Soon later, in 2010, ROS 1.0 saw the light and its development continued for 5 years, until the appearance on the market of ROS2, which is the version used in this project. The sub-versions of each release are known as *distributions* and represent "versioned set of ROS packages" [65]. They are published following the alphabetical order, e.g. Foxy (2020), Galactic (2021), Humble (2022) and Iron (2023). The distribution adopted for the project is ROS2 Foxy.

As anticipated above, ROS offers an invaluable process intercommunication feature which allows many processes and programs to work and exist independently and exchange the useful data among each other. This feature is enabled by the structure of ROS itself based on nodes, messages, topics, services and actions.

Nodes are "processes that perform computation" and usually each system is composed by many nodes. Nodes communicate each other through *messages* which are formally defined as a "strictly typed data structure". Messages are

[9] This concept refers to the capability of executing different tasks independently but sharing the same process' resources

based on primitive data types (integer, floating point, Boolean, etc.) but can easily be constituted by arrays or compositions of these basilar types, nested arbitrarily deep. Each node "sends a message by publishing it to a given *topic*". A node that writes on a topic is called *publisher*, while a node that is interested and then reads the data sent on that topic is known as *subscriber*. Each topic can include multiple subscribers and/or publishers.

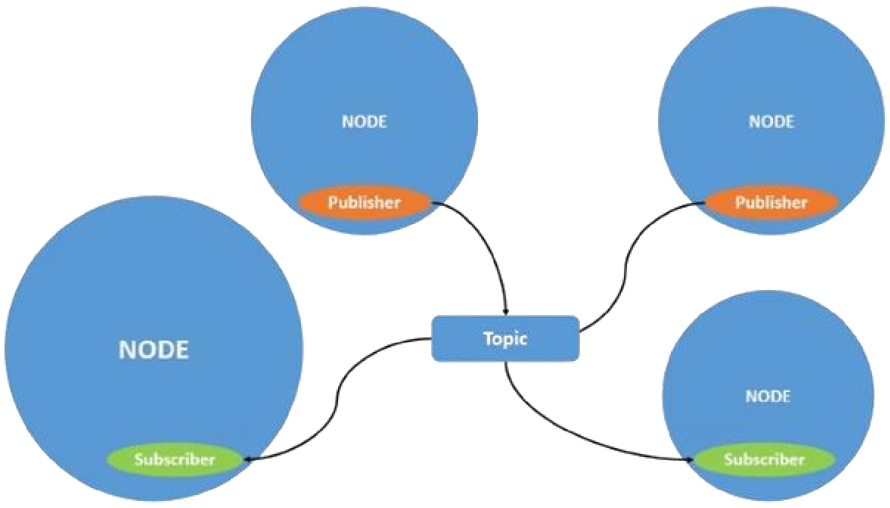


Figure 4.1: ROS nodes and topics - Source [74]

When more complex features are involved, like synchronous transactions, the nature of nodes and topics is not sufficient anymore. Hence, a more refined design of the node is required. This approach introduces the so-called *services* that are defined as "a string name and a pair of strictly typed messages: one for the request and one for the response". Note that unlike topics, services accept only one publisher and one subscriber. [66]

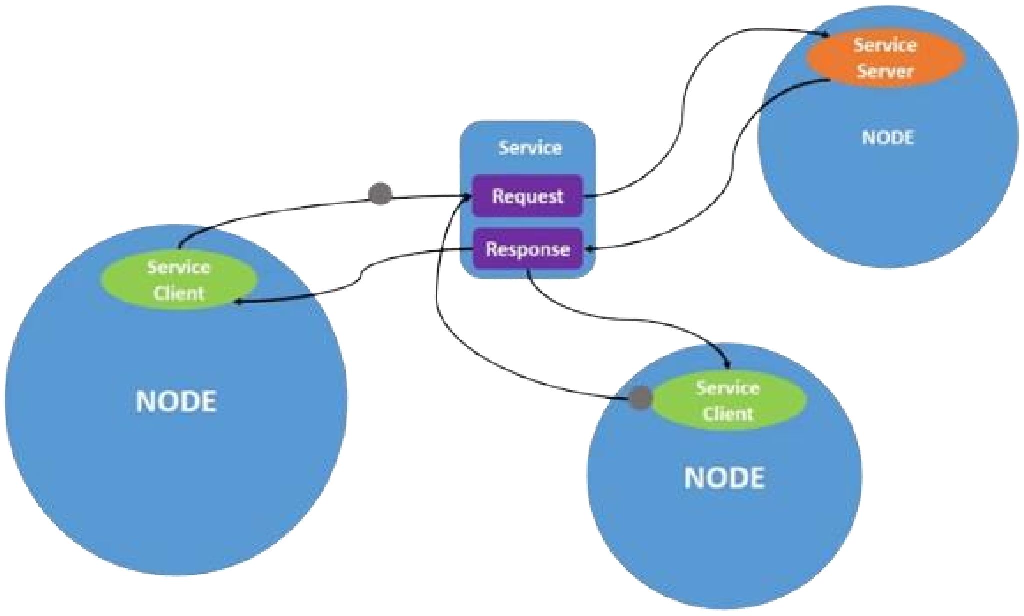


Figure 4.2: ROS services - Source [74]

The last communication type that is part of the Robotic Operating System framework is represented by the *actions*. They are designed for long-running tasks and are composed of three parts: goal, feedback and result. Actions are built on topics and services and their functionality is similar to them indeed. They differentiate from the services from their possibility to be preemptable (i.e. cancellable while executed) and the feedback that they provide in addition to the single response. Actions are based on a client-server model similar to the publisher-subscriber one described in Figure 4.1. An action server node sends a request (also known as goal) to the action server node, which replies with a response and a feedback data stream.

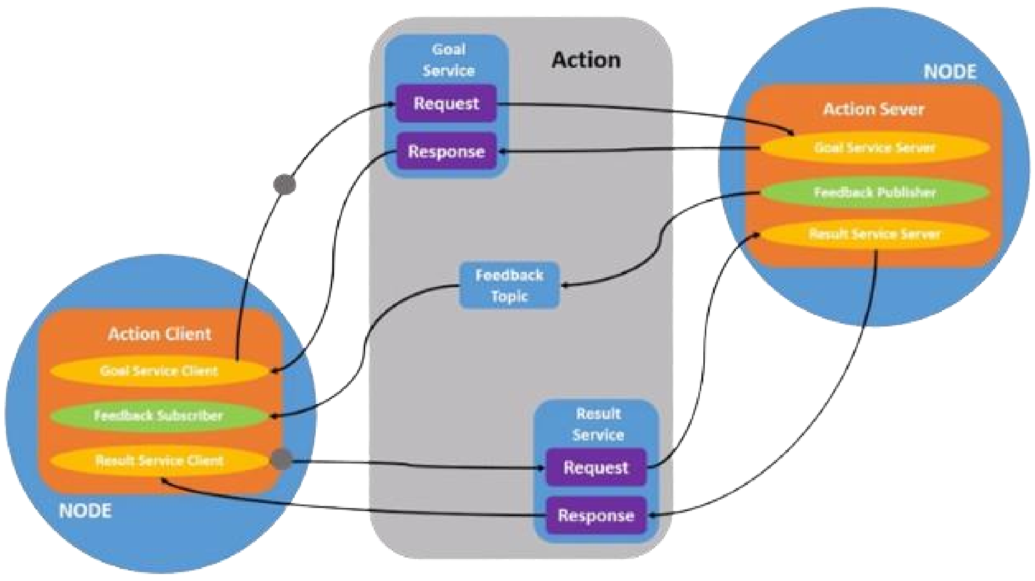


Figure 4.3: ROS actions - Source [74]

Given the complexity of the structure mentioned above, ROS offers a useful tool to visualize graphically the nodes and their interactions inside the system. It is known as `rqt` and represents an alternative graphical user interface to the command line to ease the interaction with the robot system. It includes many sub-tools as the `rqt_graph` and the `rqt_console`. The first prints down the active nodes, topics, services or actions in the system with a layout similar to the one shown in the pictures above. It enables the designer to immediately understand which are the publisher-subscriber relations and efficiently operate with them. The latter allows instead to introspect log messages. They are usually shown in the terminal, but with this GUI tool they can be collected over time, visualized in a more organized manner, filtered, and saved for further usage.

4.2 TF library

The TF library, acronym of Transform library, was designed to keep track of the reference frames in a complex robotic system, commonly recognized as a source of problems by many users. The task of this library is to manage the coordinate frames and transform data within a whole system in such a way that users working with individual component can be confident that the data is in the correct and desired reference frame without the need of being acquainted with all the ones in the system. When working with robots, whichever their level of complexity is, it is essential to be aware of their position over time and of the relative location of the objects in the space. Since any system is made up of many components, like sensors, actuators, rotating parts, motors, etc., each of them with its own coordinate frame, it becomes imperative to manage the existing relations between them to accomplish this task. In addition to that, as a robotic system grows in complexity the ability of any subsystem to be totally aware of the rest of the system reduces, and the designers of a single-standing component must properly take into account what information is necessary for their module to perform an action. The TF library does just that, since it enables programmers to know which is the relation between two or more RF of their interest without considering intermediate links or other components playing a side role in that specific task.

The library is constituted by two basic modules: the *broadcaster* and the *listener*. The first part is spreading transform data to the entire system. The second part receives the transform information and stores it for further usage. It is then able to provide an answer to the request about the resultant transform between different coordinate systems. More precisely, the listener fills a sorted list with the collected values and when queried can interpolate between the two nearest ones. Since the broadcaster sends transforms at regular intervals, the listener does not ever assume the future presence of a coordinate frame. The broadcasting frequency should be selected so as to be high enough that spherical linear interpolation (SLERP) is capable of approximating the joint motion between the two samples. [67]

The transform library can be closely related to scene graphs, a common data structure employed in 3D rendering. Scene graphs typically consist of a tree of objects connected with some relations of a kind. Every item is attached to a parent object with a position and another additional information. Depending on the application this second datum may vary from visualization meshes for pure rendering purposes to inertial properties for the simulators. Translating this concept to reference frames, the nodes inside the graph can be intended as coordinate systems and the connections between them represent instead the transformations. To enable immediate look ups the tree must be quickly inspectable. If the graph is limited to a tree this feature is met. This becomes important as graph complexity increases. An example of these trees is provided in Figure 4.4. One key distinction between the scene graph and the TF tree data structure lies in their respective purposes: the scene graph is intentionally designed for periodic iteration, whereas the if tree is tailored for asynchronous querying of specific values.

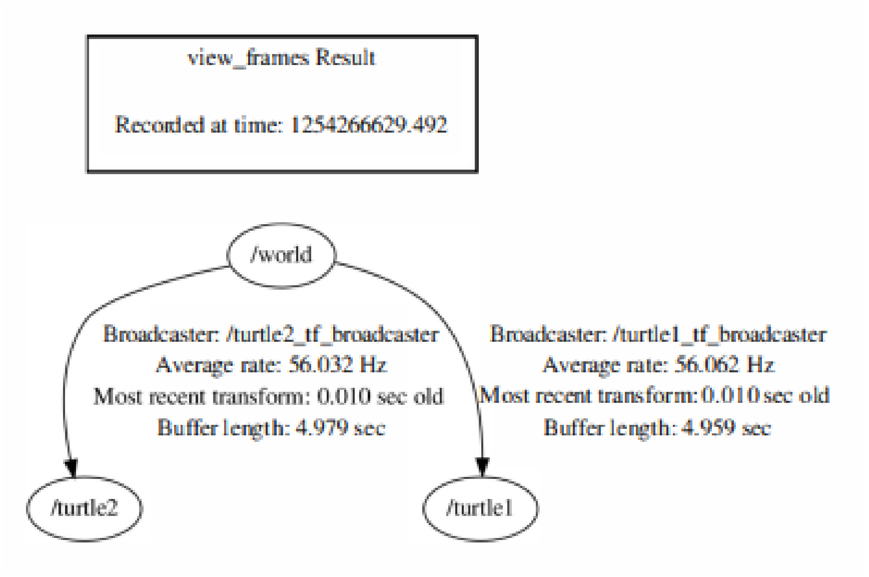

view_frames Result
Recorded at time: 1254266629.492
/world
Broadcaster: /turtle2_tf_broadcaster
Average rate: 56.032 Hz
Most recent transform: 0.010 sec old
Buffer length: 4.979 sec
Broadcaster: /turtle1_tf_broadcaster
Average rate: 56.062 Hz
Most recent transform: 0.010 sec old
Buffer length: 4.959 sec
/turtle2
/turtle1

Figure 4.4: tf example - Source [74]

Another benefit of the tree structure lies in the possibility to manage dynamic changes and continuous updates without the need of any additional information.

For the proper functioning of the if library, it is imperative that all data candidate to transformation includes two critical pieces of information: firstly, the coordinate frame in which it is initially defined, and secondly, the timestamp denoting its validity. Collectively, these two essential components are referred to as a *Header*. Any data incorporating this information can then undergo transformation for known data types.

The most critical step is represented by the transformation itself. When it comes to operate with two nodes of the tree only, the transformation is performed as described in 3.7 Coordinate conversion, while when multiple nodes are involved, the nature of the graph is exploited, considering the intermediate links. [4.1] shows an example of transform with 3 coordinate frames.

$$T_c^a = T_b^a \cdot T_c^b \qquad [\ 4.1\ ]$$

In the use case the TF library has been employed to manage the relations between the global reference frame in which the trajectory is written, the stereo camera RF and the vehicle one, to properly considering its heading.

4.3 RVIZ

ROS plugin RVIZ serves as a powerful and versatile visualization platform that plays a pivotal role in facilitating the development, debugging, and testing of robotic applications. RVIZ, short for "ROS Visualization," offers an extensive array of features and functionalities meticulously designed to assist engineers and researchers. This dynamic tool provides a comprehensive three-dimensional visualization environment where users can interactively display and manipulate a wide range of data generated by robots and their associated sensors. RVIZ's versatility allows for the simultaneous visualization of robot models, sensor data, occupancy grids, point clouds, and trajectory paths, among others, thus enabling a comprehensive understanding of the robot's behavior and environment. Its user-friendly graphical interface, accompanied by a vast library of visualization displays, markers, and configuration options, empowers users to tailor the visualization environment precisely to their specific needs.

Furthermore, while RVIZ primarily caters to the visualization needs of robotics and engineering effort, its applications extend into industrial engineering and beyond. It serves as a foundational tool in the development of automation systems, industrial robots, and other mechatronic applications.

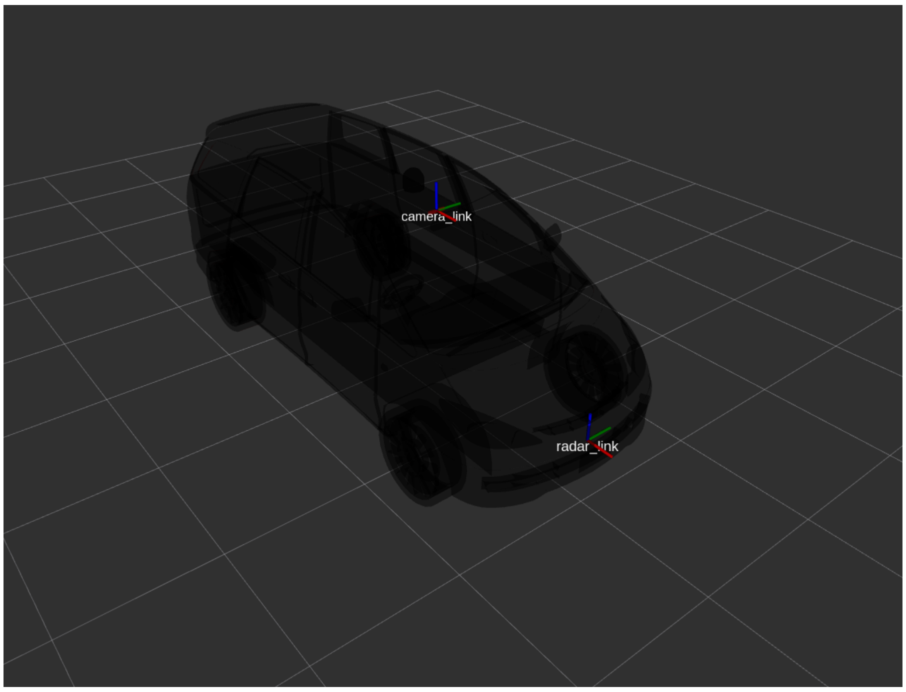


Figure 4.5: Vehicle representation in RVIZ

In Figure 4.5 it is possible to visualize two of the different reference frames playing a role in the project. In RVIZ environment they are identified as *links* and here the camera and the radar ones are showed. Despite the actual presence of three cameras installed on the dashboard of the vehicle, the process of stereo vision allows to merge their outputs as if it was one single perception element, whose reference frame is centered in the position depicted by the figure.

4.4 URDF

The definition of the vehicle's model seen in Figure 4.5 has been possible thanks to a powerful tool offered by ROS: URDF. Acronym for *Unified Robot*

Description Format, it represents a standardized format for robot modelling in a simulation environment. Its origins coincide with the appearance of ROS in 2009 and since then it is employed to describe the "kinematics, dynamics, and geometries of robots, independently of software programs" [68]. This unified description comes with a file with .urdf extension which is accepted by a large variety of tools. It is a human-readable XML file which contains indeed the dynamic and kinematic parameters of the robot, its visual representation and the employed collision model. The robot model is organized in links and joints. The former indicate rigid bodies with inertia and mass properties, identified by a reference frame. The latter refer instead to the entities connecting the links and can be of various types (planar, revolute, continuous, etc.).

The visual description of the robot is instead made possible by the *URDF Bundle*. It includes the urdf file itself and the mesh file describing the 3D geometry of the model. Also in this case, many extensions are accepted such as `.dae`, `.obj` or `.stl`.

It comes without saying that the vehicle model used in this project cannot be strictly compared to a robot, but with the proper tweaking, its URDF model has been created. First, the shape of the vehicle has been imported from a mesh file, then the position of the camera and the radar sensor has been fixed to determine the respective reference frames. See A5 - URDF file to visualize the urdf file.

4.4 GPS Emulator

As clearly deducible from the description of the project provided so far, the whole algorithm relies on the acquisition of the GPS signal from the devoted sensor installed on the eWolf. It represents the starting point for the map-matching phase, which plays a pivotal role in the lane detection process.

Unfortunately, the sensor was not yet available at the time of this project, thus an emulator was utterly needed in order to test the performance and the capabilities of the designed system. The first step consists in a manual extraction of a series of geographical coordinates from the OpenStreetMap website around the trajectory run by the test vehicle. Such trajectory has been retrieved from the recorder stereo camera images employed in the simulation environment of ROS, while the logic of extraction of the coordinates tries to resemble the sparse nature of the GPS acquisition. After this non-automatized phase, the emulator employs a function defined above in 3.5.1 Point density increment, precisely the one with fixed density parameter. It fills the gaps between each location obtained on the map with the desired number of points. The result is a series of geographical coordinates almost randomly distributed around the trajectory followed by the eWolf. It does not exactly replicate the behavior of a real GPS sensor but provides a great base for the following steps.

Figure 4.6 offers a graphical representation of the aforementioned concept.

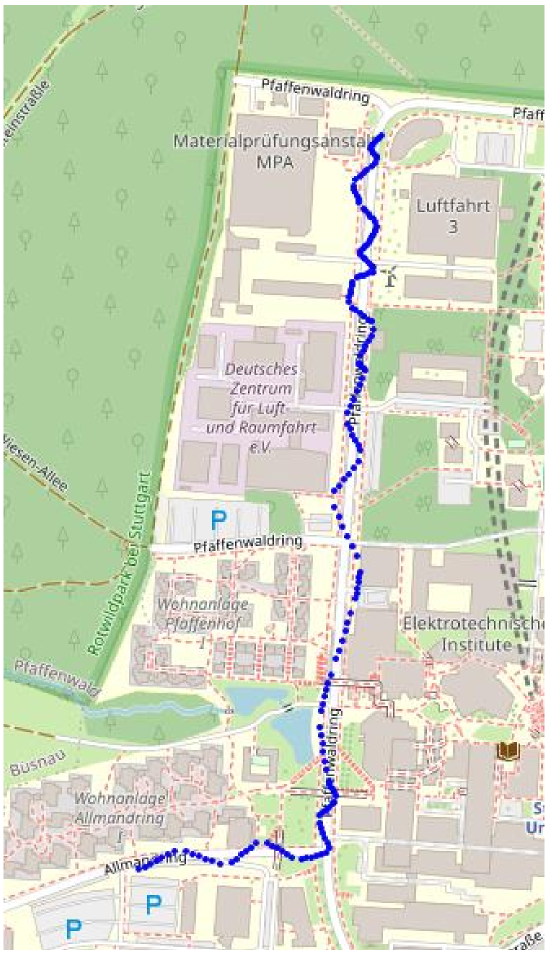


Figure 4.6: Emulated GPS coordinates around the followed path

4.5 IMU Emulator

As well as the GPS signal, the Inertial Measurement Unit output plays a crucial role in the algorithm framework. It is necessary to compare the current heading of the vehicle (sometimes also referred to as *yaw angle* as seen in 3.8 Vehicle lateral dynamics control) to the direction imposed by the reference trajectory. If an ideal situation is considered, this signal is provided by a specifically designed sensor as a digital quantity that once elaborated returns the yaw, pitch and roll angles of the vehicle with a predefined frequency.

Nevertheless, as with the GPS sensor, this information was not available at the moment of the project, so the design of an emulator was needed also in this case. It starts from the acquisition of the reference directions embedded in the trajectory data generated by the `path_finder` module. Then a random float number in the range `[2; −2]` is subtracted to each direction value to emulate the data acquired by the IMU sensor instant by instant.

This leads to the final goal of comparing the reference direction and the simulated IMU output to provide a correction for the heading of the vehicle, quantity used as input in the lateral dynamics controller.

4.6 Actual project implementation

With thanks to the previous introductory part, it is now easier to understand the framework of the project in the Robot Operating System. At the beginning an overview on the elements and the content of the different topics is given, followed by an in-depth study of each node.

4.6.1 RQT Graph

As reported in 4.1 ROS Basics, the `rqt_graph` tool enables the programmer and other users to visualize the connections between nodes and topics inside the ROS framework. It comes particularly useful when such connections are multiple, and it is not immediately evident which node is subscribed or publishes to which topic. Figure 4.7: rqt graph for the case-study shows the output of the `rqt_graph` command for the case-study.

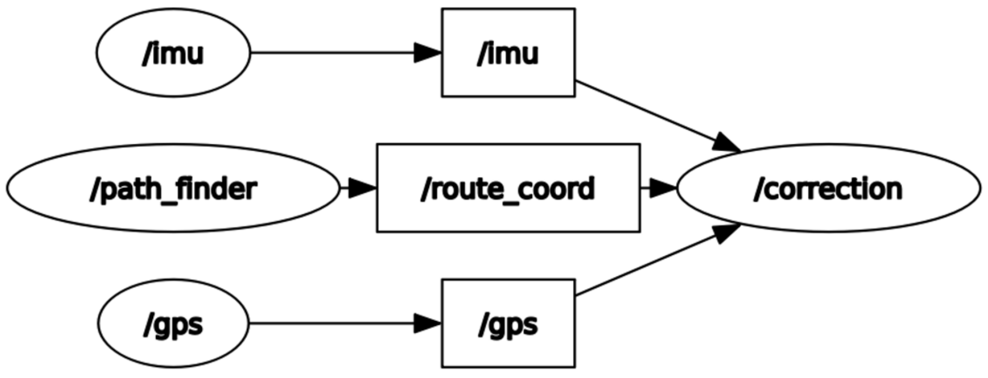


Figure 4.7: rqt graph for the case-study

The blocks depicted with ovals represent the nodes, while the rectangular entities stand for the topics. Analyzing the arrows, it can be stated that either imu, path_finder and gps nodes only publish on one topic each, respectively "/imu", "/route_coord" and "/gps". The first carries a standard message of type Float32 representing the emulated yaw angle caught from the IMU sensor. The second one contains a user-defined message of type Mypath2:

```
# custom_messages/msg/Mypath2.msg

# Indication of the frame where the trajectory is
published

std_msgs/Header frame_id

# Definition of the array
MyPose[] poses
```

It comprises two main components: a header which identifies the frame where the trajectory is published and a list of custom messages of type MyPose:

```
# custom_messages/msg/MyPose.msg

float64 x
float64 y
float64 z
float64 theta
```

Each item in the previous list is a set of four floating point values representing a point of the trajectory in terms of its x, y, and z[10] coordinates with the addition of the reference direction towards its subsequent. As the first topics, "/gps" is populated by a message of type Float32 as well, constituting the output of the GPS receiver emulator.

4.6.2 Imu node

The task of this node is very simple yet fundamental. As described in 4.5 IMU Emulator it reads a text file containing a list of values and, after converting them to floating-point values, publishes on the appropriate topic what it has just read. See appendix A1 - IMU emulator for the full script.

4.6.3 GPS node

Its working principle is strictly similar to the imu node since it is an emulator as well, but with the difference that the output values are not read from a text file but are produced starting from a list of manually extracted coordinates later refined with a point density increment function. See appendix A2 – GPS emulator for the full script.

4.6.4 Path_finder node

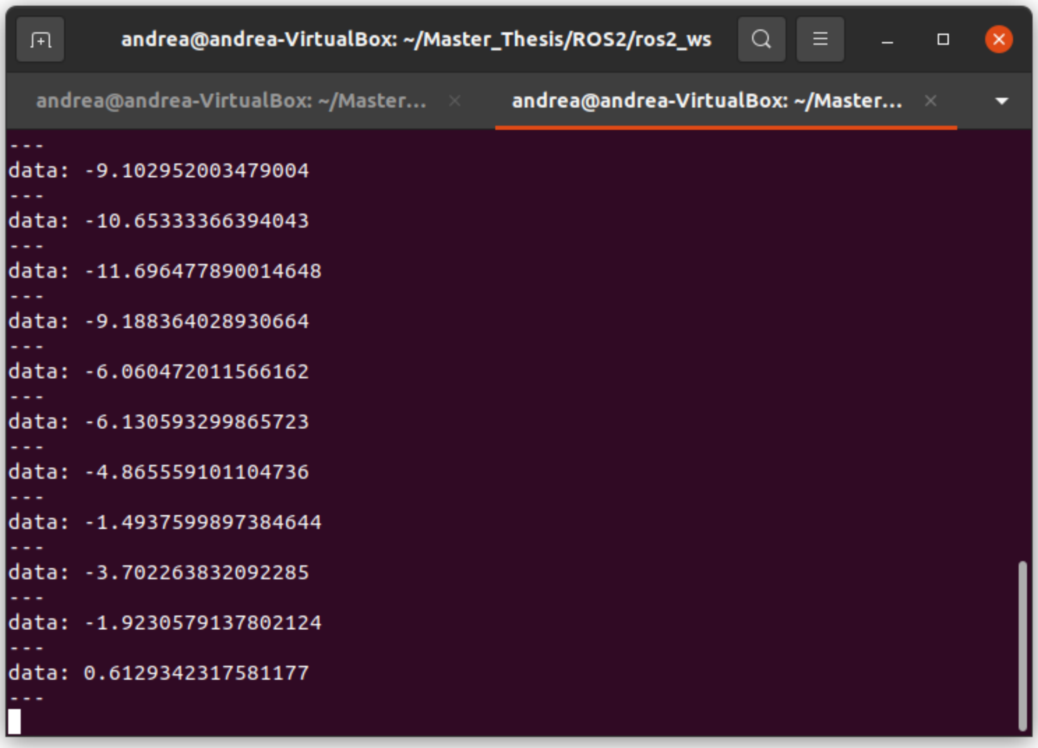
Even though the output of this node may seem simple, it is the result of a large set of nested functions. First, it must fetch the file containing the portion of digital map of interest, then it applies the A* search algorithm to find the set of nodes constituting the shortest path between the two input locations. After that, the trajectory points density is incremented using the

[10] Always equal to zero since the vehicle is supposed to work in 2D

functions outlined in 3.5.1 Point density increment, and finally the spline method can be applied to sharp the corners of the route. Nevertheless, this only produces the geographical coordinates of each point of the trajectory, so a further step is needed to enrich these data with the reference directions. See appendix A3 – Path_finder for the full script.

4.6.5 Correction node

The operation of this node is the core of the whole framework since it produces the control quantity that is fed to the controller which rules the lateral dynamics of the vehicle. Subscribing to the imu, gps and route topics it fetches the information about these quantities to complete a series of steps. First, this node checks the correct reception of the three messages and initializes appropriate variables with them once the check is completed. If the three variables do not contain null values, the first operation is carried out, i.e. the research of the closest point on the trajectory based on the GPS signal. As soon as it is found, the direction associated to it is compared with the IMU signal and their difference in degrees is published on the output topic to feed the controller block. This output can be visualized in Figure 4.8. See appendix A4 – Correction for the full script.



andrea@andrea-VirtualBox: ~/Master_Thesis/ROS2/ros2_ws
andrea@andrea-VirtualBox: ~/Master...
andrea@andrea-VirtualBox: ~/Master...

```
---
data: -9.102952003479004
---
data: -10.65333366394043
---
data: -11.696477890014648
---
data: -9.188364028930664
---
data: -6.060472011566162
---
data: -6.130593299865723
---
data: -4.865559101104736
---
data: -1.4937599897384644
---
data: -3.702263832092285
---
data: -1.9230579137802124
---
data: 0.6129342317581177
---
```

Figure 4.8: /correction topic content

Conclusions and future work

At this point of the dissertation, it is possible to analyze the goals that have been met and the ones that require additional work on.

The aim of the project was to develop an algorithm able to cooperate with the camera-based lane detection system and aid it in its failure situations. This requirement has been successfully met since the proposed method provides high-level guidance to the eWolf, empowering it to overcome intersections, roundabouts or other complex driving scenarios leveraging on the concept of map-matching. After gathering data from the GPS and the IMU sensors, the algorithm is able to acquaint the Electronic Control Unit of the vehicle with the right direction to follow, comparing the pose of the vehicle itself with the reference trajectory.

The achievement of this result has been possible thanks to the design of two emulators, aimed at simulating the behavior of real sensors. It is the case of the Global Positioning System and the Inertial Measurement Unit sensors. There is much space for development in this domain since the utilization of non-simulated data makes possible to evaluate the behavior of the system in the real environment with higher reliability. Additionally, the IMU sensor can provide a large number of supplementary details which can be employed to estimate the pose of the vehicle more accurately.

Furthermore, the Robot Operating System simulation environment turned out to require high-performing hardware to run and visualize the results properly. Hence, due to lack of sufficiently-capable parts of the available equipment, it has not been possible to plot the output of the algorithm in a more user friendly way, rather than just a bunch of numbers on the command line.

In the end, a validation phase is needed to test the designed system on the real vehicle. In fact, this is the only way to assess its capabilities and performances in a real-world scenario, closing the V-cycle of the product.

Appendix

A1 - IMU emulator node

```python
# import statements
import rclpy
from rclpy.node import Node
from std_msgs.msg import Float32

class IMU(Node):

    # Node initialization
    def __init__(self, filename):
        super().__init__("imu_api")
        self.imu_pub = self.create_publisher(Float32, '/imu', 10)
        self.timer_ = self.create_timer(1, self.spinner)
        self.filename = filename
        self.i = 0

    # Creation of the function that every second publishes the value of
the yaw angle reading from the default file, emulating the IMU sensor
    def spinner(self):
        msg = Float32()
        imu_list = []
        with open(self.filename, 'r') as f:
            lines = f.readlines()
            for line in lines:
                theta = float(line)
                imu_list.append(theta)
        msg.data = imu_list[self.i]
        self.imu_pub.publish(msg)
        self.i += 1

def main(args=None):
    rclpy.init(args=args)
    filename =
'/home/andrea/Master_Thesis/ROS2/ros2_ws/src/my_traj/my_traj/thetas_ran
domized.txt'
    node = IMU(filename)
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

A2 – GPS emulator node

```
# import statements
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Point
import utm
import numpy as np

class GPS(Node):

    # Node initialization with the set of coordinates
    def __init__(self):
        super().__init__("gps_api")
        self.coords = [(48.75093,9.10330),
               (48.75081,9.10311),
               (48.75070,9.10323),
               (48.75050,9.10288),
               (48.75030,9.10299),
               (48.75014,9.10322),
               (48.74999,9.10311),
               (48.74990,9.10296),
               (48.74974,9.10320),
               (48.74967,9.10290),
               (48.74944,9.10282),
               (48.74927,9.10320),
               (48.74895,9.10295),
               (48.74886,9.10306),
               (48.74845,9.10278),
               (48.74813,9.10303),
               (48.74777,9.10258),
               (48.74717,9.10300),
               (48.74684,9.10291),
               (48.74629,9.10262),
               (48.74585,9.10244),
               (48.74534,9.10255),
               (48.74515,9.10269),
               (48.74499,9.10242),
               (48.74469,9.10260),
               (48.74462,9.10216),
               (48.74480,9.10169),
               (48.74457,9.10122),
               (48.74472,9.10061),
               (48.74454,9.10007)]

        self.gps_pub = self.create_publisher(Point, '/gps', 10)
        self.timer_ = self.create_timer(1, self.spinner)
        self.i = 0

    # Function used to add points between every item in the list above
    def increase_density_fixed(self, route, density=4):
        smoothed_x, smoothed_y = [], []

        for (x0, y0), (x1, y1) in zip(route, route[1:]):
            smoothed_x.extend([x0 + k * (x1 - x0) / density for k in
range(1, density)])
            smoothed_y.extend([y0 + k * (y1 - y0) / density for k in
range(1, density)])

        smoothed_x.append(route[-1][0])
        smoothed_y.append(route[-1][1])
        route = list(zip(smoothed_x, smoothed_y))

        return route

    # Cooridnates conversion to UTM values for better handling
    def to_utm(self, route):
        x, y = zip(*route)
        out = utm.from_latlon(np.array(x), np.array(y))
        route_conv = list(zip(out[0], out[1]))

        return route_conv

    # Function publishing the GPS coordinate every second, emualting
the GPS receiver
    def spinner(self):
        msg = Point()
        route_new =
self.to_utm(self.increase_density_fixed(self.coords))
        msg.x = route_new[self.i][0]
        msg.y = route_new[self.i][1]
        self.gps_pub.publish(msg)
        self.i += 1

def main(args=None):
    rclpy.init(args=args)
    node = GPS()
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

A3 – Path_finder node

```
# import statements
import rclpy
from rclpy.node import Node
from pyroutelib3 import Router
import numpy as np
import bezier
import math
from custom_messages.msg import Mypath2, MyPose
import utm
from scipy.interpolate import splprep, splev

class MyPathFinder(Node):
    # Node initialization with the default locations and map
    def __init__(self):
        super().__init__("path_finder")
        self.declare_parameter('source_geo', [48.75115,9.10454])
        self.declare_parameter('dest_geo', [48.74422,9.09843])
        self.declare_parameter('filename', '/home/andrea/campus.osm') #
change directory according to the position of the .osm file in your pc
        start =
self.get_parameter('source_geo').get_parameter_value().double_array_val
ue
        end =
self.get_parameter('dest_geo').get_parameter_value().double_array_value
        filename =
self.get_parameter('filename').get_parameter_value().string_value
        self.route_publisher_ = self.create_publisher(Mypath2,
'/route_coord',10)
        timer_period = 1  # seconds
        self.timer = self.create_timer(timer_period, lambda:
self.router_callback(start, end, filename))

    # Function generating the reference trajectory
    def router_callback(self, source_geo, dest_geo, filename):

        # Message initialization
        msg = Mypath2()

        # Initialize the router with the desired mean of transport and
the map file
        router = Router('car', filename)
        start = router.findNode(source_geo[0],source_geo[1]) # Find
start and end nodes
        end = router.findNode(dest_geo[0],dest_geo[1])

        status, route = router.doRoute(start, end) # Find the route - a
list of OSM nodess

        if status == 'success':
            routeLatLons = list(map(router.nodeLatLon, route)) # Get
actual route coordinates
```

```
        # Increase the density of points
        distances = [math.dist((x0, y0), (x1, y1)) for (x0, y0), (x1,
y1) in zip(routeLatLons, routeLatLons[1:])]  # Calculate distances
        minimum = min(distances)

        # Precompute range values
        ranges = [range(math.floor(dist / minimum)) for dist in
distances]

        augm_x, augm_y = [], []
        for (x0, y0), (x1, y1), r in zip(routeLatLons,
routeLatLons[1:], ranges):
            augm_x.extend(np.linspace(x0, x1, len(r)))
            augm_y.extend(np.linspace(y0, y1, len(r)))

        routeLatLons = list(zip(augm_x, augm_y))
        routeLatLons = self.to_utm(routeLatLons)

        # Create the spline
        no_rep = list(dict.fromkeys(routeLatLons))
        coords = np.array(no_rep)
        tck, u = splprep(coords.T, s=25)
        new_points = splev(u, tck)
        routeLatLons = list(zip(new_points[0], new_points[1]))

        # Define the message
        poses = []
        for coord in routeLatLons:
            pose = MyPose()
            pose.x = coord[0]
            pose.y = coord[1]
            pose.theta = self.yaw(routeLatLons, coord)
            poses.append(pose)

        msg.poses = poses
        msg.frame_id.stamp = self.get_clock().now().to_msg()
        msg.frame_id.frame_id = 'map'

        self.route_publisher_.publish(msg)

    # Cooridnates conversion to UTM values for better handling
    def to_utm(self, route):
        x, y = zip(*route)
        out = utm.from_latlon(np.array(x), np.array(y))
        route_conv = list(zip(out[0], out[1]))

        return route_conv

    # Function determining the angle between two consecutive points
    def yaw(self, traj, pt):
        ind = traj.index(pt)
        if ind + 1 < len(traj):
            next_point = traj[ind + 1]
            delta_x, delta_y = next_point[0]-pt[0], next_point[1]-pt[1]
            yaw = math.degrees(math.atan2(delta_y, delta_x))
        else:
            yaw = 0.0

        return yaw


def main(args=None):
    rclpy.init(args=args)
    node = MyPathFinder()
    # node.use_parameters()
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

A4 – Correction node

```
# import statements
import rclpy
from rclpy.node import Node
from custom_messages.msg import Mypath2
from geometry_msgs.msg import Point
from std_msgs.msg import Float32
```

```
import math


class Correction(Node):

    # Node initialization with the subscription to the desired topics
    def __init__(self):
        super().__init__("correction")
        self.pose_subscriber = self.create_subscription(Mypath2,
'/route_coord', self.route_callback, 10)
        self.gps_subscriber = self.create_subscription(Point, '/gps',
self.gps_callback, 10)
        self.imu_subscriber = self.create_subscription(Float32, '/imu',
self.imu_callback, 10)
        self.correction_publisher = self.create_publisher(Float32,
'/correction',10)

        # Initialization of the GPS, path and IMU messages
        self.latest_gps_msg = None
        self.latest_route_msg = None
        self.latest_imu_msg = None
        self.timer = self.create_timer(1, self.try_compute_and_publish)

    # Verify the correct reception of the GPS message
    def gps_callback(self, msg: Point):
        self.latest_gps_msg = msg
        self.try_compute_and_publish()
        # self.get_logger().info("I heard: " +
str(self.latest_gps_msg))

    # Verify the correct reception of the route message
    def route_callback(self, msg: Mypath2):
        self.latest_route_msg = msg
        self.try_compute_and_publish()
        # self.get_logger().info('I heard: "%s"' % msg.poses[1])

    # Verify the correct reception of the IMU message
    def imu_callback(self, msg: Float32):
        self.latest_imu_msg = msg
        self.try_compute_and_publish()
        # self.get_logger().info("I heard: " +
str(self.latest_imu_msg))

    # Once correctly received all the messages, compute the correction
to be applied to vehicle's heading
    def try_compute_and_publish(self):
        if self.latest_gps_msg is not None and self.latest_route_msg is
not None and self.latest_imu_msg is not None:
            result = self.correction(self.latest_route_msg,
self.latest_gps_msg, self.latest_imu_msg)
            self.correction_publisher.publish(result)
            self.latest_gps_msg = None
            self.latest_route_msg = None
            self.latest_imu_msg = None

    def correction(self, msg1: Mypath2, msg2: Point, msg3: Float32):

        # Message initialization
        out = Float32()

        # Research of the closest point on trajectory, compared to the
GPS singal
        closest_point = None
        min_distance = float('inf')
        buffer_radius = 20
        for i in range(len(msg1.poses)):
            distance = math.sqrt((msg1.poses[i].x - msg2.x)**2 +
(msg1.poses[i].y - msg2.y)**2)

            if distance <= buffer_radius and distance < min_distance:
                min_distance = distance
                closest_point = msg1.poses[i]

        # Retrieval of the correct direction value
        theta_correct = closest_point.theta

        # Retrieval of the emulated IMU sinal
        theta_actual = msg3.data

        # Application of the correction
        out.data = theta_actual-theta_correct
        return out

def main(args=None):
    rclpy.init(args=args)
    node = Correction()
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

A5 - URDF file

```
<?xml version="1.0"?>
<robot name="car" xmlns:xacro="http://ros.org/wiki/xacro">
  <!-- Define car constants -->
  <xacro:property name="base_width" value="2.5"/>
  <xacro:property name="base_length" value="5.0"/>
  <xacro:property name="base_height" value="1.8"/>

  <!-- Car Base -->
  <link name="base_link">
    <visual name="base_visual">
      <origin xyz="1.0 0.0 0.0" rpy="1.57 0 4.71" />
      <geometry>
        <mesh filename="package://my_model/meshes/estima_white.stl"
scale="0.01 0.01 0.01"/>
      </geometry>
    </visual>

    <collision>
      <geometry>
        <box size="${base_length} ${base_width} ${base_height}"/>
      </geometry>
    </collision>
  </link>

  <!-- Footprint -->
  <link name="base_footprint"/>

  <joint name="base_joint" type="fixed">
    <parent link="base_link"/>
    <child link="base_footprint"/>
    <origin xyz="1.0 0.0 0.0" rpy="0 0 0"/>
  </joint>

  <!-- Sensor Link -->
  <!-- IMU Link -->
  <!--
  <link name="imu_link">
    <visual>
      <geometry>
        <box size="0.1 0.1 0.1"/>
      </geometry>
    </visual>

    <collision>
      <geometry>
        <box size="0.1 0.1 0.1"/>
      </geometry>
    </collision>
  </link>

  <joint name="imu_joint" type="fixed">
    <parent link="base_link"/>
    <child link="imu_link"/>
    <origin xyz="0 0 1.5"/>
  </joint>
  -->

  <!-- radar Link -->
  <link name="radar_link">
    <inertial>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <mass value="0.125"/>
      <inertia ixx="0.001"  ixy="0"  ixz="0" iyy="0.001" iyz="0"
izz="0.001" />
    </inertial>
```

```xml
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
         <cylinder radius="0.0508" length="0.055"/>
      </geometry>
    </collision>

    <visual>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
         <cylinder radius="0.0508" length="0.055"/>
      </geometry>
    </visual>
  </link>

  <joint name="radar_joint" type="fixed">
    <parent link="base_link"/>
    <child link="radar_link"/>
    <origin xyz="3.0 0 0.5" rpy="0 0 0"/>
  </joint>

  <!-- camera Link -->
  <link name="camera_link">
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <box size="0.015 0.130 0.022"/>
      </geometry>
    </visual>

    <collision>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <box size="0.015 0.130 0.022"/>
      </geometry>
    </collision>

    <inertial>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <mass value="0.035"/>
      <inertia ixx="0.001"  ixy="0"  ixz="0" iyy="0.001" iyz="0"
izz="0.001" />
    </inertial>
  </link>

  <joint name="camera_joint" type="fixed">
    <parent link="base_link"/>
    <child link="camera_link"/>
    <origin xyz="1.5 0 1.5" rpy="0 0 0"/>
  </joint>


</robot>
```

Bibliography

[1] L. F. A. León and Y. Aoyama, "Industry emergence and market capture: The rise of autonomous vehicles," *Technol Forecast Soc Change*, vol. 180, p. 121661, 2022.

[2] R. Fan, J. Jiao, H. Ye, Y. Yu, I. Pitas, and M. Liu, "Key Ingredients of Self-Driving Cars," Jun. 2019, [Online]. Available: http://arxiv.org/abs/1906.02939

[3] Fabio Sciarra, "La corsa è ripartita," *Quattroruote* , pp. 156–157, Aug. 2023.

[4] J.-A. Pattinson, H. Chen, and S. Basu, "Legal issues in automated vehicles: critically considering the potential role of consent and interactive digital interfaces," *Humanit Soc Sci Commun*, vol. 7, no. 1, p. 153, Nov. 2020, doi: 10.1057/s41599-020-00644-2.

[5] D. Parekh *et al.*, "A Review on Autonomous Vehicles: Progress, Methods and Challenges," *Electronics (Basel)*, vol. 11, no. 14, p. 2162, Jul. 2022, doi: 10.3390/electronics11142162.

[6] © SAE International, "SURFACE VEHICLE RECOMMENDED PRACTICE Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles," *SAE J3016*, 2021.

[7] R. Fan, L. Wang, M. J. Bocus, and I. Pitas, "Computer stereo vision for autonomous driving," *arXiv preprint arXiv:2012.03194*, 2020.

[8] N. Ahmad, R. A. R. Ghazilla, N. M. Khairi, and V. Kasi, "Reviews on various inertial measurement unit (IMU) sensor applications," *International Journal of Signal Processing Systems*, vol. 1, no. 2, pp. 256–262, 2013.

[9] S. Thrun *et al.*, "Stanley: The robot that won the DARPA Grand Challenge.," *J. Field Robotics*, vol. 23, pp. 661–692, Jan. 2006.

[10] M. Montemerlo *et al.*, "Junior: The Stanford Entry in the Urban Challenge," *J Field Robot*, vol. 25, pp. 569–597, Sep. 2008, doi: 10.1002/rob.20258.

[11] C. Urmson *et al.*, “Autonomous driving in urban environments: Boss and the Urban Challenge.,” *J. Field Robotics*, vol. 25, pp. 425–466, Jan. 2008.

[12] W. Zong, C. Zhang, Z. Wang, J. Zhu, and Q. Chen, “Architecture Design and Implementation of an Autonomous Vehicle,” *IEEE Access*, vol. 6, pp. 21956–21970, 2018, [Online]. Available: https://api.semanticscholar.org/CorpusID:13719357

[13] V. Badrinarayanan, A. Kendall, and R. Cipolla, “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation,” *IEEE Trans Pattern Anal Mach Intell*, vol. 39, no. 12, pp. 2481–2495, 2017, doi: 10.1109/TPAMI.2016.2644615.

[14] P. and B. T. Ronneberger Olaf and Fischer, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, J. and W. W. M. and F. A. F. Navab Nassir and Hornegger, Ed., Cham: Springer International Publishing, 2015, pp. 234–241.

[15] R. Smith and P. Cheeseman, “On the Representation and Estimation of Spatial Uncertainty,” *Int J Rob Res*, vol. 5, Feb. 1986, doi: 10.1177/027836498600500404.

[16] S. Lefèvre, D. Vasquez, and C. Laugier, “A survey on motion prediction and risk assessment for intelligent vehicles,” *ROBOMECH Journal*, vol. 1, no. 1, p. 1, 2014, doi: 10.1186/s40648-014-0001-z.

[17] C.-E. Framing, F.-J. Heßeler, and D. Abel, “Learning scenario-specific vehicle motion models for intelligent infrastructure applications,” *IFAC-PapersOnLine*, vol. 52, no. 8, pp. 111–117, 2019, doi: https://doi.org/10.1016/j.ifacol.2019.08.057.

[18] M. Schreier, V. Willert, and J. Adamy, “An Integrated Approach to Maneuver-Based Trajectory Prediction and Criticality Assessment in Arbitrary Road Environments,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 10, pp. 2751–2766, 2016, doi: 10.1109/TITS.2016.2522507.

[19] M. Araki, “PID control,” *Control Systems, Robotics and Automation: System Analysis and Control: Classical Approaches II*, pp. 58–79, 2009.

[20] G. C. G. S. F. Graebe and M. E. Salgado, “Control system design.” 2000.

[21] M. Morari, C. E. Garcia, and D. M. Prett, “Model predictive control: theory and practice,” *IFAC Proceedings Volumes*, vol. 21, no. 4, pp. 1–12, 1988.

[22] elmexx, “LaneDet.” Accessed: Dec. 20, 2023. [Online]. Available: https://github.com/elmexx/eWolf_ROS2_LaneDetection

[23] P. Xingang, S. Jianping, L. Ping, W. Xiaogang, and T. Xiaoou, “Spatial As Deep: Spatial CNN for Traffic Scene Understanding,” Hong Kong , 2017.

[24] Z. Qin, H. Wang, and X. Li, “Ultra Fast Structure-aware Deep Lane Detection,” in *The European Conference on Computer Vision (ECCV)*, 2020.

[25] Z. Tu *et al.*, “RESA: Recurrent Feature-Shift Aggregator for Lane Detection,” 2020.

[26] L. Tabellini, R. Berriel, T. M. Paix, C. Badue, A. Ferreira De Souza, and T. Oliveira-Santos, “Keep your Eyes on the Lane: Real-time Attention-guided Lane Detection,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[27] L. Liu, X. Chen, S. Zhu, and P. Tan, “CondLaneNet: a Top-to-down Lane Detection Framework Based on Conditional Convolution,” 2021.

[28] R. B. Langley, “The UTM Grid System,” 1998.

[29] “OpenStreetMap Wiki.” Accessed: Dec. 20, 2023. [Online]. Available: https://wiki.openstreetmap.org/wiki

[30] “OpenStreetMap.” Accessed: Dec. 20, 2023. [Online]. Available: https://www.openstreetmap.org

[31] C. Pan, X. Zhou, and C. Yang, “Study on mobile terminals adaptive matching algorithm based on road bounding box,” *Journal of Geomatics*, pp. 23–26, 2013.

[32] Y. Si, R. Li, and Meng, “An Improved Road-Matching Algorithm.,” *Journal of Geomatics Science and Technology*, pp. 438–442, 2010.

[33] N. R. Velaga, M. A. Quddus, and A. L. Bristow, “Developing an Enhanced Weight-Based Topological Map-Matching Algorithm for Intelligent Transport Systems,” *Transport. Res.C-Emer.*, pp. 672–683, 2009.

[34] G. Jiang, M. Li, and Z. Zhu, “Map Matching Algorithm Based on Topological Relations for Urban Bus Intelligent Monitoring System,” *Microcomputer Information*, pp. 118–120, 2009.

[35] J. C. Zhang, Y. H. Wang, and W. J. Zhao, “An improved probabilistic relaxation method for matching multi-scale road networks,” *Int J Digit Earth*, pp. 635–655, 2018.

[36] Y. Yang, Z. Gu, L. Hu, Z. Rong, and G. Luo, “Map Matching Algorithm for Vehicle Navigation System Based on Probability Decision Rule,” *Aut.Eng*, pp. 897–901, 2006.

[37] Y. Li, X. Zhang, and Y. Bao, “Algorithm on Real-Time Map-Matching of Multi-Weight Probability,” *J. Electr. Measur.Instr.*, pp. 166–170, 2012.

[38] Y. Zhou and Y. Cheng, “Map-Matching Algorithm Based on Curvefitting Model.,” *J. Transp. Syst. Eng. Inf. Technol.*, pp. 68–70, 2004.

[39] J. Bi, Y. Zhu, and Y. Cheng, “A Comprehensive Map Matching Algorithm Based on Curve Fitting and Network Topology,” *Journal of Transport Information & Safety.*, pp. 127–131, 2014.

[40] H. Su, G. Wang, and J. Wang, “Map Matching Algorithm Based on Fuzzy Neural Networks,” *Journal of University of Science and Technology Beijing.*, pp. 43–47, 2012.

[41] A. V. Zelenkov, “Calculation of the Parameters of Hidden Markov Models Used in the Navigation Systems of Surface Transportation forMapMatching: A review.,” *Automatic Control and Computer Sciences*, pp. 309–323, 2010.

[42] J. Song, G. Y. Li, N. N. Li, and Y. N. Zhang, “A Fuzzy-Logic-Based Map Matching Algorithm for the GPS/DR System,” *Comput. Eng. Sci.*, pp. 30–32, 2008.

[43] K. Li, Y. Yang, and X. Qiu, “An Improved Map Matching Algorithm Based on D-S Evidence Theory in City Vehicle,” *Acta Geodaetica et Cartographica Sinica.*, pp. 208–213, 2014.

[44] H. Xu, H. Liu, and C. Tan, “Development and Application of an Enhanced Kalman Filter and Global Positioning System Error-Correction Approach for Improved Map-Matching,” *J.Intell. Transport. S.*, pp. 27–36, 2010.

[45] H. Yin and O. Wolfson, "A Weight-Based Map Matching Method in Moving Objects Databases.," in *16th International Conference*, Santorini, Greece, Jun. 2004, pp. 437–438.

[46] Y. H. Wang, H. L. Guan, and Q. S. Zhang, "Incrementally Detecting Change Types of Spatial Area Object: A Hierarchical Matching Method Considering Change Process.," *ISPRS Int. J.Geo-Inf.*, pp. 1–14, 2018.

[47] M. Hashemi and H. Karimi, "A Weight-Based Map-Matching Algorithm for Vehicle Navigation in Complex Urban Networks.," *J. Intell. Transport. S.,* pp. 573–590, 2016.

[48] W. Fang and S. Huang, "Research and Implementation of GPS/MM Vehicle Navigation System," *MicrocomputerInformation*, pp. 217–225, 2007.

[49] Y. J. Lee, J. K. Suhr, and H. G. Jung, "Map Matching-Based Driving Lane Recognition for Low-Cost Precise Vehicle Positioning on Highways," *IEEE Access*, vol. 9, pp. 42192–42205, 2021, doi: 10.1109/ACCESS.2021.3065746.

[50] J. Min Kang, T. S. Yoon, E. Kim, and J. B. Park, "Lane-level map-matching method for vehicle localization using GPS and camera on a high-definition map," *Sensors (Switzerland)*, vol. 20, no. 8, Apr. 2020, doi: 10.3390/s20082166.

[51] X. Wang, Y. Qian, C. Wang, and M. Yang, "Map-Enhanced Ego-Lane Detection in the Missing Feature Scenarios," *IEEE Access*, vol. 8, pp. 107958–107968, 2020, doi: 10.1109/ACCESS.2020.3000777.

[52] B. Flade, M. Nieto, G. Velez, and J. Eggert, "Lane Detection Based Camera to Map Alignment Using Open-Source Map Data," in *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, Institute of Electrical and Electronics Engineers Inc., Dec. 2018, pp. 890–897. doi: 10.1109/ITSC.2018.8569304.

[53] R. Sadli, M. Afkir, A. Hadid, A. Rivenq, and A. Taleb-Ahmed, "Map-Matching-Based Localization Using Camera and Low-Cost GPS for Lane-Level Accuracy," in *Procedia Computer Science*, Elsevier B.V., 2021, pp. 255–262. doi: 10.1016/j.procs.2021.12.237.

[54] Matt Forrest, "A Brief History of Web Maps," Towoards Data Science.

[55] MKuranowski, "pyroutelib3." Accessed: Dec. 20, 2023. [Online]. Available: https://github.com/MKuranowski/pyroutelib3/wiki

[56] W. Zeng and R. L. Church, “Finding shortest paths on real road networks: the case for A*,” 2007.

[57] Michael E. Mortenson, *Mathematics for Computer Graphics Applications*. 1999.

[58] “Bézier Curve in Python.” Accessed: Dec. 20, 2023. [Online]. Available: https://bezier.readthedocs.io/en/stable/python/reference/bezier.curve.html

[59] H. Prautzsch, W. Boehm, and M. Paluszny, “B-spline representation,” 2002, pp. 59–75. doi: 10.1007/978-3-662-04919-8_5.

[60] S. Briot and W. Khalil, “Homogeneous Transformation Matrix,” 2015, pp. 19–32. doi: 10.1007/978-3-319-19788-3_2.

[61] H. B. PACEJKA and I. J. M. BESSELINK, “Magic Formula Tyre Model with Transient Properties,” *Vehicle System Dynamics*, vol. 27, no. sup001, pp. 234–249, Jan. 1997, doi: 10.1080/00423119708969658.

[62] M. A. Johnson and M. H. Moradi, *PID control*. Springer, 2005.

[63] A. AbdElmoniem, A. Osama, M. Abdelaziz, and S. A. Maged, “A path-tracking algorithm using predictive Stanley lateral controller,” *Int J Adv Robot Syst*, vol. 17, no. 6, p. 1729881420974852, 2020.

[64] L. Joseph, *Robot Operating System for Absolute Beginners*. Berkeley, CA: Apress, 2018. doi: 10.1007/978-1-4842-3405-1.

[65] “ROS Distributions.” Accessed: Dec. 21, 2023. [Online]. Available: https://docs.ros.org/en/rolling/Releases.html

[66] M. Quigley *et al.*, “ROS: an open-source Robot Operating System.” [Online]. Available: http://stair.stanford.edu

[67] V. E. Kremer, “Quaternions and SLERP.”

[68] D. Tola and P. Corke, “Understanding URDF: A Dataset and Analysis,” Aug. 2023.

[69] © 2023 Robert Bosch GmbH, “Bosch Mobility.” Accessed: Dec. 06, 2023. [Online]. Available: https://www.bosch-mobility.com/en/

[70] GISGeography, “How GPS Receivers Work – Trilateration vs Triangulation.” Accessed: Dec. 06, 2023. [Online]. Available: https://gisgeography.com/trilateration-triangulation-gps/

[71] Gianni Rossi, “Concetti base sul sistema WGS84 del GPS.” Accessed: Dec. 06, 2023. [Online]. Available:

https://blog.topgeometri.it/2022/12/24/concetti-base-sul-sistema-wgs84-del-gps/

[72] Politecnico di Torino, “Navigation Systems ,” in *Communication and Network Systems*, 2022.

[73] Politecnico di Torino, “Lane Keeping Assist,” in *Driver Assistance Systems Design A*, 2022.

[74] “ROS2 Foxy Documentation.” Accessed: Jan. 02, 2024. [Online]. Available: https://docs.ros.org/en/foxy/index.html