FEDERAL UNIVERSITY OF SANTA CATARINA
CAMPUS JOINVILLE
POST GRADUATION PROGRAM OF ENGINEERING AND MECHANICAL SCIENCES

Marcus Vinícius Leal de Carvalho

A Review of ROS Based Autonomous Driving Platforms to Carry Out Automated Driving Functions

Joinville
2022

Marcus Vinícius Leal de Carvalho

A Review of ROS Based Autonomous Driving Platforms to Carry Out Automated Driving Functions

Dissertation submitted to the Post Graduation Program of Engineering and Mechanical Sciences of Federal University of Santa Catarina for obtaining the title of Master in Engineering and Mechanical Sciences.
Advisor: Prof.Roberto Simoni, Dr.

Joinville
2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Leal de Carvalho, Marcus Vinícius
A Review of ROS Based Autonomous Driving Platforms to Carry Out Automated Driving Functions / Marcus Vinícius Leal de Carvalho ; orientador, Roberto Simoni, 2022.
164 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Campus Joinville, Programa de Pós-Graduação em Engenharia e Ciências Mecânicas, Joinville, 2022.

Inclui referências.

1. Engenharia e Ciências Mecânicas. 2. Autonomous Vehicles. 3. Path Planning. 4. Advanced Driving Assistance Systems (ADAS). 5. Robotics' Simulators. I. Simoni, Roberto. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia e Ciências Mecânicas. III. Título.

Marcus Vinícius Leal de Carvalho

A Review of ROS Based Autonomous Driving Platforms to Carry Out Automated Driving Functions

The present Master Dissertation in Master level was evaluated and approved for the following board of examiners:

Prof.Andrea Piga Carboni, Dr.
Federal University of Santa Catarina

Prof.Lucas Weihmann, Dr.
Federal University of Santa Catarina

Prof.Ricardo Tellez Lara
U.P.C Technical University of Catalonia
University of LaSalle from Barcelona, Spain

We certify that this is **the original and final version** of the concluding work, which was judged appropriate to acquire the title of Master of Engineering and Mechanical Sciences.

Documento assinado digitalmente
Diego Alexandre Duarte
Data: 03/02/2022 11:06:26-0300
CPF: 047.147.229-80
Verifique as assinaturas em https://v.ufsc.br

Prof. Diego Alexandre Duarte, Dr

Program Coordination

Documento assinado digitalmente
Roberto Simoni
Data: 03/02/2022 11:02:52-0300
CPF: 033.398.969-41
Verifique as assinaturas em https://v.ufsc.br

Prof. Roberto Simoni, Dr

Advisor

Joinville, 2022.

" Just the well well done stands the test of the time, only the creative reaches into the future."

(Pecon)

RESUMO EXPANDIDO

Introdução

No mundo moderno, o fluxo de pessoas e de bens nos meios de transporte disponíveis é excepcionalmente elevado. Se por um lado as redes de transporte e veículos sofrem cada vez mais melhorias, promovendo e facilitando a locomoção das pessoas, por outro lado aumenta-se o número de acidentes. Acidentes em rodovias são a quinta maior causa de motalidade no mundo. Estatisticamente, este número corresponde a mais de um milhão e duzentos mil óbitos e mais de 20 milhões de pessoas lesionadas por ano, sendo que mais de 94% destes acidentes são consequências de erro humano. Na tentativa de reduzir, quando não possível eliminar, as consequências das colisões entre veículos, a indústria automobilística vem criando à décadas sistemas críticos de auxilío ao motorista: sistemas de segurança passivo e sistema de segurança ativo. Posteriormente, com a evolução da eletrônica e computação, este setor passou a desenvolver e disponibilizar os Sistemas Avançados de Assistência à Direção (ADAS). Ainda que estas tecnologias, tenham reduzido o número de acidentes ou ao menos atenuado o impacto, a indústria e pesquisadores perceberam que quanto maior o nível de automação veicular, reduz-se sensivelmente a quantidade de acidentes. A partir desta análise, nas últimas décadas os fabricantes de automóveis têm direcionado esforços para alcançar o última fase de automação de direção: o veículo totalmente autônomo (nível 5). Para se chegar neste nível de automação, onde até mesmo a presença do motorista no veículo não se faz necessária, a estrutura computacional, eletrônica, inteligência artificial e robótica presentes neste veículo devem ser excepcionalmente robustas, eficientes e com elevado grau de confiabilidade. Para atingir essa melhor performance, o software que comanda este veículo-robô é fracionado em diversos módulos: mapeamento, localização,percepção, detecção, planejamento de trajetória, tomada de decisão e controle. Este trabalho fornece um estudo aplicado e comparativo das mais renomadas Pilhas de Navegação Autônomas (ADS) de código aberto (Open-Source) disponíveis em âmbito mundial.

Objetivos

O objetivo principal destre trabalho é realizar simulações em cenários relevantes no contexto de veículos autônomos, para obter-se benchmark das diferentes Pilhas de Navegação utilizadas, bem como dos módulos que as compõem, com enfoque na camada de planejamento de trajetória. Para atingir este objetivo, as metas especifícas são: investigar e simular aplicações baseadas na framework ROS1 e ROS2 e selecionar a mais adequada; definir cenários relevantes para simular, a partir da literatura; implementar o código dos cenários e do comportamento dos agentes participantes desta simulação (pedestres e veículos); aplicar os algoritmos mais eficazes disponíveis nas Pilhas de Navegação baseads

em ROS para cumprir com as tarefas exigidas à um veículo autônomo; configurar a comunicação das camadas de comunicação do ROS com os simuladores a serem utilizados; extrair aos canais com fluxo de dados relevantes, subscrevendo-os e plotando os dados obtidos; avaliar os dados e reportá-los.

Metodologia

As Pilhas de Navegação são compostas por diversos módulos, bibliotecas e extensos algoritmos que requerem alto poder computacional para serem executados de maneira eficaz. Para habilitar estes módulos com o simulador foi selecionado uma máquina virtual, locada na empresa Paperspace, com as seguintes configurações: Unidade de Processamento Aritmético (CPU) de 16 núcleos, Unidade de processamento gŕafico Quado P-6000, Memória de Acesso Rápido (RAM) de 60 Gigabytes e Sistema Operacional Ubuntu 20.04. Os simuladores utilizados para interfacear com as Pilhas de Navegação foram o Gazebo e o SVL da empresa LG Eletronics. O primeiro foi utilizado para comunicação com a estrutura de navegação baseada em ROS1: Navigation Stack. O segundo foi utilizado para a comunicação com as Pilhas de Navegação: Autoware.AI (ROS1), Autoware.Auto (ROS2) e Apollo.Auto6.0 (Cyber-Rt). Para as simulações realizadas com o Navigation Stack, o ambiente simulado foi modelado, via código (.world file), visando representar o complexo e laboratório CARISSMA, presente na universidade THI. O modelo do carro foi baseado no PRIUS, cujo código de Format Unificado de Descrição do Robô (URDF) deste veículo, encontra-se disponível em meu github. Para as simulações envolvendo o simulador da LG (SVL), foram utilizados os veículos disponibilizados e configurados para cada Pilha de Navegação: Jaguar (Autoware.AI), Lexus (Autoware.Auto), Lincoln MKZ (Apollo.Auto). Os planejadores de trajetória utilizados, assim como algoritmos especifícos de cada módulo das Pilhas de Navegação, foram os disponibilizados por estes softwares. Todavia, as alterações pontuais para funcionamento adequado destes algoritmos com o simulador, podem ser encontradas tambe no meu repositório do gitbub: códigos_do_mestrado. Quanto à determinação dos estudos de caso para serem simulados, diferentes metodologias foram empregadas para definição destes: O método ontológico apresenta agentes interagindo com o veículo autônomo em ambiente virtual de teste combinatorial, com subsequentes análises de cenários critícos apontados como saída de técnicas de machine learning; o estudo de avaliação do modo de dirigir do veículo autônomo em conjunto com o Fator de Indicação de Performance (KPI) para definição da análise de segurança da condução autônoma e respectiva determinação de cenários critícos do sistema sob teste (SUT); o método fornecido pelo consórcio das indústrias automotivas alemãs com o governo, nomeado de "Projeto para Estabelecimento Geral de critérios, ferramentas e métodos de qualidade Aceitos, assim como Cenários e Situações" (PEGASUS). A partir destes métodos em conjunto com critérios que definem cenários críticos, como um reduzido Tempo para Colisão (TCC), foram determinados e posteriormente programados no simulador,

os cenários estático e dinâmicos simulados nesta dissertação. Para programação destes cenários e posterior comunicação com os módulos das Pilhas de Navegação para troca de dados, foi utilizado a aplicação PythonAPI do simulador da LG. Desta forma, foi possível receber dados provindos dos sensores veiculares presentes nos carros simulados, tais como LiDar (velodyne) e câmera, processamento interno nas Pilhas de Navegação (Localização, Deteccção, Planejamento e Tomada de Decisão), e por fim, a saída dos dados gerados pelos planejadores de trajetória chegaram em uma camada de conversão para controle veicular dos atuadores do veículo emulado pelo simulador, representando de maneira fiel o comportamento de um veículo autônomo real.

Resultados e Discussão

Utilizando o software Navigation Stack, baseado em ROS1, foi verificado que a maior parte dos planejadores de trajetória disponíveis, com exceção do TEB, são incompatíveis com as restrições não-holônomicas de um veículo-robô, que seguem as restrições da geometria do modelo de direção de Ackermann. Desta forma, as simulações realizadas com este software não foram satisfatórias para evitamento de colisão e movimentos suaves, que correspondem com um motorista humano. Isso se explica pelo fato deste software ser amplamente aplicado para outros tipos de robôs: holônomicos e não-holonômicos diferenciais. Porém, a partir da utilização das Pilhas de Navegação este problema é suprimido, uma vez que os módulos destas são particularmente definidos para uso de robôs do tipo carro. Utilizando o software Autoware e Apollo.Auto foi possível testar 5 diferentes planejadores de trajetória: Parking Planner (Autoware.Auto) , Freespace Astar (Autoware.AI), Freespace Lattice (Autoware.AI), Open Planner (Autoware.AI) e Public Road (Apollo.Auto). O primeiro obteve performance satisfatória, sendo possível realizar manobras de estacionamento em cenário especifíco. Porém, nota-se que ainda são necessários ajustes no módulo "Behavior Planner", uma vez que a transição entre o Lane Planner e o Parking Planner não está bem definida, gerando trepidações e súbitas mudanças bruscas na direção. Por fim, para o cenário onde o veículo autônomo se depara com um obstáculo, foram aplicados os demais quatro planejadores trajetórias descritos anteriormente. Eles foram avaliados em dez diferentes métricas. A performance dos planners: Open Planner e Public Road foram de maneira geral melhores que as providas pelo planejador de trajetória Global (Freespace) e seus respectivos planejadores de trajetória locais (Astar e Lattice), uma vez que apresentaram um comportamento mais similar ao de um motorista que viesse a se deparar com a mesma situação.

Conclusão

O presente trabalho propôs a utilização das Pilhas de Navegação de Código Aberto

para geração de trajetórias para o robô do tipo carro em diferentes cenários. Uma análise gráfica e comparativa dos diferentes planejadores de trajetória foi realizada, e por fim um Benchmark do módulos que compõem estas estruturas. Verifica-se destas análises que importantes módulos da framework Autoware, como o módulo de localização e o de mapeamento, apresentam desempenho insatisfatório em cenários de maior extensão. Em contrapartida o módulo de Percepção da framework Apollo.Auto6.0 se mostra falho para correta detecção de obstáculos, forçando o uso de uma percepção modular virtual, que publica os dados dos obstáculos presentes na simulação diretamente, através da ponte que conecta o simulador a esta framework. Ainda assim, de maneira geral, a Pilha de Navegação Apollo.Auto6.0 se mostrou mais robusta, apresentando resultados mais constantes durante a execução das simulações, sendo mais precisa na geração e cumprimento das trajetórias. A framework Autoware.AI se mostra mais modular com possibilidade de utilização e intercambiação de diversos pacotes de algoritmos, no caso do controle, por exemplo, pode-se escolher entre MPC e Pure-Pursuit, enquanto a framework Apollo oferece apenas o último. Em relação aos planejadores de trajetória, o Open Planner foi responsável por gerar a trajetória mais segura nos aspectos de distância lateral e distância miníma deixada do obstáculo durante a ultrapassagem, em contrapartida, essa maior margem de segurança fez com ele gerasse o caminho mais extenso para o veículo percorrer. No aspecto de conforto, o planejador de trajetória Freespace Lattice obteve a pior performance, uma vez que gerou uma trajetória pouco suave e apresentou elevadas oscilações no perfil de aceleração ao longo da execução da trajetória. Em termos de esforço computacional, conclui-se que o planejador de trajetória Public Road e os módulos da framework Apollo.Auto obtiveram os maiores consumos de memória e de CPU, significando uma maior exigência em termos de Hardware e sistemas embarcados para utilização em um véiculo real e também para ambientes de simulação. Verificando a performance de cada planejador de trajetória nota-se que não há um planejador de trajetória universal, otimizado para todos cenários e com melhor performance em todos parâmetros de avaliação, porém pode-se otimizar os parâmetros de cada planner, ou ainda selecionar diferentes planejadores de trajetória para diferentes situações para quando se deseja obter melhores performances em aspectos especifícos.

Palavras-Chave: Veículo Autônomo, Cenários, Simulador, Pilhas de Navegação, ROS, Planejadores de Trajetória, Módulos.

Abstract

In the last decades, advanced driving assistance systems (ADAS) have avoided or mitigated high road accidents. Lane-keeping, lane departure warning, blind-spot detection, automatic emergency braking, and important technological innovations are being developed and optimized each year by the automotive industry. As in other sectors, automation, electronics, and robotics technology continuously modify automobile production. The ultimate step from this process is autonomous driving. Conducting a car-like robot from point A to point B, using the shortest path, avoiding collisions comfortably and safely, is the final goal of this system. This work delineates the autonomous driving process, from mapping to control, focusing on the planning layer. Available open-source frameworks, called autonomous driving stacks (ADS), containing the mentioned modules are employed. Different sorts of mission and motion planning algorithms available on these frameworks are tested to address the path planning topic. The planning and control problem is solved for different scenarios, globally and locally. The studied test cases are performed with a real environment emulator: the LGSVL simulator. In the end, an analysis of the autonomous vehicle (AV) generated trajectories, its performance, and benchmark from Autoware.AI and Apollo.Auto 6.0 ADS modules are performed.

Keywords: ADAS, AV, Obstacle Avoidance, LGSVL, Path Planning, Safety, Scenarios.

Acknowledgements

This work would not have been possible without the partnership between the Federal University of Santa Catarina (UFSC) from Brazil and the Technische Hochschule Ingolstadt (THI) from Germany instigated by the AWARE network. Therefore, I am very grateful for these government institutions, which spread knowledge and technology worldwide.

At first, a capital "THANK YOU" must be given to my adviser, Dr.Prof Roberto Simoni, who introduced me to this exciting robotics world and guided my way to success. Under his supervision, I have learned multiple computational and research skills, which enriched my knowledge exorbitantly. He put me on another level and expanded my academic and professional horizons. I am very grateful for his unconditional support and trust in all decisions and steps I have chosen during this journey.

Additionally, I would like to thank Dr.Prof Hüber from THI, who provided me an opportunity to extend this research in Germany's best autonomous driving lab: CARISSMA. His classes in ADAS subjects and the practical classes formed the spark that triggered the beginning of this research. Watching his effort and passionate work on this topic have motivated me to overcome this survey challenge.

My deepest gratitude to my lab partner and friend Bruno Mamoru Kato, who often helped me with bug fixing and simulations optimization.

I am also extremely grateful to my supervisor at Audi, Moritz Würth, who opened me an internship opportunity at AUDI in the Radar & Laser-Scan department. This fantastic experience in this automotive industry was essential to deepen and better master the ADAS subject. I have learned a lot from him since programming better to become a better professional.

Above all, I want to give special thanks to my father, Wanderley de Carvalho. He provided me financial and emotional support during this challenging period of the covid-19 pandemic, which part of it I found me alone in another country.

I thank the groups: LG, The Construct, AWF, Apollo, and the whole ROS wiki community for their technical assistance during the development of this work. Thanks to dear people who participated in this work directly or indirectly: Florian Denk (CARISSMA), Eric Boise (LG), Duong Le, Ricardo Tellez, Arif Rahman.

I thank the institutions FAPESC and BAYLAT for the respective scholarships in Brazil and Germany during this research period.

Finally, I let warm gratitude to my family and friends, as without their continuous belief in me, I would not make it to the end: Vera, Heloisa, Johny, and Rafael (family). Stefany Carvalho, Dimitris Kisimov, Hitesh Deriya, and Jesuslene Gomes (friends).

List of Figures

List of Tables

List of abbreviations and acronyms

ABS Anti-Lock Brakes

ACC Adaptive Cruise Control

ACTor Autonomous Campus Transportation

ADAS Advanced Driving Assistance Systems

AD Autonomous Driving

ADS Autonomous Driving Stacks

AI Artificial Intelligence

ALV Autonomous Land Vehicle

AMCL Adaptive Monte Carlo Localization

AMR Autonomous Mobile Robot

AN Autonomous Navigation

AUV Autonomous Undewater Vehicles

AV Autonomous Vehicles

AVP Autonomous Valet Parking

AWF Autoware Foundation

AWS Amazon Web Services

BMWi German Federal Ministry for Economic Affairs and Energy

CAN bus Controller Area Network bus

CARISSMA Center of Automotive Research on Integrated Safety Systems and Measurement Area

CASs Collision Avoidance Systems

CCW CounterClock Wise

CG Conjugate Gradient

CHOMP Covariant Hamiltonian Optimization for Motion Planning

CNN Convolutional Neural Network

CPU Central Processing Unit

CR-RRT Closed Loop RRT

CVM Curvature Velocities Techniques

CW Clock Wise

DARPA Defense Advanced Research Projects Agency

DART Dynamic Animation and Robotics Toolkit

DDS Data Distribution Service

DGPS Differential GPS

DP Dynamic Programming

DSC Dynamic Stability Control

DWA Dynamic Window Approach

EEM External Errors

EKF Extentend Kalman Filter

ESP Electronic Stability Program

EST Expansive-Spaces Tree

FCL Flexible-Collision-Library

FZD Technische Universität Darmstadt

GA Genetic Algorithm

Greedy BFS Greedy Best First Search

GNSS Global Navigation Satellite System

GPU Graphic Processing Unit

GPS Global Positioning System

HAD Highly Automated Driving

HD-Map High Definition Map

HDR High Dynamic Range

HEIs	Human Error Detection Methods
HET	Human Error Template
HDRP	High Definition Render Pipeline
HIL	Hardware in the Loop
Hp	Horsepower
ICR	Instantaneous Center of Rotation
IMU	Inertial Measurement Unit
IRA	Independent Reasoning Agent
JSON	JavaScript Object Notation
KPIECE	Kinodynamic Planning by Interior-Exterior Cell Exploration
KPI	Key Performance Indicator
LaSiN	Naval Simulation Laboratory
LiDAR	Light Detection and Ranging
LRR	Large Radar Range
MPC	Model Predictive Control
MRR	Medium Radar Range
NAV2	Navigation Stack 2
NDT	Normal Distribution Transform
ODD	Operation Design Domain
ODE	Open Dynamics Engine
ODO	Overall Direct Optimization
OGRE	Object Oriented Graphics Rendering Engine
OMPL	Open Motion Planning Library
Op	Open Planner
OpenCV	Open source Computer Vision and Machine Learning Software Library

OS Operating System

OTA Over-the-air

PCD Point Cloud Data

PEGASUS Project for the Establishment of Generally Accepted

PELOPS Program for Development of Longitudinal Traffic Processes

PID Proportional Integral Derivative

PQP Proximity Query Package

PRM Probabilistic Roadmap

QP Quadratic Programming

RaDAR Radar Detecting and Ranging

RAM Random Access Memory

RL Rear Left

ROS Robot Operating System

ROS2 Robot Operating System 2

RQT ROS QT

RRT Rapidly-exploring Random Tree

RT Random Trees

Rviz ROS Visualization

SAE Society of Automotive Engineers

SBL Single-query Bi-Directional Probabilistic Roadmap Planner

SBPL Search-based Planning Library

SDF Simulation Description Format

SDR Standard Dynamic Range

SIL Software in the Loop

SLAM Simultaneous Localization and Mapping

SRR Short Radar Range

SSRRT*	Spline based RRT
S**TOMP**	Stochastic Trajectory Optimization for Motion Planning
SUT	System Under Test
SyCLOP	Synergistic Combination of Layers of Planning
TEB	Time Elastic Band
TCC	Time To Collision
TTE	Time To Enter
THI	Technische Hochschule Ingolstadt
UFES	Federal University of Espirito Santo
UKF	Unscented Kalman Filter
UGV	Unmanned Ground Vehicle
URDF	Unified Robot Description Format
VFF	Virtual Force Field
VFH	Vector Field Histogram
V2N	Vehicle-to-Network
V2P	Vehicle-to-Pedestrian
V2V	Vehicle-to-Vehicle
V2VCR	Vehicle-Vehicle Collision Probability Relation
V2X	Vehicle-to-Everything
V21	Vehicle-to-Infrastructure
YOLO	You Only Look Once

List of symbols

5G	Fifth Generation Technology Standard for Broadband Cellular Networks
$\mathbf{acc_{lin}}$	Linear acceleration
$\mathbf{acc_{net}}$	Net acceleration
$\mathbf{a_{tan}}$	Tangential acceleration
$\mathbf{a_{cent}}$	Centripetal acceleration
$\mathbf{a_{res}}$	Resultant acceleration
$\boldsymbol{\alpha_1}$	coefficient factor to increase or decrease the H1 term weight
$\boldsymbol{\alpha_2}$	coefficient factor to increase or decrease the H2 term weight
$\boldsymbol{\beta_{p_i}}$	Bubble Configuration
$\boldsymbol{\Delta_{t_i}}$	Instantaneous time
$\mathbf{d_{(i)}}$	Instantaneous displacement
$\mathbf{d_x}$	Robot's longitudinal control space
$\mathbf{d_y}$	Robot's lateral control space
$\mathbf{d_\theta}$	Robot's heading control space
f_cost	Cost to travel from the origin cell to the neighbor cell, in the path planning context
$\mathbf{f_{(i)}}$	Instantaneous force
$\mathbf{F_{(i)}}$	Heuristic Function
$\mathbf{G_{(i)}}$	Cost to reach the final waypoint
h_cost	Represents an heuristic distance from a node n to the goal location
$\mathbf{H1_{(i)}}$	Distance between the point i and the guideline
$\mathbf{H2_{(i)}}$	Distance from g(i) to the target
$\mathbf{Q_o}$	Start Point

$\mathbf{Q_{free}}$	Indicates a grid cell with free space
$\mathbf{Q_{obs}}$	Indicates a grid cell occupied by obstacle
$\mathbf{q_{init}}$	Root node of the tree (origin)
$\mathbf{q_{goal}}$	Final node of the tree
$\mathbf{K_s}$	Path Curvature of length s
$\mathbf{N_s}$	Start Point
$\boldsymbol{\eta_n}$	Final vertex of lattice path
$\mathbf{p_{i_i}}$	Free-space robot configurations
$\boldsymbol{\mathcal{P}}$	Power Consumption (Hp)
$\boldsymbol{\psi}$	Front's wheel steering angle
$\mathbf{s_i}$	Vehicle Pose
$\mathbf{v}$	Velocity
$\boldsymbol{\delta}$	Steering Angle
$\boldsymbol{\phi}$	Steering Control
$\boldsymbol{\mathcal{S}}$	Smoothness of Trajectory
$\boldsymbol{\tau}$	Time Interval
$\boldsymbol{\mathcal{T}_t}$	Jerk (acceleration derivative)
$\dot{\mathbf{x}}$	Longitudinal Velocity
$\dot{\mathbf{y}}$	Lateral Velocity
$\boldsymbol{\omega}$	Angular acceleration

Contents

1 Introduction

Each hour of the day worldwide, millions of people plan to go from point A to B. Whether with their vehicles or using public transportation. Unfortunately, not all of these displacements are completed. Road traffic accidents are the fifth leading cause of morbidity and mortality worldwide (Baiee et al., 2020). Which numerically means 1.2 million deaths and more than 20 million injured people every year. Up to 94 % of these road traffic crashes are consequences of human behavior (Najafi and Arghami, 2020).

Automobile manufacturers have created critical systems and tests to counterattack the number of victims on highways and in urban driving areas. They took the first actions over passive safety systems with the creation of the crash test (the 1930s), seat belt(1950s), padded instrument panels(1960s,) and airbag (1970s). The above systems effectively reduce bodily injuries during the crash event but do not avoid collisions. To act on accident prevention, the automotive industry developed the first active safety system in the 1980s: the Anti-Lock Brakes (ABS), Electronic Stability Program (ESP), Dynamic Stability Control (DSC). To increase passenger comfort and security in the late 1990s, cruise control and, later the Active Cruise Control were also developed and classified as driver assistance systems. In this century, the automotive industry focuses on Advanced Driving Assistance Systems (ADAS), which ultimate phase is autonomous driving (Aeberhard, 2017).

According to Battiston (2015), an autonomous vehicle is defined as a passenger vehicle that drives by itself. This vehicle must perceive its surrounding environment and decide which route to take to reach its destination and drive. These car-like robot decisions are taken on planner modules to create collision-free waypoints. These waypoints will prevent the car from colliding with static or dynamic obstacles and enable the vehicle to reach the final target (Marin-Plaza et al., 2018).

An autonomous vehicle relies upon numerous sensors, decision-make algorithms, and is data-dependent. Therefore, the lack of reliable information, and an eventual gap in getting data, and a wrong algorithm computation will probably lead the vehicle to undesired behavior, which can directly impact passengers' lives. Legalizing these vehicles requires reliable software and hardware. The automotive industry has split autonomous driving into functions that compose modules divided into Perception, Localization, Planning, and Control to ensure robust software. The automotive sector hires numerous AI software companies to develop these specific functions to improve the modules. The frameworks made up by these modules are called Autonomous Driving Stacks (ADS).

This dissertation provides an outline of the open-source ADS. It gives an overview of the current software applied by the automotive industry and their respective layers and features. It takes focus on the trajectory planners domain. It covers how they integrate

the driving functions and performance in different autonomous driving scenarios.

1.1 Motivation

Severe injuries and deaths are expected consequences of vehicles accident. More than 90% of worldwide road collisions are caused by human error (Masuri et al., 2012; Yurtsever et al., 2020). According to the driving tasks, these errors emerge and vary in intensity and form. Najafi and Arghami (2020) used the Human Error Template (HET) to classify external errors (EEMs) in the form of human error detection methods (HEIs). The technique figured out 12 errors patterns done by the drivers when they drive in the following scenario:

> "the driver departs from the park and moves in a pre-determined direction. A few moments later, he speeds up to overtake the front car. Then, he continues driving. By getting the destination, he exits the path and parks the car". The survey concluded that the majority of the drivers commit the mistake called "Fail to Execute" while executing the task called "Scrolling the route," equalling 48% of all errors. Technically the error was distributed in the following ratio: 49%, 33% 18% for "acceleration changing," "line changing," and "distance adjusting" respectively.

Similar study (Hale et al., 1990) indicates a close relationship between human errors and the type of scenario the driving is submitted. Therefore, the specific scenario is essential for ADAS developers' teams. To mitigate these errors and assist the drivers while they are traversing these scenarios, the automotive industries brought through ADAS essential driving functions, such as:

- **Adaptive Cruise Control (ACC)**: The ACC automatically adjusts the vehicle speed to maintain a safe distance from vehicles ahead ("**A**" in Figure 1);
- **Emergency Brake Assist**: It assists the vehicle achieve a maximum braking effectiveness in an emergency situation.It's controlled automatically via the Electronic Stability Control system, which senses how much pressure the driver has applied to the brake himself and automatically increases it, if necessary ("**B**" in Figure 1);
- **Rear Cross Traffic Alert**: Informs to the driver if another vehicle is approaching from either direction when the vehicle ins in reverse and is backing out of a parking place ("**C**" in Figure 1);
- **Lateral Control**: The lateral control is responsible for regulating the vehicle's lateral control, steering the vehicle's wheels for path tracking, maintaining it in center of lane ("**D**" in Figure 1);

- **Blind Spot Monitoring (BSM)**: It assists the driver with noise warnings, while the vehicle is switching lane, if the BSM detects that a car is in the vehicle blind spot ("**E**" in Figure 1);
- **Intelligent Headlamp Control**: This system uses a video camera to measure the ambient brightness and to estimate the distance from vehicles in front and oncoming traffic (Bosch, 2021). This data is used to implement a variety of light functions ("**F**" in Figure 1);
- **Traffic Sign Assist**: Detects traffic signs with multi-function camera and assists the driver by displaying detected speed limits and overtaking restrictions in the instrument cluster (Ford, 2021), ("**G**" in Figure 1);
- **Active Lane Departure Warning**: The system emits audible warnings to the drivers when they unintentionally leave their lanes ("**D**" in Figure 1);
- **Surround View**: Fused data (from radar, camera and lidar) to create a model of the surrounding objects around the vehicle. It ensures better obstacle detection. The same term can also be recognized as the camera system which provides the driver several views like top, view, rear and panorama view to assist the driver while parking ("**H**" in Figure 1);
- **Longitudinal Control**: Control the vehicle's longitudinal motion speed ("**A**" in Figure 1);
- **Parking Guidance System (PGS)**: This system helps the drivers aprk with greater precision, using guidance system technology that rivals ultrasonic and other camera-based solutions with superior, advanced technology (CARandDriver, 2021);
- **Intersection Assistant**: This system employs the camera and radar sensor technology to detect oncoming traffic while attempting to turn left. If there's a potential collision with an oncoming vehicle, the vehicle can alert the driver and apply the brakes;
- **Tire-Pressure Monitoring (TPMS)**: Inform the drivers which tire has low air pressure via the TPMS readout in the instrument panel. Individual sensors in each tire monitor pressure and transmit the data to a receiver in the system. The pressure reading for each tire is displayed in the instrument panel (Honda, 2021);
- others.

The Figure 1 shows some of the main ADAS functions:

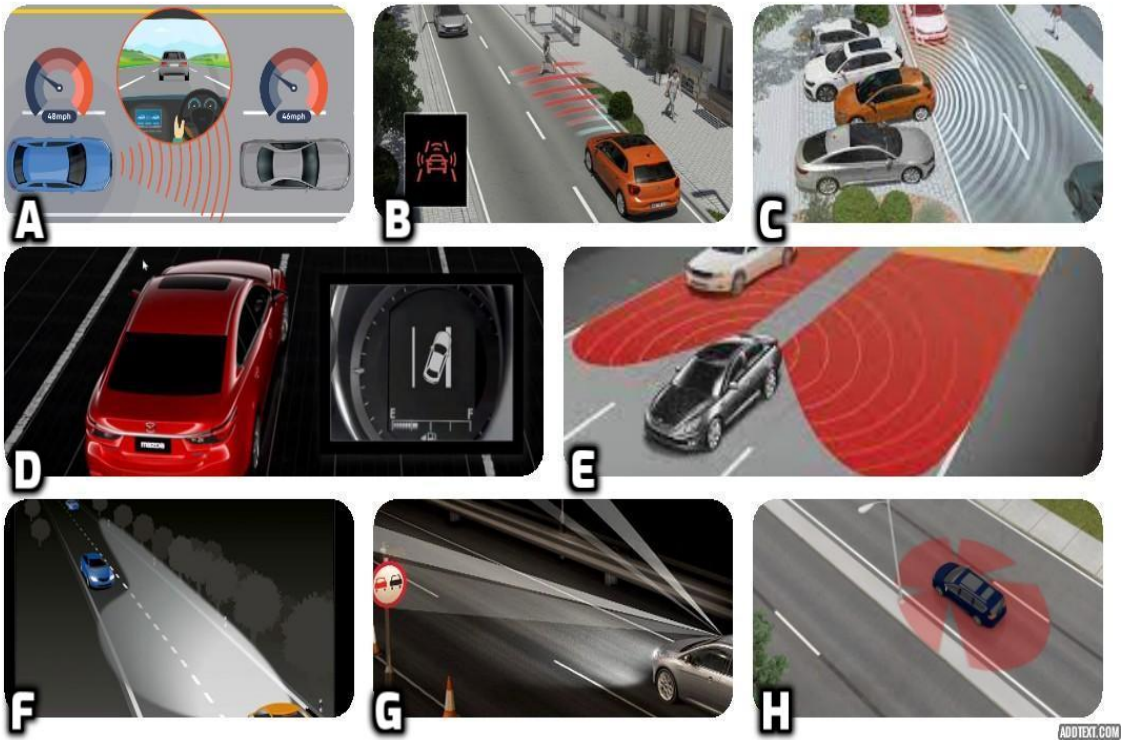


Figure 1 – ADAS Functions Source: Hella (2021); Ford (2021); Volkswagen-Newsroom (2018)

The automotive industry is continuously improving these technologies in partnership with companies supplying sensors, software, and related technologies. ADAS has demonstrated to be a good driver's ally, taking into account it assists the vehicle operator to drive safer, with more comfort and constantly acts to prevent collisions, issuing warnings or directly taking actuators control (steering, braking, or throttling). McKinsey and Company (2019) pointed out that heavy investment is being made in developed countries. in average, $2.3 Billion were applied in the last five years to AD-related software development. The Figure 2 depicts the previous decade's trends.

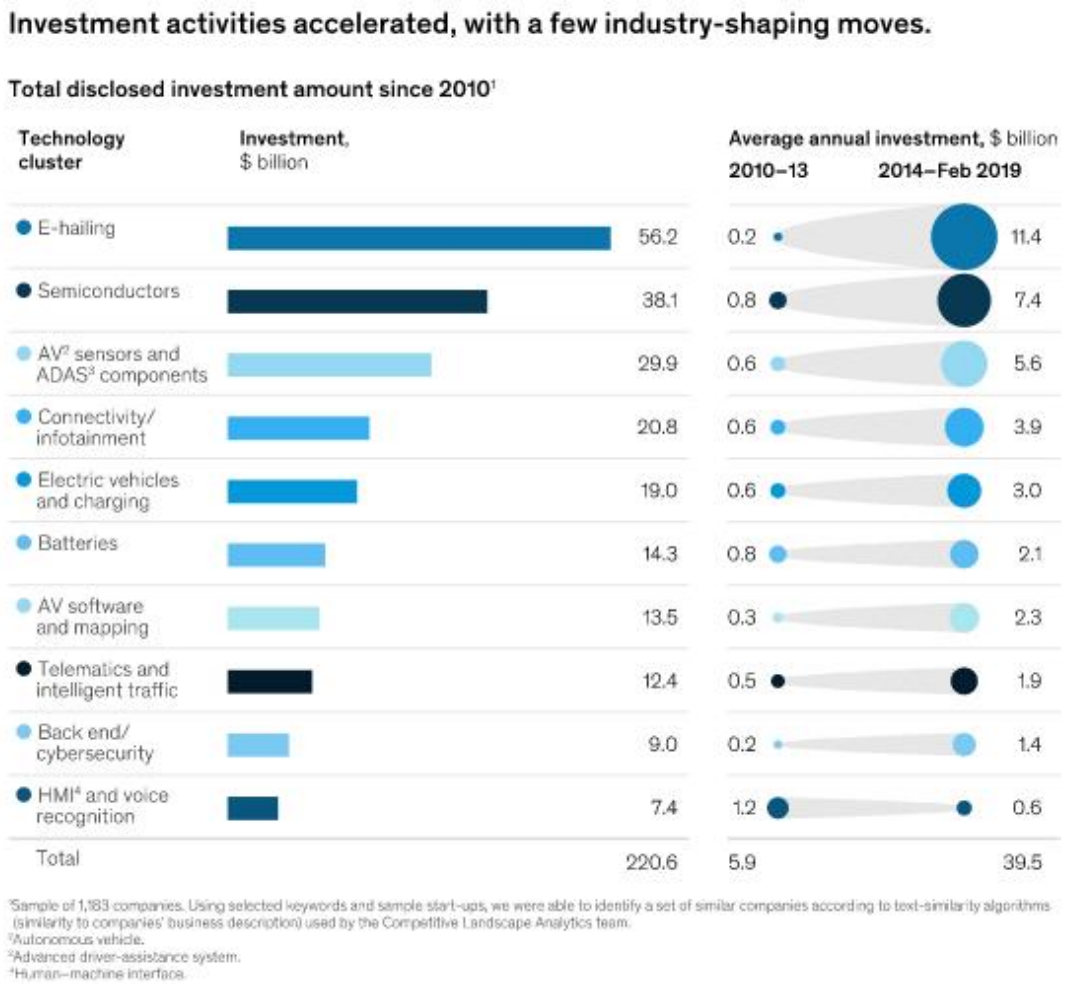


Figure 2 – Technology Investment Trend. Source: McKinsey and Company (2019)

It is possible to check that AVs and ADAS have attracted more and more investments in the last years. In 2020 Tesla gained authorization to drive on public roads in specific cities in the USA, Apollo in particular places in China, and recently, the Robo-taxi on specific conditions in Germany was allowed (Juliussen, 2020). Vehicle connectivity is also a trend in the autonomous driving context. It is divided into:

- The network-based communication: allows the cars to use the cellular network to communicate with nearby vehicles, pedestrians, and the infrastructure around them. This technology is known as vehicle-to-network (V2N) communication. the V2N has a more extensive range of communication compared with direct methods (McKinsey, 2019)

- Direct Communication: Enables the vehicles to interact directly with nearby surroundings, including communication with other vehicles (V2V). Contact with the traffic light, for example, is also possible, preventing an AV from passing the red light.

The Figure 3 depicts the sorts of network and direct communications. These technologies are currently being tested and improved each year due to improvements in electronics and communications technology, such as 5G.

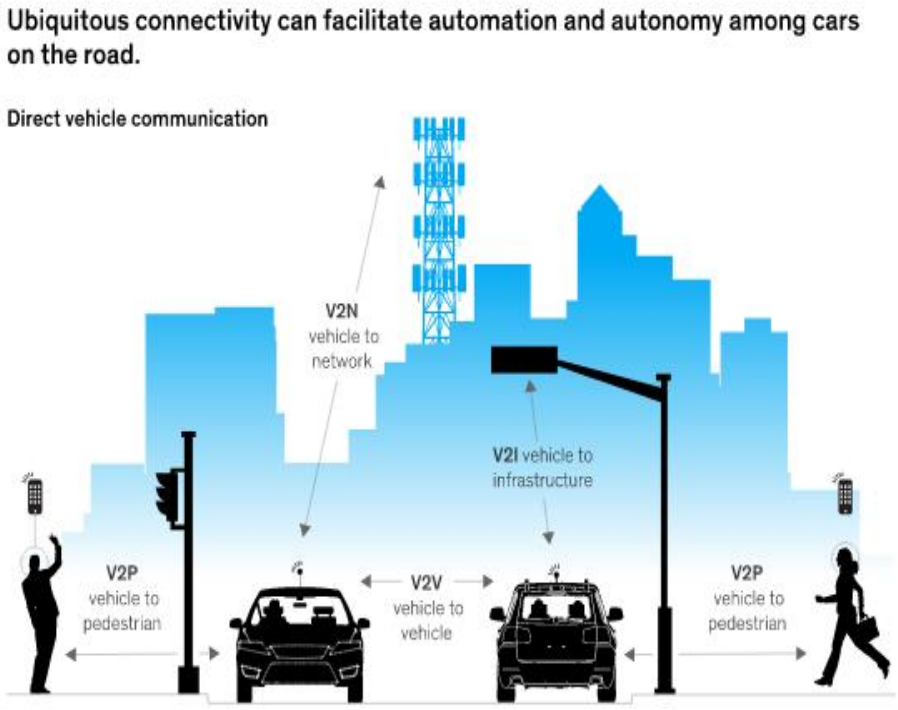


Figure 3 – Network and Direct Communication among cars on the road. Source: McKinsey (2019)

These kinds of communication depend directly on reliable and fast-speed data transmission. For this reason, the fifth-generation wireless technology, 5G, will be a vital enabler of this technology (McKinsey, 2019). The 5G is already being implemented in the most developed countries and started this year in Brazil (Feitosa, 2021). Regarding Brazil, the government already has exciting projects in autonomous vehicles. Badue et al. (2021) developed a complete architecture of a self-driving car at the Federal University of Espirito Santo (UFES), named intelligent Autonomous Robotics Automobile(IARA). This AV was the first Brazilian car to travel autonomously 74 km on urban roads and highways. The Figure 4 depicts this AV architecture.

Figure 4 – Intelligent and Autonomous Robotic Automobile (IARA), the first Brazilian self-driving car to travel autonomously 74 km on urban roads and highways. Source: Badue et al. (2021)

Autonomous driving is not fully developed yet. The automotive industry and research groups study and test different scenarios, also called the operation design domain (ODD). These groups are trying the AD functions in as many scenarios as possible. According to Kalra and Paddock (2016), the automated driving systems cannot keep being validated by test drives and conventional methods, such as driving thousands of kilometers on the high road. This problem, however, must be solved by a scenario-based approach (Pütz et al., 2017). That is the only way these functions will reach the maturity, robustness, and reliability needed to become legalized. **Covering different scenarios is the primary strategy to enable the AV to address adequately all situations that may occur in specific circumstances in a distinct scenario** (Ponn, Gnandt and Diermeyer, 2019; Schuldt et al., 2013; Klück et al., 2018; Nitsche et al., 2018; Rogic et al., 2016).

Autonomous Driving Systems (ADS) can bring positive impacts and benefits to society. They can: reduce the vehicle's miss behavior caused by human distraction, increase the mobility of elderly people, reallocate driving time and ride-sharing (Yurtsever et al., 2020). Battiston (2015) foresee the ADS will eliminate crashes, improve energy efficiency, increase safety, bring more comfort and reduce driving stress.

The potential noble social and economic contribution provided by autonomous driving, the optimistic perspective for this market, allied to the developed projects by this University's lab, in particular the ones based on ROS frameworks, inspire the survey on ROS based Autonomous Driving Stacks, the topic of study of this dissertation. Aligned to ADAS and ADS goals, it is intended on this work to investigate the performance of

the driving tasks executed by the planning module. To this end, relevant scenarios will be covered within the autonomous driving context. Assess the trajectory planners and decision-making taken by this module in different situations regularly present on traffic roads.

1.2 Contextualization of the work

This survey was partially developed in Naval Simulation Lab (LaSiN) and Technische Hochschule Ingolstadt (THI). Both laboratories cover several robotic's research topics and use ROS-based frameworks. The first idea was to test OMPL path planners for AUV vehicles (present in LaSin) using MoveIt ROS-based software. Similarly, test distinct path planners for AV vehicles in Navigation Stack ROS-based framework. The last goal was accomplished with success, and we preferred to focus more on the AV side than AUV for lack of time reasons. Therefore, more powerful ROS-based software was tested to reach the previously described features (see Section1.1) with more properties.

1.2.1 Main Objective

The main objective of this research is to perform autonomous driving simulations in different scenarios, applying distinct ADS and exploring its features, such as collision avoidance, addressing safety, and smooth maneuvers to the vehicle through the usage of standard trajectories planners.

1.2.1.1 Specific Aims

As described previously, to accomplish the primary goal of this work, the following aims need to be addressed:

- Investigate and test ROS1 and ROS2 based applications and choose the most suitable ones to AD context.
- Define relevant driving scenarios through the state-of-the-art literature review.
- Implement the scenario, agent's behavior, and position.
- Apply the most suitable algorithms available on ROS based ADS frameworks, to carry out the following tasks:
  - **Sense**: Environment Perception
  - **Plan**: Decision Making Planners
  - **Action**: Trajectory execution and motion control

- Set the ROS layers communication with the real-world scenarios represented by the simulator.
- Extract the relevant data-flow, subscribing to the topics and plot them.
- Evaluate this data and report the results.

1.3 Outline

This work is structured in the following chapters:

- **Chapter 2** covers the definition, challenges, benefits, providing a contextualization of autonomous vehicles.
- **Chapter 3** introduces the path planning concept, libraries, and first software available to test and investigate the path planners. Starting from the standard planners used for robot navigation to those developed for self-driving cars application.
- **Chapter 4** presents the Robot Operating System (ROS), the leading middleware used in this thesis to test different driving tasks scenarios. An overview of this framework and the earlier ROS-based software are presented: MoveIt and Navigation Stack. Afterward, some of the path planners introduced on **chapter 3** supported by Navigation Stack are integrated, and the performed simulations are displayed. Lately, an insight into ROS-based Autonomous Driving applications has been provided, which will further be detailed on **chapter 5**.
- **Chapter 5** covers the software applied to address the autonomous driving tasks, known as Autonomous Driving Stacks (ADs). It describes the ROS-based ADs with a significant focus on the planning module architecture since it is the principal-agent responsible for executing the driving tasks, the topic of interest of this work.
- **Chapter 6** presents the methodologies and simulations produced for the relevant scenarios discussed on **chapter 2** implemented over the frameworks described on **chapter 5**.
- **Chapter 7** present the simulation results by the use of extracted graphics from ROS and Cyber topics. Analysis and discussion are done in this chapter.
- **Chapter 8** Concludes the work confronting the initial goals defined on **chapter 1** with the results reported on **chapater 7**. The gaps are identified, and the next challenges are proposed.

2 Autonomous Vehicles

The driverless technology has long been enticing. It has the potential to transform users' experience of commuting, reduce the tiredness and stress of long journeys, take people out of high-risk working environments and streamline the industries.

2.1 Definition

An autonomous vehicle is defined as a passenger vehicle that drives by itself. A fully autonomous car does not require any operator to go it. These futuristic vehicles are usually driverless, driver-free, self-driving, autopiloted, or car-like robots. They can perceive its environment, decide which route must be taken to achieve the desired destination, and drive to it (Battiston, 2015).

2.2 History

Autonomous Vehicles have existed as prototypes and demonstrations since the 1960s (Battiston, 2015). The first automated vehicle was created by the electrical and computer engineer Robert Fenton and his team at Ohio State University in the mid-1960s. It is believed to be the first land vehicle to have used a computer. They placed a current-carrying wire laid down the center of the roadway and ran current through it, creating a magnetic field that enabled their control of the cars. The car was equipped with a large bump of electronics stuck out from the bumper to sense the fixed wire (Beasley, 2019). According to the inventor, the vehicle achieved higher speeds: "It worked beautifully. We were automatically steering at speeds up to 85 mph". A blueprint of the followed principle is sketched below:

In 1977 the first self-driving vehicle was built by Tsukuba mechanical engineering lab. The car reached speeds up to 30 km/h by tracking white lanes marked on the street for 50 meters.

In March of 2004, the Defense Advanced Research Projects Agency (DARPA) launched Darpa's Grand Challenge. This competition had as aim to develop autonomous vehicles able to navigate on desert trails and roads at high speeds. The first event had no winners since no competitor could complete the challenge. However 2005, 5 vehicles were able to finish the race of 244 km under 7 hours (Buehler et al., 2009). The encouragement of this event was a response to a congressional mandate that a third of US military ground vehicles should be unmanned by 2015 and as consensus that only research programs would not be able to carry out the necessary technology to meet this goal. Therefore DARPA

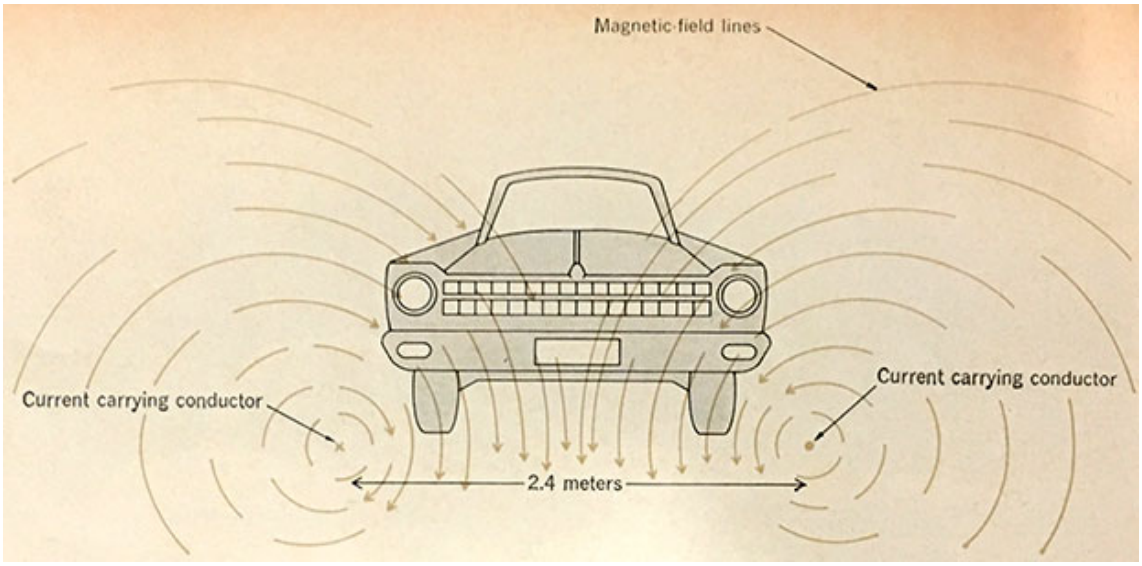


Figure 5 – An image from the original 1969 article illustrating the idea of a vehicle drawing current for propulsive power from the roadway. Source: Ackerman (2016)

was vital for the vertiginous advance of this technology.

Nowadays, companies such as Google, Nissan, Volvo, General Motors, Volkswagen/Audi, Tesla, Nissan, Toyota, BMW, and Mercedes-Benz have started research on this topic and keep investing in better implementations of AVs to fulfill the required requirements government obligations.

2.3 Automation Levels

The Society of Automotive Engineers (SAE) developed an industry-standard scale from zero to five to describe this continuum, although there are many gray areas where features might overlap. Here's what those levels generally means (Tettamanti et al., 2016):

- Level 0: No Automation. The driver is entirely responsible for controlling the vehicle, performing tasks like steering, braking, accelerating, or slowing down. Level 0 vehicles can have safety features such as backup cameras, blind spot warnings, and collision warnings. Even automatic emergency braking, which applies aggressive braking in an imminent collision, is classified as Level 0 because it does not act over a sustained period;
- Level 1: Driver Assistance. At this level, the automated systems start to take control of the vehicle in specific situations but do not fully take over. An example of Level 1 automation is adaptive cruise control, which controls acceleration and braking,

typically in highway driving. Depending on the functionality, drivers can take their feet off the pedals;

- Level 2: Partial Automation. At this level, the vehicle can perform more complex functions that pair steering (lateral control) with acceleration and braking (longitudinal control), thanks to a greater awareness of its surroundings;

- Level 2+: Advanced Partial Automation. While Level 2+ is not one of the officially recognized SAE levels, it represents an important category that delivers advanced performance at price consumers can afford. Level 2+ includes functions where the vehicle systems are essentially driving, but the driver must still monitor the vehicle and step in if needed. (By contrast, Level 3 represents a significant technology leap, as it is the first level at which drivers can disengage from the act of driving — often referred to as "mind off." At Level 3, the vehicle must safely stop in the event of a failure, requiring much more advanced software and hardware.) Examples of Level 2+ include highway assistance or traffic jam assistance. The ability for drivers to take their hands off the wheel and glance away from the road ahead for a few moments makes for a much more relaxing and enjoyable experience, so there is strong consumer interest;

- Level 3: Conditional Automation. At Level 3, drivers can fully disengage from the act of driving, but only in specific situations. Conditions could be limited to certain vehicle speeds, road types, and weather conditions. But because drivers can apply their focus to some other task — such as looking at a phone or newspaper — this is generally considered the initial entry point into autonomous driving. For example, features such as traffic jam pilot mean that drivers can sit back and relax while the system handles it all — acceleration, steering, and braking. The vehicle alerts the driver to regain control when the car gets through a traffic jam and the vehicle speed increases in stop-and-go traffic. The vehicle must also monitor the driver's state to ensure that the driver resumes control and come to a safe stop if the driver does not;

- Level 4: High Automation. At this level, the vehicle's autonomous driving system is fully capable of monitoring the driving environment and handling all driving functions for regular routes and conditions. However, depending on the vehicle's operational design domain (ODD), the system may, on rare occasions, need a driver to step in. In those cases, the car can alert the driver of an environmental condition that requires a human in control, such as heavy snow;

- Level 5: Full Automation. Level 5-capable vehicles are fully autonomous. No driver is required behind the wheel at all. Level 5 vehicles might not even have a steering wheel or gas/brake pedals. Level 5 vehicles could have "smart cabins" so that passengers

can issue voice commands to choose a destination or set cabin conditions such as temperature or choice of media.

Each level of automation requires additional layers of sensors, as the vehicles increasingly assume functions previously controlled by the driver. For example, a Level 1 vehicle might only have one Radar and one camera. A Level 5 vehicle, which must navigate any environment it encounters, will require full 360-degree sensing across multiple sensor types. For example, Aptiv's autonomous cars have included dozens of sensors and redundant, fail-operational architectures, such as Aptiv's Smart Vehicle Architecture™.

2.4 Sensors

The AVs can easier apply sensors with different work-principle to map their surrounding. The roboticists usually use more than one sensor to identify the obstacles and the AV's spot. Schröder et al. (2019) developed a network architecture based on Lidar and camera fusion, which increased traceability of the entire system, allowing to deal with real-world challenges such as sensor failures or blindness. Winner et al. (2014) presents the usual sensors (Yaqoob et al., 2019) employed to solve the perception phase of AN applied into AV. These sensors are:

- Camera: A battery-powered or energy harvesting device, which transfers data (2D or 3D) in the form of images or videos. A usual new sort of this device currently being applied by AVs is the smart camera, which detects accidents, measures the traffic, detects wrong way-drivers, etc (RENESAS, 2020). Elon Musk believes that any other vision system than cameras is needed for a self-driving car (Musk, 2019). This affirmation demonstrates how relevant is this sensor for AV's perception. This device is mainly used for the object detection phase, and its processing is quite complex, involving stages such as image segmentation, calibration, convolution, and other techniques. Object detection relies on renowned image processing algorithms and machine learning libraries, such as you only look once (YOLO) and open-source computer vision and machine learning software library (OpenCV)(Szeliski, 2010).
- RaDAR: Radio detection and ranging is an electromagnetic device that detects distant objects and infers their distances through the reception of the reflected signal when it collides with the obstacle. The time calculated between the wave transmission and the reception allows the object's spot location. Estimating the obstacle's speed and capturing its pose more accurately is possible in more sophisticated radars (Mark et al., 2010). In general, Radar can measure the object's distance, angle, velocity, and cross-section. An AV can use multiple radars, from the rear to the front part of the car (usually fixed in bumpers). The Radar can be classified by type (but is

not the unique classification): Short Radar Range (SRR), Medium Radar Range (MRR), and Large Radar Range (LRR), reaching the range from 70 meters to 250 meters. The rear Radar is typically mounted on the rear and is used for blind-spot detection and parking assist systems. The MRR and LRR are mounted on the front and responsible for traffic jam assist, ACC, and emergency braking, respectively (Skolnik, 1962).

- LiDAR: Light Detection and Ranging (LiDAR) is an optical technology of remote detection which measures the light properties to obtain the distance and the shape of the surrounding environment. Its working principle is similar to the Radar regarding the reflection phenomena but not concerning the device's nature. The output from a LiDAR is point cloud data (PCD). The LiDAR is employed for object detection, tracking, and vehicle localization. The raw data from this device must be post-processed to remove useless, problematic, and redundant data. Its processing involves ground downsampling, PCD fusing, filtering, object detection, shape extraction (Petrovskaya and Thrun, 2009). One well-known Lidar is from the Velodyne enterprise, which presents a significant range and high resolution.

- Ultrasonic: The main component of the ultrasonic sensor is the transducers, which emit sound waves in the air and, after touching an object, receive this sound wave again. This sensor data is processed using the trilateration method to get the obstacle distance from the bumper. A combination of four or six sensors in the front or rear bumper and advanced tracking algorithms, which sense the steering angle is responsible for realizing the autonomous parking.

The sensors described above integrate the perception module of AD. After the sensor fusion procedure, they produce an object list containing reliable information from the objects surrounding the AV (Aeberhard, 2017). These sensors are also applied in charge of the localization functionality, where they are fused with AV's internal sensors, such as Inertial Measurement Unit (IMU), wheel encoders, and odometry. As these data come from different sources, at different rates and qualities, a filtering process using a Kalman filter or a nonlinear filter (EKF or UKF) is necessary (Caliskan, 2020).

The Figure 6 shows the primary AV sensors applied in a simulation using Autoware.Auto ADS.

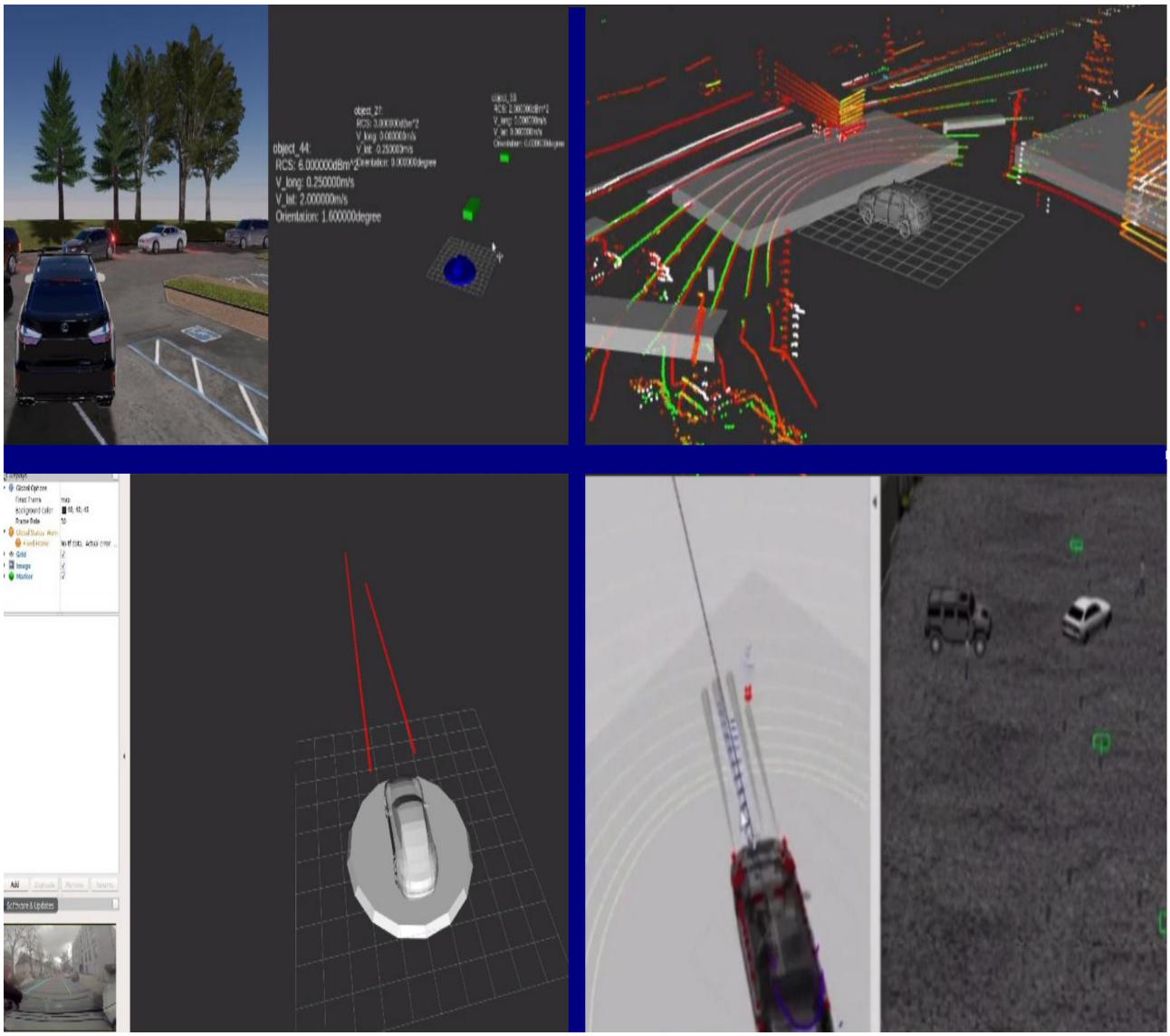


Figure 6 – AV's Sensors: Up Radar Object List (Left), LiDAR PCD (Right), Bottom: Camera Lane Tracking (Left), Ultrasonic Pedestrian Detection (Right). Source: Own Authorship

2.5 Path Planners & Collision Avoidance for AVs: An Overview

As related in the previous subsection 2.4, the last decades were marked by numerous technologies to deal with autonomous driving. One of the first significant events regarding autonomous driving was promoted by DARPA, which first took place in 2007. In one of these events, the winner implemented an RRT-based motion planner to deal with the driving in real-time (Kuwata et al., 2009). The algorithm calculated all feasible nodes through a sample of candidate nodes, which fed the controller. To find an optimal path, a cost-to-go function was set, the implemented heuristic allowed the vehicle to avoid the

stooping nodes, which reduced the wavy trajectories. As a result, the AV did not need to stop route recalculation and performed smoother maneuvers.

The adapted RRT contained a pure-pursuit fused with a PID controller model to adjust the moving control and speed. Besides that, the algorithm recalculated the robustness of the most feasible generated path and also the repropagation of the best route. The CR-RRT algorithm allowed the race car to perform all the required maneuvers from the DARPA competition, such as getting in and out of the parking, applying evasive maneuvers to avoid a collision, and finishing the race (Kuwata et al., 2009).

As well as other projects initiated by the defense and army departments, autonomous driving has also become a target of interest for large companies in the automotive industry and university researchers. The BMW group sponsored a project to develop autonomous driving on German highways (Aeberhard et al., 2015). The vehicles used redundant information collected by 12 embodied sensors to build the road map. A hybrid system was designed to control the lateral and the longitudinal displacement to guarantee safety and smooth driving. These AVs embed a robust technology that detects how fast the other vehicles are, measuring their speed through a combination of a static and a dynamic model. Conducted simulations comparing scenarios with human driving and autonomous driving inside the Program for Development of Longitudinal Traffic Processes (PELOPS) presented results that support the Highly Automated Driving (HAD) insertion, once the data exhibited a considerable number of collisions reduction when applying the AV in replacement to the human being This enterprise idea is reducing the number of accidents next to 0 (Bahram et al., 2014).

Despite some advantages over the sensory part and more straightforward mapping than AUV, the AV also faces particular adversities in their applications. The overtaking maneuvers, sunlight reflections, blindness, passing at crossroads are investigated in this subsection.

To reduce the overtaking risk, Li et al. (2014) presented a method to merge maps generated by different vehicles aided by the use of a Genetic Algorithm (GA). The map can be used to avoid collision in autonomous driving. Once a car is located back, another will share the view of this vehicle. The algorithm is developed through the Overall Direct Optimization (ODO) technique. In this method, the similar areas of each vehicle's map are locked, and the remaining grids are filled based on an objective function (F_c), which measures the consistency degree of the cell's equivalence between the maps. Mutations are realized in several iterations to achieve the most significant value of F_c to generate the optimum merged map. The study demonstrated that the GA solves the convergence of almost all map points on average with only ten iterations. Thanks to the data fusion obtained by this Evolutionary Algorithm, the driver with limited vision in advance can check whether there are obstacles in front of the vehicle he desires to overtake. The Figure

7 depicts an image of this technique application.

Figure 7 – a) Car B with vision occlusion b) Car B "see" through occlusion after GA application. Source: Li et al. (2014)

Yang et al. (2019) developed an optimization in the dynamic control in autonomous driving. The study applied the Convolutional Neural Network (CNN) to implement deep learning to perform autonomous driving. This artificial intelligence model was elaborated by combining the Feature-Based Map and the Structural Errors Based Map. The most activated neurons built these maps feeding the machine learning software composed by the TORC-Matlab software framework. In this training phase, the heading and dynamic properties were adjusted to achieve the best trajectory generation with the shorter Euclidian Loss, requiring numerous iterations to achieve the optimum path. The results of the driver simulator revealed small oscillations in the vehicle's maneuvers, presenting a behavior similar to human driving. The maximum spatial errors verified in experiments were about 37 cm in position and 0.6 degrees for steering.

An intense collision, which unfortunately generated the first death in autonomous driving. Paul and Chung (2018) applied High Dynamic Range (HDR) imaging algorithms to resolve the video image processing issue in direct sunlight situations. The Standard Dynamic Range (SDR) imaging algorithm used in Tesla Model S was insufficient to predict a white tractor when the car faced high-intensity bright sunlight. To solve this issue, the scientists off the Lawrence Technological University applied the HDR algorithm into the Autonomous Campus Transportation (ACTor) vehicle to verify and compare the produced image quality. The HDR algorithms assigned fair values for each generated pixel through an average calculus for each pixel captured by the camera in a period or directly setting visual parameters, such as contrast and saturation, in real-time. The simulations proved that the HDR algorithms, especially the Mertens HDR, delivered a better image of the environment than the SDR algorithm, preserving the image data and the computer vision

features (contrast, color). This image optimization dramatically improves the perception phase of AN, and thus it can prevent the vehicle from collisions in direct sunlight cases.

Noh (2019) addressed the collision issue at road intersections combining the motion prediction, threat assessment, and decision-making procedures. A predicted future path digital map was used conjointly with a threat assessment algorithm to foresee the vehicle's move and eventual collision probability. The algorithm is generated through threats estimation was given by the awareness situation, situation assessment, and maneuver decision modules. The modules consider the temporal and spatial relationships between the vehicles near the same road intersection. Then an independent reasoning agent (IRA) values area is assigned for each vehicle-vehicle collision probability relation (V2VCR), highlighting the most relevant (dangerous) automobiles. Bayesian Networks and Time Window Filtering methods overcame the loss of information and uncertain noise data acquired during the road track. The Time To Enter (TTE) strategy (the time a vehicle enters a collision area, based on its current position) was crucial to elaborate the collision likelihood function used in the algorithm. Using Danger, Attention, and Safe area metrics, the experimental vehicle avoided imminent threats by applying proper behavior without resorting to overly conservative safety behavior. The method resolved with excellent reliability and efficiency the AN also in unsignalized environments and when other vehicles violated traffic rules (such as happened with Google AV in 2016 (Dolgov, 2016)).

Yoon et al. (2018) optimized the path planning and obstacle avoidance applying the spline-based RRT (SSRRT*) algorithm. The method consists of creating Bezier Curves through the use of path-following controllers represented by control points related to the car's shape and dynamic. The Bezier Curves are drawn over the two extreme ends of a rectangle. These points are described by vectors representing the rear inner axle and the front outer corner of the vehicle. These positions support the dominant generating trajectories among all the courses caused by all of the car's corners from possible paths (curves). The SSRRT* algorithm was improved, modifying the non-optimal feature intrinsic to the traditional RRT algorithm. The algorithm solved the collision avoidance phase requiring less computational effort (reduced the time complexity). It decreased the path cost for a reasonable number of expanded nodes, compared to traditional classes of RRT algorithms, such as Rectangle Sample RRT (RSRRT*) and SRRT*. Besides that, particularly for narrow corridors, the algorithm showed to be the most suitable to avoid the collision. The simulations showed that this check collision algorithm-generated much fewer nodes during the tree expansion when the vehicle was approximated near these corridors. The authors point out others algorithms did not work suitable with the constraints of the nonholonomic vehicle and presented worse results, colliding in an obstacle dilatation scenario. The SSRRT* algorithm discretizes the UGV model using angularly discretized rectangles. As a result, they adjusted the nonholonomic vehicle constraints (turn radius) and calculated the required space of the moving vehicle with more accuracy than other

methods. Through this improvement and modifying the original RRfT search algorithm, the SSRRT* became more able to plan paths with variable curvatures continuously and efficiently.

Numerous other challenges are part of the HAD, and there are several state-of-the-art techniques and algorithms proposed to solve them. Some of the graph search and CASs used to fit the AN can be contemplated in the Table 1.

Table 1 – Collision Avoidance Systems for AVs (CASs).

Obstacle Avoidance Strategies	**Literature**
Traditional Algorithms	
Bug Algorithms	Kamon et al. (1996); Lumelsky and Stepanov (1987); Lumelsky and Skewis (1990)
Vector Field Histogram (VFH)	He et al. (2015); Borenstein and Koren (1989)
VFH+	Ulrich and Borenstein (1998)
VFH*	Ulrich and Borenstein (2000)
The Bubble Band Technique	Khatib et al. (1997); Urmson et al. (2007); Yoon et al. (2018)
Elastic Band Concept	Quinlan and Khatib (1993)
Curvature Velocities Techniques (CVM)	Yang et al. (2019); Simmons (1996)
Dynamic Windows Approaches	Aeberhard et al. (2015); Minguez et al. (2002); Fox et al. (1997); Brock and Khatib (1999)
The Schlegel Approach	Schlegel (1998)
Nearness Diagram	Minguez and Montano (2004); Mínguez and Montano (2002); Iturrate et al. (2009)
Virtual Force Field (VFF) Methods	
Gradient Methods	Rostami et al. (2018)
Worst Case Estimation	Konolige (2000)
Bacterial Potential Field	Montiel et al. (2015)
Genetic based Algorithms	
Biological Approach	Chen et al. (1997)
Hybrid VFF-Genetic Algorithms	
Evolutionary Behaviour based on Genetic Programming	Clemente et al. (2018)
Geometrical Methods	
Collision Cone	Snape et al. (2011)
Fuzzy / Neurofuzzy Relational Products	Tzafestas and Tzafestas (1999)
Anti-target Approach Laws	
Cone's Geometry based Calculated Rule	Chakravarthy and Ghose (1998)
Graph search algorithms	
RRT, BFS, PF, and so on	*will be discussed in detail in Section 3.5

Source: Own Author.

3 Path Planning: A Deep Insight

3.1 Challenges of Autonomous Mobile Robot Navigation

The main feature of autonomous vehicles is moving independently from a starting point to any choice point, through a changing environment, without collisions. To achieve this goal, the autonomous car relies on solving a series of tasks that must be progressive and synchronized. The Figure 8 depicts an overview of the sub-problems that an Autonomous Mobile Robot (AMR) must accomplish to perform autonomous navigation. Roboticists create ROS packages that integrate a specific module to solve the correlated sub-problem.

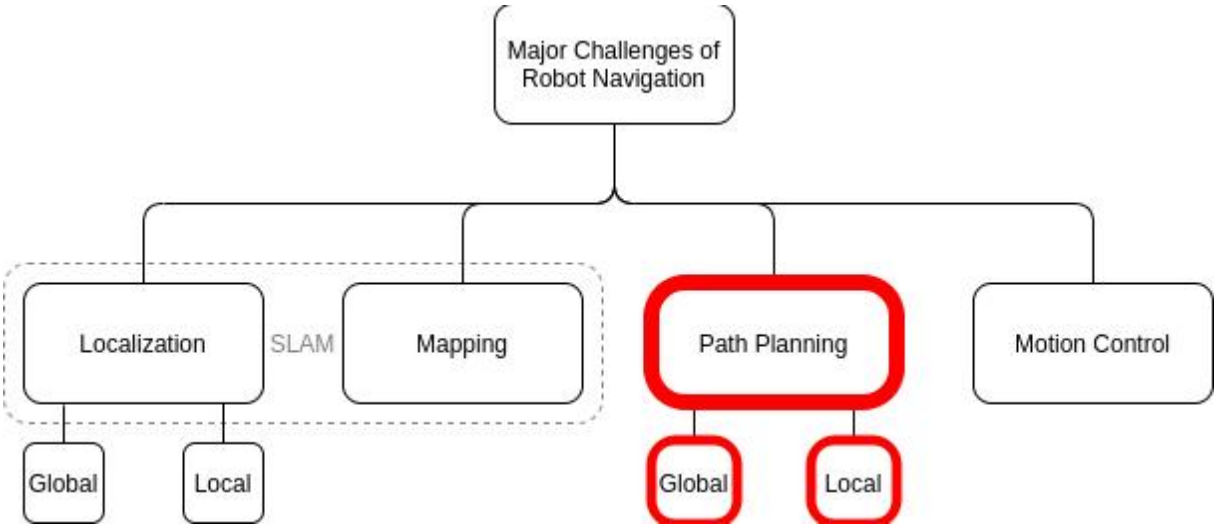


Figure 8 – Robot Autonomous Navigation Interrelated sub-problems. Source: Zegers (2020)

As the Figure 8 brings us, an autonomous vehicle must solve least sensing, perception, localization, path planning, and motion control tasks to bring the AV from point A to point B. Despite the need to use all the modules related previously, it is not the scope of this dissertation to cover all of them detailed but focus on the path planning theme.

3.2 Path Planning Concept

Path planning is one of an autonomous robot's most basic and vitally essential skills, enabling it to carry out assigned tasks. An autonomous vehicle context consists of finding an adequate route that the vehicle can follow to move from a start point to a parking spot and target another parking spot in another city without hitting obstacles. Navigating

safely around people and other vehicles are features that must be assured. A path planning algorithm would take the start and goal location as input and produce a sequence of valid waypoints according to the local's High digital (HD) map. The selection of path planners will depend on the environment, situations, and robot kinematics. Path planning can be either online or offline, and these methods are also usually called static or dynamic. In any jargon, the distinction refers to whether the entire path is calculated before the motion begins, with a previously existing map or incrementally, during action using recent sensor information. Concerning the kinematic constraints, path planning can be classified into holonomic if these constraints are considered or nonholonomic path planning if they are not. "If the generated path also considers constraints on velocity and acceleration, the term kinodynamic path planning is used." The Figure 9 depicts the path planning classification.

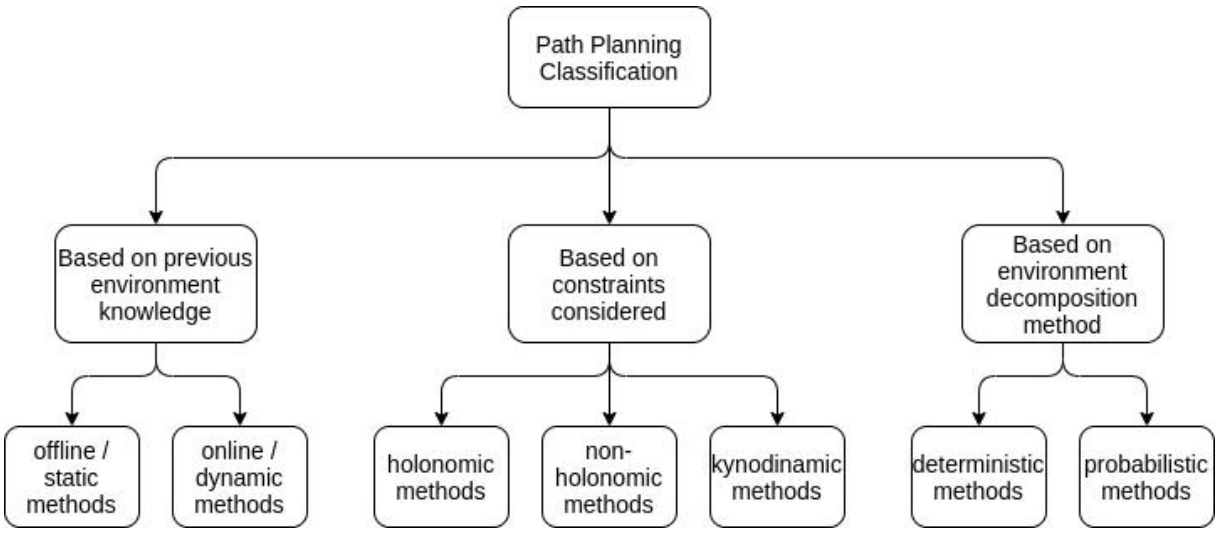


Figure 9 – Path Planning Classification. Source: (Zegers, 2020)

Both global and local planners are needed during an autonomous vehicle mission. The global planner 'uses a previous know map of the environment. This map is used to find the shortest path. On the other hand, the local path planner does not use a previously known map. This planner gets and generates a local map based on sensor readings. Therefore a robust autonomous navigation system relies at least on the use of 2 path planners: a global and a local, in a two-level planning architecture. "The global path planner is concerned with long-range planning and uses the available map information, which can be slow, but is key to finding the most efficient path to a distant goal. It is not concerned with the robot's dynamics or avoiding unexpected obstacles left to the local path planner. In this way, each planner deals with only one set of concerns: finding a traversable path to a distant goal and following that path while reacting to unforeseen situations like the appearance of obstacles. Therefore, global and local path planning can be complementary solutions and are commonly built into path planning systems in real-world applications.

3.3 Planning Libraries

In this section, the planning libraries and usual algorithms for path planning will be prescribed.

3.3.1 OMPL

A key goal of robotics is to demonstrate and validate the improvement of specific parameters of a new motion planning algorithm implementation. For that, the researcher must develop accurate metrics and methods to compare the algorithms against each other and obtain reliable Benchmarks. This task, however, is often very challenging and exhaustive. The Kavraki Laboratory from Rice University developed the open motion planning library (OMPL) to overcome this arduous and time-consuming process.

The OMPL is a library designed for sampling-based motion planning. It contains the implementations of state-of-the-art algorithms, such as PRM, RRT, EST, SBL, KPIECE, SyCLOP, and several variants of these planners. The OMPL is limited to providing these algorithms functionalities, which means there is no environment specification, no collision detection, or visualization. Although OMPL can not directly simulate collision detection, once the robot's geometry, control, and environment be not defined within it, this library contains the classes and functions that will allow easy and effective integration with other software. The Figure 10 depicts the main courses that can be instantiated by other software to accomplish with the specific robot's characteristics into a simulation:

This arrangement allows OMPL to be easily integrated into systems that provide additional needed content, such as OMPL.app, MoveIt and the Robot Operating System (ROS) Sucan et al. (2012). To use OMPL's features, the user needs:

- Instantiate the Space to plan (car's case)
- Create the planning context
- Specify the function that distinguishes the valid states
- Specify the input start and goal states
- Finally compute the solution

However, the above steps are implemented in robotics software such as the ROS manipulation software stack known as MoveIt and on OMPL's graphical front end such as OMPL.app and A Primer (Kavrakilab, 2020).

The graphical interface enables the user to tune desired planners parameters, set the start/goal positions and other features. The OMPl.app, for example, contains a

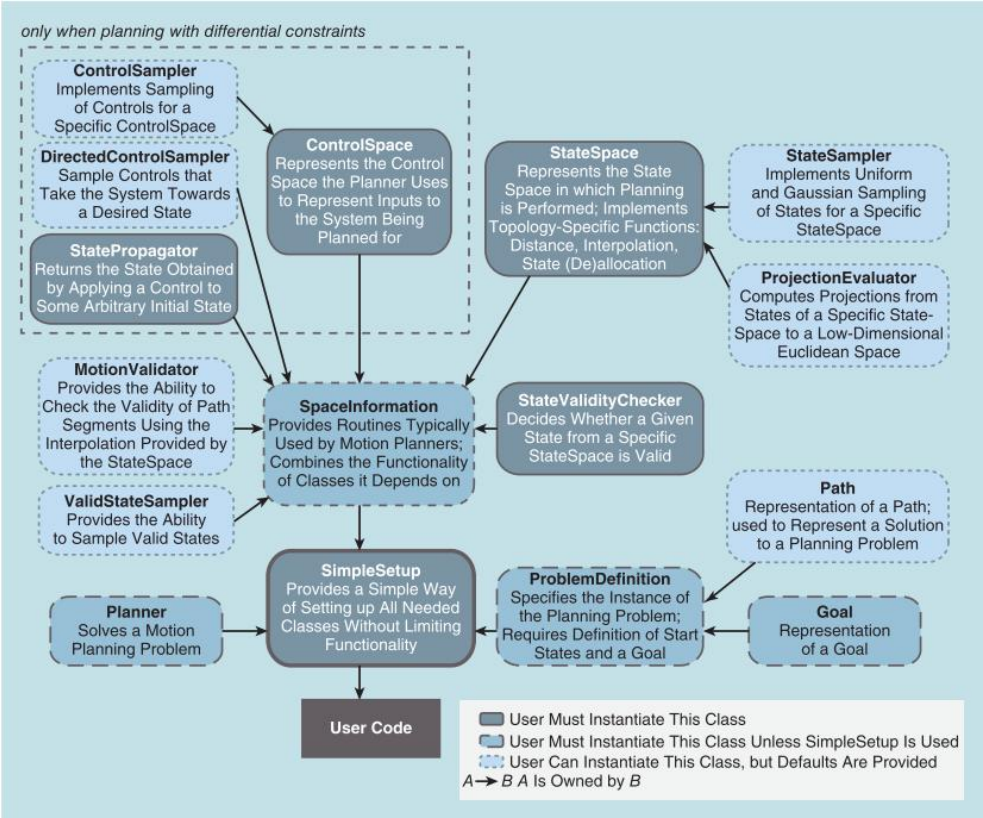


Figure 10 – Overview of OMPL structure

Source: Sucan et al. (2012)

lightweight for Flexible-Collision-Library (FCL) and Proximity Query Package (PQP) collision checkers. It also allows the loading of 3-D models and geometries for the robot and environment, ensuring a rigid body motion planning simulation functionality. The Figure 11 depicts a simulation using the OMPL.app graphical interface for the Reed's-Shepp car geometry.

This interface enables OMPL to simulate some rigid bodies and vehicles types (Plaku, 2020). A more optimized and close-to robot's reality simulation can be done thanks to integrating Robot Operating System (ROS) and Gazebo simulator implemented by Willow Garage Team. In the end, a log text file can be used programmatically to generate a complete Benchmark to assess and compare the planners.

3.4 SBPL

3.5 Global Planners Work Principle

To find a feasible path between the start position and the goal position, the robot path planners' algorithms must calculate the trajectory over a map. In robotics, this map is

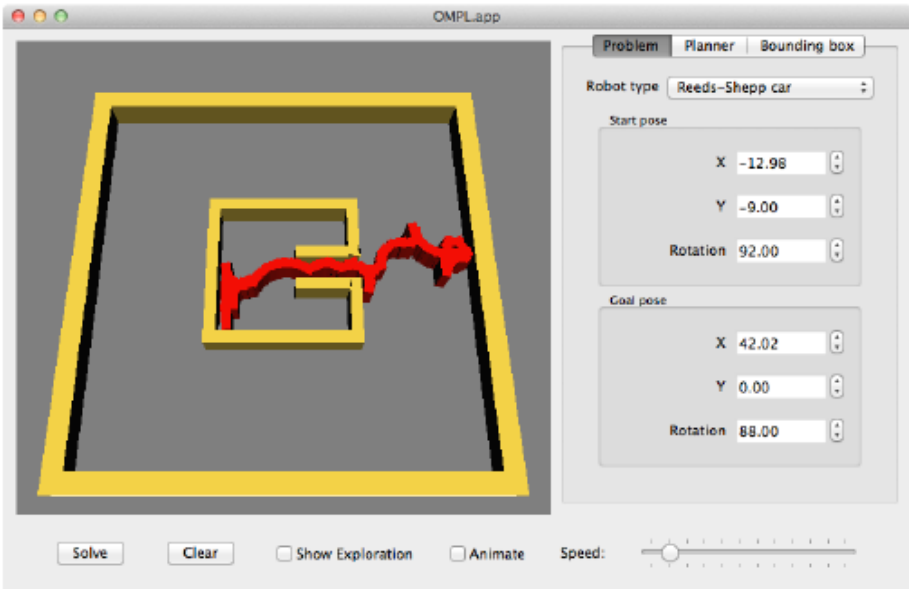


Figure 11 – A solution path is shown for a car-like robot driving out of a "bug trap" environment. Source: Moll et al. (2011)

created by getting, processing, and updating sensor data, usually from Laser Detection and Range (LIDAR) sensor. An occupancy grid is a discretized grid surrounding the current ego robot position. This discretization can be done in two or three dimensions. Each square of the occupancy grid indicates if a stationary or dynamic object is present in that grid location (Sebastian Thrun, 2005). If yes or not, the grid is classified as occupied or free. Each square takes a binary value, indicating if it is busy or not, as displayed the Figure 12:

Once the robot already generates the occupancy grid map, this one must apply a sampling strategy of waypoint generation which will configure the final trajectory. This strategy is the working principle of a path planner itself. The 0 value assigned means a free from collision space, and 1 means an occupied space. Below, the sub-sections will describe the most valuable planners applied to this research project.

3.5.1 Dijkstra

The Dijkstra algorithm keeps track of the shortest distance from the start node to each grid cell. This metric is known as the node's g_cost. Before the search process starts, the starting point gets a g_cost of 0. Later, as the algorithm progresses, these values will be updated to the actual shortest distance from the start node. This algorithm relies on numerous iterations and cycles of visiting the neighbor grids, adding them to the available list, verifying if this neighbor has the smaller g_cost against the other grids added to this open list. Afterward, the visited grids are put into the closed list, and these cells' g_cost are compared against each other to verify which one presents the smaller value. The neighbor cells with more expensive g_cost are removed from the closed list to remain

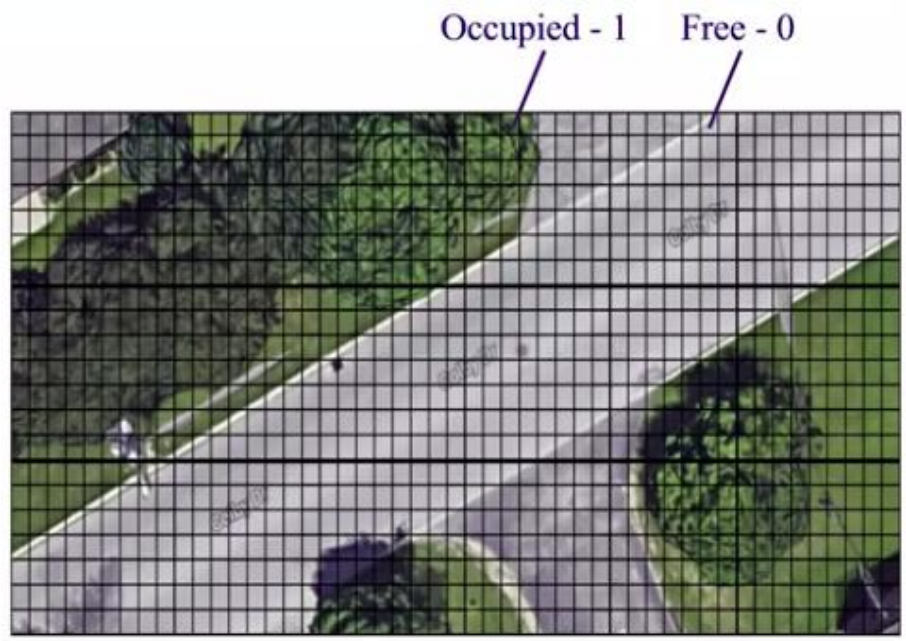


Figure 12 – The occupancy grid map of a city environment.

Source: Waslander (2021)

on the sequence of cells with the minor cost and, therefore, the shortest path. The current node is continuously updated when all possibilities of the neighboring cell are exhausted, and the least costly one is checked. This process is done repeatedly until the robot achieves the goal spot. The closed list is programmatically in inverse order. The last grid cells added are the first numbers on this list. Then to finally find the path, the robot must follow the backtracking process is required to reverse this list. A snapshot of this process is depicted below, where Q_0 is the start point and N_s the goal point.

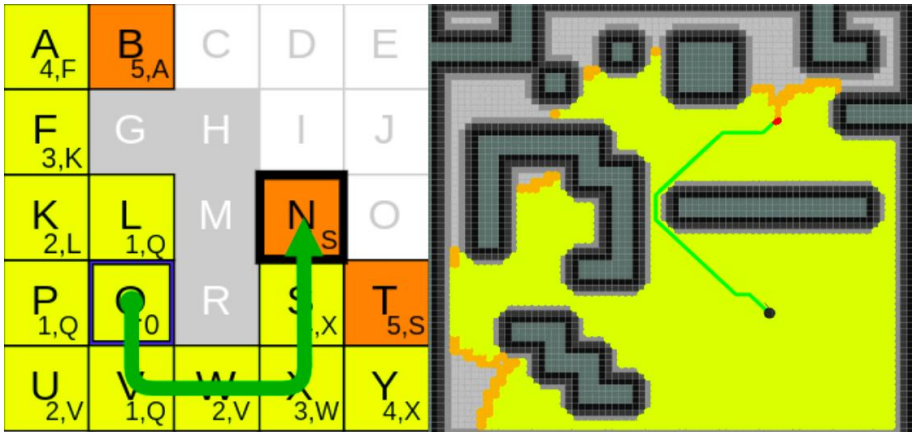


Figure 13 – The Dijkstra's algorithm grid-search strategy on the left. On the right the application of this algorithm on Robot Ignite Academy Development Studio

Source: Zegers (2020)

The yellows cells on Figure reffig:Djkistra represents the already visited nodes during the Dijkstra process; therefore, the nodes are put into the closed list. The Orange squares represent the nodes inside the empty list. Finally, the green arrow indicates the shortest path calculated by Dijkstra. As we can verify in the image above, the algorithm searched in all directions in a radial pattern (yellow area). This occurs because Dijkstra is an uninformed search algorithm known as blind search. This means that it is now aware if a grid cell is better than others to be chosen during its expansion process. This feature makes it generally slower than informed search strategies such as A* (see subsection 3.5.3) and RRT (see subsection 3.5.4) especially as the map size increases.

Although Dijkstra leads to an optimal solution, it expands to many nodes. Thus, it is a very CPU-intensive algorithm. This may result in a slow path search process or failure to find the path in time, in low time to react. Additionally, the further the search expands, the more memory is needed. This can sometimes be a problem since memory requirements increase exponentially as the map size or space dimensionality increases. Last but not least, this algorithm calculates the trajectory for the static environment. This means that it produces a rigid path, and as a result, if the environment changes, the robot could move along a potentially obsolete way Zegers (2020).

3.5.2 Greedy BFS

The Greedy-Best-First Search (Greedy BFS) algorithm shares most of its code with Dijkstra's shortest path algorithm, except that it expands by selecting the node closest to the goal. A heuristic function is used to estimate how far from the goal any node is. This heuristic allows the algorithm robot to focus its traveling in a specific direction. Heuristics often used for this purpose are the Euclidian Distance, an imaginary straight line between two points (goal node and start node), and the Manhattan Distance, which is the distance between two points measured along the axes at right angles. Although these heuristics do not guarantee an optimal solution, they have the advantage of significantly speeding up the path solution finding process, running much quicker than Dijkstra. The image below the Greedy BFS behavior.

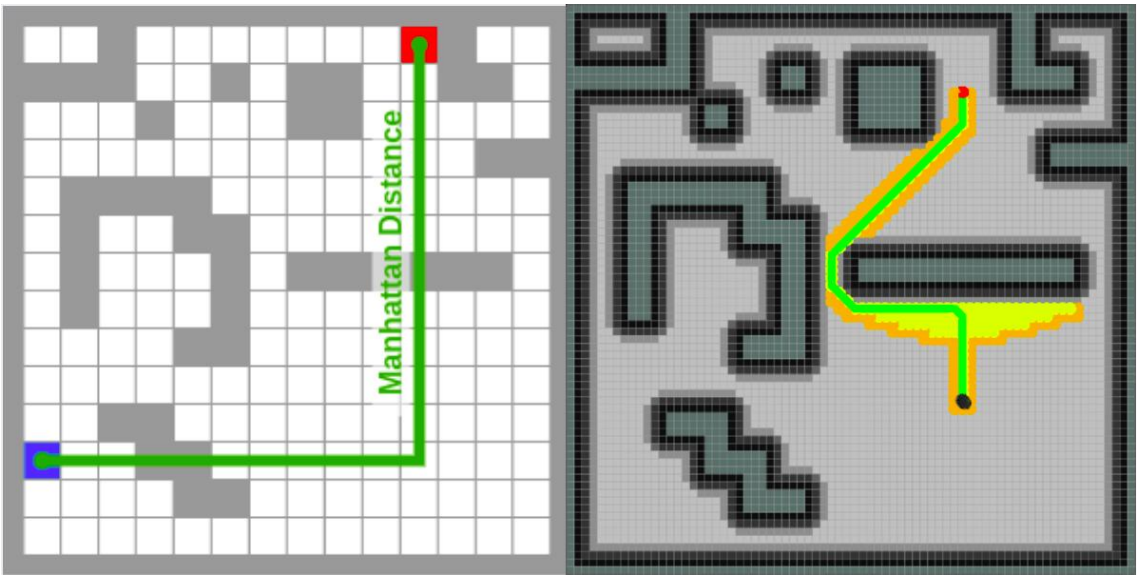


Figure 14 – The Manhattan Heuristic strategy. The output of this heuristic implemented on greedy BFS algorithm with the robot search results on Robot Ignite Academy Development Studio

Source: RIA (2020)

As we can visually check, the Greedy BFS focused its exploration to the desired direction, north, and avoided visiting neighbor nodes in other directions in contrast with Dijkstra. Although Greed BFS presents good results in a significant number of environments, it can be easily trapped if find regions with significant or concave obstacles. When this algorithm relies on these scenarios, it produces long paths far from optimal, which is a considerable disadvantage in energy consumption and travel time to the robot executing this path. The Figure 15 demonstrates this situation:

In the same scenario, Dijkstra would produce the shortest path at the cost of time. And the Greedy BFS the inverse. Every algorithm presents advantages and disadvantages, and the most suitable can be chosen for a specific project's needs.

3.5.3 A*

In contrast with the Dijkstra algorithm and the Greedy BFS, the A* is an informed search algorithm. This means it utilizes the information about the goal location to guide the search towards the target and produce a more efficient map exploration. Therefore the exploration process is not open to all directions but focuses on the selected and desired specific one. A* has proven efficient and effective as it is quick and finds the shortest paths. The critical factor of this success is that it combines the distance from the start node to the current node as Dijkstra does and the estimated distance from the current node to the goal node, like Greedy Best-First Search does. These two pieces of information are

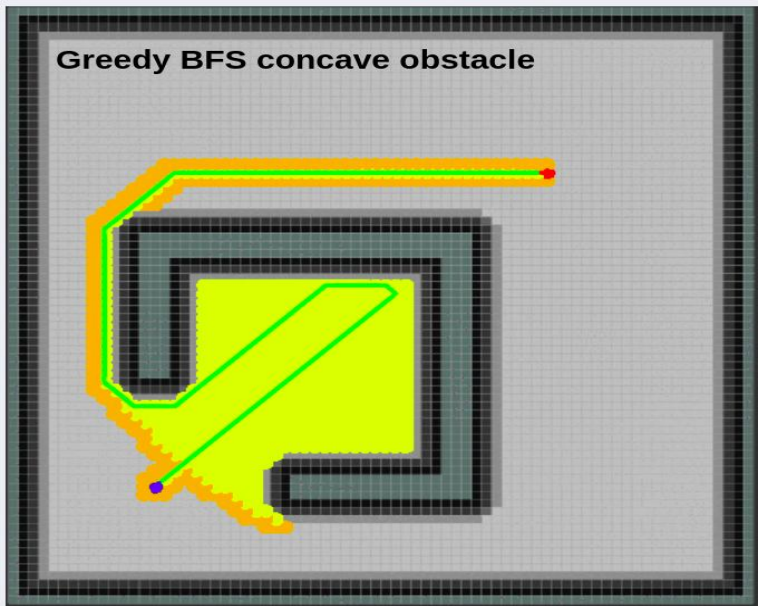


Figure 15 – Greed BFS behavior when facing a concave obstacle environment.

Source: RIA (2020)

combined to create a value known as the total cost of the node, denoted as f_cost. The total cost f_cost is equaled the g_cost summed to h_cost, where:

- g_cost represents the exact travel distance from the starting point to any node n
- h_cost represents a heuristic distance from a node n to the goal location

Whenever A* expands its search to a new node, each candidate is evaluated to its total cost, given by f_cost. The node with the smallest f_cost is selected as the next node to be explored. The output of the A* implemented on RIA development studio for the turtlebot robot is depicted below:

As we can check, A* focused the search on the specific direction and avoided at the same time concave traps. The algorithm efficiently combined the advantages brought by Dijkstra and Greedy BFS, generating the trajectory with satisfactory speed and time.

A* is a complete algorithm, meaning that it will always find the solution if it exists. A* is also an optimal algorithm, once it will always find the shortest path if one exists. These features, however, come at a cost. A so-called deterministic algorithm is required to see a complete and optimal solution. A* is a deterministic path planning algorithm, meaning that it always produces the same path (given the same start, goal, and map input), following the same computation steps. Deterministic algorithms, however, do not scale well with the map size. The more nodes to process, the more difficult it becomes to keep up with the planning time requirements. In addition, large maps require a lot

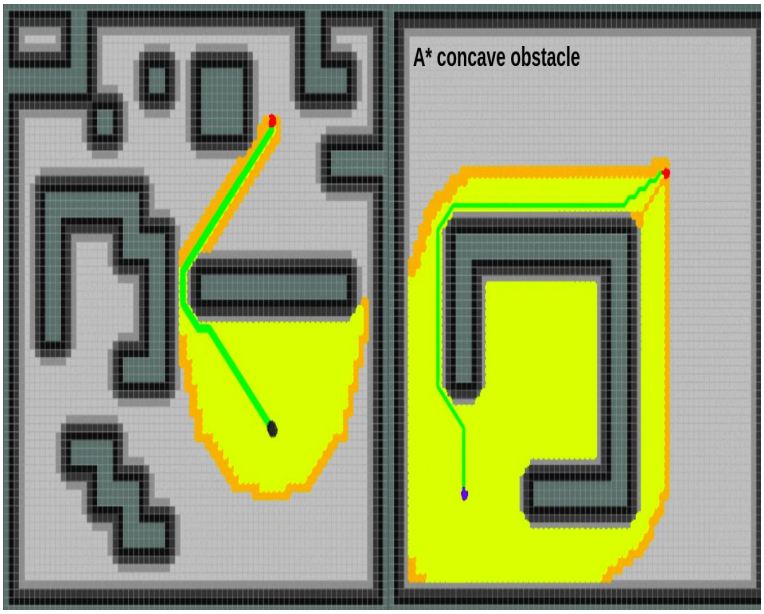


Figure 16 – A* algorithm trajectory output when facing a concave obstacle scenario at left and traveling to a north goal spot on the right.

Source: RIA (2020)

of memory since each node discovered has to be accounted for. The robotics community relies on probabilistic algorithms that sacrifice optimality and completeness to win in computational efficiency to overcome this memory issue related to the large area to cover. A usual employed for these use cases is the RRTs, which are better described in the following subsection.

3.5.4 RRT

The probabilistic path planning method is prevalent and suitable for solving complex, high-resolution, and high-dimensional path planning problems. Instead of meticulously examining directly connected grid cells, sampling-based algorithms generate candidate waypoints at completely random positions in the map and determine if these positions can be connected without hitting obstacles. The Rapidly-Exploring Random Trees is one such algorithm. It is regarded as one of the most efficient tools for robotic path planning in higher dimensions (Zegers, 2020).

The RRT plans a navigation path in a Q C space, where Q_{free} is the free space and Q_{obs} is the space that contains an obstacle. The root node of the tree q_{init} starts expanding a G tree randomly from the root node until one of its branches reaches the final point q_{goal}. The nodes are added to connect these points (initial and goal nodes), generating the path R (route). The q_{near} is the closest point of a random node generated through the predecessor node, and the q_{new} is the leaf node (node located of one of the extremities).

The distance between the q_{new} and the q_{near} is Δq, and if there is no obstacle between them (no Q_{Obs} interception), the tree is extended. The same collision procedure is executed between the q_{new} and the q_{goal} to achieve the desired position. The Figure 17 shows a pseudocode for the RRT algorithm.

Algorithm 1 RRT Algorithm Pseudocode

```
1:  procedure RRT(Q, q_init, q_goal, Δq, l_d, n)
2:      G ← { }
3:      R ← { }
4:      s ← 0
5:      i ← 0
6:      while (s = 0) and (i ≤ n) do
7:          i ← i + 1
8:          q_rand ← RAND_CONFIG(Q)
9:          q_near ← NEAREST_VERTEX(q_rand, G)
10:         q_new ← NEW_CONFIG(q_near, q_rand, Δ_q)
11:         if q_near q_new do not intercept Q_obs then
12:             EXTEND(G, q_near, q_new)
13:             if d(q_new, q_goal) ≤ l_d and (q_new q_goal) do not
    intercept Q_obs) then
14:                 EXTEND(G, q_new, q_goal)
15:                 s ← 1
16:                 R ← ROUTE(q_init, q_goal)
```

Figure 17 – RRT Pseudocode. Source: Véras et al. (2019).

The RRT is an optimization of the random trees (RTs) methods, which creates nodes in an unexpected direction. Once its working principle differs from RTs, it builds the nodes' links (edges) toward the goal. In this way, the RRT finds the goal node faster. A more advanced RRT model in the optimality sense is the RRT* algorithm which rewires the tree to form the shortest paths. As more samples nodes the programmer includes in the model, the shorter (optimum) the path planner becomes. On the other hand, the search time increases. The Figure 18 displays a comparison of the previous short-described methodologies acting to achieve the goal in a local minimum scenario.

In Figure 18, the goal node was set in the exact spatial location, and then the RT, RRT, and RRT* search algorithms were applied. The RT search algorithm used more than 3500 nodes to find the goal node, and it failed. When the user input more nodes, the system crashed and did not produce more nodes. On the other hand, the RRT and RRT* found the goal node more quickly, requiring 431 and 501 nodes, respectively. The difference between them is that the RRT* generates an optimum path than RRT. Although the RRT* used more nodes to find the goal, it generated a shorter route than RTT. In this way,

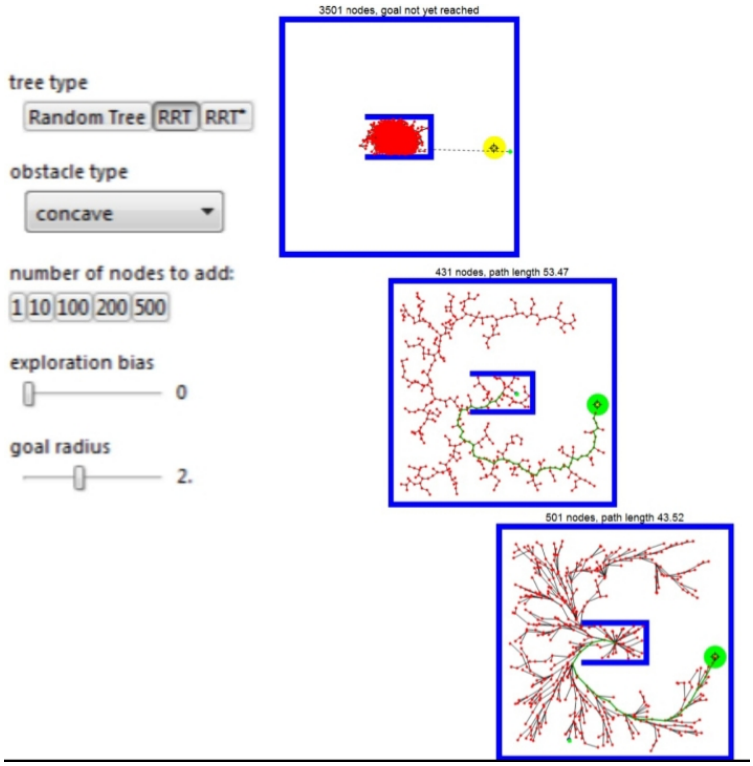


Figure 18 – Comparison of optimality from RTs, RRT and RRT*. Source: Adapted from Aaron and Huang (2019)

with little computational effort (to process more nodes), the vehicle spends less energy and less time performing the trajectory.

All these displayed sample-based algorithms related to the random trees are probabilistically complete, which means that if a nonzero width path exists, the tree will eventually find the trail. These algorithms can be configured in a uniform or non-uniform standard. The RRT searches the goal node in a uniform configuration in a uniform spatial distribution. This non-uniform configuration can still be subdivided into two different sampling strategies: in the second type, the sample collection process can be biased by determining the most promising regions of the search space.

- Importance Sampling: The RRT expands to the regions with a significant interest in the C space;
- Adaptive Sampling: The sampling changes during the planning whenever restrictions are encountered in the navigation space.

These sampling processes are based on a non-uniform distribution, which reduces the number of samples needed to find a feasible path compared to a uniform distribution.

In the first one, the expansion of RRT can be regionally biased, collecting samples next to the significant regions of interest on the C space. This approach is called importance

sampling. In the second, the sampling is changed during the planning, depending on the restrictions encountered in the navigation space. This last approach is called adaptive sampling. By adopting one of these approaches, the sampling process is biased by a non-uniform distribution, which reduces the number of samples needed to find a viable path. A diagram of the essential operation of any RRT is shown in Figure 19:

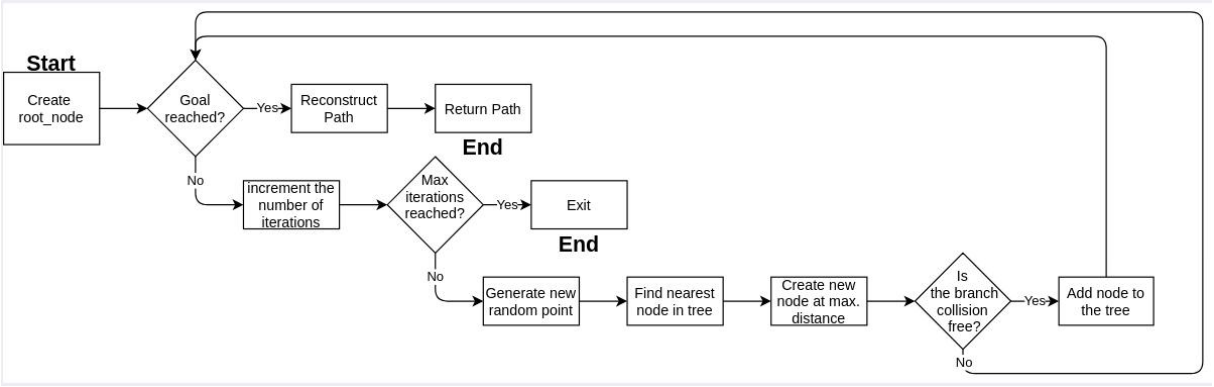


Figure 19 – RRT process overview Zegers (2020)

The exploration process behavior of RRT can be checked in the Figure 20.

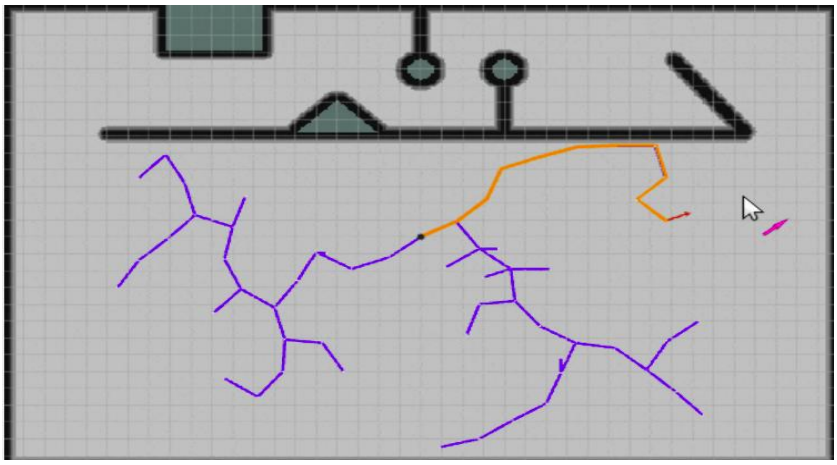


Figure 20 – Implementation of RRT algorithm on Robot Ignite Academy Development Studio. Source: RIA (2020)

There are numerous sorts of RRTs variations and implementations to fit specific project's needs and features, such as creating more nodes near obstacles, taking into account nonholonomic constraints, and others. These variations gain RRT derivative names, such as RRT Connect, RRT*, RRTX, RRT*-Smart, MP-RRT, and many others. Umari and Mukhopadhyay (2017) developed an RRT-based algorithm for the ROS framework. His

algorithm detects unknown frontier regions and enables the robot's exploration in space, finding a free-collision path. This algorithm was simulated in this work for the car-like robot. The Figure 21 depicts the exploration process of RRT in a simulated scenario. The scenario represents the THI's university courtyard in the vicinity of the CARISSMA complex lab.

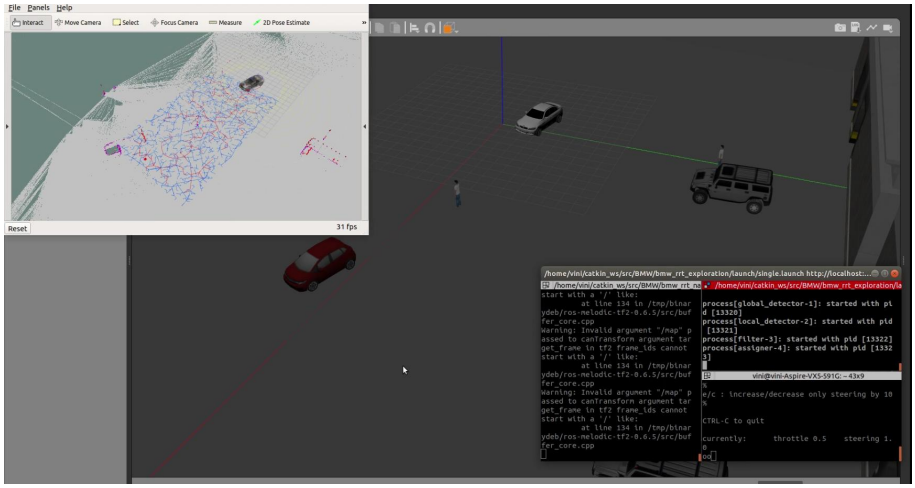


Figure 21 – RRT exploration process

Source: Own Authorship

3.6 Local Planners Work-Principle

The local planners are also known as obstacle avoidance planners. They are responsible for updating the global planner trajectory according to the environment's sensor updates. In other words, the local planner deforms the global planner trajectory if the robots find a new obstacle occupying the grid map where the original course was defined.

3.6.1 Elastic Band

The elastic band algorithm relies on bubbles to represent the maximum local subset of the free space around the robot. Using these bubbles, a collision-free path can be guaranteed to the robot. Khatib et al. (1997) defines how this algorithm works:

> [...]Given a $\{p_{i}\}_{i=1...n}$ configurations in free-space such that the bubbles defined around two consecutive configurations, $\beta(p_i)$ and $\beta(P_i+1)$ overlap it is possible to construct a collision-free path from p_1 to p_n. This path is an elastic band paved with bubbles or a bubble band. This band will be deformed according to two kinds of forces: internal and external.[...] The internal forces remove slack in the band (energy minimization). The external forces move the band away from obstacles. [...]

Within this algorithm, the robot follows the generated path connecting the center points of the band. The implementation of this planner for ROS supports just holonomic robots (Quinlan and Khatib, 1993). An adaptation of this algorithm for differential drive machines can be visualized (Connete, 2019). The Figure 22 shows this algorithm applied for the car-like robot used in one of the simulations of this work (see section 6.1).

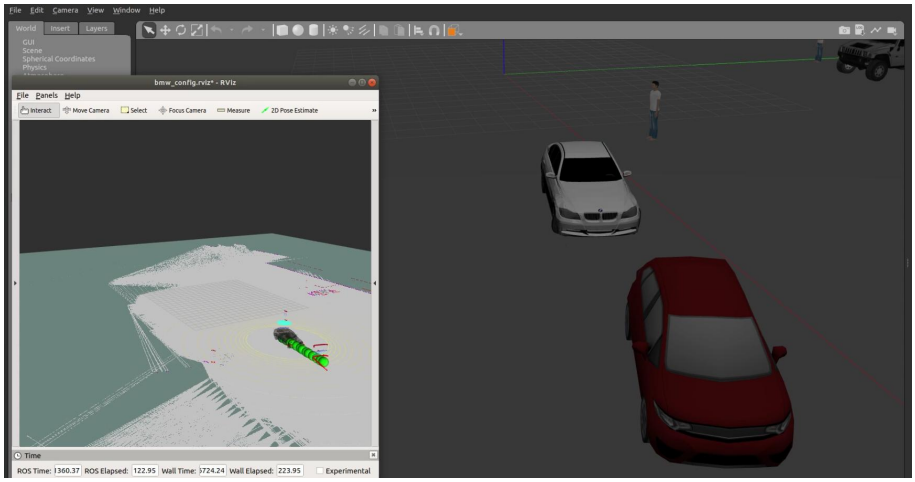


Figure 22 – Elastic Band Motion Planner

Source: Own Authorship

3.6.2 Dynamic-Window Approach (DWA)

In the dynamic window approach, the search for commands controlling the robot is carried out directly in the space of velocities. The robot dynamics are incorporated into the method by reducing the search space to those velocities that are reachable under the dynamic constraints. In addition to this restriction, only velocities are considered, which are safe concerning the obstacles. This pruning of the search space is done in the first step of the algorithm. In the second step, the objective function's velocity maximizing is chosen from the remaining velocities in the second step.s (Fox et al., 1997)

The Dynamic Window Approach (DWA) discretely samples a square in the robot's control space (dx, dy, dtheta). Then a forwarded simulation from each sample velocity from the robot's current state is done to predict what would happen if the sampled speed were applied for some (short) period. Then an assessment of each resulting trajectory, using proximity to obstacles, proximity to the goal, proximity to the global path, and speed metrics to discard and score the trajectories. Afterward, the highest-scoring trajectory is getting and sent to the robot. A sketch of this behavior can be visualized in Figure 23:

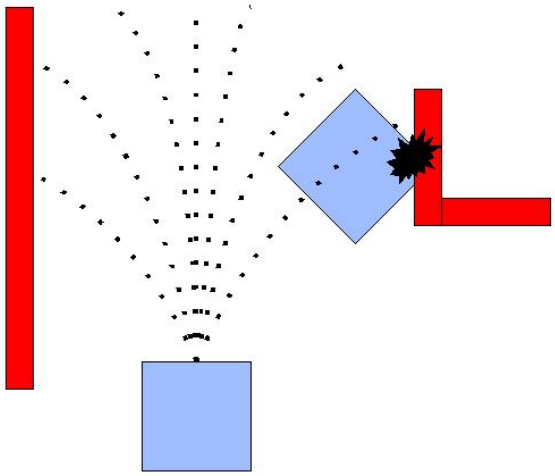


Figure 23 – Dynamic-Window Approach working-principle

Source: Moll et al. (2015)

The DWA local planner algorithm was simulated for one of the car-like-robot. This simulation is shown in Figure 24.

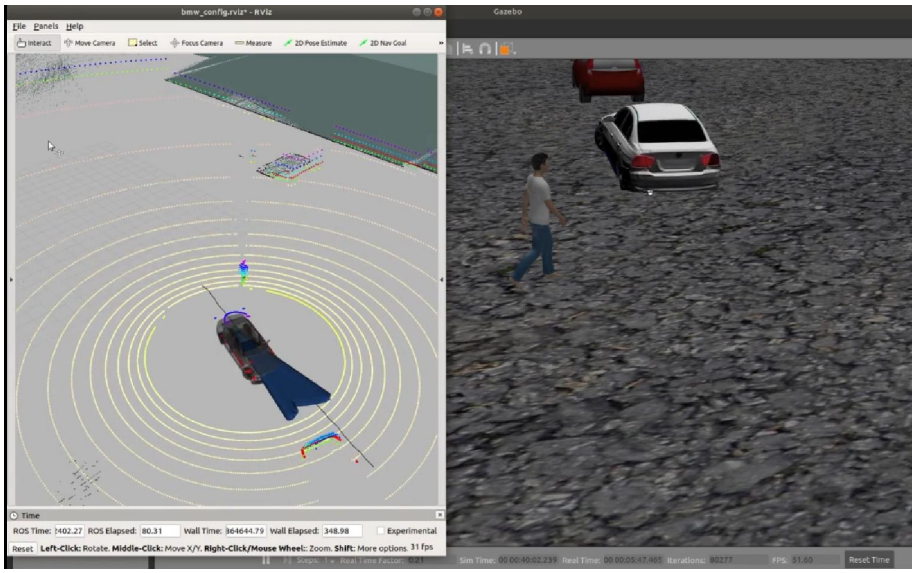


Figure 24 – DWA on ROS Navigation Stack

Source: Own Authorship

3.6.3 Time Elastic Band (TEB)

The Time Elastic Band (TEB) is a local planner that creates a sequence of intermediate vehicle poses that modifies the initial global plan (Rösmann et al., 2013). These poses are

given by:

$$s_{\mathrm{i}} = (x_{\mathrm{i}}, y_{\mathrm{i}}, \theta_{\mathrm{i}})^T \tag{1}$$

$\mathrm{s}_i \in R$ x S^1

These poses configurations, robot position (x_i, y_i) and orientation β are denoted in a global frame (map). The TEB augments this representation by incorporating the time intervals between two consecutive configurations, resulting in a sequence of n-1 time intervals Δ T_i:

Each time interval denotes the time the robot requires to transit from the current configuration to the following format in the sequence Q. The TEB is defined as a tuple of both sequences:

$$B := (Q, \tau) \tag{2}$$

The algorithm requires the vehicle's velocity and acceleration limits, the security distance of the obstacles, and the geometric, kinematic, and dynamic constraints of the car. All of this configuration generates a set of commands for speed(v) and steering angle (δ) required to achieve the intermediate waypoints while the vehicle is moving, once:

$$cmd_i = (v_i, \delta_i)^T \tag{3}$$

According to Marin-Plaza et al. (2018) the drawback of this method is the influence of the dynamic obstacle's direction in the generation of the recalculated local path.

An overview of The TEB sequences of configurations in a global map can be visualized below:

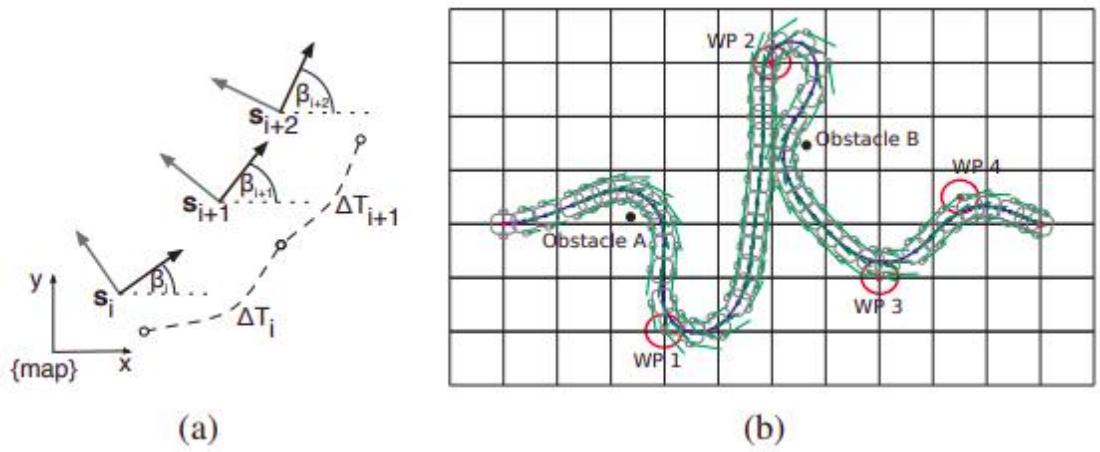


Figure 25 – (a) TEB: sequences of structures and time differences and (b) Large scenario with consideration of waypoints and obstacles Rösmann et al. (2013)

The Figure 26 shows the TEB algorithm simulated with one of the car-like-robots used in this work:

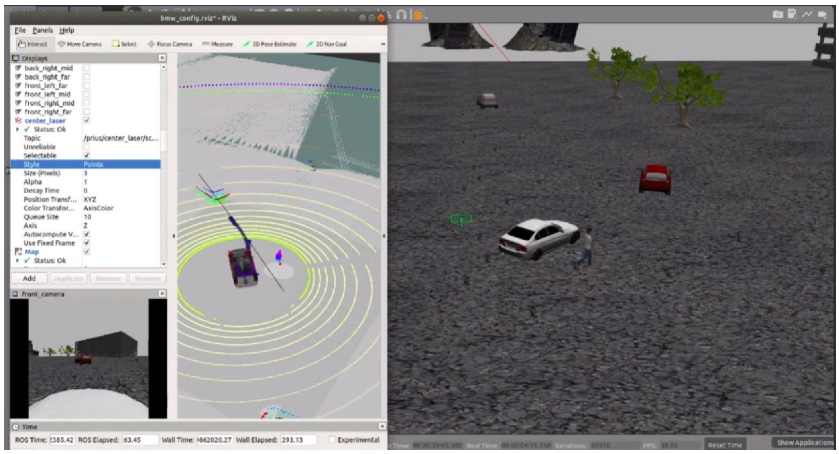


Figure 26 – TEB on ROS Navigation Stack

Source: Own Authorship Moll et al. (2015)

3.6.4 RRT

Some algorithms derived from RRT can be used as local planners (Tian et al., 2007). A Particular RRT variation, called HSL-RRT*, implemented by Le et al. (2019) proved to be a robust algorithm to find a viable path in complex environments. This RRT* optimization method uses the RRT* algorithm in SL coordinate using historical planning data. His article demonstrated the HSL-RRT* effectiveness against classical algorithms, such as Lattice (see subsection 5.2.2.2) in enabling the AV's navigation through many obstacles disposed of sequentially arranged in different highway lanes. The behavior of this algorithm is demonstrated in Figure 27.

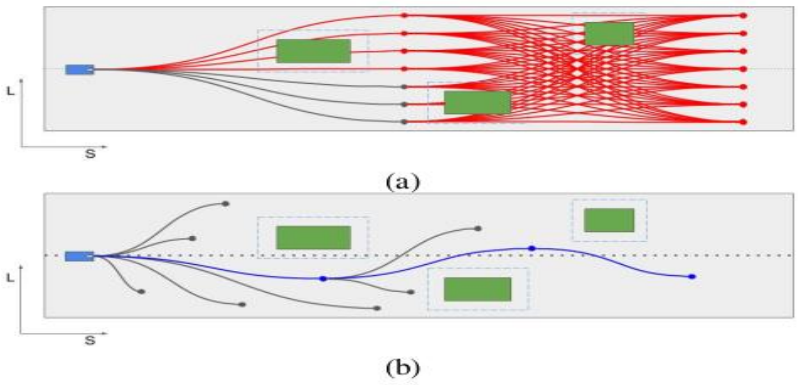


Figure 27 – a) Lattice path planner failing to generate a path. Effectiveness of HSL-RRT* in finding the feasible path on the highway, avoiding multiple obstacles (b). Source: Le et al. (2019)

A presentation of ROS framework explanation in the self-driving car context, as well as the simulations, were done with the Navigation Stack can be found on GitHub:

Master_Thesis_Simulations.

3.7 The ROS Planning Libraries Integration

The planning libraries and some helpful software depicted in this current section are essential to better understanding the planner's functionality and implementation. However, it only comes to practical reality in robotics when applied to a robot. ROS becomes the more feasible and widely used framework for these planners' application, observation, and improvement. In path planning, the robot kinematic model is of extreme relevance. A path planner is often implemented to fit a particular robot shape and handle specific constraints.

3.7.1 Kinematic Models

The kinematic model from a robot defines its motion behavior. The kinematics of a robot can be of two different types: Holonomic or Nonholonomic.

A holonomic robot can move forward, backward, and translate left or right. It also can perform rotations around the Z-axis (CCW or CW). The holonomic robots can slide sideways while the wheel drives forward or backward without slipping in that direction. A nonholonomic robot, however, cannot move directly sideways (Zegers, 2020). The most habitual nonholonomic robots are described below.

- Unicycle Robot: This is the most basic nonholonomic robot. It is based on a single wheel, which moves at the desired velocity $\mathcal{V}$ and with a specific heading ϕ;
- Differential Drive robot: A robot of this type contains two independently driven wheels that rotate about the same axis.
- Car-Like-robot consists of two steered front wheels and two fixed-heading rear wheels. To prevent slipping of the front wheels, they are driven using Ackermann steering (Krishna, 2020). The center of rotation of the car's chassis lies on the line passing through the rear wheels at the intersection with the perpendicular bisectors of the front wheels.

In this dissertation, the focus is the car-like robot. In this robot, only the front wheels can turn. A representation of this system is in Figure.

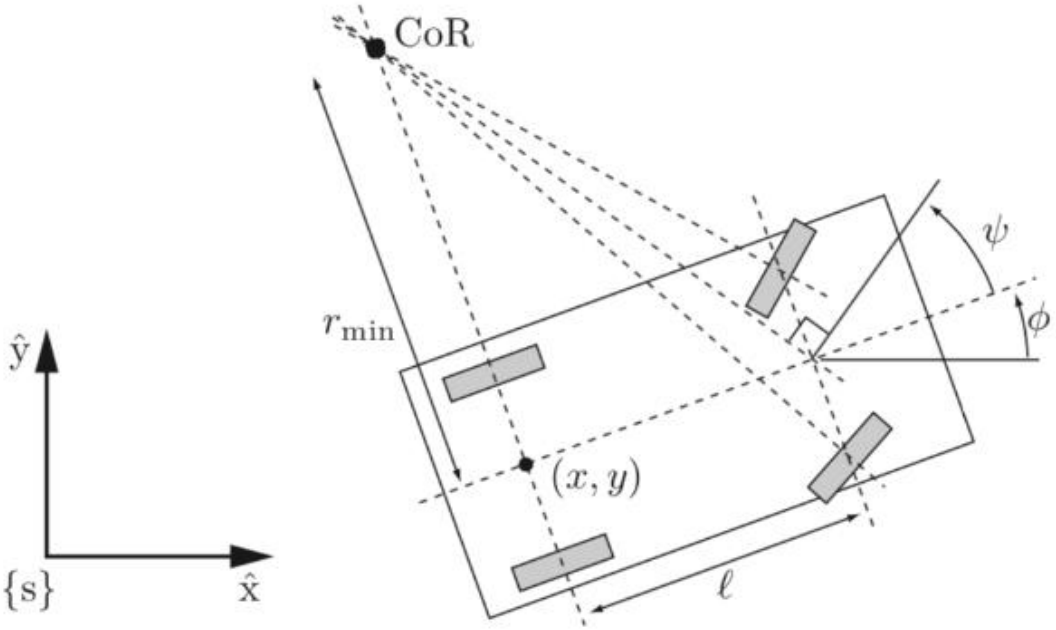


Figure 28 – Kinematic of a car-like-robot. Source: Zegers (2020)

The ψ angle represents the front wheels' angle (steering angle), and ϕ angle represents the steering control. The equations that model this system are:

$$\dot{x} = vcos(\phi + \psi) \tag{4}$$

$$\dot{y} = vsin(\phi + \psi) \tag{5}$$

$$\dot{\phi} = v/lsin(\psi) \tag{6}$$

$$\dot{\psi} = \sigma \tag{7}$$

And the steering angle is given by:

$$\psi = arcsin(wl/v) \tag{8}$$

Where l is the distance between the back and front wheels.

The ROS integration with the planning libraries and the kinematic models they support is necessary to describe better the ROS framework composition, which is found in the chapter 4.

4 The Robot Operating System (ROS)

Lately, robotics has been attracting non-roboticist experts to generate and upgrade new robot functionalities provided by robotics branches, such as AI, Machine Learning, and others. The problem is that heterogeneous systems obstruct constant development. Different programming languages, operational systems, and hardware are being adopted to address these complex functionalities.

To mitigate these issues, uniform the whole development structure for robot programming, and enable the reusing of specific functionalities already implemented, the Willow Garage generated the Open-Source framework called ROS. The words "OS" from ROS means a middleware that runs on Unix-based operational systems (Nayoga University, 2016).

In this way, the robot's developers can combine software developed by other ROS developers to optimize a project, reducing the implementation time and focusing just on the implementation on the specific area the roboticist wish to work on.

4.1 The ROS Framework

The ROS is a trending robot application development platform that provides numerous features such as message passing, distributed computing, code reusing, etc. The ROS project began in 2007 with the name *Switchyard* by Morgan Quigley as part of the Stanford STAIR robot project. The ROS community and the number of ROS developers worldwide have grown more and more. Most of the high-end robotics companies are porting their software to ROS. A trend noted in industrial robotics, once the companies are also migrating from proprietary robotic applications to ROS (Joseph, 2015). The ROS framework is based on nodes. The multiple nodes can publish information such as sensors reading, motor commands, or planners ordering in a topic, and other nodes can simultaneously subscribe to this information through the ROS master.

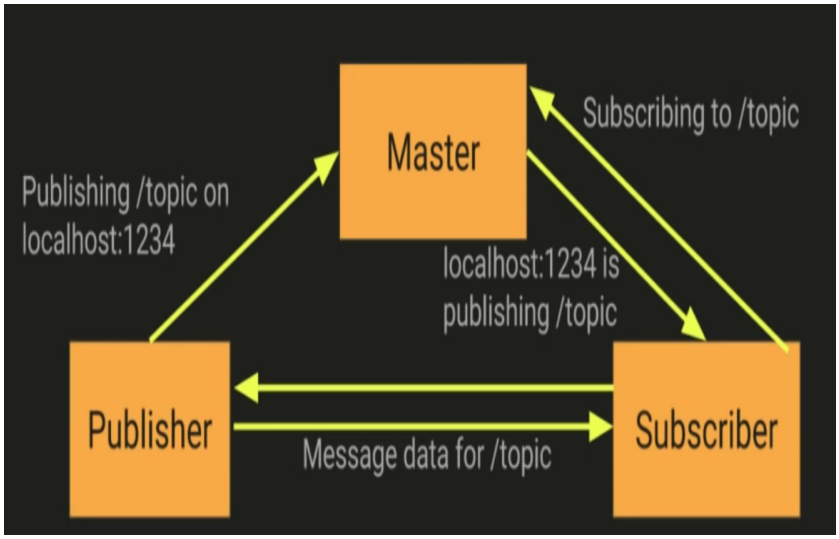


Figure 29 – ROS communication framework. Source: Huang (2019)

ROS is a framework for integration. ROS presents high-end capabilities such as SLAM and Adaptive Monte Carlo Localization(AMCL) packages, which can be used to perform AN. ROS contains the MoveIt package for motion planning of robot manipulators and numerous other capabilities stored in the called packages. ROS also presents multiple tools for debugging language codes to visualize the nodes that are working and how they are integrated. ROS supports high-end sensors and actuators, such as LIDAR and Dynamixel servos. The programming code language also is interchangeable in ROS, and it was generated to work with inter-platform operability, supporting, for example, C++, Python, or Java. One of the best features of ROS is the concurrent resource handling and its modularity. These features allow the robot to keep working if some issue happens and some specific node crashes. For example, if the node related to the robot's right arm fails, the other sensors keep working. In other words, the robot will not completely stop because of a punctual problem.

More detailed description of the content of the ROS packages and how they work-integrated can be found in (Joseph, 2015).

The Robot Operating System is wrapped by nodes and packages, which communicate. Each node performs a task, and the aggregation of studies defines a function. Therefore a certain number of nodes forms a package that can get (subscribe) and send (publish) data to another node through specific transmission channels: topics, services, actions. These different sorts of communication, the ROS components as well as the main ROS features are described below:

[0] Available in:
https://www.autoblog.com/2018/11/28/audi-demonstrates-pop-up-next-air-taxi/slide-1301558/

- Package: A package contains ROS nodes, a ROS independent library, a data-set, configuration files, or anything that logically constitutes a useful module.
- Node: The ROS node is the unit that handles a robot part. It can be an image processing program, a joint controller program or anything else which addresses the robot's function (Robotics Back-end, 2018).

An illustrative example of ROS nodes and packages structure is depicted below.

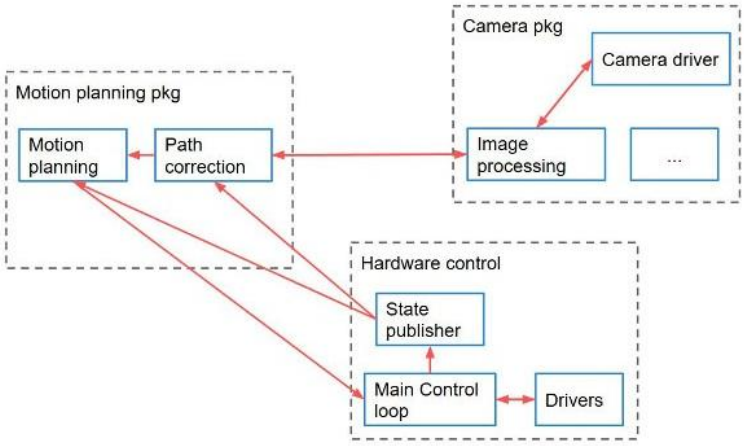


Figure 30 – ROS Packages and Nodes Structure. Source: Robotics Back-end (2018)

The ROS nodes are responsible for processing specific functions in the particular packages (Camera, Motion Planning, and Hardware Control, in the example above).

In the communication aspect, ROS uses

- Topic: ROS topic is the pipeline in which the ROS nodes communicate with each other. It is the spot where the ROS messages flow. A Node can subscribe and publish to a Topic. The node that subscribed info from a topic will get this info infinitely until the user's input command or coding interrupts this subscription. This kind of information is asynchronous.
- Service: ROS services are synchronous communication. In this type of transmission, the ROS service client requests a node and information at a specific time, and the ROS service server replies to this request sending the requested information.
- Action: ROS action is a more sophisticated communication. In this process, the ROS action client requests info from a ROS node, this node (the action server) confirms the receipt from this request through a feedback message. Differently from service, during this process, the action client can cancel the order(request), and once in a while the, the action server sends the status from the request (which step of the

program it is realizing). The ROS node client can do other things during this period and is not "frozen" waiting for this message. Finally, when the ROS action server finishes processing the program, it returns a response message (Aderinola, 2020).

ROS contains Powerful tools, which are worthy of describing:

- Rviz: It is the ROS viewer. A 3D visualizer can display sensor data and state information from ROS. It converts raw data from topics to visible human data through standard or customized plugins.
- RQT: It is an assistant in which the user can manage the active ROS topics, debugging the messages transmitted separately. It can also check the robot transforms, visualize rosbag info, display topics data on dynamic graphics, etc.

ROS also contains numerous valuable tools, such as the rosbag, which can record topics' data in a compressed ".bag" file. This feature is exciting and with the assistance of other ROS tools, such as PlotJuggler (Faconti, 2010), the user can plot, visualize and debug different data from a robot's mission. ROS is very extensive and more info can be taken in (The Construct, 2020; ROS Wiki, 2020).

Gazebo was developed by professor Dr. Andrew Howard and his student Nate Koenig. Gazebo was integrated into ROS in 2009 and is widely used by design roboticists to test algorithms rapidly, design robots, perform regression testing, and train AI systems using realistic scenarios. When a robot is not available to test the robot's ROS directly with it, ROS also has easy integration with the Gazebo simulator.

This multi-robot simulator contains numerous valuable features such as:

- Dynamics Simulations: The user can access multiple high-performance physics engines, including ODE, Bullet, Somebody, and DART;
- Advanced 3D Graphics: Using the OGRE, Gazebo provides realistic rendering of environments, including high-quality lighting, shadows, and textures.
- Sensors and Noise: Generate sensor data, optionally with noise, from laser range finders, 2D/3D cameras, Kinect style sensors, contact sensors, force-torque, and others.
- Plugins: It contains developed plugins designed for a robot, sensor, and environmental control. The plugins provide direct access to Gazebo's APIs.
- Robot Models: Gazebo provides numerous robots, including PR2, Pioneer2 DX, iRobot Create, and TurtleBot. It is also possible to build the own personalized robot using SDF format.

- TCP/IP Transport: Allow to run the simulation on remote servers and interface to Gazebo through socket-based message passing using Google Protobufs.
- Cloud Simulation: It is possible to use cloudSim on Amazon AWS and GzWeb to interact with the simulation through a browser.
- Command Line tools: Extensive command-line tools are provided to facilitate simulation introspection and control.

4.2 ROS AD Driving Application Timeline

With a better comprehension of the planning libraries (see section 3.3) and the ROS framework (see section 4.1), this section will give a brief overview of the ROS-based applications (software) implemented to generate collision-free trajectories. They will be delighted in a timeline according to the advancement of this project.

4.2.1 ROS1 based Frameworks

4.2.1.1 MoveIt

MoveIt is a ROS-based software develop to address industrial robots application's tasks, such as pick and place, assembly, and grasping tasks (MoveIt Applications, 2021). MoveIt provides a friendship interface that enables the roboticist to define the robot's links, joints, control, and other features.

Although MoveIt is widely employed for industrial arms application, its use can also be adapted and extended to holonomic robots in 3D environments as done on (Youakim et al., 2017). MoveIt is designed to work with many different types of planners from:

- Open Motion Planning Library (OMPL)
- Pilz Industrial Motion Planner
- Stochastic Trajectory Optimization for Motion Planning (STOMP)
- Search-Based Planning Library (SBPL)
- Covariant Hamiltonian Optimization for Motion Planning (CHOMP)

This massive amount of available planners makes MoveIt ideal for benchmarking analysis. (MoveIt Planners, 2021)

In principle, this application was thought to achieve autonomous driving simulations, taking advantage of the complete and robust Benchmark this framework provides. MoveIt has implemented the Planner Arena Benchmark functionality, which allows the user to

assess the planners' behavior compared to other planners in several relevant aspects such as:

- Time
- Solution Difference
- Solution Length
- Solution Segments
- Solution Smoothness
- Solution Clearance
- Correct Solution
- Number of Iterations
- Memory
- Others

These parameters are relevant in robotics and computer science since they can help the roboticist determine the most suitable algorithms for a specific application.

However, during software experimentation, it was recognized the impossibility of using it for simulations involving nonholonomic robots nor to simulate complex environments required to study autonomous vehicles. The main drawbacks which avoided MoveIt application for this research were:

- MoveIt was designed to emulate the robot's three-dimensional displacements, not bidimensional, which is the case for autonomous vehicles.
- MoveIt do not have available an AV kinematic model
- MoveIt is not able to emulate a nonholonomic robot ROS Answers -MoveIt (2020)
- MoveIt is not able to simulate the static or dynamic contact between the wheels and the asphalt ROS Answers (2020)

4.2.1.2 Navigation Stack

The Navigation Stack was first implemented to address industrial robots displacement. ROS is responsible for getting robot data, such as odometry and sensor streams information, processing it, and sending the output commands to the robot's mobile base to enable navigation. Therefore the first Navigation Stacks path planners and sensors available were in the beginning functional just for holonomic robots (See section 3.7.1). An overview of the Navigation Stack Setup is shown in the Figure 31.

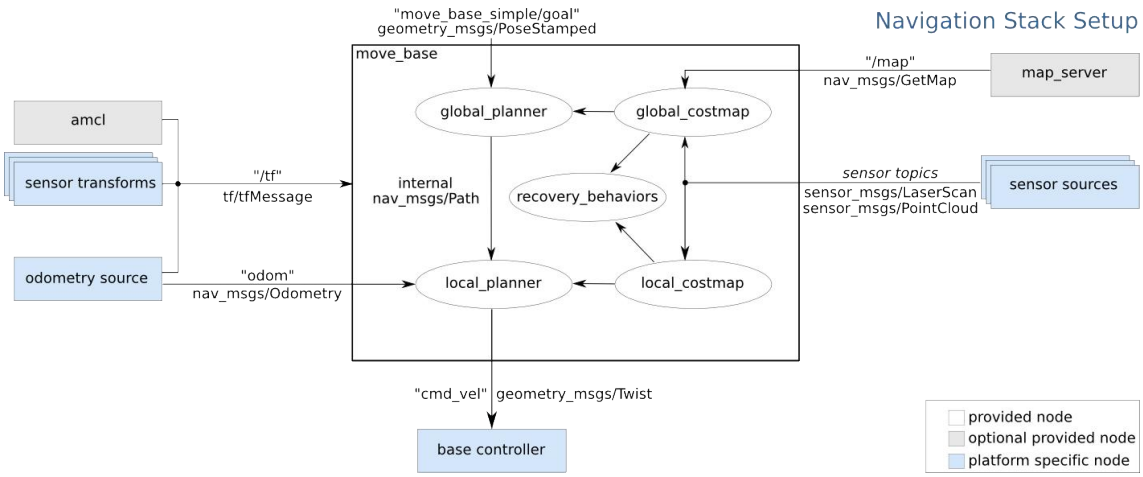


Figure 31 – Navigation Stack Setup

Source: ROS Wiki Navigation (2018)

Even though the Navigation Stack is a relatively robust ROS-based robot navigation software, it is not explicitly developed for AV. In addition, in contrast with MoveIt, the Navigation Stack does not provide a Benchmark tool with ready-to-use metrics for an accurate path planners investigation.

In a recent development of navigation stack, it was enabled path planners for car-like robots. The Dortmund University implemented the Time Elastic Band (TEB) path planner, considering the vehicle's nonholonomic behavior and the Ackerman constraints (Macenski et al., 2020). This platform was tested for the AD simulations: the standard ROS path planners (Globals and locals) or the TEB.

The simulation results are described below:

- Eband: The Elastic Band local planner computes an elastic band within the local costmap, and attempts to follow the path generated by connecting center points of the band using various heuristics. The Figure 22 displays the simulation using this planner in a car-like robot.
- RRT: The RRT (See section: 3.5.4) was applied in the car-like robot using navigation

stack only as a global planner. Unfortunately, at the moment of this research, any available RRT-based path planner was known as a local planner.

- DWA: The Dynamic Window Approach algorithm (See section 3.6.2) presented an excellent lateral and longitudinal move, compared to Eband. However, it was not able to avoid the obstacle.
- TEB: The Time Elast Band algorithm, as described

The usage of ROS1 within the Gazebo simulator was essential for a better understanding and code familiarization with different modules. However, the ROS-based frameworks were limited to emulate real and extensive driving scenarios sensors and didn't provide a path planner suitable for robots with nonholonomic constraints. For these reasons, experts from the automotive industry in integration with the ROS community and university researchers created the Autonomous Driving Stacks. These complex frameworks have proven to be more robust for simulation and integration with actual vehicles.

4.3 The next stage: Autonomous Driving Stacks

Due to the limitations detected regarding vehicle planning and control and the simulator simplicity, the research groups started to implement ROS-based applications to address automated driving functions. These applications are known as Autonomous Driving Stacks. The ADS comprises numerous algorithms, packages, and modules that work-integrated will address specific driving tasks on specific scenarios. A sketch of the architecture can be visualized in Figure 32:

An Autonomous Driving Stack is composed of multiple layers (stacks):

- Hardware Stack: Consist of the highest level of vehicle, such as sensors, actuators, electronics, etc
- Off Board Software and Data: Composed by modules responsible for processing specific pieces of information, such as mapping (creation, update, or distribution), data recorder (in AV's recording data it's mandatory, it works as black-box) and so on;
- Methodologies: Part handles testing, integration, security, and validation processes, such as HIL, SIL.
- On-Board Software: The lowest level of the AV, composed of the algorithms and modules, such as localization, perception, planning, and control.

[0]HIL: Hardware in the Loop, SIL: Software in The Loop

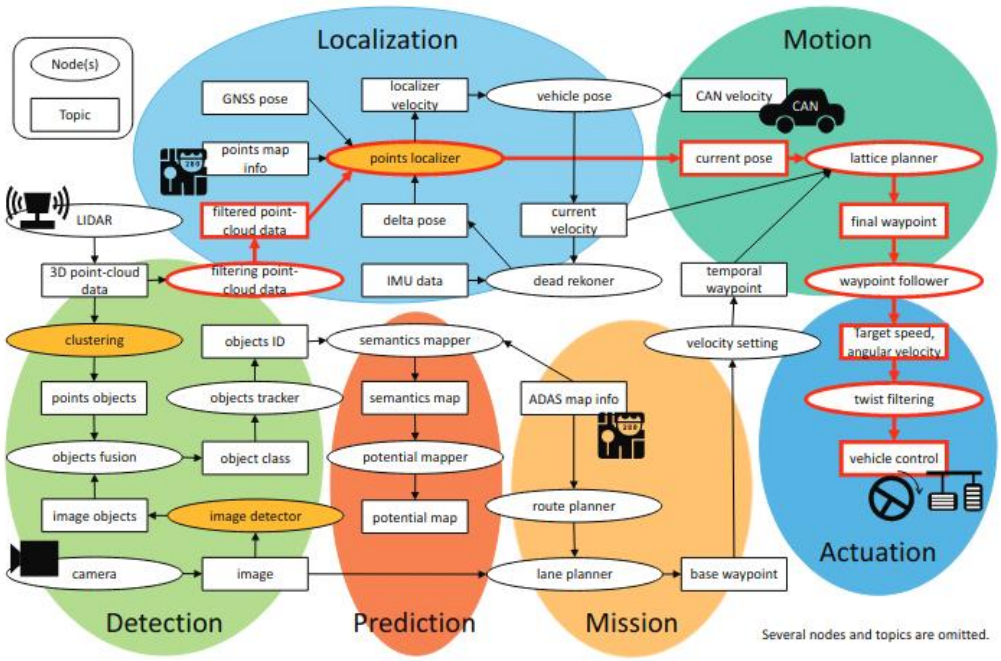


Figure 32 – AD Stack Configuration. Source: Kato et al. (2018)

- Product: The integration from the previous layers designed for the user applications.

In a nutshell, an autonomous driving stack architecture is composed of sense, plan and act layers. A scheme of the interaction between these layers is shown in Figure 33.

Figure 33 – AV High Level Architecture. Source: Whitley (2021)

In addition, for an automation level 3, a monitoring system is needed to observe the driver conditions; once in this level, the driver must take the direction if some issue happens with the software or hardware.

Autonomous driving, the sensors have a close relationship with the automated driving functions, as shown in Figure 34.

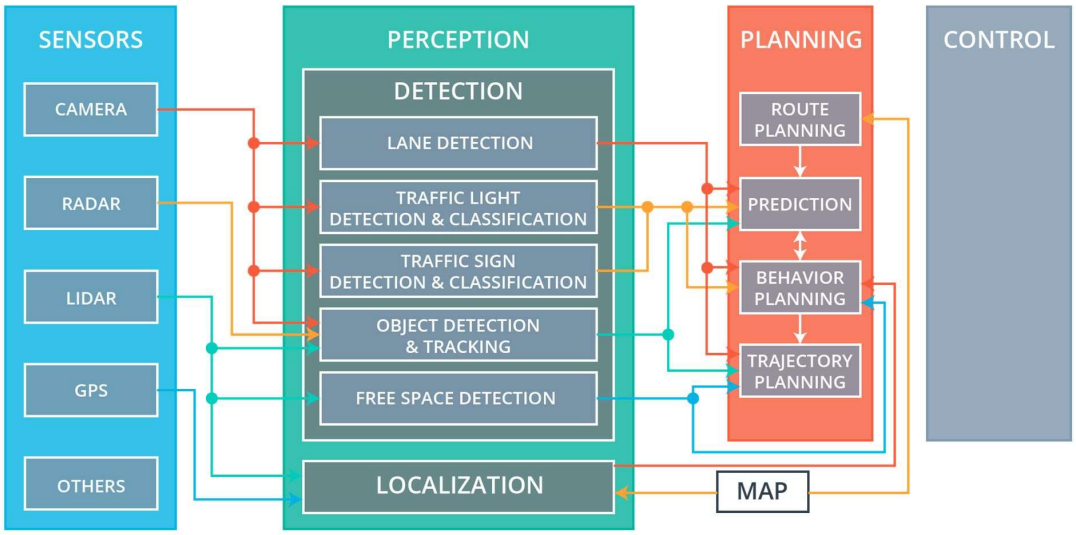


Figure 34 – AV Signal Flow. Source: Whitley (2021)

Figure 34 makes it possible to check how the camera is directly linked to the Lane Detection. Therefore, it directly influences the lateral vehicle control in the trajectory planning module. Similarly, the GPS sensor is directly connected to the localization module, which impacts route planning, behavior planning, and as a result, trajectory planning.

In Autonomous driving level 4 and level 5, the driver becomes a simple passenger, and the interaction of the AD stack layers suffers particular modifications, especially in the planning layer. This layer is now composed of the Strategic, Tactical, and Reactive levels.

These different levels on the planning layer require other times to vehicle action and consequently affect the control layer.

At the operational level, the vehicle active safety systems operate, such as automatic emergency brake, for example. The vehicle navigation system works on the mission level, and it has an extended period to act. However, the Reactive Control must act in a period of milliseconds to avoid a collision with an unexpected obstacle that entries the vehicle's front part, for example.

A self-driving car must handle different system critical levels. The AV needs to address

a failure in task execution from the low-critical system level. Still, it is not dangerous to the user, to the safety-critical system level, which can lead the human being to injury or death, usually those involved with the need of steering the vehicle in hard-real-time.

5 Autonomous Driving Stacks

5.1 Autoware

From the breakthrough of Dortmund group research, new groups worldwide studied and kept the Navigation Stack development for autonomous driving. Nav2 of Steve Macenski is an upgraded example of some of the Navigation Stack features (Macenski et al., 2020). However, many limitations were detected in Navigation Stack, mainly regarding the absolute scenario loyalty that this platform along Gazebo Simulator could offer and the previously path planners limitation described in Section 4.2.1.2. To overcome these limits, Shinpei Kato at Nayoga University started the Autoware project in 2015. The idea was to develop an "All-in-One" open source software for autonomous driving technology. Autoware is an open-source software based on ROS. It can be pushed on Github for autonomous driving research and development. Autoware wraps up standard autonomous driving modules, allowing 3-D map generation, localization, perception, planning, and control functionalities. An overview of the first Autoware arrangement is shown in Figure 35.

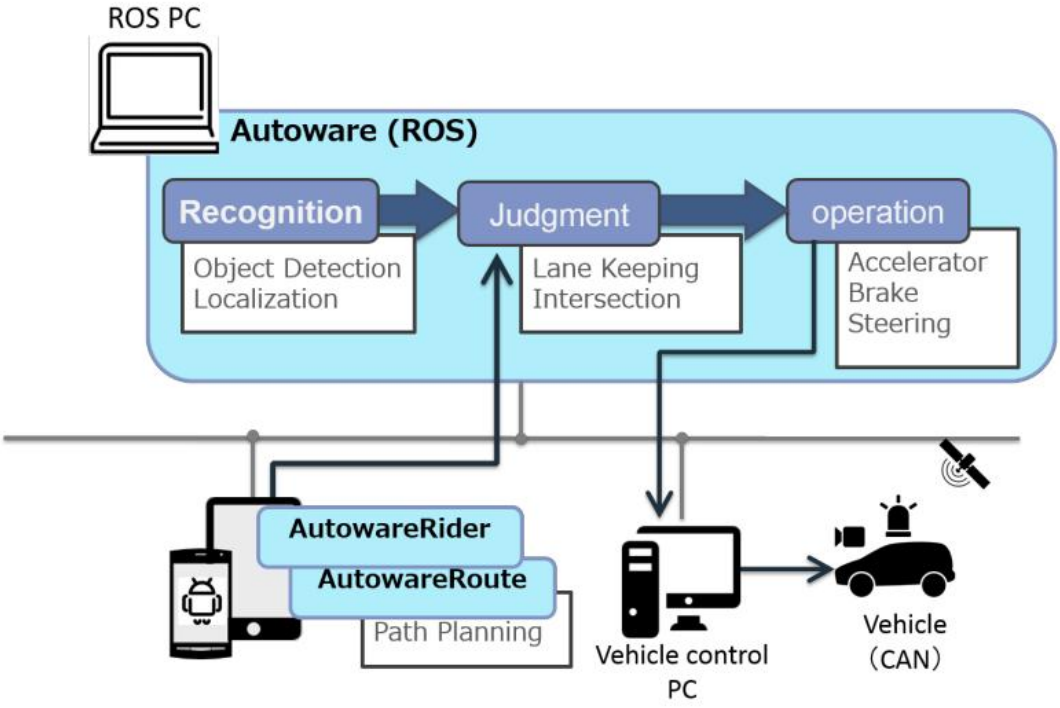


Figure 35 – Autoware Overview. Source: Nayoga University (2016)

Detecting the project potential, the Autoware Foundation was launched in 2018. This group started as a non-profit organization open-source, composed of relevant academic, industry, and government members, as shown in Figure 36.

Figure 36 – Autoware Foundation Members. Source: Nayoga University (2016)

5.2 Autoware.AI

As described previously in the section 5.1, Autoware.AI is based on the first Autoware project, inherited many of its functionalities, improved, and created others. Autoware.AI provides a friendly user interface called Runtime Manager, in which the user can select different algorithms and launch the packages related to the autonomous driving modules. There are numerous distinct packages available from mapping, localization, detection, prediction to be chosen, and different planners and driving strategies for the mission planning, motion planning, and decision-making modules, respectively. Figure 37 shows an overview of the AD modules and the available packages to integrate them into the runtime manager interface.

Autoware.AI can detect surrounding objects, such as pedestrians, vehicles, traffic lights, etc. Autoware.AI uses Light Detection and Ranging (LIDAR), and on-vehicle cameras localize the ego-car position. The vehicle's localization is done through the Global Navigation Satellite System (GNSS) or by use of the Normal Distribution Transform (NDT), a scan-based matching strategy that uses the 3-D map of Point Cloud Data (PCD) format and LIDAR data (Nayoga University, 2016). Afterward, making judgments to drive/stop at lanes or intersections and the driving assistance and safety diagnosis support are performed with an embedded multi-core CPU.

An overview of Autoware.AI modules integration is shown in Figure 38.

5.2.1 Planning Module

The planning module in Autoware.AI is set in 2 sub-modules:

- **Mission Planning**: Given the destination, is the global trajectory generated based

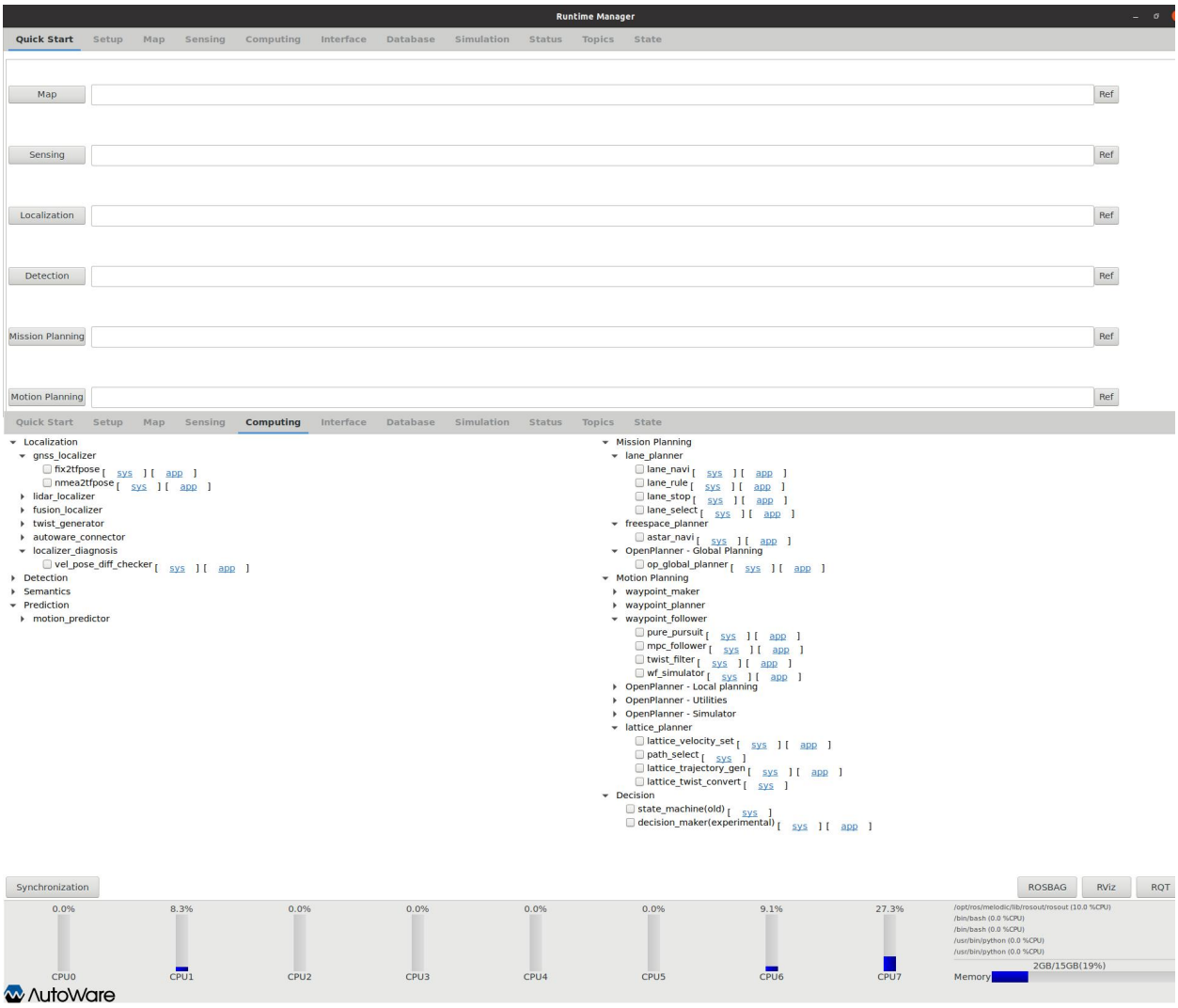


Figure 37 – Runtime Manager Tabs: QuickStart at the top, and Computing in the bottom

Source: Own Authorship

on AV's current position. The mission planning uses a rule-based system to determine the path courses, in which it considers the driving states, such as lane changes, merges, and passing. This sub-module needs to be fed with a high-definition 3D map, which contains information from the static road features and is used to calculate the global path. The basic policy of the mission planner is to follow the center lines of the lanes over the route generated (Kato et al., 2018).

- **Motion Planning**: The motion planner is in charge of generating feasible local trajectories along with the previously given global path, considering the vehicle states, the drivable area indicated by the 3D map, the surrounding objects, the

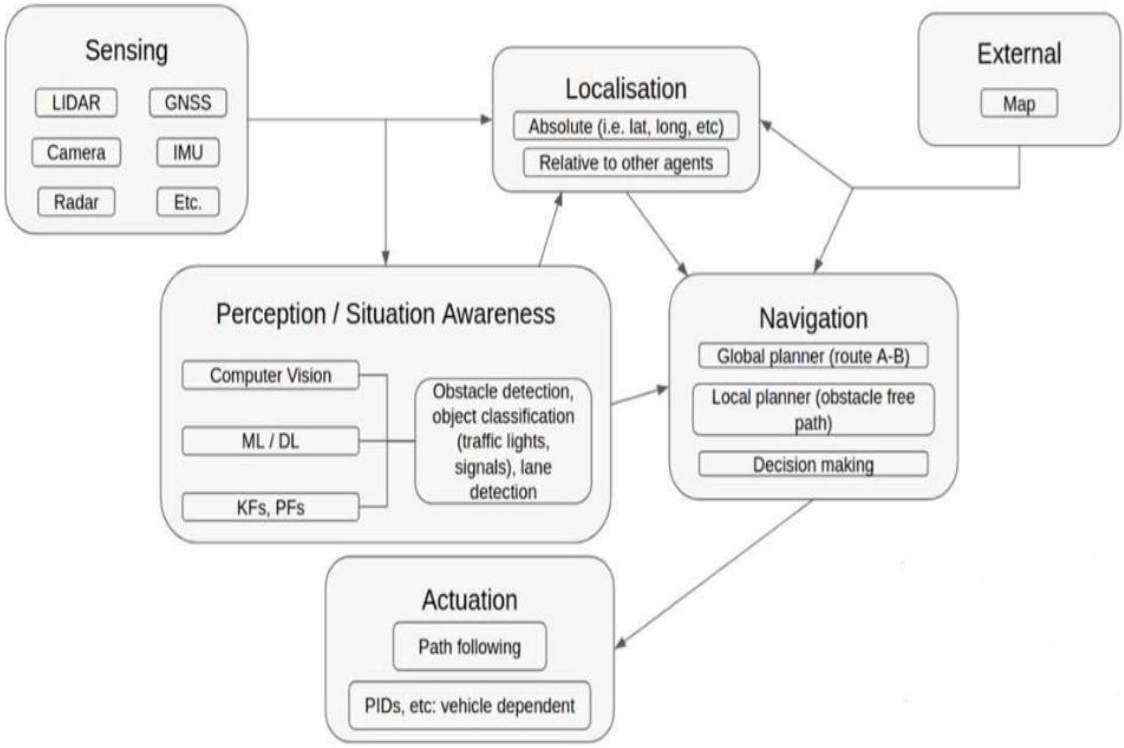


Figure 38 – Autoware.AI architecure. Source: Nayoga University (2016)

traffic rules, and the desired goal.

The tested mission plannings and motion plannings applied in this research are outlined in sections 5.2.2, 5.2.3 and 5.2.2.1, 5.2.2.2, 5.2.3.2 respectively.

5.2.2 Guideline-based A-Star (Freespace)

The guideline-based A-Star algorithm optimizes the classic A* algorithm (see section 3.5.3) for the autonomous land vehicle (ALV). To express the driver's intention, the algorithm applies the following heuristic function:

$$F(i) = G(i) + H1(i) \times \alpha 1 + H2(i) \times \alpha 2 \tag{9}$$

Figure 39 represents the physical meaning of the expression above. As the classical A*, G means the cost to reach the final waypoint (from initial position), H means an estimated value from Av's current position to the target (last waypoint). In this A* optimization, H was split into two components: H1 and H2. The H1 component corresponds to the distance between point i and the guideline, and H2 is the distance from g(i) to the target, α_1 and α_2 are coefficients applied to penalize the AV's move in specific directions, increasing or decreasing the weight from the H1i or H2i costs (Erke et al., 2020).

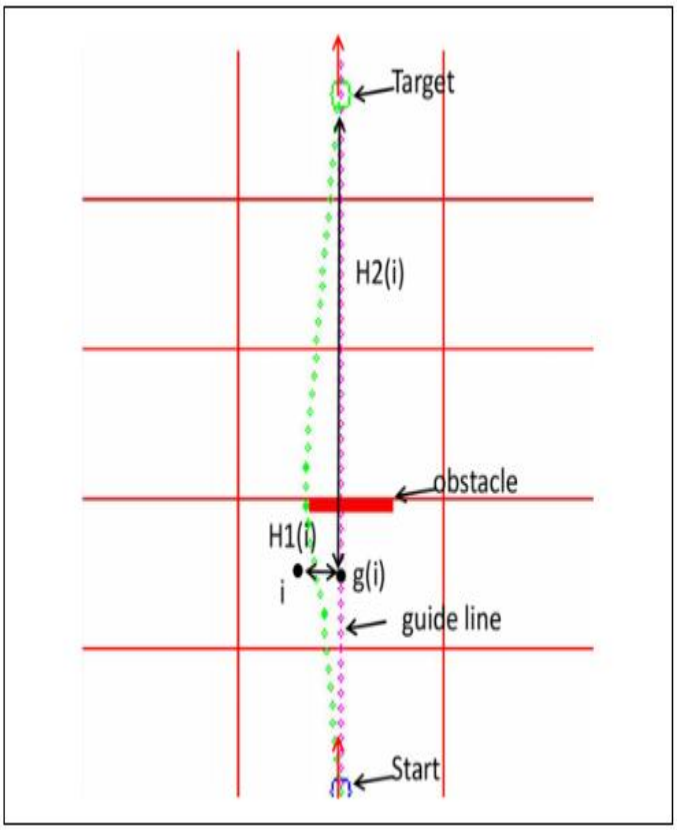


Figure 39 – Guideline-based A-Star working principle. Source: Erke et al. (2020)

5.2.2.1 Hybrid A* (Local Planner)

The Hybrid A* is a modification of the classical A* algorithm. This adjustment aims to create a kinematically feasible path that satisfies the vehicle's non-holonomic constraints. In Hybrid A* implementation, a 4D search space (x,y, θ, r), where the fourth dimension (r) represents the current direction of motion (forward or reverse). The possible drivable directions are used to apply penalties on the path-cost function if the length of the segments driven falls on the reverse or an abrupt change in the robot's orientation. The classical A* and the Hybrid A* algorithms search graphs are shown in Figure 40.

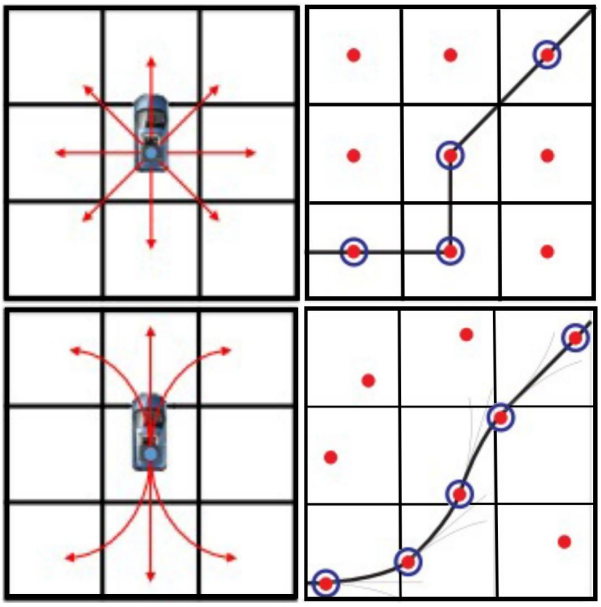


Figure 40 – Graphical Comparison of search algorithms: Searching directions of the original A* (Left-Up) and Hybrid A* (Left-Bottom). Graph Expansion: A* visit states following centers of cells (Right-up), Hybrid A* visit states following a continuous state in each cell (Right-Bottom). Source: Dolgov et al. (2010); Tu et al. (2019).

As we can see in Figure 40, the Hybrid A* algorithm expands the nodes in a continuous kinematically feasible trajectory. This model guarantees a drivable path.

The Hybrid A* uses two phases to the path generation:

- **Non-holonomic-without-obstacles**: Ignore obstacles, just take into account the non-holonomic nature of the car. This heuristic computes the shortest path to the goal from every point in the 4D space (x, y, θ, r) in some discretized neighborhoods of the goal. A positive effect is assigning high costs to search branches that approach the target with the wrong heading (Dolgov et al., 2010);
- **Holonomic-with-obstacles**: This heuristic ignores the vehicle non-holonomic constraint but uses the obstacle map to compute the shortest distance to the goal by performing dynamic programming (DP) in 2D (Dolgov et al., 2010).

In summary, the first heuristic consists of a heuristic search in continuous coordinates, producing drivable trajectories that lie in a neighborhood of the global optimum. The second uses numerical optimization in constant coordinates to improve the solution quality locally. As reported by Dolgov et al. (2010) The usage of these heuristics reduce the number of expanded nodes by about 45% compared to the Euclidian 2D distance usage.

The heuristics above describe the forward search expansion using a discretized control actions (steering) space. This means that the search will not find an actual continuous-coordinate goal state, depending on an analytic expansion. The Hybrid-A* used the Reed-Sheep model, which leads to noticeable gains in replanning speed compared to regular forward node expansions. At that phase, the Hybrid-A* local path can still be sub-optimal and contain unnatural swerves, requiring excessive steering. Therefore, a fourth step is needed: Trajectory Optimization. To accomplish that, a non-linear optimization program on the coordinate of the path vertices is applied, improving the length and smoothness of the solution. The optimization the use of conjugate-gradient (CG) descent (Dolgov et al., 2010). The third and fourth described steps can be visually checked below.

Figure 41 – Analytic Reed Sheep Expansion (Up). Trajectory Smoothing process (bottom)

Source: Dolgov et al. (2010)

The integration of Freespace global planner and Hybrid A* local planner with Autoware.AI for the path generation is demonstrated in Appendix A.1.

5.2.2.2 Modified Lattice

The classical state lattice (Pivtoraiko et al., 2009) is a method for inducing a discrete graph search on a continuous state space while respecting differential constraints. State lattice was applied in 2007 DARPA Urban Challenge to handle with parking maneuvers (Likhachev and Ferguson, 2009). Lattice methods are most applicable to unstructured environments, and the original lattice formulation is not readily applicable to on-road driving scenarios. The modified Lattice (Autoware Lattice planner) adapts from the classical Lattice. In contrast with the classical, it plans a dynamic feasible motion in structured environments, such as roads with moving traffic, by conforming the Lattice to the environment. Figure 42 displays the modified Lattice adapted for structured environments.

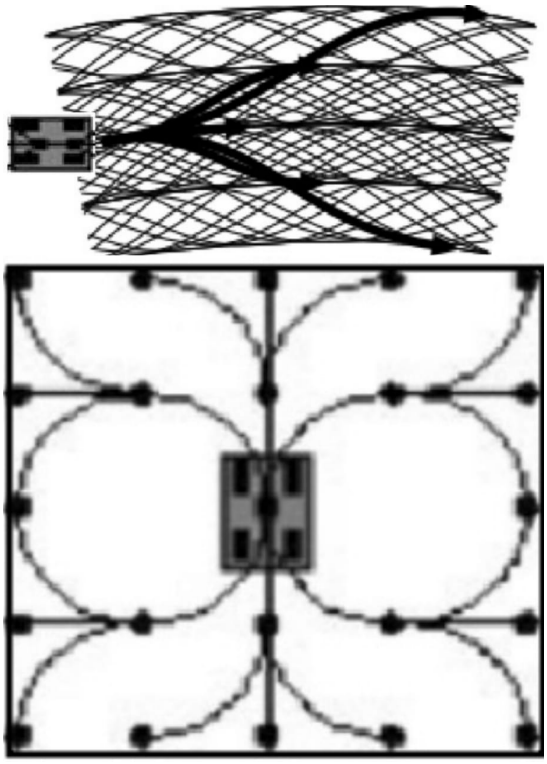


Figure 42 – Bottom:Regular state Lattice in unstructured environment. Up: State Lattice conformed to a structured environment).Source: Pivtoraiko et al. (2009); McNaughton et al. (2011)

This modified Lattice was implemented to assume more responsibility in trajectory persecution. This means that the lattice planner is more independent from the global planner path. This method was implemented to cover mismatches between the planners experienced by the authors, especially when higher-level planners give commands to lower-level planners that are unfeasible (McNaughton et al., 2011).

The modified Lattice samples many steering actions that take the vehicle away from the lane centerline and then back towards it. As stated by McNaughton et al. (2011):

> [...] Our lattice is constructed around a lane center line defined as a sampled function [x(s) y(s) $\theta(s)$ k(s)] where s is the arc length s, and the points p(s,l) defines the lateral offset (l) from the road to the center line. k(s) is the curvature of the path, such that $\theta(s,l) = \int_0^d k(s,l)ds$
>
> [...] To construct our lattice we define a discrete grid points to station and latitude using a simple linear multiplication s(i) = $a_s i$, l(j) = $a_l + b_l(j)$, so that the station is monotonic increasing starting from zero and moving right in the coordinate frame. The latitude may be positive(left) or negative (right) w.r.t the centerline. A plan through this structure is a sequence of smooth lateral shifts joined with Lattice poses aligned to the road[...]
>
> [...] The path given by Lattice is defined as a cubic polynomial spiral, which means that the curvature of the path is a cubic polynomial function of arc length: $k(s) = a + bs + cs^2 + ds^3$
>
> [...] The path $\mathcal{T}_p$ is a continuous curve through the state space [x y θ k] defined by a start state $x_0 = [x_0\ y_0\ \theta_0\ \mathrm{k}_0]$ and a set of cubic polynomial spiral parameters. Since our paths are defined as functions of curvature w.r.t are length, they can be readily followed by the vehicle at any speed, subject to constraints on steering rate, lateral acceleration, and the like [...] McNaughton et al. (2011)

Multiple splines are created from the described modified lattice path generation. As seen in Figure 43, in the lattice algorithm, multiple trajectories converge into the same lattice vertex.

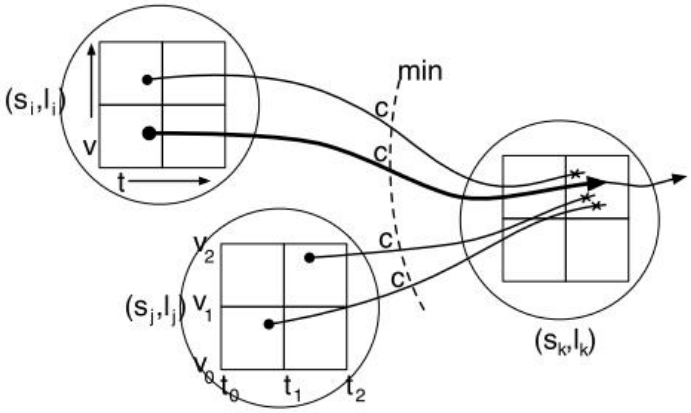


Figure 43 – Trajectories conversion in Lattice. Each vertex in the Lattice is identified with a quadrant corresponding to a pose on the road and a range of times and velocities. In each vertex, the ending time and velocity value from the incoming trajectory with the minimum cost is assigned to the lattice vertex's (t,v) coordinates. These values are used in turn for outgoing trajectories from the vertex). Source: McNaughton et al. (2011)

The modified Lattice uses a cost function to select the final trajectory, which includes terms such as obstacle avoidance, vehicle's physical limitations, such as the rate of change of path curvature w.r.t time. This function also penalizes lateral acceleration to promote more comfort to the passenger. At first, the cost function assesses the terms depending on (x,y θ, k), afterward evaluating the terms depending on a,t,v. The best trajectory is represented by the continuous sequence of trajectories through the Lattice to the final vertex with the smallest cost. In addition, to specify the last vertex, a minimum planning horizon t_h in terms of time is applied. This strategy excludes long and non-feasible trajectories w.r.t time. The modified lattice implementation enables rapid progress at a reasonable cost. To accomplish this, the selected final vertex n_f is the one that minimizes.

$$arg_{n_f} min \hat{c}(n_f) - k_s(s(n_f)) + k_t(t(n_f)) \tag{10}$$

A weighted sum of the trajectory cost to reach the vertex nf. They assigned a bonus for driving further with a penalty for taking extra time. Finally, they traced backward through Lattice from n_f to the start state, reconstructing the trajectory. The weights are tuned manually to achieve the desired balance of forwarding progress against a low trajectory cost. In this modified lattice implementation, the vehicle remains roughly parallel to the road.

According to the authors, a drawback of the modified Lattice is the maneuverability limitation while constructing courses at low speeds. This algorithm requires an exhaustive search to sweep all possible expanded nodes and formed trajectories regarding computa-

tional effort. In a high-road experiment (McNaughton et al., 2011), the GPU needed to evaluate about 400.000 trajectories per cycle.

The integration of Freespace global planner and the modified lattice local planner with Autoware.AI for the path generation is demonstrated in Appendix A.2.

5.2.3 Open Planner-Global

The Open Planner was designed for a structured environment. In contrast with OMPL and Navigation Stack planners, which need a cost map to generate feasible paths, the global Open Planner needs to be fed with a High Definition Map (HD-map). The HD map is a particular type of map which provides information on the road, such as Lane Network information, intersection, traffic lights, signs, curbs, lane boundaries information, and other specific information for the AV. The HD-map, the vehicle start position, and the destination pose (goal) are the necessary inputs for the Open Planner to plan the global path. The open planner plans its global trajectory close to the center of the lane road, as represented by a straight blue line in Figure 44.

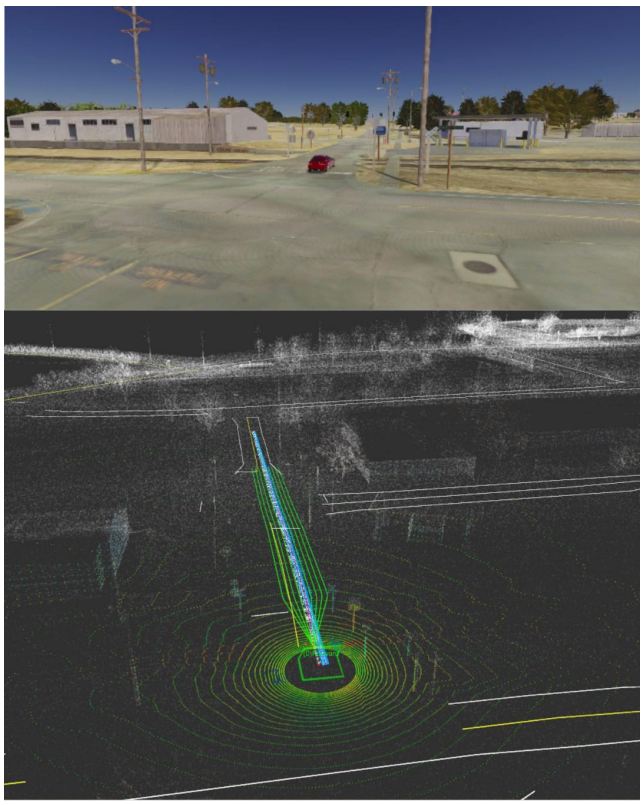


Figure 44 – Up: SVL simulator graphical interface. Bottom: Op Global Planner taking the center of the lane (Blue) and the rollout local planner trajectories at lateral sides (Green) in rviz. Source: Own Authorship

The AV remains at the center of this line traveling in the right direction and changing

the lane only when allowed and needs to move to the other one to perform a right or left turn. Nevertheless, the lane persecution region can be customized, tuning the path planner algorithm's or control parameters, as demonstrated in this study (Flessner, 2020).

The general architecture of Open Planner is shown in Figure 45.

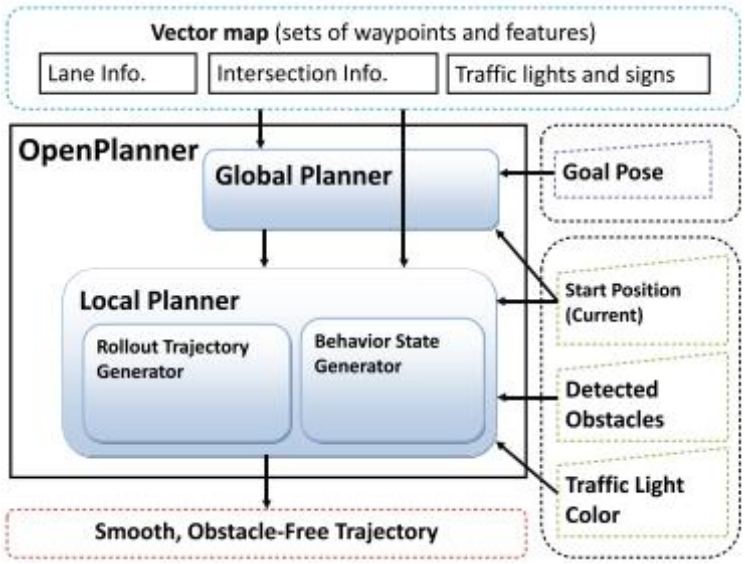


Figure 45 – Open Planner Architecture. Source: Darweesh et al. (2017)

Thanks to the HD-map information, in Open Planner, kinematic optimization becomes more straightforward than the others described in this work because it already provides the AV the unfeasible and not allowed routes, making it easier for the car-like robot to avoid awkward paths.

5.2.3.1 Open Planner Behavior State Machine

Another component of Open Planner architecture is the behavior generator (or behavior planner). This part is responsible for the task planning. It relies on state machine usage to represent tasks and apply the rules that govern transitions between these tasks. The behavior states transition conditions, and an illustrative example of its use while accomplishing autonomous navigation on Nayoga University is available in Annex B.1. Each state represents a traffic situation. The transitions between these states are controlled by intermediate parameters, calculated using the current traffic information and pre-programmed traffic rules.

5.2.3.2 Open Planner-Local

The local open planner (Op) is integrated into the global planner and runs parallel to the reference path. The Op local planner generates multiple rollouts, starting from the

vehicle's center and added perpendicular to the reference path. The rollouts are linearly sampled and are post-optimized to satisfy the vehicle kinematics. The rollout generation obeys a maximum time of 0.1 seconds, allowing the controller to respond quickly to the changes in velocity. The inputs for the rollout creation are current position, planning distance, number of rollouts, and the next section of the global path. The output is n smooth trajectories, running from the vehicle's center out to the maximum planning distance. These steps are seen in Figure 46.

.

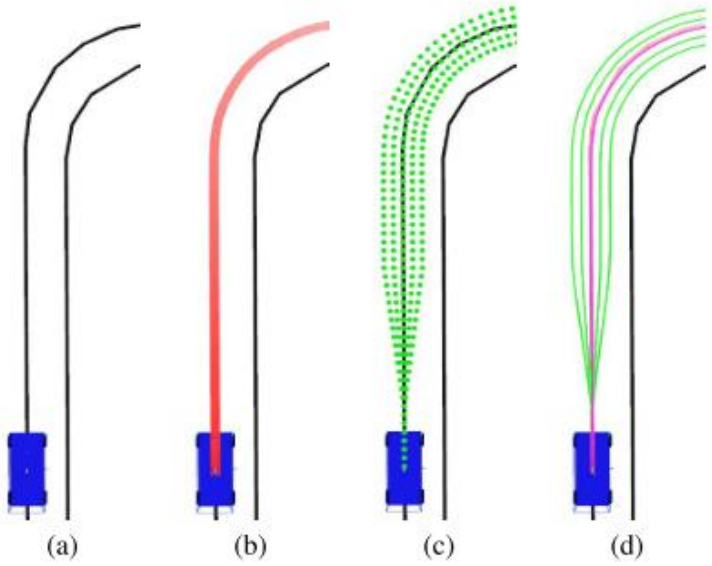


Figure 46 – Steps for generating local trajectories: (a) original map, (b) path section extracted from the global path, (c) sampling phase, (d) smoothing using the conjugate gradient. Source: Darweesh et al. (2017)

As it can be seen, the sampled rollouts are split into three sections: the car tip margin, roll-in margin, and rollout section detailed described here (Darweesh et al., 2017). Object tracking and detection is an essential feature for the Op local planner to eliminate the false negatives and false positives detections. After the rollouts generation step, considerable candidate trajectories need to be analyzed. To select the one with the lowest cost, a cost function is applied, considering the rollouts and the detected obstacles (inputs).

This cost function calculates three different normalized cost measurements: priority, collision, and transition. Afterward, the smallest cost is selected. A complete description of these costs can be found in (Darweesh et al., 2017).

The integration of Open Planner with Autoware.AI for the path generation is demonstrated in Appendix A.3.

5.3 Autoware.Auto

Autoware.AI was a considerable step to automotive research groups reaching simulations close to an AV's reality. Nevertheless, the Autoware team detected the lack of security in data transmission and the lack of robustness, being one of the factors closely correlated to the limitations of ROS1 middleware. In addition, companies' members from the foundation, such as APEX.AI, have envisioned a commercial perspective for the software. These factors culminated in the development of a new project: Autoware.Auto.

Autoware.Auto is ROS2 based. The main difference between ROS2 and ROS1 is that ROS2 uses Data Distribution Service (DDS) for publishing and subscribing instead of a custom message handler. The DDS allows real-time data sharing across network-connected devices. Therefore compared to ROS1, ROS 2 has a better transmission performance. The DDS feature is considered the heart of ROS2. A profound explanation can be accessed here (Corsaro, 2020).

In addition, ROS2 provides a life cycle management for the nodes, allowing greater control over the state of the ROS system. It ensures that all components have been instigated correctly before enabling them to execute their behavior. It also allows the node to be restarted or replaced online, ensuring more robot safety (Génération ROBOTS, 2019).

Autoware.Auto was conceived to optimize Autoware.AI limitations in terms of security and solve the real time data flow issue. Autoware.Auto can work with hard-real time data flow and, therefore, can handle time-critical ensuring a high safety degree compared to Autoware.AI. In addition, different from Autoware.AI, a framework oriented for autonomous driving modules evaluation, Autoware.Auto focus on addressing a specific mission in a particular real scenario. Therefore the range of available modules, from perception to planning, is restricted. Autoware.AI is the ideal platform for testing packages and modules. Autoware.Auto, however, is oriented to cover accurate autonomous driving tests in particular scenarios, as explained here by Autoware.Auto System Architect (Tellez, 2021).

Autoware.Auto aims to test as many different and specific scenarios as possible to accomplish the driving tasks with more reliability and ensure the software's robustness. They established the Operation Design Domain (ODD) strategy, which is shown in the Figure 47.

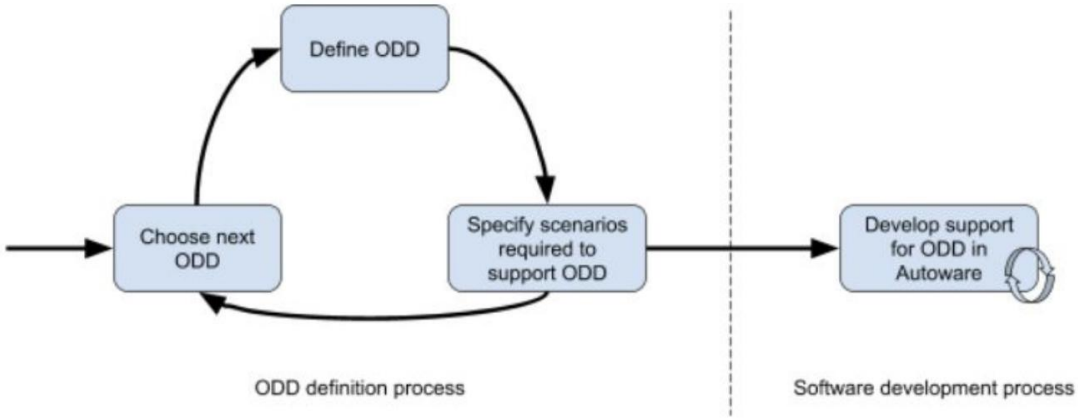


Figure 47 – Autoware Foundantion (AWF) development Cycle. Source: Whitley (2021)

An ODD is defined to test specific automated driving functions, such as parking. The selection of an appropriate ODD relies on automotive industry experience, and usually, it follows a self-driving car's engineer methodologies, as described (Lee et al., 2020). A complete diagram with a deeper investigation of the ODD working principle is available in Annex B.2.

Up to the present time of this research, the AWF team and collaborators had accomplished just with the first ODD: Autonomous Valet Parking (AVP). The compatible modules for this ODD were developed, tested, and achieved the final goal. The system architecture that the AWF team designed to address this ODD is displayed in Figure 48.

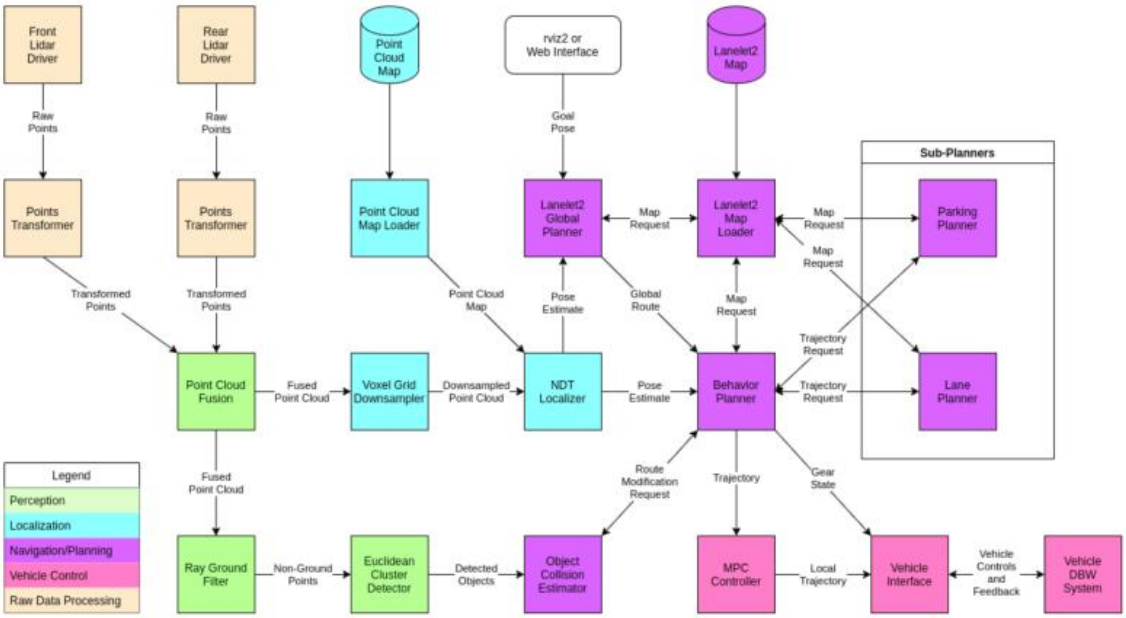


Figure 48 – AWF AVP Architecture. Source: Whitley (2021)

Figure 48 shows all the applied modules to reach the AVP mission. A complete explanation of the integration and composition of each one of the depicted modules can be found on the AWF course (AWF, 2021). As the goal of this work is planning, just this module will be covered.

5.3.1 Planning Module

Because the framework application under discussion has a more commercial rather than academic orientation, unfortunately, no article was found describing the planning module. Even so, valuable technical documents discussions (AutowareAuto git, 2021) and the motion control live class (Longo, 2020) can help clarify the planning module working-principle. An overview of autonomous navigation architecture with focus on motion planner is available at Annex B.3.

5.3.1.1 Lanelet2 Global Planner

The Autoware.Auto global planner name derives from a proper map format for Autonomous Driving: The Lanelet2. The lanelet map shrinks encapsulate indispensable data for autonomous safety driving (Poggenhans et al., 2018), such as:

- Points: Express the coordinates of one point (represented by poles and punctual structures);

- Linestring: Express a line. Used to describe curbs, road markings, facades, fences, or traffic lights;
- Lanelet: An atomic section. It is created in the unit of attribute change and intersection. It is represented by lanes, pedestrian crosses, and rails;
- Areas: Indicates where AV movement is not possible, such as green areas, squares, parking areas, or buildings;
- Polygon: Similar to line string, however, assumes that the first and last point of the polygon is connected to close the shape. The difference with the "Area" object is that it does not have routing connection information.
- Regulatory Elements: Express Traffic Regulations. Refers to traffic rules, speed limits, priority rules, traffic lights, etc.

Except for the Regulatory Elements, all the Lanelet2 data primitives are derived from the first one (Points). An HD-map content and a lanelet2 data primitives can be checked in Annex B.5.

The Lanelet2 is also a ROS library that handles the Lanelet2 HD-map data. This library can interpret and process complex scenarios from HD maps (ROS Wiki- Lanelete2, 2020).

The Lanelet2 global planner working principle is trivial: It takes the AV's starting pose from the Localizer and the goal position and then calculates a route following the HD-map semantics(traffic rules, allowed road directions, signs, etc.). The global planner in Autoware.Auto works as an onboard GPS, which calculates a route in a road network. For this reason, it is also called a route planner.

5.3.1.2 Behavior Planner

Similar to Open Planner behavior state machine

- 1: Receives the global route from the lanelet2 global planner;
- 2: Subdivides the global route into "subroutes" to find out the order of Trajectory Planners to activate;
- 3: For each subroute:
  - Activate the appropriate trajectory planner (Lane Planner or Parking Planner) by sending an Action;
  - The Behavior Planner receives the trajectory by topic from the selected trajectory planner while activated;

- The Behavior Planner calls the Object Collision Estimator's service to modify the velocity of trajectory;
- The Behavior Planner relays the received trajectory from Object Collision Estimator to control.

In a summarize, the behavior planner monitors the phase of the self-driving trip, triggering the Lane Planner in the beginning and during the road navigation and controlling the AV's speed, with the assistance of the Object Collision Estimator module, and deactivating the Lane Planner and triggering the parking planner algorithm when the AV is approaching the goal (parking spot).

5.3.1.3 Trajectories Planners (Lane Planner, Parking Planner)

The Trajectories Planners are activated when one of one-of-them receives a subroute from the Behavior Planner. Once the subroute is received, it will request relevant map information from the map provider module and start planning the trajectory that navigates from the start point to the goal point of the given subroute. While the trajectory planner (Lane or Parking) is activated, it broadcasts the trajectories to the Behavior Planner for monitoring purposes. The trajectory planner keeps calculating and sending points to build the subroute trajectory until the vehicle reaches the goal of this subroute. When the "sub-goal" is called, the trajectory planner returns a code to stop planning using Action's Result (A ROS type communication) described in section 4.1.

5.3.1.4 Object Collision Estimator

The Object Collision Estimator module is an essential piece from the navigation module because it tracks the distance and speed at the vehicle will collide with the front car. This module receives the trajectory from the Behavior Planner. Then it compares the trajectory with the detected objects by the perception modules, modifying the velocity profile of this trajectory. This module is responsible for smoothing the velocity profile to enable a feasible speed for the control module (MPC). Finally, the Object Collision Estimator sends back the modified trajectory to the Behavior Planner, further transmitting to the Model Predictive Control (MPC) controller.

The visual explanation of the whole planning process is shown in Figure 49.

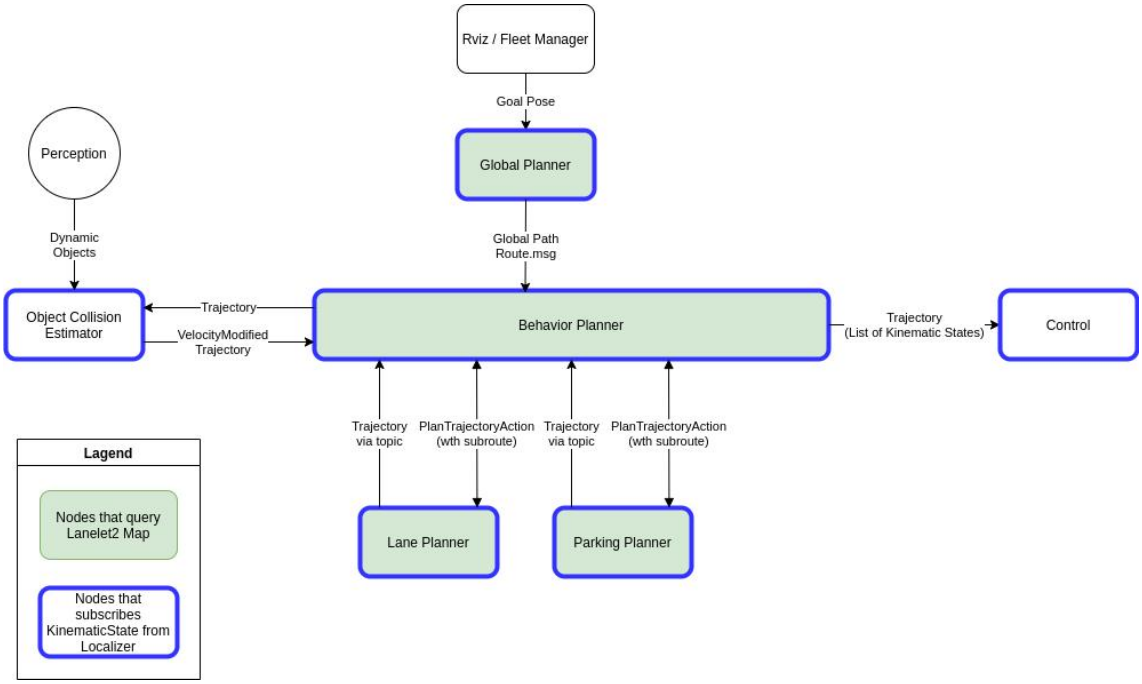


Figure 49 – AVP Planning Architecture. Source: AutowareAuto git (2021)

The AVP planning module and its planners were tested in this dissertation. The figures from this simulation and respective notes are found in Section 6.5.

Recently, the second ODD was set by Autoware.Auto team: Autonomous Cargo Delivery. The third, fourth, and fifth ODDs are objects of discussion: Highway ACC and driver assistance and Autonomous Bus Rapid Transit and Active Campus Navigation, respectively.

5.4 Apollo.Auto

Apollo.Auto is software developed by Baidu. In the first releases, Apollo was implemented based on ROS. However, similarly to Autoware.Auto, this framework migrated from ROS1 to Cyber-RT [1] protocol to ensure more safety and robustness. Apollo.Auto contains more than six different versions. Figure 50 refers to Apollo's latest version used in this research (6.0 release).

[1] Cyber-RT is protocol communication analogous to ROS2

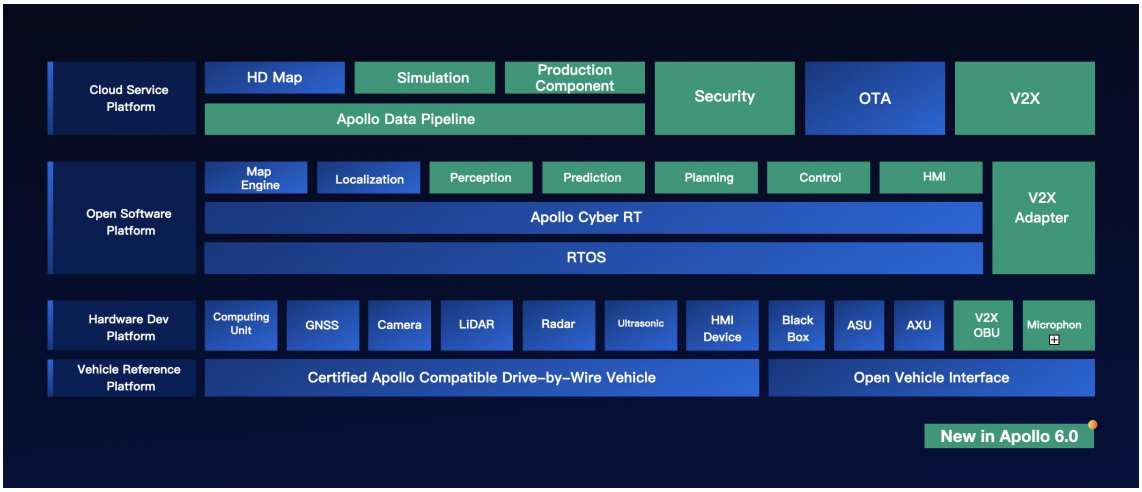


Figure 50 – Apollo.Auto Architecture Diagram.Source: Baidu (2021)

The architecture of Apollo.Auto is similar to the Autoware frameworks described previously. To perform autonomous navigation, it relies on the same standard modules: Mapping, Perception, Localization, Prediction, Planning, and Control. One interesting feature added in Apollo, in contrast with Autoware, is the addition of a safety module, called guardian, which performs the function of an Action Center, and intervenes if the Monitor detects a failure (Baidu, 2021). Figure 51 displays the integration of Apollo Modules.

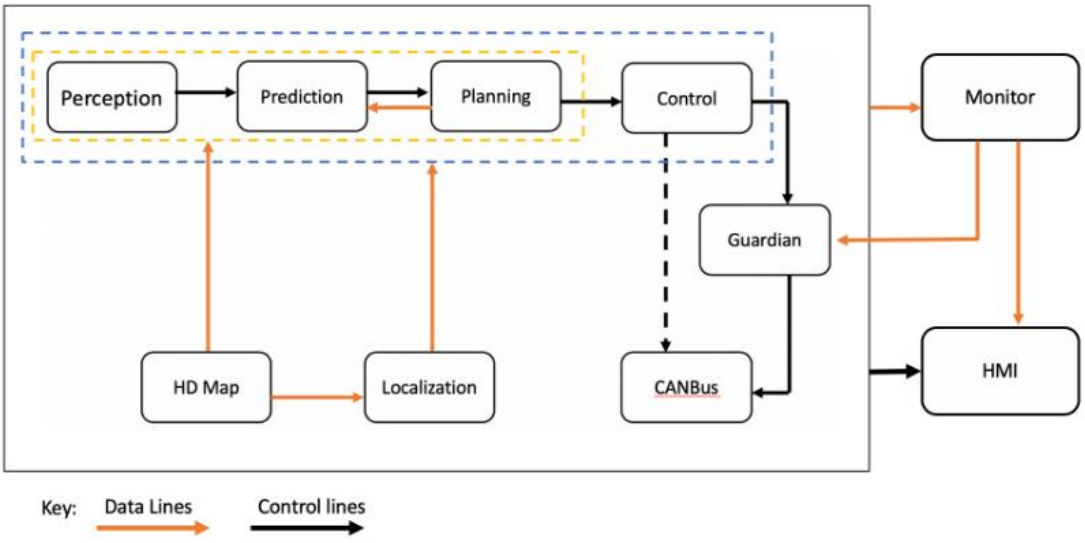


Figure 51 – Interaction of Apollo.Auto modules. Source: Baidu (2021)

An in-depth explanation of each apollo module can be found in (Baidu, 2021). In this work, just the planning module will be described in detail.

5.4.1 Planning Module

The Public Road Trajectory Planner is based on The EM-type iterative algorithm (Dempster et al., 1977). This planner deals with multiple requirements for level-4 on-road autonomous driving. The algorithm comprises three layers that address a safety trajectory at the end of the process. The first layer deals with lane determination. A multilane strategy covers both nonpassive and passive lane-change scenarios. For the candidate lanes, the obstacles and environment information are projected on lane-based Frenet frames [2]. Under this framework, each candidate lane forms a best-possible trajectory based on the lane level. In the end, the trajectory decider will resolve which route to choose based on both the cost operating and safety rules.

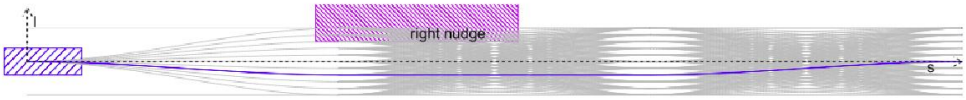


Figure 52 – Selected Lane, based on cost operating and safety rules

Source: Fan et al. (2018)

The second layer address the lane-level motion planning. To generate an optimal trajectory, the EM planner optimizes the path and speed iteratively using Dynamic Programming (DP) and Quadratic Programming (QP).

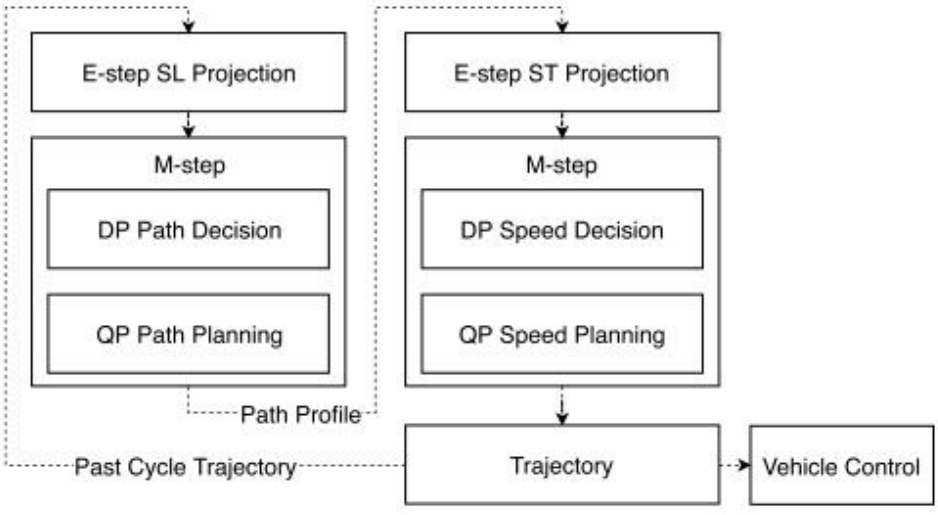


Figure 53 – EM path-speed decoupled optimizers. Source: Fan et al. (2018)

[2]Representation from coordinates in a lateral(d) and a longitudinal(s) distance from the origin (see Annex 98

The third layer covers traffic regulations and decisions. To avoid obstacles and follow a reasonable trajectory, the DP calculates a rough and feasible trajectory. The interaction between barriers and the Ego vehicle's course is measured in a second step. Finally, the QP-based smoothing spline is used to smooth the trajectory. The output is a feasible and smooth solution (Fan et al., 2018).

An overview of the applied steps in the EM planner to generate the path and speed splines can be visualized in Figure 54.

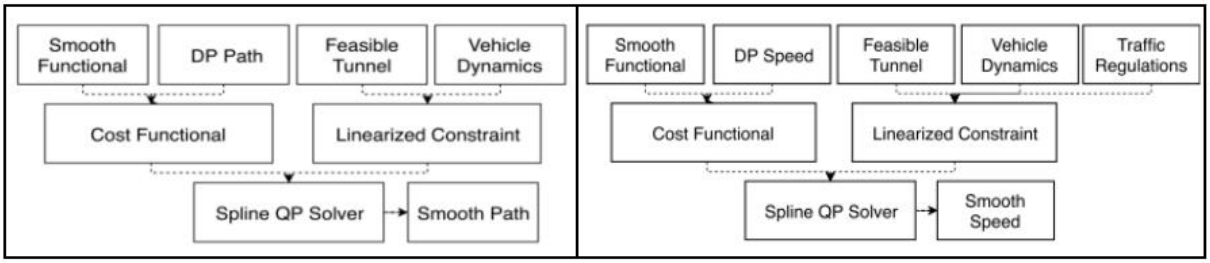


Figure 54 – Path and speed spline smoothing process. Source: Fan et al. (2018)

Finally, the optimized trajectory is generated, as shown in Figure 55.

Figure 55 – Optimized Trajectory after refinement of second layer Source: Fan et al. (2018)

The integration of EM Planner (Public Road) with Apollo.Auto for the path generation is demonstrated in Appendix A.4.

5.5 Proprietary AD Stacks

There are others AD Stacks usually employed for research groups of this area. However, they are not open source. Between them, it is worthy of describing:

5.5.1 Nvidia

The Nvidia AD Stack is commercial software. It consists of 4 blocks. Each driver (block) is responsible for managing different modules:

- Drive AV: Handles with the Planning, Mapping and Perception modules;
- Drive IX: Responsible for Visualization interface, for monitoring the system (failures) as well as the driver state ;
- DriveWorks: Responsible for calibration of the Ego vehicle motion and networks connection. The drive core submodule is responsible for sensors processing, such as image processing and PCD processing;
- Drive OS: the reference operating system and associated software stack explicitly designed for developing and deploying autonomous vehicle applications. NVIDIA DRIVE OS delivers a safe and secure execution environment for safety-critical applications, with secure boot, security services, firewall, and over-the-air (OTA) updates (Whitley, 2021).

Nvidia supports different hardware platforms for AD, such as Xavier and Pegasus.

The Nvidia DriveWorks is displayed in Figure 56.

Figure 56 – Nvidia Drive Constellation Architecture. Source: Whitley (2021)

5.5.2 Elektrobit

The Elektrobit ADS is based on the AI Behavior Arbitration (AIBA), an algorithm designed to arbitrate between different driving strategies, or vehicle behaviors, based on

AIBA's understanding of the relations between the scene objects. These ADS focused on an improvement of the AV's decision-making process. This system constructs a description and understanding model of the driving scene based on human driver behavior (Trăsnea et al., 2019). An overview of these frameworks is shown in Figure 57.

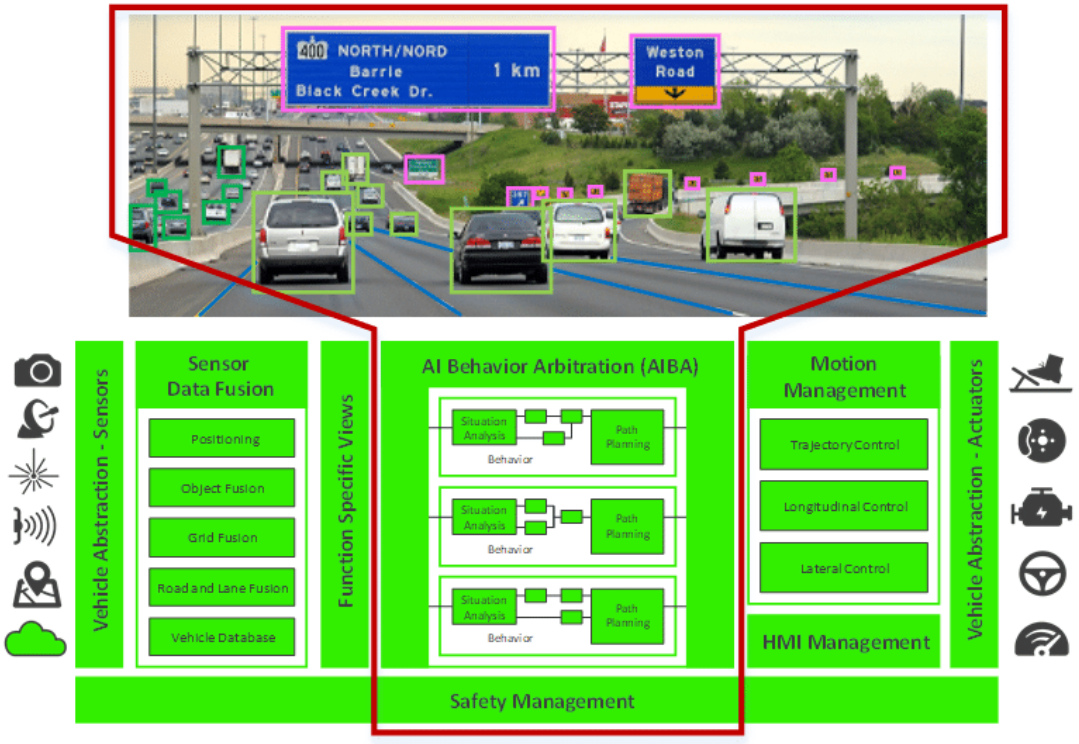


Figure 57 – Elektrobit Architecture. Source: Elektrobit (2021)

Currently, numerous software enterprises are working to develop AD modules and functions. Each ADS, open-source or proprietary, concentrates more effort in developing and optimizing different modules.

6 Simulation Setup

To carry out the simulations, a cloud machine from paperspace [1] was used. The specifications are shown in Table 2.

Table 2 – Machine Description

Parameter	**Specification**
CPU	
Board	4.7.6.6.3/ HVM domU (Xen)
Name	Interl(R)Xeon(R)CPUES-2023v4@2.60GHz
Description	1 physical processor; 16 cores, 16 threads
Configuration	16x 2600.09 MHz
Memory	61810544 kiB
GPU	
Graphics Processor	Quadro P-6000
CUDA cores	3840
Total Memory	24576 MB
Computer	
Machine Type	Virtual
Memory	61810 MB
Operating System	Ubuntu 20.04.5 LTS
Display	
Resolution	2560x1440 pixels

Source: Own Author.

To simulate different AD planners and scenarios, the Gazebo (see section 4.1) and LGSVL simulator (see section 6.2). The simulators used are ideal because they enable the sensors arrangement (sorts, position, frequencies, etc.), scenarios, vehicles, environments, etc. It is also almost inexpensive.

The performed simulations with the correspondent chronological order provided by section 4.2 will be demonstrated in this chapter's subsequent sections.

6.1 Navigation Stack Simulation Setup

6.1.1 City

The Navigation Stack works by default with the Gazebo simulator (described in section 4.1). The environment set up for the self-driving car simulation on this software, named as .world file, was defined based on the Center of Automotive Research on the Integrated Safety Systems and Measurement Area (CARISSMA). This environment is composed of

[1] A cloud platform which rent powerful computers in terms of GPU and CPU

static standing pedestrians and static vehicles (hatchback, SUV, pickup) and buildings, trees, and other usual objects present in this place.

6.1.2 Vehicle Model

The autonomous vehicle configuration was based on the Prius car model, which contains 16 beam lidar on the roof, eight ultrasonic sensors, four cameras, and two planar lidars (Tellez, 2017). A configuration relative similar to the BMW research AV (Aeberhard, 2017) displayed in Figure 58.

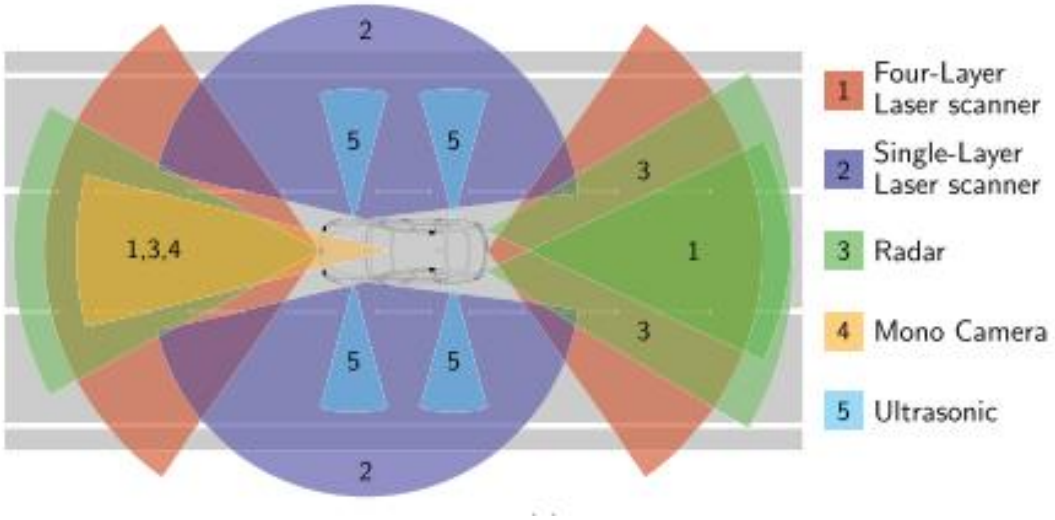


Figure 58 – BMW 5 Series test vehicle. Source: Aeberhard (2017)

The vehicle description on ROS, written on the unified description robot format (URDF) format, is substantially modular. Therefore, the sensors' plugin can be easily replaced in the URDF file. However, since the research purpose of this work is to investigate the trajectory planning module and not the perception, it was decided to use the already available sensors defined in the Prius model to advance quickly to the primary goal. And, afterward, if the case, exchange the types and number of sensors. The external vehicle's Mesh chosen was the BMW's 335i, a model present on the lab CARISSMA during an internship in 2020. The MMeshwas acquired on 3D Warehouse rearranged on Sketch-up and Blender software.

The sensors described on URDF were later re-positioned according to the new dimensions and geometry to fit the Mesh above correctly.

6.1.3 Simulation Results

The Navigation Stack is an ideal framework to test indoor robots subject to holonomic constraints. However, it is not efficient for car-like-robots simulations since the available

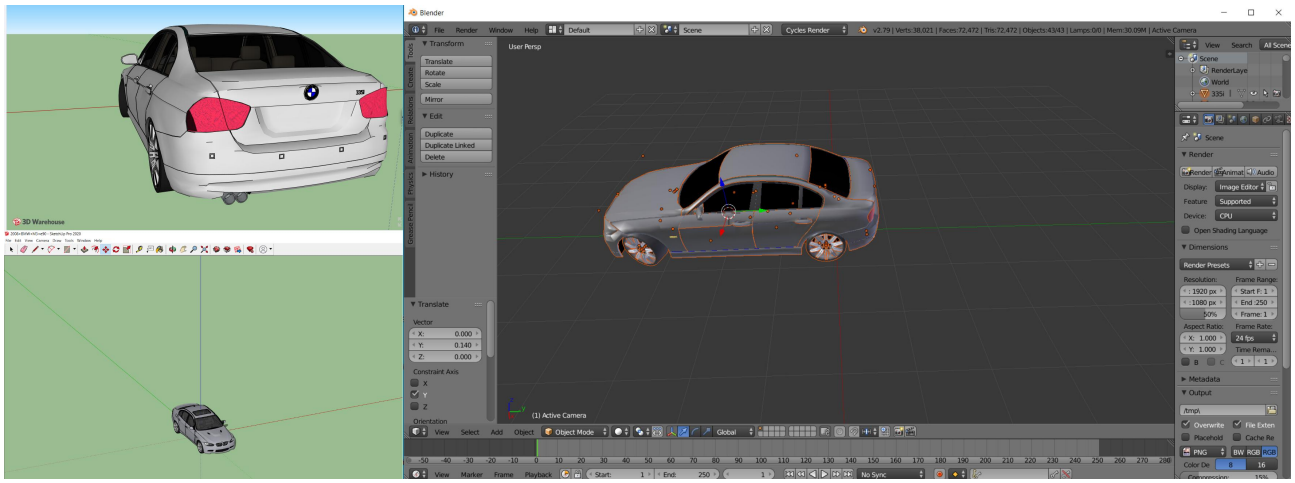


Figure 59 – Mesh Acquisition on 3D Warehouse on top-left, rearrangement on SketchUp Pro at left-bottom. Reallocation of mMesh'scenter of mass with Blender on the right. Source: Own Authorship

path planners do not appropriately fit robots with non-holonomic constraints. Even so, among the tested planners, the TEB local planner was integrated with success, and the AV was able to reach the target avoiding collisions along the way, as demonstrated in subsection 4.2.1.2.

6.2 LGSVL Simulator

6.2.1 Selection

LGSVL can import, edit and export the most common HD-maps formats, such as OpenDRIVE, Lanelet2, and Apollo HD map. It is worthy of citing that other commercial and proprietary simulators are also widely used with AD stacks, such as CarMaker, ADAMS, ANSYS, NVIDIA's Drive Constellation, etc., as well as the open-source, such s Airsim, CARLA, and Deepdrive. The SVL simulator from LG was selected due to the extensive number of tutorials, documentation, fast and excellent support provided in GitHub (LGsvl git, 2021), and mainly for the high realistic design.

6.2.2 Conception

The LGSVL simulator is a Unity-based high-fidelity simulator for autonomous driving. It is developed using the Unity game engine, taking advantage of its latest technologies, such as the High Definition Render Pipeline (HDRP) (*Unity Technologies*, 2021). This feature simulates a photo-realistic virtual environment very close to reality.

The LGSVL simulator communicates with Baidu and Autoware Foundation Autonomous Driving Stacks: Autoware.AI, Autoware.Auto, Apollo.Auto and recently also has an interface with the Navigation Stack2 (Nav2) (Macenski et al., 2020).

6.2.3 AD Stack -LGSVL Bridge

The simulator and the recent related frameworks are communicated through the related AD's bridges: ROS, ROS2, and Cyber-RT, respectively. These bridges bypass messages from the AD stack modules to the simulator and the inverse. Usually, ROS/Cyber nodes or packages make the message type conversion during the process. Afterward, the vehicle's sensors receive the data from the simulator. They are sent to the AD Stack modules. These modules process the data and send control commands to the vehicles to stop, swerve or accelerate through the bridge. An illustrative image of this integration process is shown in Figure 60.

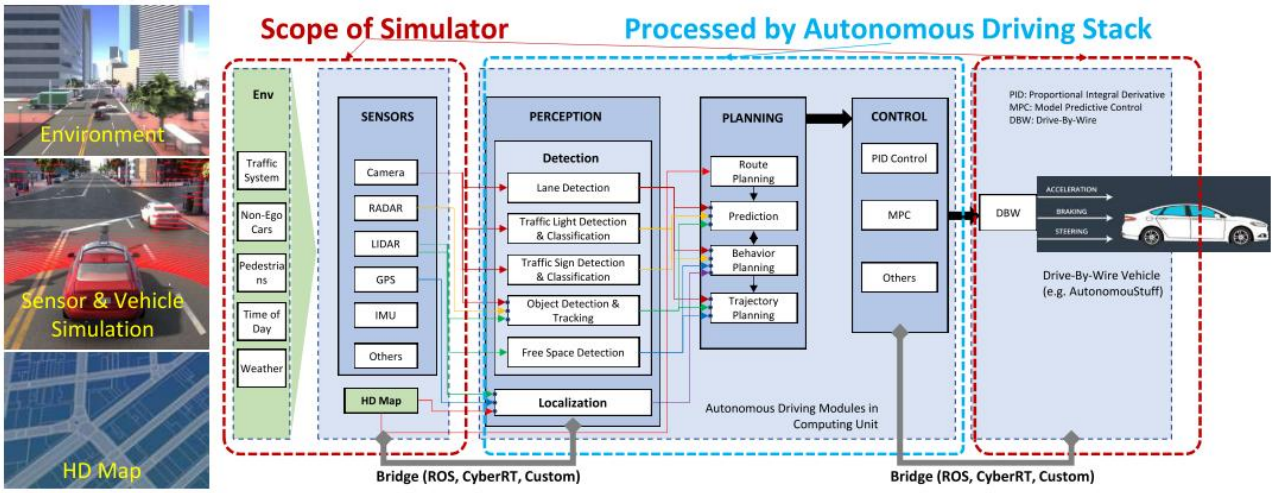


Fig. 4. High-level architecture of autonomous driving system and the roles of the simulation engine

Figure 60 – High-level architecture of an autonomous driving system and the roles of the LGSVL simulator engine. Source: Rong et al. (2020)

The messages conversion from AD stacks to the simulator and the transmission to real car vehicles via the CAN bus is not the focus of this work. A detailed work from this process can be found in (Battiston, 2015).

6.2.4 Simulation Engine

The LGSVL simulator handles four functions:

- Environment Simulation: Include traffic simulation, physical environment simulation such as weather and time-of-the day;
- Sensor Simulation: It accepts JSON formats to sensor intrinsic and extrinsic parameters configuration. In addition, it provides sensor models which contain parameters that math the real-world counterpart, e.g., Velodyne VLP-16 LiDAR, for example, which generate point cloud data in the same format of the real sensors;
- Vehicle Dynamics: The simulator provides an essential vehicle dynamics model for the ego vehicle. However, it allows integration of external third party dynamic models;
- Control of the Ego Vehicle in Simulation.

6.2.5 Scenarios

Relevant attributes from SVL are the possibility of scenario customization, modifying road conditions, and adding multiple agents (static or dynamic vehicles or pedestrians). The SVL has available multiple AD scenarios, from AV's test facilities such as GoMentum and Shalun to cities such as Borregas Avenue and San Francisco (Rong et al., 2020).

6.2.6 Related Functionalities

Advancements features from the simulator, such as multiple AD stacks communication, V2X (V2V, V21, V2P) system, interface with Software-in-the-loop (SIL) and Hardware-in-the-loop (HIL), are better described in (Rong et al., 2020).

6.3 Determination of Test Cases Scenarios

One general method to assure a safety system in autonomous driving is testing the AV in a certain amount of miles (Kalra and Paddock, 2016). Other researchers, however, propose validating AD systems testing these vehicles under critical scenarios on simulation platforms.

The scenarios were configured using the LGSV PythonAPI (LGsvl Python API, 2020).

For the test-cases selection, numerous methodologies can be applied. Klück et al. (2018) makes use of ontology of the environments in which the AV is interacting as input of combinatorial testing. The output from this interaction is obtaining critical test scenarios in which the faults on vehicle capabilities can be detected. An overview of the ontology methodology proposed (Klück et al., 2018) is shown in Figure 61.

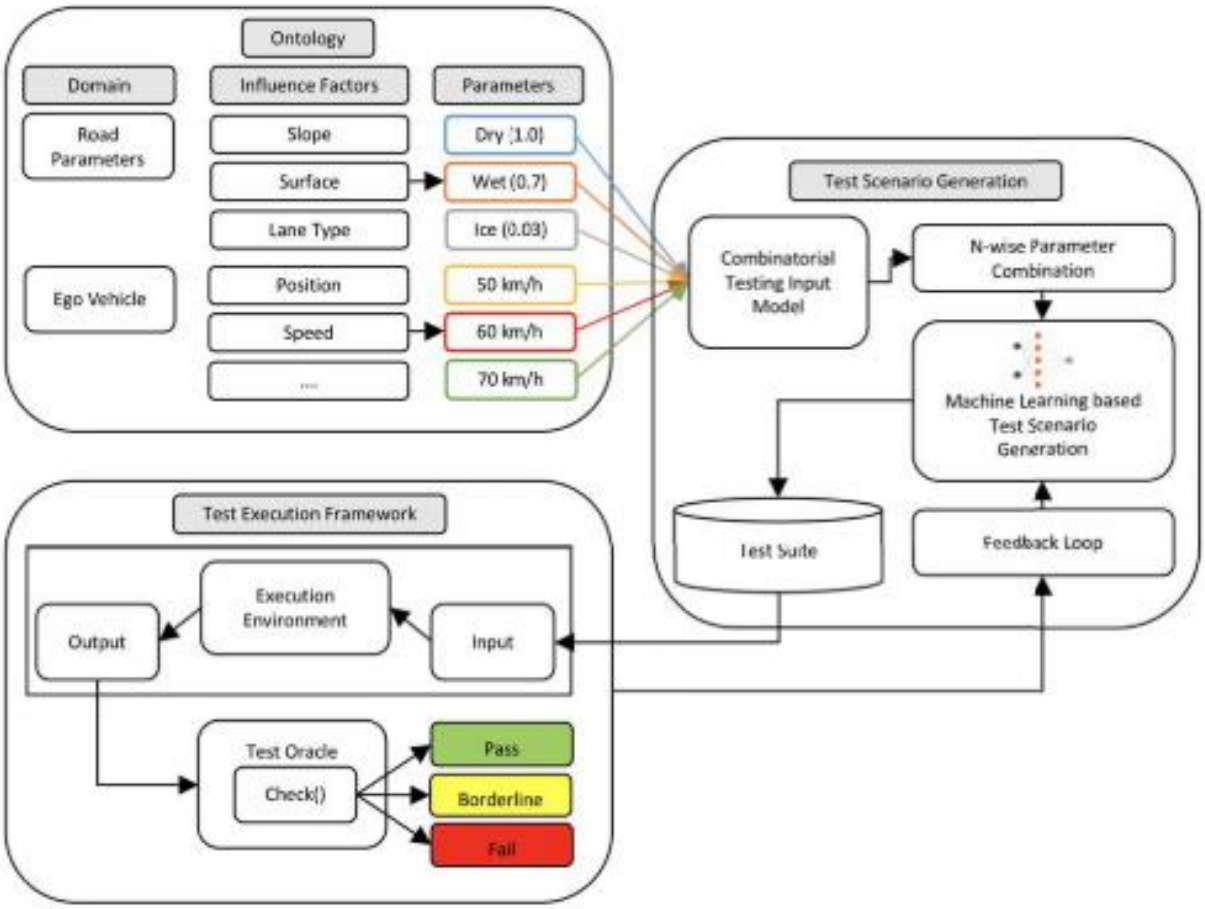


Figure 61 – Architecture of three main pillars of general automated testing approach for verification and validation of automated and autonomous driving functions. Source: Klück et al. (2018).

Ponn, Schwab, Diermeyer, Gnandt and Záhorskỳ (2019), on the other hand, firstly evaluated the AV driving behavior in specific scenarios, and through this behavior with the conjunction of Key Performance Indicator (KPI) metrics, it defines system-specific challenging scenarios for the safety assessment. Ponn, Schwab, Diermeyer, Gnandt and Záhorskỳ (2019) also describes and chronicles the type of scenarios, from scenes to critical scenarios. An overview of this method is shown in Figure 62.

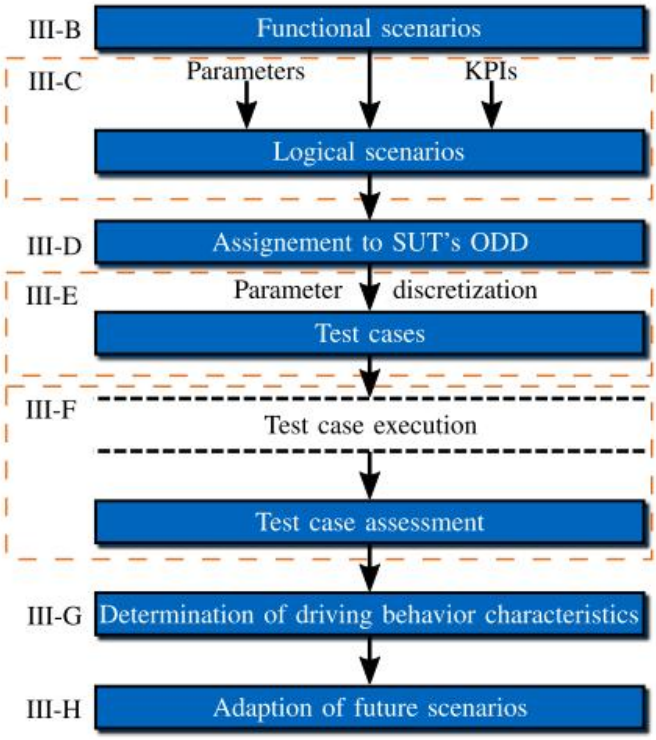


Figure 62 – Overview of the described methodology. Source: Ponn, Schwab, Diermeyer, Gnandt and Záhorskỳ (2019).

At step III-G, the features from the driving behavior of the system under test (SUT) are identified. The KPI and weaknesses in the driving behavior of the SUT are defined based on expert knowledge and are done manually. Then these results are evaluated using the KPIs, and the weak spots from this behavior are determined (a delay or jerk in the lateral or longitudinal control, for example). Finally, the scenario that generated this critical AV behavior creates more similar critical scenarios. Validating autonomous driving is relatively new, and there is still no standard model or official regulations for that. However, numerous testing and validating methods, such as the described previously, can be found in the literature. Nalic et al. (2020) have investigated multiple approaches in this regard.

Motivated by the lack of standard criteria to the scenarios selection for AD validation functions, the research project PEGASUS (**P**roject for the **E**stablishment of **G**enerally **A**ccepted quality criteria, tools and methods as well as **S**cenarios and **S**ituations) was promoted. The PEGASUS project is based on an argumentative structure, called the "safety argument," to reach a method for the assessment of Highly Automated Driving Functions (SAE level 3+). PEGASUS project uses as inputs huge Database with analysis of accident data, take into account comparisons between the driver and the AV and other numerous indicators to build a combined safety argumentation.

As shown in the figure, PEGASUS applies a systematic description of scenarios split into six independent layers for scenario modeling 63.

Figure 63 – Systematic description model of scenarios with six independent layers by PEGASUS. Source: Audi and Volkswagen (2020)

In a further phase, the selected scenarios and corresponding tested automated driving functions are evaluated by a systematic concept. This process workflow can be checked in Annex B.6.

A detail from the process can be found in the PEGASUS manual (Audi and Volkswagen, 2020). This manual assists in the selection of critical scenarios, as well as suggest simulation methods for AD validation. However, one limitation of its manual is its scalability, once it country has particular laws and regulations. Although the project initially aimed to automate driving safely on the national level, in the foreground. International conferences and dialogues are being held at the continental level (Europe, Asia, North America). Therefore the conclusions of the PEGASUS project can serve as a basis for other countries. PEGASUS project is supported by German Federal Ministry for Economic Affairs and Energy (BMWi) and contain prominent partners, such as Audi AG, BMW Group, Daimler AG, Robert Bosch GmbH, Volkswagen AG, as well as universities such as Technische Universität Darmstadt (FZD), Forschungsgesellschaft mbH, Aachen (fka) and other renowned automotive industries and research groups.

A concrete scenario was set first for the path planner's evaluation based on the

guidelines above. Then more complex scenarios which could significantly affect a safety measurement indicator, the Time To Collision (TCC), were also programmed. As these scenarios reflect proximity to an accident, they can be considered critical scenarios (Ponn, Schwab, Diermeyer, Gnandt and Záhorskỳ, 2019). The scenarios chosen for this work are:

- Static Truck: A truck is parked on the right side of the Ego Vehicle Lane, obstructing its passage.
- Crossing the Crosswalk: A pedestrian suddenly traverses the crosswalk while the Ego vehicle turns left in a pedestrian direction.
- Stop and Go: A sedan moves forward the Ego Vehicle and suddenly stops and goes successively, while the Ego vehicle moves closely behind it.
- Cut-in: The Ego Vehicle is moving forward in its lane when a Sedan abruptly changes from its road-lane and entries in front of the Ego Vehicle.

Figure 64 shows the selected scenarios for testing the planners in this work.

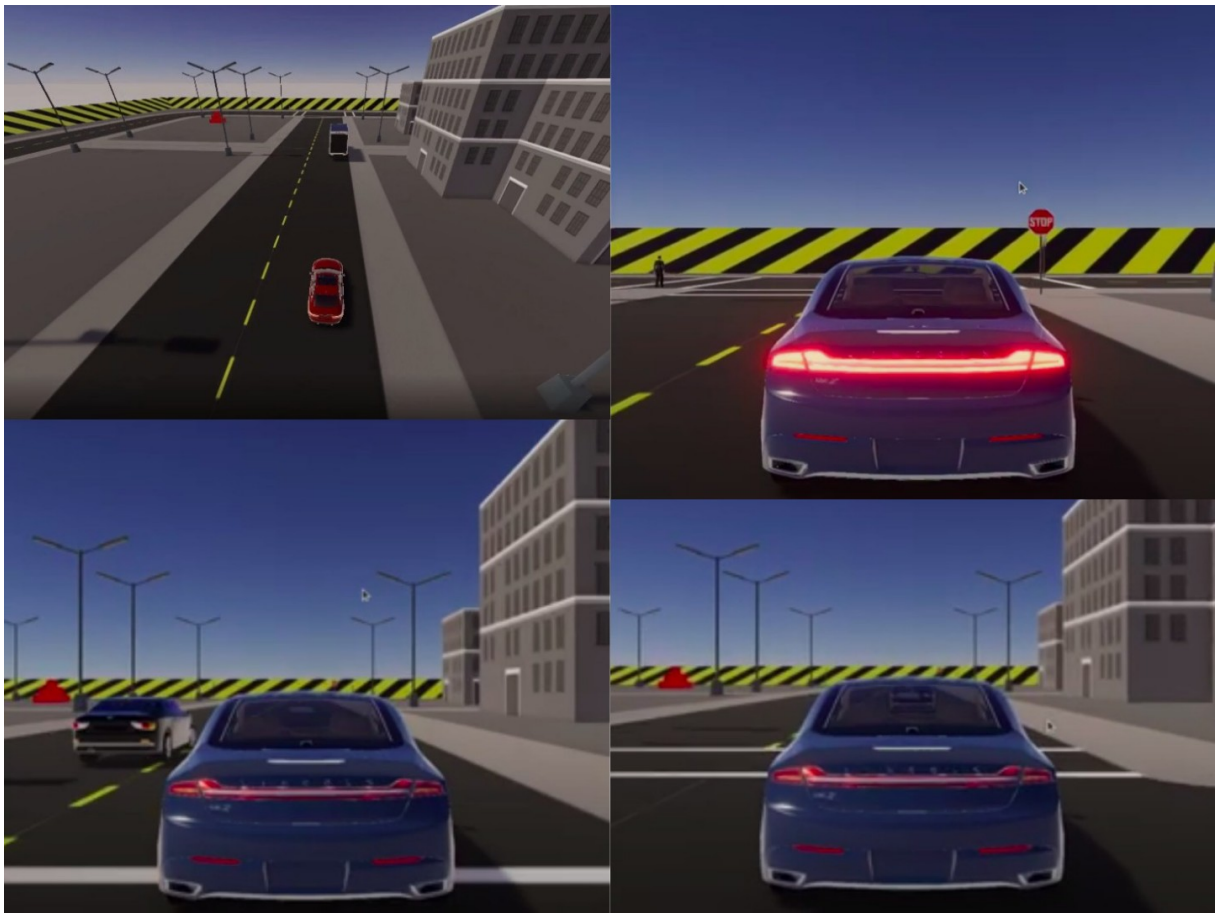


Figure 64 – Test Cases. Up: Static Truck(left), Crossing the Crosswalk (right). Bottom: Cut-in(Left), Stop and Go(Right). Source: Own Authorship

6.3.1 Scenario Remarks

A video from the test cases was recorded and is available (Carvalho, 2020).

In this research, all the trajectory planners tested the first scenario (concrete scenario). The AV's driving behavior for each trajectory was evaluated in Section 7.9. Unfortunately, unpredictable behavior occurred due to severe failures in the perception module and unexpected behavior of the decision-making module. The critical scenarios are under simulation tests. In the "Stop and Go" test case, the perception does not recognize the dynamic vehicle and drive straight the road, crossing the obstacle. On the "Cut-in" test case, the decision-making module chooses the right side from the dynamic obstacle to overtake it, and the AV climbs over the curb, as displayed in Figure 65.

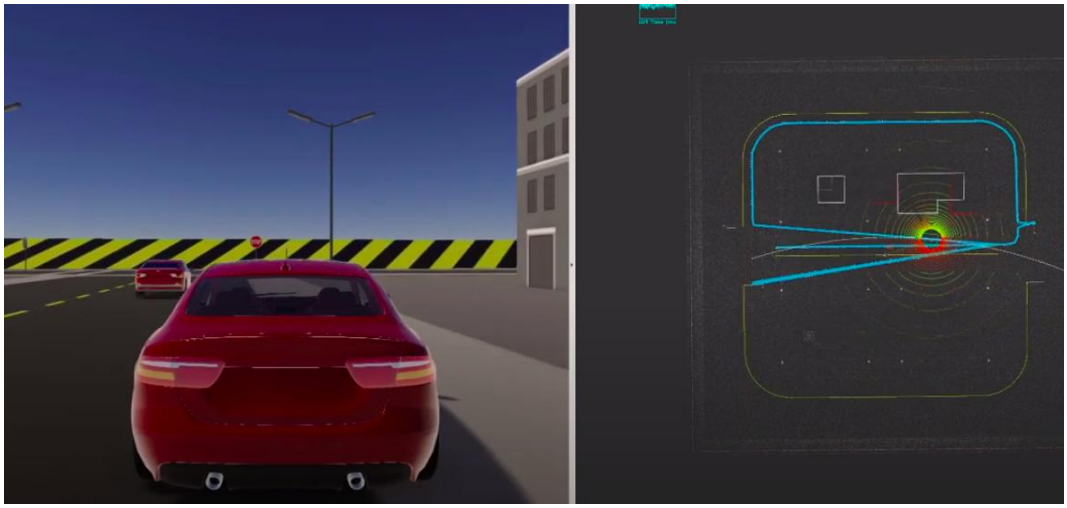


Figure 65 – Unexpected wrong choice taken by decision-making module in "Cut-in" test case: Ego climbs the curb. Source: Own Authorship

These failures and behaviors need to be studied more deeply to tune the parameters or fix the errors to test these challenging scenarios in a future opportunity.

6.4 Autoware.AI

6.4.1 Vehicle Configuration

The AWF foundation made available the urdf from the following vehicles: Lexus, Prius, and Jaguar. The Jaguar was selected for the simulation on this framework since it already has ROS1 bridge configured. The vehicle configuration from Autoware.AI as well as Autoware.Auto and Apollo.Auto follows the JSON format. And the amount and types of sensors and their respective parameters can be set in the LGSVL web interface. It is in this configurator interface and in JSON format [2], where the sensors parameters can be set, as well as the topics or channels in which they subscribe and publish messages. A sample of the description above can be visualized in Appendix A.5.

6.4.2 Tested Scenarios

Autoware.AI currently has seven different HD maps available on its official repository. Most of them were simulated in this work, especially Borregas Avenue, Cube Town, and Shalun. The Autoware.AI localization module proved effective in these maps. However, it did not work correctly with extensive and more complex HD maps such as San Francisco Borregas Avenue. For the final simulation selected scenario, the Cube Town map was

[2] JavaScript Object Notation is an open standard file format and data interchange format that uses human-readable text to store and transmit data objects consisting of attribute-value pairs and arrays

adopted. The reason is that this map contains a two-way highway available for overtaking maneuvers, as well as fewer traffic lights and signs information, which at-the end interfere in vehicle behavior and decision-making module significantly. As the focus of this survey is path planning, the other scenarios prejudice this evaluation once the vehicle got stuck or presented weird behaviors influenced by the decision-making module.

Negative influences from traffic lights during the simulations are displayed below. The AV stops infinitely behind the NPC agent with the right side of the highway available for lane changing and goes straight to the agent's rear, colliding with it (instead of stopping or changing the route) respectively.

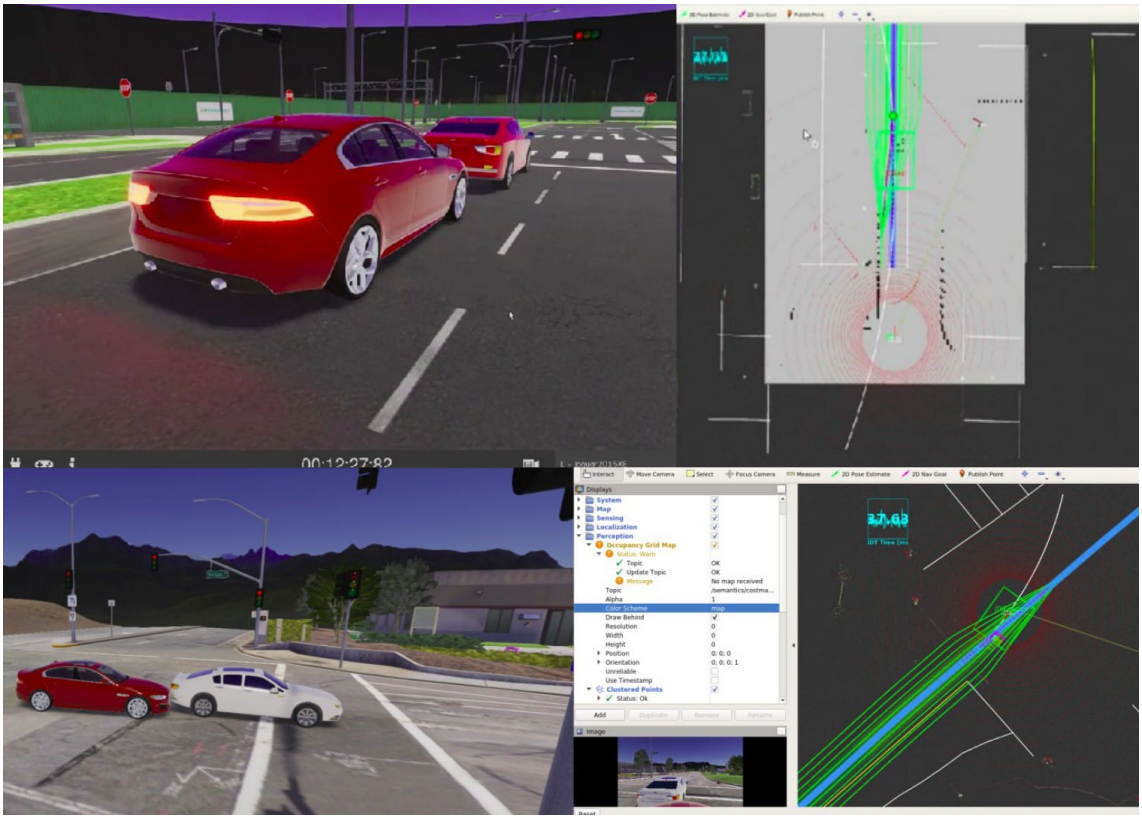


Figure 66 – Scenario Influence on Decision-Making Module

Source: Own Authorship

6.5 Autoware.Auto

6.5.1 Vehicle Configuration

For Autoware.Auto, the Lexus vehicle model, was selected once it had been already configured with the ROS2 bridge and its urdf was also available. In this way, this was the chosen model in the LGSVL simulator.

6.5.2 Tested Scenarios

As described in the section 5.3, the Autoware.Auto project was launched to cover specific ODDs. Until this work, only the first ODD had been implemented yet: the AVP. For this reason, the AWF has made available just the "Autonomous Stuff" PCD map. This map has numerous parking spots in which the vehicle can park. The Figure 67 displays a piece of this map while the AV global planner is guiding it to the parking spot:

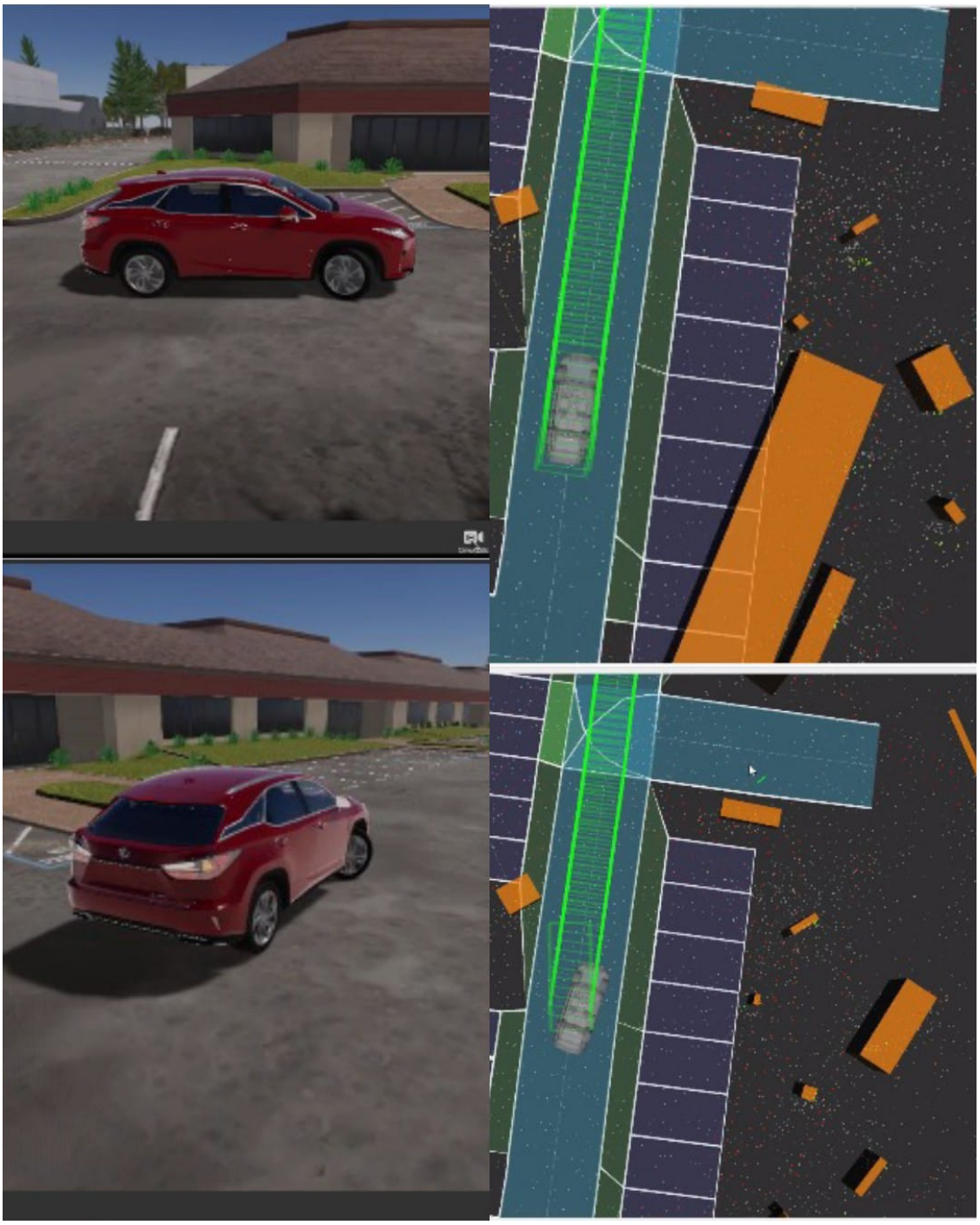


Figure 67 – Lanelet2, behavior and lane planners in action

Source: Own Authorship

In Figure 67 (up), the Lexus vehicle is controlled by the lanelet2 global planner and follows the produced trail (in green). Simultaneously, the behavior planner is activated and calculates a possible park spot (bottom from the previous figure), switching to the park planner when recognizing a candidate (green rectangle trying to "escape" the rail). As the goal spot is not the close parking spot available, Lexus returns to the rail center and keeps its search.

Finally, when the AV is closest to the goal spot, the behavior planner triggers the parking planner and vehicle parks. At first, a rear parking maneuver was tested. Then later, a front parking maneuver was tested. These maneuvers are depicted in Figure 68.

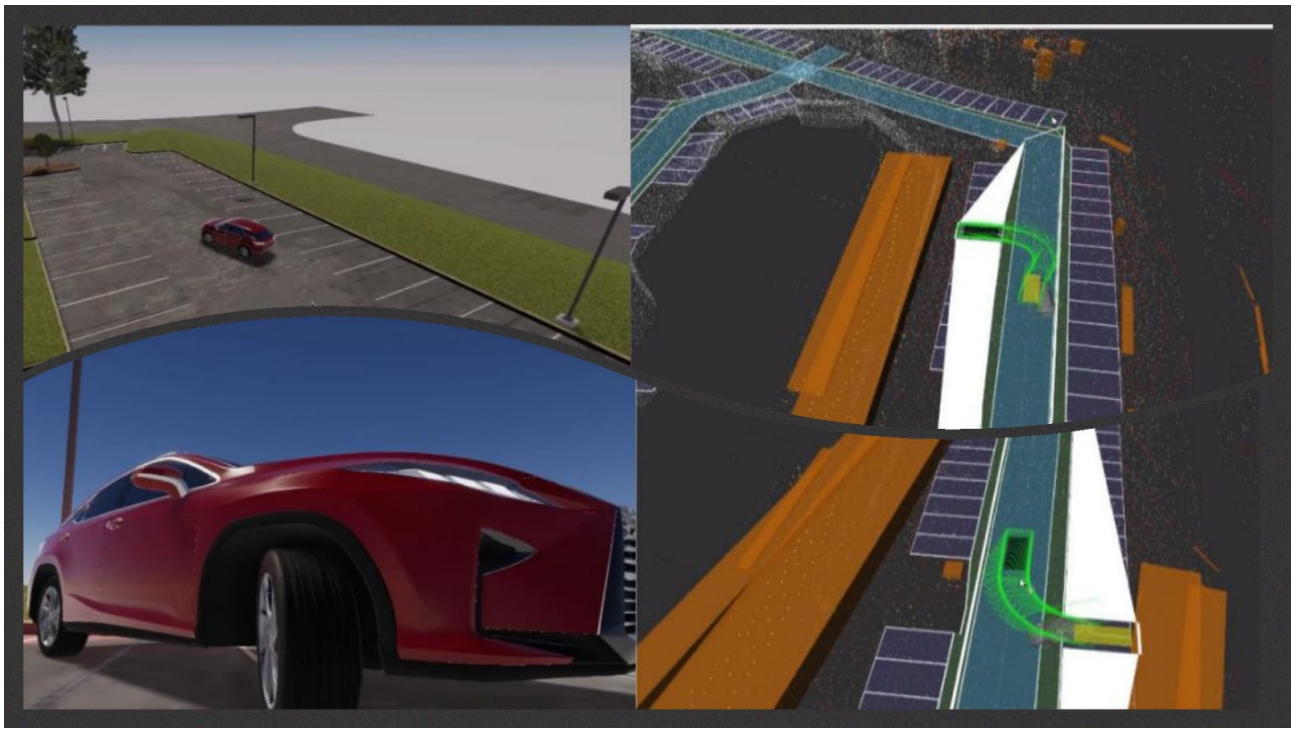


Figure 68 – Parking Maneuvers Tests. Up: Front Park, Bottom: Rear Park

Source: Own Authorship

Several simulations were needed to park the car with success. The global planner often found the goal and reached it. However, the parking planner occasionally failed to find the proper orientation to park successfully.

6.6 Apollo.Auto

6.6.1 Vehicle Configuration

In Apollo.Auto the Lincolns MKZ vehicle was selected for the simulations since it was already configured with a Cyber-RT bridge.

6.6.2 Tested Scenarios

In contrast with Autoware.AI, Apolo.Auto proved to be very robust regarding the localization module. It can navigate through the environments without the vehicle losing itself in the map, or the localization fails. For this reason, more simulations in different sorts of scenarios were possible. The AV was able to perform overtake maneuvers and travel long distances in the most extensive and complex LG landscape available: The San Francisco city map. Figure 69 displays an overtake maneuver into this city.

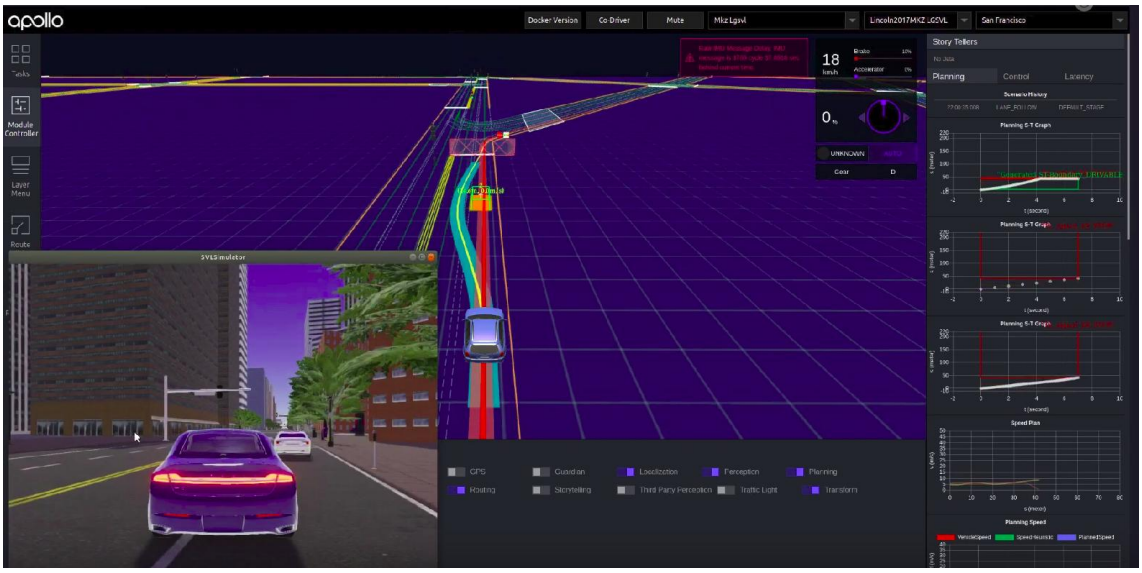


Figure 69 – Lane Change to avoid collision with a static vehicle on Apollo.Auto

Source: Own Authorship

The use of other available planners by Apollo.Auto. The standard planner called "Public Road" by Apollo Team optimized EM planner features was simulated. The Lattice planner was also tested; however, the module failed due to a lack of computational resources. So further analysis from this planner within this framework was not possible.

7 Simulation Assessment and Discussion

In this section, the simulated scenarios executed with the previous autonomous driving stacks are assessed in terms of relevant parameters in the robotics context (Gillani et al., 2016). The sensors raw data extracted from the LGSVL bridge, as well as the data science(Wikipedia, 2020) methods(codes), applied to process this data, are available on Github:Master_Thesis_Simulations.

This section brings plotted charts from meaningful data extracted for all available path planners of each AD stack, presented in the section 5, except for Autoware.Auto. These data describe the vehicle's behavior at each Timestamp along its journey and cooperate with the metrics visualization and evaluation. The data refers to the first test-case scenario, described in Section 6.3.

7.1 Trajectory

The final trajectory calculated by the applied path planners is discussed and visually demonstrated in the following subsections.

7.1.1 AutowareAI Trajectories Planners

The following subsections depict the application of the trajectory planners available in the open-source ADS used in this work. The planners were applied in a simulation environment (see Section 6.2), demonstrating what the behavior of the real vehicle would be for each generated trajectory.

7.1.1.1 Open Planner

Open Planner let a safety distance from the obstacle, and the vehicle decision-making forced it to switch the lane, as shown in Figure 70.

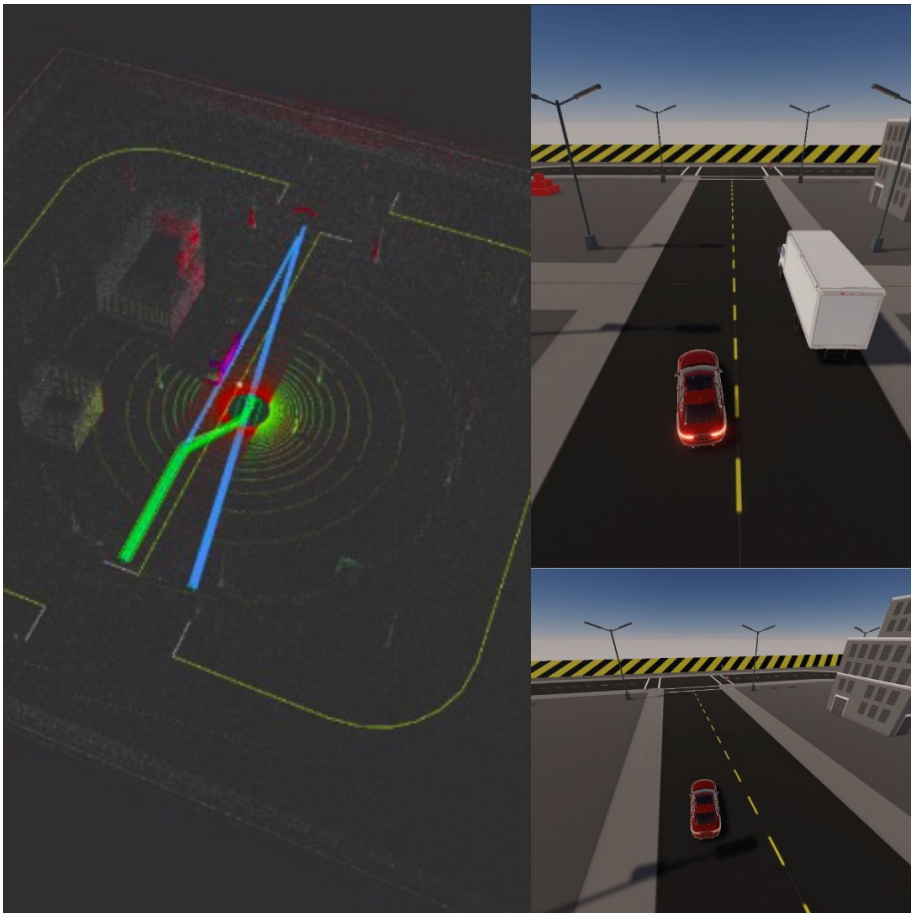


Figure 70 – Open Planner Calculated Trajectory

Source: Own Authorship

7.1.1.2 Freespace

The Freespace global planner calculates free waypoints over the obstacle. A sketch of its behavior can be visualized in Figure 71.

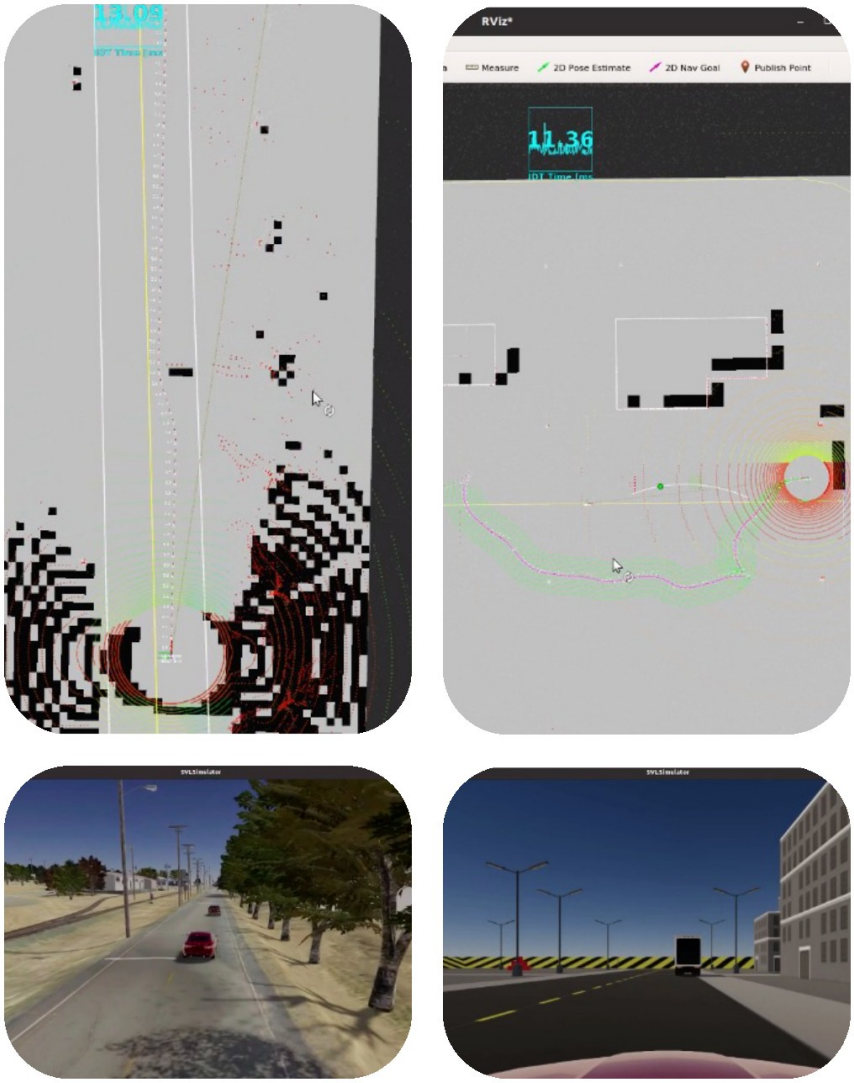


Figure 71 – Freespace Trajectory Generation. Source: Own Authorship

Two different local planners were simulated with Freespace: Astar and Lattice

7.1.1.2.1 Local Planner: Astar

Figure 72 displays the executed trajectory by Freespace Global planner working with A* as local Planner.

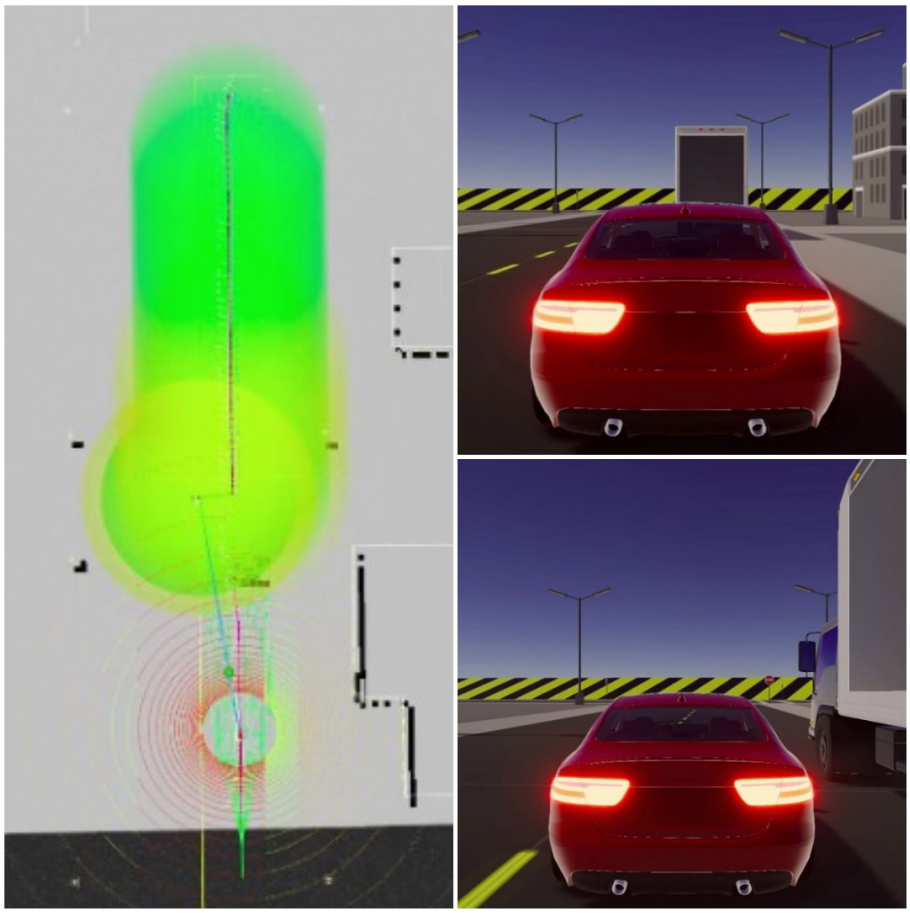


Figure 72 – Astar Deforming Freespace Global Planner Trajectory

Source: Own Authorship

7.1.1.2.2 Local Planner: Lattice

Figure 73 displays the executed trajectory by Freespace Global planner working with Lattice as a local Planner.

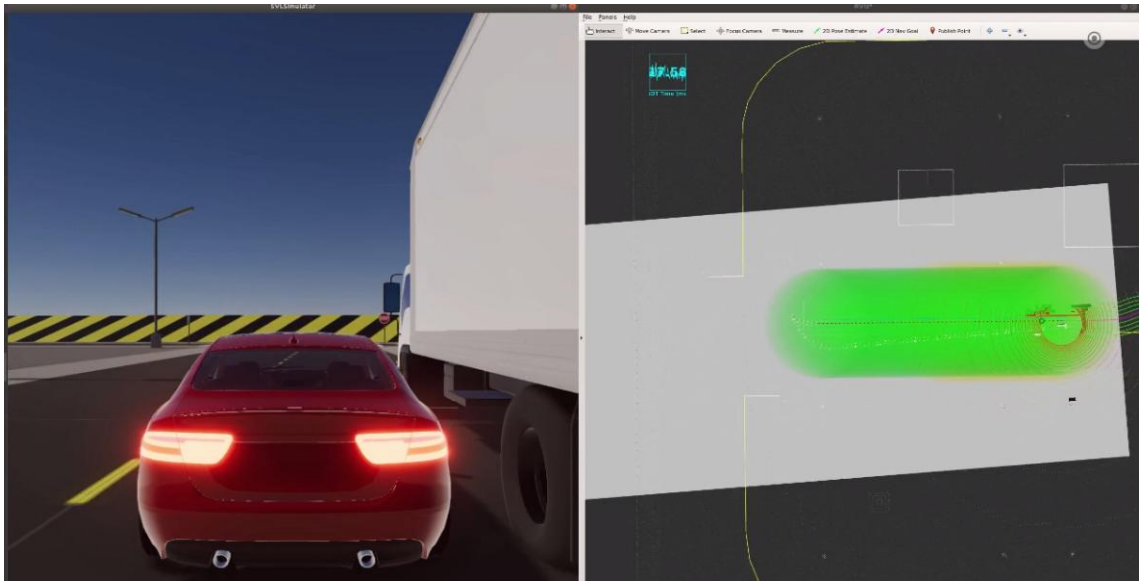


Figure 73 – Lattice Deforming Freespace Global Planner Trajectory

Source: Own Authorship

7.1.1.3 Public Road

The Public Road path planner behavior is shown in Figure 74.

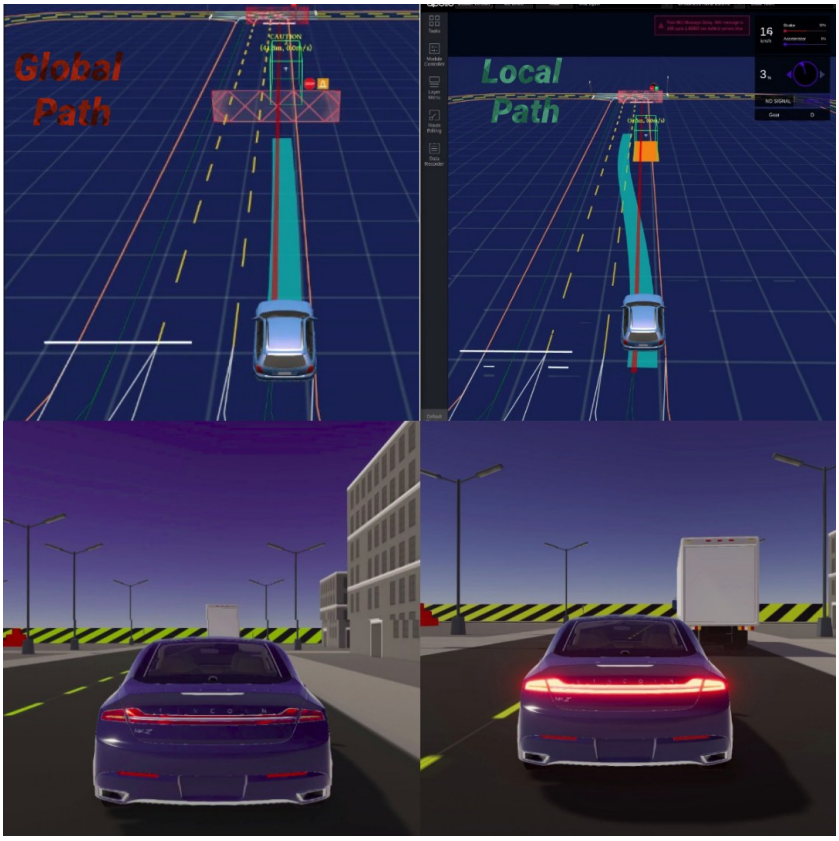


Figure 74 – Public Road Planner Trajectory. Source: Own Authorship

The previous trajectories can be checked in the figure

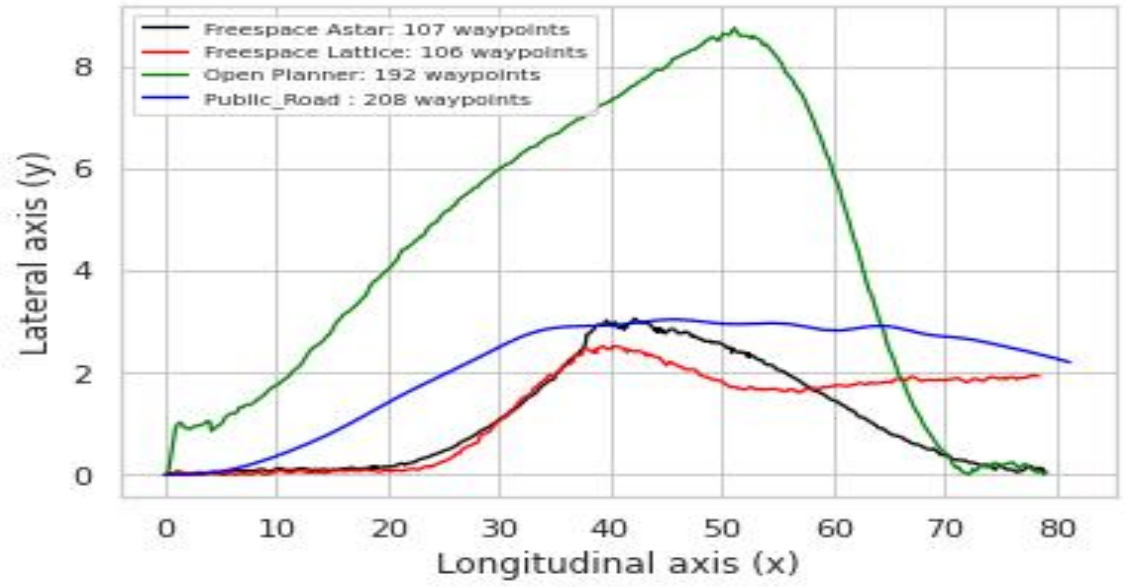


Figure 75 – Trajectory profile from simulated planners. Source: Own Authorship

7.2 Lateral Displacement Region

As presented in the trajectories profiles, in Figure 75, the path planners have generated paths with different lateral offsets from the obstacle. The portion of this offset in the y-axis was calculated. This information represents the path planning distortion from the original global way (a straight line on the x-axis).

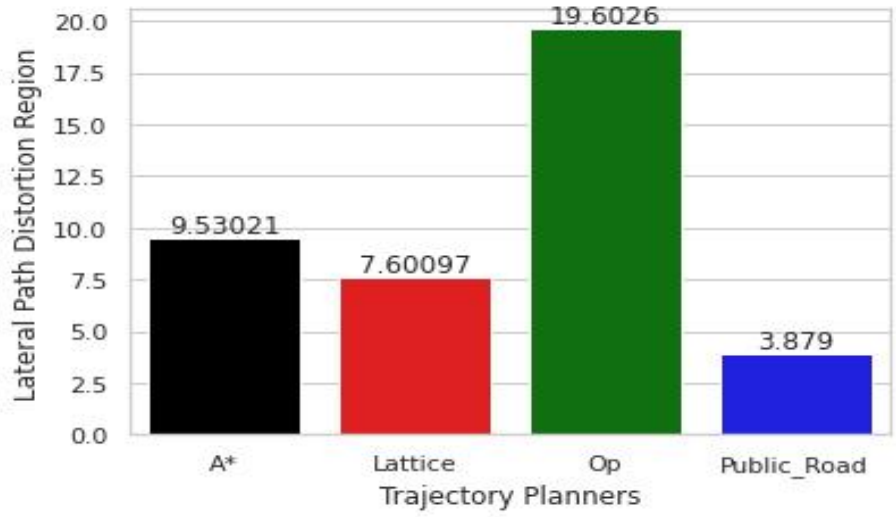


Figure 76 – Lateral Displacement from Global Path. Source: Own Authorship

7.3 Distance Traveled

Figure 77 displays the full path traveled by the AV. The distance was calculated by setting the minimum x distance traveled by all the path planners. That adjustment was needed due to different threshold and goal points sent to each Planner. This slight deviation occurs because the goal point is sent manually in the Autonomous Driving Software Interface.

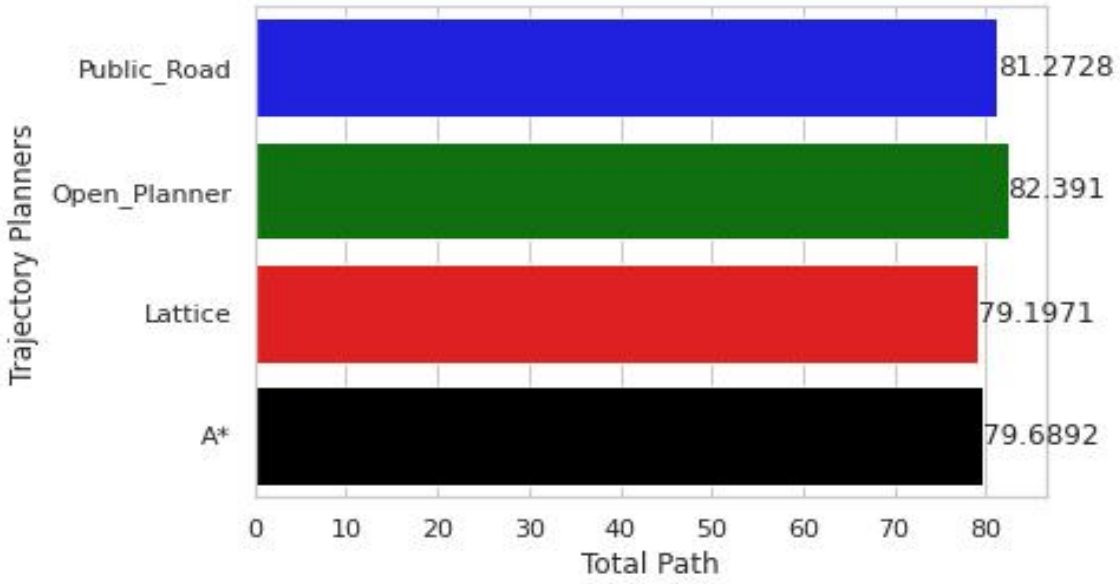


Figure 77 – Total Distance Traveled. Source: Own Authorship

As expected, the Open Planner produced the longest path. Compared to the other trajectory planners, the wide distortion explains this from the global Planner, which created a more considerable margin and lateral distance from the obstacle. Contrarily, the Freespace Lattice has produced the minor path in exchange for a minimum safety lateral distance from the obstacle.

7.4 Distance From Obstacle

An indicator from the safety is the minimum distance in which a trajectory planner stays from the obstacle along its trajectory (Ferrer Sánchez et al., 2018; Gillani et al., 2016).

Figure 78 represents the distance the AV remained from the obstacle at each waypoint, along with the persecution of the trajectory.

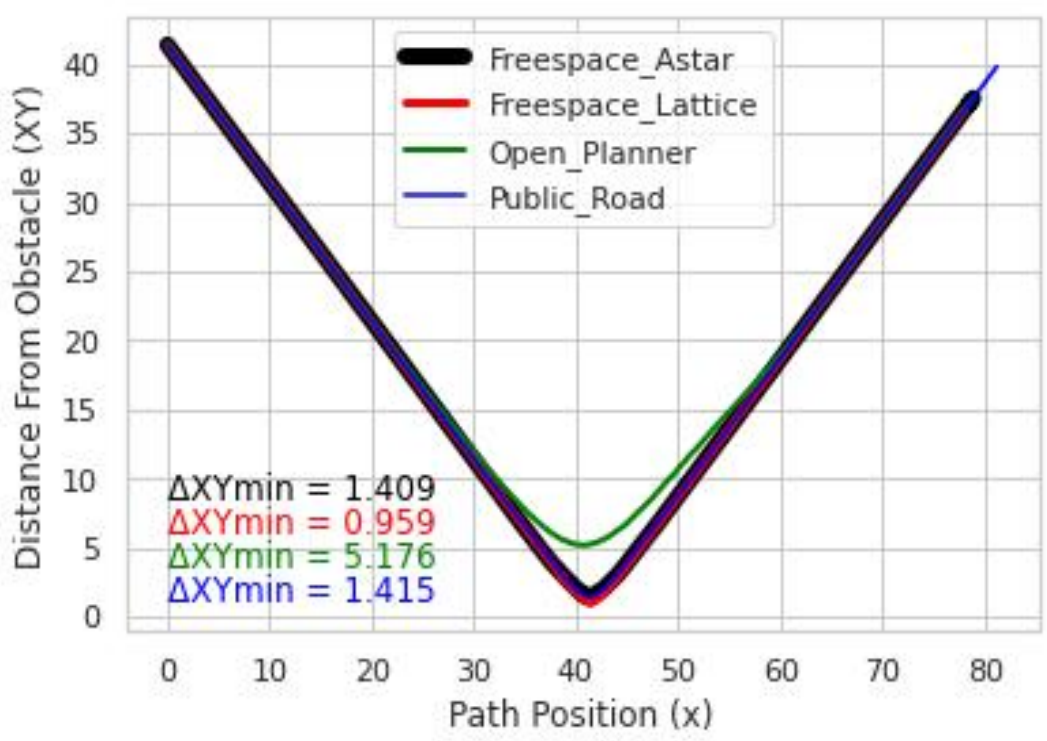


Figure 78 – Distance From Obstacle. Source: Own Authorship

The obstacle position and dimensions were extracted from the Velodyne data using regex; the reference point selected was the Truck rear left (RL) point from the rectangular model, demonstrated in Figure 79, as it is the close point the AV remains before the obstacle avoidance maneuver.

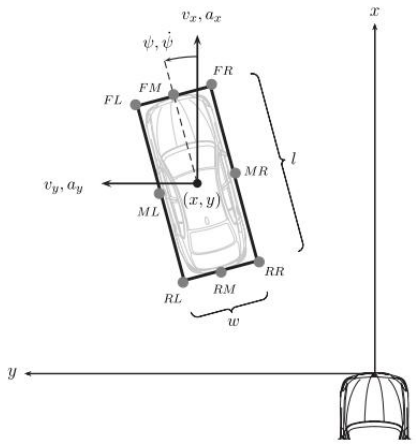


Figure 79 – Rear Left (RL) point selection from planar rectangular object model. Source: Aeberhard (2017).

7.5 Smoothness of Trajectory

Smoothness: The smoothness $\mathcal{S}$ of a trajectory can be measured as a function of jerk, the time derivative of acceleration:

$$\mathcal{T}(t) = \frac{da(t)}{dt} \tag{11}$$

For a given trajectory $\mathcal{T}$, the smoothness is defined as the sum of squared Jerk along $\mathcal{T}$.

$$\mathcal{S} = \int_{ti}^{tf} \mathcal{T}(t)^2 dt \tag{12}$$

where t_i and t_f are the initial and final time, respectively (Gillani et al., 2016). The smoothness of the trajectory is related to the rider's comfort during the trip (Caesar et al., 2021).

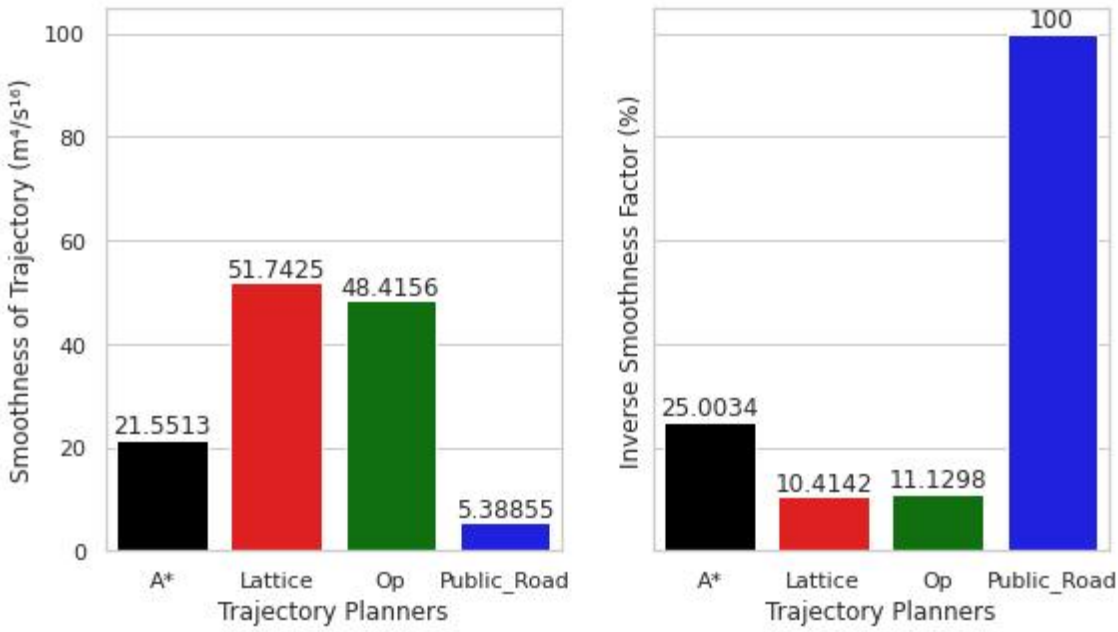


Figure 80 – Smoothness of Trajectory (left). Relational Factor of Smoothness (right).

Source: Own Authorship

The Power of Jerk gives the smoothness of a trajectory. Jerk is found taking the derivative of the robot's acceleration. From Figure 80, it is possible to check that the worst smooth trajectory is from Lattice and the smoother generated by the Public Road algorithm. Therefore the highest smoothness value, the less smooth is the trajectory. As more Jerky is, the trajectory is more difficult for the vehicle to travel along with it due to kynodynamic constraints. As a consequence, the AV spends more energy during the

displacement. Jerky contributes to power consumption directly. It is a relevant factor but not a determinant. Suppose we check the power consumption graph 84, it is clear that OP and Lattice spent more energy rather than A*. However, Open Planner's low Jerky and smoothness value consumed more Power. This happened because the Public Road planner executed the trajectory in higher velocities and spent less than half of the time accomplishing its trajectory than the other planners. This factor contributed to the increase in the Power consumed, directly related to the velocity.

7.6 Evasive Maneuver Duration

Total Simulation Time: The time the path planner takes to generate the path added by the time the robot travels through this planned path.

The Simulation Time is the measurement the path planner takes to generate the path added by the time the vehicle travels through this planned path. The time taken to persecute the trajectory on each is a crucial measurement indicator because it demonstrates how well the path fits the vehicle dynamics (Peralta et al., 2020). Unfortunately, path generation time was not available during the data process. However, this test case does not significantly influence the Simulation time benchmark, once this time is usually shorter than a second.

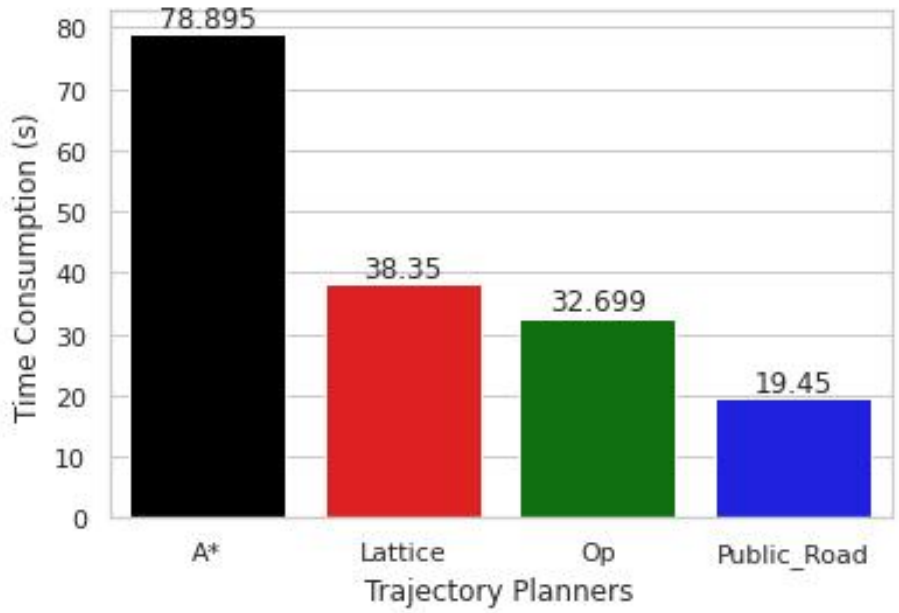


Figure 81 – Time Consumption to accomplish the generated trajectories

Source: Own Authorship

7.7 Power Consumption

The Energy Consumption that the AV spent persecuting the trajectory. While the AV is moving, it continuously changes its steering angle (α) and its Instantaneous Center of Rotation (ICR). This factor influences the calculation of the curvature radius (R). For this reason, the power consumption needs to be measured at each vehicle timestamp.

The parameters above are shown in Figure 82.

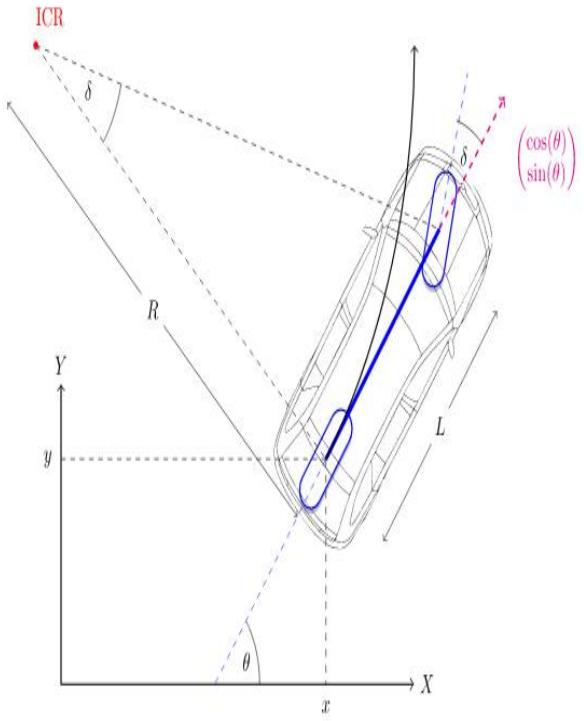


Figure 82 – Vehicle Orientation in the bicycle kinematic model. Source: Krishna (2020).

The total amount of power consumed $\mathcal{P}$ by the AV to move from start to the goal state is computed as:

$$\mathcal{P} = \sum_{i}^{n} \frac{f_i d_i}{\Delta t_i} \tag{13}$$

where f_i,d_i and t_i are the force, displacement and time respectively.

At first, the force at each trip instant was calculated:

$$\mathcal{F}_l = Mass \times acc_{lin} \tag{14}$$

$$\mathcal{F}_r = Mass \times acc_{net} \tag{15}$$

where $acc_{lin} = a_x$, since the velocity component in y axis is equals to 0 (extracted data from the ROS topic current_velocity and divided by the delta in time between sequential waypoints). The resultant rotational acceleration from the AV is given by:

$$acc_{net} = \sqrt{a_{tan}^2 + a_{cent}^2} \tag{16}$$

The tangential acceleration is given by:

$$a_{tan} = \alpha \times r \tag{17}$$

The centripetal acceleration is given by:

$$a_{cen} = \omega^2 \times r \tag{18}$$

where and ω^2 is the Angular Velocity and r is given by the AV's Ackermann Steering Model:

$$r = L \times tan(\alpha) \tag{19}$$

Where L is the robot's wheelbase (given by LGSVL Unix Engine) and α is the steering angle (yaw angle) at each Timestamp (extracted from ROS topic imu_raw in quaternions format and converted to roll, pitch, yaw). Figure 83 depicts the Ackermann steering geometry.

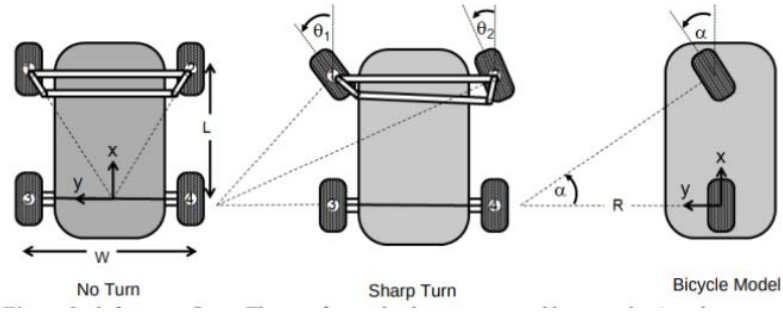


Figure 83 – Steering Angle (yaw) for an Ackermann Steering Geometry. Source: Kelly (2010).

Found the a_net acceleration, it is decomposed in x and y components. The a_net x component is summed with the acc_lin x component and then the resultant acceleration is found:

$$a_{res} = (a_{netx} + a_{linx})^2 + (a_{nety})^2 \tag{20}$$

Finally, the force can be found, multiplying the vehicle Mass by the a_{res}

$$F = M \times a_{res} \tag{21}$$

The vehicle Mass is given by the LGSVL simulator Unix engine. The last term, distance traveled at each Timestamp, is extracted from the ROS topic current_pose. Having all these data, the Instantaneous Power(W) is obtained for each waypoint, dividing the AV's work()J) by the delta time(s). And the total power consumed by the vehicle is found, as displayed in the first equation.

The average power consumption when the vehicle is accelerating, and the friction consumption when the car is braking, were calculated and are shown in Figure 84.

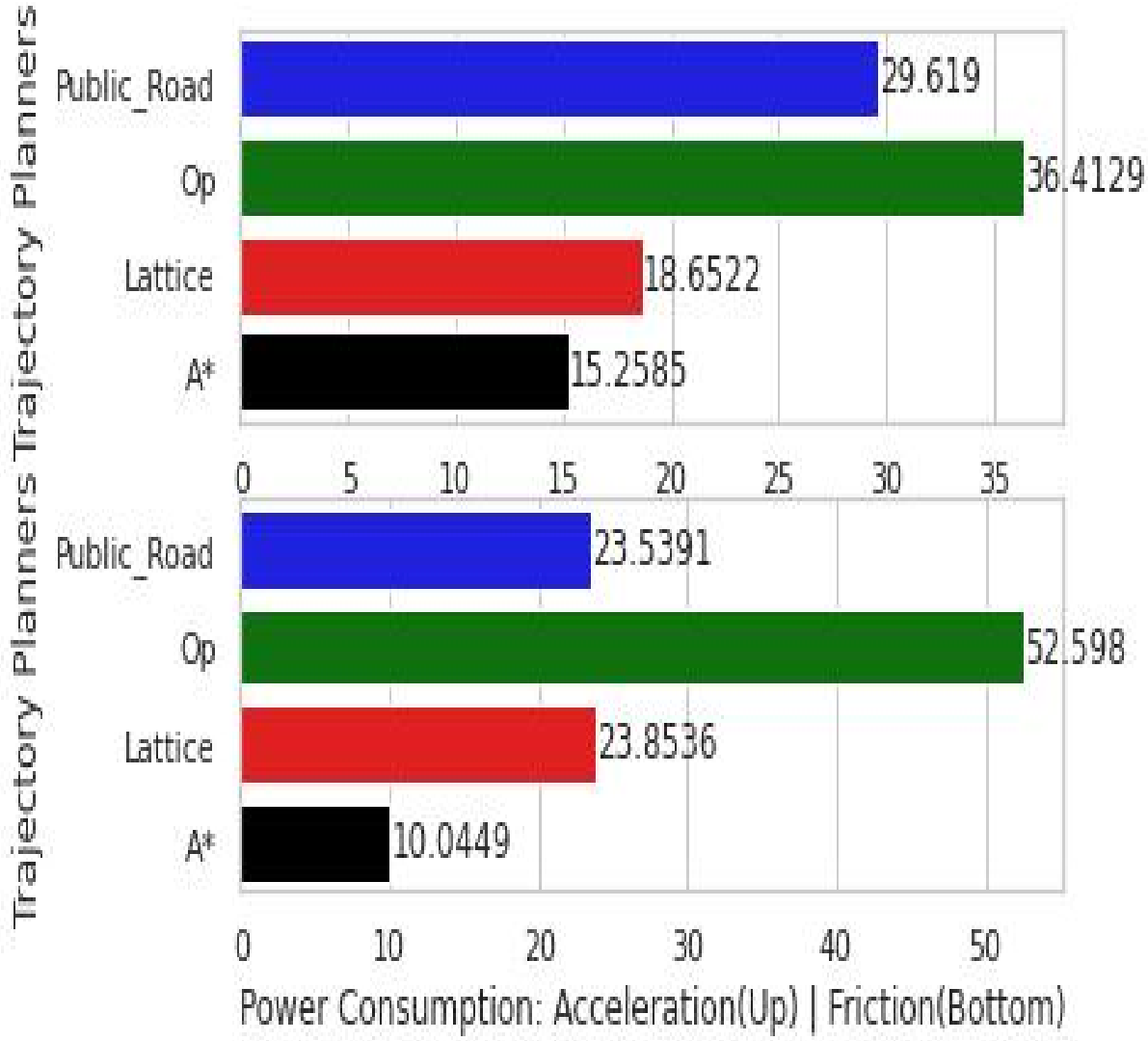


Figure 84 – The average Power consumed (HorsePower) by the car-like robot while moving along the solution path and overall average. Source: Own Authorship

7.8 Persecution of Trajectory: Path deviation

While the AV tries to follow the trajectory, the AV does not follow it with 100 percent accuracy. A path error is caused and is directly related to the control module efficiency. The figures below display the errors in XY pose and heading while following this trajectory.

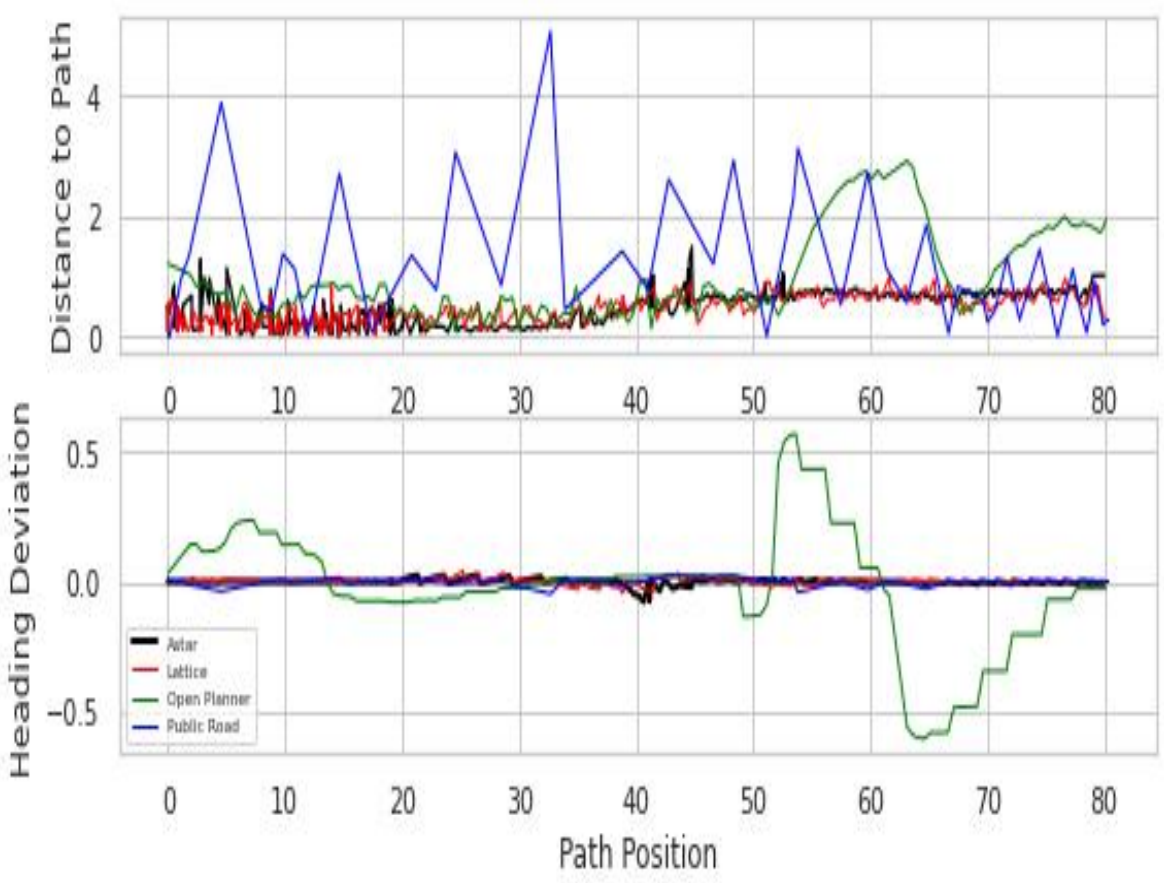


Figure 85 – Distance To Path Error: Position (Up) and Heading (Bottom)

Source: Own Authorship

The Public Road presented the more significant deviation in following the x, y waypoints produced by the trajectory planners at each Timestamp. A possible reason for this behavior is the most considerable speed that the vehicle developed while following the generated trajectory, bringing more hardness for the controller module. The Open Planner has also presented a substantial deviation from the planned path, mainly in the heading component. The more significant deviations coincided with the beginning of the trip when it turned abruptly in the CCW direction to avoid the obstacle. At the end of the journey, it turns back in CW's direction to return to the road's right lane. The deviations from the path are closely related to the control layer. This module, however, is not the focus of this work. Other works involving AD stack controllers discuss typical controllers' performance on self-driving maneuvers, such as MPC and Pure Pursuit (Karunainayagam, 2020).

7.9 Vehicle Behavior

The AV changed the velocity, acceleration, and heading during the trip. These features are highlighted below:

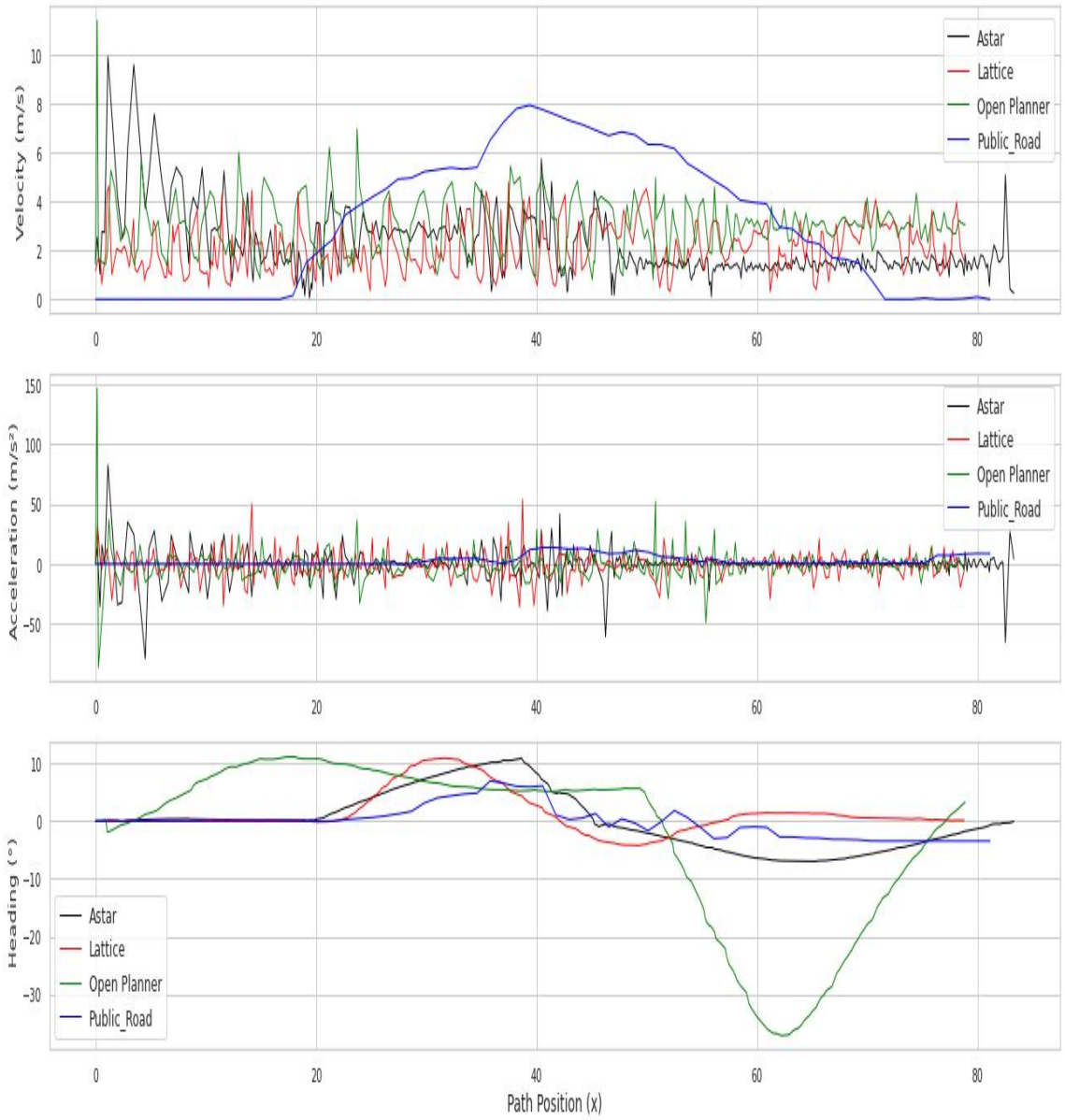


Figure 86 – Vehicle Behavior Source: Own Authorship

.

The Public Road planner achieved more significant velocities than the other planners, which collaborated directly to reach the more considerable power consumption. The Lattice planner presented the most oscillation behavior during the continuous time, showing a less smooth trajectory. Regarding the heading, Open Planner performed the most significant deviation from the vehicle center of rotation while switching the lane.

7.10 Computational Effort

The simulation involved the usage of numerous modules, packages, and libraries. Each Planner has its own needs and therefore requires a different amount of Random Access Memory (RAM), Graphics Process Unit (GPU), and Central Process Units (CPU) cores. The graphics in Figure 87 display the amount of usage for each Planner:

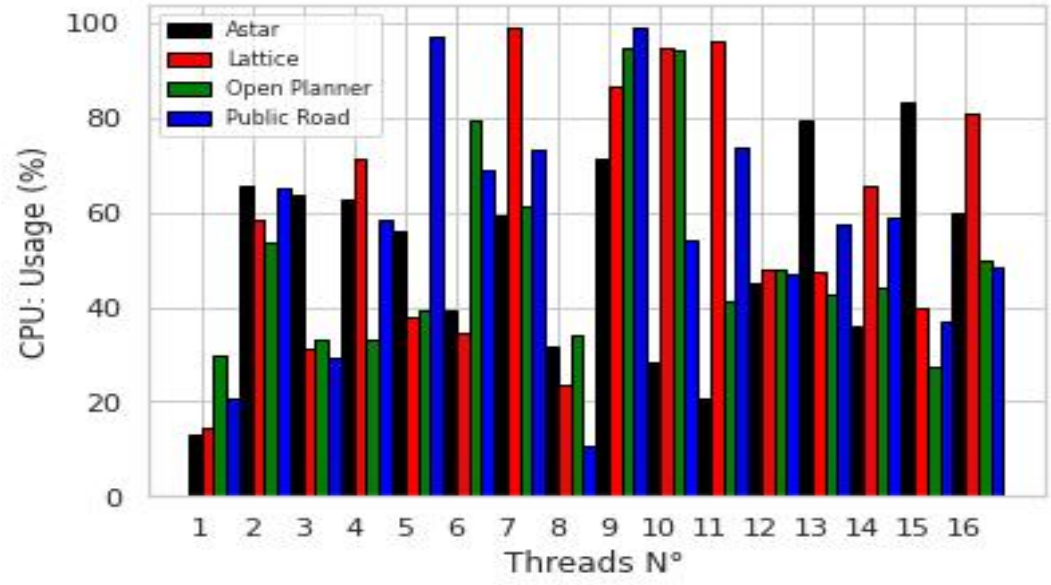


Figure 87 – Trajectory Planner's CPU usage for each thread. Source: Own Authorship

The median and mean from the CPU consumption were also calculated:

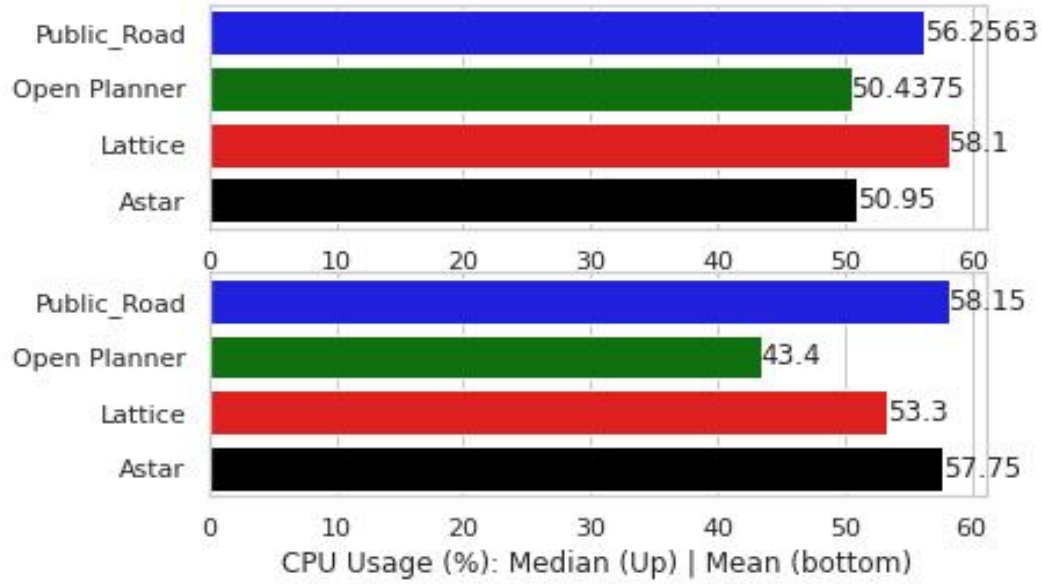


Figure 88 – Trajectory Planner's average of CPU usage. Source: Own Authorship

Using the respective trajectory planners, the RAM consumed by each framework can be visualized in Figure 89.

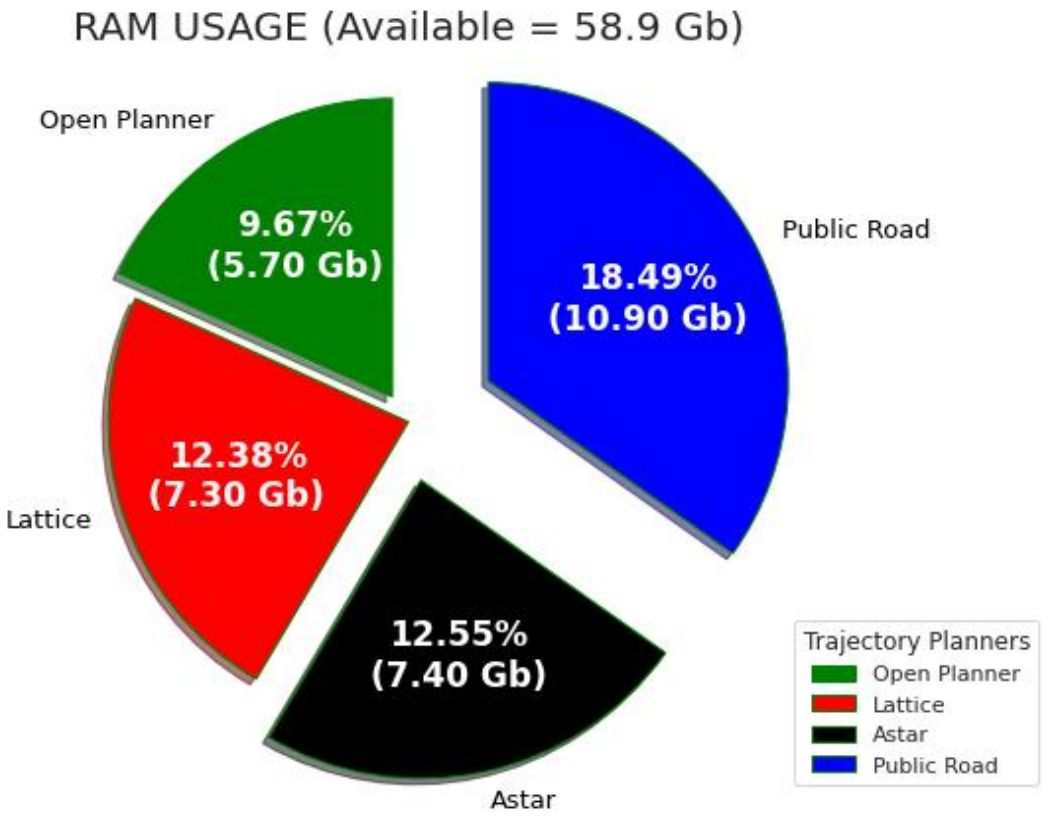


Figure 89 – Trajectory Planner's Memory consumption. Source: Own Authorship

The Public Road from Apollo.Auto autonomous driving stack was the planning module, along with the other modules (perception, localization, controlling), which has required more memory (RAM) and arithmetical computational effort (CPU) to function. The Autoware platform required less computational effort, however, Apollo.Auto AD stack proved to be more robust in planning and localization. Autoware presented more failures during the simulation and was unable to maintain the vehicle localization working in all the scenarios and presented more failures during the planning process. Even so, Autoware demonstrated better performance in the perception module than Apollo.Auto 6.0. In Apollo AD Stack, a modular perception was needed to be activated within the LGSVL simulator once it could not demonstrate a robust behavior in detection using its perception module.

8 Conclusion and future work

This work covered the path planners applied for self-driving cars using open source platforms based on ROS, from Navigation Stack to the most popular Autonomous Driving Stacks: Autoware.AI, Autoware.Auto and Apollo.Auto. The integration with a powerful simulator from LGSVL was also done, representing realistic scenarios for assessment of the AD stacks modules, with a focus on planning. The behavior from the path planners and a complete benchmark involving the evaluation of numerous parameters were also realized. A benchmark from the open-source AD platforms was also performed and is shown in Table 3. Analogously, a comparative Table from the applied trajectory planners is available in Table 4.

Table 3 – Autonomous Driving Stack Performance

Parameters	**Autoware**	**Apollo**
Modules Functionality		
Mapping	Moderate	High
Localization	Low	High
Perception	High	Moderate
Planning	Moderate	High
Control	Moderate	High
Quality Factor Raju et al. (2019)		
Distributed System	High	High
Maturity	Moderate	Moderate
Performance	Moderate	High
Robustness	Moderate	High
Project Development		
Modularity	High	Moderate
Complexity	High	Moderate
Developer Manual	Low	Moderate
Simulation Tools	High	High

Source: Own Author, adapted from Raju et al. (2019).

From Table 3 it is possible to check that the Autoware.AI project lack robustness in important modules, such as **localization**. As stated in Section 6.4.2, this ADS could not handle vehicle mapping and localization in several HD-Maps provided by the LGSVL simulator. In this regard, Apollo proved to be more robust, and the Simultaneous Localization and Mapping (SLAM) algorithms were able to perform the **mapping** and the vehicle localization in all tested maps. In addition, Apollo provides to the user a more friendly user interface, the Dreamview (Baidu-Apollo, 2020). This interface makes available dynamic graphics, contributing to monitoring and debugging purposes. In the **perception** aspect, however Apollo.Auto6.0 failed to perceive and track the obstacles in the simulated scenarios (see Section 6.3). To overcome this issue a modular perception was

needed (LGsvl-Doc, 2020). Autoware.AI perception algorithms did not fail, and the object detection and tracking could be optimized by tuning parameters of the costmap generator package. In trajectory **planning and control** aspects, and Apollo demonstrated a more consistent and regular behavior. The Autoware.AI trajectory planners neither planned the path nor executed the same path for the same simulated test cases. For these reasons, Apollo.Auto ADS can be considered more **robust** and with a better **performance**.

Both platforms lack on presenting a complete **usage manual**, Apollo.Auto, however, has more active support on GitHub, in which the developer can get more information. Evaluating the **modularity**, the Autoware project, specially Autoware.AI, is one step ahead of Apollo. As we can check in Runtime Manager Interface, Autoware.AI provides many different ROS packages and algorithms for the autonomous driving modules, which can be switched. For example, in the control module, it is possible to choose between MPC or Pure-pursuit control. At the same time, Apollo 6.0 fixed the usage for MPC, requiring complex code implementation to enable alternative algorithms to work with its modules and the Cyber-Rt system. Another detailed evaluation from these aspects was performed (Raju et al., 2019) and analyzed in their paper for those interested in more comparison parameters from this benchmark.

In Table,4 a score from 1 to 5 is assigned to the trajectory planners, in which one corresponds to lousy performance, and five corresponds to optimum performance. The evaluated parameters are based on the section 7 results. From this table, it is possible to check that Open Planner generated the most **safety trajectory** in terms of lateral distance (see section 7.2) and minimum distance left from the obstacle (see Section 7.4). this safety margin however increased the total displacement of the vehicle (**path length**). The **passenger comfort** parameters were assessed based on the trajectory smoothness (see Section 7.5) and vehicle jerk data (see Section 7.9). As the table demonstrates, the Freespace Lattice had the worst performance, once it presented the worst value for smoothness of trajectory (see Section 7.5) and had high oscillation in acceleration profile (see Section 7.9). To the **Time Trip** aspect, a direct relation was taken from Section 81 to assign the Path Planner grades. To measure the **feasibility of trajectory** the data from Section 7.8 was taken into account since the frenet coordinates (s,d) and the vehicle's heading angle deviation are directly related to how suitable were the waypoints generated by the planner to the control module track. As the Open Planner and Public Road had the most significant deviation in this topic, the lower notes were assigned to them. In terms of **computational effort**, while the Public Road was being executed, it reached the most prominent memory and CPU usage consumption. The Open Planner had the lowest values for these metrics, presenting less computation burden. This metric, however, is not just related to the trajectory planner usage, but all the modules from the ADS (from mapping to control) being executed at the simulation running time. This measurement is also a good indicator of the ADS computational requirements. Since the Public Road is

the trajectory planner implemented by Apollo.Auto ADS, it is possible to infer that this framework requires higher computational effort than Autoware.AI in exchange for offering more robustness, as displayed in the table 3.

Table 4 – Trajectory Planners Performance

Evaluated Metrics	**Freespace A***	**Freespace Lattice**	**Open Planner**	**Public Road**
Path Length	5	5	2	4
Safety Maneuver (Collision Risk 7.4)	2	1	5	2
Passenger Comfort	3	1	2	4
Time trip	1	3	3	5
Trajectory Feasibility	3	4	2	2
Computational Effort	3	3	4	1

Source: Own Authorship

Testing the AV in diverse scenarios and accomplishing specific AD functions, such as parking, was achieved. Although the initial goals from this work have been reached, it is desired for the next work, to try the AD stacks in more critical scenarios and use more specific technical performance metrics for autonomous driving. Another relevant goal is to test the AD Stack and automated driving function in a real vehicle to assess its behavior in the real environment.

Bibliography

Aaron and Huang (2019), 'Rapidly exploring random tree (rrt) and rrt*'.
URL: *http://demonstrations.wolfram.com/RapidlyExploringRandomTreeRRTAndRRT/*

Ackerman (2016), 'The electronic highway: How 1960s visionaries presaged today's autonomous vehicles'.
URL: *https://spectrum.ieee.org/cars-that-think/transportation/self-driving/the-electronic-highway-of-1969*

Aderinola, B. (2020), '[ros in 5 mins] 034 – what is ros action?'.
URL: *https://www.theconstructsim.com/ros-5-mins-034-ros-action/*

Aeberhard, M. (2017), *Object-level fusion for surround environment perception in automated driving applications.*

Aeberhard, M., Rauch, S., Bahram, M., Tanzmeister, G., Thomas, J., Pilat, Y., Homm, F., Huber, W. and Kaempchen, N. (2015), 'Experience, results and lessons learned from automated driving on germany's highways', *IEEE Intelligent transportation systems magazine* 7(1), 42–57.

Audi, A. and Volkswagen, A. (2020), 'Description of the pegasus-method'.

AutowareAuto git (2021), 'Implement semantic-map-based navigation and planning'.
URL: *https://gitlab.com/autowarefoundation/autoware.auto/AutowareAuto/-/issues/447*

AWF (2021), 'Self-driving cars with ros and autoware'.
URL: *https://www.autoware.org/awf-course*

Badue, C., Guidolini, R., Carneiro, R. V., Azevedo, P., Cardoso, V. B., Forechi, A., Jesus, L., Berriel, R., Paixao, T. M., Mutz, F. et al. (2021), 'Self-driving cars: A survey', *Expert Systems with Applications* 165, 113816.

Bahram, M., Ghandeharioun, Z., Zahn, P., Baur, M., Huber, W. and Busch, F. (2014), Microscopic traffic simulation based evaluation of highly automated driving on highways, *in* '17th International IEEE Conference on Intelligent Transportation Systems (ITSC)', IEEE, pp. 1752–1757.

Baidu (2021), 'Apollo 3.5 software architecture'.
URL: *https://github.com/ApolloAuto/apollo/blob/master/docs/specs/Apollo*$_3$.5$_S oftware_A rchitectu$

Baidu-Apollo (2020), 'Dramview usage table'.
URL: *https://github.com/ApolloAuto/apollo/blob/master/docs/specs/dreamview$_u$sage$_t$able.md*

Baiee, H. A., AL-Araji, K. and Mohammed, A. J. (2020), 'Road traffic fatalities in babylon province-six years epidemiologic study.', *Indian Journal of Forensic Medicine & Toxicology* 14(1).

Battiston, A. (2015), 'Software in c++ for communication between can bus and ros in a robot vehicle'.

Beasley (2019), 'Taking the lead in self-driving cars'.
URL: *https://car.osu.edu/news/2019/08/taking-lead-self-driving-cars*

Borenstein, J. and Koren, Y. (1989), 'Real-time obstacle avoidance for fast mobile robots', *IEEE Transactions on systems, Man, and Cybernetics* 19(5), 1179–1187.

Bosch (2021), 'Intelligent headlight control'.
URL: *https://www.bosch-mobility-solutions.com/en/solutions/assistance-systems/intelligent-headlight-control/*

Brock, O. and Khatib, O. (1999), High-speed navigation using the global dynamic window approach, *in* 'Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)', Vol. 1, IEEE, pp. 341–346.

Buehler, M., Iagnemma, K. and Singh, S. (2009), *The DARPA urban challenge: autonomous vehicles in city traffic*, Vol. 56, springer.

Caesar, H., Kabzan, J., Tan, K. S., Fong, W. K., Wolff, E., Lang, A., Fletcher, L., Beijbom, O. and Omari, S. (2021), 'nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles', *arXiv preprint arXiv:2106.11810* .

Caliskan, M. (2020), 'State estimation for localization'.
URL: *https://gitlab.com/ApexAI/autowareclass2020/-/tree/master/lectures/10$_L$ocalization*

CARandDriver (2021), 'What is park assist? quick guide'.
URL: *https://www.caranddriver.com/research/a32814141/park-assist/ /*

Carvalho, M. (2020), 'Test cases for autonomous driving'.
URL: *https://www.youtube.com/watch?v=mSIJ8c$_o$eQolist* $=$ $PL_{dY}iRmUA3nnTRmF8QJKyPoEQhrdvpP2index = 11$

Chakravarthy, A. and Ghose, D. (1998), 'Obstacle avoidance in a dynamic environment: A collision cone approach', *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 28(5), 562–574.

Chen, C.-T., Quinn, R. D. and Ritzmann, R. E. (1997), A crash avoidance system based upon the cockroach escape response circuit, *in* 'Proceedings of International Conference on Robotics and Automation', Vol. 3, IEEE, pp. 2007–2012.

Clemente, E., Meza-Sánchez, M., Bugarin, E. and Aguilar-Bustos, A. Y. (2018), 'Adaptive behaviors in autonomous navigation with collision avoidance and bounded velocity of an omnidirectional mobile robot', *Journal of Intelligent & Robotic Systems* 92(2), 359–380.

Connete, C. (2019), 'Path optimization by elastic band'.
URL: *https://www.youtube.com/watch?v=KJgHAhJxUr0t=28s*

Corsaro, Angelo; Strahm, S. (2020), 'Autoware course lecture 4: Platform hw, rtos and dds'.
URL: *https://www.youtube.com/watch?v=IyycN6ldsIst=5465s*

Darweesh, H., Takeuchi, E., Takeda, K., Ninomiya, Y., Sujiwo, A., Morales, L. Y., Akai, N., Tomizawa, T. and Kato, S. (2017), 'Open source integrated planner for autonomous navigation in highly dynamic environments', *Journal of Robotics and Mechatronics* 29(4), 668–684.

Dempster, A. P., Laird, N. M. and Rubin, D. B. (1977), 'Maximum likelihood from incomplete data via the em algorithm', *Journal of the Royal Statistical Society: Series B (Methodological)* 39(1), 1–22.

Dolgov, D. (2016), 'Google self-driving car project monthly report', *Published September* .

Dolgov, D., Thrun, S., Montemerlo, M. and Diebel, J. (2010), 'Path planning for autonomous vehicles in unknown semi-structured environments', *The international journal of robotics research* 29(5), 485–501.

Elektrobit (2021), 'Autonomous driving - the future of mobility'.
URL: *https://www.elektrobit.com/trends/autonomous-driving/*

Embotech (2021), 'Hierachical architecture in autonomous driving'.
URL: *https://gitlab.com/ApexAI/autowareclass2020/-/blob/master/lectures/12*$_M$*otion*$_C$*ontrol/1*$_a$*rchitecture*$_A$*D.pdf*

Erke, S., Bin, D., Yiming, N., Qi, Z., Liang, X. and Dawei, Z. (2020), 'An improved a-star based path planning algorithm for autonomous land vehicles', *International Journal of advanced Robotic Systems* 17(5), 1729881420962263.

Faconti, D. (2010), 'Plot juggler: Juggle with data'.
URL: *https://github.com/facontidavide/PlotJuggler*

Fan, H., Zhu, F., Liu, C., Zhang, L., Zhuang, L., Li, D., Zhu, W., Hu, J., Li, H. and Kong, Q. (2018), 'Baidu apollo em motion planner', *arXiv preprint arXiv:1807.08048* .

Feitosa (2021), '5g no brasil: quando chega? precisa trocar de celular? veja respostas'.
URL: *https://g1.globo.com/economia/tecnologia/noticia/2021/11/02/leilao-do-5g-acontece-nesta-semana-quando-chega-precisa-trocar-de-celular-veja-respostas.ghtml*

Ferrer Sánchez, J. et al. (2018), 'Implementation and comparison in local planners for ackermann vehicles'.

fjp.github.io (2018), 'Trajectory planning in the frenet space'.
URL: *https://fjp.at/posts/optimal-frenet/*

Flessner, D. (2020), Investigation and Implementation of Available Software and Algorithms for Autonomous Vehicle Development, PhD thesis, The Ohio State University.

Ford (2021), 'What is ford traffic sign recognition'.
URL: *https://www.tch.co.uk/about/why-choose-ford/traffic-sign-recognition/*

Fox, D., Burgard, W. and Thrun, S. (1997), 'The dynamic window approach to collision avoidance', *IEEE Robotics & Automation Magazine* 4(1), 23–33.

Gillani, M., Akbari, A. and Rosell, J. (2016), Physics-based motion planning: Evaluation criteria and benchmarking, *in* 'Robot 2015: Second Iberian Robotics Conference', Springer, pp. 43–55.

Génération ROBOTS (2019), 'Ros vs ros2'.
URL: *https://www.generationrobots.com/blog/en/ros-vs-ros2/*

Hale, A., Stoop, J. and Hommels, J. (1990), 'Human error models as predictors of accident scenarios for designers in road transport systems', *Ergonomics* 33(10-11), 1377–1387.

He, Y., Chen, C., Bu, C. and Han, J. (2015), 'A polar rover for large-scale scientific surveys: design, implementation and field test results', *International Journal of Advanced Robotic Systems* 12(10), 145.

Hella (2021), 'Lighting systems'.
URL: *https://www.hella.com/hella-si/en/Headlamps-202.html*

Honda (2021), 'Tire pressure monitoring system (tpms) with tire fill assist (select models)'.
URL: *https://www.hondainfocenter.com/Shared-Technologies/Safety/Tire-Pressure-Monitoring-System-TPMS-with-Tire-Fill-Assist/: :text=The%20Feature%3A,displayed%20in%20the%20instrument%20panel.*

Huang (2019), 'Ros tutorial: Publishers and subscribers'.
URL: *https://www.youtube.com/watch?v=bJB9tv4ThV4list=PL$_{dY}$iRmUA3kP488BvwEw5r7aaha*
$4t = 357s$

Iturrate, I., Antelis, J. M., Kubler, A. and Minguez, J. (2009), 'A noninvasive brain-actuated wheelchair based on a p300 neurophysiological protocol and automated navigation', *IEEE Transactions on Robotics* 25(3), 614–627.

Joseph, L. (2015), *Mastering ROS for robotics programming*, Packt Publishing Ltd.

Juliussen (2020), 'Germany leads the autonomous vehicle regulation race'.
URL: *https://www.eetimes.eu/germany-leads-the-autonomous-vehicle-regulation-race/*

Kalra, N. and Paddock, S. M. (2016), 'Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?', *Transportation Research Part A: Policy and Practice* 94, 182–193.

Kamon, I., Rivlin, E. and Rimon, E. (1996), A new range-sensor based globally convergent navigation algorithm for mobile robots, *in* 'Proceedings of IEEE International Conference on Robotics and Automation', Vol. 1, IEEE, pp. 429–435.

Karunainayagam, N. (2020), Entwicklung und Bewertung von Methoden zur Trajektorienplanung für die Simulation automatisierter Fahrfunktionen, PhD thesis, Technische Hochschule Ingolstadt.

Kato, S., Tokunaga, S., Maruyama, Y., Maeda, S., Hirabayashi, M., Kitsukawa, Y., Monrroy, A., Ando, T., Fujii, Y. and Azumi, T. (2018), Autoware on board: Enabling autonomous vehicles with embedded systems, *in* '2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)', IEEE, pp. 287–296.

Kavrakilab (2020), 'The open motion planning library app'.
URL: *https://ompl.kavrakilab.org/*

Kelly, A. (2010), A vector algebra formulation of kinematics of wheeled mobile robots, *in* 'International conference on Field and Service Robotics', pp. 1–14.

Khatib, M., Jaouni, H., Chatila, R. and Laumond, J.-P. (1997), Dynamic path modification for car-like nonholonomic mobile robots, *in* 'Proceedings of International Conference on Robotics and Automation', Vol. 4, IEEE, pp. 2920–2925.

Klück, F., Li, Y., Nica, M., Tao, J. and Wotawa, F. (2018), Using ontologies for test suites generation for automated and autonomous driving functions, *in* '2018 IEEE International symposium on software reliability engineering workshops (ISSREW)', IEEE, pp. 118–123.

Konolige, K. (2000), A gradient method for realtime robot control, *in* 'Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)(Cat. No. 00CH37113)', Vol. 1, IEEE, pp. 639–646.

Krishna, K. A. R. A. Y. (2020), 'Mobile robot kinematics'.
URL: *http://www.cs.cmu.edu/ rasc/Download/AMRobots3.pdf*

Kuwata, Y., Teo, J., Fiore, G., Karaman, S., Frazzoli, E. and How, J. P. (2009), 'Real-time motion planning with applications to autonomous urban driving', *IEEE Transactions on Control Systems Technology* 17(5), 1105–1118.

Le, D., Liu, Z., Jin, J., Zhang, K. and Zhang, B. (2019), Historical improvement optimal motion planning with model predictive trajectory optimization for on-road autonomous vehicle, *in* 'IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society', Vol. 1, IEEE, pp. 5223–5230.

Lee, C. W., Nayeer, N., Garcia, D. E., Agrawal, A. and Liu, B. (2020), Identifying the operational design domain for an automated driving system through assessed risk, *in* '2020 IEEE Intelligent Vehicles Symposium (IV)', IEEE, pp. 1317–1322.

LGsvl-Doc (2020), 'Modular testing'.
URL: *https://www.svlsimulator.com/docs/archive/2020.06/modular-testing/*

LGsvl git (2021), 'Lgsvl simulator github issues'.
URL: *https://github.com/lgsvl/simulator*

LGsvl Python API (2020), 'Python api guide'.
URL: *https://www.svlsimulator.com/docs/python-api/python-api/*

Li, H., Tsukada, M., Nashashibi, F. and Parent, M. (2014), 'Multivehicle cooperative local mapping: A methodology based on occupancy grid map merging', *IEEE Transactions on Intelligent Transportation Systems* 15(5), 2089–2100.

Likhachev, M. and Ferguson, D. (2009), 'Planning long dynamically feasible maneuvers for autonomous vehicles', *The International Journal of Robotics Research* 28(8), 933–945.

Longo, Stefano;Merkli, S. (2020), 'Motion planning and control'.
URL: *https://www.youtube.com/watch?v=fQJpAVRQBrIt=3s*

Lumelsky, V. J. and Skewis, T. (1990), 'Incorporating range sensing in the robot navigation function', *IEEE Transactions on Systems, Man, and Cybernetics* 20(5), 1058–1069.

Lumelsky, V. J. and Stepanov, A. A. (1987), 'Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape', *Algorithmica* 2(1-4), 403–430.

Macenski, S., Martín, F., White, R. and Clavero, J. G. (2020), The marathon 2: A navigation system, *in* '2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)', IEEE, pp. 2718–2725.

Marin-Plaza, P., Hussein, A., Martin, D. and Escalera, A. d. l. (2018), 'Global and local path planning study in a ros-based research platform for autonomous vehicles', *Journal of Advanced Transportation* 2018.

Mark, A. R., James, A. S. and William, A. H. (2010), 'Principles of modern radar: Basic principles'.

Masuri, M. G., Isa, K. A. M. and Tahir, M. P. M. (2012), 'Children, youth and road environment: Road traffic accident', *Procedia-Social and Behavioral Sciences* 38, 213–218.

McKinsey (2019), 'Development in the mobility technology ecosystem—how can 5g help?'.
URL: *https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/development-in-the-mobility-technology-ecosystem-how-can-5g-help*

McKinsey and Company (2019), 'The future of mobility is at our doorstep'.
URL: *https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/the-future-of-mobility-is-at-our-doorstep*

McNaughton, M., Urmson, C., Dolan, J. M. and Lee, J.-W. (2011), Motion planning for autonomous driving with a conformal spatiotemporal lattice, *in* '2011 IEEE International Conference on Robotics and Automation', IEEE, pp. 4889–4895.

Mínguez, J. and Montano, L. (2002), 'Robot navigation in very complex, dense, and cluttered indoor/outdoor environments', *IFAC Proceedings Volumes* 35(1), 397–402.

Minguez, J. and Montano, L. (2004), 'Nearness diagram (nd) navigation: collision avoidance in troublesome scenarios', *IEEE Transactions on Robotics and Automation* 20(1), 45–59.

Minguez, J., Montano, L. and Khatib, O. (2002), Reactive collision avoidance for navigation with dynamic constraints, *in* 'IEEE/RSJ International Conference on Intelligent Robots and Systems', Vol. 1, IEEE, pp. 588–594.

Moll, M., Şucan, I. A., Bordeaux, J. and Kavraki, L. E. (2011), Teaching motion planning concepts to undergraduate students, *in* 'Advanced Robotics and its Social Impacts', IEEE, pp. 27–30.

Moll, M., Sucan, I. A. and Kavraki, L. E. (2015), 'Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization', *IEEE Robotics & Automation Magazine* 22(3), 96–102.

Montiel, O., Orozco-Rosas, U. and Sepúlveda, R. (2015), 'Path planning for mobile robots using bacterial potential field for avoiding static and dynamic obstacles', *Expert Systems with Applications* 42(12), 5177–5191.

MoveIt Applications (2021), 'Moveit applications'.
URL: *https://moveit.ros.org/documentation/applications/*

MoveIt Planners (2021), 'Moveit planners'.
URL: *https://moveit.ros.org/documentation/planners/*

Musk, E. (2019), 'Elon musk on cameras vs lidas for self driving and autonomous cars'.
URL: *https://www.youtube.com/watch?v=HM23sjhtk4Q*

Najafi, S. and Arghami, S. (2020), 'Predictive assessment of driver errors using human error template (het)', *Iran Occupational Health* 16(6), 53–65.

Nalic, D., Mihalj, T., Bäumler, M., Lehmann, M., Eichberger, A. and Bernsteiner, S. (2020), Scenario based testing of automated driving systems: A literature survey, *in* 'Proc. FISITA Web Congr.', p. 30.

Nayoga University (2016), *Autoware User's Manual*, Nayoga University.

Nitsche, P., Welsh, R. H., Genser, A. and Thomas, P. D. (2018), A novel, modular validation framework for collision avoidance of automated vehicles at road junctions, *in* '2018 21st International Conference on Intelligent Transportation Systems (ITSC)', IEEE, pp. 90–97.

Noh, S. (2019), 'Decision-making framework for autonomous driving at road intersections: Safeguarding against collision, overly conservative behavior, and violation vehicles', *IEEE Transactions on Industrial Electronics* 66(4), 3275–3286.

Paul, N. and Chung, C. (2018), 'Application of hdr algorithms to solve direct sunlight problems when autonomous vehicles using machine vision systems are driving into sun', *Computers in Industry* 98, 192–196.

Peralta, F., Arzamendia, M., Gregor, D., Reina, D. G. and Toral, S. (2020), 'A comparison of local path planning techniques of autonomous surface vehicles for monitoring applications: The ypacarai lake case-study', *Sensors* 20(5), 1488.

Petrovskaya, A. and Thrun, S. (2009), 'Model based vehicle detection and tracking for autonomous urban driving', *Autonomous Robots* 26(2), 123–139.

Pivtoraiko, M., Knepper, R. A. and Kelly, A. (2009), 'Differentially constrained mobile robot motion planning in state lattices', *Journal of Field Robotics* 26(3), 308–333.

Plaku (2020), 'The open motion planning library app'.
URL: *https://ompl.kavrakilab.org/*

Poggenhans, F., Pauls, J.-H., Janosovits, J., Orf, S., Naumann, M., Kuhnt, F. and Mayr, M. (2018), Lanelet2: A high-definition map framework for the future of automated driving, *in* '2018 21st International Conference on Intelligent Transportation Systems (ITSC)', IEEE, pp. 1672–1679.

Ponn, T., Gnandt, C. and Diermeyer, F. (2019), An optimization-based method to identify relevant scenarios for type approval of automated vehicles, *in* 'Proceedings of the ESV—International Technical Conference on the Enhanced Safety of Vehicles, Eindhoven, The Netherlands', pp. 10–13.

Ponn, T., Schwab, A., Diermeyer, F., Gnandt, C. and Záhorskỳ, J. (2019), A method for the selection of challenging driving scenarios for automated vehicles based on an objective characterization of the driving behavior, *in* '9. Tagung Automatisiertes Fahren'.

Pütz, A., Zlocki, A., Bock, J. and Eckstein, L. (2017), 'System validation of highly automated vehicles with a database of relevant traffic scenarios', *situations* 1, E5.

Quinlan, S. and Khatib, O. (1993), Elastic bands: Connecting path planning and control, *in* '[1993] Proceedings IEEE International Conference on Robotics and Automation', IEEE, pp. 802–807.

Raju, V. M., Gupta, V. and Lomate, S. (2019), Performance of open autonomous vehicle platforms: Autoware and apollo, *in* '2019 IEEE 5th International Conference for Convergence in Technology (I2CT)', IEEE, pp. 1–5.

RENESAS (2020), 'Smart camera for automotive'.
URL: *https://www.renesas.com/us/en/application/automotive/adas/automotive-smart-camera*

RIA (2020), 'Path planning basics, online course implementation on robot ignite academy (ria) development studio'.
URL: *https://www.theconstructsim.com/robotigniteacademylearnros/ros − courses − library/path − planning − basics/*

Robotics Back-end (2018), 'What is a ros node?'.
URL: *https://roboticsbackend.com/what-is-a-ros-node/*

Rogic, B., Bernsteiner, S., Samiee, S., Eichberger, A. and Payerl, C. (2016), Konzeptionelle virtuelle absicherung von automatisierten fahrfunktionen anhand eines sae level 3 fahrstreifenwechselassistenten, *in* 'VDI/VW-Gemeinschaftstagung: Fahrerassistenz und automatisiertes Fahren'.

Rong, G., Shin, B. H., Tabatabaee, H., Lu, Q., Lemke, S., Možeiko, M., Boise, E., Uhm, G., Gerow, M., Mehta, S. et al. (2020), Lgsvl simulator: A high fidelity simulator for autonomous driving, *in* '2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)', IEEE, pp. 1–6.

ROS Answers (2020), '3d navigation for robot without arm'.
URL: *https://answers.ros.org/question/273447/moveit-with-ompl-for-mobile-platform-naviagtion/*

ROS Answers -MoveIt (2020), 'Moveit with ompl for mobile platform navigation'.
URL: *https://answers.ros.org/question/273447/moveit-with-ompl-for-mobile-platform-naviagtion/*

Rösmann, C., Feiten, W., Wösch, T., Hoffmann, F. and Bertram, T. (2013), Efficient trajectory optimization using a sparse model, *in* '2013 European Conference on Mobile Robots', IEEE, pp. 138–143.

Rostami, S. M. H., Sangaiah, A. K., Wang, J. and Kim, H.-j. (2018), 'Real-time obstacle avoidance of mobile robots using state-dependent riccati equation approach', *EURASIP Journal on Image and Video Processing* 2018(1), 79.

ROS Wiki (2020), 'Ros wiki documentation'.
URL: *http://wiki.ros.org/Documentation*

ROS Wiki- Lanelete2 (2020), 'Lanelet2'.
URL: *http://wiki.ros.org/lanelet2*

ROS Wiki Navigation (2018), 'Navigation stack setup'.
URL: *http://wiki.ros.org/navigation/Tutorials/RobotSetup*

Schlegel, C. (1998), Fast local obstacle avoidance under kinematic and dynamic constraints for a mobile robot, *in* 'Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No. 98CH36190)', Vol. 1, IEEE, pp. 594–599.

Schröder, E., Braun, S., Mählisch, M., Vitay, J. and Hamker, F. (2019), Feature map transformation for multi-sensor fusion in object detection networks for autonomous driving, *in* 'Science and Information Conference', Springer, pp. 118–131.

Schuldt, F., Saust, F., Lichte, B., Maurer, M. and Scholz, S. (2013), 'Effiziente systematische testgenerierung für fahrerassistenzsysteme in virtuellen umgebungen', *Automatisierungssysteme, Assistenzsysteme und Eingebettete Systeme Für Transportmittel* .

Sebastian Thrun, Wolfram Burgard, D. F. (2005), Occupancy grid mapping for an overview of how occupancy grids are generated, *in* 'Probabilistic robotics', MIT Press, Cambridge, MA, chapter 9, pp. 221–242.

Simmons, R. (1996), The curvature-velocity method for local obstacle avoidance, *in* 'Proceedings of IEEE international conference on robotics and automation', Vol. 4, IEEE, pp. 3375–3382.

Skolnik, M. I. (1962), 'Introduction to radar', *Radar handbook* 2, 21.

Snape, J., Van Den Berg, J., Guy, S. J. and Manocha, D. (2011), 'The hybrid reciprocal velocity obstacle', *IEEE Transactions on Robotics* 27(4), 696–706.

Sucan, I. A., Moll, M. and Kavraki, L. E. (2012), 'The open motion planning library', *IEEE Robotics & Automation Magazine* 19(4), 72–82.

Szeliski, R. (2010), *Computer vision: algorithms and applications*, Springer Science & Business Media.

Tellez, R. (2017), 'How to start with self-driving cars using ros'.
URL: *https://www.theconstructsim.com/start-self-driving-cars-using-ros/*

Tellez, Ricardo; Whitley, J. (2021), 'Ros developers podcast 071: The autoware foundation (self-driving cars) with joshua whitley'.
URL: *https://www.youtube.com/watch?v=X9mzQwsP7gw/*

Tettamanti, T., Varga, I. and Szalay, Z. (2016), 'Impacts of autonomous cars from a traffic engineering perspective', *Periodica Polytechnica Transportation Engineering* 44(4), 244–250.

The Construct (2020), 'Ros developers' course library'.
URL: *https://www.theconstructsim.com/robotigniteacademylearnros/ros − courses − library/*

Thompson, S. (2021), 'Hd maps for autonomous driving: Part 1'.
URL: *https://gitlab.com/ApexAI/autowareclass2020/-/blob/master/lectures/14HDMaps/HDMapsforAutonomousDrivingPartI.pdf*

Tian, Y., Yan, L., Park, G.-Y., Yang, S.-H., Kim, Y.-S., Lee, S.-R. and Lee, C.-Y. (2007), Application of rrt-based local path planning algorithm in unknown environment, *in* '2007 International Symposium on Computational Intelligence in Robotics and Automation', IEEE, pp. 456–460.

Trăsnea, B., Pozna, C. and Grigorescu, S. M. (2019), Aiba: An ai model for behavior arbitration in autonomous driving, *in* 'International Conference on Multi-disciplinary Trends in Artificial Intelligence', Springer, pp. 191–203.

Tu, K., Yang, S., Zhang, H. and Wang, Z. (2019), Hybrid a based motion planning for autonomous vehicles in unstructured environment, *in* '2019 IEEE International Symposium on Circuits and Systems (ISCAS)', IEEE, pp. 1–4.

Tzafestas, C. S. and Tzafestas, S. G. (1999), Recent algorithms for fuzzy and neuro-fuzzy path planning and navigation of autonomous mobile robots, *in* '1999 European Control Conference (ECC)', IEEE, pp. 4736–4743.

Ulrich, I. and Borenstein, J. (1998), Vfh+: Reliable obstacle avoidance for fast mobile robots, *in* 'Proceedings. 1998 IEEE international conference on robotics and automation (Cat. No. 98CH36146)', Vol. 2, IEEE, pp. 1572–1577.

Ulrich, I. and Borenstein, J. (2000), Vfh/sup*: Local obstacle avoidance with look-ahead verification, *in* 'Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)', Vol. 3, IEEE, pp. 2505–2511.

Umari, H. and Mukhopadhyay, S. (2017), Autonomous robotic exploration based on multiple rapidly-exploring randomized trees, *in* '2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)', IEEE, pp. 1396–1402.

Unity Technologies (2021).
URL: *https://unity.com/*

Urmson, C., Anhalt, J., Bartz, D., Clark, M., Galatali, T., Gutierrez, A., Harbaugh, S., Johnston, J., Koon, P., Messner, W. et al. (2007), A robust approach to high-speed navigation for unrehearsed desert terrain, *in* 'The 2005 DARPA Grand Challenge', Springer, pp. 45–102.

Véras, L. G. D., Medeiros, F. L. and Guimaráes, L. N. (2019), 'Systematic literature review of sampling process in rapidly-exploring random trees', *IEEE Access* 7, 50933–50953.

Volkswagen-Newsroom (2018), 'City emergency braking'.
URL: *https://www.volkswagen-newsroom.com/en/city-emergency-braking-3667*

Waslander, Sebastian; Kelly, J. (2021), 'Occupancy grids'.
URL: *https://www.coursera.org/learn/motion-planning-self-driving-cars/lecture/oJcwU/lesson-1-occupancy-grids*

Whitley, J. (2021), 'Autoware 101'.
URL: *https://gitlab.com/ApexAI/autowareclass2020/-/tree/master/lectures/06*$_{A}utoware_{1}01$

Wikipedia (2020), 'Data science'.
URL: *https://en.wikipedia.org/wiki/Data*$_{s}cience$

Winner, H., Hakuli, S., Lotz, F. and Singer, C. (2014), *Handbook of driver assistance systems*, Springer International Publishing Amsterdam, The Netherlands:.

Yang, S., Wang, W., Liu, C. and Deng, W. (2019), 'Scene understanding in deep learning-based end-to-end controllers for autonomous vehicles', *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 49(1), 53–63.

Yaqoob, I., Khan, L. U., Kazmi, S. A., Imran, M., Guizani, N. and Hong, C. S. (2019), 'Autonomous driving cars in smart cities: Recent advances, requirements, and challenges', *IEEE Network* 34(1), 174–181.

Yoon, S., Lee, D. and Jung (2018), 'Spline-based rrt* using piecewise continuous collision-checking algorithm for car-like vehicles', *Journal of Intelligent & Robotic Systems* 90(3-4), 537–549.

Youakim, D., Ridao, P., Palomeras, N., Spadafora, F., Ribas, D. and Muzzupappa, M. (2017), 'Moveit!: autonomous underwater free-floating manipulation', *Ieee Robotics & Automation Magazine* 24(3), 41–51.

Yurtsever, E., Lambert, J., Carballo, A. and Takeda, K. (2020), 'A survey of autonomous driving: Common practices and emerging technologies', *IEEE access* 8, 58443–58469.

Zegers, R. (2020), 'Path planning basics, online course of the construct by roberto zegers'.
URL: *https://www.theconstructsim.com/robotigniteacademy*$_{l}earnros/ros - courses - library/path - planning - basics/$

A Appendix

The next pages contain an outline for the rosgraphs recorded in the simulation time for the trajectory planners:

- Freespace A*: Appendix A.1:
- Freespace Lattice: Appendix A.2:
- Open Planner: Appendix A.3:
- Public Road: Appendix A.4:

The original and complete rosgraphs are available on github:Rosgraphs. It can be visualized using the rqt graph on a ROS installation or with the Runtime Manager Interface, in Autoware.AI (See Figure 37). A complete explanation from these rosgraphs meaning, as well as this master thesis content can be found in my master thesis video: Youtube_Master_Thesis_Marcus

The last Appendix A.5 illustrates the LGSVL configurator interface, which allow the user defines the AV's sensors, frequencies, data publishing rates, topics, etc.

A.1 Appendix 1: RosGraph Outline: Freespace-AstarPath Generation

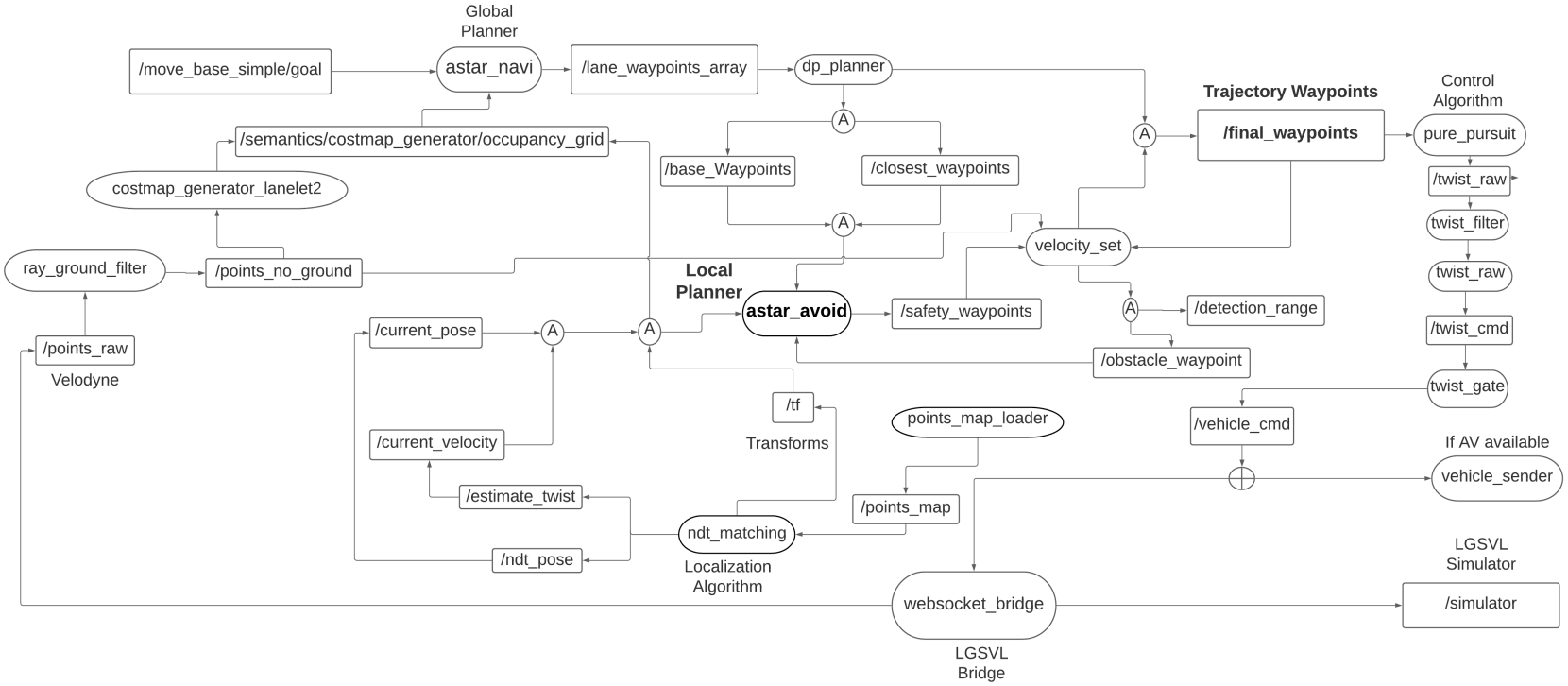


Figure 90 – A deep insight of Freespace Astar Trajectory Generation

A.2 Appendix 2: RosGraph Outline: Freespace-Lattice Path Generation

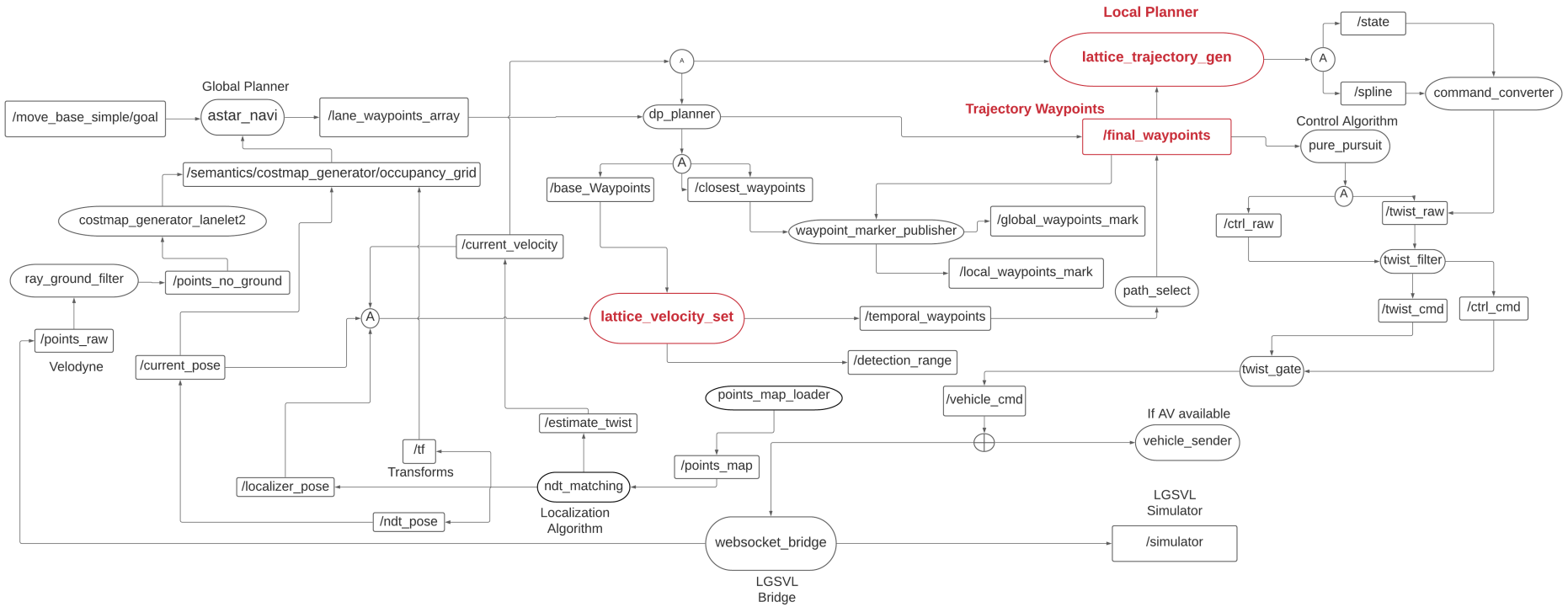


Figure 91 – A deep insight of Freespace-Lattice Trajectory Generation

A.3 Appendix 3: RosGraph Outline: Open Planner Path Generation

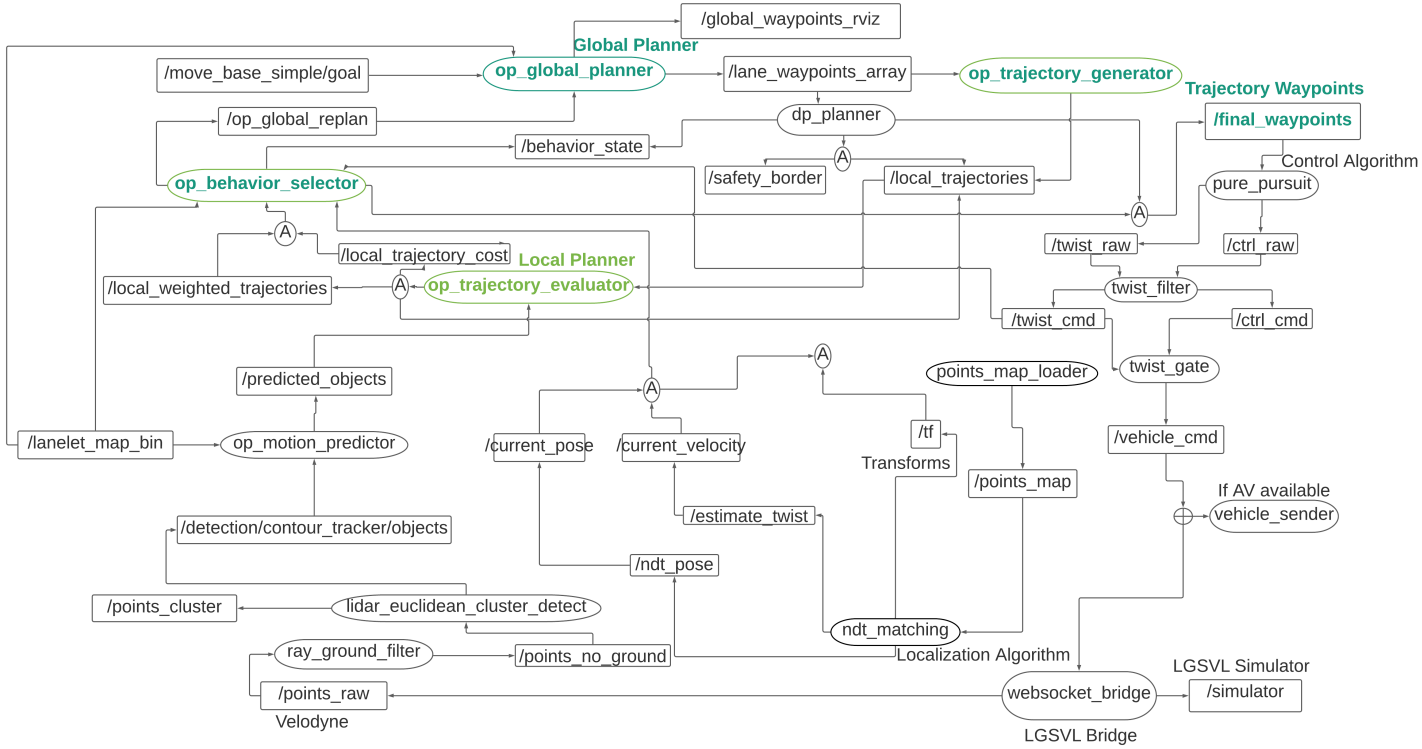


Figure 92 – A deep insight of Open Planner Trajectory Generation

A.4 Appendix 4: Cyber-Graph Outline: Public Road Path Generation

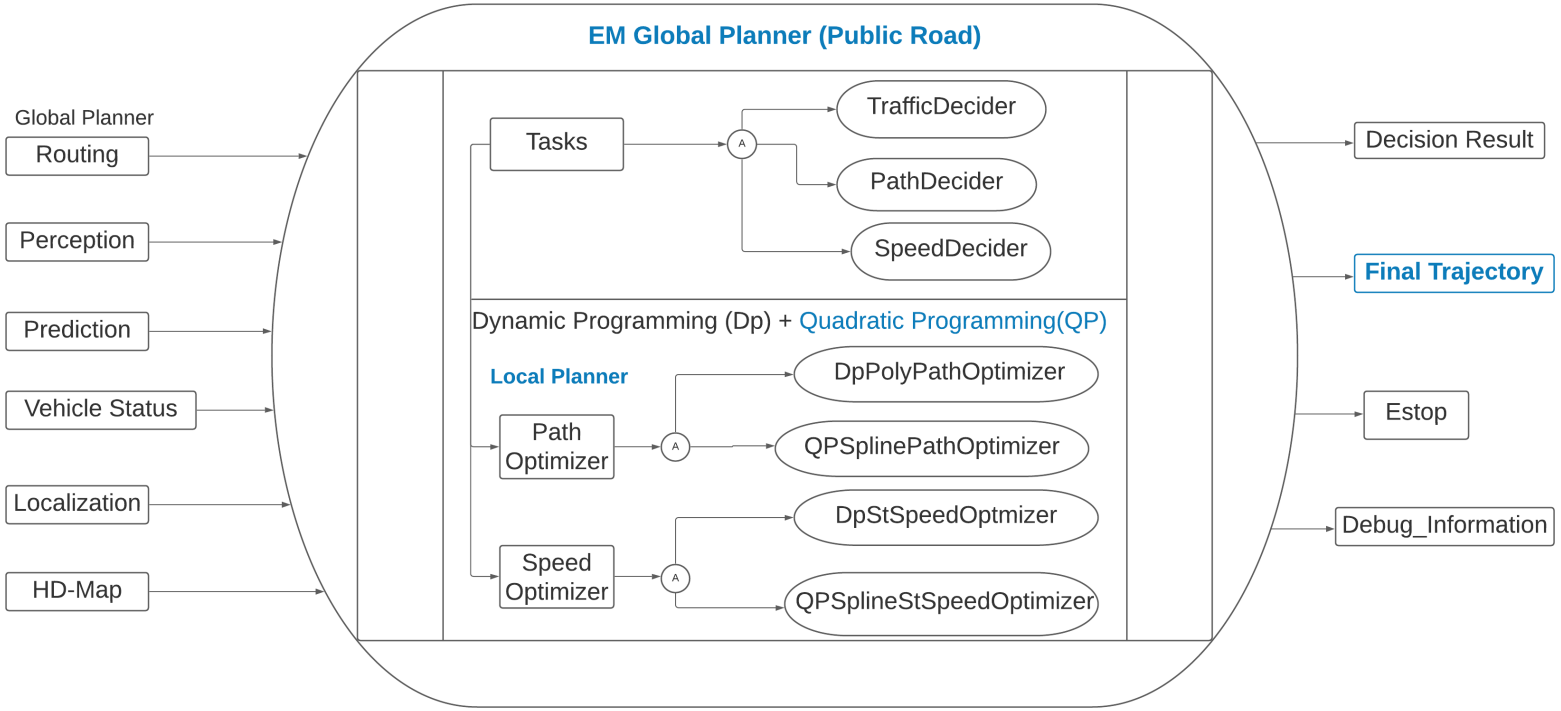


Figure 93 – A deep insight of Public Road Trajectory Generation

A.5 Appendix 5: Vehicle Configuration in SVL simulator

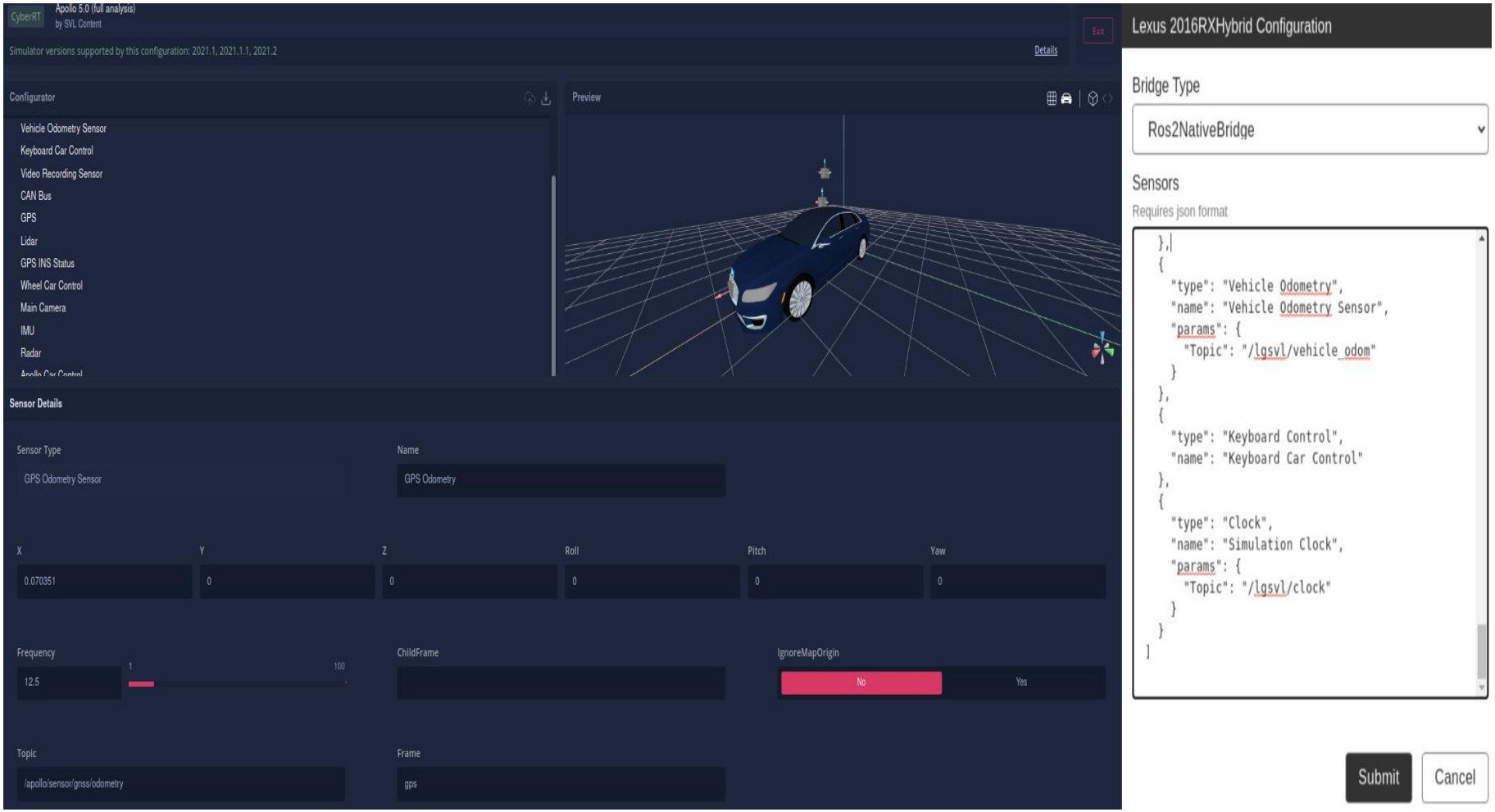


Figure 94 – Vehicle Configurator. Left: SVL configurator interface., Right: JSON format.

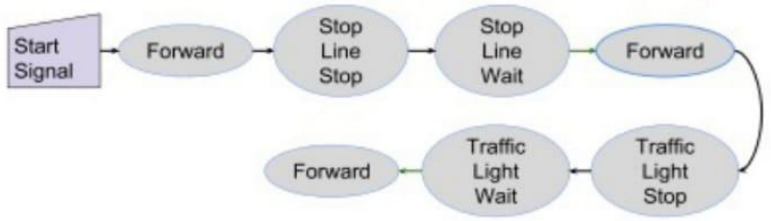
B Annex

The next pages contain relevant figures and schemes that complement the dissertation understanding.

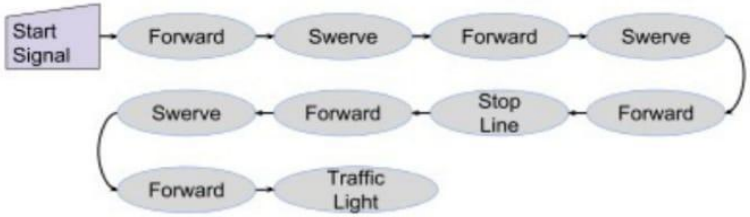
B.1 Annex 1: Open Planner Decision-Making Structure

State	Switch to	Condition
Start	Forward	Receive start signal from joystick
Forward	Swerve	Current trajectory is blocked, not all trajectories are blocked
Forward	Follow	All trajectories are blocked
Forward	Traffic light stop	Traffic light is red within the stopping distance range
Forward	Stop sign stop	Stop sign within the stop distance range
Forward	Mission accomplished	Goal position within the distance range
Swerve	Follow	All trajectories are blocked
Swerve	Forward	Drive parallel to the center
Follow	Forward	Not all trajectories are blocked
Traffic light stop	Traffic light wait	Not green light and velocity is zero
Traffic light stop	Forward	Green traffic light
Traffic light wait	Forward	Green traffic light
Stop sign stop	Stop sign wait	Velocity is zero
Stop sign wait	Forward	After stopping for time range
Any state	Emergency stop	Receive emergency stop signal
Emergency stop	Forward	No emergency stop signal

a) Behavior states transitions conditions



b) Behavior state flow while navigating the simulated vector map without obstacles



c) Behavior state flow while navigating the simulated vector map with obstacles

Figure 95 – Open Planner behavior states illustration. Darweesh et al. (2017)

B.2 Annex 2: Operation Design Domain Diagram

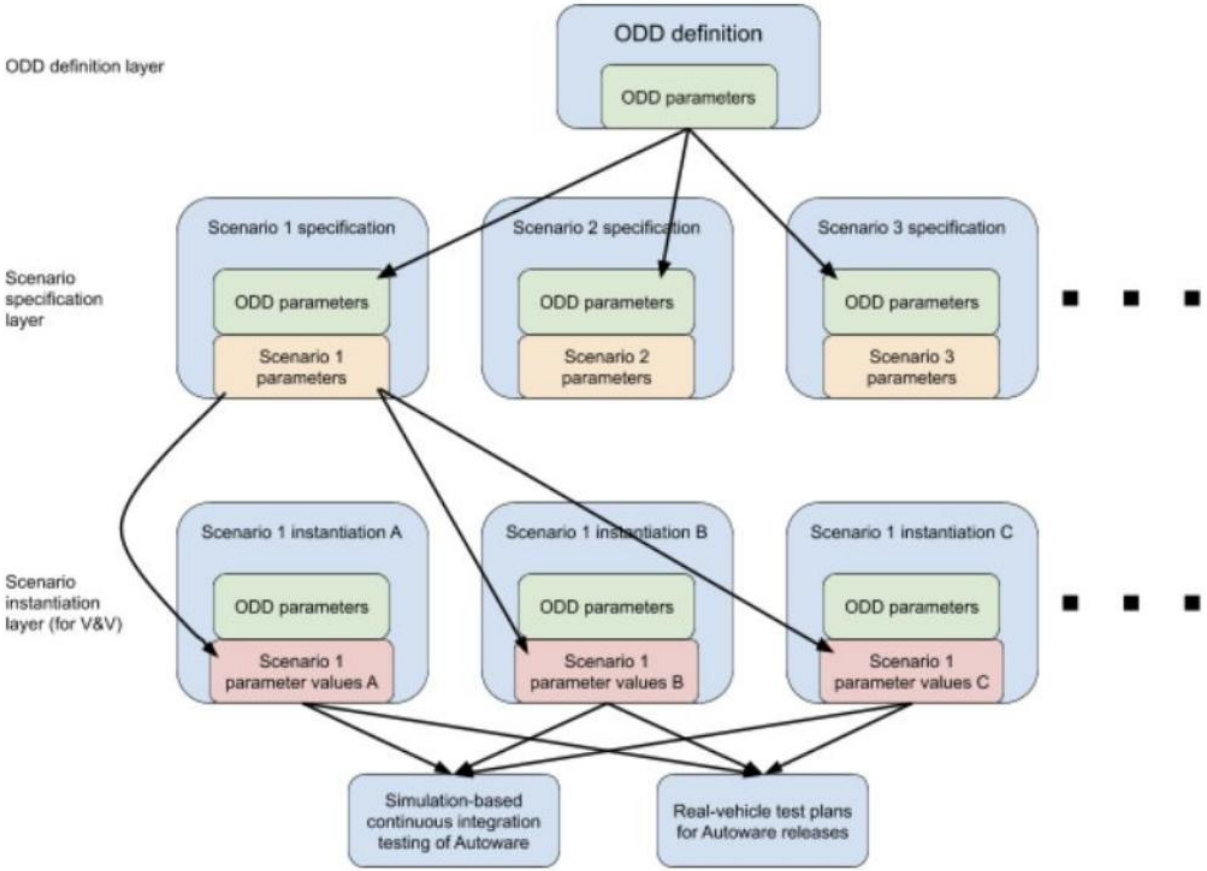


Figure 96 – ODD based Development Workflow. Source: Whitley (2021)

B.3 Annex 3: Autonomous Driving Architecture

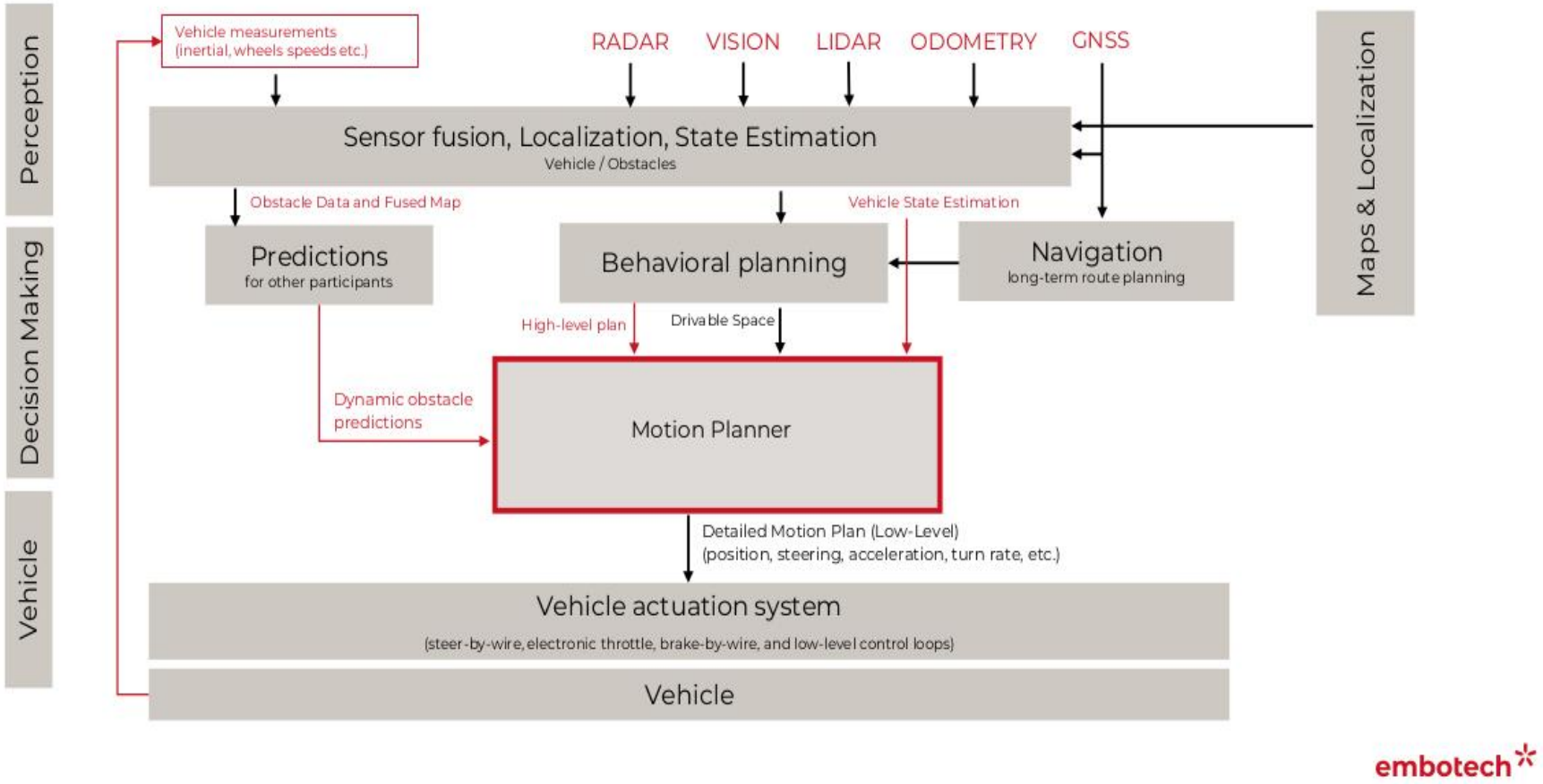
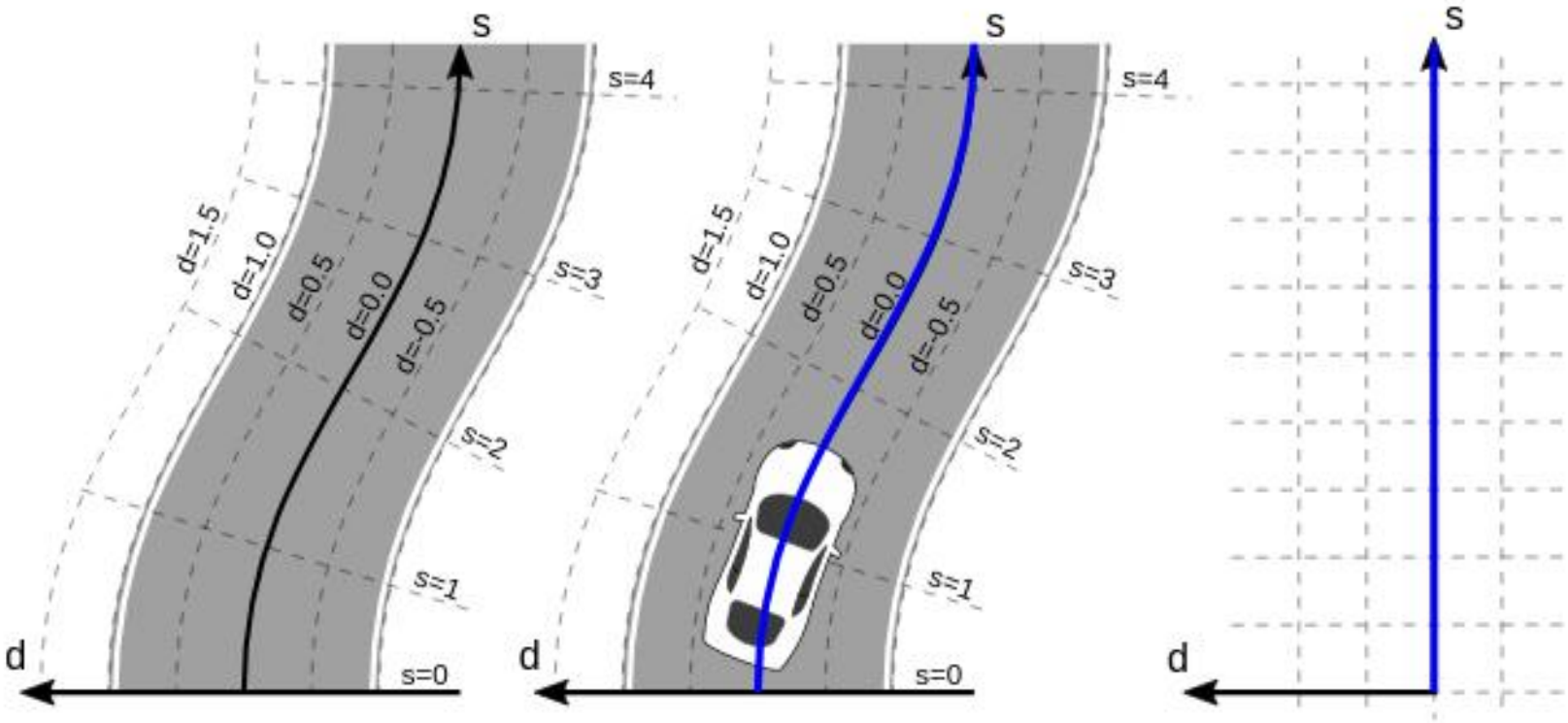


Figure 97 – A faithful autonomous driving representation of Autoware.Auto architecture, with motion planner focus.

Source: Embotech (2021)

B.4 Annex 4: Frenet Coordinates



Representation of a reference path in Frenet coordinates (s,d) on a road segment.

Figure 98 – Representation of a reference path in Frenet coordinates(s,d) on a road segment. Source: fjp.github.io (2018)

B.5 Annex 5: High Definition Map

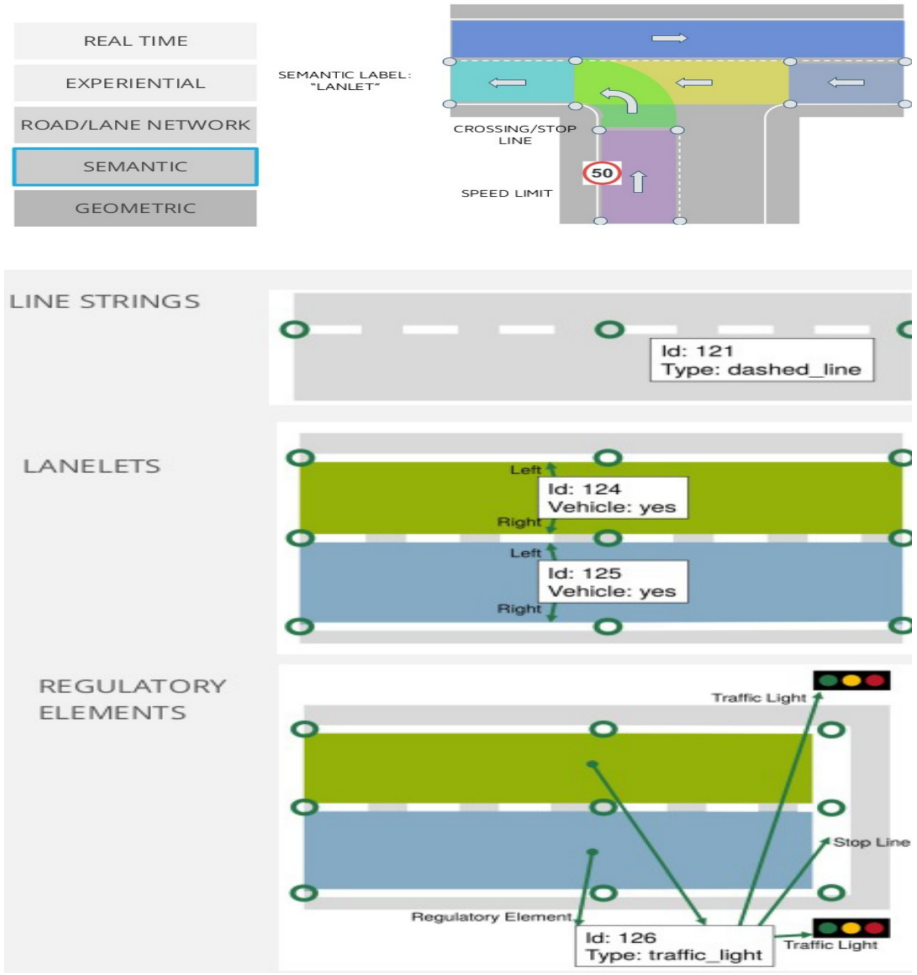


Figure 99 – UP: High Definition Map Content. Bottom: Lanelet2 HD-map format composition. Source: Thompson (2021)

B.6 Annex 6: Pegasus Method

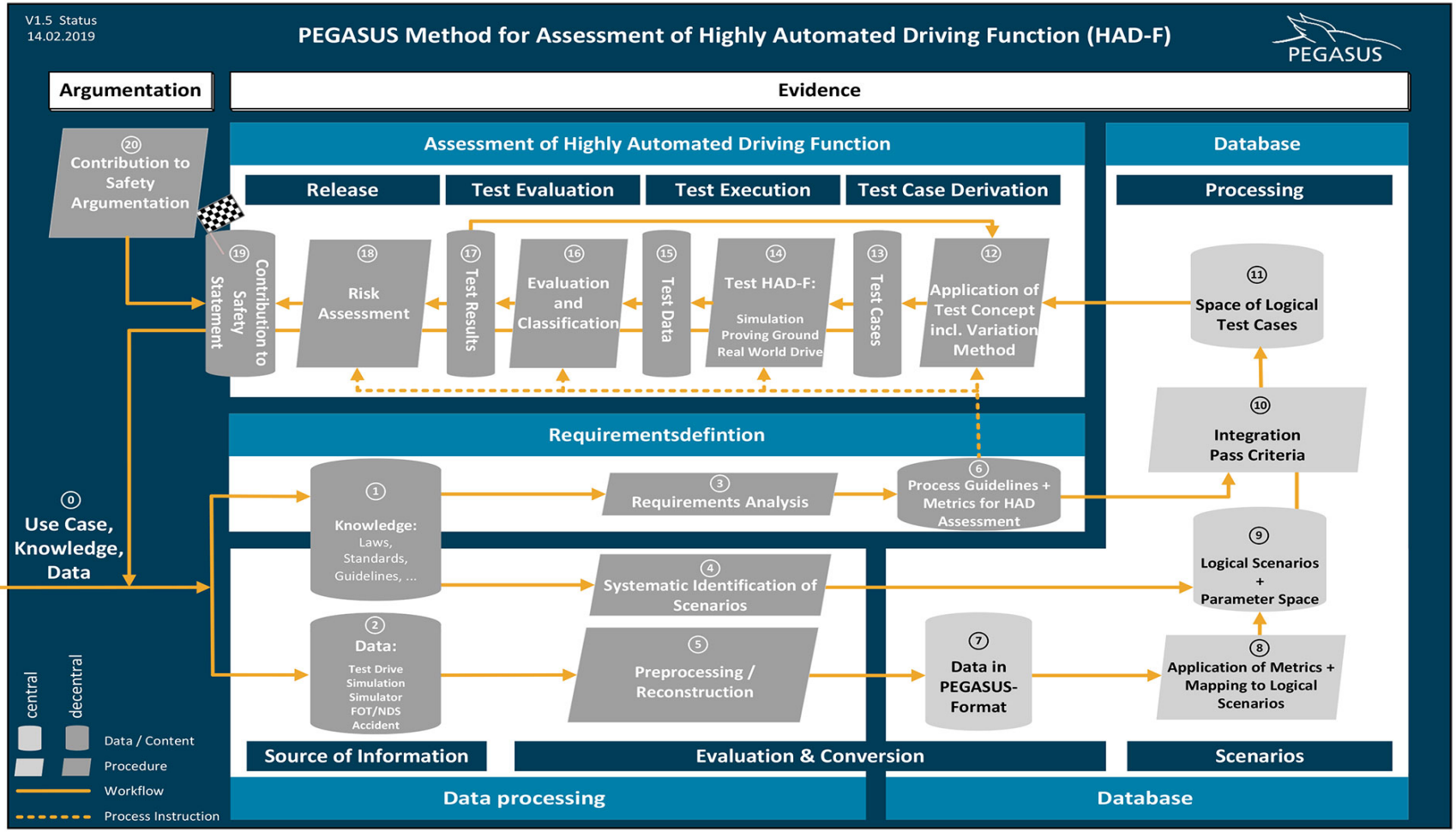


Figure 100 – PEGASUS validation method workflow. Source: Audi and Volkswagen (2020)

B.7 Annex 7: Autonomous Stack Full Composition

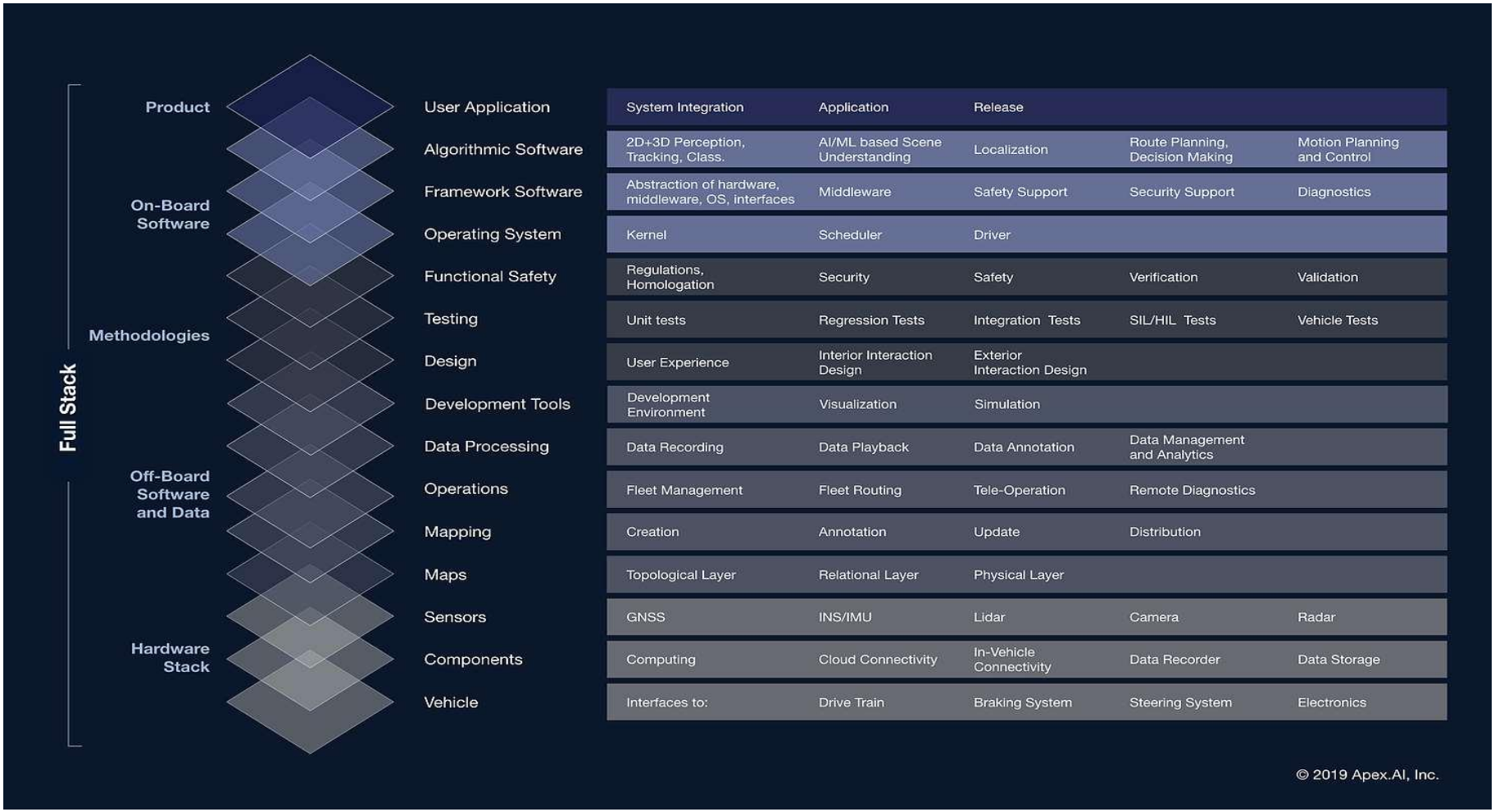


Figure 101 – Autonomous Driving Stack Composition. Source: Whitley (2021)

Appendix C - **SETUP OF ADS ON LINUX MACHINES**

Autonomous Driving Stacks: Installation Procedures

Run Autoware.Ai and Apollo.Auto with LGSVL simulator is straightforward. Any additional docker setup is needed to launch the containers. For this reason I am going to share just the weblink of the installation procedures:

Autoware.AI: Autoware.AI-LGSVL-Simulator

Apollo.Auto 6.0: Apollo.Auto 6.0 -LGSVL-Simulator

Autoware.Auto: Autoware-Auto-LGSVL-Simulator

The Autoware.Auto ADS, however have more complex additional steps and workaround which I will describe more detailed in lines below:

Componentes of ADS - Autoware.Auto - Ros2 - Lgsvl Simulator

The steps below are suitable for :

Graphics Card: Nvidia-GPU-driver Operational System: Ubuntu 20.04 ROS2-Distro: Foxy

Obs: The lines that follow the symbol "$" are command lines that must be just copied and pasted in terminal.

Autoware.Auto && ROS2

Autoware.Auto comes already with the most recent Ros2 version pre-installed in git clone step. However it is possible to install a desired ros2 distro from source:

source-installation

1) Check if you have the Nvidia-driver and Cuda installed on machine:

$ nvidia-smi

I have the most recent version, was tested and is working with ubuntu 20.04:

Driver graphic Cards: Nvidia 460 Cuda: 11.2

Other older versions also works fine but is necessary to check for compatibility.

To check for the driver-cuda compatibility:

Cuda-Comatibility or cuda-nvidia

It is important to install nvidia-driver and the suitable cuda version to allow GPU consumers application (such as rviz2, LGSVL simulator, and others) work properly without crash, return errors, or open and close issues.

```
autoware-auto-ros1@marcus-ros2-foxy:~ (master #%)$ nvidia-smi
Mon Mar 22 12:14:50 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.39       Driver Version: 460.39       CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  GeForce GTX 105...  Off  | 00000000:01:00.0 Off |                  N/A |
| N/A   47C    P0    N/A /  N/A |    427MiB /  4042MiB |      1%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A      1767      G   /usr/lib/xorg/Xorg                338MiB |
|    0   N/A  N/A      1936      G   /usr/bin/gnome-shell               38MiB |
|    0   N/A  N/A      2400      G   /usr/lib/firefox/firefox            1MiB |
|    0   N/A  N/A      3605      G   /usr/lib/firefox/firefox            1MiB |
|    0   N/A  N/A      3994      G   /usr/lib/firefox/firefox            1MiB |
|    0   N/A  N/A      4195      C   ...ffice/program/soffice.bin       43MiB |
+-----------------------------------------------------------------------------+
```

Figure 102 – Graphics Card Configuration Required

If $ nvidia-smi command line above did not return the installed driver and cuda, probably it is not installed.Then go to the installation procedures, described in link below and install it:

docs-nvidia

2) Intall Docker Container, in order to enable the download of Autoware.Auto repo images in a virtual environment.

There are 2 possibilities here.

A.1) Choose the method “Install using repository” in the link below:

A.2) Stop when find the recommendation of “linux post install” bolded in blue and click over it, then go: A.3) Repeat all the instructions and commands below the following chapters: “Manage Docker as a non-root user” and “Configure Docker to start on boot” (optional)

or B.1) install in a single step the most recent Docker version: $ curl https://get.docker.com | sh sudo systemctl –now enable docker

later install nvidia-container-toolkit to allow build and run GPU accelerated Docker containers.

B.2) Follow the steps written of the chapter “Setting up NVIDIA Container Toolkit”:

nvidia-container-toolkit

B.3) After redo the steps above in "Setting up Docker" and “Setting up NVIDIA Container Toolkit”, it is important to follow linux post install steps, described in A.2 and A.3 section previously: docker-post-install

This toolkit allows to leverage NVIDIA GPUs and run high graphical consumers applications such as rviz2, Autonomous Driving stack within Lgsvl simulator, etc.

3) Return to the requirements of Autoware.Auto installation here: ade-cli-requirements

The "Requirements" section of this website were accomplished throug the steps above. If not follow the instruction of the link above. Then now install the most recent ade-cli project following the commands bellow the chapter “Linux x86_64 and aarch 64” from item 1 until finish the item 3 inside the gray box.

4) Now follow the link below to start Autoware.Auto repository installation: Autoware-Auto-istallation

and repeat the chapters:

“Setup ADE home and project checkout” That is basically the clone of Autoware.Auto image on github (download AD stack pkgs and functionalities)

“Sharing files between the host system ADE” Syncronize bashrc from host and docker (optinal) There is a typo error in cmd line in this section: replace "ade-home" to "adehome"

“Entering the development enviroment” In step 4 of this third chapter (in red above) is importan to launch all the most recent container images listed (foxy) : - .aderc-amd64-foxy .aderc-amd64-foxy-lgsvl Also dashing version if available (last version of ubuntu 18.04)

For my case rviz2 just work with the bolded image above...for rviz2 does not crash with the standard foxy image (.aderc-amd64-foxy) or the other ones, it is necessary to modify the .aderc file. But this will be showed in further section of this manual.

In my case I have chosen to build the image below, once it contains the LG simulator inside the virtual machine for quick tutorial learnings:

$ ade –rc .aderc-amd64-foxy-lsvl start –update –enter

However It is interesting the standard image too (it is possible to build a lot of images in docker, and they do not conflict with each other). So you can exit the built container image above and built other images: $ logout $ ade stop $ cd adehome/AutowareAuto $ade –rc .aderc-amd64-foxy start –update –enter

Then follow the steps found in this link to check if the container image is running ok and how to enter and use the ade container image.

5) To test the packages functionalities of Autoware Auto build the packages through the following command:

First navigate to AutowareAuto folder if you not be inside it: autoware-auto-ros1@ade:

$cd AutowareAuto

Now issue the command to build the packages (similar to catkin_make in ROS1) autoware-auto-ros1@ade:

$ colcon build

After build the packages probably an error of compilation will raise (in my case this happened). This is an issue of missing libraries, packages or not-well written CmakeList, package.xml or another c++ file... To fix this and update the AutowareAuto folder with the most recent and correct and additional ros2 packages run the commands found in the first gray box below the chapter “Starting from a clean state” , found on the bottom of the following webpage below:

Autoware-building

obs: In current section, there is a "git checkout 1.0.0 release" step. In this git branch the AutowareAuto team probably forgot to add the autoware_auto_launch package inside the adehome/AutowareAuto/src/tools folder. This package is necessary to launch important demos such as visualization, perception and others in rviz. In this way if you had this issue you will need to dowload it from the repo and copy and past e this missed folder into the directory cited above. You will find below the link to get it: Autoware-tools

Then build/compile the packages again using the command: autoware-auto-ros1@ade:

$ colcon build

In this way you will have more packages built that were outside the AutowareAuto repo folder that are also important to launch demos, etc...

Congratulations the AutowareAuto installation is done!!!

Now you need to install the lgsvl simulator to work along AutowareAuto

Obs.: There is a internal LGSVL simulator that was installed with the container image above. Probably the simulator will crash or open and close every time you restart and reenter the container image (because of libraries will reinstalled every time the container image start). In this way it is required to remove this library every time you enter the container, through the cmd below:

$ sudo apt remove mesa-vulkan-drivers

This internal simulator does not have a real web-socket connection to exchange ros-simulator messages, therefore there are not a real car/actuator/sensors conversion to the

ros messages type and vice-versa. It is a kind of easy use or shortcut for AutowareAuto team test quickly some autonomous driving functionalities. To set up scenario, car, sensors and launch the internal LG simulator follow the link below steps:

Autoware-lgsvl

The most real use of LGSVL simulator is its installation outside the docker container. It needs to be installed in the host system. In this way it will really simulate a communication between ROS2 and an outside framework (LGSVL simulator that represents the real car and the scenario perception) through a ros2 bridge.

6) INSTALLATION OF LGSVL SIMULATOR OUTSIDE DOCKER (RECOMMENDED) In this way the installation steps of the LGSVL simulator on the host system are found below:

LGSVL simulator linux

6.1) Go to this website and dowload the the simulator package lgsvl-simulator

6.2) Run the executable of the simulator If the simulator open and close quickly, issue the following command: sudo apt remove mesa-vulkan-drivers

6.3) Click over "Open web browser" icon Register an account and dowload the scenarios, cars, etc.

ROS2 LGSVL Briged installation

7) INSTALATION OF ROS2-LGSVL INTERFACE BRIDGE

Follow the steps found in the link below INSIDE THE $ade image (start the docker container and enter it)

Follow the steps below to install the ros2-lgsvl simulator bridge: ros2-bridge

I will reproduce the commands found in the link above to make easy:

$sudo apt update

$sudo apt install python3-colcon-common-extensions

$sudo apt install libboost-all-dev

$git clone https://github.com/lgsvl/ros2-lgsvl-bridge.git

$source opt/ros/foxy/setup.bash

$cd ros2-lgsvl-bridge

$git checkout foxy-devel

$colcon build –cmake-args '-DCMAKE_BUILD_TYPE=Release'

$source install/setup.bash

$lgsvl_bridge

After issue the last command you should have as output Listening on port 9090

Good! Now it is possible to make ros2 works with lgsvl simulator simulating a CAN bus communication and ros2-car actuators messages conversion!!! To see the integration reproduce a usage tutorial of rviz2 and simulator!

Obs: It is good to be aware that you will find some issues with ros2 bridge and simulator if or install an old or new version of the simulator or ros-distro regarding communication and rviz2... It is always good to read the documentation to be sure your simulator installed match the ros2-distro version, etc. Also take care in tutorials that were made to internal simulator, they have different ROS2 topics set than the external simulator. Also were done for different ROS versions. Therefore it is pretty important to check ROS2 topics list and configure the Car json model with the correct topics inside the simulator. I have issues to see lidar point cloud in rviz for example because the tutorial was done for internal simulator, suggesting rviz to subscribe to "Lidar_Front" frame and topic "Lidar_Front/points_raw". However for the external simulator the correct was to set the fixed frame to "Velodyne" and topic "lgsvl/lidar/points_raw"

8) You will probably face rviz2 issues, then goes to appendinx section to learn hot to fix this issue! If rviz2 does not open it may be necessary to remove the following line present in .aderc file:

ADE_DISABLE_NVIDIA_DOCKER=true