ESCOLA POLITÉCNICA
UNIVERSIDADE DE SÃO PAULO

SIEMENS

MILTER SHINITI PESCE

MIGUEL AGOSTINHO PEREIRA NETO

TRAJECTORY PLANNING FOR ARTICULATED VEHICLES

2017

São Paulo, Brazil

ESCOLA POLITÉCNICA
UNIVERSIDADE DE SÃO PAULO

SIEMENS

MILTER SHINITI PESCE

MIGUEL AGOSTINHO PEREIRA NETO

TRAJECTORY PLANNING FOR ARTICULATED VEHICLES

This work was presented to the coordination of the Mechatronics Engineering course of the *Escola Politécnica da Universidade de São Paulo* as partial requirement for obtaining the title of Bachelor of Engineering.

Advisor: Prof. Thiago de Castro Martins

2017

São Paulo, Brazil

Neto, Miguel Agostinho Pereira
TRAJECTORY PLANNING FOR ARTICULATED VEHICLES / M. A. P. Neto, M. S. Pesce -- São Paulo, 2017.
59 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.GEOMETRIA COMPUTACIONAL 2.OTIMIZAÇÃO GLOBAL 3.ENGENHARIA AUTOMOBILÍSTICA 4.PLANEJAMENTO DE TRANSPORTES I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II.t. III.Pesce, Milter Shiniti

ACKNOWLEDGMENTS

We gratefully acknowledge the support Prof. Thiago de Castro Martins who provided insight and expertise that greatly assisted this project. We would also like to show our gratitude to Siemens for proposing the theme of this project and for providing assistance with its software. Without them, the present study could not have been completed.

DEDICATION

We dedicate this work to our families, friends and loved ones who have supported us throughout our entire graduation.

“Dans la vie, rien n'est à craindre, tout est à comprendre.”

Marie Curie

ABSTRACT

The transport industry is passing through a revolution driven by the Advanced Driver Assistance Systems. They are designed for helping the driving, with benefits in terms of security, comfort, time, money and energy. These technologies are a growing market, and they are being improved towards autonomous vehicles.

However, the advance of Advanced Driver Assistance Systems is a challenge that requires the development of new technologies. Among them, the trajectory planning is one of the most relevant. The problem becomes more complex when it involves articulated vehicles, like trucks and buses, since the kinematics conditions differs from a regular vehicle.

This Project was conceived with the goal of design a solution for the trajectory planning of articulated vehicles. To do so, a tool was developed based on an algorithm called Rapidly-exploring random tree. The problem was also modeled using a simulation software in order to test the viability of the trajectories created by the tool.

Besides the details regarding the development of this tool, this document will also present the results of this project. The tool was capable of generating trajectories with success and the simulation proved them to be feasible.

Keywords: Articulated vehicles, trajectory planning, Rapidly-exploring random tree.

RESUMO

O setor de transportes passa atualmente por uma revolução impulsionada pelos Sistemas Avançados de Assistência ao Motorista (*Advanced Driver Assistance Systems*, em inglês). Eles são desenhados para colaborar na tarefa de condução, apresentando benefícios em termos de segurança, conforto, tempo, dinheiro e energia. Tais tecnologias estão em ascensão, sendo aprimoradas progressivamente rumo aos veículos autônomos.

No entanto, o avanço de Sistemas Avançados de Assistência ao Motorista é um desafio que requer novos desenvolvimentos em tecnologias. Dentre elas, o planejamento de trajetórias é um dos mais relevantes. O problema se torna mais complexo quando envolve veículos articulados, como caminhões e ônibus, pois as condições cinemáticas diferem de um veículo comum.

Este projeto foi concebido com o objetivo de desenvolver uma solução para o planejamento de trajetórias para veículos articulados. Para isso, uma ferramenta foi desenvolvida com base em um algoritmo de planejamento chamado *Rapidly-exploring random tree*. O problema também foi modelado através de um software de simulação para testar a viabilidade das trajetórias criadas pela ferramenta.

Além dos detalhes por trás do desenvolvimento desta ferramenta, este documento apresentará também os resultados deste projeto. A ferramenta proposta foi capaz de gerar trajetórias com sucesso, e tais trajetórias se mostraram viáveis através da simulação.

Palavras-chave: Veículos articulados, planejamento de trajetória, *Rapidly-exploring random tree.*

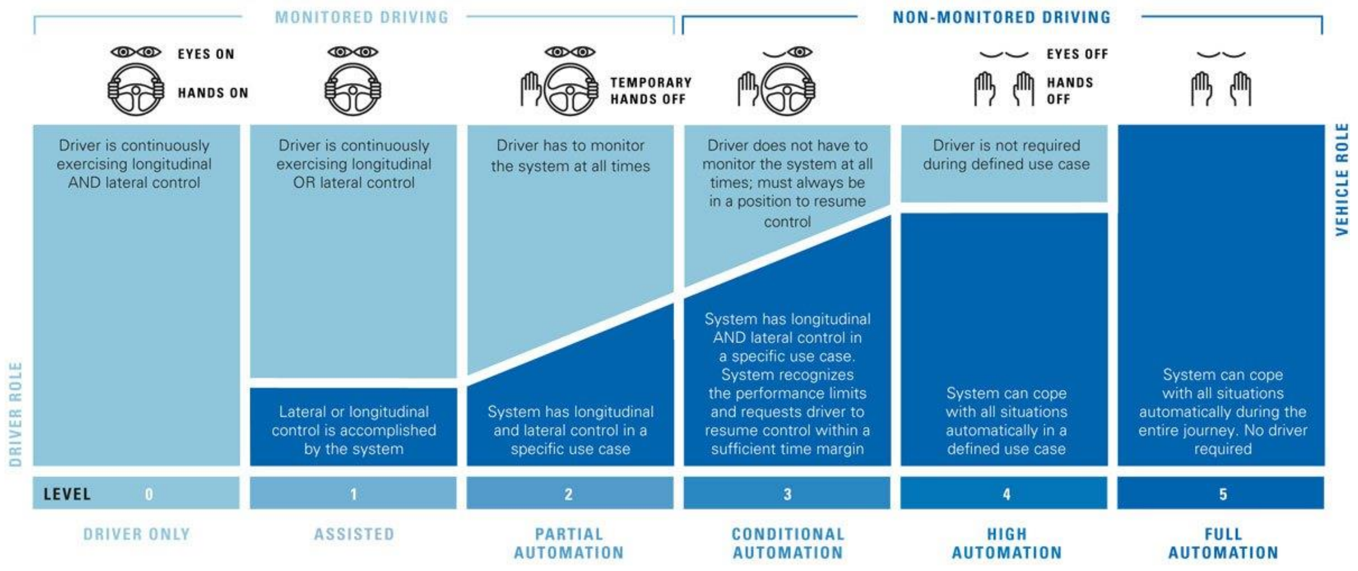
TABLE OF CONTENTS

1. INTRODUCTION

1.1. Advanced Driver Assistance Systems

Advanced Driver Assistance Systems (ADAS) are intelligent systems developed to automate, adapt and enhance vehicles systems for safety and better driving, helping the driver during the driving process. Safety features are able to alert the driver to potential problems, avoid collisions and even take control of the vehicle, whereas adaptive features might provide adaptive cruise control, automate lighting and braking, keep the driver in the correct lane, among other features.

These technologies are often split into six different categories concerning the level of automation. Level 0 represents no automation at all while level 5 corresponds to full automation. Today's level is between 1 and 2 since there are examples of vehicles on the market such as Tesla Model S and Mercedes-Benz S65 AMG which allow the driver to keep his hands temporarily off the steering wheel, even though road and traffic must be constantly monitored. The figure below describes better all ADAS automation levels.



MONITORED DRIVING
NON-MONITORED DRIVING
EYES ON
HANDS ON
TEMPORARY HANDS OFF
EYES OFF
HANDS OFF
Driver is continuously exercising longitudinal AND lateral control
Driver is continuously exercising longitudinal OR lateral control
Driver has to monitor the system at all times
Driver does not have to monitor the system at all times; must always be in a position to resume control
Driver is not required during defined use case
Lateral or longitudinal control is accomplished by the system
System has longitudinal and lateral control in a specific use case
System has longitudinal AND lateral control in a specific use case. System recognizes the performance limits and requests driver to resume control within a sufficient time margin
System can cope with all situations automatically in a defined use case
System can cope with all situations automatically during the entire journey. No driver required
DRIVER ROLE
VEHICLE ROLE
LEVEL
0
1
2
3
4
5
DRIVER ONLY
ASSISTED
PARTIAL AUTOMATION
CONDITIONAL AUTOMATION
HIGH AUTOMATION
FULL AUTOMATION

Figure 1: ADAS automation levels. Reprinted from (1).

With ADAS progressively improving towards fully autonomous vehicles, many applications can be found on the field of trajectory planning. In the future, cooperative trajectory planning may be responsible for the entire control of the traffic within big cities, automating intersection crossings for instance. Moreover, applications in industry are vast, mainly in areas where the automation represents reducing costs and enhancing productivity such as the agricultural industry.

The trajectory planning becomes an even more challenging subject when applied to articulated vehicles. The reason is that the presence of a spherical joint increases the number of variables needed to describe the system.

1.2. Objectives

In this context, this project aims to work with ADAS within the heart of the automobile’s academia and industry, researching deeper the problem of trajectory planning for articulated vehicles in diverse contexts.

The objective is to develop a trajectory planning method to be applied on articulated vehicles. Additionally, the project also plans to model physically this kind of vehicle and simulate the trajectories generated. This way, the planned trajectory as well as the method itself, can be tested and validated.

1.3. Engaged parts and motivations

In order to achieve the objectives traced and presented above, three parts committed to work together: the students and authors of the project; the academic project advisor; and a private company.

The project was developed by Miguel Agostinho Pereira Neto and Milter Shiniti Pesce, both current mechatronics engineering students at *Escola Politécnica da Universidade de São Paulo* (EPUSP) and double-graduated in general engineering at *École Centrale de Lyon*, France.

Miguel has been part of the Advanced Driver Assistance Systems team of Siemens Industry Software for 6 months during his internship at Lyon, where he could be in touch with ADAS and autonomous vehicles, the subject of this final-year project. Milter has work for 6 months as a software developer intern at Gemalto, a digital security multinational, in the region of Paris in France.

Working within the engineering industry field abroad gave both students the opportunity to see the real context of creating and developing cutting-edge technology, which unfortunately is rare in Brazil. For two mere engineering students, the possibility to achieve big goals and be part of researches that could lead to great changes in the human society even during the university is an unparalleled opportunity.

ADAS and autonomous vehicles are subjects that fit perfectly into the high-level research topics and are intensively growing in industry. This truly motivated the students to develop and study trajectory planning methods and simulations.

The academic project advisor is Thiago de Castro Martins, professor in the Department of Mechatronics and Mechanical Systems Engineering of EPUSP. He has B.S. degree in mechanical engineering and a Ph.D. degree, both from EPUSP. He has already had experience with the matter as he has worked with trajectory planning at the *Centre National de la Recherche Scientifique* (CNRS), in France.

The third part of the project is the company Siemens, here represented by the engineer Pierric Toulemont. Largest engineering company in Europe, Siemens works closely to many car makers and OEMs (Original Equipment Manufacturer). Through the feedbacks and demands that both development and engineering service teams received from its clients, Siemens became aware that the market is looking for trajectory planning technologies. The company already received demands from many sectors including agricultural industry, trucks automation, port maintenance and even valet parking for cars.

One of Siemens' many divisions is focused on the development of Industry Softwares. The office based in Lyon (France) has about 150 people developing, providing engineering support for and promoting the 1D Multiphysics Siemens' simulation software, called LMS Imagine.Lab Amesim.

Amesim is a system simulation platform which allows characterizing static and dynamic behavior of a component or a system. The physical modeling is based on a bond graph representation, and the solver uses time derivative equations to compute the simulations. Its big advantages lie on its usability and its small simulation time compared to 3D simulation software. It allows an easy simulation of big and complex multi-physics systems. Therefore, Amesim is mainly used in the domains of automotive and aeronautics where systems are obviously complex to configure and analyze, even as ADAS and autonomous vehicles.

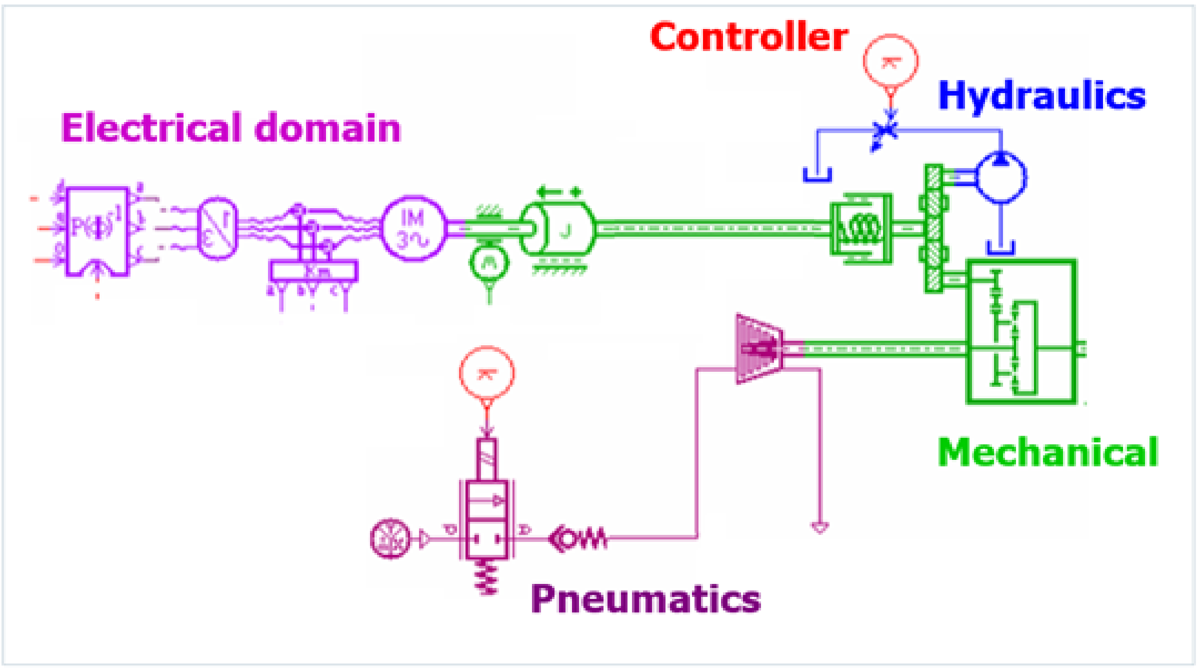


Figure 2: Amesim Multiphysics usage example. Reprinted from (2)

The partnership with the company Siemens Industry Software provided Amesim academic licenses for the students. Thus, Amesim became the software used to model the articulated vehicles and simulate its behavior when executing planned trajectories. Amesim provides trustworthy modeling of all subsystems of a vehicle and it is a very powerful tool to enhance the simulation part of the project.

Therefore, this project represents the point where industry meets university and real problems faced by several companies can be studied and solved together. For the university, working on a relevant and recent theme is quite important because it allows the institution to participate on present industrial problem solving. For the industry, a scientific research assuring that its solutions and products are efficient and providing a solid theoretical background is extremely useful to promote its products, mainly to its customers and supporters.

2. STATE OF THE ART

In this section, a description of the work already developed concerning trajectory planning will be described (3). The goal is to introduce the basics concepts that are used in this project as well as to present the existing tools to attack the trajectory planning problem.

First of all, the difference among path and trajectory must be clarified.

A path consists on a sequence of **configurations** taking into account boundary conditions at the beginning and at the end. It basically means a geometrical trace that the concerned vehicle should follow without any collision.

Finally, a trajectory represents a sequence of **states** visited by the vehicle, parameterized by velocity, time, and kinematics. Trajectory planning tries to outline the actual vehicle's transition from one feasible state to the next, regarding kinematics limits based on vehicle dynamics and route boundaries.

Essentially trajectory planning encompasses path planning in addition to planning how to move based on velocity, time, and kinematics.

2.1. Planning technique

In general, planning for autonomous or intelligent driving is divided into four classes (3), including route planning, path planning, maneuver choice and trajectory planning. The flowchart of the trajectory planning can be seen in Figure 3.

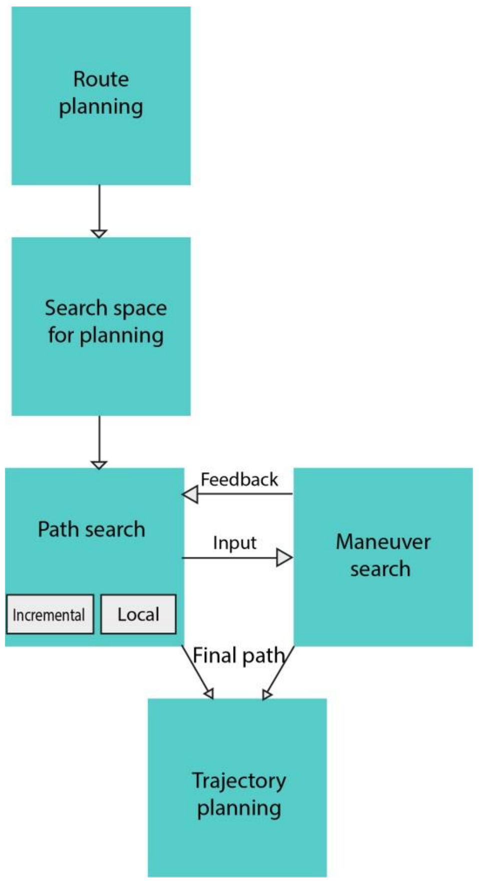


Figure 3: Flowchart of trajectory planning. Recreated from (3)

Route planning is simply finding the global route from a given origin to a destination. It does not take vehicular dynamics into account, so it is not within the scope of the project.

Once the route is defined, it is necessary to represent the environment in a way that enables the path planning. Therefore, the physical space shall be transformed in a state space that represents the set of all possible states that a vehicle can be in. The state space includes information like the vehicle position and orientation. Some search space algorithms are presented in 2.2.

Path planning is the task to find a path in the state space that connects an initial configuration to a final configuration and that does not collide with any obstacle. The path planning can be tied to a maneuver search. A brief description of path planning algorithms is presented in 2.3.

Maneuver is a high-level characterization of the motion of the vehicle, with regard to the vehicle's position, speed and steering. For instance, a maneuver can be "going straight forward" or "turning left".

So, the path planning acts as input to the search for the best maneuver, i.e. the maneuver which places the vehicle as close to the planned path as it is possible. Based on the best maneuver, the path search can change, as shown with a feedback loop between these two modules.

Finally, once the path is finalized, the final trajectory planning is generated.

2.2. Search space for planning

When planning a vehicle motion, the space must be discretized and digitally represented in a way that the physical space is transformed into a state space describing the vehicle position, orientation, velocities and all other useful measures. The efficiency of this space description is essential to optimize the computational speed.

Five algorithms are briefly described next and are figuratively represented in Figure 4. These algorithm techniques can be simultaneously employed, improving the planning capabilities. Once the search space is built, the planning algorithms are ready to start looking for the best path and trajectory.

2.2.1. Voronoi Diagrams

Voronoi Diagrams, also called Dirichlet tessellation, is a partitioning of a plane into regions based on distance to points in a specific subset of the plane. Basically, the algorithm tries to minimize the distance between the vehicle and its surrounding obstacles. This kind of technique is usually employed on static environments such as parking lots, once dynamic obstacles may cause discontinuities and be unsuitable for non-holonomic vehicles.

2.2.2. Occupancy grids and costmaps

These both methods work similarly, discretizing the state space into a grid whose cells are associated to the probability of each cell being occupied by an obstacle or to the proportional cost of traversing such cell.

2.2.3. State Lattices

State Lattices are seen as a generalization of grid methods once Lattices are constructed of simple motion primitives connecting one state to another, in terms of position, curvature or time. Consequently, it connects the initial state to the final one, regarding the boundary conditions.

2.2.4. Driving corridors

Driving corridors represent a continuous collision-free space that is limited by environment and obstacles boundaries. Each vehicle has its own driving corridor and its center line will form the path around which the trajectory to be followed is planned.

Once this technique is very dependent of the environment representation, some constraints may appear and compromise real-time planning.

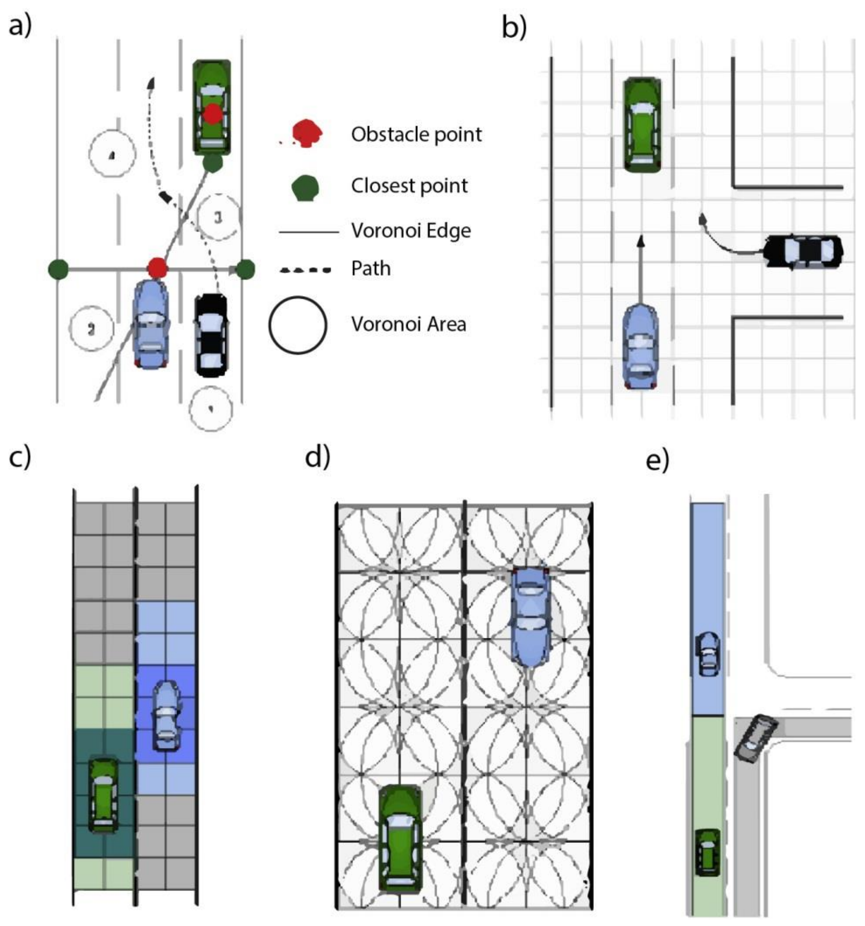


Figure 4: (a) Voronoi Diagram; (b) Occupancy Grid; (c) Costmap; (d) State Lattice; and (e) Driving Corridor. Recreated from (3).

2.3. Path Planning

The step of finding the best geometric path for the vehicle to follow is usually divided into two different approaches.

The first one uses incremental sampling or discrete geometric structures to find the best sequence of actions to be realized. It re-uses information from previous searches to increase search speed. Two incremental search methods will be presented in this section: the Rapidly-exploring random tree and the Lattice Planners.

The second one is a local search that uses multiple final states to find the single best action.

2.3.1. Rapidly-exploring random tree (RRT)

The Rapidly-exploring random tree (RRT) algorithm creates a data tree. This tree basically consists of feasible paths that are built online by stochastically extending branches towards randomly generated target configurations.

RRT is probabilistic complete, which means if there is a solution for the path planning problem, the probability of the algorithm finding it goes to one as the iteration time goes to infinity. Moreover, this method can be easily implemented in real-time and guarantees kinematics feasibility. However, it can create jerky paths and does not verify collision checking, what may be problematic and time-consuming in an environment full of obstacles. Furthermore, there is always a compromise between optimization and exploration speed.

2.3.2. Lattice planners

As already described, state lattices construct a discrete search space which enables relevant state continuity. Instead of randomly explore the states, this method acquires the goal state in a deterministic way, satisfying the differential constraints of the vehicle. It reduces computational time and has a good performance for non-holonomic vehicles.

This method guarantees optimality and smoothness of the solution once it does not introduce discontinuities. Also, the path plan is very close to the real motion of the vehicles. However, exhaustive sampling may lead to unnecessary computational complexity and oscillations may be present due to problematic discretization in the heading angle.

2.3.3. Local search

Searching the entire graph in real-time is not always efficient, so a local search uses a different approach trying to reduce the search space regarding distances and time. Probably one of the most employed methods consists on lateral shifting a given geometric curve, generally splines or clothoids. The results are then evaluated by a cost function taking into account several parameters as distances, time and collision checking. This method may not perform very well inside complex environments and the Figure 5 describes it better.

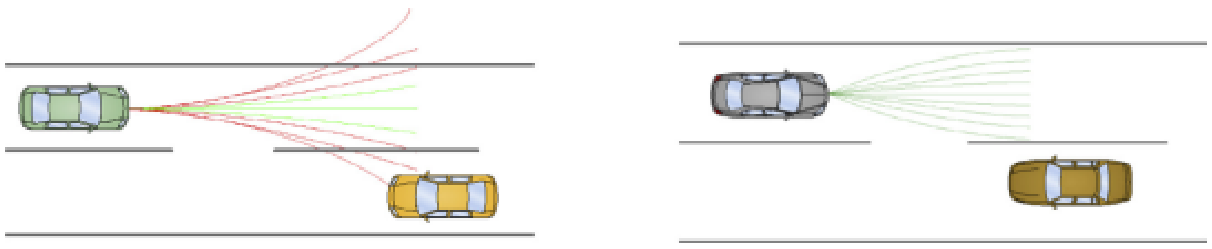


Figure 5: Local search examples. Reprinted from (3).

2.4. Experimental issues: robot pulling a trailer

An application of the trajectory planning method is the experimental work performed by a French research team from the *Laboratoire d'Analyse et d'Architecture des Systèmes - CNRS* (4). The trajectory planning was performed on a nonholonomic system, i.e. a system whose state depends on the path taken in order to achieve it, represented by a mobile robot pulling a trailer (see Figure 6). Car-like vehicles are examples of nonholonomic systems.

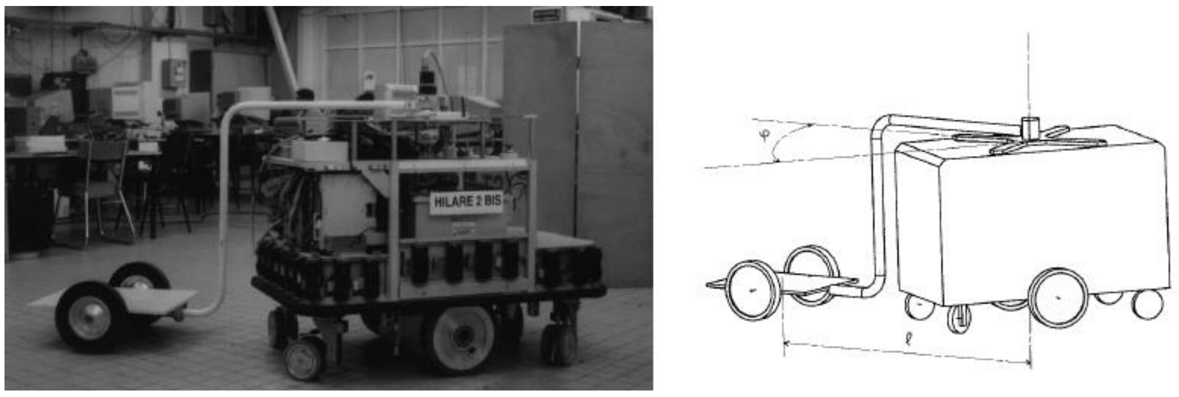


Figure 6: Robot and trailer used in experiments. Reprinted from (4)

The robot was a two driving-wheels mobile robot equipped with an odometer capable of giving the position and the direction of the robot as well as an angular encoder that gives the absolute direction of the trailer. The robot was attached to a trailer, so the system was an articulated vehicle. It also had a computer environment composed of Unix workstations and on-board processors.

The experimentation can be divided in three main steps: the path planning, the trajectory planning and the control law.

The path planning method implemented here relied on a geometric and on a local planner. The geometric planner computed a collision-free path using the Random Path

Planner algorithm (RPP), the random path planner presented in (5). It does not take into account the kinematic constraints. This path is then approximated by a sequence of collision-free and feasible paths computed by the local planner, without taking into account the obstacles.

Once a path is defined, the computing of the velocities of each wheel along the planned path gives the trajectory. The velocity and acceleration constraints must be taken into account. The challenge at this level is to find the minimum-time trajectory. For instance, if the shorter path is not regular enough, the robot will have to stop at some points, increasing trajectory time.

Finally, to get the trajectory to the motion of the system, a simple control law was chosen. When the robot goes forward, the trailer is ignored and the robot is stabilized using this control law. But when the robot goes backward, it is necessary to define a virtual robot which is symmetrical to the real robot with respect to the wheel axle of the trailer (see Figure 7). The same control law is then applied to the virtual robot.

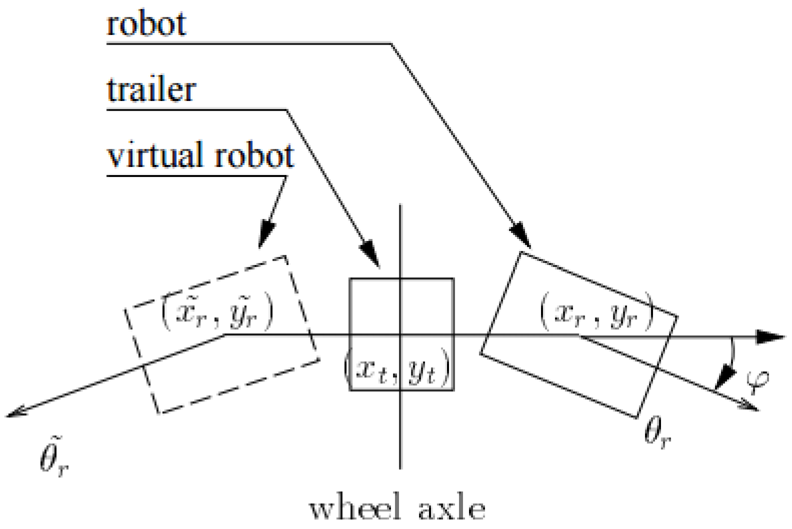


Figure 7: Virtual robot. Reprinted from (4).

The experiment results are showed below. In every scenario, the goal was reached with accuracy of approximately 10 cm. The length of the paths is around 15 meters and the average linear velocity is 0.5 m/s. It is foreseen that improving the control law can lead to a better accuracy.

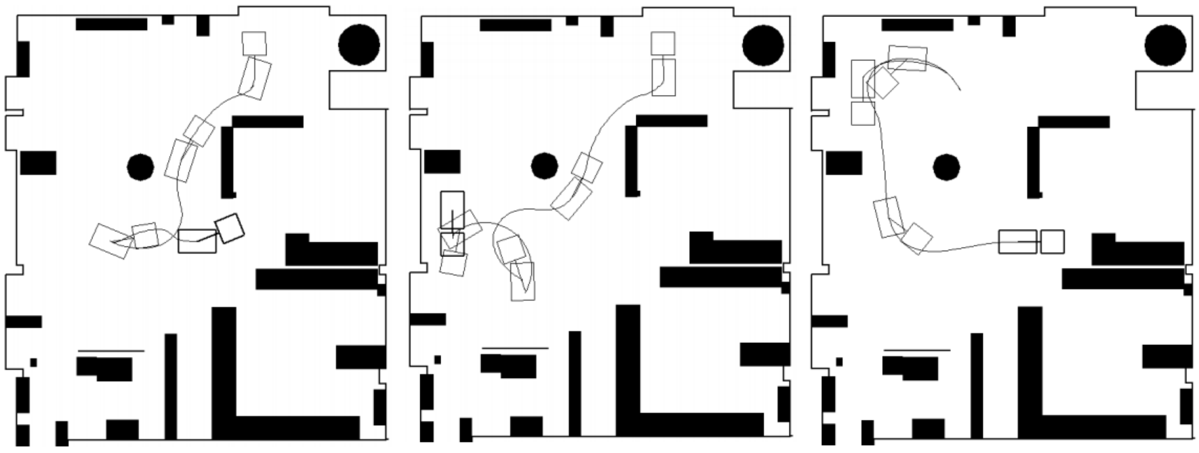


Figure 8: Results of the experiment. Reprinted from (4).

3. PROJECT OVERVIEW AND DELIVERABLES

By analyzing the state of art, the timescale of the project and the available resources, the three parts involved (the students, Siemens and the project advisor) decided the guidelines of the project. In order to achieve the main objective, which is to develop a trajectory planning method and to simulate it with Amesim, some decision were taken. In this section, these decisions will be introduced in order to give an overview of the project.

Considering that the project aims to explore the trajectory planning of an articulated vehicle, it is mandatory to define the kinematics modeling for this type of vehicle. A theoretical modeling of an articulated vehicle, including the differential equations of its movement, was defined and applied in the trajectory planning method.

A much more realistic model was created with Amesim. This model is an important part of the project as it provides a reliable way to simulate how real articulated vehicles would behave when following the planned trajectories. Both theoretical and Amesim models are introduced in section 4.

The method here developed to plan trajectories follows the technique presented in the state of art, section 2.1.

For this project, it is considered that the articulated vehicle must travel in a bi-dimensional space with static obstacles. Since this project will consider static and pre-defined environments, SLAM (Simultaneous Localization and Mapping) planning won't be primarily studied because it emphasizes mostly obstacle prediction and traffic environment modeling.

To plan a trajectory in such environment, the space is discretized and represented as a state space. Each state is defined by the configuration of the vehicle (Cartesian coordinates and orientation of the front and rear vehicle) and the motion primitives connecting it to previous states (speed, steering angles and time). In other words, the state is tied to the dynamic conditions that led the articulated vehicle to assume a specific configuration in that particular moment. This approach is similar to the State Lattices from 2.2.3.

The path planning developed in the project was based on two algorithms. The main path planning algorithm chosen was the RRT, briefly introduced in the state of art. The algorithm is detailed and discussed in section 5, along with the reasons that guided to this choice. A secondary path planning algorithm, called Dubins path, was used to complement the RRT.

Python has been chosen as the main programming language once it grants flexibility to code and experiment new features. Even though there are other languages with a better computation power than Python, this criterion has not been a major priority.

A simplified maneuver search was incorporated into the path planning algorithm. As a result, the algorithm is capable of providing the trajectory for articulated vehicles.

A use case was designed in which the trajectory planning algorithm is implemented. The algorithm output is then simulated in the Amesim model. The goal is to validate if the planned trajectory can be followed by a real articulated vehicle. The results from these use cases are discussed in sections 7.5 and 8.3.

Therefore, the resulting deliverable of this project is the script that implements the trajectory planning method.

4. KINEMATIC MODELING OF AN ARTICULATED VEHICLE

4.1. Theoretical model

The non-holonomic RRT algorithm requires the input of the articulated vehicle's kinematics equations since the modeling is a 4-dimensions problem and depends on the four variables that describe the movement of this kind of vehicle.

Basically, the parameters are $\mathrm{x(t)}, \mathrm{y(t)}, \theta(\mathrm{t})$ and $\alpha(\mathrm{t})$, where $x(t)$ and $y(t)$ are the coordinates of the front vehicle's rear axle, $\theta(\mathrm{t})$ represents the orientation of the front vehicle and $\alpha(\mathrm{t})$ is the relative bend of the rear vehicle. In this project, a tractor-trailer model will be considered, corresponding to a 4-wheel front tractor and a 2-wheel passive trailer, both linked by a revolute joint. The joint acts as a mechanical stop to the movement. The modeling can be better understood seeing the Figure 9:

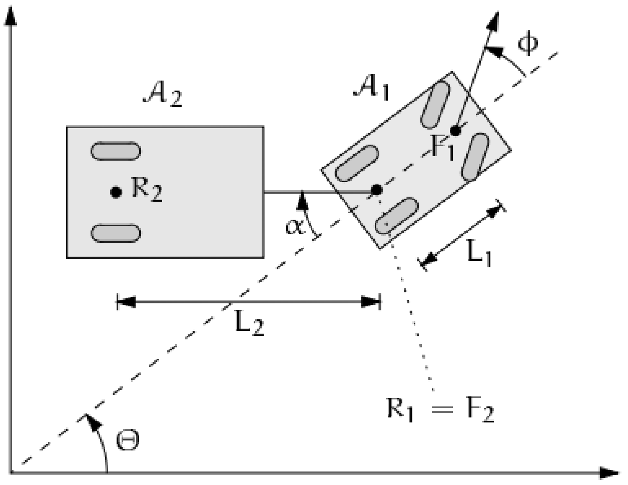


Figure 9: A model for the tractor-trailer vehicle. Reprinted from (6).

Some assumptions are done in order to use the modeling. First, the bodies move in a plane and the contact between each wheel and the ground is a pure rolling contact. Also, the revolute joint connecting the two parts of the vehicle is located at the middle of the tractor's rear axle – which means that it is an on-axle model. These propositions imply that there are a tractor's maximal steering angle and a maximal bending angle.

So, if the geometry of the vehicle is correctly derived, the set of equations that describe the articulated vehicle kinematic is (7):

$$\dot{x}(t) = v(t)cos\theta(t)$$

$$\dot{y}(t) = v(t)sin\theta(t)$$

$$\dot{\theta}(t) = \frac{v(t)}{L_1}tan\varphi(t)$$

$$\dot{\alpha}(t) = \ v(t) * \left[\frac{tan\varphi(t)}{L_1} - \frac{\sin(\alpha)}{L_2}\right]$$

Where $v(t)$ denotes the velocity of the tractor's rear point, $\varphi(t)$ the tractor's steering angle, L_1 the length between the tractor's axles and L_2 the length between the trailer's rear axle and the hitch point.

These equations have a non-holonomic nature so it is not possible to integrate them. Nevertheless, if we consider that $v(t)$ and $\varphi(t)$ are constants in time, the system turns out to be integrable.

4.2. Amesim model

The theoretical model used in the non-holonomic trajectory planning algorithm is only a simplified model of an articulated vehicle chassis. It means that an actual vehicle would have a lot of different subsystems including tire, suspension, chassis, steering and powertrain systems that need hundreds of different parameters to simulate the behavior of an articulated vehicle as accurately as possible.

Furthermore, the trajectory planning algorithm uses simplified equations to simulate a predicted displacement of the vehicle in the space. But once it is intended to apply the solution to a real articulated vehicle, hardware turns out to be essential in order to convert the trajectory planning outputs into inputs to the vehicle. Therefore, it could accelerate, brake and steer towards trying to follow the planned path.

Hence, all these subsystems are modeled in Amesim. So a trustworthy simulation is able to replace an entire real vehicle and its hardware. The model and how its components are related are represented in the draft below. The subsystems are described in the next subsections.

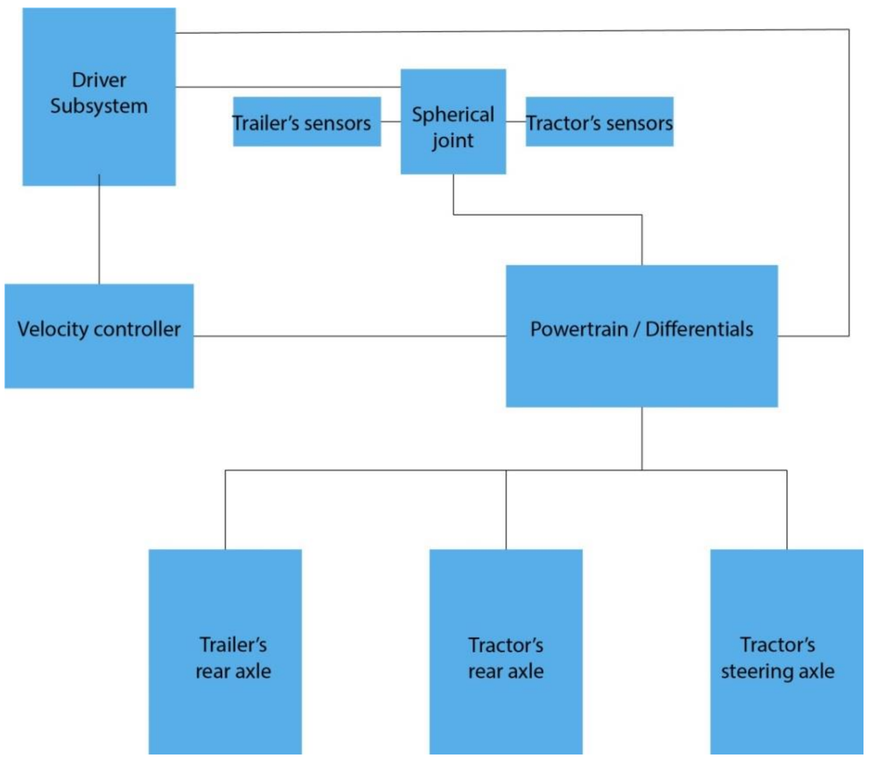


Figure 10: Complete Amesim physical model.

4.3. Chassis, suspension and steering subsystems

The chassis is modeled by three blocks representing the axles of the vehicle. Two of them do not steer (both rear axles), whereas the tractor's front axle may steer and consequently has the steering model connected to it. The axle block is responsible for updating the states that describe the position of the wheels with respect to the chassis. It takes into account suspension, steering, brakes and engine effects. In order to improve the model, an elastokinematic subsystem could also be attached to it.

Both rear axles have a simple suspension using a spring-damper model. The steering axle illustrated in Figure 11 contains more detailed description including an anti-roll bar to avoid rolling effects and an oscillating axle – a more resilient suspension employed on heavy transport.

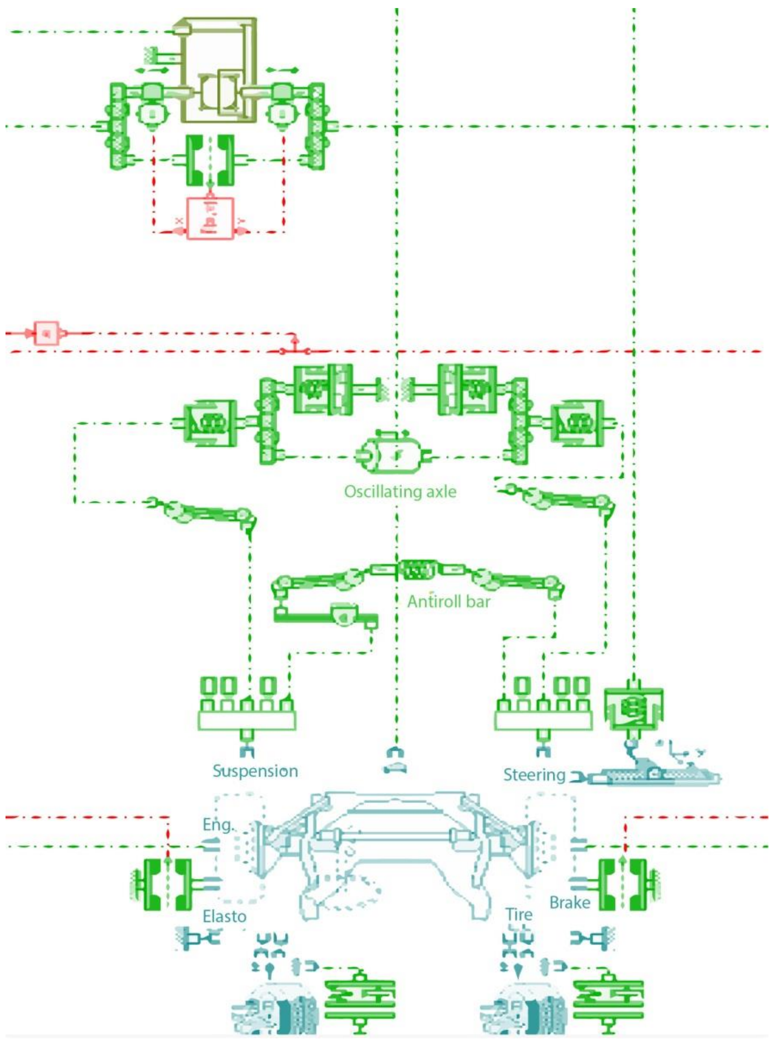


Figure 11: Chassis, suspension and steering subsystems.

The steering is modeled by a pinion-rack connected to a rotary spring-damper which receives the input of the steering wheel. Here, the steering angle is limited to 30 degrees.

The engine effect comes from the torque transmitted by a differential and the brake effect comes from a simple friction torque generator. Both engine and braking inputs will be further described.

After all the three axles are modeled, they are connected to a spherical joint. It is noted here that a revolute joint is used in the theoretical model. This is explained by the fact that the theoretical model considers only the x-y plane while Amesim does its calculations for the 3D space.

4.3.1. Tire subsystem

The tire model is illustrated in the image below. It is composed of 5 main components: road model, road grip model, tire model, tire belt model and tire kinematics model.

The road model creates a contact between the tire and the road.

The road grip model represents the adherence between the tire and the road, so it allows a simulation in many different soils (dry, wet, snowy, etc.) depending on the road grip parameter inputted.

The tire model generates the contact force at the tire/soil interface. This is a pure dynamic block and allows a longitudinal/lateral behavior analysis of the tire.

The tire belt model allows the computation of characteristic inputs of tire models such as side slip angle, longitudinal slip, camber angle, vertical load of the tire and turn slip.

The tire kinematics model is used to compute all kinematic elements of center of tire contact. It outputs variables such as absolute velocity of contact point and wheel rotary velocity. The tire stiffness is inputted to this model and a simple spring-damper has been used.

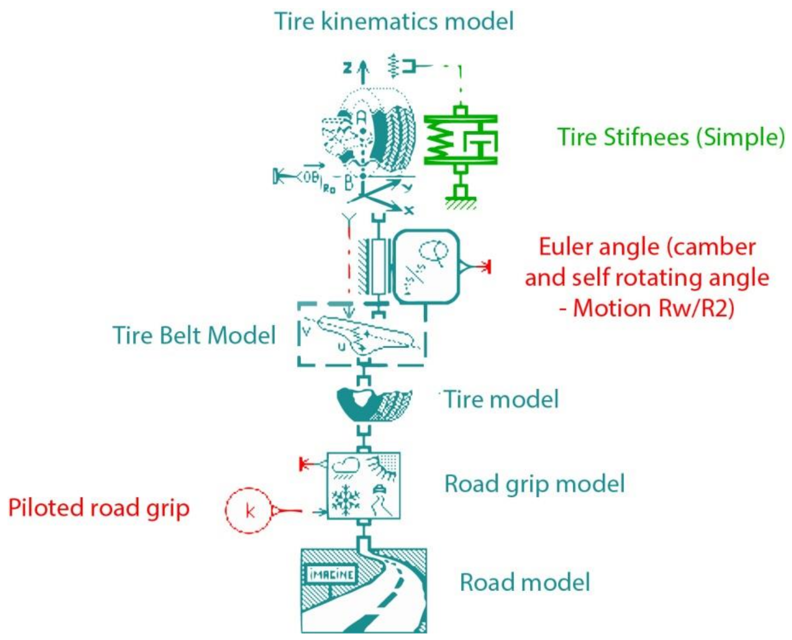


Figure 12: Amesim tire model. Reprinted from Amesim manual.

4.3.2. Powertrain subsystem and brake control

The powertrain subsystem is responsible by the engine modeling as well as the power transmission. In this project, the powertrain has a simple model. The throttle signal comes from the driver subsystem and is converted into torque by a torque converter. Then, a rotary load computes the inertia of the vehicle, outputting a rotary velocity. The torque goes through a reducer, a rotary spring-dumper and finally comes to a central differential. The central differential is connected to two other differentials corresponding to each axle of the tractor.

Moreover, simple control takes charge of calculating the right throttle to be transformed in torque. It takes into account the maximum power of the engine, 150 kW in this example, and then is multiplied by a first order system which adds a lag to the response of the throttle.

The brakes control is very similar to the throttle control. An input from the driver is multiplied by gains and a first order system, also adding a lag to the response. The signal is received by the friction torque generator. The friction torque is then applied between the wheel and the spindle.

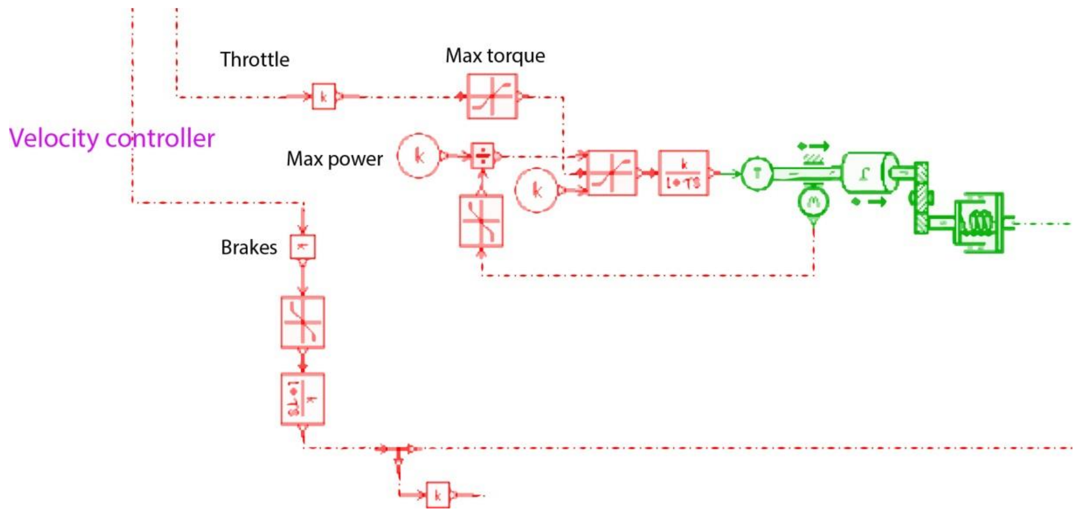


Figure 13: Powertrain subsystem and brake control.

4.3.3. Driver subsystem

The driver subsystem used in this project is a built-in Amesim feature that works as a path follower. It is composed by five main blocks that control longitudinal and lateral driver based on inputs as coordinates of the path to be followed ((x, y) coordinates and radius of the curve, with 0 meaning straight line) and target speed at each point of the path.

Given a trajectory in a .data file, the driver block produces steering wheel commands -that will be further inputted in the steering subsystem - in order to follow the trajectory. Two PIDs are responsible to correct the trajectory of the vehicle, one acting on the distance to the reference trajectory and the other acting on the heading. Signal coming from sensors, such as speed and lateral acceleration, feedbacks the controllers in order to minimize errors.

The longitudinal driver handles the accelerator (throttle) and brake pedals according to the target speed orders. The orders feed the powertrain and brake subsystems.

Generally speaking, the driver subsystem is supposed to be the component responsible for converting the trajectory planner outputs into braking, accelerating and steering inputs. The simulation of all the subsystems described along with the trajectory planning inputs is a good prediction of a real articulated vehicle behavior.

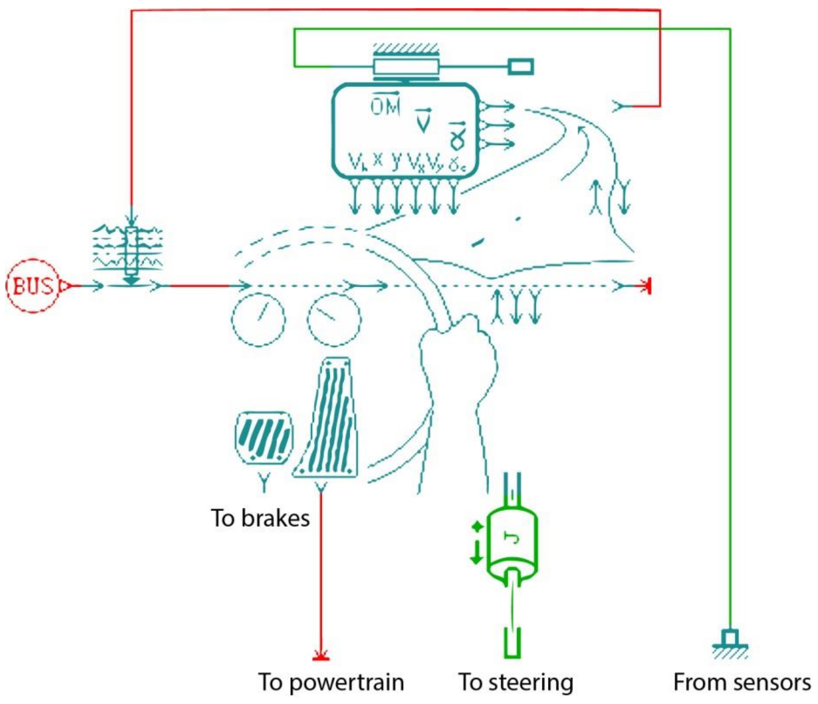


Figure 14: Driver subsystem. Reprinted from Amesim manual.

5. CHOOSING THE RRT ALGORITHM

A deeper analysis on the trajectory planning methods mentioned on the state of the art section leaded us to a single choice concerning the best method to implement on this case. The RRT seems to be the most appropriate one for several reasons.

First of all, it is a method that has been extensively used in recent years for path planning, including many cases of autonomous driving, which proves that it is a top research field on this subject. Yet RRT is a probabilistically complete algorithm (8) and even if it does not necessarily converge to an optimal solution, finding a solution for a complex high-dimensional problem like this one is already satisfactory. RRT also guarantees kinematic feasibility and handles general dynamical models. Finally, it can easily be implemented in real-time, and even if this is not the scope of this project, this is an advantage as it allows future work on the matter to implement real-time planning.

On the other side, RRT main drawbacks lie in the jerky paths it randomly creates, as well as the need of a collision checking routine for every step when developing the random tree. In a workspace with many obstacles it can result in computational complexity.

However, since the algorithm has already been extensively applied and studied, there are many optimizations that were presented in the literature that allows overcoming the algorithm drawbacks.

Global path planner	Vehicle type		Hitching		Cusps?		Optimization				Other	
	Single trailer	Multiple trailers	On-axle	Off-axle	Yes	No	Feasibility	Number of cusps	Length	Distance from obstacles	Runtime [s]	Length of path [m]
Two-step (or RPP)	✓		✓		✓		✓				6	15
RRT		✓	✓		✓		✓				-	-
grid search	✓			✓	✓		✓	✓	✓		30	100
Feasible velocities polygon		✓		✓		✓	✓		✓		-	-
Least squares	✓			✓	✓		✓				20	40
MPC	✓		✓			✓			✓		-	-
Lattice		✓	✓		✓				✓		-	-
'Car track' method		✓		✓		✓	✓		✓		20	200

Table 1: Evaluation of path planning methods. Reprinted from (9).

The Table 1 presented on (9) shows a comparison among many path planning methods already employed, including the RPP (or "Two step") used by Laumond on (4), RRT, grid search, MPC and Lattice planners already mentioned in this report. In addition, this paper proposes an alternative method called "Car track method" based on lattice states and local search approaches.

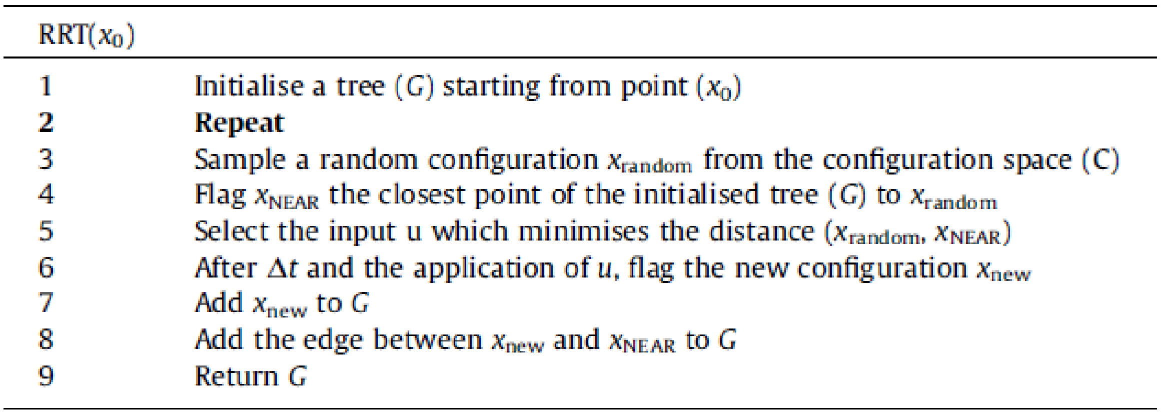
Regardless of lack of information concerning RRT runtime and length of path in the evaluation above, we can state that the algorithm is one of the most applied when planning paths.

In this section, a more detailed explanation of the RRT method will be presented as well as a comparison among the standard algorithm and its variations and improvements possibilities.

The RRT algorithm considers a system composed by a known obstacle region, an obstacle-free space, a goal region and an initial state. It presents basically seven main functions that are described below (10).

1) Sampling: This function randomly samples a state in the obstacle-free space.
2) Nearest: It returns the nearest node from a randomly sampled state according to a cost function. Without differential constraints, the cost will be the Euclidian distance.
3) Steer: The Steer function returns a control input that drives the system from a state to another, commonly from the randomly sampled state to its nearest node.
4) Collision Check: This function is not a part of the RRT, so a collision check method must be chosen and incorporated in the algorithm in order to verify if the planned path lies inside the collision-free space.
5) Near-by vertices: It returns the vertices that are near an input node, generally according to an area or volume function.
6) Insert Node: It inserts the new node to the tree, creating a connection between the node and its parent. It also assigns a cost to the new node.

The pseudo-code and the diagram presented below can explain better the method. In the algorithm, G is the tree topological graph, C is the configuration space, x_{random} is a configuration randomly sampled from C, x_{near} is the vertex which is closest to x_{random} in terms of distance, u is a selected input minimizing the distance between $x_{random}$and x_{near}, and x_{new} is the new configuration.



```
RRT($x_0$)
1    Initialise a tree ($G$) starting from point ($x_0$)
2    **Repeat**
3    Sample a random configuration $x_{\text{random}}$ from the configuration space (C)
4    Flag $x_{\text{NEAR}}$ the closest point of the initialised tree ($G$) to $x_{\text{random}}$
5    Select the input u which minimises the distance ($x_{\text{random}}$, $x_{\text{NEAR}}$)
6    After $\Delta t$ and the application of $u$, flag the new configuration $x_{\text{new}}$
7    Add $x_{\text{new}}$ to $G$
8    Add the edge between $x_{\text{new}}$ and $x_{\text{NEAR}}$ to $G$
9    Return $G$
```

Figure 15: RRT simplified algorithm. Reprinted from (3).

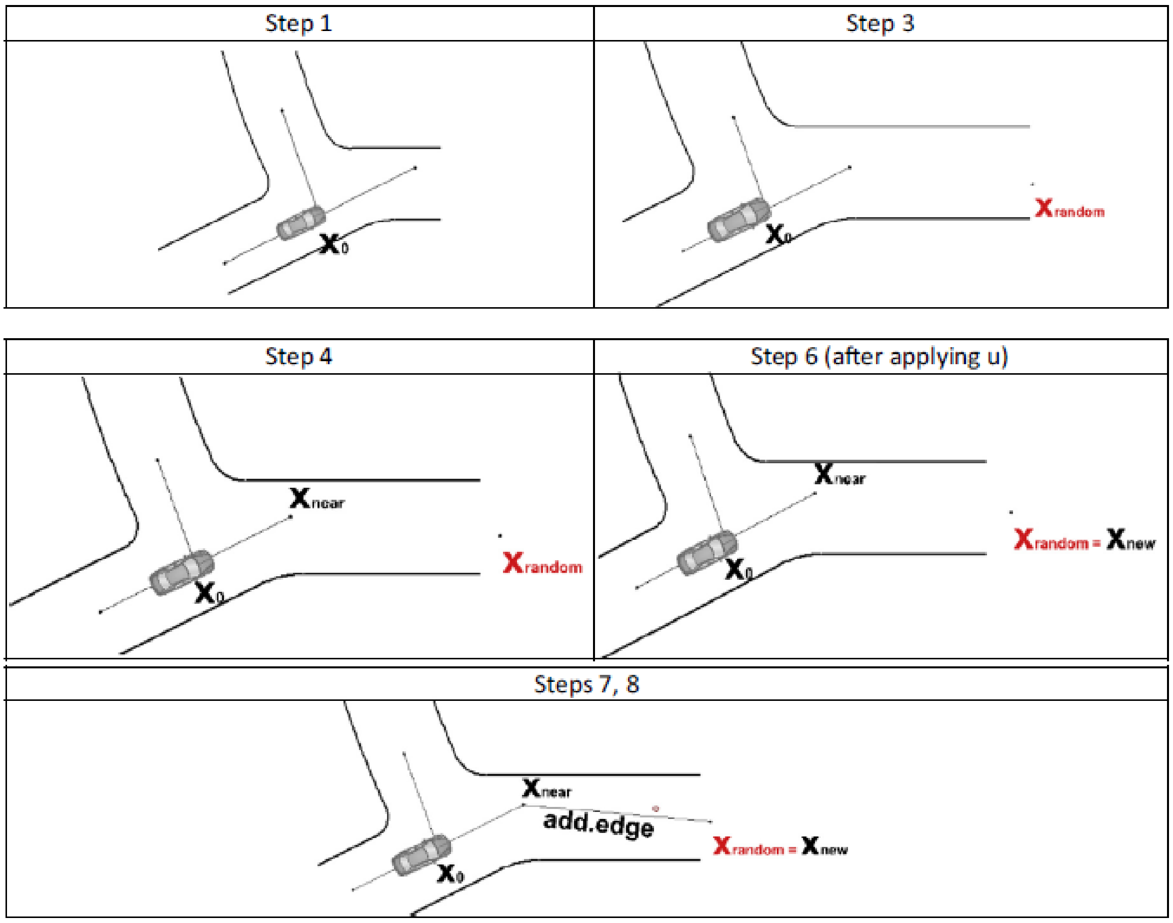


Figure 16: RRT's steps and random tree algorithm. Reprinted from (3).

Briefly, the RRT starts with an empty tree that is incrementally filled by sampling random configurations. The samples are added to the tree choosing as its parent the nearest state in the tree that can be reached with an input. A collision check is done in order to verify if the tree contains a feasible path.

The next subsection will introduce an improvement to the RRT method which proves that a rewiring function is capable to avoid high cost paths as a final solution – a negative point of RRT algorithm.

5.1. RRT*

Regarding to effectiveness, RRT is a sample-based approach that usually relax completeness requirements in order to achieve computational efficiency. Even though it is probabilistically complete, it is proved that its probability of convergence to an optimal solution is actually zero (11). It is explained by the fact that RRT always choose the nearest node as parent of the randomly sampled state. It means the algorithm does not analyze the tree in terms of cost and perhaps the nearest node is not the best choice.

Hence, the algorithm has been improved and a new method has been come up with, the RRT* (called “RRT-star”). This method is an alternative with asymptotic optimality which means that it is almost-sure it will converge to an optimal solution. It makes RRT* an advantageous solution to real-time applications, once it quickly finds a feasible motion plan and also improves this plan toward the optimal solution during the execution time of the current plan.

The biggest difference to the RRT method is that the RRT* considers all the nodes in a defined neighborhood of the random sample and evaluates the cost of choosing each of the node inside this neighborhood as parent of the random sample. It allows a rewiring phase that usually reduces the cost of reaching the sampled nodes.

The neighborhood space is generally defined as a circle of volume $k = \alpha \left(\frac{log(n)}{n}\right)^{d}$, where α is a fixed number and d is the search space dimension.

Furthermore, extensions may improve the RRT* method. For instance, in online applications, the Committed Trajectory (11) technique starts an iterative planning phase just after the initial planning phase is completed. It considers a committed trajectory – a piece of the initial planned path – and while riding it, the vehicle tries to optimize the remaining portion of the trajectory. Thus, the path is optimized iteratively while the vehicle is executing the previously planned path.

5.2. Closed-loop RRT

RRT has been previously employed to autonomous urban driving (12) (13). The main difference in this case, comparing to driftless robots, is the complex and instable dynamics of these vehicles.

To compensate this, RRT can be employed with a closed-loop stabilizing controller (12). The algorithm grows a tree of feasible trajectories originating from the current vehicle state that attempts to reach a specified goal set. It runs a forward simulation using a vehicle model and the controller to compute the predicted state trajectory $x(t)$. Given a reference input r, the controller is used to give high rate commands u taking into account the vehicle dynamics.

Even though the controller is not the goal of this project, the possibility to use the RRT with a closed loop was considered as an advantage.

Each time a difference is observed between the actual position and the predicted position, the closed loop RRT must perform an online repropagation, respecting the committed trajectory whose end coincides with the beginning of the new trajectory to be calculated.

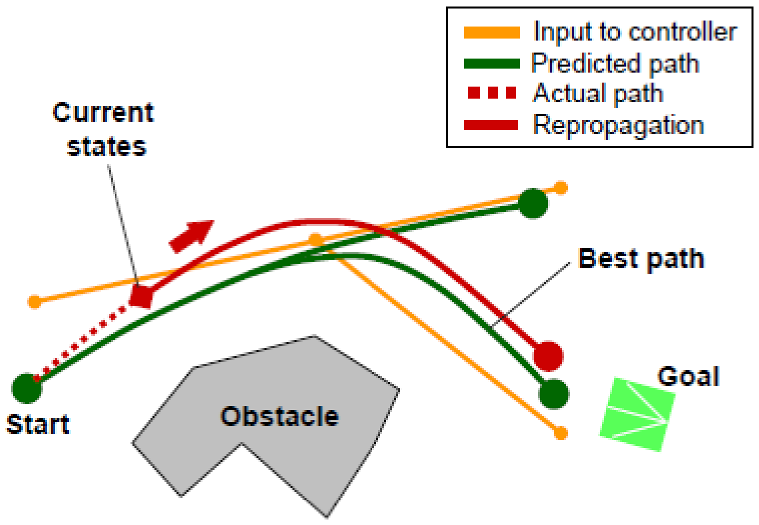


Figure 17: Repropagation. Reprinted from (12).

6. HOLONOMIC RRT*

At a first moment, the algorithm developed was a holonomic path planning (since maneuvers and time were not considered in this first approach, as well as the kinematics constraints of the vehicle), for reasons of learning the structure of a simpler RRT. Also, once the RRT* is a variant of the RRT with some functions in addition, this method has been primarily implemented for the holonomic planning. A further implementation that has taken into accounts the kinematics constraints and degrees of freedom of an articulated vehicle and will be explained in section 7.

The functions of the RRT* has successfully been developed such as Sampling, Nearest Nodes, Node Insertion, Re-wiring and Collision Detection. The latter uses a Ray Casting (14) method to identify if a node is inside a pre-determined obstacle. The method says that a ray starting from a point and pointing to any direction will cross the borders of a polygon an odd number of times if and only if the point is inside the polygon.

In addition, a GUI helps the algorithm to be debugged and visualized in a more intuitive way. At a first glance, it was decided to use the library PySide (a Python alternative for the famous C++ QT library) to develop the graphical features. However, it seemed easier and faster to use the library PyGame which meet greatly the expectations. Moreover, this library has been used several times in other path/trajectory-planning projects.

Concerning the tree exploration, the script was supposed to use a kd-tree, a space-partitioning data structure for organizing points in a k-dimensional space (15). Nevertheless, it was found out that this structure does not allow updates and therefore requires a full complete tree to make a search within. Thus, a structure using simple nodes containing information related to cost (Euclidian Distance through the tree nodes from the start point) and parent node has been used.

The holonomic RRT* algorithm showed some interesting results. The developed example consists in linking a start point (upper left corner at Figure 18) and an end point (lower right corner at Figure 18). For that, nodes are randomly created and inserted in the nodes tree. If needed, the points are successfully "re-wired" as the algorithm imposes. The graphical feature allows clearly a better understanding of the algorithm.

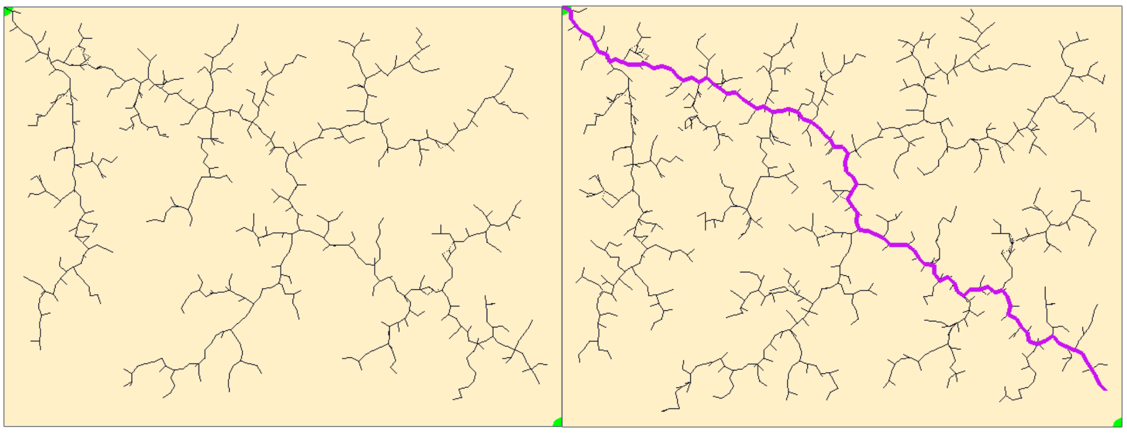


Figure 18: Propagation of the nodes tree (left) and exploration for finding the best final path (right).

These preliminary tests did not include obstacles and consequently were not very conclusive. Therefore, the collision detection was developed and allowed a still better comprehension of what happens when obstacles are added to the environment and what is the expected behavior of the RRT* algorithm (Figure 19)

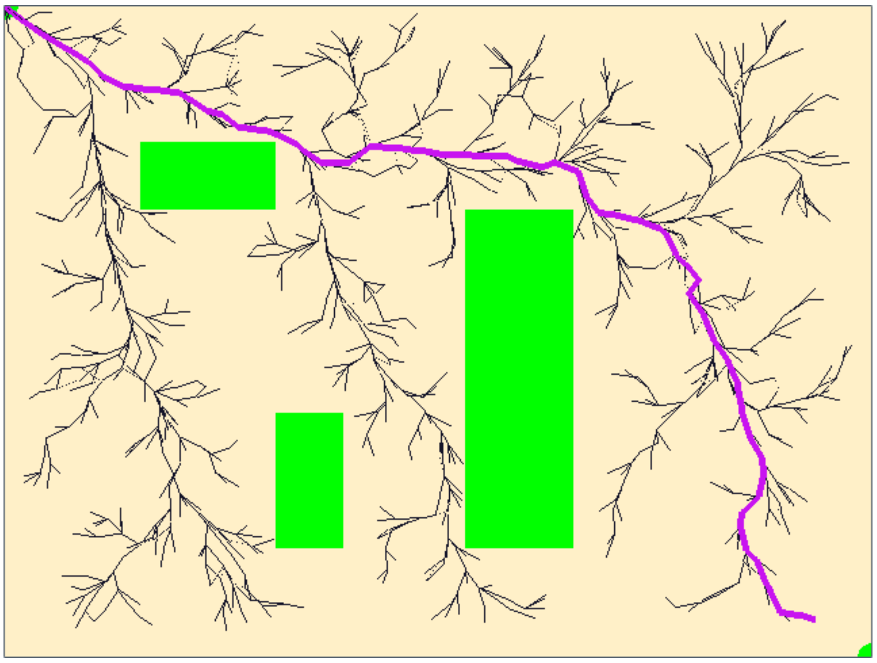


Figure 19: Propagation of a RRT* algorithm in an environment with obstacles (green rectangles).

7. TRAJECTORY PLANNING ALGORITHM FOR NON-HOLONOMIC ARTICULATED VEHICLES

After the holonomic RRT* implementation, the differential equations presented in 4.1 were integrated to the algorithm, transforming it into a non-holonomic planner. It means that the displacement of the articulated vehicle is now bounded by its kinematics equations. Besides that, the whole algorithm was widely modified in order to produce more robust software. It is also important to note that the non-holonomic RRT* is much more complex than the holonomic RRT* once the latter can connect two configurations without taking into account the vehicle's movement constraints. That is why it was decided to implement the traditional RRT for non-holonomic trajectory planning.

The next subsections will present in detail the main functions and classes of the algorithm, as well as the first results of this algorithm.

7.1. Nodes Tree

The RRT algorithm is based on a Tree composed by numerous Nodes. Each Node is connected to another through an Edge. Considering that, a complete Tree data structure has been developed.

The Tree contains a Node Map represented by a Python dictionary. In this Node Map, the array containing the articulated vehicle configuration (x, y, θ, α) is the key and the Node structure itself represents the value. Thus, it is always possible to have access to a Node if its coordinates are precisely known. The entire Tree is also accessible via the Node Map keys. In addition, the Tree permits to add and remove an Edge.

The Edge structure keeps information concerning the path between two Nodes. In this way, an Edge has variables informing its source and its destination, as well as the cost between the two Nodes which is the distance the vehicle needs to travel to get from one Node to another following the outputted path. The necessary inputs (velocity v and tractor's steering angle φ) to get from the source to the destination and the intermediary points between them - once the path is discretized in many points between two Nodes - are also stored.

The Node Structure saves four main contents. First, it contains its configuration as already described. It also knows which Node is its parent, i.e., the Node that comes right before it in the Tree and is bonded to it by an Edge. The distance from the root of the Tree and the Node is also stored. The distance of a Node is always the distance of its parent summed with the cost of the Edge that connects both. Lastly, a Node knows all its adjacent Edges in which it is a source.

7.2. Collision Detection

At a first moment, the project was supposed to use a built-in Python library to treat the collision detection between the vehicle and the obstacles in the surrounding environment. The Box2D (16) – an engine for simulating rigid bodies in 2D – has been considered for that purpose. However, the library has been analyzed and it uses the Axis-Aligned Bounding Box (AABB) (17) method to verify whether there is a collision or not. In this way, it was decided to develop from scratch this collision detection method since it could let the code more fluid and the project would be less dependent on 3rd party libraries.

The AABB method consists on finding for each obstacle or vehicle the axis-aligned minimum bounding box, i.e., the minimum bounding box rectangle in which the edges of the box are parallel to the coordinate axis of the system. That way, the vehicle and the obstacles are all represented by axis-aligned "rectangles", which let the collision verification very easy to be done.

The Figure 20 allows a clear understanding of how the method works. As it can be seen, the AABB method indicates a collision even in situations where the two compared bodies do not collide but only their bounding boxes.

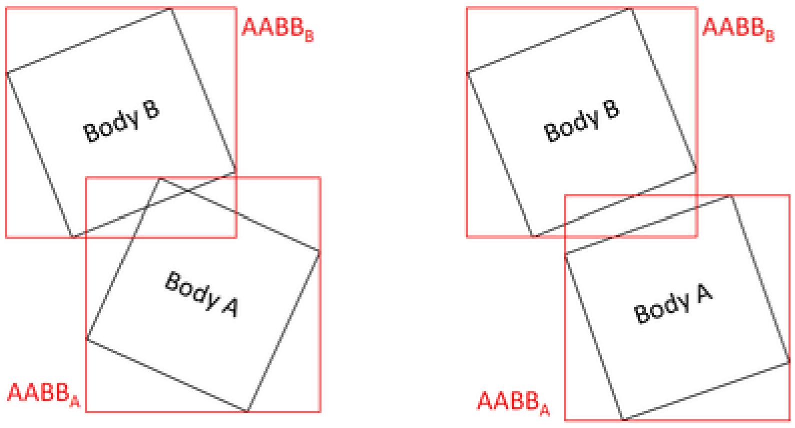


Figure 20: Axis-aligned bounding box. Reprinted from (18).

Theoretically, it could be a problem as the method indicates a "fake collision" even before the actual collision occurs. However, if the problem is considered in real life and not only in simulation, the margin taken into account is actually desirable for two reasons. First, it is not very safe to have the vehicle riding very close to obstacles, so a better path planning solution would not even generate possible paths alongside obstacle. Also, the accuracy of the controller that will drive the trajectory in practice is not known. It means that when inputting the planned trajectory into an articulated vehicle controller, the vehicle would not follow exactly the path. This situation is clearer when comparing the path driven by the Amesim controller and the path planner output. Therefore, a margin space between the vehicle and the obstacles turns out to be necessary.

Concerning the collision detection calculations, it is easily implemented. The obstacles are static bodies, so their bounding boxes are calculated at the beginning of the algorithm and do not change. The articulated vehicle is a dynamic body which means that its configuration changes with time. Therefore, at each time step the bounding box of the vehicle is recalculated. The dimension of the vehicle being constant, the algorithm takes the vehicle's configuration and applies geometric equations in order to find the corners of both tractor and trailer. Once the corners of the two parts of the vehicle (tractor and trailer) are known, the vehicle bounding box is delimited by the maximum and the minimum x and y coordinates amongst all the corners.

Having all the bounding boxes of the environment calculated and aligned, the algorithm does n comparisons between the existent bounding boxes where n is the number of obstacles in the environment. The collision detection is made by verifying the veracity of the following set of inequations:

$$x_{min_{vehicle}} > x_{max_{obstacle}}$$
$$y_{min_{vehicle}} > y_{max_{obstacle}}$$
$$x_{max_{vehicle}} < x_{min_{obstacle}}$$
$$y_{max_{vehicle}} < y_{min_{obstacle}}$$

Where $x_{min/max}$ and $y_{min/max}$ are the minimum/maximum coordinate values of a bounding box, considering all of its corners.

7.3. Trajectory Planner

The trajectory planner follows a traditional non-holonomic RRT implementation. First of all, the Nodes Tree is initialized and the pre-defined start configuration is set as root of the Tree. The algorithm aims to find a trajectory that connects this start configuration to an end configuration. For that, an N number of Nodes are randomly placed on the environment. For each Node, some steps are taken:

7.3.1. Random Configuration

A random configuration $q_{random} = (x, y, \theta, \alpha)$ is created following some boundaries:

$$(x_{min}, y_{min}) < (x, y) < (x_{max}, y_{\max})$$
$$-\pi < \theta < \pi$$
$$-\frac{\pi}{3} < \alpha < \frac{\pi}{3}$$

Where the (x, y) boundaries are related to the pre-defined map and the α boundaries are related to the maximum hitch angle an articulated vehicle may have. In this project, it is limited to 60º. As it can be seen, θ has no boundary limits.

In theory, this random configuration should be connected directly to the Tree as it is done in the holonomic RRT*. However, the non-holonomic case imposes some kinematic constraints once the vehicle displacement follows some defined equations. This way, it would be very costly to find a trajectory starting from an existing Node Tree and finishing in the exact random configuration – obviously respecting the movement constraints. Actually, it consists on the problem this project is trying to find a solution. But to face that in a smarter way, the next step is taken.

7.3.2. Nearest Node

So once the q_{random} is created, the algorithm verifies whether it lies on an obstacle. If it is true, the configuration is rejected and another one is created. If it is not, the code continues and tries to find the Nearest Node $q_{Nearest}$ of the random configuration among the Nodes already contained in the Tree. Then, these two configurations, $q_{Nearest}$ and q_{random} should be connected. Instead, the algorithm apply some inputs (velocity v and tractor's steering angle φ) to $q_{Nearest}$, verifying where the resulting configurations lay down on the environment – called q_{News}. The new configuration q_{New} will not be the random configuration but that one among the q_{News} that is spatially closer to q_{random}, i.e., the configuration that minimizes the distance to q_{random}.

7.3.3. Steering

In order to find q_{News}, a Steering function is called. It is responsible for applying the inputs to an existent configuration, calculating the articulated vehicle movement equations and so generating a new configuration. The inputs are pre-defined on the algorithm and it has been considered that the vehicle will ride in low velocities and turn under low angles. The set of inputs used is:

$$v \in \{1.0\} \quad \left(\frac{m}{s}\right)$$

$$\varphi \in \left\{-\frac{\pi}{24}, -\frac{\pi}{12}, 0, \frac{\pi}{12}, \frac{\pi}{24}\right\} \quad (rad)$$

This way, five different input combinations are used considering the product between v and φ.

Between two configurations, the inputs are applied n times, where $n \in N^*$ and is always defined in relation to the maximum value of φ, in such a way that applying the maximum steering angle input n times, the vehicle will vary its heading in approximately $\dot{\theta} = \pm\frac{\pi}{2}\ rad$. The equation used to find n is:

$$n * \dot{\theta} = n * \frac{v}{L_1} tan\varphi \cong |\pm\frac{\pi}{2}|$$

Where L_1 is the distance between the axles of the trailer.

Applying the inputs n times is useful to discretize the path between two configurations. It means that there will be n intermediate points that indicate more precisely the trajectory between the configurations. When plotting the path, it also provides a better visualization.

7.3.4. Node Insertion

Following with the algorithm, q_{New} is selected among all the q_{News} generated and then the collision condition is verified by updating the vehicle bounding box according to the new configuration. The (x, y) coordinates are also analyzed in order to guarantee that q_{New} lies on the defined map. A last test is also done to guarantee that the Tree does not contain q_{New}, even though it is an improbable situation. The three conditions guaranteed, a new Node is created and q_{New} is assimilated to it. A new Edge is created between $q_{Nearest}$ and q_{New}, its cost is defined as the traveled distance between the two configurations (if φ is equal to 0, the cost is the Euclidian distance between the Nodes), and the intermediate points are stored in the Edge as well as the input set applied. Finally, having the $q_{Nearest}$ distance from the root and the cost of the Edge, it is possible to find and set the q_{New}'s distance from the root as illustrates the equation below:

$$distance(q_{New}) = distance(q_{Nearest}) + cost(Edge_{q_{New}-q_{Nearest}})$$

7.4. Dubins path planning

As discussed in 5, the RRT algorithm randomly creates nodes in order to build a space tree. When planning a trajectory connecting two points it is extremely unlikely for a node corresponding to the final configuration to be randomly generated. Therefore, a method to take the vehicle from a node in the RRT tree to its exact final position is required.

The Dubins path planning is a simple but effective solution to this issue. This solution was first reported by Lester Eli Dubins (19) in 1957, and a description of its equations is presented in (8).

However, the Dubins path method is only applied for the front vehicle, whose position and orientation can be defined. As in the RRT trajectory planning, the orientation of the rear vehicle can be calculated but cannot be controlled as it is only a consequence of the front vehicle's movement.

7.4.1. Dubins car

The Dubins version of a simple car assumes that the vehicle has a constraint on the curvature of the path and that it can only travel forward. If the vehicle can also travel in reverse, then the path follows the Reeds–Shepp curve (8). Since this project deals with articulated vehicles, we will not consider the reverse gear.

The kinematics equations of the Dubins car are the tractor's equations of an articulated vehicle. These equations were already presented in 4.1.

7.4.2. Dubins path segments

Dubins showed that the shortest path for the Dubins car can always be expressed as a combination of no more than three path segments. This combination can be a sequence CCC or CSC, where C is an arc of a circle of radius ρ and S is a straight line segment. Each arc C can represent either a curve to the left (L) or a curve to the right (R). Therefore, there are six admissible Dubins paths: LSL, RSR, RSL, LSR, RLR and LRL.

Note that the radius ρ of the C segments depends on the tractor's steering angle φ and on the length between the axles of the car/tractor L_1.

$$\rho = L_1/tan(\varphi)$$

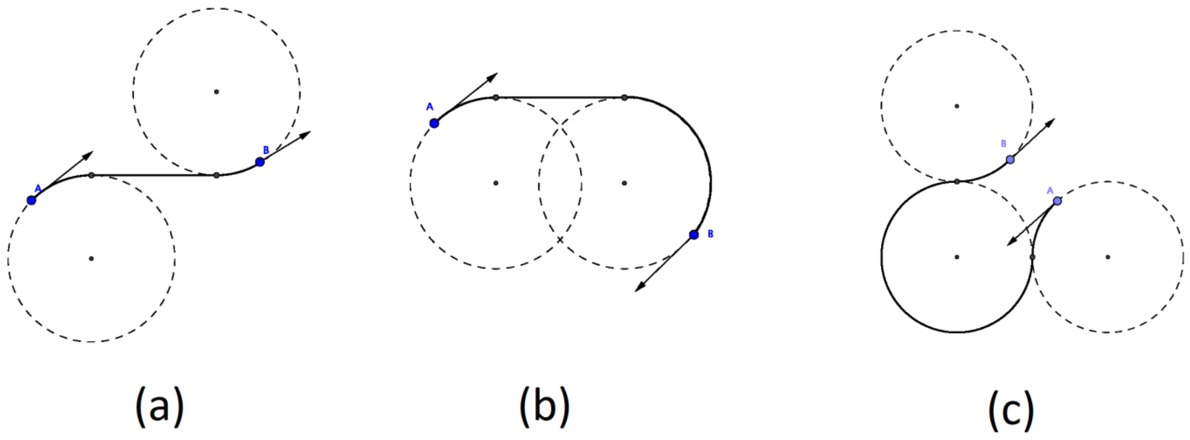


Figure 21: Example of the three Dubins path: (a) RSL, (b) RSR, (c) LRL. Reprinted from (20)

7.4.3. Dubins path planning algorithm

The algorithm developed for this project was based on the algebraic solution published in (21) and inspired in the C library from (22).The goal is to connect an initial configuration q_i to a final configuration q_f of the Dubins car, each one defined by the coordinates (x, y) and the orientation θ of the front vehicle. Note that the relative bend of the rear vehicle α(t) is not taken into account, so a configuration for the Dubins algorithm is different from the configuration considered so far.

$$q = (x, y, \theta)$$

The solution described in (21) consists of a set of equations for each admissible Dubins path. These equations are used to obtain the lengths t, p and q of each constituent segment of the Dubins path. For instance, if the Dubins path is a RSL curve, t is the length of the R segment, p is the length of the S segment and q is the length of the L segment.

However, this solutions was designed considering a normalized state (Figure 22) where the initial and final configuration are $q_i^{norm} = (0, 0, \alpha)$ and $q_f^{norm} = (0, d, \beta)$ and the circle radius is equal to one. In order to apply the equations, we must first translate q_i and q_f to this normalized configuration. At the end, the lengths t, p and q calculated for this particular state must be multiplied by the radius ρ to give the lengths of each segment in the original configuration.

The algorithm calculates t, p and q for every one of the six admissible Dubins path. The path with the shortest total length is elected the optimal Dubins path for the configuration.

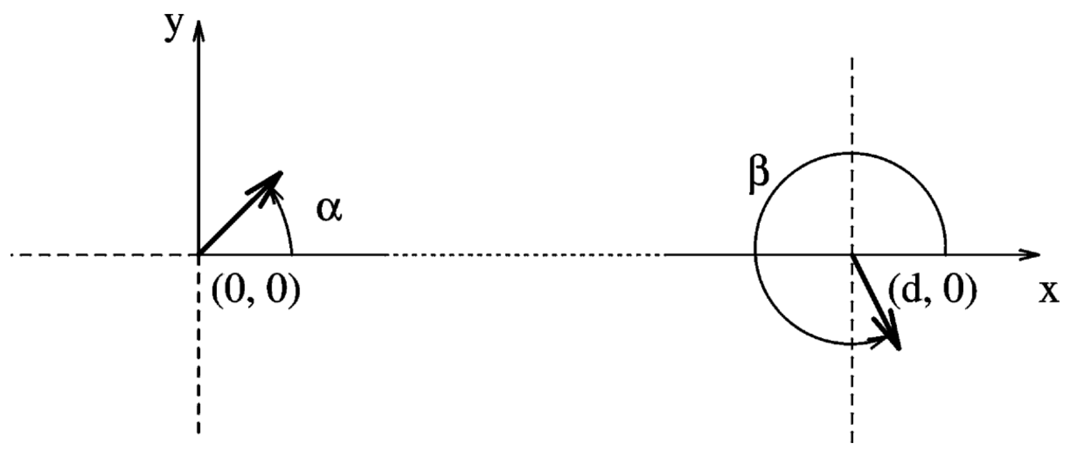


Figure 22: Normalized configuration for the Dubins path equations. Adapted from (21).

With the best Dubins path and its lengths, it is possible to apply the kinematics equations from 4.1 to define the vehicle's states during the Dubins path, i.e., $x(t), y(t), \theta(t)$ and $\alpha(t)$. The information is used to complete the RRT tree and connect the RRT trajectory with the final destination.

7.5. First results

Having the algorithm finished, some tests were done in order to prove the proper functioning of what has been developed. A first example considers a $400m\ x\ 400m$ map where the initial configuration of the articulated vehicle is $(50m, 50m, 0°, 0°)$ and its final one is $(350m, 350m, 0°, 0°)$. There are three known obstacles that the vehicle must avoid. The obtained result is showed in Figure 23:

As it can be seen, the planner really avoids the obstacles with a margin in order to prevent any collision either in simulation or in a possible real-life implementation. The red line indicates the first traced path and the yellow line represents the Dubins path, generated to approximate the vehicle to its final configuration. The vehicle stops at $(349.999m, 349.999m, 0.262°, \alpha)$, where alpha is an arbitrary angle once the Dubins path does not control the trailer's orientation.

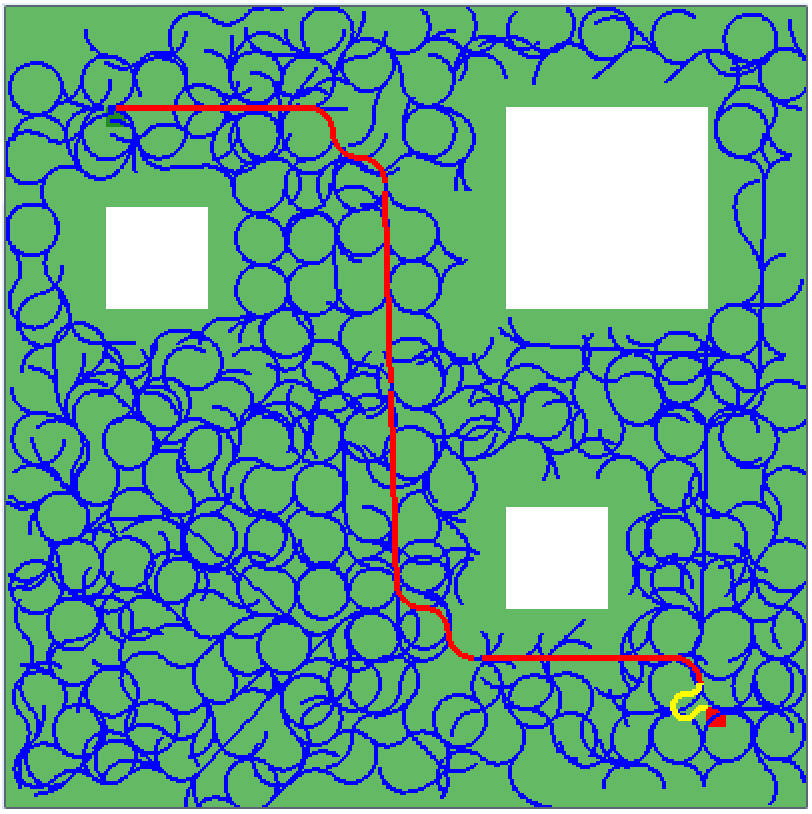


Figure 23: First example planned trajectory.

As a first experiment, it can be concluded that the algorithm presents a very good behavior. The trajectory is continuous and does not contain any cusps or singularities. Also, the output has enough straight lines, meaning that among the entire built Tree, the algorithm has chosen the less costly and curly path. Moreover, the output is not as jerky as the state of the art proposed, even if there are some curves in a row that could affect the lateral behavior of the vehicle.

The outputted Nodes has been inserted into Amesim in order to verify how is the behavior of the vehicle being driven by the Amesim built-in controller, whose inputs come from the developed algorithm. Figure 24 shows the driven trajectory between the points with coordinates $(120m, 50m)$ and $(320m, 250m)$.

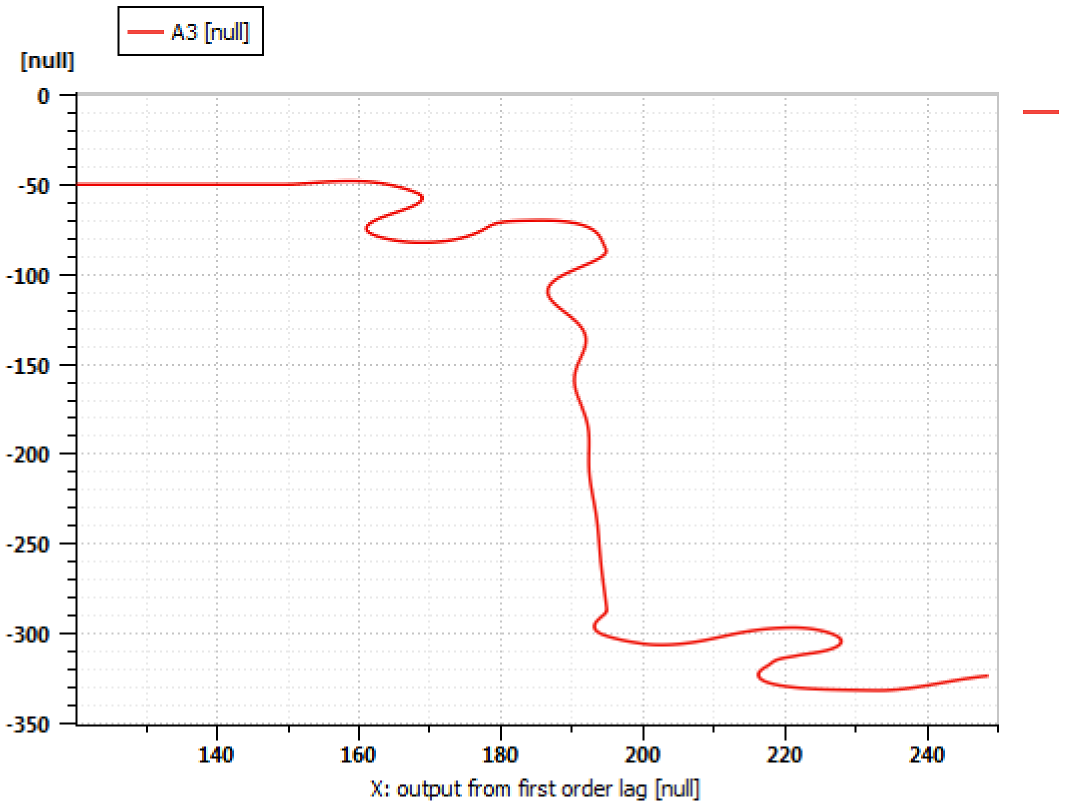


Figure 24: First example generated trajectory from Amesim.

It can be concluded that the controller cannot precisely handle the ride though the proposed path is followed. When performing a curve, the controller takes a long time to react to heading perturbations, and a kind of overshooting seems to let the response to another entry a little slow. The overshooting causes a curve that was supposed to have a heading change of 90º degrees to have a higher change.

Moreover, the two sequences of three curves in a row proposed by the planner should have similar behaviours when followed by the controller. However, the second sequence is a bit more flattened, probably because the errors during the ride are accumulated.

This example evidences the importance of having a correctly tuned controller when trying to reproduce a planned trajectory in real-life (or in simulation at least). As the Amesim controller have two PIDs – in fact, the PID that controls the distance to the reference trajectory has the derivative gain set to 0 and the PID that controls the direction discrepancy has only the proportional gain – if all the gains are not very well tuned, the driven trajectory is expected to be like the results seen in Figure 24. For instance, a derivative gain could make the system react faster and the integral gain could reduce the interference of accumulative errors during time.

8. PROPOSED IMPROVEMENTS

As the first results can exhibit, few point improvements might have been assembled to the proposed solution. It is important to state that there will always be several possible optimizations, once the RRT is constantly researched and many improvements have already been considered. However, four improvements have been considered in this project:

1) The α angle is proposed to tend to 0. A trajectory planning is supposed to control the four components of the articulated vehicle. Though, in practice, great part of useful applications would require both tractor and trailer aligned ($\alpha = 0°$). Besides that, the precise control of the trailer's heading along with the tractor's heading would need the backwards motion to be controlled. The reverse control of an articulated vehicle seems to be a very complex task due to the system instability and additional constraints such as the jackknife avoidance (23), where the hitch point angle increases and the tractor and the trailer fold together.
2) The fewer curves the planner proposes the better in order to have a more realistic trajectory and to prevent error accumulation in Amesim. Therefore, it is proposed a 3-curves optimization in which the planned path is revisited and a sequence of three curves in a row (if it exists) is tried to be reduced to only one curve. For that, the Dubins path algorithm is recalled.
3) The Amesim controller containing the two PIDs may be better tuned in order to have a more flat, realistic and trustworthy response.
4) A Djikstra algorithm to find the shortest paths between nodes in a graph is developed for future implementations.

In this project the first three improvements here proposed were developed and used in order to optimize the first algorithm developed and presented in section 7. The results of these improvements will also be presented and discussed. Then, Djikstra's algorithm will be introduced for future works.

8.1. Reaching α=0

Analyzing the formula of the derivative of α when the steering angle is zero ($\varphi(t) = 0, \forall t$) and the velocity is a positive constant, it can be understood the behavior of the trailer when riding a straight line:

$$\dot{\alpha}(t) = v(t) * \left[\frac{tan\varphi(t)}{L_1} - \frac{\sin\alpha(t)}{L_2}\right] = -\frac{vsin\alpha(t)}{L_2}$$

It is a first order non-linear ordinary differential equation of kind $y' + k * siny = 0$, where k is $\frac{v}{L_2}$. With help of a mathematical solver (24), its solution can be found and corresponds to:

$$\alpha(t) = 2\cot^{-1}(e^{c_1 + kt})$$

The inverse of the cotangent function tends to zero when the time tends to infinity. Therefore it proves that more the articulated vehicle rides forward in a straight line more α approximates zero, the proposed configuration.

This way, the developed optimization proposes to the algorithm to plan, if possible, a trajectory where the end configuration is 20 meters distant from the actual goal and the tractor is aligned with the desired orientation. After that, the vehicle only needs to drive forward 20 meters and whatever is the value of α, it will tends to zero. The straight line distance is variable, meaning it could be a shorter path in case the map does not have a free 20 meters space around the final configuration.

8.2. Curves optimization

The curves optimization works after the planner has already found the complete trajectory between the start and the end configurations. Then, the optimization step analyzes every four subsequent Nodes contained in the final path. If it finds three curves in a row (generally, RLR or LRL, using the notation presented 7.4.2), the algorithm tries to connect the first Node with the last Node using the Dubins path algorithm. This way, if the steering angle proposed to the Dubins planner is the correct one, the originated Dubins path will contain either two short curves and a long straight line or a single long curve, a short straight line and a short curve (if a CSC is the minimum cost path calculated by the Dubins planner).

As it is difficult to estimate the best steering angle for an arbitrary set of curves, eight Dubins paths are generated each one under a different steering angle. The steering angles are between $0.2 * (R_1 + R_2 + R_3)$ and $2 * (R_1 + R_2 + R_3)$, where R_1, R_2, R_3 denote the radius of each curve. In fact, this is a good approximation. For example, if we consider three curves with the same radius R, they could be replaced by a single curve of radius $3R$ approximately, which lays inside the boundaries of the tested steering angles. The Figure 25 illustrates better the given example:

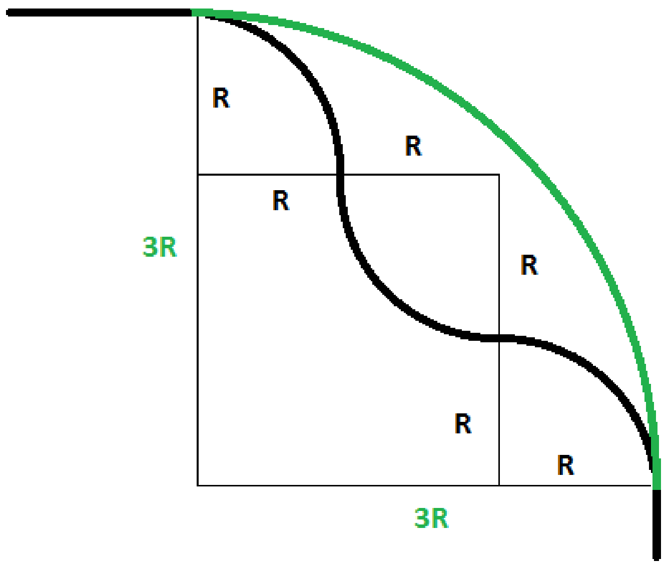


Figure 25: Three opposite curves being replaced by a single one.

The lengths of the generated paths are then compared and the shortest path is selected as optimal. If this path is shorter than the original 3-curves length, the three curves are replaced in the Tree by the Dubins path output.

This optimization shows that even after having the final path, many optimizations might be done by reconnecting Nodes in different ways. A complete optimization would not only consider three subsequent curves, but would compare each Node of the path with the rest of the entire path, trying to find the lowest cost. This project has not considered a large scope of optimization algorithms, but this one can show that there are plenty possibilities to find a better trajectory.

8.3. Results with proposed improvements

When applying the optimizations described in 8.1 and 8.2, clear changes may be seen in the trajectory planned. The figure below describes a simulation whose results are similar to that presented in section 7.5 but with the optimizations applied. The final goal continues being $[350.0m,\ 350.0m,\ 0.°,\ 0.°]$, the simulation time was 60.446s and the optimization time, 1.636s.

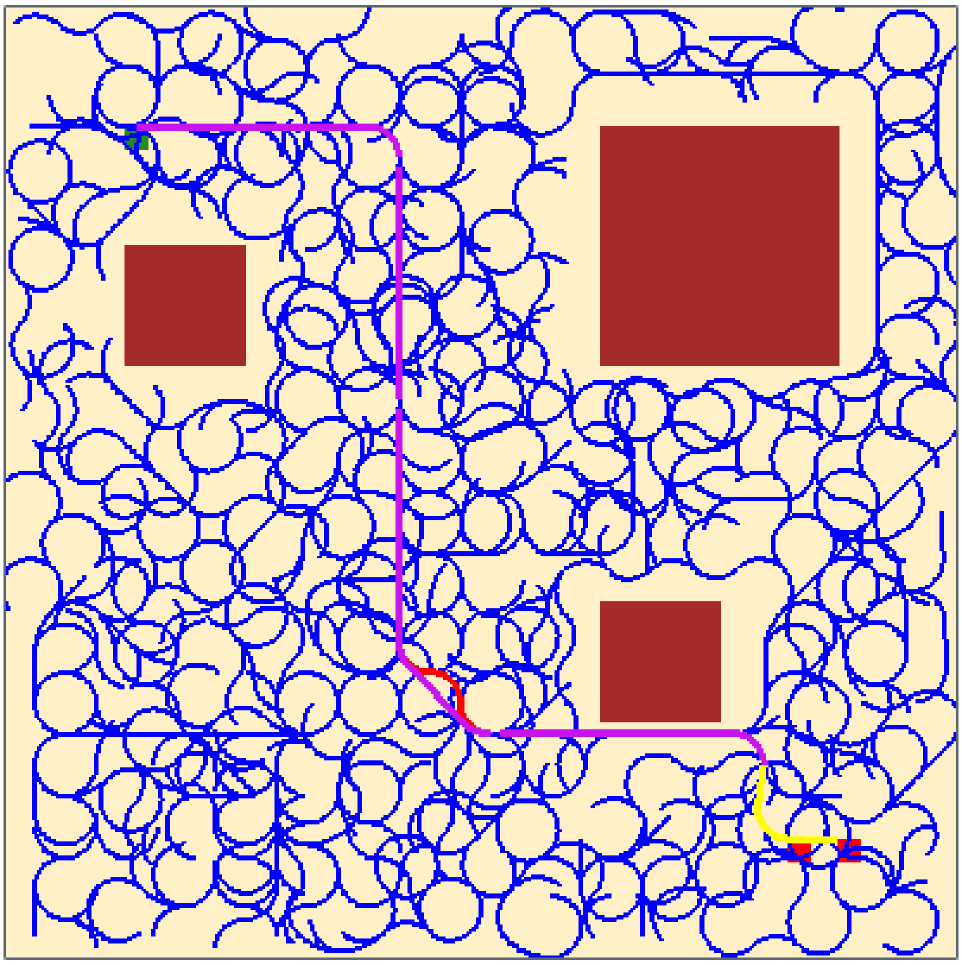


Figure 26: Optimized planned trajectory.

The yellow line keeps representing the Dubins path, but it also represents the final straight line that makes α tends to zero. The first planned trajectory is still represented in red, but the optimization traces a new purple trajectory over it. Near the middle of the figure, a set of three curves is replaced by a new Dubins path, which generates two short curves and one long straight line just as explained in 8.2. This time, the vehicle reaches the final position at $(349.985\,m, 350.024\,m, 0.°, -0.026°)$. The precision is mainly improved when analyzing the orientation of the vehicle. Now, the α angle has a precision in the order of hundredth degrees.

It can be verified that the trajectory is very close to one of the obstacles. It may be a problem when simulating in Amesim, but conclusions may be taken only after simulation.

For that purpose, the generated path is converted in $(x, y, radius)$ points that are inputted in Amesim. The simulation runs through 600 seconds and the variables are printed each 0.05 second, even though the integrator is set with a variable step. The obtained trajectory is showed in Figure 27:

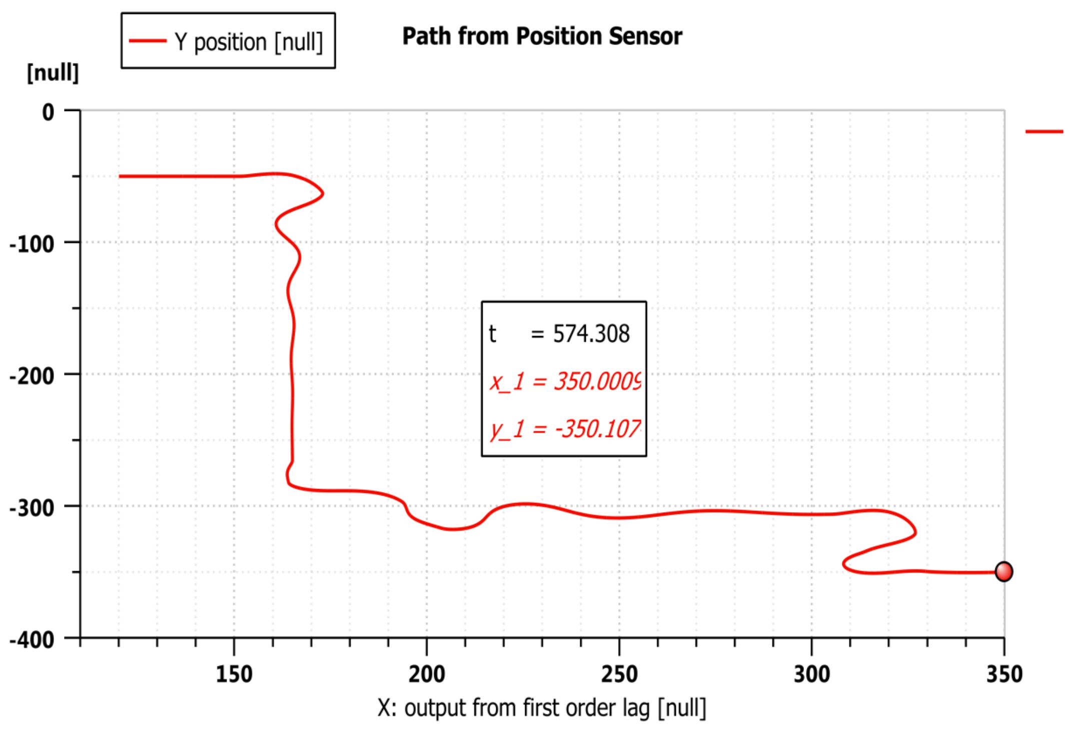


Figure 27: Optimized trajectory generated in Amesim.

As it can be easily verified, the results are way better than the first results obtained. The number of curves – and mainly subsequent curves – is reduced, so the vehicle behaves in a more realistic way once the controller has time enough to control the errors in relation to the reference trajectory.

Also, analyzing the final point at 574.308 seconds, the vehicle stops at the Cartesian position of $(350.0009\ m, 350.1074\ m)$. It means the controller can precisely handle the task of reaching its goal. Indeed, the position error is in the order of only 10 centimeters.

Moreover, the orientation of the tractor $(\theta = \theta_1)$ and the trailer (θ_2) can be observed as well as the α angle representing the relative bend of the trailer. Figure 28 below shows not only these three angles, but also its derivatives, i.e, the rotary velocity correspondent to each angle:

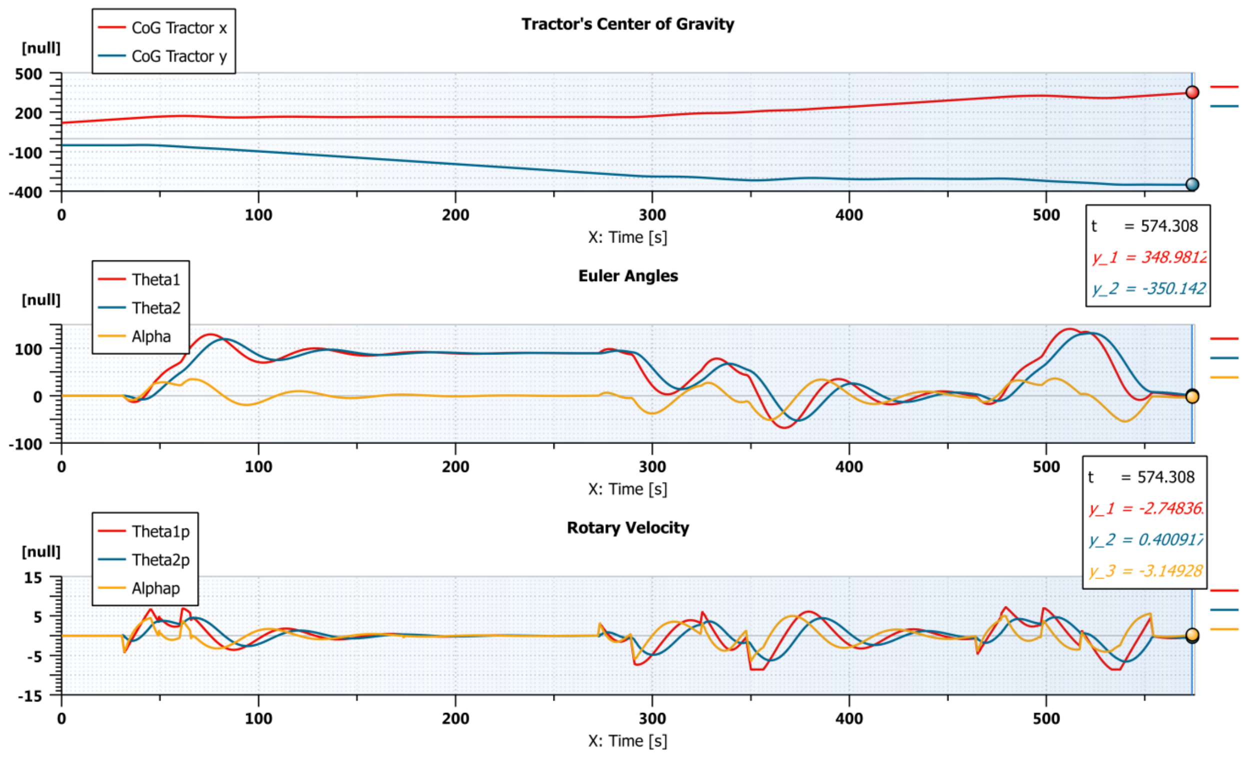


Figure 28: Euler angles and rotary velocities of the tractor and the trailer vehicles

The final orientation of the tractor values $\theta_1 = -2.74836°$, the final orientation of the trailer values $\theta_2 = 0.400917°$ and the difference between them is $\alpha = -3.14928°$. Therefore, comparing these values with the expected results $(\theta_1, \theta_2, \alpha) = (0.°, 0.°, 0.°)$, it can be concluded that the controller can also reach the angle goals with a good precision. The total error is approximately in the order of 4 degrees.

Concerning the rotary velocities, it may be observed that the orientation angle variations $(\dot{\theta})$ do not exceed 10 degrees/s and the hitch angle variation $(\dot{\alpha})$ does not exceed 5 degrees/s. Therefore, the rotary velocities keep inside a good safety margin and do not attain huge values that could compromise the hitch between the tractor and the trailer.

A last analysis consists of verifying if the vehicle collides with the obstacles or not since the planner generated a path in which the vehicle gets very close to one of them. For that, Figure 29 shows the path of all the corners of the vehicle as well as its center of gravity. A zoom in the critical part of the path is done in order to let it easier to analyze.

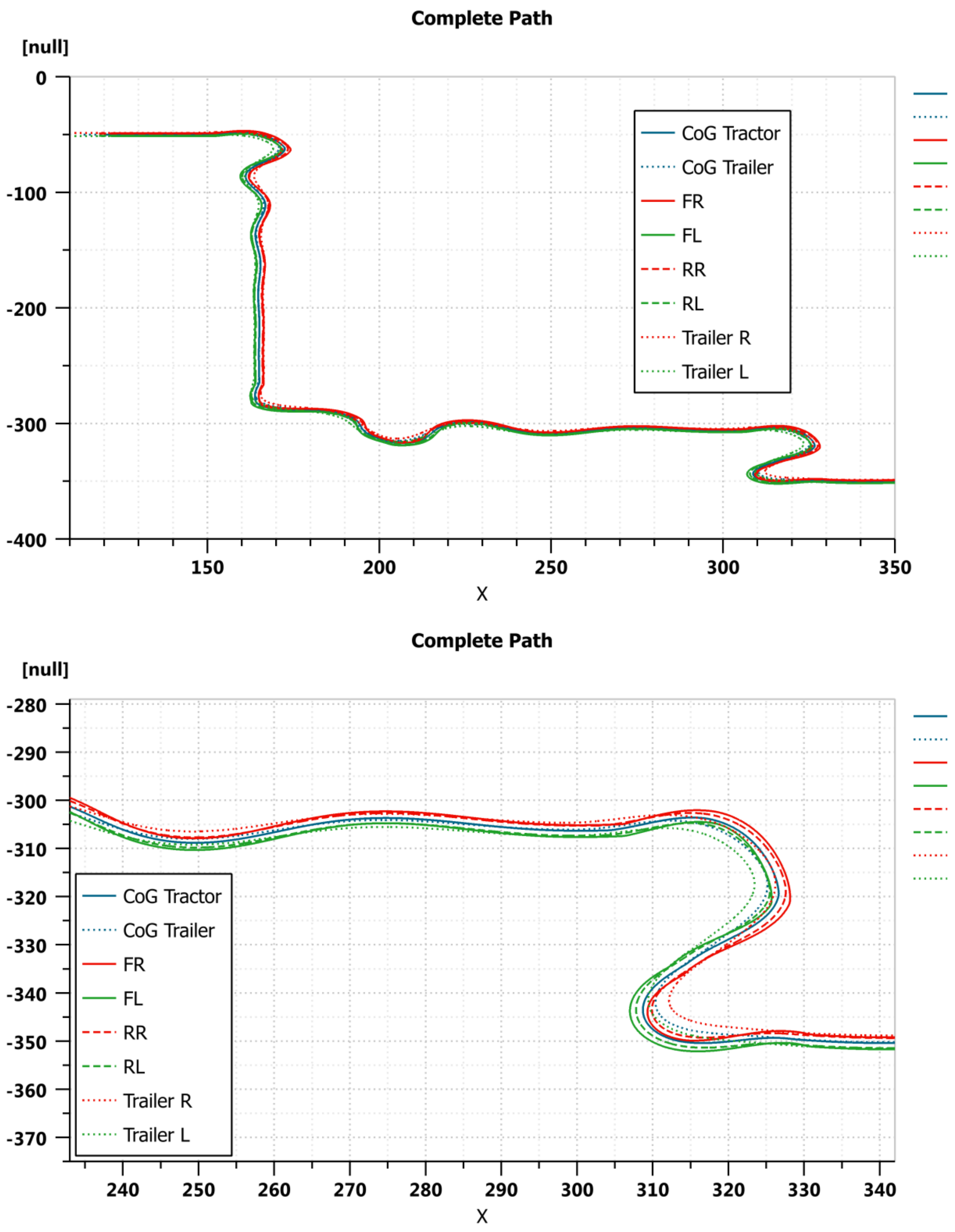


Figure 29: Completed description of the driven path by the center of gravity of the vehicle as well as all of its corners. (CoG: center of gravity, FR: tractor's front right corner, FL: tractor's front left corner, RR: tractor's rear right corner, RL: tractor's rear left corner, Trailer R: trailer's rear right corner, Trailer L: trailer's rear left corner)

The boundaries of the nearest obstacle are $[(250, 250), (250, 300), (300, 300), (300, 250)]$, therefore there's no collision. Indeed, it can be seen that the closet point to the obstacle is $(274.6088\, m, 302.2755\, m)$ in which the left side of the tractor drives a little bit more than 2 meters distant of the obstacle. Perhaps, if the generated trajectory was curlier, the Amesim controller would not have been able to avoid the obstacle and a collision would have happened. It highlights the importance of safety when planning a vehicle trajectory and shows that the planner should be very careful when avoiding obstacles, as well as the vehicle controller that needs to be prudent and precise.

8.4. PIDs parameters optimization

Even though the Amesim controller contains two PID controllers whose outputs are summed in order to control the steering angle, this optimization tries to tune better only one of them, the PID that controls the distance to the reference trajectory. Until now, this PID was actually a PI, where $\boldsymbol{k_p} = \mathbf{50}$ and $\boldsymbol{k_i} = \mathbf{0.1}$, the derivative gain was set to zero.

Some tuning techniques could be used to find the PID gains such as the Ziegler-Nichols method. However, as already mentioned, the built-in Amesim function does not contain two pure PIDs solely, but it is actually a block that tries to make several corrections. This way, a fine tuning would consider the whole block and not only one PID. Thus, a manual tuning was decided to be done, inputting different gain values, simulating them and comparing their results.

The next image illustrates one of the comparative simulations done. The original PI has been compared to a pure PD and two other PIDs in order to try to identify the influence of each gain in the controller behavior.

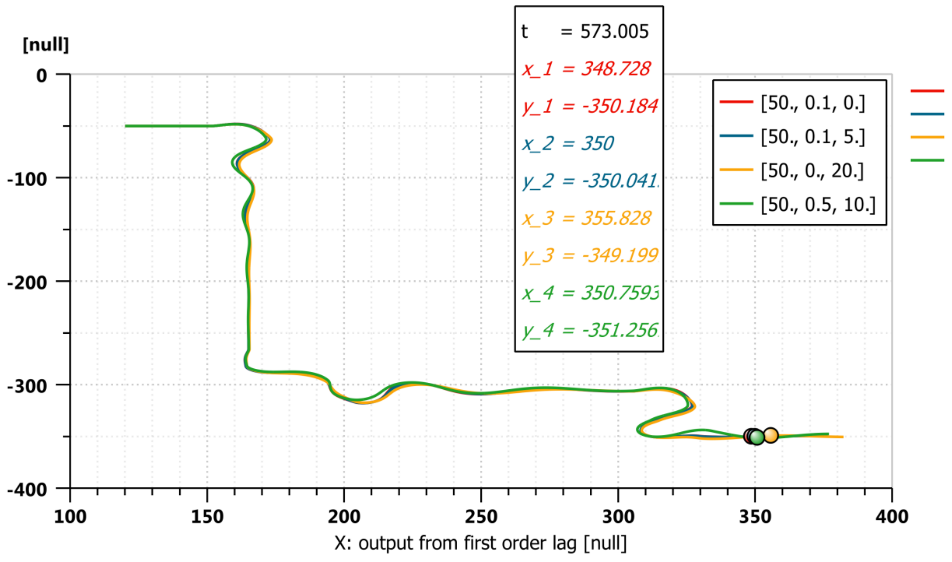


Figure 30: Comparison among the simulated PIDs

The pure PD ($\boldsymbol{k_p = 50, k_i = 0, k_d = 20.}$) and one of the PID's ($\boldsymbol{k_p = 50, k_i = 0.5, k_d = 10.}$) presented an additional error in relation to the reference trajectory, even if they seemed to have a less intense overshooting. The final point where $x = 350\ m$ is further from the goal if compared with the original PI.

The other PID ($\boldsymbol{k_p = 50, k_i = 0.5, k_d = 5.}$) has a behavior very similar to the original PI. Indeed, their differences consist only on the added derivative gain. The generated paths present almost the same behavior. But when comparing the final point, the PID manages to get closer to the goal. As the figure shows, its final point is $(350\ m, 350.041\ m)$, closer than the final goal of the PI, $(350.0009\ m, 350.1074\ m)$. The final angles of the PID maintained almost unchanged if compared to those of the PI. Even if the difference is minimal, this comparison shows that even the controller can be improved in order to guarantee a trustier driven path.

8.5. Djikstra's algorithm

The trajectory planning problem often includes map discretization. It means that the environment is discretized once and then its configuration is stored. Whenever a new trajectory planning is needed, the discretization is not necessary anymore because the map is stored. It allows choosing a different start point at each simulation and does not impose a fixed one. Moreover, the map could present circularities, so a Node would not have only one parent as the developed algorithm requires. At this point, the Djikstra's algorithm is fundamental in order to find the best solution. When a Tree allows multiple parents, it faces the "shortest path" problem and the Djikstra's algorithm may solve it. For the moment, the developed Djikstra's algorithm is not implemented because the project has restricted its application to cases in which the start and the end points are known as well as the surrounding environment but could certainly be used in a future work.

This algorithm is commonly used for finding the shortest paths between nodes in a graph. Normally, the algorithm has a computational cost equal to n^2 (25), where n corresponds to the number of nodes. However when the algorithm is implemented in parallel with a priority queue (26), the computational cost is considerably reduced which makes this kind of search a very powerful method to use in a trajectory planning problem.

In this project, a Djikstra's algorithm has been developed along with a binary "min heap", which turns out to be a priority queue method. A min heap is a tree-based data structure that satisfies the following property: if P is a parent node of node C, then the value of node P is less than or equal to the value of the node C (27). So the lowest value is always the root's value. A heap is generally constructed using a single array alone. The first element contains the root, the next two elements contain its children, the next four elements contain their children and so on. So, the children of a node at position k will be at position $2k + 1$ and $2k + 2$.

The Djikstra's algorithm has a quite simple logic. A node is defined as the source and the algorithm will find the shortest path to all the other nodes of the tree. For this, the algorithm initially assigns the distance between the source and all other nodes as infinity. Then, it iteratively visits the nodes, calculates a tentative distance to them and assigns this distance to the node if the distance is smaller than its current one.

9. CONCLUSION

The intention of this project has always been to work with cutting-edge technologies and recent subjects that are currently being studied. The Advanced Driver-Assistance Systems are surely technologies that will be responsible to cause several changes in the future world and modify the way the society interacts with its environment. Thanks to the industry, represented here by Siemens, the project could receive insights and the context of the project was structured better, leading the students to work with a real problem faced by many companies and researchers: the trajectory planning for articulated vehicles.

The three committed parts– students, industry and university – decided to study the existent techniques of trajectory planning and develop a tool able to generate feasible trajectories for an articulated vehicle given a known environment. After an extensive analysis of the state of the art, it was decided that the Rapidly-exploring Random Tree (RRT) would be the guideline method in order to develop the planner. Due to its relevance in the trajectory planning world, probabilistic completeness, kinematics feasibility and several optimizations methods, the method was implemented and confirmed its capacity of providing solid results.

Moreover, once the project involves articulated vehicles, a kinematics study of this kind of vehicle was necessary. Describing its movement correctly and identifying the kinematics constraints is a main role to guarantee a feasible solution. The vehicle modeling also permitted the understanding of different degrees of details when doing a simulation. For instance, the Amesim software proved that a much more complete modeling could be done, allowing the analysis of several different variables as well as the understanding in a trustworthy way what would be the behavior of a real vehicle if it tried to ride the generated trajectory.

The kinematics modeling and the planner algorithm were assembled in order to generate a final solution. The results were very satisfactory and the presented use case could prove it. The planner was able to generate a feasible, realistic and accurate trajectory, with a precision in the order of hundredth of centimeters and degrees. The final results in the Amesim simulation obtained an accuracy of 10cm in relation to the Cartesian coordinates and less than 4 degrees in relation to the orientation angles of the vehicles, proving that it is very important to have a trusty hardware. Both Amesim and algorithm simulations managed to avoid collision with environment obstacles.

Also, the algorithm could be also optimized which is a great achievement once the optimization is a very important part in any software development. Three main improvements were implemented and used in this project. It included tending the α angle to zero, reconnecting curves of the generated trajectory and tuning the PID contained on the Amesim controller responsible for following the path outputted by the planner.

Therefore, it can be concluded that the project have done a great part of what it has proposed at the beginning of its timeline. Doubtlessly, it still leaves space for future works in which more optimizations could be done such as path reconnections and smoothing, modeling of the reverse movement of an articulated vehicle and Amesim controller improvements in order to find the best PID parameters that rides a more realistic path. A hardware implementation could be aimed in order to compare the Amesim simulation with real-life test. Finally, the developed solution could be used in order to study a real use case where the generation of trajectories for articulated vehicle is very important.

REFERENCES

1. Autonomous cars must progress through these 6 levels of automation. [Online] [Cited: December 11, 2016.] http://safety.trw.com/autonomous-cars-must-progress-through-these-6-levels-of-automation/0104/.

2. **Siemens.** LMS Imagine.Lab Amesim: Siemens PLM Software. [Online] [Cited: December 11, 2016.] http://www.plm.automation.siemens.com/en_us/products/lms/imagine-lab/amesim/index.shtml.

3. **Katrakazas, Christos, et al., et al.** Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies.* November 2015, Vol. 60, pp. 416-442.

4. **Sekhavat, Sepanta , et al., et al.** Motion planning and control for Hilare pulling a trailer: experimental issues. *IEEE international conference on robotics and automation.* 1997, pp. 3306-3311.

5. **Barraquand, Jerome and Latombe, Jean-Claude.** Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research.* 1991, Vol. 10, 6, pp. 628-649.

6. **Svestka, Petr and Vleugels, Jules.** Exact motion planning for tractor-trailer robots. *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on.* 1995, Vol. 3, pp. 2445-2450.

7. **Elhassan, Amro.** *Autonomous driving system for reversing an articulated vehicle.* 2015. Dissertation.

8. **LaValle, Steven M. .** *Planning algorithms.* s.l. : Cambridge university press, 2006.

9. **Rimmer, Amy J. and Cebon, David.** Planning Collision-Free Trajectories for Reversing Multiply-Articulated Vehicles. *IEEE Transactions on Intelligent Transportation Systems.* July 2016, Vol. 17, 7, pp. 1998 - 2007.

10. **Iram, Noreen, et al., et al.** A Comparison of RRT, RRT* and RRT*-Smart Path Planning Algorithms. *International Journal of Computer Science and Network Security (IJCSNS).* 2016, Vol. 16, 10, p. 20.

11. **Karaman, Sertac, et al., et al.** Anytime Motion Planning using the RRT*. *Robotics and Automation (ICRA), 2011 IEEE International Conference on.* 2011, pp. 1478-1483.

12. **Kuwata, Yoshiaki, et al., et al.** Real-Time Motion Planning With Applications to Autonomous Urban Driving. *IEEE Transactions on Control Systems Technology.* September 2009, Vol. 17, 5, pp. 1105 - 1118.

13. **Peng, Cheng, Zuojun, Shen and La Valle, S.** RRT-based trajectory design for autonomous automobiles and spacecraft. *Archives of control sciences.* 2001, Vol. 11, 3/4, pp. 167-194.

14. Ray-casting algorithm. *Rosetta Code.* [Online] https://rosettacode.org/wiki/Ray-casting_algorithm.

15. k-d tree. *Wikipedia, the free encyclopedia.* [Online] [Cited: June 29, 2017.] https://en.wikipedia.org/wiki/K-d_tree.

16. **Catto, Erin.** Box2D - A 2D Physics Engine for Games. [Online] http://box2d.org/.

17. Minimum bounding box. *Wikipedia, the free encyclopedia.* [Online] [Cited: November 2017, 04.] https://en.wikipedia.org/wiki/Minimum_bounding_box#Axis-aligned_minimum_bounding_box.

18. **Eichner, Hubert.** Game Physic - 2D Collision Detection. *myselph.de.* [Online] 2014.

19. **Dubins, Lester.** On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics.* 1957, Vol. 79, 3, pp. 497-516.

20. Dubins path. *Wikipedia, the free encyclopedia.* [Online] [Cited: November 15, 2017.] https://en.wikipedia.org/wiki/Dubins_path.

21. **Shkel, Andrei M. and Lumelsky, Vladimir.** Classification of the Dubins set. *Robotics and Autonomous Systems.* 2001, Vol. 34, 4, pp. 179-202.

22. **Walker, Andrew.** Dubins-Curves: an open implementation of shortest paths for the forward only car. [Online] 2008. [Cited: November 2, 2017.] https://github.com/AndrewWalker/Dubins-Curves.

23. **Chiu, Jimmy and Goswami, Ambarish.** The critical hitch angle for jackknife avoidance during slow backing up of vehicle–trailer systems. *Vehicle System Dynamics.* 2014, Vol. 52, 7, pp. 992-1015.

24. *WolframAlpha.* [Online] [Cited: November 11, 2017.] https://www.wolframalpha.com/input/?i=y%27(t)+%2B+k*sin(y(t))+%3D+0.

25. Dijkstra's algorithm. *Wikipedia, the free encyclopedia.* [Online] [Cited: September 21, 2017.] https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm.

26. Priority queue. *Wikipedia, the free encyclopedia.* [Online] [Cited: September 21, 2017.] https://en.wikipedia.org/wiki/Priority_queue.

27. Heap (data structure). *Wikipedia, the free encyclopedia.* [Online] [Cited: September 21, 2017.] https://en.wikipedia.org/wiki/Heap_(data_structure).