UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

A trajectory deformation algorithm for intelligent vehicles

Victor Hugo Sillerico Justo

Dissertação de Mestrado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP SÃO CARLOS

Victor Hugo Sillerico Justo

A trajectory deformation algorithm for intelligent vehicles

Dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Master in Science. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Fernando Santos Osório

USP – São Carlos
July 2023

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

J96t

Justo, Victor Hugo Sillerico
A trajectory deformation algorithm for intelligent vehicles / Victor Hugo Sillerico Justo; orientador Fernando Santos Osório. -- São Carlos, 2023.
93 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2023.

1. ROBÔS MÓVEIS INTELIGENTES. 2. ROBÓTICA . 3. VEÍCULOS AUTÔNOMOS . 4. ALGORITMOS . I. Osório, Fernando Santos , orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Victor Hugo Sillerico Justo

Um algoritmo de deformação de trajetória para veículos inteligentes

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Fernando Santos Osório

USP – São Carlos
Julho de 2023

This work is dedicated to my grandmother Patricia, to my mother Jeannette, to my sister Karen, and to my brother Alejandro.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank God for seeing me through this challenging academic adventure, and for always showing me a light of hope in times of difficulty, worry, and anguish. Next, I would like to thank my loving family for always motivating me in accomplishing my goals, and for inspiring me to continue improving every day.

I want to express my deepest and most sincere gratitude to my advisor, Fernando Santos Osório, for providing me with an incredibly welcoming experience at the Mobile Robotic Laboratory (LRM). Words cannot express how thankful I am for his level of patience, guidance, encouragement, and insight. His unconditional support and trust, comparable to that of a Father, were essential during my studies in Brazil, and subsequent training in Canada.

I thank the National Council for Scientific and Technological Development (CNPq) for the financial support to carry out my research activities in Brazil. I also thank the Department of Foreign Affairs, Trade and Development (DFATD) and the Government of Canada for providing the funding for my research stay at the University of Manitoba (UofM).

I want to thank my laboratory colleagues who were always cordial and attentive to me. Special thanks to Viviana, Carlos and Alberto for their guidance and good advice. I also want to thank Luis and Junior for their friendship, and for always being willing to share their experience and technical knowledge with me, their support was essential in directing my research work.

My sincerely appreciation to my friend and mentor Pablo Zamora who encouraged and helped me a lot to apply to the master program at ICMC-USP. My appreciation as well to my friend Paulo Loma who was always available to give me valuable advice in difficult times.

Finally, I would like to thank my supporting friends in São Carlos: Omar, Devis, Mariano, Mario, Carlos, Angelo, Victor, Leonel, Elvis, Drahcir, and Andres. They helped me in the challenging moments and made my life easier when I started this academic journey.

“Intelligence is the ability to adapt to change.”
(Stephen Hawking)

RESUMO

JUSTO, V. H. S. **Um algoritmo de deformação de trajetória para veículos inteligentes**. 2023. 93 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2023.

Veículos autônomos exigem algoritmos de planejamento robustos para calcular a sequência de movimentos de um ponto de partida a um objetivo final, considerando as restrições do ambiente. É desafiador garantir manobras seguras em todos os cenários de tráfego possíveis e o módulo de planejamento de movimento recalcula as trajetórias inicialmente planejadas quantas vezes forem necessárias para resolver essas situações complexas. No entanto, o custo computacional aumenta quando o processo de planejamento é repetido muitas vezes para a mesma tarefa e as soluções atuais não permitem vincular as preferências do usuário ao comportamento de movimento do veículo. Uma alternativa é gerar novas trajetórias com base em trajetórias planejadas já disponíveis. Propomos um algoritmo que leva em conta fórmulas de Lógica Temporal de Sinal (STL) que representam as restrições impostas pelo usuário para modificar trajetórias inválidas e orientar o planejamento do movimento a respeitar requisitos de segurança como distância mínima a obstáculos estáticos ou entre veículos. Usamos um planejador baseado em reticulados para gerar caminhos candidatos e incluímos um recurso de multi-resolução para gerar quantos reticulados forem necessários dependendo do contexto. Então, o valor de robustez STL quantifica o nível de respeito que os caminhos iniciais têm pelas especificações STL e ativa o processo de reparo que gera novos reticulados com base no caminho inicial selecionado. A medida de robustez também define uma nova resolução para gerar reticulados e influencia a função custo para garantir a seleção do caminho que mais respeita as fórmulas STL. A versão deformada do reticulado inicial é usada para gerar a trajetória para um horizonte de planejamento especificado usando uma abordagem de simulação. O custo computacional da estratégia de reparo proposta é menor do que recalcular a trajetória completa do zero e é especialmente conveniente quando não há muitas violações de regras próximas à região do objetivo.

Avaliamos nossa abordagem usando as ferramentas automotivas do simulador de robôs Webots considerando diferentes cenários de tráfego envolvendo desvio de obstáculos. A eficiência do nosso método é demonstrada comparando trajetórias usando restrições STL com trajetórias que não consideram regras STL.

Palavras-chave: Veículos Autônomos, Planejamento de Movimentos, Reparação de Trajetórias.

ABSTRACT

JUSTO, V. H. S. **A trajectory deformation algorithm for intelligent vehicles**. 2023. 93 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2023.

Autonomous vehicles require robust planning algorithms to compute the sequence of movements from a starting point to an ending goal while considering the constraints in the environment. It is challenging to ensure safety maneuvers in all possible traffic scenarios and the motion planning module recalculates initially-planned trajectories as many times as necessary to resolve those complex situations. However, the computational cost rises up when the planning process is repeated many times for the same task and current solutions do not allow to link user preferences to the vehicle's motion behavior. An alternative is to generate new trajectories based on planned trajectories already available. We propose an algorithm that takes into account Signal Temporal Logic (STL) formulas that represent the constraints imposed by the user in order to modify invalid trajectories and guide the motion planning into respecting safety requirements such as the minimum distance to static obstacles or between vehicles. We use a lattice-based planner to generate candidate paths and include a multi-resolution feature to generate as many lattices as it is necessary depending on the context. Then, the STL robustness value quantifies the level of respect that initial paths have for STL specifications and activates the repairing process that generates new lattices based on the initial selected path. The robustness measure also defines a new resolution to generate lattices and influences the cost function to ensure the selection of the path that has more respect for the STL formulas. The deformed version of the initial lattice is used to generate the trajectory for a specified planning horizon using a simulation approach. The computational cost of the proposed repairing strategy is less than recalculating the complete trajectory from scratch and it is specially convenient when there are not many rule violations near the goal region.

We evaluate our approach using the automobile tools of the robot simulator Webots considering different traffic scenarios involving obstacle avoidance. The efficiency of our method is demonstrated by comparing trajectories using STL constraints with trajectories that do not consider STL rules.

Keywords: Autonomous Vehicles, Motion Planning, Trajectory Repairing.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
AMoD	Autonomous Mobility on Demand
ARA*	Anytime Repairing A*
CAD	Computer-Aided Design
CL-RRT	Closed-Loop Rapidly-exploring Random Trees
CLP	Conformal Lattice Planner
DRRT	Dynamic Rapidly-exploring Random Trees
GPS	Global Positioning System
IDM	Intelligent Driver Model
IMU	Inertial Measurement Unit
LED	Light-Emitting Diode
LTL	Linear Temporal Logic
MTL	Metric Temporal Logic
ODE	Open Dynamics Engine
PRM	Probabilistic Road Map
ROS	Robot Operating System
RRT	Rapidly-exploring Random Tree
RSS	Responsibility-Sensitive Safety
STL	Signal Temporal Logic
STL-LP	STL-based Lattice Planner
TBA*	Time-Bounded A*
TL	Temporal Logic
UGV	Unmanned Ground Vehicle
VRML	Virtual Reality Modeling Language

CONTENTS

CHAPTER

1

INTRODUCTION

1.1 Motivation

Driverless technology has been the focus of extensive research efforts by industry and academy because it has the potential to change the paradigm of mobility as a service. Different platforms promise to take on-road action the next years such as delivery robots, single passenger self-driving cars, and heavy-duty autonomous trucks. However, further research is necessary to improve the navigation stack (perception, prediction, planning, and control) to get safety guarantees for fully autonomous driving.

The motion planning process is usually conceived as a hierarchical structure that provides solutions to sub-problems such as global planning, behavioral planning, and local planning. Typical approaches deal with the motion planning problem decomposing it into two sub-problems: behavior planning, and trajectory planning (PADEN *et al.*, 2016). However, decomposition approaches present some drawbacks as mentioned by McNaughton *et al.* (2011), the behavior planning depends on a flawed model of the underlying trajectory planning that leads to unstable or infeasible trajectory optimization. Moreover, Sadat *et al.* (2019) points out that behavior and trajectory planners independently optimize different objective functions, so modifications in any of the functions requires to repeat the tuning or designing process of the objective function for the trajectory planner.

A lattice planner is a sampling-based method that solves the aforementioned issues by coupling the behavior and trajectory planning sub-problems using a discrete temporal space representation to find the optimal trajectory through graph search (SUN *et al.*, 2020a). It generates multiple dynamically-feasible motions for a variety of traffic situations. The performance of each trajectory and the user preferences can be used to select the best option among all the trajectories available (TABOADA, 2020). However, this planning method focuses on the reachability problem of computing a trajectory to the goal region. In general, sampling-based algorithms (e.g.

Probabilistic Road Map (PRM), Rapidly-exploring Random Tree (RRT), Lattice-based) were not designed to account for high-level specifications provided by formal models such as temporal logic (PLAKU; KARAMAN, 2016).

Temporal Logic (TL) is a formalism to specify the relative order of events that represent the properties of time-dependent systems (BAIER; KATOEN, 2008). Temporal Logic is used in motion planning to describe temporal and/or spacial constraints in the trajectories of robots. For instance, it can specify the importance, frequency, and time of a task execution using multiple operators (TABOADA, 2020). Linear Temporal Logic (LTL) is a variant of temporal logic used for program specification and verification in software engineering (SINGH, 2020). Signal Temporal Logic (STL) is an extension of LTL that includes real-time and real-valued properties in its formulation, and it is used for many continuous and hybrid systems such as autonomous vehicles.

STL translates user preferences into formulas that represent time and space restrictions related to actions of the robot in the environment. Moreover, a robustness value quantifies either the satisfaction or violation of STL formulas. The notion of robustness associated to STL constrains is useful to determine situations that accept certain level of rule violation (TABOADA, 2020). For instance, STL constraints can define the minimum distance that a robot should keep from obstacles (BARBOSA *et al.*, 2019), so a negative robustness value signals a violation of the STL constraint, while a positive value means that the STL formula is satisfied.

Autonomous driving applications require robust algorithms to plan motion in tight, confined environments while eliminating the possibility of collisions with either static or dynamic obstacles (MANGETTE; TOKEKAR, 2020). Urban traffic structures and rules constrain vehicles to a narrow set of paths that may not completely satisfy user preferences. (TABOADA, 2020). Moreover, it is difficult to ensure safety because the planned trajectories do not consider all traffic rules or they are physically infeasible. Replanning the complete trajectory is not the best alternative due to the high computational cost involved in the process (LIN; MAIERHOFER; ALTHOFF, 2021). Therefore, efficient methods to resolve those complex situations are required.

This research work takes into account STL constraints to ensure safety in situations that involve interactions between the autonomous vehicle and its environment such avoiding obstacles or other vehicles. In particular, the project aims to develop a method to deform or repair an initially-planned trajectory so that the modified trajectory is conditioned by STL formulas that represent the safety constraints imposed by the user, and the associated computational cost is less than recalculating the complete trajectory from scratch, which is more convenient when there are not many rule violations near the goal region.

1.2 Scope

Motion planning algorithms are essential for different applications. For instance, autonomous cars can use centralized motion planners to increase the traffic flow in urban environments (MANGETTE; TOKEKAR, 2020). However, many approaches do not offer safety guarantees because the trajectories do not stick to traffic rules. Moreover, a safe action may not be available in the remaining time prior to a collision, which is so common in highly dynamic environments (LIN; MAIERHOFER; ALTHOFF, 2021).

This research project considers an autonomous vehicle operating in a road network governed by traffic rules. The main focus is on the trajectory deformation or repairing problem that is about generating a new trajectory based on planned trajectories already available, and considering STL formulas to include safety requirements such as the minimum distance to static obstacles or between vehicles. The evaluation of the proposed method is made by comparison between the performance of trajectories with and without STL specifications.

Specific conditions are set for the evaluation of the proposed solution. This include a constant nominal velocity (15 m/seg) for the autonomous vehicle and a fixed distance for obstacle detection (30 m). Additionally, extreme curvature on the road is avoided because this condition will be treated in future extensions of this research work.

Further assumptions include: ideal sensing capabilities with fixed tolerances to localization tasks, so the vehicle and obstacle boundary obtained are accurate, we also we adopted an "ideal/perfect" sensor simulation and predefined fixed limits of tolerance - since sensing is not the main subject in this work; ideal path follower, so the controlling criteria is not part of the scope of the project; and ideal communication capabilities, so there is not any delay in the transmission of messages.

1.3 Problem Formulation

The motion planning process for autonomous vehicles involves the minimization of standardized objectives functions. We define the trajectory deformation problem for autonomous vehicles based on (ZIEGLER *et al.*, 2014; ALTHOFF; KOSCHI; MANZINGER, 2017; LIN; MAIERHOFER; ALTHOFF, 2021). Consider $X \subset \mathbb{R}^n$ as the state space of the possible set of states x, $X_{obs} \subset X$ as the set of collision states, $X_{free} = X \backslash X_{obs}$ as the resulting set of permissible states in the free space, and $U \subset \mathbb{R}^m$ as the set of admissible control inputs u. The motion of the vehicle can be modeled by a dynamical system as follows.

$$\dot{x} = f_{(x,u)} \tag{1.1}$$

The initial state is denoted as x_0, and the set of desired states in the goal region of the planning problem is denoted as $X_{goal} \subset X_{free}$. A candidate solution requires the ego vehicle to

reach a goal region X_{free} without causing collisions with obstacles. The occupancy of the vehicle $O_{(x)}$ must be in the free space $X_{free(t)} \subset \mathbb{R}^2$ for all $t \in [t_0, t_f]$. In addition, the vehicle has to obey constraints $g_{(x,u,t)} \leq 0$ such as the speed limit and other traffic rules.

As mentioned by Althoff, Koschi and Manzinger (2017), there are objectives codifying conditions which are included in the cost function J_C with input u, terminal cost Ψ_C, and running cost L_C.

$$J_C(x,u,t_0,t_f) = \Psi_C + \int_{t_0}^{t_f} L_C dt \tag{1.2}$$

Standard cost functions are categorized into Running Costs (L_C) based on acceleration, jerk, steering angle, steering , yaw rate, lane center offset, velocity offset, orientation offset, distance to obstacles, path length, inverse duration, and Terminal Costs (Ψ_C) based on time (ALTHOFF; KOSCHI; MANZINGER, 2017).

To sum up, in the trajectory deformation problem we are interested in solving:

$$\begin{aligned} &\min_{u(\cdot)} J_C(x,u,t_0,t_f) \\ &\textit{subject to} \\ &\dot{x} = f_{(x,u)}, && O_{(x)} \in X_{free}, \\ &g_{(x,u,t)} <= 0, && x_{(t_0)} = x_0, && x_{(t_f)} = x_f \end{aligned} \tag{1.3}$$

1.4 Research Questions

The expected outcome is to visualize the influence of Signal Temporal Logic constraints in the trajectory repairing process which aims to modify invalid trajectories to avoid replanning them from scratch. Therefore, this project addresses the following research questions:

- How do STL constraints deform invalid trajectories to ensure safety specifications?
- How is the performance and impact, related to user comfort and computational cost/processing time, of deformed trajectories with STL constraints compared to trajectories planned without STL?

The main goal of this project is to develop an algorithm to deform invalid trajectories considering safety requirements imposed by the user. The hypothesis is that STL constraints can be used in the trajectory deformation process to generate safer trajectories using a robustness value that quantifies the level in which a specified preference is broken. Including additional safety specifications in the trajectory repairing process should prevent undesirable collisions with obstacles and between vehicles. In the worst case, the algorithm should generate the least violating deformed trajectories to ensure a minimum level of safety in the interactions between

the vehicle and its environment, so the planner manages and resolves potential collision situations at the local level within the shared space enclosing the conflict.

The main contributions of this work are summarized as follows:

1. We combine STL specifications with a lattice-based motion planner to deform initial trajectories using spatially constrained candidate paths that respect the preferences imposed by the user. Moreover, we use the STL robustness metric to develop a multi-resolution lattice approach that repairs the candidate paths only when it is necessary.

2. We proposed a weighted function to encode user-defined preferences in its parameters. This function is useful to classify the lattices into different categories according to the risk level associated to each candidate path.

3. We tested, validated and analysed the results of the proposed algorithm using the automobile tools of the open source robot simulator Webots (MICHEL, 2004). We compare the performance of deformed trajectories with STL constraints with trajectories that do not consider STL rules

1.5 Outline

Chapter 2 presents background information including descriptions of motion planning algorithms, temporal logic concepts and definitions. Chapter 3 presents a discussion of related research works available up to date. Chapter 4 describes the methodology used to develop a motion planning algorithm that includes STL specifications to generate trajectories with a safety component to handle complex situations on the autonomous vehicle's environment. Chapter 5 present a description of the simulation experiments and a discussion of the main results, and Chapter 6 presents the conclusions of the research work.

CHAPTER

2

TECHNICAL BACKGROUND

This chapter presents the main concepts, techniques, and software tools that are used to develop the project. First, a description of the characteristics and modern methods to solve the motion planning problem in traffic scenarios is discussed. Second, the elements and techniques involved in the development of a lattice-based planning algorithm are described. Then, this chapter presents a contextualization of Signal Temporal Logic including formal definitions and theorems. Finally, the main software tool to set up autonomous driving experiments in simulation is presented.

2.1 Motion Planning

Autonomous robots require algorithms to convert high-level specification tasks from humans into low-level descriptions of how to move. The term motion planning is often used to describe this kind of problem (LAVALLE, 2006). This problem refers to how to move a robot from a "start" location to a "goal" location avoiding obstacles. It is sometimes referred to as the "move from A to B" or the "piano movers problem" (how do you move a complex object like a piano in an environment with lots of obstacles, like a house) (BULLO; SMITH, 2020)

Motion planning concerns about finding a path and time-related values (velocity, acceleration, linear force, torque) to precisely define the movements of a robot in an environment. In particular, autonomous vehicles must overcome challenging problems related to motion planning such as generating dynamically feasible trajectories that account for the behavior of either surrounding vehicles or pedestrians to avoid collisions, and the robust executions of planned motions using control techniques (SUN *et al.*, 2020a),(PADEN *et al.*, 2016).

The formulation of the motion planning problem determines the selection of the most appropriate method. Firstly, the problem formulation is strongly related to the inherited data for the scene representation, this data could be either discrete or continuous, algebraic or analytic,

static or dynamic. Thus, the selection of the sensors is essential to implement an efficient motion planning algorithm. Secondly, motion planning combines unavoidable aspects, such as state estimation, evolution over time, action planning, criteria optimization, and compliance with constraints. The outlook of the problem depends on how these aspects are addressed (CLAUSSMANN *et al.*, 2019).

Claussmann *et al.* (2019) describe the State-of-the-Art motion planning techniques for autonomous vehicles as follows:

1. **Space Configuration:** It is based on the decomposition of the evolution space to handle motion generation or deformation. It considers geometric aspects, and refers to either a predictive method with a coarse decomposition or a reactive approach with a finer distribution. The main challenge is to find the appropriate parameters of the space configuration to get a good representation of the motion and the environment. For instance, with a coarser discretization the kinematic constraints may be ignored, while with a finer discretization the real-time performance may be poor. The main techniques in this group are: Sampling Points, Connected Cells, and Lattices.

2. **Pathfinding Algorithms:** It solves combinatorial problems using a graph representation, which can be either weighted or oriented with sampling points, cells, or maneuvers nodes. The main goal is to find a path that optimizes a cost function, which in the context of highway autonomous driving is typically associated to traveled distance, fuel consumption, and comfort. The graph resolution depends on logic and heuristics, which refers to the decision function even when they just involve a selection. The main techniques of this family are: Dijkstra, A*, RRT, and RRT*.

3. **Attractive and Repulsive Forces:** It is a biomimetic-inspired method that solves the problem in a continuous space representation. Attractive forces represent desired motions (e.g. legal speed), and repulsive forces represent obstacles (e.g. road borders, lane markings, obstacles). It provides a reactive behavior to deal with the dynamic environment, and it does not require explicit space decomposition because the motion of the ego vehicle depends on resultant vector forces. Parabolic and canonical functions are used as potential functions. The resolution of the resultant vector is achieved by either the gradient descent method (vehicle model is not considered) or Newton's second law (kinematic constraints are considered). The main techniques in this group are: Artificial Potential Field, Velocity Vector Field, and Elastic Band.

4. **Parametric and Semi-Parametric Curves:** It is a suitable option for path planning on highways for at least two reasons. First, the highway roads are compound by simple and predefined curves such as lines, circles, and clothoids. Second, a predefined set of curves is easy to implement. They may take into account the kinematic constraints of the vehicle, and they are used as a complement to other methods. Geometric considerations are

usually separated from the dynamic constraints analysis, thus the vehicle requires a long time reaction in case of a blocking situation. Curve planners are suitable for predictive approaches and for replanning stages. The main techniques are based on either parametric curves such as Line and Circles, Clothoids, and Sigmoids, or semi-parametric curves such as Polynomials, Splines, Béziers.

5. **Numerical Optimization:** The optimization problem for motion planning is defined as a solve-algorithm based on logic and heuristic approaches. Optimization involves the minimization of a cost function in a sequence of states variables under a set of constraints. In motion planning applications, there are two domain of interest: the first one focuses on finding efficient algorithms to solve complex problems reducing the computation time; the second domain focuses on the mathematical perspective of the problem to deduce properties to derive predictive solutions in a restrictive space. The main methods are: Linear Programming, Non Linear Programming, Quadratic Programming, Model Predictive Control, and Dynamic Programming.

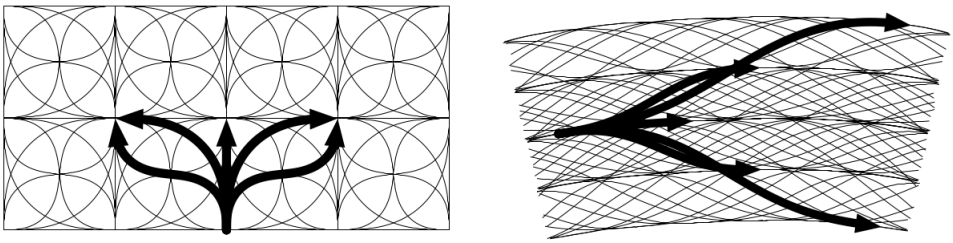
6. **Artificial Intelligence (AI):** These techniques reproduce and simulate the reasoning and learning of human drivers, so that the autonomous vehicle is able to think and act consistently with the environment, to present a memory structure, and to draw inferences. AI algorithms are useful for the decision making process, as they are flexible, adaptive, and reactive to their environment. In addition, AI-based approaches can deal with huge, incomplete, or inaccurate data, and they answer generic questions and absorb new modifications without changing the algorithm structure. AI gathers a wide diversity of methods from logic to cognitive representation. Four techniques can be distinguished: AI Logic (if-then-rules, Finite State Machine, Dynamic Bayesian), AI Heuristic (Agent, Support Vector Machines, Evolutionary), AI Approximate (Fuzzy Logic, Artificial Neural Networks, Belief Network), and AI Cognitive (Risk Estimators, Game Theory).

2.2 Lattice Planner

González *et al.* (2015) describe the sampling-based method as one alternative to solve the motion planning problem for automated vehicles. The main idea is to avoid the need of a explicit representation of the configuration space by conducting a search over it using a sampling scheme.

A graph searching algorithm uses lattices to represent the planning area. Lattice-based planning is one of the most successful methods used in motion planning for robots with differential constraints. It is based on the discretization of the state space into a lattice (see Fig. 1). Lattice-based motion planners have been extensively used for motion planning in off-road scenarios with some applications in on-road scenarios (OLIVEIRA, 2019).

Figure 1 – Left: regular state lattice in an unstructured environment. The five paths are rigidly transformed to create a graph of feasible actions. Right: state lattice adapted to a structured environment.



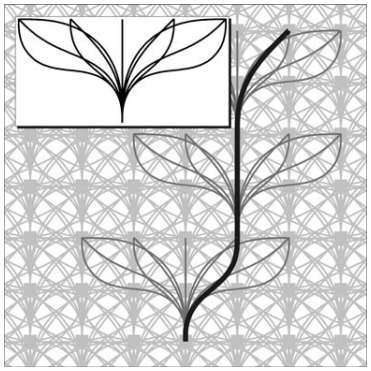
Source: McNaughton *et al.* (2011).

Search Space

The development of the search space is challenging due to the kinematic and dynamic constraints of the autonomous vehicle. Candidate paths are found using a graph search technique on a uniformed grid which is a discrete representation of the Cartesian space. These paths are suitable for robot platforms that are able to drive straight and turn on the spot. However, in complex systems such as car-like vehicles, the kinodynamic constraints difficult the execution of the candidate solutions making these approaches inappropriate for the motion planning problem.

Lattice planners consider the motion constraints of the vehicle in the discretization process, it samples the state space forming a self-repeating tile named lattice (see Fig. 2).

Figure 2 – The state lattice is compound by a repeated and regular pattern of vertices and edges.



Source: Pivtoraiko, Knepper and Kelly (2009).

One approach is to define first a basic set of motion primitives with three possible motions: turn left, right and drive straight, all in the forward direction, then proceed to apply them to generate the discrete representation of the vehicle states. Any candidate solution will result from the combination of these basic motions. Other approach is usually to define first the discretization of the configuration space, then using steering methods or using numerical optimization it is possible to compute motion primitives connecting the discretized vehicle states.

The lattice can be represented as a graph $G = (V, E)$ in which vertices V correspond to vehicle states, and edges E between these vertices represent the motion primitives. In order to

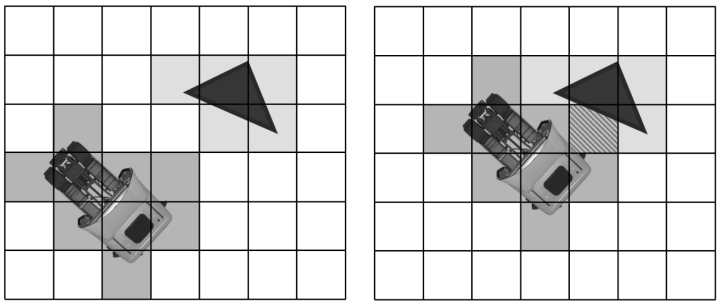
plan a path it is necessary to compute the shortest segment between the current and goal vehicle states by finding the sequence of edges with the lowest cumulative cost connecting them. The standard cost corresponds to the motion primitive length which results in planned motions with minimum length. It is possible to include curvature cost to get smooth paths. Moreover, including a direction of motion cost could result in the vehicle preferring to drive forwards instead of backwards (OLIVEIRA, 2019).

Collision Checking

The planner have to perform collision checking procedures to evaluate if a given vehicle state (vertex) or a motion primitive (edge) are in collision, if that is the case, those paths should be removed from the search graph to guarantee collision-free paths.

Collision checking can be done in different ways. The environment can be represented as an occupancy grid with each cell classified either as obstacle or as free (see Fig. 3). An overlap between the vehicle occupancy and the obstacle occupancy indicates a collision.

Figure 3 – Collision checking using occupancy grid



Source: Oliveira (2019).

Lattice planners usually compute the path swaths offline, then store and use them online to reduce the computational cost of the collision checking procedure. A path swath is the occupancy grid associated to a motion primitive which corresponds to a sequence of states. There is a trade-off between the fidelity of the discrete representation and the time required for computing (OLIVEIRA, 2019).

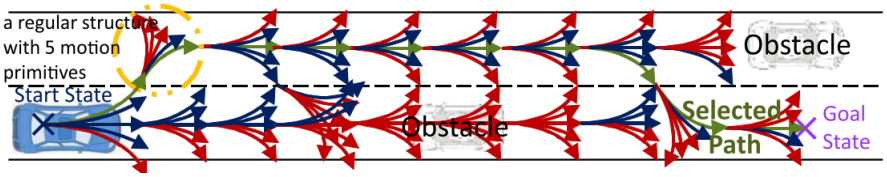
Graph Search Techniques

A solution path is found using a graph search technique. A* search (pronounced "A-star search") is one of the most widely used best-first search algorithm. It drives the search using the evaluation function $f_{(n)} = g_{(n)} + h_{(n)}$ which is the estimated cost of the cheapest solution through n that combines $g_{(n)}$, the cost to reach the node, and $h_{(n)}$, the estimated cost to get from the node to the goal. A reasonable strategy is to look for the cheapest solution in the node with the lowest value of $g_{(n)} + h_{(n)}$ provided that the heuristic function $h_{(n)}$ satisfies certain conditions (NORVIG; RUSSELL, 2016).

Different modifications and extensions to A* have been explored to search on graphs. Anytime Repairing A* (ARA*) was developed for planning under limited computational time, it uses an "ε" parameter to inflate the value of the heuristic function, thus reducing both the node expansion and the computational time (LIKHACHEV; GORDON; THRUN, 2003). Time-Bounded A* (TBA*) is useful for real-time applications, it provides a temporary path solution that is improved on each planning cycle (BJÖRNSSON; BULITKO; STURTEVANT, 2009). D* Lite deals with dynamic environments in which a robot gets new information while traversing a path, it avoids replanning from scratch by efficiently recomputing shortest paths (KOENIG; LIKHACHEV, 2002). Other popular algorithm to search on nonconvex, high-dimensional spaces is the Rapidly-exploring Random Tree (RRT) (LAVALLE *et al.*, 1998), but it is more useful in non-structured environments.

To sum up this section, in motion planning, a lattice is a generalization of a grid, thus it is a regular spatial structure. It requires motion primitives that connect all the states of the lattice, and the states that are feasible compound a graph of feasible maneuvers (see Fig. 4). Lattice methods are mainly used in predictive planning. The motion described in a lattice representation implicitly considers the kinematic constraints and spaciotemporal issues. In addition, the lattice can be computed offline to enable a quick replanning (SUN *et al.*, 2020a). However, further research is necessary to integrate and to project lattice-based solutions into some critical and forthcoming issues of motion planning in traffic scenarios, such as cooperation and coordination of Connected Automated Vehicles, or planning using safety constraints.

Figure 4 – Lattice-based solution for motion planning in highways.



Source: Claussmann *et al.* (2019).

2.3 Signal Temporal Logic

In the context of intelligent vehicles, constraints are used to ensure safety, road compliance, and the satisfaction of traffic rules. However, autonomous vehicles sometimes need to violate certain constraints for driving progress, such as crossing lane markings when another vehicle blocks the current lane. Constraints can be modeled by temporal logic which is a user-defined set of formulas with conditions to apply constraints to the vehicle's motion behaviors with spatial and/or time restrictions (TABOADA, 2020; KARLSSON *et al.*, 2021).

STL is particularly well-suited to model constraints for hybrid systems such as autonomous vehicles that involve continuous dynamics like vehicle states and trajectories as well

as discrete dynamics, such as phases of a traffic light. STL's inherent quantitative semantics allow for quality evaluation of a proposed solution, e.g., how much and how long a trajectory violates the specification (ARECHIGA, 2019; KARLSSON *et al.*, 2021).

The STL definitions are described in (MALER; NICKOVIC, 2004; FAINEKOS; PAPPAS, 2009; DONZÉ; MALER, 2010; TABOADA, 2020; LEUNG; ARÉCHIGA; PAVONE, 2020) as follows. Let us consider an STL atomic predicate,

$$\mu = \begin{cases} \top & \iff f_{(\mathbf{x})} \geq c \\ \bot & \iff f_{(\mathbf{x})} < c, \end{cases}$$

where $\mathbf{x}$ is a time traced signal, $f_{(\mathbf{x})}$ is a real-valued evaluation function that provides an abstraction of the continuous signal $\mathbf{x}$ such that $f : \mathbb{R}^n \to \mathbb{R}$, and $c \in \mathbb{R}$ (ARECHIGA, 2019).

Definition 1 (STL Grammar)**.** The basic grammar of STL is defined as follows.

$$I := (a,b) \mid (a,b] \mid [a,b) \mid [a,b]$$
$$\phi := True \mid \mu \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \Diamond_I \phi \mid \Box_I \phi \mid \phi_1 U_I \phi_2,$$

where μ is a predicate, $\neg$ is the Boolean negation, $\wedge$ the conjunction operator, and $\vee$ the disjunction operator. In addition, $\Diamond$ (Eventually), $\Box$ (Always) and U (Until) are temporal operators that have an associated time interval I where $0 \leq a < b$. In order to simplify notation, I is dropped from the grammar when $a = 0$, $b = \infty$.

STL formulas are evaluated over time series data, packaged in a data structure $(\mathbf{x},t)$ named *timed trace* by Arechiga (2019). A timed trace s is compound by a sequence of states and their associated time, $s = (\mathbf{x}_0,t_0),\ldots,(\mathbf{x}_n,t_n)$ where $t_{i-1} < t_i$ and $\mathbf{x}_i \in \mathbb{R}^n$. A value of the trace at a given time is represented by $s_{(t_i)} = \mathbf{x}_i$, and the notation $s_i = (s,t_i)$ is used to refer to the *tail* of trace s that contains all of the same data s from time t_i onwards.

Definition 2 (STL Boolean Semantics)**.** The satisfaction relation $(\mathbf{x},t) \models \phi$ defines the satisfaction of a formula ϕ by a signal $\mathbf{x}$ at time t. Other useful satisfaction relations are defined recursively as follows.

$$\begin{aligned}
(\mathbf{x},t) &\models f_{(\mathbf{x})} \geq c \iff f_{(\mathbf{x})} \geq c \\
(\mathbf{x},t) &\models \neg\phi \iff \neg((\mathbf{x},t) \models \phi) \\
(\mathbf{x},t) &\models \phi_1 \wedge \phi_2 \iff ((\mathbf{x},t) \models \phi_1) \wedge ((\mathbf{x},t) \models \phi_2) \\
(\mathbf{x},t) &\models \phi_1 \vee \phi_2 \iff ((\mathbf{x},t) \models \phi_1) \vee ((\mathbf{x},t) \models \phi_2) \\
(\mathbf{x},t) &\models \Diamond_I \phi \iff \exists t' \in I \oplus t \; s.t. \; (\mathbf{x},t') \models \phi \\
(\mathbf{x},t) &\models \Box_I \phi \iff \forall t' \in I \oplus t \; s.t. \; (\mathbf{x},t') \models \phi \\
(\mathbf{x},t) &\models \phi_1 U_I \phi_2 \iff \exists t' \in I \oplus t \; s.t. \; ((\mathbf{x},t') \models \phi_2) \wedge ((\mathbf{x},t') \models \Box_{[0,t']} \phi_1)
\end{aligned}$$

Considering a timed trace (s,t) starting at time t, satisfying $\Diamond\phi$ implies that at some time along the sequence, ϕ is true at least one time in the trajectory, it is known as the *Eventually*

operator. $\Box\phi$ means that the condition ϕ has to be satisfied every time during the trajectory, it is known as the *Always* operator. The *Until* operator, $\phi_1 U \phi_2$, represents the fact that the condition ϕ_1 must wait before another event ϕ_2 is fulfilled in the trajectory. It is also possible to combine operators. For instance, the *Always* and *Eventually* operators are used to create new operator modalities such as *Always Eventually* or *Eventually Always*.

Definition 3 (STL Quantitative Semantics)**.** The value of the robustness degree is calculated recursively as follows.

$$\begin{aligned}
\rho_{(\mathbf{x},t,True)} &= \rho_{max} > 0 \\
\rho_{(\mathbf{x},t,f_{(\mathbf{x})} \geq c)} &= f_{(\mathbf{x})} - c \\
\rho_{(\mathbf{x},t,\neg\phi)} &= \neg\rho_{(\mathbf{x},t,\phi)} \\
\rho_{(\mathbf{x},t,\phi_1 \wedge \phi_2)} &= \min(\rho_{(\mathbf{x},t,\phi_1)}, \rho_{(\mathbf{x},t,\phi_2)}) \\
\rho_{(\mathbf{x},t,\phi_1 \vee \phi_2)} &= \max(\rho_{(\mathbf{x},t,\phi_1)}, \rho_{(\mathbf{x},t,\phi_2)}) \\
\rho_{(\mathbf{x},t,\Diamond_I \phi)} &= \max_{t' \in I \oplus t} (\rho_{(\mathbf{x},t',\phi)}) \\
\rho_{(\mathbf{x},t,\Box_I \phi)} &= \min_{t' \in I \oplus t} (\rho_{(\mathbf{x},t',\phi)}) \\
\rho_{(\mathbf{x},t,\phi_1 U_I \phi_2)} &= \max_{t' \in I \oplus t} (\min(\rho_{(\mathbf{x},t',\phi_2)}, \min_{t'' \in [t,t']} (\rho_{(\mathbf{x},t'',\phi_1)})))
\end{aligned}$$

The robustness value for strict and non strict inequalities is the same (ARECHIGA, 2019). Moreover, the robustness trace is defined to describe the robustness value of each timed trace sub-sequence.

Definition 4 (STL Robustness Trace)**.** Let us consider a timed trace $(\mathbf{x},t)$ staring at time t_0, and a STL formula ϕ. The robustness trace is defined as follows.

$$\begin{aligned}
\rho_{(\mathbf{x},t_0,\phi)} &= \rho_0, \rho_1, \ldots, \rho_n \\
&= \rho_{(\mathbf{x},t_0,\phi)}, \rho_{(\mathbf{x},t_1,\phi)}, \ldots, \rho_{(\mathbf{x},t_n,\phi)} \\
&= \rho_{(\mathbf{x}_0,\phi)}, \rho_{(\mathbf{x}_1,\phi)}, \ldots, \rho_{(\mathbf{x}_n,\phi)}
\end{aligned}$$

Theorem 1. Considering any $\mathbf{x} \in \mathbf{X}$ and the STL formula ϕ, if $\rho_{(\mathbf{x},t,\phi)} < 0$ then $\mathbf{x}$ does not respect ϕ at time t, and if $\rho_{(\mathbf{x},t,\phi)} > 0$ then s satisfies ϕ at time t. This fact is formalized as follows.

$$\begin{aligned}
\rho_{(\mathbf{x},t,\phi)} > 0 &\Longrightarrow (\mathbf{x},t) \models \phi \\
\rho_{(\mathbf{x},t,\phi)} < 0 &\Longrightarrow (\mathbf{x},t) \not\models \phi
\end{aligned} \tag{2.1}$$

Refer to (MALER; NICKOVIC, 2004; DONZÉ; MALER, 2010) for a proof of Theorem 1.

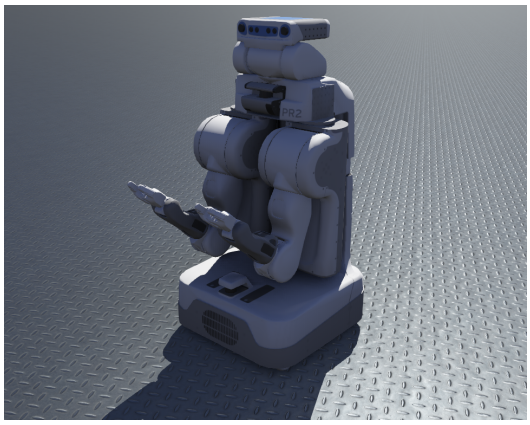
The robustness metric is the link between the STL rules and their impact on the motion planning process. The STL robustness value uses real values to represent how much a signal is violating or respecting the preferences specified by the user.

2.4 Webots Simulator

Webots[1] is an open source and multi-platform desktop application to model, program and simulate robots, and t is widely used in industry, education and research (see Fig. 5). Webots was initially developed by Dr. Olivier Michel at the Swiss Federal Institute of Technology (EPFL) in Lausanne, Switzerland and then by Cyberbotics Ltd. as a proprietary licensed software (MICHEL, 2004).

Figure 5 – Simulated robots in Webots

(a) PR2 - Willow Garage



Source: Webots (2022).

(b) Khepera III - K-Team Corporation

Source: Webots (2022).

Webots includes models of robots, sensors, actuators and objects. In addition, the user can build new models from scratch, import them from 3D Computer-Aided Design (CAD) software and specifies both the graphical (shape, dimensions, position, orientation, colors, texture) and the physical (mass, friction factor, spring and damping constants) properties of the objects. In addition, Webots uses the Open Dynamics Engine (ODE) library to simulate physical properties of objects such as velocity, inertia and friction. It includes the most popular sensor and actuators used in robotics: lidars, radars, proximity sensors, light sensors, touch sensors, Global Positioning System (GPS), accelerometers, cameras, emitters and receivers, servo motors, position and force sensor, Light-Emitting Diode (LED), grippers, gyros, compass, Inertial Measurement Unit (IMU), etc.

Webots has different tools to develop robotics systems from scratch. The simulated worlds are stored in *.wbt files which is a cross-platform format based on the Virtual Reality Modeling Language (VRML). Moreover, it has a simple API to program the robot controllers in different programming languages including C, C++, Python, Robot Operating System (ROS), Java and MATLAB. Webots interface allow users to take screenshots, record, and interact with robots and objects while the simulation is running. Webots can also stream a simulation on web browsers using WebGL.

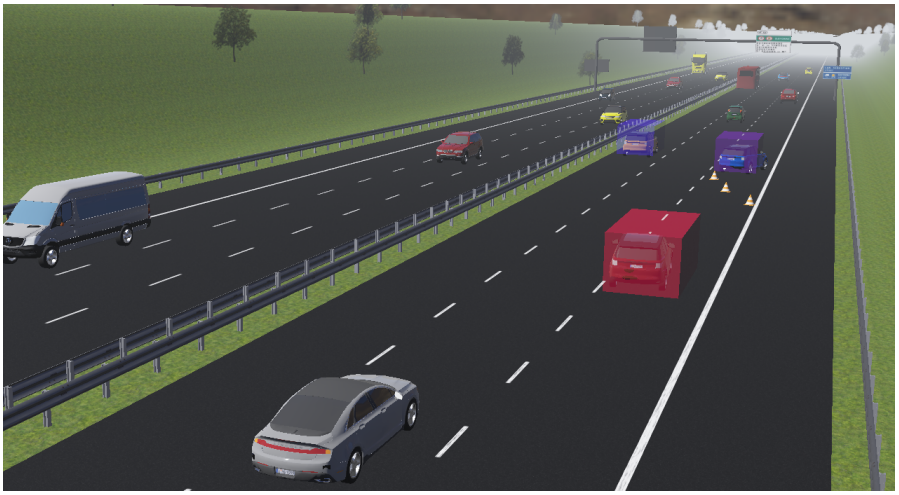
[1] Webots from Cyberbotics: <https://cyberbotics.com/>

Webots is a useful software tool for different fields of study:

- Wheeled and legged robotics.
- Mobile robotics.
- Swarm intelligence and multi-robot systems.
- Autonomous vehicles.
- Artificial life and evolutionary robotics.
- Simulation of adaptive and reactive behaviours
- Modular Robotics.
- Experimental environment for computer vision.

In this project we use the automobile tools of the Webots simulator to create traffic scenarios to study trajectory repairing for obstacle avoidance maneuvers (see Fig. 6).

Figure 6 – Traffic scenario in Webots.



Source: Webots (2022).

2.5 Final Considerations

This chapter presented the main concepts and techniques to solve the trajectory deformation problem for autonomous vehicles. Different motion planning techniques were briefly introduced. In particular, the lattice-based planner was pointed out as an attractive alternative to generate candidate paths. The main characteristics and advantages of the lattice planner over other planning methods were discussed. The concepts and formal definitions of Signal Temporal Logic were also described in this chapter. Finally, the robot simulator Webots was briefly introduced as the main software tool to set up experiments for autonomous driving systems.

CHAPTER

3

RELATED WORKS

This chapter presents a discussion about relevant works of the motion planing literature related to trajectory repairing. It also describes research works that have used Temporal Logic to ensure safety for autonomous driving applications. Finally, it presents the main features of previous lattice-based implementations and compared them to the proposed planning algorithm.

3.1 Trajectory Repairing

Reactive behaviors need to account for changes in the environment to avoid unexpected danger. In this context, replanning is necessary, but is hard to decide when and how to modify the current plan, how to integrate the sensory information into the current world model, or how to use prior plans to improve future plans. An alternative is to consider prior motion trees because any deviation from the initial plan during execution only invalidates certain parts of the motion tree, so that the rest of the trajectory can be used as a reliable guide for future expansions. Thus, the replanning running time will be reduced as large parts of previous computations are reused. There is an increased in computation efficiency by locally repairing the initial plan when dealing with new obstacles (FERGUSON; KALRA; STENTZ, 2006).

The conventional research works on mobile robotics about trajectory deformation do not include neither the constraints associated to the environment nor the nonholonomic restrictions of car-like vehicles (LIN; MAIERHOFER; ALTHOFF, 2021). For instance, Ferguson, Kalra and Stentz (2006) developed the Dynamic Rapidly-exploring Random Trees (DRRT) algorithm to repair trajectories produced by a RRTs removing just the newly-invalid parts of the tree and regrows a solution from what remains. Pham and Nakamura (2015) proposed a trajectory correction algorithm to deform an initially-planned trajectory in order to avoid unexpected obstacles while preserving affine-invariant properties of the original motion such as smoothness, periodicity, and velocity.

There are few studies on trajectory repairing for autonomous vehicles. For example, Rösmann *et al.* (2012) developed the Time Elastic Band approach that deforms an initial path composed of a sequence of way points into a trajectory with temporal information that considers both the geometric and the dynamic constraints to control the robot in real-time. A multi-objective least-squares optimization process is used to repair the initial trajectory, but the deformation may generate a non drivable trajectory. Ziegler *et al.* (2014) introduced a local, continuous method for trajectory planning. In order to include the new sensor information that is available as the vehicles moves on the environment, a re-planning scheme was designed to modify parts of the original trajectory. However, it is not possible to determine the time intervals of the remaining parts.

Lin, Maierhofer and Althoff (2021) presented a sampling-based trajectory repairing method that uses criticality assessment to detect the invalid trajectory section that must be modified. Moreover, the repairing problem is separated using a hierarchical structure, and quintic polynomials are used as a reference path into the informed Closed-Loop Rapidly-exploring Random Trees (CL-RRT) algorithm to efficiently repair the invalid trajectory.

3.2 Planning with Temporal Logic

The intertwined dependencies make it hard to combine motion planning problems with dynamics and temporal logic specifications, so practical approaches have been proposed to overcome the related issues. Sampling-based methods have been developed to expand a motion tree by adding collision-free and dynamically-feasible trajectories as branches. Another works have focused on designing reactive controllers synthesized from LTL specifications and discrete abstractions. A line of research has focused on high-dimensional robotic systems with non-linear dynamics; these works treat the motion planning problem with LTL specifications using a probabilistic search over a hybrid space compound by discrete and continuous elements. Moreover, some methods place restrictions on the LTL specifications to support only syntactically co-safe LTL formulas. Another works study how to include specifications of time using Metric Temporal Logic (MTL) that extends LTL with time intervals (PLAKU; KARAMAN, 2016).

In the context of autonomous vehicles, temporal logic has been used to guarantee safety in their motion and behavior. Maierhofer *et al.* (2020) used Metric Temporal Logic to mathematically express a comprehensive set of traffic rules that are useful to evaluate trajectories generated by a motion planner. The proposed MTL formulas were evaluated on recorded real-world data and proved to be efficient for automated traffic rule monitoring. Arechiga (2019) used STL specifications to express a set of contracts on the input-output behavior of an autonomy stack for self-driving cars. The proposed contracts enable safety testing, provide monitoring capabilities for safety performance using the STL robustness semantics, and they can be integrated into industry development processes.

Some works combine STL with formal definitions of safe driving behavior such as the Responsibility-Sensitive Safety (RSS) model. Hekmatnejad *et al.* (2019) encoded the RSS model in STL. The STL robustness was used to monitor the interactions between the ego vehicle and its surrounding environment. It also was used to know why a specification was satisfied or violated, and to infer when a vehicle behavior came close to a violation. The direct mapping between STL formulas and RSS rules implies that the greater the robustness value, the less likely is the ego vehicle to violate the RSS requirements. The RSS rules written in STL were successfully used to monitor multiple real driving data scenarios. Karlsson *et al.* (2021) presented a new method to encode human driving styles in motion planning frameworks using STL and its robustness metrics. The parameters of a penalty structure were calibrated to model different automated driving styles. A set of STL formulas based on the RRS model were combined with the penalty structure to generate a least-violating trajectory.

Recent works successfully include STL constraints in the motion planning process. Barbosa *et al.* (2019) developed an approach to influence the movement of an Unmanned Ground Vehicle (UGV) in an exploration task adding spatial constraints as STL fragments to guide its trajectories, the violation of user-defined spatial preferences was minimized through a cost function that integrates the STL robustness value. Taboada (2020) developed a RRT* sampling-based algorithm to identify the best trajectories for multiple robots analyzing the cost of all possible paths. His approach used the robustness metric of STL to express either the satisfaction level or violation level of user preferences for each path, then that robustness value was transformed into a gain to determine the costs of the corresponding paths. The notion of robustness attached over the trajectories was considered to select the paths with smaller costs that respect the user specifications.

3.3 Lattice-based Planning

Classic navigation algorithms for mobile robots have been adapted and modified to deal with the challenges of road networks and driving rules. González *et al.* (2015) consider the implementation mechanism of planning techniques to classify them in four categories: graph search, sampling, interpolating and numerical optimization.

In particular, a graph search based method that uses lattices has been successfully applied in autonomous vehicles, in this context lattice-based planners are implemented using two approaches. The first method uses state lattices to decompose the environment in a local variable grid. For instance, Pivtoraiko and Kelly (2005) discretize the configuration space to build a state lattice that is used to make motion planning queries as a graph search in which edges represent feasible paths up to a given resolution. This connectivity scheme is suitable for efficient motion planning because no time is wasted either generating, evaluating, or fixing infeasible plans. The state lattice formulation allows resolution complete planning queries because if a vehicle can

travel from one node to another node then the lattice provides a sequence of paths to perform this maneuver.

Other works reported by the motion planning research community also follow the same approach and use a discrete representation of the environment to search feasible paths.

Ferguson, Howard and Likhachev (2008) used a high-fidelity lattice planner in unstructured environments, this planner uses a backward search process from the goal to the initial pose in order to generate a path consisting of a sequence of collision-free maneuvers.

Howard *et al.* (2008) proposes a state space sampling method to deal with high-speed navigation in constrained environments such as trails, roadways, and dense off-road obstacle fields, which limit the space of acceptable motions. Search spaces by sampling in state space remain within the lane or are oriented to do so shortly, and are more expressive when navigating in complex environments.

Likhachev and Ferguson (2009) proposes a multi-resolution lattice planner which uses a high-resolution action space near the robot and the goal, and a low-resolution action space elsewhere. It can considerably reduce the full computational cost, but it is necessary that the high-resolution and low-resolution lattices connect together smoothly. It was used for planning maneuvers in parking lots, in geometric road following in off-road areas, and in error recovery scenarios during on-road driving.

The second method to implement lattice-based planners uses spatio-temporal lattices which considers time and velocity dimensions (GONZÁLEZ *et al.*, 2015). Many papers in the motion planning literature discuss this type of implementation.

Ziegler and Stiller (2009) build a space-time manifold by combining configuration space and time axis, from this manifold and using deterministic sampling they get a geometric acyclic graph named spatio-temporal state lattice. The workspace is reparametrized to align the lattice to the course of the road, reducing the number of configurations space samples required to describe vehicle dynamics and accomplish real time requirements. They propose sampling from the state space in a deterministic fashion, so that each sample becomes a vertex in a graph, and each edge has a geometric representation of a path connecting its adjacent vertices. The shortest path is found within the graph with standard graph searching methods.

Kushleyev and Likhachev (2009) provide real-time performance using a time-bounded lattice for planning with dynamic obstacles. This graph structure has two different types of states: six-dimensional (x, y, θ, v, w, t) states to create the time-parameterized portion of the trajectory, and two-dimensional (x, y) states for 2D search, each transition in this graph is a short-term motion primitive between the corresponding pair of these states. It combines short-term planning in time with long-term planning without time for computing and re-computing paths that are optimal or nearly-optimal.

McNaughton *et al.* (2011) propose a spatiotemporal search graph representation to

evaluate a large number of variations in time and velocity without increasing the size of the search space. Moreover, by conforming the lattice to the environment, it means constraining edges to join specified points in the spatial dimensions while considering any value in the temporal dimensions, it was possible to generate dynamically-feasible motions in structured environments for a variety of traffic situations.

Werling *et al.* (2012) propose a semi-reactive planning method for long-term maneuver tasks with short-term collision avoidance. It considers multiple final states in a discretized manifold to take advantage of the road environment structure, it combined with short replanning cycles leads to a reactive layer, which is highly responsive to changes in traffic scenarios.

Gu *et al.* (2013) propose an on-road planning method that reduces irrelevant sampling by focusing the search in a reachable/desirable subset of the large spatio-temporal space, reducing the computation time. In addition, it is combined with optimization-based techniques to produce human-like reference trajectories that can efficiently account for road geometry, obstacles and higher-level directives in urban and highway driving.

Furgale *et al.* (2013) proposed a lattice-based planner for automated parking maneuvers, it provides collision-free trajectories to reach the intended parking slot or the target charging station. In order to reduce the processing time, the motion primitives, the driving swath, and a heuristic look-up table are pre-calculated.

Learning methods have been applied to improve lattice-based planners. Iaco, Smith and Czarnecki (2019) propose to learn from a representative data set of vehicle paths in order to compute a sparse lattice planner control set that is suited for a particular application. They develop an algorithm to evaluate a given control set according to a scoring measure. The control actions are selected using an objective function that promotes sparsity and rewards improvements in matching the data set. The experiments involving real and synthetic data show that this method generates smaller control sets with high degree of manoeuvrability that reduce the computation time when compared with other state-of-the-art lattice control set computation technique. The learned control sets are effective to capture the driving style of the data set during the planning process.

A recent paper report the use of lattice-based motion planners for autonomous vehicles in highway scenarios. Sun *et al.* (2020a) proposes a novel Feedback Enhance Lattice Planner, it uses an Intelligent Driver Model (IDM) as a speed feedback policy for the ego vehicle. The path and the initial state are used to determine the velocity at the end of the trajectory and the execution time, so it does not need a discrete representation of the acceleration, velocity, an the time dimension, as a consequence the lattice remains in the 2D spatial space. In order to simplify the collision checking process and the identification of relative positions, the vehicles can register onto the directed-graph map representation.

Table 1 presents and compare the main features of lattice-based planners reported in the

motion planning literature. The Technique column referees to how it was implemented, state technique means that non temporal lattices were used, while spacio-temporal technique consider time and speed dimensions. The Environment column refers to nature of the place where the lattice-based planner was adapted, it could be either unstructured or structured. The remaining column refers to the consideration of Signal Temporal Logic in the implementation.

Table 1 – Comparison of lattice-based planners.

Paper	Technique	Environment	STL formulas
Pivtoraiko and Kelly (2005)	state	unstructured	-
Ferguson, Howard and Likhachev (2008)	state	structured	-
Howard *et al.* (2008)	state	structured	-
Likhachev and Ferguson (2009)	state	structured	-
Ziegler and Stiller (2009)	spacio-temporal	structured	-
Kushleyev and Likhachev (2009)	spacio-temporal	unstructured	-
McNaughton *et al.* (2011)	spacio-temporal	structured	-
Werling *et al.* (2012)	spacio-temporal	structured	-
Gu *et al.* (2013)	spacio-temporal	structured	-
Furgale *et al.* (2013)	spacio-temporal	unstructured	-
Iaco, Smith and Czarnecki (2019)	state	structured	-
Sun *et al.* (2020a)	state	structured	-
Proposed Solution	spacio-temporal	structured	✓

Source: Elaborated by the author.

3.4 Final Considerations

This Chapter briefly described different approaches to solve the trajectory deformation problem. Few studies on trajectory repairing for autonomous driving applications are currently available and none of them considered STL constraints in their solutions. Moreover, few papers apply Signal Temporal Logic to ensure safety in the motion planning process. Some papers represent traffic rules with STL formulas but none of them considered a lattice-based approach to generate the final trajectory. The proposed planning method differs from previous works because it repairs invalid trajectories using a lattice planner combined with STL specifications related to ensure safety in scenarios involving obstacle avoidance maneuvers. Moreover, our solution relies on a multi-resolution lattice approach to guarantee the efficiency of the planning algorithm and reduce the time-consuming in the process.

CHAPTER

4

METHODOLOGY

This chapter describes the methodology to solve the trajectory deformation problem for autonomous vehicles. It includes an explanation of both the mathematical model that captures the dynamic of the vehicle, and the circle decomposition representation of the vehicle for collision-checking procedures. In addition, this chapter presents the proposed algorithm to deform trajectories and describes in detail the procedure to generate lattices with spatial constraints to ensure safety maneuvers in the environment. This chapter includes an explanation of the lattice generator, a review of the STL specifications to avoid obstacles in the safest way, and a description of the STL robustness metric that was used in the cost function to influence the path selection. Finally, the procedure to create the trajectory based on the deformed path is presented.

4.1 Vehicle Modeling

The kinematic single-track model is used to represent the autonomous vehicle with only two wheels, where the front and rear wheel pairs are each lumped into one wheel. As mentioned by Lin, Maierhofer and Althoff (2021) it captures the relevant vehicle dynamic, and it does not consider any tire slip. In this model the velocity vector v at the center of the rear axle is always aligned with the link between the front and rear wheel.

The kinematic single-track model has been used in the development of many motion planning algorithms. It is useful in situations in which the non-holonomic constraints (e.g. minimum turning radius) of the autonomous vehicle must be considered, such as parking applications. As explained by Althoff, Koschi and Manzinger (2017), the required variables are the velocity v, the velocity of the steering angle v_δ, the steering angle δ, the heading Ψ, and the parameter l_{wb} describing the wheelbase. The differential equations of the kinematic single-track model are defined as follows,

$$\begin{aligned}
\dot{\delta} &= v_\delta, \\
\dot{\Psi} &= \tfrac{v}{L_{wb}} \tan(\delta), \\
\dot{v} &= a_{long}, \\
\dot{s}_x &= v\cos(\Psi), \\
\dot{s}_y &= v\sin(\Psi).
\end{aligned} \tag{4.1}$$

The kinematic single-track model may differ in publications, depending on whether the steering angle or the steering velocity is considered as an input, or the vehicle velocity or the vehicle acceleration is an input for the system (ALTHOFF; KOSCHI; MANZINGER, 2017).

State Space Model

The construction of the state space representation is based on the information provided in (ALTHOFF; KOSCHI; MANZINGER, 2017; LIN; MAIERHOFER; ALTHOFF, 2021). In order to represent the kinematic single-track model in state-space form, it is necessary to define the state variables as follows,

$$\begin{aligned}
x_1 &= s_x, \\
x_2 &= s_y, \\
x_3 &= \delta, \\
x_4 &= v, \\
x_5 &= \psi.
\end{aligned} \tag{4.2}$$

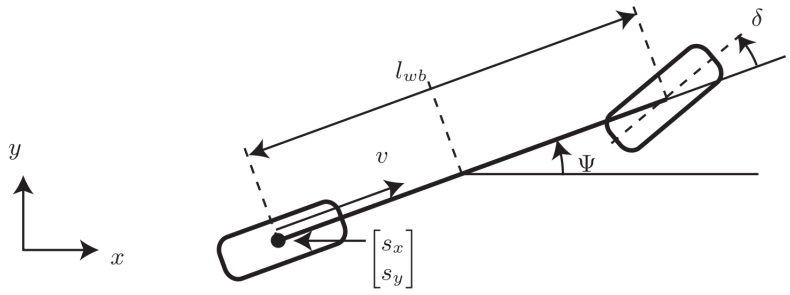
The input variables are defined as follows,

$$\begin{aligned}
u_1 &= v_\delta, \\
u_2 &= a_{long}
\end{aligned} \tag{4.3}$$

The five dimensional state vector $x = [s_x, s_y, \delta, v, \Psi]^T$ is formed by the two-dimensional postion at the center of the rear axle $[s_x, s_y]^T$, the steering angle δ, the longitudinal velocity v, and the orientation Ψ. The control input vector $u = [v_\delta, a_{long}]^T$ contains the steering velocity v_δ and the longitudinal acceleration a_{long} (see Fig. 7) . The kinematics can be written in state-space form as follows.

$$\begin{aligned}
\dot{x}_1 &= x_4\cos(x_5), \\
\dot{x}_2 &= x_4\sin(x_5), \\
\dot{x}_3 &= u_1, \\
\dot{x}_4 &= u_2, \\
\dot{x}_5 &= \tfrac{x_4}{l_{wb}} \tan(x_3)
\end{aligned} \tag{4.4}$$

Figure 7 – Kinematic single-track model with the pure-pursuit strategy for tracking the current target line.



Source: Althoff, Koschi and Manzinger (2017).

Parameters

The parameters of the Kinematic Single-Track model are listed in Table 2 and the constraint parameters are presented in Table 3.

Table 2 – Vehicle Parameters.

Name	Symbol	Value	Unit
vehicle length	l	4.508	[m]
vehicle width	w	1.610	[m]
wheelbase	l_{wb}	2.578	[m]

Source: Althoff, Koschi and Manzinger (2017).

Table 3 – Constraint Parameters.

Name	Symbol	Value	Unit
minimum steering angle	$\underline{\delta}$	-1.066	[rad]
maximum steering angle	$\overline{\delta}$	1.066	[rad]
minimum steering velocity	$\underline{v}_\delta$	-0.4	[rad/s]
maximum steering velocity	$\overline{v}_\delta$	0.4	[rad/s]
minimum velocity	$\underline{v}$	-13.6	[m/s]
maximum velocity	$\overline{v}$	50.8	[m/s]
switching velocity	v_S	7.319	[m/s]
maximum acceleration	a_{max}	11.5	[m/s]

Source: Althoff, Koschi and Manzinger (2017).

4.2 Circle Decomposition

As mentioned by Waslander (2018), collision checking is a challenging and computationally intensive problem present in many domains from gaming to autonomous driving applications. It refers to the procedure of ensuring that a planned trajectory guides an object

through the environment without any collision with obstacles. It requires perfect information about the environment and a limited computation power to calculate precisely for complex shapes in the environment. However, autonomous cars have neither of these available, the information available from other modules in the navigation stack is an imperfect estimate, which means that it is necessary to add buffers to the collision checking algorithms to make them more error tolerant.

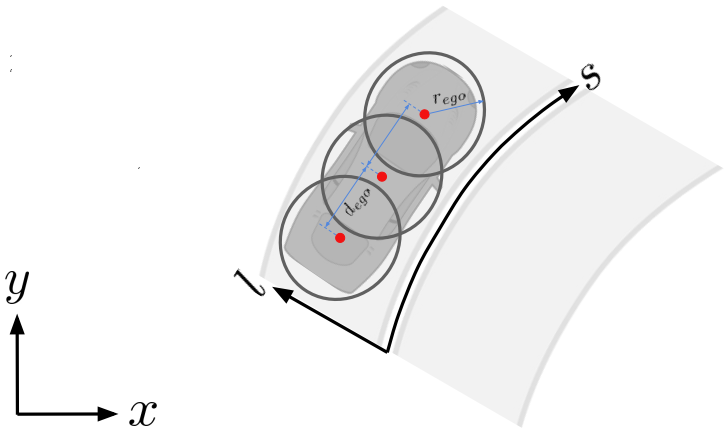
A conservative approximation to collision checking is useful to handle imperfect information and extensive computation requirements. It reduces optimality, which is based on an appropriate objective function, but improves speed and robustness. An approximation to collision checking is useful to gain computing performance, but that approximation must be overly conservative. Moreover, a good approximation should offer algorithmic speed-up without compromising safety, and at the same time it should minimize the degree to which the trajectories become sub-optimal. In this context, a conservative approximation reports a collision along a path even when it does not exist, but will never report no collision along a path if one does actually occur (WASLANDER, 2018).

The circle-based collision checking method is a conservative approximation that uses overlapping circles to encompass the footprint of the vehicle's rectangular body so that the swath generated by the rectangle will be a subset of the swath generated by the overlapping circles. As described in (MENG *et al.*, 2019), in order to perform the collision checking procedure, it is necessary to represent the shape of the ego vehicle with vehicle discs as shown in Fig. 8, in this case the ego vehicle is decomposed into three discs with radius r_{ego}, the distance between the centers of the discs is d_{ego}, and the red points represent centers of the discs. The curves representing the Frenet frames (SUN *et al.*, 2020b) in the lane are also depicted in Fig. 8.

The circle-based approach is conservative because an obstacle inside the circle footprint, but not inside the car footprint, may result in a location being reported as collision even though one would not actually occur. On the other hand, an obstacle that lies outside of the circles will not generate a collision alarm and it will not occur. The collision checker may report some false positive collisions, but will not generate false negative alarms which results in a practical buffer for autonomous vehicles that helps to handle the imperfect information available from other modules of the navigation stack.

The collision checking calculations are made all in discrete form since they are perform on a computer. The accuracy depends on the resolution to perform the discretizations. Consider two swaths calculated for the same path and footprint, but one of the swaths is computed with much coarser resolution than the other. It turns out that a coarser resolution generates large gaps in the swath, which can result in errors if obstacles are located at those positions. The finer the resolution of the collision checker, the more accurate it will be. However, a higher resolution also has a higher computational cost. Therefore, it is important to get the appropriate balance between accuracy and computational cost.

Figure 8 – Discs representing the ego vehicle body.



Source: Elaborated by the author.

4.3 Spatially Constrained Lattices

It is possible to imposed spacial restrictions when generating lattices, it refers to control the space between the lattices. It is useful to avoid unnecessary maneuvers that make the vehicle to move far away to avoid a particular obstacle, instead it allows the vehicle to be safe while performing the evasive maneuver without making huge oscillations.

4.3.1 Lattice Generator

The high level objective of the conformal lattice planner is to plan a feasible collision-free path from the autonomous vehicle current position to a given goal state. The Conformal Lattice Planner (CLP) (MCNAUGHTON *et al.*, 2011) considers the structured nature of roads to speed up the planning process while avoiding obstacles, and to produce plans that closely resemble human driving. It creates smooth path options that swerve slightly to the left or right of the goal path. This planner also takes into account the fact that the autonomous car should never leave the road unless there is an emergency scenario. It generates a sequence of alternate goal states by laterally offsetting them from the central goal state with respect to the heading of the road (WASLANDER, 2018).

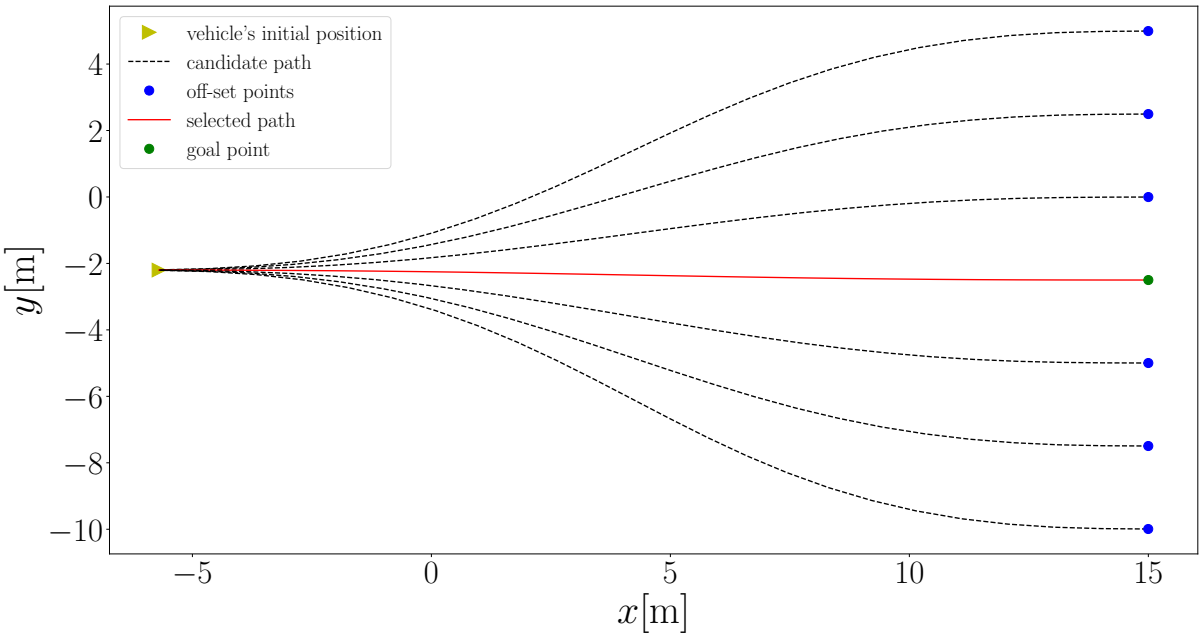
As in (MCNAUGHTON *et al.*, 2011), we consider the method developed by Pivtoraiko, Knepper and Kelly (2009) to adapt the lattices to the structured of the road. The lattice is built around a lane center line represented as a sampled function $[x_{(s)}\ y_{(s)}\ \theta_{(s)}\ \kappa_{(s)}]$ of arc length s. In addition, $p_{(s,l)} = [x_{(s,l)}\ y_{(s,l)}\ \theta_{(s,l)}\ \kappa_{(s,l)}]$ defines the points on the road at lateral off-set l from the center line, which are calculated as follows

$$\begin{aligned} x_{(s,l)} &= x_{(s)} + l\cos(\theta_{(s)}) \\ y_{(s,l)} &= y_{(s)} + l\sin(\theta_{(s)}) \\ \theta_{(s,l)} &= \theta_{(s)} \\ \kappa_{(s,l)} &= (\kappa_{(s)}^{-1} - l)^{-1}, \end{aligned} \tag{4.5}$$

$\kappa_{(s)}$ represents the curvature of the path, such that $\theta_{(s,l)} = \int_0^s \kappa_{(s,l)}$.

Fig. 9 illustrates the idea that the end point of each path is laterally offset from the central path, which corresponds to a goal point on the road. As mentioned by Waslander (2018), it discards paths that may not result in forward progress along the ego lane, so it reduces the search space keeping the planner computationally tractable. If a goal state that is close to the current ego vehicle position is chosen, then the computational time required to find a path to the goal point is reduced as well. However, it also reduces the ability of the planner to avoid obstacles farther down a path, in a smooth and comfortable manner.

Figure 9 – Goal and off-set points.



Source: Elaborated by the author.

Goal Horizon

The goal horizon can be either dynamically calculated or fixed. In this work we consider a fixed goal horizon, and take the goal point as the point along the center line of the lane, that is a distance ahead of the vehicle equal to the established goal horizon. A short look-ahead improves computation time, but reduces the possibility to avoid obstacles (WASLANDER, 2018). Fig. 9 presents the goal and off-set points for the lattices, the green point corresponds to the specified goal location, the blue points correspond to the laterally off-set goal points, which are used as endpoint constraints to calculate each spiral in the lattice. The lane central line has arc length equal to the selected goal horizon. At each planning cycle, the goal point is recomputed based on the same horizon, so that autonomous vehicle makes forward progress along the lane. So, there is a trade-off between the size of the lattice (grid) and the computation time: as far is the goal horizon, greater will be the grid and the computation time, and so, if the the goal horizon is near, the computation the time will be shorter.

Generating Spirals

A path is represented with a cubic polynomial spiral. Therefore, the curvature of the path is a cubic polynomial function of arc length represented as follows.

$$\kappa_{(s)} = a + bs + cs^2 + ds^3 \tag{4.6}$$

The cubic spirals required to reach the off-set goal points are calculated solving an optimization problem. Initially the approach focuses on generating feasible paths that account for the kinematic constraints of the vehicle, it does not verify if the paths are collision free.

The parameters of the cubic function representing the curvature can be calculated to define a path connecting any pair of endpoints (x, y, θ, κ). Considering the work of Howard (2009), we use the parameterization $p = [p_0\ p_1\ p_2\ p_3\ s_f]$, so that the curvature can be expressed as follows,

$$\kappa_{(s)} = a_{(p)} + b_{(p)}s + c_{(p)}s^2 + d_{(p)}s^3, \tag{4.7}$$

where s_f is the arc length of the curve between the boundary constraints. The coefficient functions a, b, c, d in Eq. 4.7 are selected to verify the following equations.

$$\begin{aligned} \kappa_{(0)} &= p_0 \\ \kappa_{\left(\frac{s_f}{3}\right)} &= p_1 \\ \kappa_{\left(\frac{2s_f}{3}\right)} &= p_2 \\ \kappa_{(s_f)} &= p_3. \end{aligned} \tag{4.8}$$

The boundary conditions determine both the starting and ending state, and numerical approximation is required because a closed form solution is not available for the spiral end position. Then, it is possible to generate a spiral that satisfies the boundary conditions by optimizing the spiral parameters and its length s_f.

Trapezoidal Rule Integration

As mentioned by Kelly and Nagy (2003), a closed form solution of the position along the spiral is not available, thus numerical integration is required. An efficient method is necessary to solve the integrals in Eq. 4.9, because they are evaluated in many points of the entire spiral.

$$\begin{aligned} x_{(s)} &= x_0 + \int_0^s \cos(\theta_{(s')})ds' \\ y_{(s)} &= y_0 + \int_0^s \sin(\theta_{(s')})ds' \end{aligned} \tag{4.9}$$

In this project, the Trapezoid rule (Eq. 4.10) is used for numerical integration. It is an interpolation approach that is more efficient than Simpson's rule because it builds each subsequent point along the curve from the previous one. In order to get all of the required points it is enough to do one sweep through the spiral. On the other hand, if Simpson's rule is used to solve the integrals, then

an integral approximation need to be solved for each point, which is much less efficient (KELLY, 2018).

$$\int_0^s f_{(x)} dx \approx \sum_{i=1}^{N-1} \frac{f_{(x_{i+1})} + f_{(x_i)}}{2} (x_{i+1} - x_i) \tag{4.10}$$

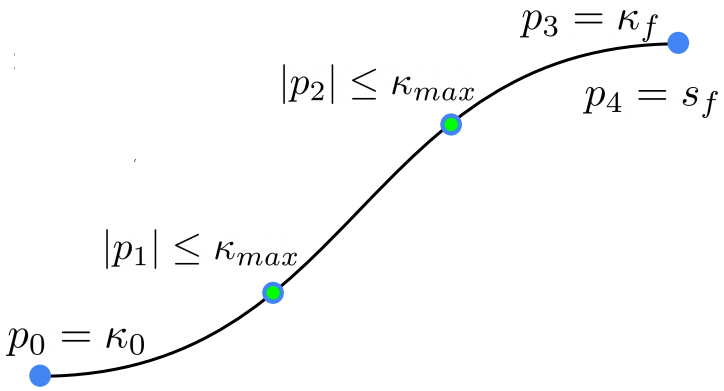
We use software libraries already available to solve those integrals. In particular, the Cumulative Trapezoid function in Python is fundamental for the implementation of the proposed algorithm. After the application of the Trapezoid rule, a discrete representation of each spiral for each goal point is available.

Waslander (2018) points out that it is important to keep track of the curvature value of each point along with the position and heading, because that information is useful to calculate the velocity profile for the whole trajectory. Closed form solutions for the curvature and heading are already available, therefore numerical integration is not required.

Curvature Constraints Approximation

Kelly (2018) mentioned that curvature constraints, which correspond to minimum vehicle turning radius, are useful to create paths that are drivable the vehicle. It is enough to constrain sampled points along the path due to well-behaved nature of spiral's curvature. For instance, curvature constraints at 1/3rd and 2/3rd's of the way along the path are enough to generate the spiral. This approach, and the following steps, were adopted in our work.

Figure 10 – Curvature constraints.



Source: Elaborated by the author.

Bending Energy Objective

The bending energy expression in Eq. 4.11 distributes curvature evenly along spiral to promote comfort. It is the integral of square curvature along the path, which has closed form for spirals, as well as the associated gradient. However, the gradient expression has many terms, so it is better to use a symbolic solver to find its value (KELLY, 2018).

$$f_{be(a_0,a_1,a_2,a_3,s_f)} = \int_0^{s_f} (a_3 s^3 + a_2 s^2 + a_1 s + a_0)^2 ds \tag{4.11}$$

Initial Optimization Problem

As described by Kelly (2018), it is possible to perform the optimization process in the frame attached to the vehicle's body to set the starting boundary condition to zero. The aforementioned constraints and objective function form the full optimization problem as follows.

$$\begin{aligned} \min\ & f_{be(a_0,a_1,a_2,a_3,s_f)} \\ subject\ to\quad & \left|\kappa_{\left(\frac{s_f}{3}\right)}\right| \leq \kappa_{max}, \quad \left|\kappa_{\left(\frac{2s_f}{3}\right)}\right| \leq \kappa_{max} \\ & x_{s(0)} = x_0, \quad x_{s(s_f)} = x_f \\ & y_{s(0)} = y_0, \quad y_{s(s_f)} = y_f \\ & \theta_{s(0)} = \theta_0, \quad \theta_{s(s_f)} = \theta_f \\ & \kappa_{s(0)} = \kappa_0, \quad \kappa_{s(s_f)} = \kappa_f \end{aligned} \tag{4.12}$$

Soft Constraints

It is challenging for an optimizer to solve problems with equality constraints. Kelly and Nagy (2003) point out that it is necessary to soften equality constraints by heavily penalizing the deviation in the objective function. Therefore, considering that it is also assumed that the initial curvature is know, which corresponds to a_0, the optimization problem in 4.12 is rewritten as follows.

$$\begin{aligned} \min\ & f_{be(a_0,a_1,a_2,a_3,s_f)} + \alpha(x_{s(s_f)} - x_f) + \beta(y_{s(s_f)} - y_f) + \gamma(\theta_{s(s_f)} - \theta_f) \\ subject\ to\quad & \left|\kappa_{\left(\frac{s_f}{3}\right)}\right| \leq \kappa_{max} \\ & \left|\kappa_{\left(\frac{2s_f}{3}\right)}\right| \leq \kappa_{max} \\ & \kappa_{s(s_f)} = \kappa_f \end{aligned} \tag{4.13}$$

Parameter Remapping

As mentioned by Kelly (2018), it is possible to remap the spiral parameters with p_0 to p_3 corresponding to curvature at four points equally spaced along path (Eq. 4.8), and p_4

corresponding to the arc length of the spiral s_f.

$$\begin{aligned} a_0 &= p_0 \\ a_1 &= -\frac{11p_0/2 - 9p_1 + 9p_2/2 - p_3}{p_4} \\ a_2 &= \frac{9p_0 - 45p_1/2 + 18p_2 - 9p_3/2}{p_4^2} \\ a_3 &= -\frac{9p_0/2 - 27p_1/2 + 27p_2/2 - 9p_3/2}{p_4^3} \end{aligned} \tag{4.14}$$

Final Optimization Problem

In order to reduce the dimensionality of the optimization problem in Eq. 4.13, p_0 and p_3 are treated as constant values since the initial and final curvature are known before hand. Since the boundary conditions are handled by the aforementioned soft constraints, the final optimization problem is expressed as follows.

$$\begin{aligned} & \min f_{be(a_0,a_1,a_2,a_3,s_f)} + \alpha(x_{s(p_4)} - x_f) + \beta(y_{s(p_4)} - y_f) + \gamma(\theta_{s(p_4)} - \theta_f) \\ \textit{subject to} \quad & |p_1| \leq K_{max} \\ & |p_2| \leq K_{max} \end{aligned} \tag{4.15}$$

The optimization problem in Eq. 4.15 is solved to get a cubic spiral from the vehicle's current location to each end location. The method discards the spirals that do not accomplish the kinematic constraints of the vehicle or are unable to reach the intended goal state, and those spirals are no longer consider in the planning process (KELLY, 2018).

Spiral Parameters

The resulting parameter vector p is calculated solving the optimization problem in Eq. 4.15. Then, it is necessary to undo the initial transformation made on the spiral coefficients (see Eq. 4.14) to retrieve them from the p vector, so the optimization variables are converted back into spiral parameters. Finally, it is necessary to sample points along the spiral to get a discrete representation of the entire path (KELLY, 2018).

Path Set

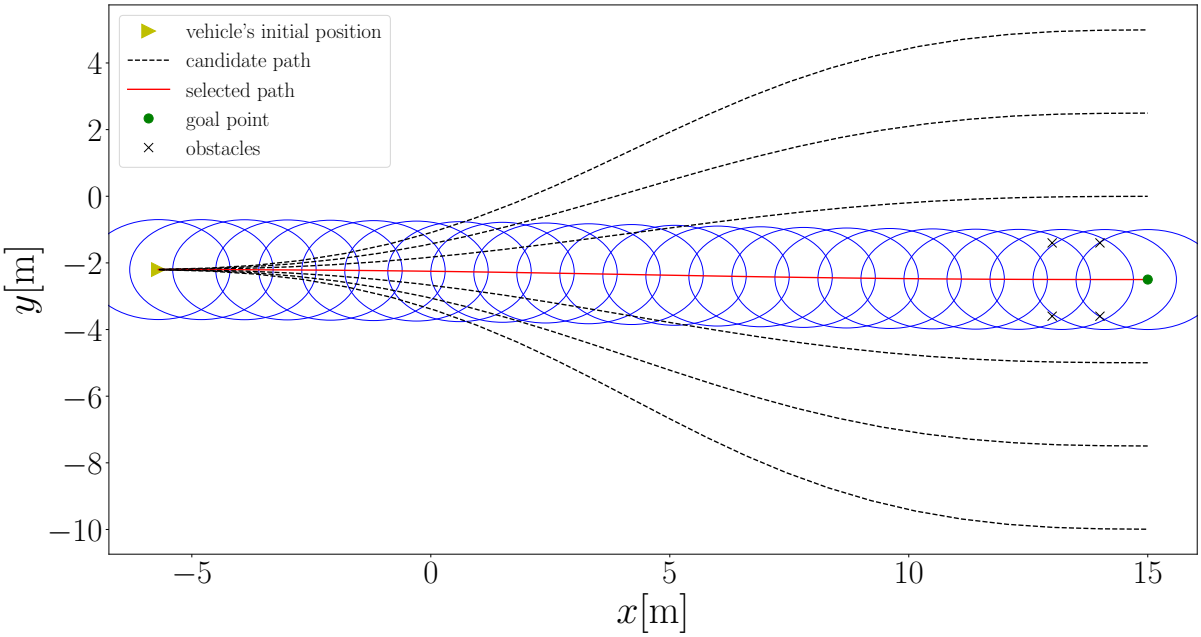
The Trapezoid rule is used to generate the full set of paths for each goal state. Then, the circle decomposition approach for collision checking discussed in a previous section is used to verify that the candidate paths are collision free.

Circle-based collision checking works well when both the ego vehicle and the obstacles in the ego lane, can be enclosed in a circle approximation. The circles representing the autonomous

car are placed at each point along the path, and checked for collisions with each obstacle that falls within the ego lane (WASLANDER, 2018).

Waslander (2018) points out that the circle approximation is very efficient because checking if a point lies within a circle does not require much computation. This approach only verifies if the distance between the obstacle and the center of the circle is less than the radius of the circle. For example, Fig. 11 shows that the position of the obstacles from the center of the circle is less than the radius of the circle on some points of the path, therefore it will generate collision warning. On the other hand, Fig. 12 shows that the obstacles lie outside the radius of each circle, so the selected path will be free from collisions.

Figure 11 – Path generating collisions with obstacles.



Source: Elaborated by the author.

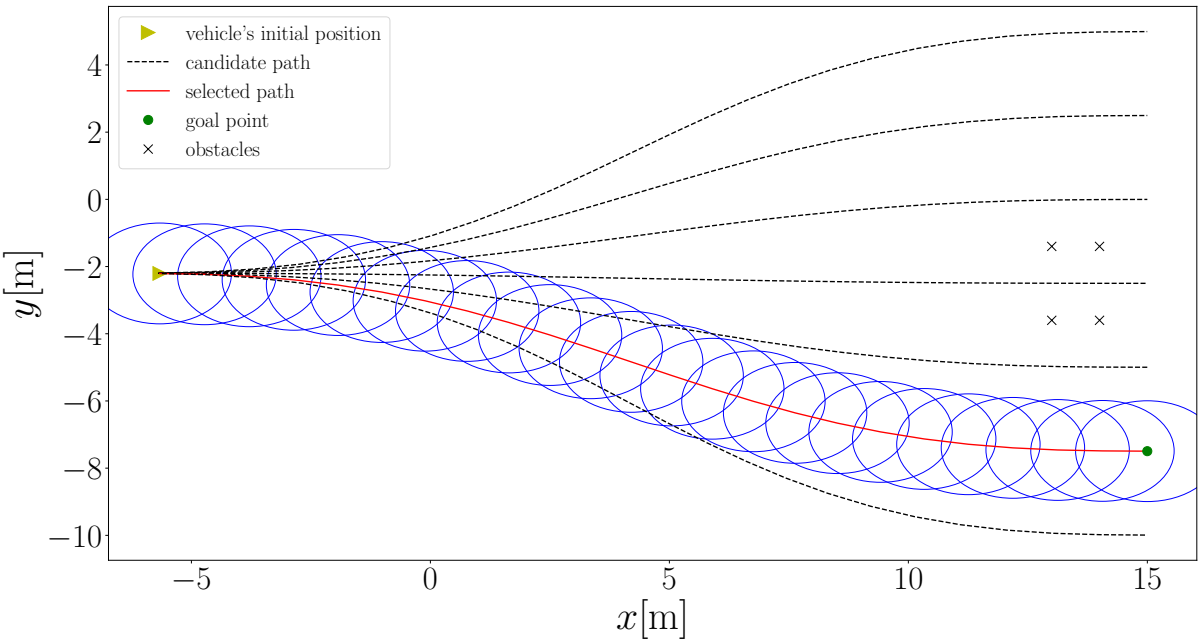
Fig. 13 illustrates the idea behind circle-based collision checking for motion planning with lattices. The autonomous vehicle need to plan around the obstacle. It turns out that after sweeping the swath out across each spiral, the collision-free paths are represented in red color, and the paths that generate collisions with the obstacles are marked with green.

Path Selection

After the set of feasible and collision-free paths is available, a method to select the best one to follow must be established. Waslander (2018) points out that multiple criteria can be considered to design the path selection mechanism, it also depends on the planning application.

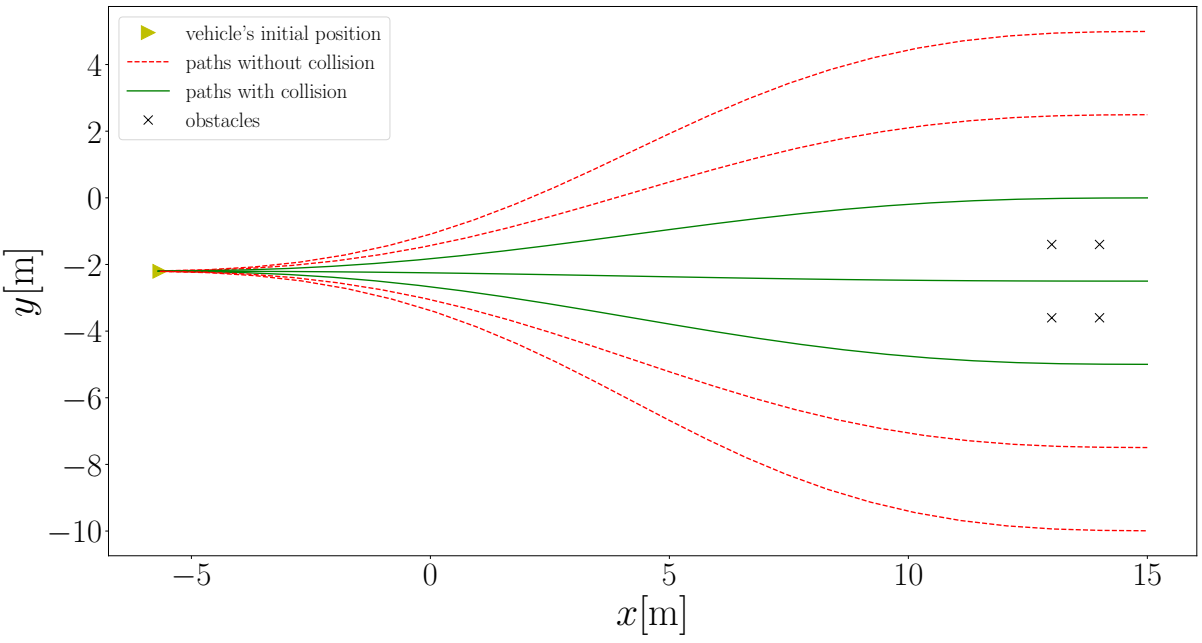
For example, in order to choose paths that are as far from obstacles as possible, it is necessary to include a term in the cost function to penalize paths that come too close to those

Figure 12 – Collision free path.



Source: Elaborated by the author.

Figure 13 – STLP lattice type



Source: Elaborated by the author.

obstacles. In addition, terms that deviate too far from the nearest lane center line are penalized to avoid splitting lanes for too long while making a lane change maneuver.

As suggested by Waslander (2018), we consider a simple metric that introduces a bias to select paths that are as close to the center goal state as possible. In this case, the penalty should increases the farther the candidate lattice get from the central goal. When the planner is biased

toward the center lattice, it follows the reference path, and only deviates from the reference when the reference path is infeasible, or if it ends up generating a collision with an obstacle.

In this project, we consider the penalty function to be the displacement from the center goal state to the goal state of the path under evaluation. Then, we iterate over every path in the path set, find the one that minimizes the penalty function, and select it to create the final path.

Full Path

The lattice creation process is repeated multiple times as the ego vehicle moves along the lane. The STL robustness values is also use to trigger the path deformation procedure which aims to correct initial invalid paths so that recomputing the whole path from scratch is not required. Therefore, the lattice planner generates a full path that converges to the goal state avoiding the obstacles in the environment.

The lattice planner proceeds in a receding horizon fashion towards the goal at the end of the lane. Therefore, it creates smooth and collision-free paths to guide the autonomous vehicle through the environment making progress in the lane (WASLANDER, 2018).

4.3.2 Spatial Preferences

Similar to (BARBOSA *et al.*, 2019; KARLSSON; BARBOSA; TUMOVA, 2020), the focused of the proposed approach is on a safety fragment defined as follows.

The formula $\Phi = \widehat{\mathbf{G}}_I \phi$, where $I = [0, D]$, and

$$\phi := \mu \mid \neg\mu \mid \phi_1 \wedge \phi_2, \tag{4.16}$$

defines a spatial preference over a set of predicates $\{\mu_1, \ldots, \mu_n\}$.

The $\widehat{\mathbf{G}}$ operator is similar to the *Always* operator $\Box_I \phi$ in STL. $\widehat{\mathbf{G}}_I \phi$ states that ϕ should hold every time in the specified interval I, and it is used to evaluate the degree of satisfaction of spacial preferences on a trajectory (KARLSSON; BARBOSA; TUMOVA, 2020).

Example 1. Let us define an upper and a lower bound that the vehicle needs to keep in reference to every obstacle in the environment. Such an specification can be written as follows.

$$\Phi = \widehat{\mathbf{G}}(d_{(\mathbf{x},O)} - D_{min} \ \wedge D_{max} - d_{(\mathbf{x},O)}),$$

in which $\mathbf{x}$ is a trajectory (signal), $d_{(\mathbf{x},M)}$ is a function that returns the Euclidean distance between the path associated to $\mathbf{x}$ and the closest obstacle $o \in O$ marked as occupied, $h_{min} = d_{(\mathbf{x},O)} - D_{min}$ and $h_{max} = D_{max} - d_{(\mathbf{x},O)}$ are predicate functions, and the corresponding predicates μ_{min} and μ_{max} are defined as follows.

$$\mu_{min} = \begin{cases} \top & \iff d_{(\mathbf{x},O)} \geq D_{min} \\ \bot & \iff d_{(\mathbf{x},O)} < D_{min} \end{cases}$$

$$\mu_{max} = \begin{cases} \top & \iff d_{(\mathbf{x},O)} \leq D_{max} \\ \bot & \iff d_{(\mathbf{x},O)} > D_{max} \end{cases}$$

It turns out that the formula Φ is evaluated as true if both μ_{min} and μ_{max} are true for all times in the specified interval.

STL is useful to determine if a user specification, such as the one presented in Example 1, is *True* or *False*. In addition, it is also possible to determine *How much* it is *True* or *False* using the quantitative semantics known as *robustness*.

The proposed approach takes into account the *Always* operator in the STL formulation. Therefore, the autonomous vehicle always respect the preferences described by the STL rules which include the minimum distance to obstacles in the road. The formulation presented in (BARBOSA *et al.*, 2019) and (KARLSSON; BARBOSA; TUMOVA, 2020), for mobile robot exploration and navigation, was adapted to represent the footprint of an autonomous vehicle with circles as describe in a previous section.

The user preferences are encoded in an STL predicate μ that is *True* when $d_{(\mathbf{x}_i,o_j)} \geq D_{min}$, where $\mathbf{x}_i$ with $i \in \{1,\ldots,n\}$ are the trajectories of each circle in the autonomous vehicle footprint, o_j with $j \in \{1,\ldots,m\}$ are the static obstacles in the environment surrounding the vehicle. It is possible to rewrite the inequality as $d_{(\mathbf{x}_i,o_j)} - D_{min} \geq 0$ to formulate a new evaluation function as follows.

$$g_k = d_{(\mathbf{x}_i,o_j)} - D_{min} \tag{4.17}$$

with $k \in \{1,\ldots,n \cdot m\}$. As a result, the general safety formula is expressed as follows.

$$\phi = g_1 \wedge g_2 \wedge \ldots \wedge g_k$$
$$\Phi = \widehat{\mathbf{G}}_I \phi \tag{4.18}$$

This formula ensures that the circles representing the autonomous car footprint will avoid collisions with obstacles in the environment by maintaining a distance of at least D_{min}. As in (TABOADA, 2020), the global operator $\widehat{\mathbf{G}}$ guarantees that the autonomous vehicle always respect the preferences imposed by the STL rules.

4.3.3 Robustness Satisfaction Signal

According to the definition is the previous section, the robustness value is calculated as follows.

$$\rho_{(\mathbf{x},t,d_{(\mathbf{x})} \geq D_{min})} = d_{(\mathbf{x})} - D_{min}$$

The proposed approach only considers the worst-case violation of STL formulas to calculate the robustness value. The robustness measure is based on the minimum distance between the circles

of the vehicle footprint and the static obstacles in its surroundings as described in the following equation.

$$\rho_{(\mathbf{x},t,\phi)} = \min_{\substack{i \in \{1,\ldots,n\} \\ j \in \{1,\ldots,m\}}} (d_{(\mathbf{x}_i,o_j)} - D_{min}) \tag{4.19}$$

The robustness degree of the formula $\Phi = \widehat{\mathbf{G}}_I \phi$ along the path associated to the trajectory $\mathbf{x}$ is expressed as follows.

$$\rho_{(\mathbf{x},\Phi)} = \min_{t \in [0,T]} \rho_{(\mathbf{x},t,\phi)} \tag{4.20}$$

Considering Eq. 4.19, the final robustness function is recursively defined as follows.

$$\begin{aligned} \rho_{(\mathbf{x},\Phi)} &= \min_{t \in [0,T]} \rho_{(\mathbf{x},t,\phi)} \\ &= \min_{t \in [0,T]} \Big(\min_{\substack{i \in \{1,\ldots,n\} \\ j \in \{1,\ldots,m\}}} (d_{(\mathbf{x}_i,o_j)} - D_{min}) \Big) \end{aligned} \tag{4.21}$$

The following example adapted from (BARBOSA *et al.*, 2019; KARLSSON; BARBOSA; TUMOVA, 2020) illustrates how to calculate the robustness value for an autonomous vehicle scenario.

Example 2. Suppose that it is desired that an autonomous car stays 2 meters away from a parked vehicle in the road, it can be expressed in STL as $\Phi = \widehat{\mathbf{G}}\phi$, and $\phi := \mu = d_{(\mathbf{x},t)} - 2$. It is know that $d_{(\mathbf{x},t_0)} = 6$, $d_{(\mathbf{x},t_1)} = 3$, and $d_{(\mathbf{x},t_2)} = 0.8$. In order to analyze the robustness of the path associated to the trajectory $\mathbf{x}$ it is necessary to calculate the robustness for each individual case using the expression in Eq. 4.19. For each individual case the results are $\rho_{(\mathbf{x},t_0,\phi)} = 4$, $\rho_{(\mathbf{x},t_1,\phi)} = 1$, and $\rho_{(\mathbf{x},t_2,\phi)} = -1.2$. The robustness on the path with respect to the formula Φ is calculated using the Eq. 4.21 as follows.

$$\begin{aligned} \rho_{(\mathbf{x},\Phi)} &= \min_{t \in [0,1]} \rho_{(\mathbf{x},t,\phi)} \\ &= \min(\rho_{(\mathbf{x},t_0,\phi)}, \rho_{(\mathbf{x},t_1,\phi)}, \rho_{(\mathbf{x},t_2,\phi)}) \\ &= -1.2. \end{aligned}$$

4.3.4 Dynamic Spatial Resolution

One of the main features of the proposed planner is the introduction of a dynamic spacial resolution value that changes on each planning cycle based on the state of the vehicle and the information of the environment. This spacial resolution value R refers to the space between the end points of the candidate paths. H_m specifies the spacial planning horizon, it indicates the length of the lattices.

The proposed approach enables to configure different spacial resolution values according to the maneuvers the ego vehicle may need to perform in the environment. For instance, consider the case of highway autonomous driving. On highways the motion planning problem is limited to high speed and small curvature roads, with specific rules attached to driving behaviors, under a

constrained environment framework (SUN *et al.*, 2020a). As a consequence, in a typical highway scenario the planner can work with a low spatial resolution value because it does not need to generate many lattices, it could be enough to produce three lattices, one of them can be used to keep the ego vehicle at the center while it moves along the current lane, and the other two lattices could be used to make basic lane change maneuvers, either on the right or on the left, in the absence of static and dynamics obstacles.

The planner also works with a high spatial resolution value to generate many lattices so that the ego vehicle could deal with unexpected traffic situations in the safest way. For example, if the leading vehicle in the front slows down and stops, then the ego vehicle needs to perform an evasive maneuver and do not impact negatively the traffic flow. In this case, the ego vehicle selects one of the multiple lattices available to keep a safe minimum distance from the obstacle while not being so conservative.

The proposed multi-resolution feature added to the lattice planner is also useful to reduce the computational load. The planner generates few candidate paths when the autonomous vehicle performs basic driving maneuvers such as driving directly along the road. However, in the case of maneuvers involving risk such as lane change, merging, and overtaking, the planner generates many candidate paths to avoid unnecessary movements related to the nature of the lattice approach, and more importantly to ensure safety while performing those risk maneuvers.

4.3.5 Lattice Cost

The cost function presented in (KARLSSON *et al.*, 2021; KARLSSON; BARBOSA; TUMOVA, 2020) is adapted to work with the lattice planning approach and in the context of autonomous vehicles. It is defined as follows.

$$J = J_\Omega + J_\Phi \tag{4.22}$$

The definition of J_Ω is based on the Euclidean distance ($\mathbf{L}_2$ norm) from the off-set point (end point) $\mathbf{p} \in \mathbb{R}^2$ of a lattice segment to the goal point $\mathbf{q} \in \mathbb{R}^2$.

$$J_\Omega(\mathbf{x}) = dist_{(\mathbf{p},\mathbf{q})} = \left(\sum_{k=1}^{2} |\mathbf{p}_k - \mathbf{q}_k|^2 \right)^{1/2} \tag{4.23}$$

The definition of J_Φ is based on the robustness degree of an STL formula (DONZÉ; MALER, 2010) which can be expressed in two ways: first, the spacial robustness that evaluates to which extent a trajectory (a signal in STL) deviates from the objective values, and second the time robustness that describes for how long the aforementioned condition of deviation has been happening. (KARLSSON *et al.*, 2021; KARLSSON; BARBOSA; TUMOVA, 2020). J_Φ is used to influence the motion planning process into selecting a path which better respects STL specifications.

The *left-time robustness* $\theta^{-}_{(\mathbf{x},t,\phi)}$ is defined in (KARLSSON *et al.*, 2021; KARLSSON; BARBOSA; TUMOVA, 2020) as the duration leading up to time t in which the STL rule ϕ has been either satisfied or violated. This robustness measure is expressed as follows.

$$\theta^{-}_{(\mathbf{x},t,\phi)} = \text{sign}(\rho_{(\mathbf{x},t,\phi)}) \max\{d \geq 0 \mid \forall t' \in [t-d,t], \text{sign}(\rho_{(\mathbf{x},t',\phi)}) = \text{sign}(\rho_{(\mathbf{x},t,\phi)})\}, \tag{4.24}$$

The cost function $J_\Phi(\mathbf{x})$ in Eq. 4.22 represents a quantitative semantics of the alternative *Always* operator $\widehat{\mathbf{G}}_{[0,T]}\phi$ as follows.

$$J_\Phi(\mathbf{x}) = -\int_0^T \theta^{*}_{(\mathbf{x},t,\phi)} \cdot w_{(\rho_{(\mathbf{x},t,\phi)})}\, \mathrm{d}t, \tag{4.25}$$

where θ^* is a modified version of the *left-time robustness*, and w is a positive duration that weighs the spatial robustness.

The *modified left-time robustness* $\theta^{*}_{(\mathbf{x},t,\phi)}$ presented in (BARBOSA *et al.*, 2019) guarantees that the cost function J_Φ is monotone, as a result J is monotone as well. This property is important to include J in the planning algorithm as a factor that influences the path selection. The *modified left-time robustness* is defined as follows.

$$\theta^{*}_{(\mathbf{x},t,\phi)} = \min(\theta^{-}_{(\mathbf{x},t,\phi)}, 0) \tag{4.26}$$

The aforementioned feature is formalized in Lemma 1. Refer to (KARLSSON; BARBOSA; TUMOVA, 2020) for the proof.

Lemma 1. Given $\theta^{*}_{(\mathbf{x},t,\phi)}$ in eq 4.26 and a positive weight function w, $J_\Phi(\mathbf{x})$ in eq 4.25 is monotonically increasing.

The weight function $w(\rho_{(\mathbf{x},t,\phi)})$ encodes user-defined preferences in its parameters. Furthermore, it expresses the compromise between a time-efficient trajectory and a spatially-preferred one (BARBOSA *et al.*, 2019).

The proposed weight function is characterized by four parameters. The idea of adding more parameters compared with the function described in (KARLSSON; BARBOSA; TUMOVA, 2020) is to provide more flexibility in penalizing the lattices that do not respect the STL constraints imposed by the user. These parameters are useful to establish different levels of risk based on the robustness value. Therefore, it is possible to distinguish between path segments that involve some acceptable risk, and those critical paths that should be avoided because they compromise the safety of the autonomous vehicle.

- α, is an intermediate bound of acceptable spatial robustness. It determines the risk-level associated to a candidate path.
- β, is the lowest bound of acceptable spatial robustness.

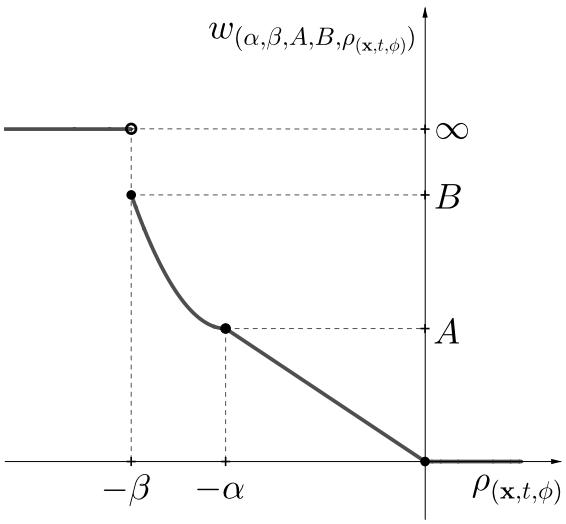
- A indicates the pace at which the trajectory is penalized as it approaches the specified safety bound α. It is the relative weighting between trajectories that violate STL formulas. In addition, this parameter expresses a measure of the flexibility related to the spatial preference set by the user to avoid collisions with obstacles.

- B indicates the pace at which the trajectory is penalized as it approaches the specified safety bound β. Similar to A, it provides a notion of the flexibility associated to the spatial preference imposed by the user to guarantee a safe performance.

Similar to (TABOADA, 2020; KARLSSON *et al.*, 2021; BARBOSA *et al.*, 2019; KARLSSON; BARBOSA; TUMOVA, 2020), the tuning parameters A, B, α, β are used to determine the importance of respecting STL formulas. The higher their values, the higher is the cost associated to candidate paths when they violate safety specifications. The higher the cost, the less likely it is for a violating path segment to be selected as part of the final trajectory. The STL formulas have less influence on selecting the path segments when the tuning parameters have low values.

The proposed function $w = g_{(\alpha,\beta,A,B,\rho_{(\mathbf{x},t,\phi)})}$ is depicted in Fig. 14 and formalized as follows.

Figure 14 – Proposed weight function.



Source: Elaborated by the author.

$$w = \begin{cases} \infty & \text{for } \rho_{(\mathbf{x},t,\phi)} < \beta \\ \frac{B}{\beta}(\rho_{(\mathbf{x},t,\phi)} + \alpha)^2 + A & \text{for } \beta \leq \rho_{(\mathbf{x},t,\phi)} < \alpha \\ -\frac{A}{\alpha}\rho_{(\mathbf{x},t,\phi)} & \text{for } \alpha \leq \rho_{(\mathbf{x},t,\phi)} < 0 \\ 0 & \text{for } \rho_{(\mathbf{x},t,\phi)} \geq 0, \end{cases} \tag{4.27}$$

Inspired by (KARLSSON; BARBOSA; TUMOVA, 2020), the proposed weight function classifies the candidate lattices into different categories as follows.

1. *Most-risky Lattices*: paths that violate the minimum safety requirements specified by the user. In this case, β represents the minimum robustness value that can be accepted, and the infinite weight means that the violation of the spacial preferences is not allowed. Similar to (KARLSSON; BARBOSA; TUMOVA, 2020), $\beta = 0$ is used to establish hard constraints.

2. *Risky Lattices*: paths that violate the spatial preference. They do not involve enough risk to violate the minimum safety limit defined by β, but they exceed the safety limit defined by α which determines the degree of the penalization based on the risk associated to the lattice. The candidate paths in this category involve risky maneuvers that increase the possibility of collisions with obstacles, so they must be heavily penalized, and they should be considered as the last alternative to reach the goal state of the ego vehicle. The weight combines the robustness value with the tuning parameters A,B,α,β in an expression to assign high costs to paths that are close to the safe limit, so that they are not considered in the planning process.

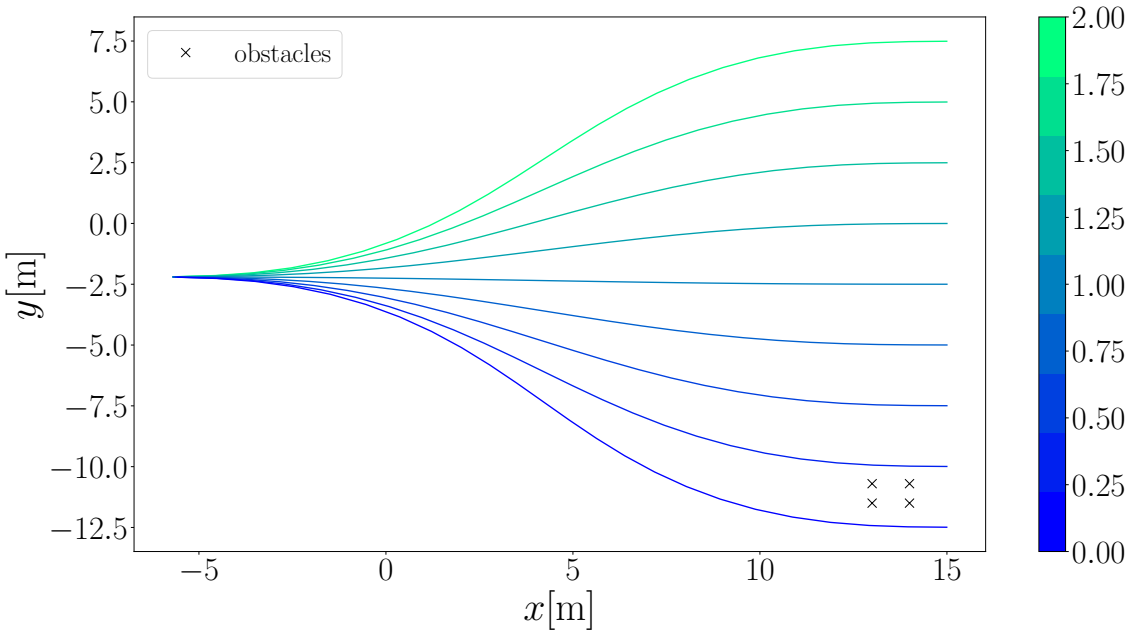
3. *Least-risky Lattices*: paths that violate the spatial preference, but not so much as to violate the limit defined by α. The candidate paths in this category are slightly penalized because involve maneuvers that are not so critical as to compromise the safety specifications. The weight is a scaled version of the robustness value that considers the relationship of the parameter A and α to define the importance of respecting STL formulas.

4. *Non-risky Lattices*: paths that satisfy the safety condition expressed by the STL formulas. A weight of value zero means that the path satisfies the spatial preferences, and therefore should not be penalized.

Fig. 15 shows how the candidate paths are classified according to the level of risk that each one represents based on the distance to the obstacles. The blue path is the most risky option and the green is safest.

The weight function w contributes to the cost function J_Φ based on how much and how long the trajectories violate the safety specification. It assigns a very high weight on trajectories that violate the STL rules and involve maneuvers that are so risky. It sets a moderated weight on trajectories that violate the user preferences about safety distance, but do not imply a considerable risk for the ego vehicle. It assigns a weight of value zero on trajectories that do not violate the safety specifications.

Figure 15 – Levels of risk for candidate paths.



Source: Elaborated by the author.

4.3.6 Safe Trajectory Creation

The final path connecting the initial and goal position is used by a function that runs a forward simulation of the vehicle to create the associated trajectory. During the trajectory generation process a closed-loop controller moves and stabilizes the autonomous vehicle along the given final path. At this stage, the trajectories are simulated not executed, so robust trajectory tracking controllers are not necessary. In this work, steering and velocity controllers are considered in the same way as in (KUWATA *et al.*, 2009; LIN; MAIERHOFER; ALTHOFF, 2021).

Steering Controller

As mention by Lin, Maierhofer and Althoff (2021), the pure-pursuit controller has certain flexibility with respect to path representations and it is easy to implement in practical form. It is used to calculate the desired steering angle as follows.

$$\delta_d = \tan^{-1}\left(\frac{2l_{wb}\sin(\eta)}{L_{fw}}\right) \tag{4.28}$$

where L_{fw} is the forward drive look-ahead distance and η is the heading of the look-ahead points on the target line from the rear axle based on the vehicle orientation (see Fig. 7).

As in (LIN; MAIERHOFER; ALTHOFF, 2021), the steering velocity is used as the lateral input to avoid undesirable motions such as sudden movements. As a result, the steering

controller combines the pure-pursuit controller with a PI controller, as follows.

$$v_\delta = K_P(\delta_d - \delta) + K_I \int_0^t (\delta_d - \delta) d\tau, \tag{4.29}$$

where K_P, and K_I are the proportional and integral gain, respectively. Refer to (KUWATA *et al.*, 2008) for details about the linear stability analysis of the steering controller.

Velocity Controller

The PI controller described in (KUWATA *et al.*, 2009) is adopted to track the desired velocity value v_{cmd}. Moreover, the rate of acceleration change must satisfy the established limit to enhance comfort by taking into account the constraint $da/dt \in [j_{min}, j_{max}]$ with j_{min} and j_{max} describing the minimum and maximum permissible jerk values, respectively.

4.4 STL-LP Algorithm

Fig. 16 describes the proposed motion planning algorithm. First, candidate paths are generated using a Conformal Lattice Planner (MCNAUGHTON *et al.*, 2011) modified to generate two-dimensional paths considering a planning horizon strategy. The planning cycle starts with the vehicle's initial state S_0 that is used to compute the resolution of the separation between lattices. The planner starts in low resolution mode so it generates few lattices to handle essential driving situations (e.g. three lattices are enough on a typical highway scenario, one to follow the current road, and two to make lane change maneuvers either on the right or on the left).

The next step is the generation of multiple lattices using R_L, the computed resolution, and S_L, the state of the vehicle at this stage. The best path among the generated lattices is selected considering the cost function to minimize. Then, the robustness value of the selected path is calculated based on the STL rules that guarantee safety. The robustness measure of the current lattice and an a predefined robustness value are jointly used to calculate a safety indicator value which is compared with a threshold to decide either to repair the selected path or to include the current path segment to the final path.

The deformation process, if necessary, involves determining a cut-off state related to the initial selected path. The cut-off state includes the breaking point that will be used as the starting point for the new candidate paths with safety constraints. Moreover, the cut-off state is used to compute the new resolution of the separation between lattices with safety constraints. The planner is in the high resolution mode during the deformation process of the initial path, as a consequence, the lattice generator will produce multiple paths aiming to provide enough safety to avoid collisions with obstacles in the environment surrounding the autonomous vehicle. This time the selection procedure of the best path is based on additional constraints related to user preferences. For each new path the robustness values are calculated and transformed into a gain to influence the selection of a new path from the cut-off point to the goal point. Once again the

safety indicator value is calculated to decide either to make a finer adjustment of the current path or to continue with the planning cycle.

The planning cycle continues to the next stage when it is not necessary to repair the generated path. The current path, either the initial path segment or the repaired path segment, is combined with the path segments previously generated in past cycles of the planning procedure.

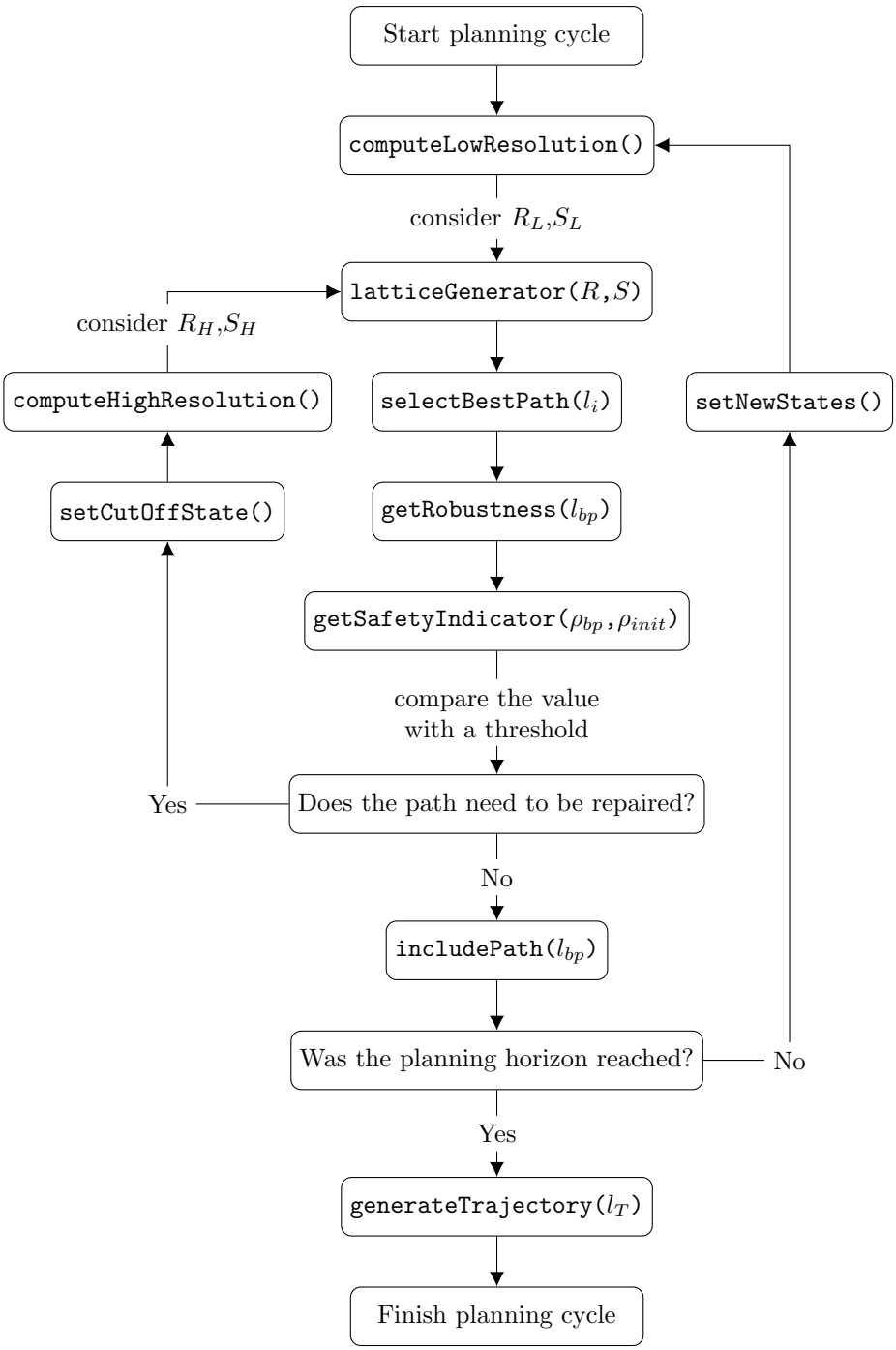
Finally, it is necessary to determine if the final path that includes all the path segments up this stage is enough to reach the specified planning horizon. New vehicle states are considered when the final path does not reach the specified limit, these states are used to generate a new path segment. On the other hand, when that planning horizon has been reached, the last step is to form the final trajectory and to finish the planning cycle.

The functions mentioned in the flowchart of Fig. 16 as detailed as follows.

- `computeLowResolution()`: This function takes as input the current state of the vehicle to calculate the values of the parameters that are necessary to create the candidate paths. It computes the low resolution degree R_L for the lattices, and the associated goal state S_L.
- `computeHighResolution()`: This function takes as input the cut-off state that is used as the starting point to generate new alternative paths that ensure safety. It computes high resolution degree R_L for the lattices, and the associated goal state S_H.
- `latticeGenerator()`: This function takes as inputs a resolution value (either R_L or R_H), and a goal state (either S_0, or S_L, or S_H) to generate candidate paths conformed to the structure of the road.
- `selectBestPath()`: This function takes as input the set of candidate lattices l_i that are the paths joining the initial state and the desired goal state.
- `getRobustness()`: This function takes as input the best candidate path l_{bp} and calculates its robustness value ρ_{bp}.
- `getSafetyIndicator()`: This function takes as inputs the robustness value of the best path ρ_{bp}, and a predefined robustness indicator ρ_{init}. It calculates a safety indicator value that is compared with a threshold to decide if the path deformation process is required.
- `setCutOffState()`: This function determines the cut-off state that is used as the initial state to build alternative paths that ensure safety maneuvers.
- `setNewStates()`: This function updates the states of the vehicle.
- `includePath()`: This function takes as input the best candidate path l_{bp} in the current planning cycle and combine it with previous segments aiming to reach the specified planning horizon.

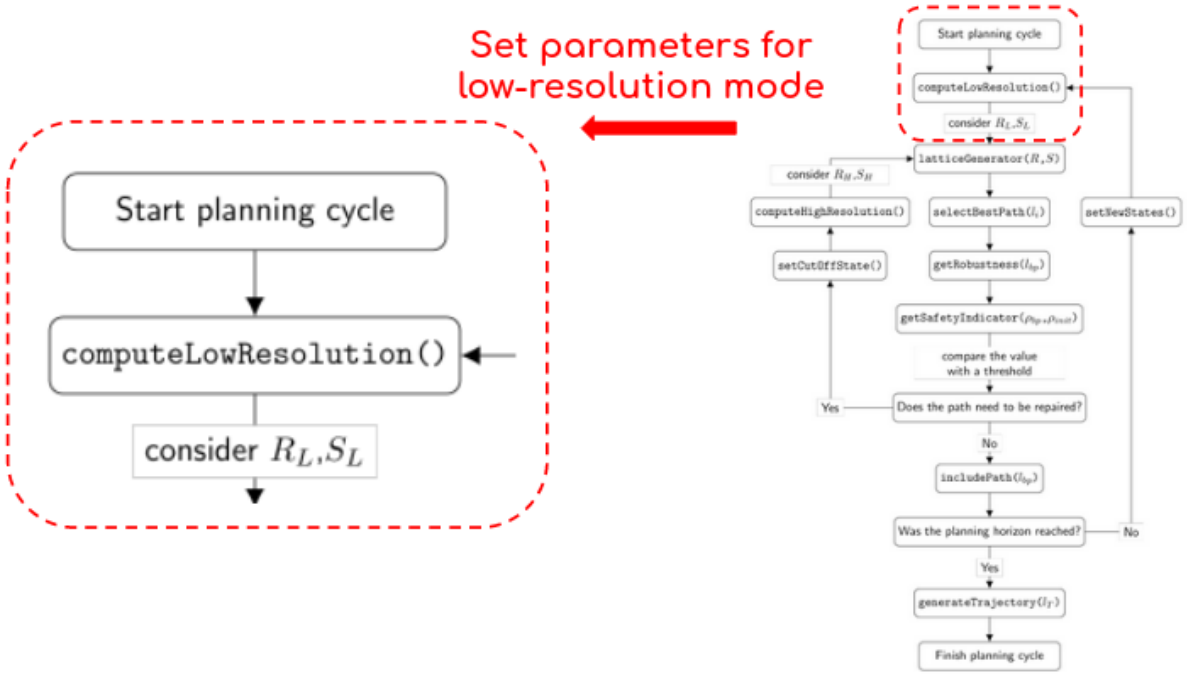
- `generateTrajectory()`: This function takes as input the final path that was deformed to ensure safety, and generates the associated trajectory.

Figure 16 – Flowchart of the trajectory deformation approach.



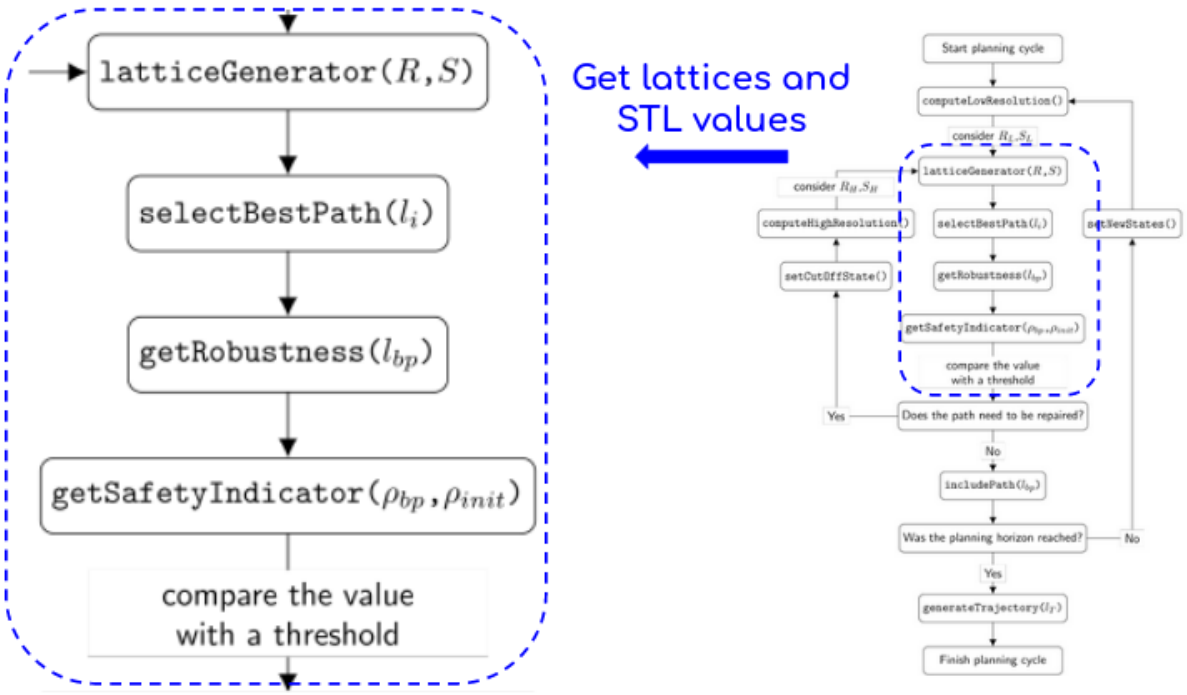
Source: Elaborated by the author.

Figure 17 – Detailed Flowchart (part 1).



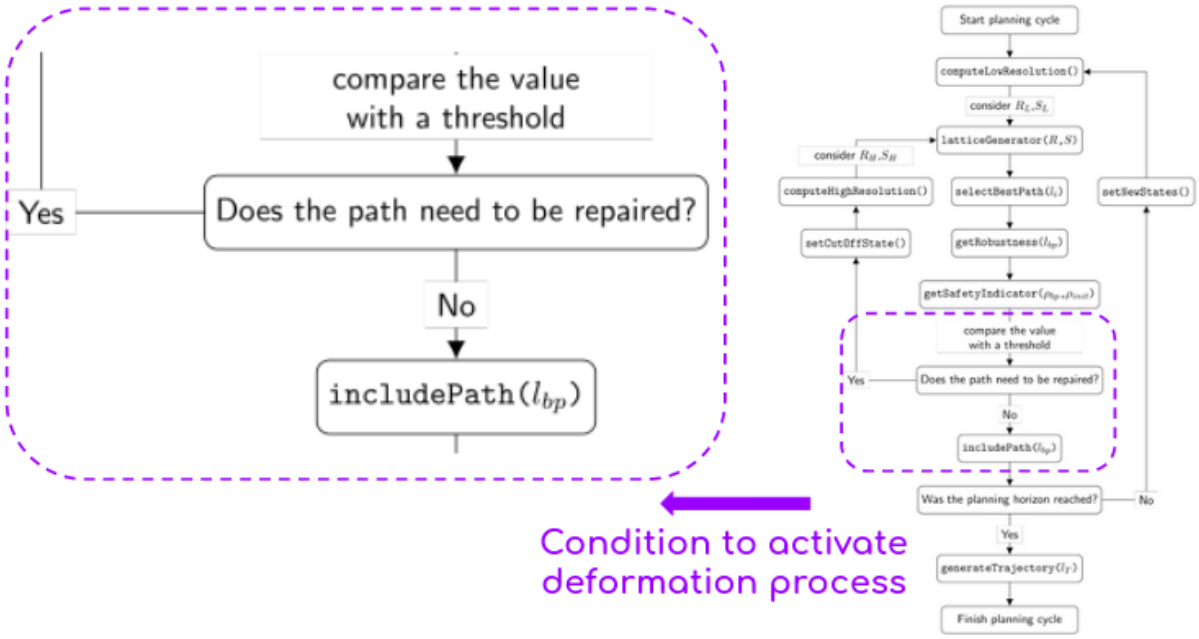
Source: Elaborated by the author.

Figure 18 – Detailed Flowchart (part 2).



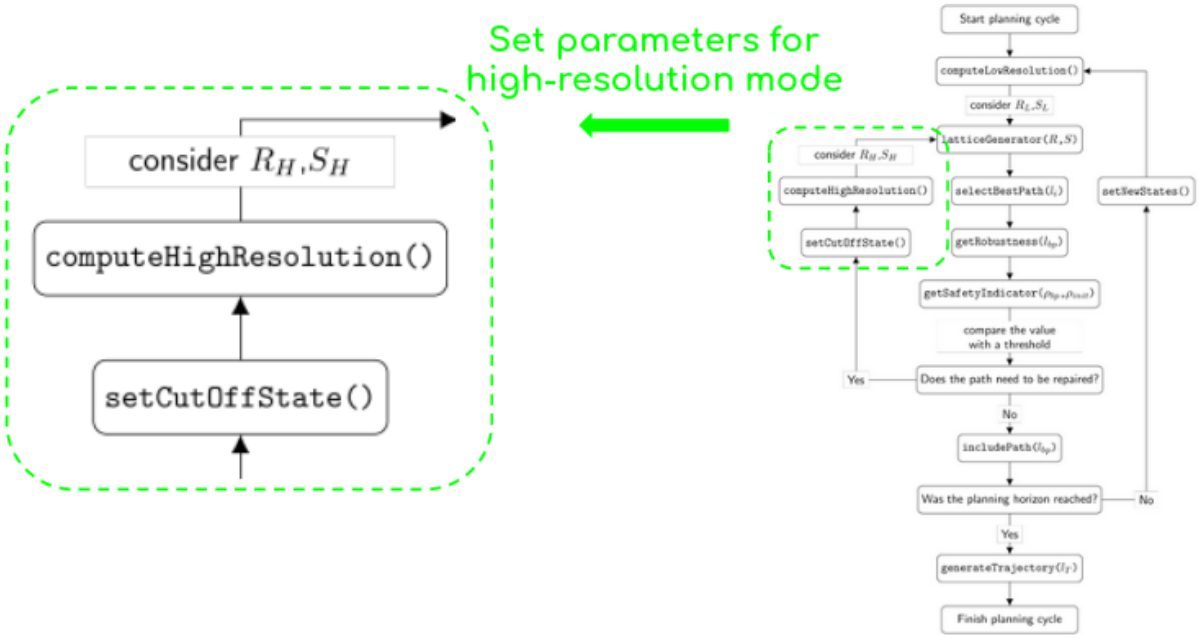
Source: Elaborated by the author.

Figure 19 – Detailed Flowchart (part 3).



Source: Elaborated by the author.

Figure 20 – Detailed Flowchart (part 4).

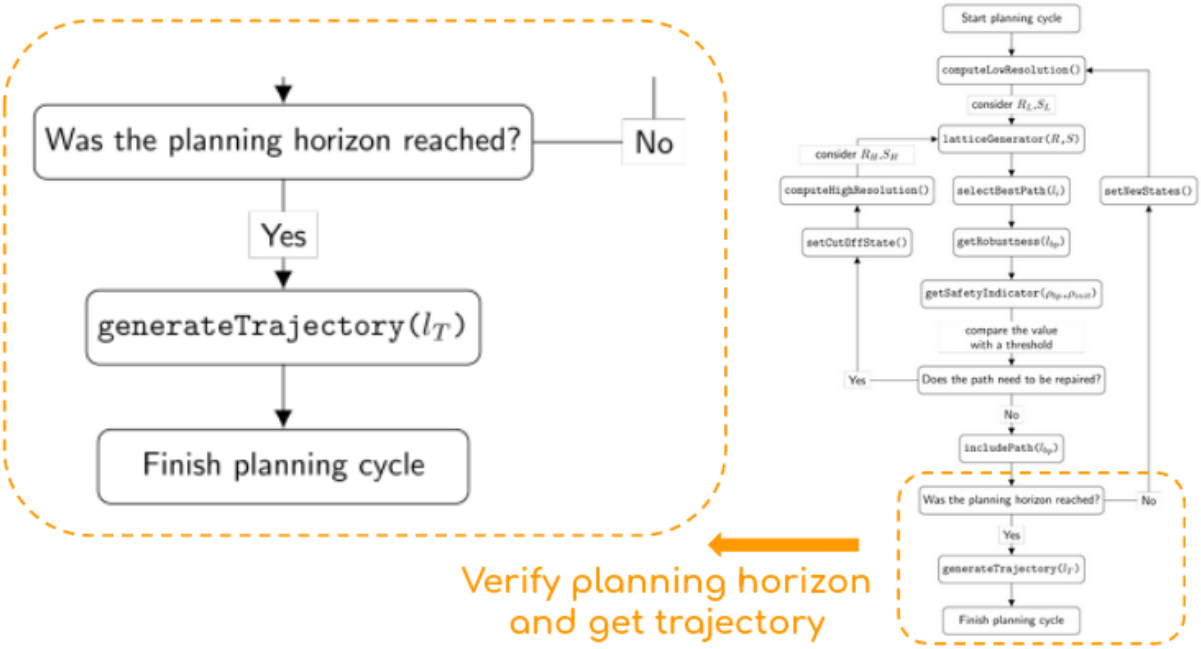


Source: Elaborated by the author.

4.5 Final Considerations

This chapter described a novel approach to solve the trajectory repairing problem. The proposed algorithm present advantages over other solutions available in the motion planning literature. It uses a lattice generator with a multi-resolution feature, it means that the planner

Figure 21 – Detailed Flowchart (part 5).



Source: Elaborated by the author.

creates as many candidate paths as necessary depending on the traffic scenario, which reduces the computation required by the planning process. Signal Temporal Logic is used to introduce constraints regarding evasive maneuvers to avoid obstacles in the environment in the safest way possible. Moreover, the STL robustness metric is used in the cost function to penalize both, the paths that lead to extremely conservative behaviors, and those paths that involve high risk. The STL robustness values is also use to trigger the path deformation procedures to correct initial invalid paths so that recomputing the whole path from scratch is not required.

CHAPTER

5

RESULTS

This chapter presents the main results achieved with the proposed approach for solving the trajectory repairing problem for autonomous vehicles. First of all, it presents the quantitative results regarding the computation effort required by the proposed algorithm. Then, it describes the results for the traffic scenarios created in the Webots simulator to evaluate the efficiency of the proposed algorithm. An analysis of the collision avoidance maneuver for both, static and moving obstacles is presented.

5.1 Computational Load Evaluation

We used the automobile tools of the robot simulator Webots to test our approach. Our approach is implemented in Python and executed on a computer with an Intel i7-4710HQ 2.50 GHz processor and 12 GB of DDR3L 1600 MHz memory. We use the method described in (MCNAUGHTON *et al.*, 2011) to generate the lattices and then deform them using our approach when necessary.

As mentioned by Sun *et al.* (2020a), the main challenges regarding existing lattice planners are the dependence in the high dimensionality of the lattice, and the lack of an appropriate vehicle behavior modeling. Moreover, existing approaches do not consider STL formulas to represent user preferences about safety.

Our solution seeks to reduce the computational load of a lattice-based planning algorithm by adding the multi-resolution feature, this means that the resolution in terms of the number of lattices varies depending on the traffic situation that the autonomous vehicle has to solve, resulting in a great advantage related to the computational effort required for the algorithm. Considering Table 4 it is notorious how the computational load increases when the resolution increases in terms of the number of lattices generated by the planner to be used as candidate paths. In general, there will be a huge preference for a high resolution, that is, the greatest number

of alternative paths is preferred to have greater freedom of movement, but this considerably increases the computational cost of the planning algorithm, and it is not necessary to have a large number of lattices available in all cases, which makes such solutions inefficient in term of the computation required.

Table 4 – Computational load for lattice generation. Time is an average over 10 runs.

Number of Lattices	Planning Time	Unit
3	281	[ms]
4	294	[ms]
5	315	[ms]
6	321	[ms]
7	349	[ms]
8	359	[ms]
9	363	[ms]
10	376	[ms]
19	499	[ms]

Source: Elaborated by the author.

Fig. 22 illustrates the generation of standard lattices, as used in (MCNAUGHTON *et al.*, 2011; WASLANDER, 2018; SUN *et al.*, 2020a), for the same static obstacle avoidance traffic scenario. Fig. 22a depicts the generation of four notably separated lattices, and given the specified cost function, the planner ends up choosing the most suitable path so that the vehicle does not collide with the obstacle in front. Fig. 22b presents the generation of six lattices for the same traffic situation of obstacle avoidance, and although there are more paths available, the planner determines that the best option is the one that moves the vehicle further away from the obstacle. Fig. 22c illustrates the generation of 10 lattices, it is evident that despite having more options available, the selected path, although not the most conservative in terms of distance to obstacles, is one that considerably distances the vehicle from the obstacle generating an unnecessary deviation, this is due to the configuration of the cost function that strongly penalizes the fact of getting too close to the obstacle.

Fig. 22d presents the generation of nineteen lattices for the same obstacle avoidance traffic scenario, and again despite having many options, the selected path generates an exaggerated deviation moving the vehicle too far from the obstacle, and in this case it is more than evident that the computational effort is greater than the options with fewer candidate paths (see Table 4). However, the solution with the largest number of lattices continues to select the most conservative path in terms of distance to the obstacle, this is due to the established cost function. Although the selected paths are the ones that avoid collisions with obstacles, this selection is not the most efficient in all cases. Considering a standard lattice-based path generation, it is not possible to control exactly how far from the obstacles the vehicle will be when performing the evasive maneuver, because that is defined by the planner cost function whose parameters are configured

in advance at the design stage, but it is unknown exactly which of the paths will be selected at the execution stage.

The configuration of a fixed and low resolution involves the risk that the maneuver is safe but extremely conservative in terms of distance to obstacles. On the other hand, with a fixed and high resolution, it is possible to have a large number of available lattices, but this does not ensure that the selected path enables to maintain a desired distance from the autonomous vehicle to the obstacles on the lane. Something that is guaranteed with a high resolution is the associated high computational load, which could be unnecessary in many situations.

In the aforementioned path selection examples (see Fig. 22), the preferences that the user could have in terms of safety are not taken into account. It is the cost function, with previously tuned parameters, which determines the selected path that in many situations can generate undesirable behaviors for the autonomous vehicle, such as moving too far away from the obstacle in a kind of excessive zig-zag movement, or on the contrary, approaching the obstacles in a dangerous way. It is really unknown how the planner will react in all possible traffic situations. For the exposed reasons, we proposed to introduce STL formulas that represent the security restrictions that must be respected in all cases, this gives greater reliability to the autonomous vehicle in terms of the evasive maneuvers that it performs.

An autonomous vehicle with a lattice planner may perform either a dangerous behavior or a conservative one depending on the partial cost functions (ALTHOFF; KOSCHI; MANZINGER, 2017) to be minimized. Fig. 23 illustrates the difference between a state-of-the-art Conformal Lattice Planer (CLP) (MCNAUGHTON *et al.*, 2011) and the proposed STL-based Lattice Planner (STL-LP). Our approach starts from a cut-off state in the initially generated lattices and builds multiple lattices with a finer resolution, it gives the possibility to be slightly far away or near the obstacle. The new deformed path is attached to STL rules imposed by the user. As a result, it is not necessary to care about unexpected traffic situation in the environment because the vehicle always keeps a safe distance to the obstacles which leads to safer behaviors.

Table 5 summarizes the results obtained by applying our method of trajectory deformation with lattices conditioned to STL formulas. It is clear that the computational cost increases based on the number of generated candidate paths, but the computational effort of applying our motion planning approach is less than that computational cost achieved with the standard lattice generation procedure reported in the motion planning literature (MCNAUGHTON *et al.*, 2011; WASLANDER, 2018; SUN *et al.*, 2020a). The proposed multi-resolution characteristic is very useful to adapt the lattice generation procedure to the current traffic situation which may end up reducing the required calculations.

The proposed planning method considers the analysis of the STL robustness value associated with the initially selected path, and based on that information determines the activation of the deformation procedure to get the a collision free path that respects user-defined STL rules. This repairing of trajectory process is activated only when it is necessary, which further reduces

the computational cost. For instance, according to Table 5 a planning algorithm using eleven standard lattices requires 397 ms of planning time in order to get a selected path. However, with our approach if the deformation process of the initial selected path is activated, then planning time falls down to 377 ms. Moreover, if the deformation process is not necessary, then the motion planning is achieved with a lower cost, because the planning time required by the algorithm using only four lattices is 294 ms as indicated in Table 4.

Table 5 – Computational load with lattice deformation. Time is an average over 10 runs.

Number of Lattices	Planning Time (Standard Lattices)	Planning Time (Deformed Lattices)	Unit
11	397	377	[ms]
12	409	401	[ms]
13	424	407	[ms]
14	437	419	[ms]
15	442	429	[ms]
16	463	453	[ms]
17	484	461	[ms]
18	486	477	[ms]
19	499	481	[ms]
20	526	489	[ms]
21	548	511	[ms]

Source: Elaborated by the author.

Fig. 24 presents the paths obtained using deformed lattices for an obstacle avoidance traffic scenario. Fig. 24a shows that it is possible to navigate with low resolution generating only five lattices at the beginning, then the resolution is increased with seven lattices, conditioned to STL formulas, that are generated only when the deformation process is activated. The activation of this additional step of repairing in the planning cycle is based on an analysis of the STL robustness value associated with the initial selected path. So for this traffic scenario involving evasive maneuvers to avoid collisions with obstacles, our algorithm generates twelve lattices in total, five standard lattices and seven STL lattices, which requires a computational load of 401 ms according to Table 5, which can be reduced to 315 ms according to Table 4, in case of not activating the deformation process, and instead working just with the initial five standard lattices.

Similarly, Fig. 24b, Fig. 24c, and Fig. 24d illustrate the advantage of our method that modifies the initial path, generating more options that are subject to user-defined constraints and translated into STL formulas that directly impact the cost function to select a path with an acceptable level of safety. Although increasing the number of lattices also increases the computational cost, our trajectory deformation method is better in terms of cost than a conventional lattice-based planner (see Table 5). Moreover, the fact that the deformation process is activated only in strictly necessary cases opens the possibility of further reducing the computational cost.

Finally, as we are considering the road scenarios (we do not consider urban traffic

scenes), so, times of order of 0.5 seconds are considered adequate to a good decision making and reaction. Also, the proposed approach is considered as "trivially parallelizable", since each curve (individual path) can be processed separately, being analysed and evaluated in parallel to the others (data processing of each curve is independent to the others). This allows to exploit parallel architectures to implement this approach, as for example, adopting GPUs, multi-core, or FPGAs based hardware solutions.

5.2 Static Obstacle Scenario

The first scenario considered for the evaluation of the proposed algorithm is an urban two-lane road. In this scenario (see Fig. 25), there are four static obstacles blocking and preventing the autonomous vehicle to make progress on the driving lane. The obstacles represent barriers work of a construction sites. The autonomous car should stop to prioritize safety, but this behavior may impact the traffic flow negatively creating undesirable traffic jams. Instead, the autonomous vehicle can perform an appropriate evasive maneuver to avoid collision while making progress towards its goal. Our algorithm is useful to generate safe and feasible trajectories to guide the autonomous car in the aforementioned traffic situation.

Initially, a motion planner may relax the planning problem considering few points to represent the obstacles, so that it requires low computational effort to explore the state space. However, this approach will end up generating either a path that collides with the obstacles or a path that is too conservative that keeps the vehicle unnecessary far from the obstacles and invades the other lane creating hazardous situations. Fig. 26 shows the results using our algorithm, which deforms the initial selected path starting on a cut-off state previously defined. Thus, our approach takes part of the initial path and builds alternative lattices that are conditioned by STL formulas representing user preferences regarding safety while performing evasive maneuvers.

As depicted in Fig. 26, the proposed method is useful to avoid excessive oscillations and extremely conservative behaviors. The autonomous vehicle keeps itself away from the obstacles as much as it is necessary to provide safety guarantees while it performs obstacles avoidance maneuvers.

The final deformed trajectory keeps the autonomous vehicle to a safe distance from the obstacles all the time, as shown in Fig. 27. The dash green line represent the minimum safe distance of 3 meters specified by the user, while the blue line shows the minimum distance to all obstacles when the vehicle makes progress on the lane using the repaired trajectory. The results show that the vehicle keeps a distance greater than the minimum distance required, which increases the confidence regarding the evasive maneuver performed by the vehicle. In the worst case scenario, the vehicle is 5 meters near the obstacles, which is an acceptable result, since the minimum required is 3 meters.

5.3 Moving Obstacle Scenario

The second traffic scenario used to evaluate the performance of our solution to the trajectory deformation process is a overtaking maneuver in a urban two-lane road. As depicted in Fig. 28, there is a vehicle in front (white) with a speed lower than the speed of the autonomous vehicle (green) that can be considered a moving obstacle, so the ego vehicle can perform an evasive maneuver, known as overtaking, to continue progressing on the road. In order to avoid traffic jams the autonomous vehicle may overtake the neighbor vehicle in front, but safety must be prioritized to avoid accidents on the road. Therefore, the autonomous vehicle can perform an appropriate evasive maneuver to avoid collision while making progress towards its goal. Our algorithm is useful to generate the safest trajectory stuck to user specifications regarding safety.

Initially, a motion planner with low resolution may generate few paths to reduce the computation, but this alternative leads to over conservative behaviors to overtake a front vehicle. For instance, if the autonomous vehicle performs an exaggerated avoidance maneuver, it could end up drifting into the left or right lane and collide with other vehicles on the road.

Fig. 29 presents the results using our approach to deform initial planned paths that lead to undesirable driving behaviors that may impact negatively the traffic flow. Our solution analyzes the initial path based on STL formulas that represent user preferences. Then, based on the robustness STL value, it activates the path correction procedure, generating alternative paths that prevent the autonomous vehicle from moving an unnecessary distance away from the moving obstacle, and on the contrary, keeping it at an appropriate distance from the neighboring vehicle, so as not to risk the safety of those involved while performing the overtaking maneuver.

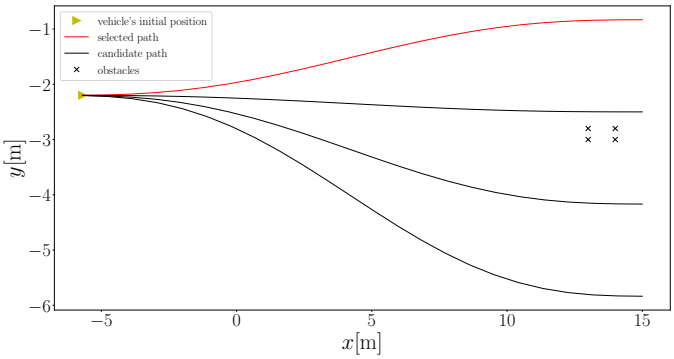
Finally, Fig. 30 depicts the fact that the autonomous vehicle keeps a safe distance from the neighbor vehicle at all times during the evasive maneuver. Once again, the green line represents the minimum safe distance desired by the user, in this case 3 meters. In addition, the blue line represents the distance to the neighboring vehicle during the overtaking maneuver. The results show that the autonomous vehicle is kept at a distance greater than that specified one all the time, and that the closest it gets to the neighboring vehicle is 4 meters, this means that the behavior achieved is the safest possible without generating an extremely conservative behavior.

5.4 Final Considerations

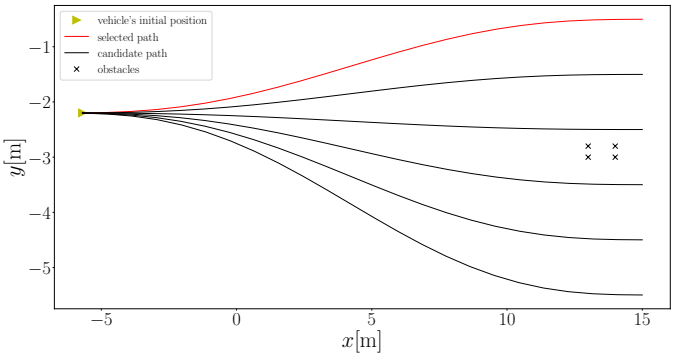
This chapter presents the main results achieved with the proposed approach for solving the repairing of trajectories problem. It turned out that the algorithm is able to repair initial invalid paths in order to generate safe motion for autonomous vehicles performing evasive maneuvers. Moreover, the corrected path ensures safety while taking into account STL formulas representing user preferences. The multi-resolution feature resulted to be very useful to reduce the computation load. The resolution depends on the traffic scenario in which the vehicle is

involved. For instance, when the vehicle is driving in a highway without obstacles the planner works well with low resolution because few lattices are required, one lattice is enough to make progress through the center of the lane, and two additional lattices are necessary to perform lane change either to the left or to the right. On the other hand, when the vehicle has to deal with unexpected obstacles, the planner can change to a high resolution to generate many lattices that lead to safer maneuvers. Additionaly, the STL robustness value of the selected path determines the activation of the deformation process that generates a final collision-free trajectory.

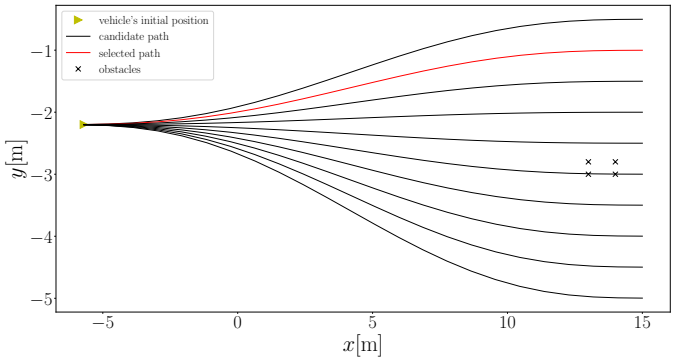
Figure 22 – Standard lattice generation.



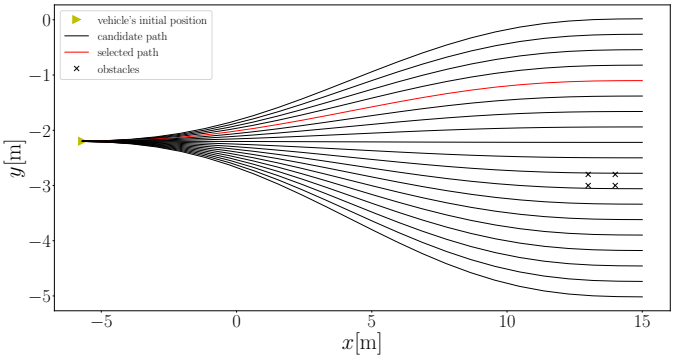
(a) 4 lattices.



(b) 6 lattices.



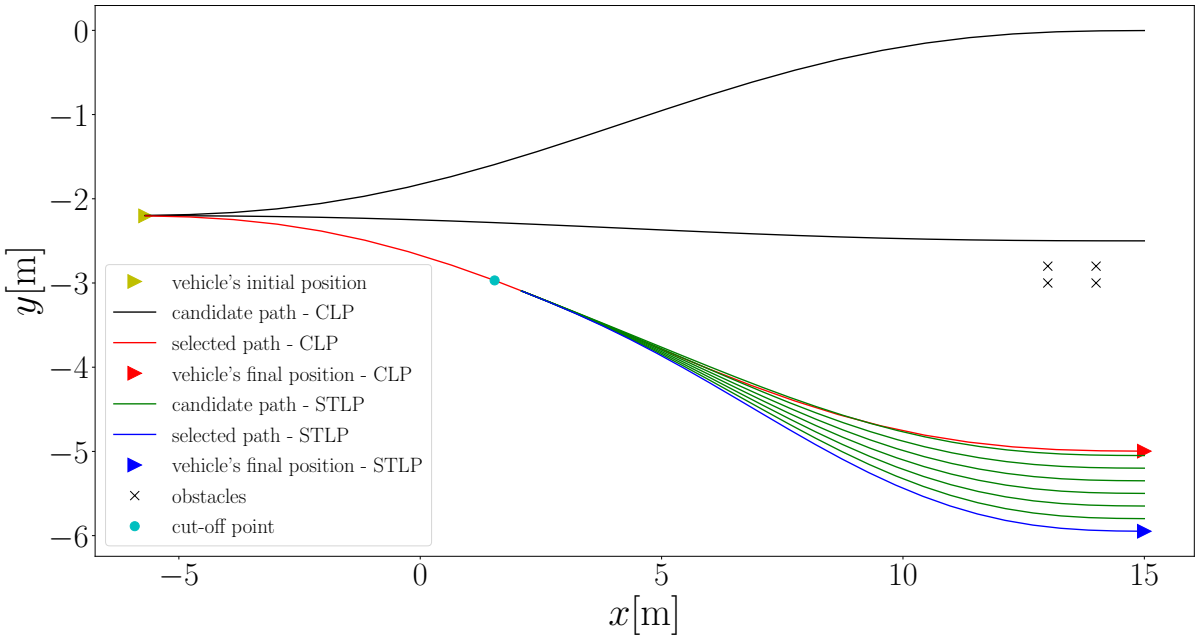
(c) 10 lattices.



(d) 19 lattices.

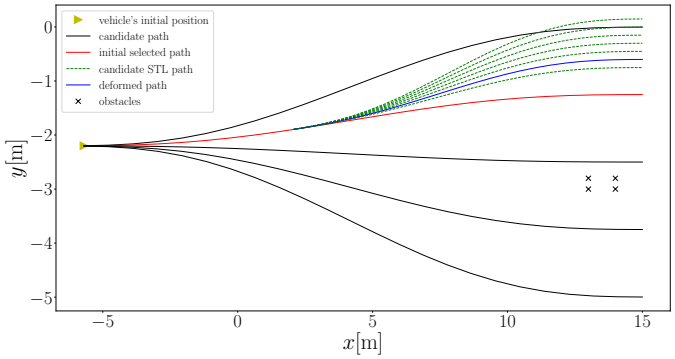
Source: Elaborated by the author.

Figure 23 – Comparison between CLP (MCNAUGHTON *et al.*, 2011) and proposed STL-LP.

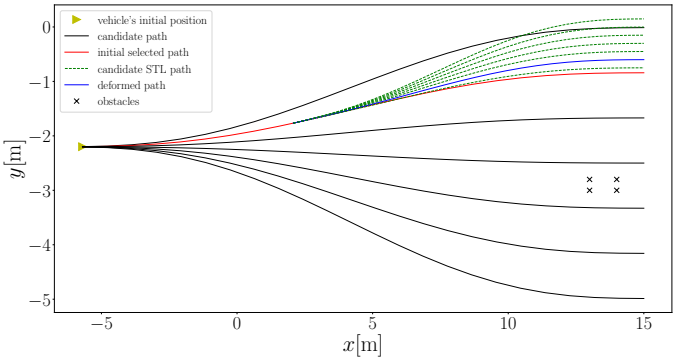


Source: Elaborated by the author.

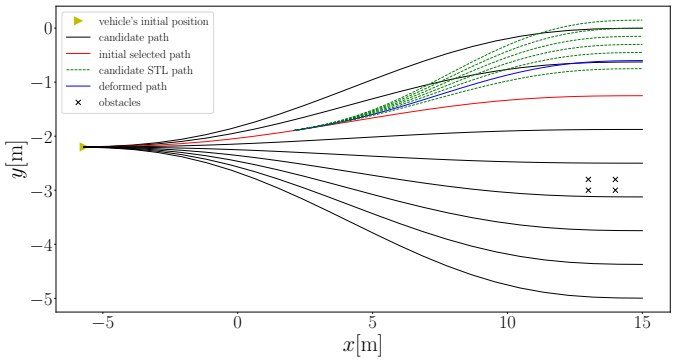
Figure 24 – Lattice deformation.



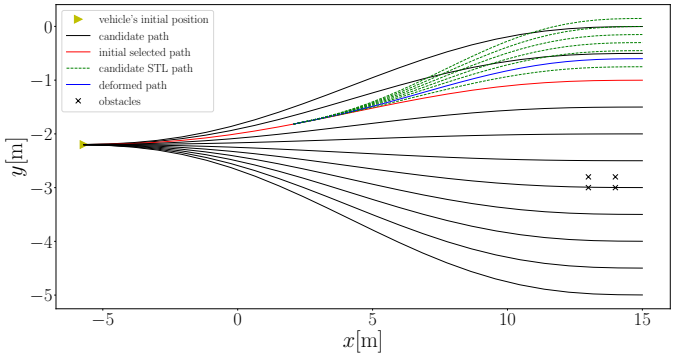
(a) 12 lattices.



(b) 14 lattices.



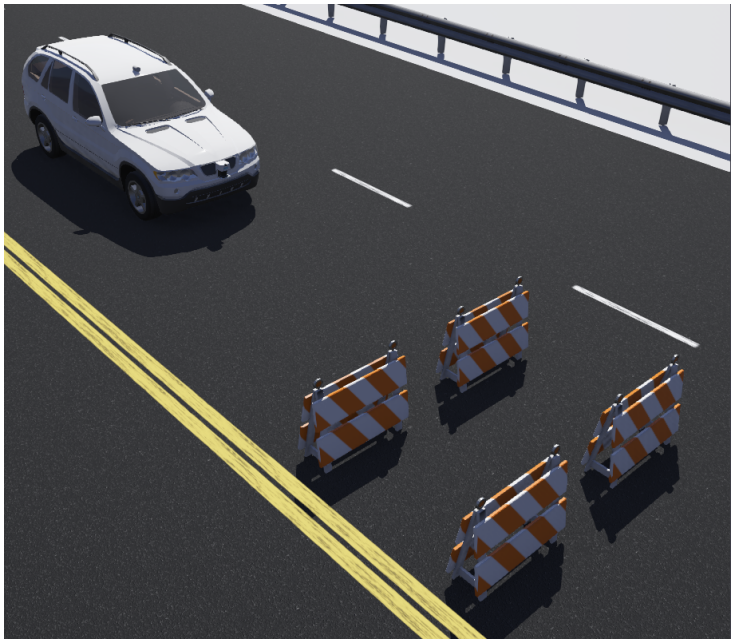
(c) 16 lattices.



(d) 18 lattices.

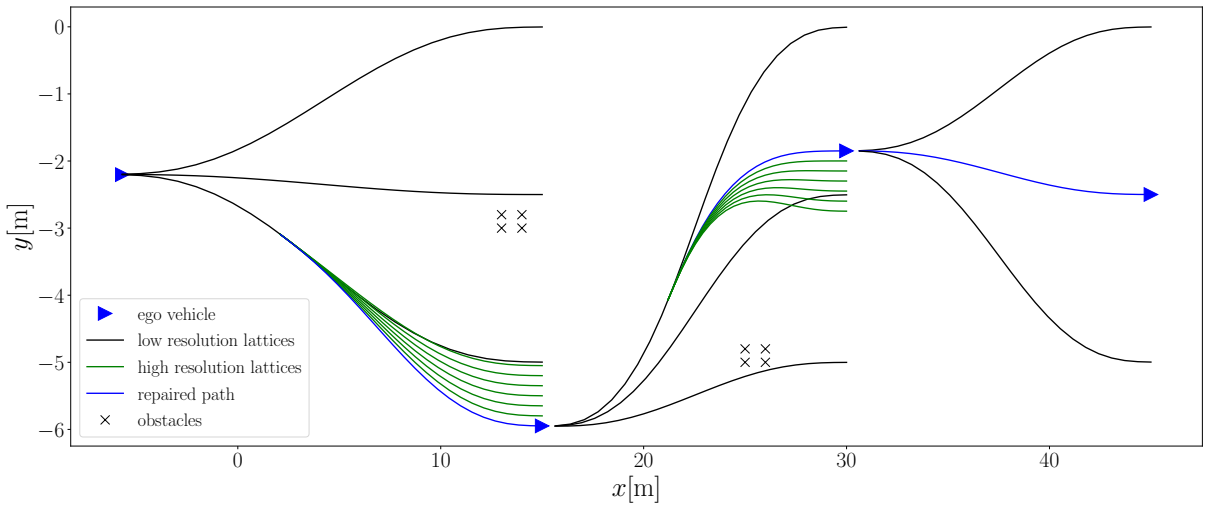
Source: Elaborated by the author.

Figure 25 – Simulated obstacle avoidance scenario in Webots. The animation of the evaluation can be found at <https://youtu.be/U8A2D_7NWbY>



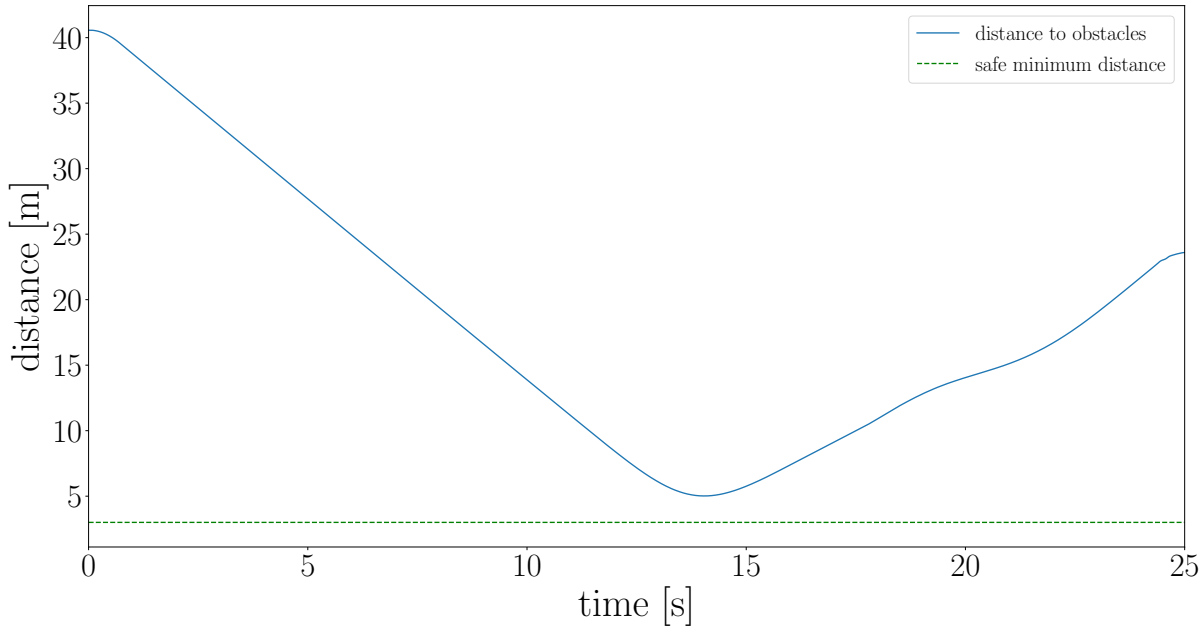
Source: Elaborated by the author.

Figure 26 – Trajectory deformation for static obstacle.



Source: Elaborated by the author.

Figure 27 – Minimum distance to static obstacles.

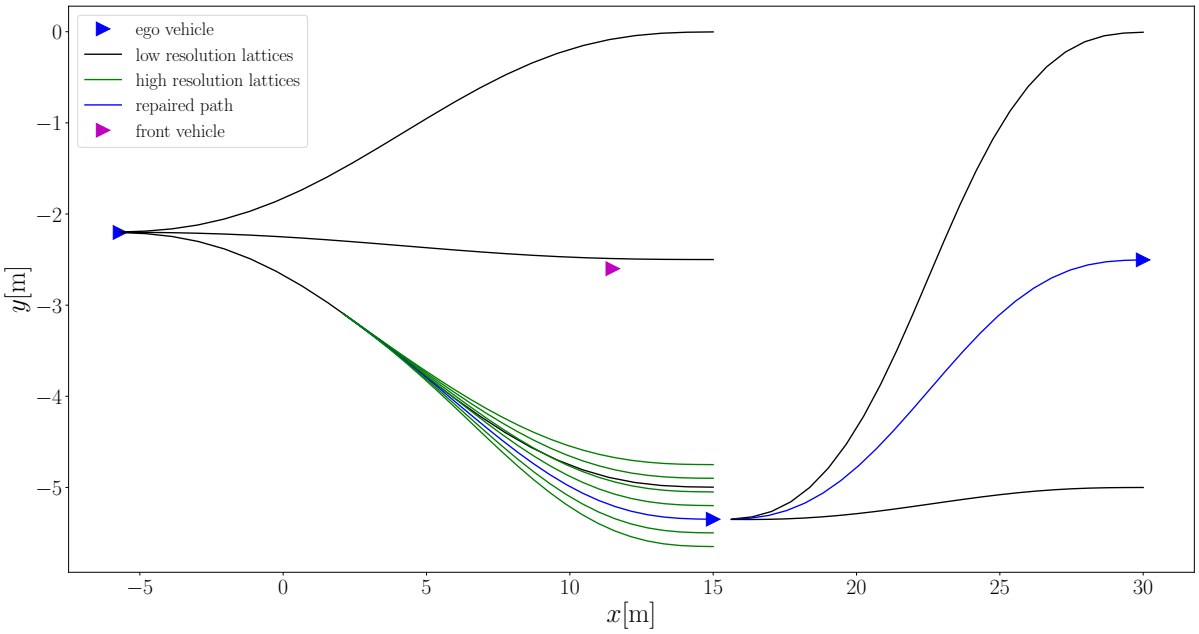


Source: Elaborated by the author.

Figure 28 – Simulated overtaking scenario in Webots. The animation of the evaluation can be found at <https://youtu.be/aaia83MUewk>

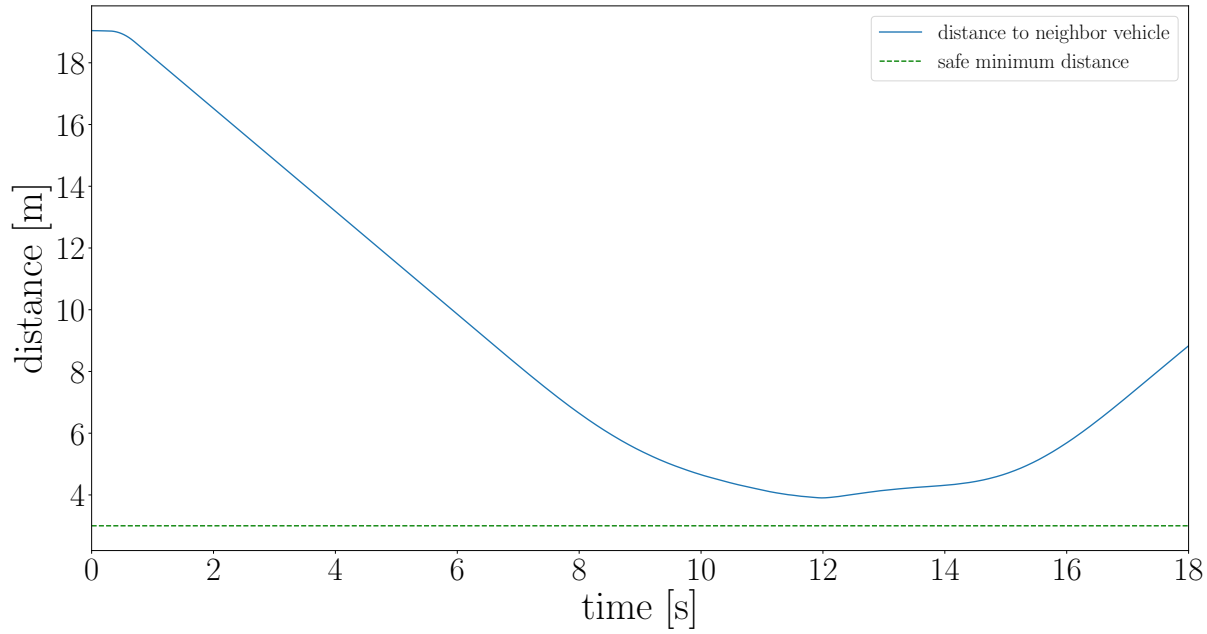
Source: Elaborated by the author.

Figure 29 – Trajectory deformation for moving obstacle.



Source: Elaborated by the author.

Figure 30 – Minimum distance to neighbor vehicle during evasive maneuver.



Source: Elaborated by the author.

CHAPTER

6

CONCLUSIONS AND FUTURE WORKS

6.1 Conclusions

Autonomous vehicles are essential to envision the future Autonomous Mobility on Demand (AMoD) systems, which will enable the quickest and safest transportation in large scale, operating with hardware and software solutions, and the most advanced Artificial Intelligence methods as never seen in the past. Although there is enough maturity in all parts of the classical self-driving stack, their performance still can be improved in order to enable the next generation of vehicles to drive themselves with level 5 autonomy on the roads. Therefore, new ideas regarding their scalability and safety must be developed to attain public confidence in this technology. In particular, ensuring the safety of autonomous cars is a challenging computational task because they have to generate collision-free feasible paths towards their goal, and ensure safety maneuvers while avoiding obstacles at the same time.

This project presented the development of a trajectory deformation algorithm to modify initial paths instead of re-calculating them from scratch, which is very convenient to reduce the computation load and insert constraints related to safety requirements. The proposed approach is also useful to avoid extremely conservative behaviors and excessive oscillations while avoiding obstacles on the road. Signal Temporal Logic was used to encode safety specifications. STL was particularly well-suited to model constraints for hybrid systems such as autonomous cars that involve continuous dynamics (e.g. vehicle states, trajectories) as well as discrete dynamics (e.g. phases of a traffic light). In addition, the robustness semantics of STL provides natural monitoring capabilities for safety performance.

The main contribution of this work is an algorithm to solve the trajectory deformation problem for autonomous vehicles. Initially, candidate path are generated using a conventional lattice planner, then those paths are evaluated with STL formulas representing safety specifications that constrain the motion of the vehicle while performing evasive maneuvers. The STL robustness

value was introduced into the planner cost function to ensure the selection of the safest path. Different resolutions are available to generate lattices, and the deformation process is activated only when it is truly required, otherwise the initial path is considered safe enough to make progress on the lane. Moreover, a weighted function was designed to classify the lattices into different categories according to the risk level associated to each candidate path, which allows selecting the level of risk involve in a maneuver and avoid extremely conservative behaviors that could take the vehicle too far from obstacles. Finally, simulations involving collision avoidance in traffic scenarios were created with the automobile tools of the robot simulator Webots to evaluate the performance of the proposed algorithm.

The results show the efficiency and usefulness of the trajectory repairing approach designed in this work. It was possible to deform initial paths while considering STL formulas related to safety specifications imposed by the user. With our approach, the computation load of re-planning a whole trajectory to handle complex traffic situations is avoided, instead our algorithm deforms an available path according to user preferences related to safety, and proceeds with the deformation in some cases based on the evaluation of the STL robustness value that provides a notion of how dangerous could be for the autonomous vehicle to follow the selected path.

6.2 Future Works

The outcomes of this research project are promising and offer the possibility to carry out further studies related to trajectory deformation for autonomous vehicles. For example, a future task is to analyze the performance of the trajectory deformation algorithm taking into consideration different scenarios for evaluation with a focus on traffic situations with various dynamic obstacles such as pedestrians or groups of vehicles at intersections. The proposed algorithm considers the planning problem of the ego vehicle, so an interesting future direction of research is to develop a trajectory repairing framework for multiple interacting agents. Furthermore, it can be interesting to extend the evaluation of the proposed algorithm considering different values for the velocity and acceleration of the ego vehicle.

More simulations can be executed in order to analyse multiple traffic scenarios with other vehicle's configuration considering different velocities and distance for obstacle detection, and taking into account the limit imposed by the road to generate the trajectories in the maneuvering and driving area. Finally, it is also important to consider cases in which there is no viable solution to avoid accidents (LI *et al.*, 2022), thus combining STL rules with trajectory deformation methods to deal with critical situations is a possible improvement to include in future research works.

BIBLIOGRAPHY

ALTHOFF, M.; KOSCHI, M.; MANZINGER, S. Commonroad: Composable benchmarks for motion planning on roads. In: IEEE. **2017 IEEE Intelligent Vehicles Symposium (IV)**. [S.l.], 2017. p. 719–726. Citations on pages 25, 26, 45, 46, 47, and 73.

ARECHIGA, N. Specifying safety of autonomous vehicles in signal temporal logic. In: IEEE. **2019 IEEE Intelligent Vehicles Symposium (IV)**. [S.l.], 2019. p. 58–63. Citations on pages 35, 36, and 40.

BAIER, C.; KATOEN, J.-P. **Principles of model checking**. [S.l.]: MIT press, 2008. Citation on page 24.

BARBOSA, F. S.; DUBERG, D.; JENSFELT, P.; TUMOVA, J. Guiding autonomous exploration with signal temporal logic. **IEEE Robotics and Automation Letters**, IEEE, v. 4, n. 4, p. 3332–3339, 2019. Citations on pages 24, 41, 57, 58, 59, 61, and 62.

BJÖRNSSON, Y.; BULITKO, V.; STURTEVANT, N. R. Tba*: Time-bounded a*. In: CITESEER. **IJCAI**. [S.l.], 2009. p. 431–436. Citation on page 34.

BULLO, F.; SMITH, S. L. **Lectures on Robotic Planning and Kinematics**. [S.l.: s.n.], 2020. Citation on page 29.

CLAUSSMANN, L.; REVILLOUD, M.; GRUYER, D.; GLASER, S. A review of motion planning for highway autonomous driving. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, v. 21, n. 5, p. 1826–1848, 2019. Citations on pages 30 and 34.

DONZÉ, A.; MALER, O. Robust satisfaction of temporal logic over real-valued signals. In: SPRINGER. **International Conference on Formal Modeling and Analysis of Timed Systems**. [S.l.], 2010. p. 92–106. Citations on pages 35, 36, and 60.

FAINEKOS, G. E.; PAPPAS, G. J. Robustness of temporal logic specifications for continuous-time signals. **Theoretical Computer Science**, Elsevier, v. 410, n. 42, p. 4262–4291, 2009. Citation on page 35.

FERGUSON, D.; HOWARD, T. M.; LIKHACHEV, M. Motion planning in urban environments. **Journal of Field Robotics**, Wiley Online Library, v. 25, n. 11-12, p. 939–960, 2008. Citations on pages 42 and 44.

FERGUSON, D.; KALRA, N.; STENTZ, A. Replanning with rrts. In: IEEE. **Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.** [S.l.], 2006. p. 1243–1248. Citation on page 39.

FURGALE, P.; SCHWESINGER, U.; RUFLI, M.; DERENDARZ, W.; GRIMMETT, H.; MÜHLFELLNER, P.; WONNEBERGER, S.; TIMPNER, J.; ROTTMANN, S.; LI, B. *et al.* Toward automated driving in cities using close-to-market sensors: An overview of the v-charge project. In: IEEE. **2013 IEEE Intelligent Vehicles Symposium (IV)**. [S.l.], 2013. p. 809–816. Citations on pages 43 and 44.

GONZÁLEZ, D.; PÉREZ, J.; MILANÉS, V.; NASHASHIBI, F. A review of motion planning techniques for automated vehicles. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, v. 17, n. 4, p. 1135–1145, 2015. Citations on pages 31, 41, and 42.

GU, T.; SNIDER, J.; DOLAN, J. M.; LEE, J.-w. Focused trajectory planning for autonomous on-road driving. In: IEEE. **2013 IEEE Intelligent Vehicles Symposium (IV)**. [S.l.], 2013. p. 547–552. Citations on pages 43 and 44.

HEKMATNEJAD, M.; YAGHOUBI, S.; DOKHANCHI, A.; AMOR, H. B.; SHRIVASTAVA, A.; KARAM, L.; FAINEKOS, G. Encoding and monitoring responsibility sensitive safety rules for automated vehicles in signal temporal logic. In: **Proceedings of the 17th ACM-IEEE International Conference on Formal Methods and Models for System Design**. [S.l.: s.n.], 2019. p. 1–11. Citation on page 41.

HOWARD, T. M. **Adaptive model-predictive motion planning for navigation in complex environments**. Phd Thesis (PhD Thesis) — Carnegie Mellon University, 2009. Citation on page 51.

HOWARD, T. M.; GREEN, C. J.; KELLY, A.; FERGUSON, D. State space sampling of feasible motions for high-performance mobile robot navigation in complex environments. **Journal of Field Robotics**, Wiley Online Library, v. 25, n. 6-7, p. 325–345, 2008. Citations on pages 42 and 44.

IACO, R. D.; SMITH, S. L.; CZARNECKI, K. Learning a lattice planner control set for autonomous vehicles. In: IEEE. **2019 IEEE Intelligent Vehicles Symposium (IV)**. [S.l.], 2019. p. 549–556. Citations on pages 43 and 44.

KARLSSON, J.; BARBOSA, F. S.; TUMOVA, J. Sampling-based motion planning with temporal logic missions and spatial preferences. **IFAC-PapersOnLine**, Elsevier, v. 53, n. 2, p. 15537–15543, 2020. Citations on pages 57, 58, 59, 60, 61, 62, and 63.

KARLSSON, J.; WAVEREN, S. van; PEK, C.; TORRE, I.; LEITE, I.; TUMOVA, J. Encoding human driving styles in motion planning for autonomous vehicles. In: **ICRA International Conference on Robotics and Automation**. [S.l.: s.n.], 2021. Citations on pages 34, 35, 41, 60, 61, and 62.

KELLY, A.; NAGY, B. Reactive nonholonomic trajectory generation via parametric optimal control. **The International Journal of Robotics Research**, SAGE Publications, v. 22, n. 7-8, p. 583–601, 2003. Citations on pages 51 and 53.

KELLY, J. **Path Planning Optimization**. 2018. University Lecture Slides. Citations on pages 52, 53, and 54.

KOENIG, S.; LIKHACHEV, M. D*lite. In: **Eighteenth National Conference on Artificial Intelligence**. USA: American Association for Artificial Intelligence, 2002. p. 476–483. ISBN 0262511290. Citation on page 34.

KUSHLEYEV, A.; LIKHACHEV, M. Time-bounded lattice for efficient planning in dynamic environments. In: IEEE. **2009 IEEE International Conference on Robotics and Automation**. [S.l.], 2009. p. 1662–1668. Citations on pages 42 and 44.

KUWATA, Y.; TEO, J.; FIORE, G.; KARAMAN, S.; FRAZZOLI, E.; HOW, J. P. Real-time motion planning with applications to autonomous urban driving. **IEEE Transactions on Control Systems Technology**, v. 17, n. 5, p. 1105–1118, 2009. Citations on pages 64 and 65.

KUWATA, Y.; TEO, J.; KARAMAN, S.; FIORE, G.; FRAZZOLI, E.; HOW, J. Motion planning in complex environments using closed-loop prediction. In: **AIAA Guidance, Navigation and Control Conference and Exhibit**. [S.l.: s.n.], 2008. p. 7166. Citation on page 65.

LAVALLE, S. M. **Planning algorithms**. [S.l.]: Cambridge university press, 2006. Citation on page 29.

LAVALLE, S. M. *et al.* Rapidly-exploring random trees: A new tool for path planning. Ames, IA, USA, 1998. Citation on page 34.

LEUNG, K.; ARÉCHIGA, N.; PAVONE, M. Back-propagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods. **arXiv preprint arXiv:2008.00097**, 2020. Citation on page 35.

LI, D.; ZHANG, J.; XIAO, B.; TANG, B.; HU, Z. Vehicle crash mitigation strategy in unavoidable collision scenarios: focusing on motion planning by considering a generalized crash severity model. **Journal of the Brazilian Society of Mechanical Sciences and Engineering**, Springer, v. 44, n. 12, p. 581, 2022. Citation on page 86.

LIKHACHEV, M.; FERGUSON, D. Planning long dynamically feasible maneuvers for autonomous vehicles. **The International Journal of Robotics Research**, Sage Publications Sage UK: London, England, v. 28, n. 8, p. 933–945, 2009. Citations on pages 42 and 44.

LIKHACHEV, M.; GORDON, G.; THRUN, S. Ara*: Anytime a* with provable bounds on sub-optimality. In: **Proceedings of the 16th International Conference on Neural Information Processing Systems**. Cambridge, MA, USA: MIT Press, 2003. (NIPS'03), p. 767–774. Citation on page 34.

LIN, Y.; MAIERHOFER, S.; ALTHOFF, M. Sampling-based trajectory repairing for autonomous vehicles. In: IEEE. **2021 IEEE International Intelligent Transportation Systems Conference (ITSC)**. [S.l.], 2021. p. 572–579. Citations on pages 24, 25, 39, 40, 45, 46, and 64.

MAIERHOFER, S.; RETTINGER, A.-K.; MAYER, E. C.; ALTHOFF, M. Formalization of interstate traffic rules in temporal logic. In: IEEE. **2020 IEEE Intelligent Vehicles Symposium (IV)**. [S.l.], 2020. p. 752–759. Citation on page 40.

MALER, O.; NICKOVIC, D. Monitoring temporal properties of continuous signals. In: **Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems**. [S.l.]: Springer, 2004. p. 152–166. Citations on pages 35 and 36.

MANGETTE, C.; TOKEKAR, P. Multi-robot coordinated planning in confined environments under kinematic constraints. In: IEEE. **2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2020. p. 7999–8004. Citations on pages 24 and 25.

MCNAUGHTON, M.; URMSON, C.; DOLAN, J. M.; LEE, J.-W. Motion planning for autonomous driving with a conformal spatiotemporal lattice. In: IEEE. **2011 IEEE International Conference on Robotics and Automation**. [S.l.], 2011. p. 4889–4895. Citations on pages 23, 32, 42, 44, 49, 65, 71, 72, 73, and 79.

MENG, Y.; WU, Y.; GU, Q.; LIU, L. A decoupled trajectory planning framework based on the integration of lattice searching and convex optimization. **IEEE Access**, IEEE, v. 7, p. 130530–130551, 2019. Citation on page 48.

MICHEL, O. Webots: Professional mobile robot simulation. **Journal of Advanced Robotics Systems**, v. 1, n. 1, p. 39–42, 2004. Available: <http://www.ars-journal.com/International-Journal-of-Advanced-Robotic-Systems/Volume-1/39-42.pdf>. Citations on pages 27 and 37.

NORVIG, P.; RUSSELL, S. **Artificial Intelligence: A modern approach**. Third. [S.l.]: Pearson Education Limited, 2016. ISBN 10: 1292153962 13: 9781292153964. Citation on page 33.

OLIVEIRA, R. **Motion planning for heavy-duty vehicles**. Phd Thesis (PhD Thesis) — KTH Royal Institute of Technology, 2019. Citations on pages 31 and 33.

PADEN, B.; ČÁP, M.; YONG, S. Z.; YERSHOV, D.; FRAZZOLI, E. A survey of motion planning and control techniques for self-driving urban vehicles. **IEEE Transactions on intelligent vehicles**, IEEE, v. 1, n. 1, p. 33–55, 2016. Citations on pages 23 and 29.

PHAM, Q.-C.; NAKAMURA, Y. A new trajectory deformation algorithm based on affine transformations. **IEEE Transactions on Robotics**, IEEE, v. 31, n. 4, p. 1054–1063, 2015. Citation on page 39.

PIVTORAIKO, M.; KELLY, A. Efficient constrained path planning via search in state lattices. In: MUNICH GERMANY. **International Symposium on Artificial Intelligence, Robotics, and Automation in Space**. [S.l.], 2005. p. 1–7. Citations on pages 41 and 44.

PIVTORAIKO, M.; KNEPPER, R. A.; KELLY, A. Differentially constrained mobile robot motion planning in state lattices. **Journal of Field Robotics**, Wiley Online Library, v. 26, n. 3, p. 308–333, 2009. Citations on pages 32 and 49.

PLAKU, E.; KARAMAN, S. Motion planning with temporal-logic specifications: Progress and challenges. **AI communications**, IOS Press, v. 29, n. 1, p. 151–162, 2016. Citations on pages 24 and 40.

RÖSMANN, C.; FEITEN, W.; WÖSCH, T.; HOFFMANN, F.; BERTRAM, T. Trajectory modification considering dynamic constraints of autonomous robots. In: VDE. **ROBOTIK 2012; 7th German Conference on Robotics**. [S.l.], 2012. p. 1–6. Citation on page 40.

SADAT, A.; REN, M.; POKROVSKY, A.; LIN, Y.-C.; YUMER, E.; URTASUN, R. Jointly learnable behavior and trajectory planning for self-driving vehicles. **arXiv preprint arXiv:1910.04586**, 2019. Citation on page 23.

SINGH, A. **Platoon Coordination under Signal Temporal Logic Specifications**. 2020. Citation on page 24.

SUN, K.; SCHLOTFELDT, B.; CHAVES, S.; MARTIN, P.; MANDHYAN, G.; KUMAR, V. Feedback enhanced motion planning for autonomous vehicles. **arXiv preprint arXiv:2007.05794**, 2020. Citations on pages 23, 29, 34, 43, 44, 60, 71, 72, and 73.

SUN, Y.; REN, D.; LIAN, S.; FAN, M.; TENG, X. An efficient generation method based on dynamic curvature of the reference curve for robust trajectory planning. **arXiv preprint arXiv:2012.14617**, 2020. Citation on page 48.

TABOADA, S. **Multi-Agent Motion Planning with Signal Temporal Logic Constraints**. 2020. Citations on pages 23, 24, 34, 35, 41, 58, and 62.

WASLANDER, S. **Conformal Lattice Planner**. 2018. University Lecture Slides. Citations on pages 47, 48, 49, 50, 52, 55, 56, 57, 72, and 73.

WEBOTS. **http://www.cyberbotics.com**. 2022. Open-source Mobile Robot Simulation Software. Available: <http://www.cyberbotics.com>. Citations on pages 37 and 38.

WERLING, M.; KAMMEL, S.; ZIEGLER, J.; GRÖLL, L. Optimal trajectories for time-critical street scenarios using discretized terminal manifolds. **The International Journal of Robotics Research**, SAGE Publications Sage UK: London, England, v. 31, n. 3, p. 346–359, 2012. Citations on pages 43 and 44.

ZIEGLER, J.; BENDER, P.; DANG, T.; STILLER, C. Trajectory planning for bertha—a local, continuous method. In: IEEE. **2014 IEEE intelligent vehicles symposium proceedings**. [S.l.], 2014. p. 450–457. Citations on pages 25 and 40.

ZIEGLER, J.; STILLER, C. Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios. In: IEEE. **2009 IEEE/RSJ International Conference on Intelligent Robots and Systems**. [S.l.], 2009. p. 1879–1884. Citations on pages 42 and 44.

APPENDIX

A

PUBLICATIONS

JUSTO, V. S. ; OSÓRIO, F. S. . A Trajectory Deformation Algorithm for Intelligent Vehicles. In: 2022 Latin American Robotics Symposium (LARS), 2022 Brazilian Symposium on Robotics (SBR), and 2022 Workshop on Robotics in Education (WRE), São Bernardo do Campo, Brazil, 2022, pp. 115-120, doi: 10.1109/LARS/SBR/WRE56824.2022.9995981.


ICMC
USP
SÃO CARLOS