TECHNICAL UNIVERSITY OF MUNICH

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

Master's Thesis in Informatics

Assessing Robustness of Driving Corridors According to Temporal Logic Specifications

Evald Nexhipi, B.Sc.

TECHNICAL UNIVERSITY OF MUNICH

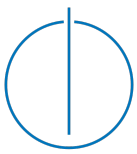
SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

Master's Thesis in Informatics

Assessing Robustness of Driving Corridors According to Temporal Logic Specifications

Bewertung der Robustheit von Fahrkorridoren anhand von Spezifikationen in temporaler Logik

Author:	Evald Nexhipi, B.Sc.
Supervisor:	Prof. Dr.-Ing. Matthias Althoff
Advisor:	Gerald Würsching, M.Sc.
Submission Date:	15.10.2023



I confirm that this master's thesis is my own work and I have documented all sources and material used.

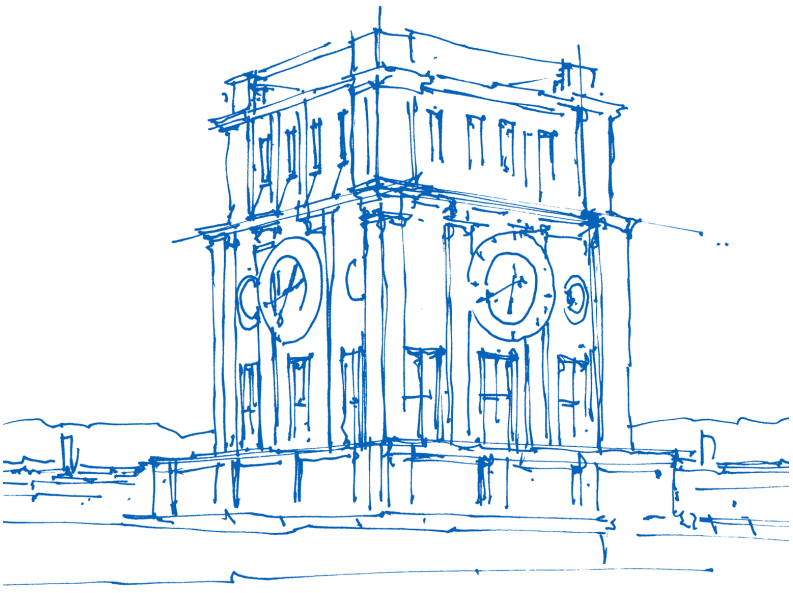
Ich versichere, dass ich diese Masterarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Munich, 15.10.2023 Evald Nexhipi, B.Sc.

Acknowledgments

I am deeply thankful to my advisor Gerald Würsching for the continuous guidance and support in this thesis in many technical and academic matters. Without his feedback and willingness to answer my countless questions and assist with technical difficulties, the compilation of this thesis would not have been possible. I also want to express my gratitude to Prof. Matthias Althoff for providing me with the invaluable opportunity to participate in the research-related projects offered within his group.

I would like to thank my family for their love and support throughout my studies. I am grateful to my mother for being an example of motivation and determination and to my father for inspiring me to overcome any challenge, regardless of the difficulty. A big thank you goes to my precious sister for her love and wonderful sense of humor.



Abstract

This thesis tackles the problem of assessing the robustness of set-based driving corridors according to Signal Temporal Logic (STL) specifications. Since driving corridors can represent specific maneuvers, a quantitative robustness degree can serve as a measure to rank different maneuvers prior to planning. To this end, we propose novel robustness measurements to quantify the degree of adherence to a given specification. We first develop an algorithmic framework for representing STL formalisms within the domain in which driving corridors are defined. This geometric representation, coupled with the proposed robustness measurements, captures intricate geometric relationships between the reachable states of the system and the representation of a specification. In contrast to previous methods that simplify geometries by representing a specification or the reachable sets as single points, or disregarding their full geometric intricacies, our robustness measurements consider the complete geometry of the reachable sets and the representation of the specification. Within this work, three measures are investigated: the first assesses geometric displacement, the second introduces a weighted perimeter calculation based on the distances between geometries, and the third refines this calculation to consider only lines relevant to specification adherence or violation.

Our numerical experiments on Commonroad scenarios demonstrate that the proposed robustness measures preserve soundness and monotonicity. To combine robustness scores along longitudinal and lateral dimensions across all driving corridors, we formulate a joint optimization problem for ranking corridors. We further evaluate the computational efficiency of our approach by varying the considered time horizon and complexity of the specifications.

Contents

1. Introduction

Navigation through dynamic and unpredictable traffic scenarios is a paramount challenge in the context of autonomous driving. The capacity to plan and execute safe trajectories can often rely on evaluating and establishing driving corridors that strictly conform to challenging spatiotemporal limitations. Set-based reachability analysis has therefore emerged as a promising approach [1], allowing for the exploration of collision-free drivable areas and the extraction of driving corridors. These corridors provide spatial and temporal constraints for various motion planning algorithms [2, 3], which, as particularly shown in [3], have exhibited remarkable computational efficiency with the rising complexity of traffic situations.

Recent works [4] have extended the applicability of reachability analysis to incorporate temporal logic specifications. This extension enables the extraction of driving corridors that ensure collision-free paths and satisfy particular specifications. However, a notable limitation of this method is the inability to quantitatively evaluate different driving corridors according to their degree of satisfaction or violation of a given specification.

In typical (complex) traffic scenarios, multiple driving corridors can exist, each corresponding to a particular maneuver for the ego vehicle. The primary objective of this thesis is to devise a robustness assessment method for driving corridors concerning the satisfaction or violation of temporal logic specifications. To this end, we propose novel robustness measurements that not only ensure the preservation of soundness and monotonicity but also effectively describe the adherence profile of driving corridors to a specification.

We utilize the *CommonRoad-Reach* toolbox [5] to compute the reachable sets of the ego vehicle. We first evaluate our approach on simple non-temporal specifications and advance to more complex temporal formalisms. Numerical experiments are conducted to demonstrate the robustness scores recorded across driving corridors given the aforementioned specifications. We further show the computational efficiency of our approach as we vary the formula size or time horizon. In summary, this thesis aims to contribute significantly to the assessment of driving corridors by introducing a novel approach for evaluating their adherence to temporal logic specifications. The outcome of this work will not only enhance the safety and reliability of autonomous vehicles but also advance our understanding of their ability to navigate complex traffic scenarios.

Within this chapter, we present the problem formulation in Section 1.1. The related work is introduced in Section 1.2. In Section 1.3, we list the contributions of this thesis. The thesis outline is described in Section 1.4.

1.1. Problem Formulation

This thesis concentrates on the problem of assessing the robustness of driving corridors according to specifications expressed in Signal Temporal Logic (STL). The robustness measurement has to obey the property of *soundness* and *monotonicity*. The first property ensures that a positive robustness corresponds to the satisfaction of the specification, and conversely, a negative robustness should reflect the violation of the specification. The second property guarantees that an increase in the robustness measurement is aligned with a more substantial satisfaction degree. In contrast, a decrease in the robustness score mirrors a stronger violation to the specification.

The robustness assessment should align with the nature of the driving corridors, encompassing many *collision-free* and *dynamically-feasible* traces. Although many works can prove the satisfaction of a specification in a set of traces, to the best of our knowledge, they lack the robustness assessment. Moreover, the robustness measurement should provide a holistic view, meaning that it should simultaneously consider multiple state attributes of the corridors, e.g., position and velocity. Therefore, this approach should capture the interdependencies between variables, in contrast to the independent evaluation of signals.

1.2. Related Work

1.2.1. Obtainment of a specification-compliant trajectory

The research pertaining to acquiring a trajectory that adheres to the specified requirements can be effectively categorized into three main classes. The first class produces discrete plans to guide the trajectory planning problem, whereas the high-level planner generates discrete plans for the low-level planner. Such applications, also known as synergetic approaches, as used in [6, 7, 8, 9, 10], are very popular applications capturing the principle of long-term routing and short-term motion planning. These approaches are mostly based on three steps:

1. Construction of a discrete abstraction for the system.
2. High-level planning for the abstraction layer using the specification and the exploration information from the low-level planner.
3. Low-level planning using the physical model and the suggested high-level plans.

An example, motivated by the work in [7] is illustrated also in Fig. 1.1. This synergy helps to systematically convey information regarding the physics of the problem from the low-level layer to the high-level layer. The constraints related to temporal formalisms, typically higher-level specifications, are systematically communicated from the high-level layer to the low-level layer using synergy. This implies that the system is designed so that the information about temporal constraints is effectively passed down to the

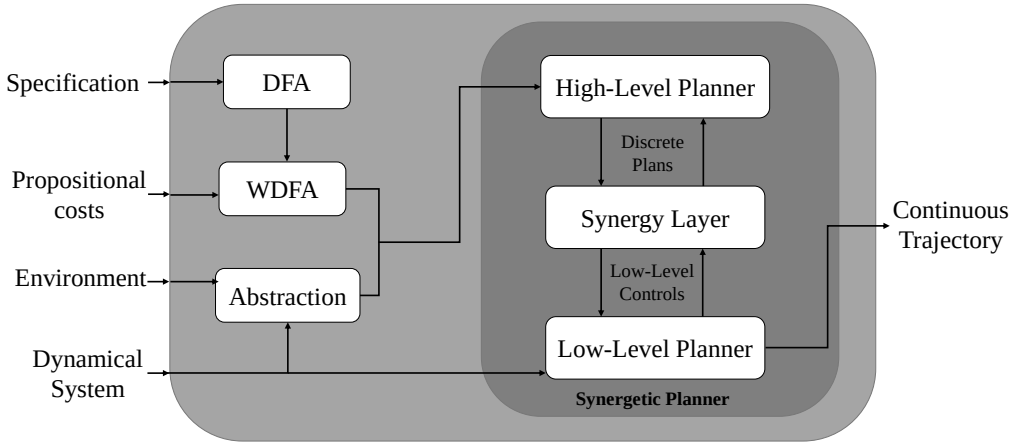


Figure 1.1.: A block diagram representation of a synergetic approach [7].

lower-level control mechanisms, ensuring that the system operates in accordance with these temporal requirements.

The second class of the related work evaluates the specifications on planned trajectories using monitors [11, 12]. Although the monitoring of the planned trajectories can be realized efficiently, even in heavy traffic scenarios as discussed in [11], it can be impossible to return a substitute trajectory in case the trajectory (under examination) is rejected. The principle of the works in [11] and [12] can be best summarized in Fig. 1.2. A set of candidate trajectories are given as input to a monitor, together with one or a set of specifications. The monitor then generates an output indicating the acceptance or rejection of the trajectories. However, as noted earlier, these approaches lack the generation of alternative trajectories in rejection cases.

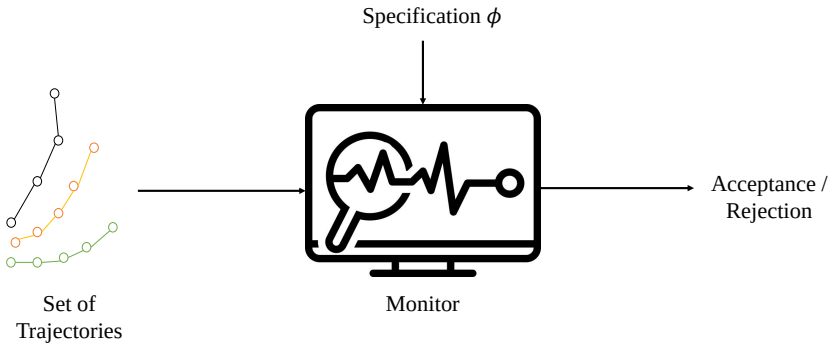


Figure 1.2.: A diagram representation of monitoring approaches.

The third class of the related work considers the evaluation of a specification in a semantic state space and the subsequent generation of maneuvers. In [13], the continuous environment is discretized, and the specifications or rules are evaluated on a high abstract level by traversing a graph representation in a semantic state space.

Similar to this work, in [14], a semantic scene representation for modeling traffic scenes is presented. The semantic space is derived from considering potential relations of the ego vehicle, and maneuvers are generated through transitions that model state changes. In [13], the specifications are ensured to hold on the lower continuous level due to the homotopy property of each trace, thus allowing for the generation of maneuvers from simple rules.

The work presented in [15] leverages a data-driven vehicle dynamics model and optimization-based maneuver planning to generate a safe, collision-free trajectory with dynamic lane changes. In [16], the problem of exhaustive exploration of all possible driving paths is avoided, by using a set-based reachability analysis to explore the collision-free drivable area. This work is further extended in [4], by incorporating specifications based on works of [17], [18], as well as hand-crafted rules, to generate specification-compliant reachable sets.

Although the approach presented in [4] generates reachable sets compliant with a given specification, it does not provide alternative trajectories when a specification is not feasible. The predicates are only position-related and do not consider velocity. Even though the recent work in [19] introduced velocity-related predicates, it still fails to compute trajectories when a specification can not be satisfied. It is the focus of our work to also further extend this approach, by considering driving corridors and maneuvers that best satisfy a given specification. Furthermore, we incorporate velocity- and position-based predicates to determine a specification and return a driving corridor or maneuver, even when a specification is not feasible.

1.2.2. Obtaining a minimum-violating trajectory

An important aspect to be considered in the related work is the generation of trajectories that minimally violate or maximally satisfy a given specification. The work can be categorized into four classes:

- Automata-based approaches.
- Optimization approaches based on CBFs.
- Machine Learning and Reinforcement Learning methodologies.
- Game-theory approaches.

The first class corresponds to automata-based approaches. The outline is described in Fig. 1.3. Potential behaviors of the system, together with underlying assumptions about the environment are captured by a transition system (TS). Specifications or requirements are modeled using formal languages. A product graph between the TS and the automaton, representing the specification is constructed and used to either verify the satisfaction of a trajectory or to generate trajectories that are both specification-compliant and short paths in the product graph.

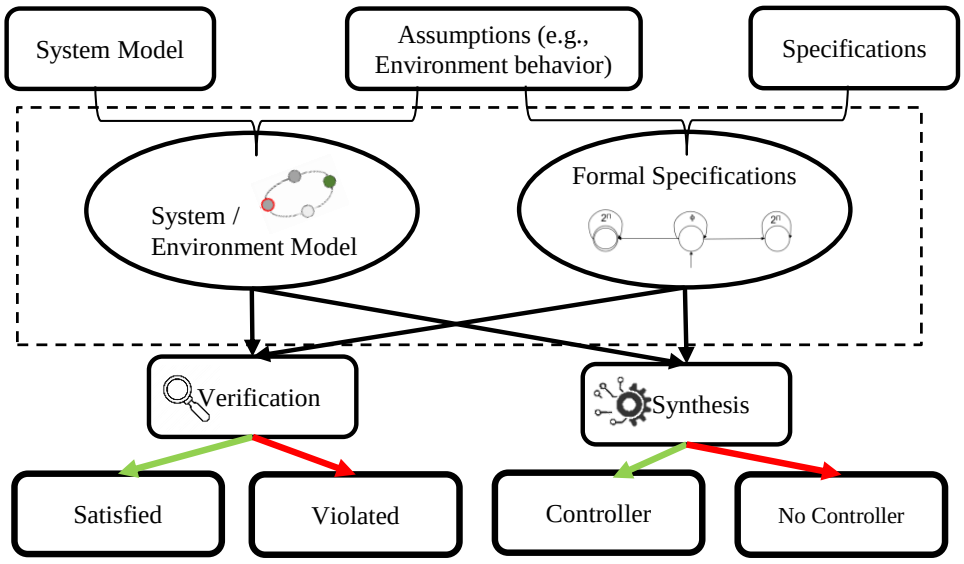


Figure 1.3.: Usage of formal methods for verification and synthesis.

The automata-based approaches can be best categorized into 3 subclasses, as shown in Fig. 1.4. The first subclass corresponds to incremental sampling-based methods. The works in [8, 9, 10, 20, 21, 22, 23, 24], define a cost function according to a preferred parameter (distance, time, driver's comfort, etc). The function is then incorporated into a planner to produce trajectories that minimize the cost. The second subclass [6, 7, 25, 26, 27, 28], produces an explicit product between a TS and an automaton, that can either reject or accept a trace, found via graph search. The third subclass performs an incremental sampling-based product between the TS and the automaton [6, 29, 30] . The product graph is then incorporated into the planner to serve high-level discrete plans.

The second class consists of optimization-related works based on Control Barrier Functions (CBFs) [31, 32, 33, 34]. The idea is to guide the agent to a desired state, considering some violation or safety margin (Fig. 1.5). It realizes this principle by putting constraints on the controls of the agent. An obvious advantage of these works is the non-discretization of the system dynamics or the environment, in contrast to automata-based approaches. However, these approaches come together with many limitations, such as the complexity of implementing CBFs for nonlinear or high-dimensional state spaces. Moreover, integrating high-level objectives, e.g., mission-specific tasks, into the motion planning framework can impose a significant challenge.

The third class is based on Machine Learning (ML) and Reinforcement Learning (RL) techniques to estimate the satisfaction or violation degree of a plan, or trajectory. The uncertainty and complexity of autonomous driving make RL and especially Deep Reinforcement Learning (DRL) appealing to use. The latter can be leveraged to optimize an expected reward as a result of the interaction with the environment. Reward shaping for DRL is a challenging task in autonomous vehicles and provides no guarantee as

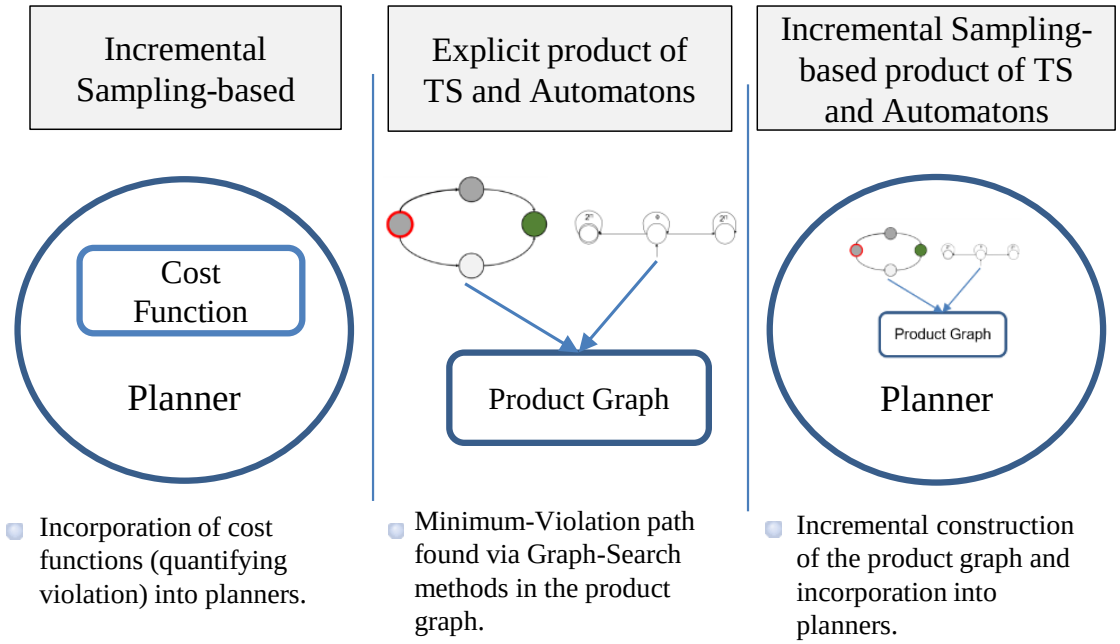


Figure 1.4.: Automata-based approaches.

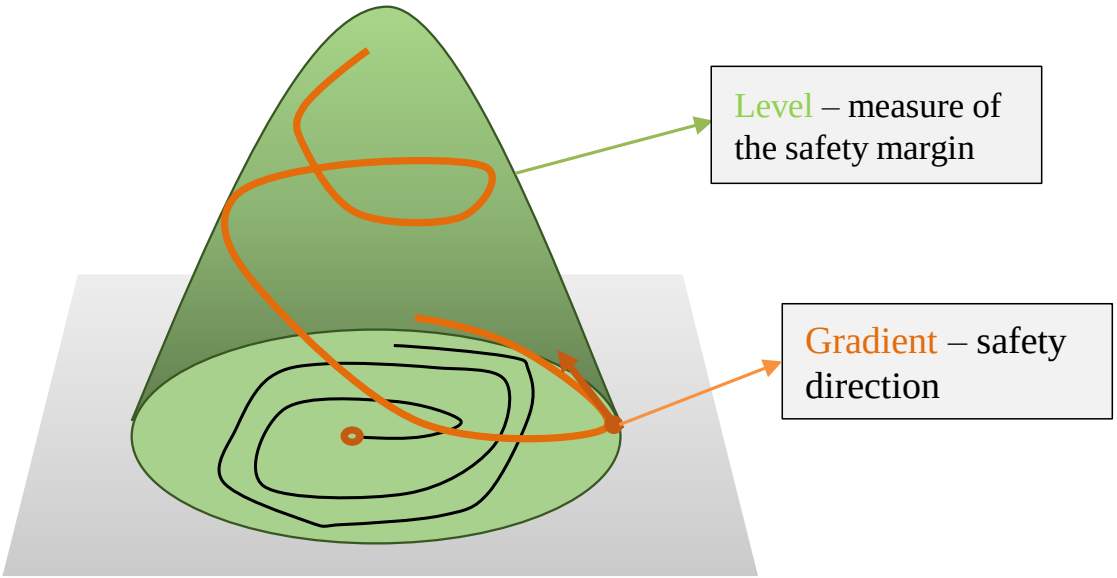


Figure 1.5.: Optimization approaches based on CBFs.

discussed in [22]. These methods are usually incorporated as high-level planners in hierarchical approaches. They can suggest an estimation for the level of satisfaction or violation of a plan, or trajectory and can be incorporated during runtime to guide the planning [35].

The fourth class consists of game-theory approaches. Specifications are organized in a hierarchical graph structure known as Rulebook [36] . Sequential games are constructed in [27], [37], which aim to find a control strategy for the agent that is optimal given that the environment plays its best response in getting the agent to violate its rulebook. However, this setting with the focus on the agent, does not guarantee a global optimal solution. Instead, multi-agent games are proposed [38], [39], by addressing the problem of optimally satisfying all participating agents.

1.2.3. Quantifying robustness

In motion planning, the focus often lies on computing trajectories that maximize the satisfaction of a given specification or minimize its violations. It is essential to quantify the degree of satisfaction or violation, respectively, in order to assess the extent to which a trajectory satisfies or violates the specification. Recent research has witnessed the development of various techniques to tackle monitoring problems, with formal methods gaining significant attention. Among these formal methods, temporal logics and their subbranches, such as Linear Temporal Logic (LTL), Computation Tree Logic (CTL), Metric Temporal Logic (MTL) [40], and Signal Temporal Logic (STL) [41], [42] offer a precise way to describe system properties over time and enable specification-based monitoring [43].

In general, LTL, CTL, and MTL are well-suited for interpreting qualitative satisfaction of specified requirements. On the other hand, with its quantitative semantics, STL can measure how strongly a set of requirements can be satisfied or violated. One significant limitation of the traditional robustness measure for STL is that it is defined based on the most critical point, which presents the highest satisfaction or violation peak. To address this issue, several works have focused on enhancing the robustness of STL. In [44], the authors propose an arithmetic-geometric mean robustness, which calculates an average-based robustness score for STL by averaging the scores of different specifications or subformulas at various time steps. This approach highlights the frequency of satisfaction and the robustness of each specification at each time point. Additionally, the robustness is normalized, enabling meaningful comparisons when dealing with specifications involving requirements over signals with different scales. Consequently, the soundness property of STL is preserved while achieving higher robustness against noise compared to the traditional metric.

The traditional evaluation of robustness for STL faces another challenge known as the *shadowing effect* [45]. This effect occurs when an increase in the robustness of conjoined propositions is not mirrored in the robustness of the conjunction as a whole. To address this issue, an accumulative robustness measurement is proposed in [46]. This approach utilizes the Riemann sum operator to accumulate positive and negative robustness

degrees over time, thereby emphasizing robustness for all sub-formulas.

In [47], the authors have established user preferences by utilizing a weighted STL. This weighted robustness is employed to differentiate signals based on a desired formula that contains subformulas with varying levels of importance or priorities, along with time preferences. This approach proves valuable in scenarios involving conflicting tasks, where complete satisfaction of all tasks may not be feasible.

Reachability analysis has gained considerable popularity as a method to verify whether a STL specification or requirement is satisfied or not. Recent works, such as [48] and [49], have embraced reachability analysis techniques to facilitate the verification of sets of trajectories. However, a limitation of these approaches is that they do not offer a quantification for the violation or satisfaction of the specified requirements. While they determine the feasibility of satisfying the STL specification by exploring the reachable set of trajectories, they do not provide a precise measure of how well the specification is met or violated in practice.

1.3. Contributions

The main contribution of this thesis is the development of novel robustness measurements to quantify the adherence of driving corridors to a temporal specification. Since driving corridors can represent specific maneuvers, measuring the satisfaction or violation degree can provide a ranking of these maneuvers. Additionally, this ranking facilitates informed decision-making in autonomous vehicle control, thus contributing to the advancement of safe and efficient autonomous driving. The rest of the contributions in this thesis are listed as follows:

1. Formulation of an algorithmic framework for representing STL specifications within the domains of reachable sets.
2. Maintenance of a minimal space of alternatives to satisfy an expression.
3. Incorporation of predicates to articulate spatiotemporal constraints.
4. Independent definition of robustness in lateral and longitudinal dimensions for curvilinear-based motion planners.
5. Establishment of a unified optimization problem integrating longitudinal and lateral robustness scores across multiple driving corridors.

1.4. Outline of the thesis

The rest of the thesis is organized as follows. Chapter 2 establishes an overview of the theoretical foundation used in this thesis, where especially the concept of driving corridors and STL is discussed. Chapter 3 introduces the algorithmic framework for representing STL specifications within the domains of reachable sets. Chapter

4 introduces eleven atomic propositions. In Chapter 5, we present the robustness measurements, and in Chapter 6, we evaluate the performance and efficiency of our approach. Finally, Chapter 7 concludes this thesis, where also directions for future work are outlined.

2. Preliminaries

In this section, we provide an introductory overview of the preliminary information necessary for this thesis. We first start by establishing the notation used in this work, as well as the main theoretical definitions. Building upon these foundational elements, we explore the theoretical background, which includes relevant theories and methodologies that underpin our work. Key theories such as reachability analysis and Signal Temporal Logic are discussed in the following sections.

This chapter is structured as follows: Main notations are introduced in Section 2.1. An overview of the considered scenarios is provided in Section 2.2, together with the introduction of the curvilinear coordinate system. Reachability analysis is discussed in Section 2.3, where the dynamics of the ego-vehicle are covered in Section 2.3.1, and the reachable sets are explored in Section 2.3.2. Driving corridors are introduced in Section 2.4. Lastly, Section 2.5 presents Signal Temporal Logic, along with its syntax (2.5.1), semantics (2.5.2), and robustness (2.5.3).

2.1. Notation

We will discretize time into discrete time steps denoted by the index $k \in \mathbb{N}_0$ which corresponds to a continuous time $t_k = k\Delta t$, where $\Delta t \in \mathbb{R}^+$ is a constant time increment. We will leverage the following indexes k_0 and k_f to specify the starting and ending time steps, respectively. Additionally, let us define the sets of admissible states and inputs as $\mathcal{X}_k \subset \mathbb{R}^{n_x}$ and $\mathcal{U}_k \subset \mathbb{R}^{n_u}$ for any dynamical system given by $x_{k+1} = f(x_k, u_k)$, where $x_k \in \mathcal{X}$ represents the state and $u_k \in \mathcal{U}$ denotes the input. We will use the notations $\underline{\square}$ and $\overline{\square}$ to denote the lower and upper bounds of the possible values of a variable $\square$, respectively.

2.2. CommonRoad scenarios and Curvilinear Coordinate System

2.2.1. CommonRoad scenarios

This thesis focuses on scenarios described in the CommonRoad format [50]. A CommonRoad scenario usually comprises the following elements:

1. A road network represented by lanelets [51], where the left and right boundaries are modeled using polylines.

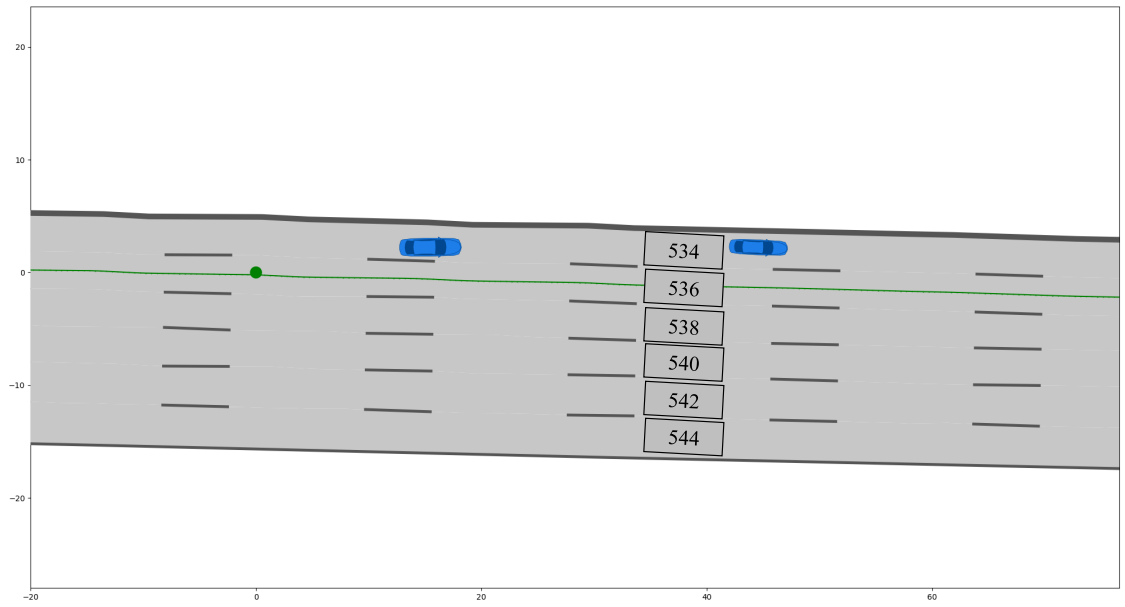


Figure 2.1.: Scenario *USA_US101-1_2_T-1*.

2. Both static and dynamic obstacles.

3. Various traffic rule components, such as traffic lights, traffic signs and road markings.

An exemplary scenario is shown in Fig. 2.1. The scenario consists of six lanelets, 2 dynamic obstacles (shown in blue) and a reference path (drawn in green) with the initial state denoted by a green circle. The ground-truth trajectories stored in the scenario are used for predicting the movement of dynamic obstacles. Additionally, when facing a planning problem, including the initial state of an ego vehicle and a set of goal states, a reference path leading to the goal lanelet is planned. This reference path also serves as the essential basis for constructing the local curvilinear coordinate system of the ego vehicle, described in Section 2.2.2.

2.2.2. Curvilinear Coordinate System

A line segment is a part of a straight line bounded by two distinct points, i.e., start and end point, that contains every point on the line in-between these two points. Furthermore, we define a polyline to be a continuous line composed of one or more connected line segments. Different approaches can be used to generate curvilinear coordinates, as shown in [52]. We leverage the model where the reference path is given as a polyline and the conversion is done using the lanelet model. A reference path $\Gamma : \mathbb{R} \rightarrow \mathbb{R}^2$ is a polyline that is commonly generated by route planners and is used to align a curvilinear coordinate system, as shown in Fig. 2.2. A projection function

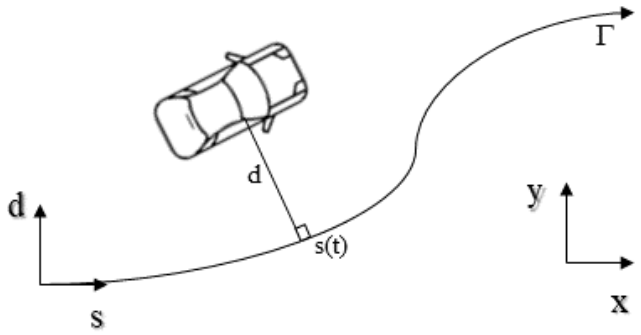


Figure 2.2.: The alignment of the curvilinear coordinate system with a reference path Γ.

$\text{proj}_{(s,d)} : \mathbb{R}^2 \to \mathbb{R}^2$ transforms a position $(x, y)^T$ (in the global Cartesian frame) to the curvilinear coordinates $(s, d)^T$. The conversion to curvilinear coordinates is realized by computing line normals and interpolating tangents between the vertices of a polyline, as outlined in [52].

2.3. Reachability Analysis

2.3.1. Vehicle dynamics

To enhance computational efficiency, a point-mass model in the curvilinear coordinate system [2] is used to simplify the vehicle dynamics for the reachable set computation. The vehicle's center is selected as the reference point, and the state vector $x = (s, v_s, d, v_d)^T$ comprises the longitudinal and lateral positions $(s, d)^T$ and velocities $(v_s, v_d)^T$, respectively [16]. The input vector $u = (a_s, a_d)^T$ containing the controls for the ego vehicle is constituted by the longitudinal and lateral accelerations, denoted by a_s and a_d. The system dynamics are expressed by the following equation:

$$x_{k+1} = \begin{pmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{pmatrix} x_k + \begin{pmatrix} \frac{1}{2}\Delta t^2 & 0 \\ \Delta t & 0 \\ 0 & \frac{1}{2}\Delta t^2 \\ 0 & \Delta t \end{pmatrix} u_k \tag{2.1}$$

To account for the kinematic feasibility, velocities and accelerations in both directions (longitudinal and lateral) are bounded by the approximation of the kinematically feasible values:

$$\begin{aligned} \underline{v_s} \leq v_{s,k} \leq \overline{v_s}, \\ \underline{v_d} \leq v_{d,k} \leq \overline{v_d}, \\ \underline{a_s} \leq a_{s,k} \leq \overline{a_s}, \\ \underline{a_d} \leq a_{d,k} \leq \overline{a_d}, \end{aligned} \tag{2.2}$$

where $v_{s,k}$ and $v_{d,k}$ represent the longitudinal and lateral velocities at time-step k, and $a_{s,k}$ along with $a_{d,k}$ are the longitudinal and lateral accelerations at time-step k. These conservative bounds are chosen to account for kinematic limitations of the real vehicle and transformation effects on the curvilinear coordinate system.

2.3.2. Reachable Sets

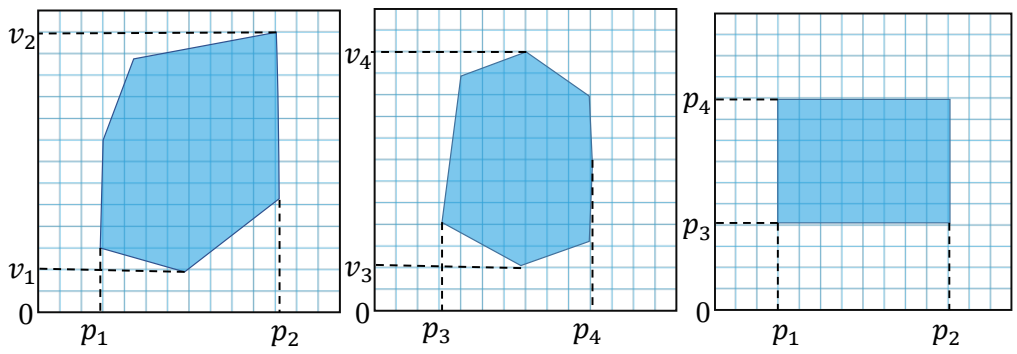
We consider the occupancy of the ego vehicle represented by $\mathcal{Q}(x_k) \subset \mathbb{R}^2$, and the occupancy of all surrounding obstacles in a given scenario by $\mathcal{O}_k \subset \mathbb{R}^2$. The set of forbidden states, $\mathcal{F}_k$, can then be defined as the collection of states in $\mathcal{X}_k$ for which the occupancy of the ego vehicle intersects with the occupancy of any surrounding obstacle, formally expressed as $\mathcal{F}_k = \{x_k \in \mathcal{X}_k | \mathcal{Q}(x_k) \cap \mathcal{O}_k \neq \emptyset\}$. Additionally, let the initial reachable set be represented by $\mathcal{R}_{k_0}$, with $\mathcal{X}_{k_0}$ being the set of possible initial states. Subsequently, the reachable set $\mathcal{R}_{k+1}$ can be described as the set of states that can be reached from the previous reachable set, $\mathcal{R}_k$, without encountering any intersections with the forbidden states $\mathcal{F}_{k+1}$, i.e.,

$$\mathcal{R}_{k+1} = \{x_{k+1} \in \mathcal{X}_{k+1} | \exists x_k \in \mathcal{R}_k, \exists u_k \in \mathcal{U}_k : x_{k+1} = f(x_k, u_k) \wedge x_{k+1} \notin \mathcal{F}_{k+1}\} \tag{2.3}$$

In this thesis, $\mathcal{R}_k$ denotes an accurate representation of the exact reachable set. The reason for utilizing an approximation is that computing the exact reachable set can be excessively costly, particularly for models with higher complexity. The set representation of the reachable set $\mathcal{R}_k$ is achieved by taking the union of base sets $\mathcal{R}_k^{(i)}$, where $i \in \mathbb{N}$. Each base set $\mathcal{R}_k^{(i)} = \mathcal{P}_{s,k}^{(i)} \times \mathcal{P}_{d,k}^{(i)}$ is formed as the Cartesian product of two convex polytopes $\mathcal{P}_{s,k}^{(i)}$ and $\mathcal{P}_{d,k}^{(i)}$, representing the reachable positions and velocities in the (s, v_s) and (d, v_d) planes (shown in Fig. 2.3a, 2.3b) . This choice of representation offers several advantages. Notably, polytopes are closed under linear maps and intersections, simplifying computational procedures. Additionally, limiting the representation to two-dimensional polytopes is particularly advantageous for the specific applications discussed in this thesis, as it mitigates the unfavorable computational complexity associated with high-dimensional polytopes. To simplify the notation, the collection of $\mathcal{R}_k^{(i)}$ is denoted as $\mathcal{R}_k$, where:

$$\mathcal{R}_k = \{\mathcal{R}_k^{(1)}, \ldots, \mathcal{R}_k^{(i)}\} \text{ and } i \in \mathbb{N} \tag{2.4}$$

In addition to the reachable set representation, we define the projection operator $\text{proj}_{\square} : \mathcal{X} \to \mathbb{R}^m$, which allows us to map each state $x \in \mathcal{X}$ to the specific elements defined by $\square$. For instance, when applying $\text{proj}_{(s,d)}(\mathcal{R}_k^{(i)})$, we project $\mathcal{R}_k^{(i)}$ onto the position domain, which results in an axis-aligned rectangle denoted as $\mathcal{D}_k^{(i)}$ (Fig. 2.3c). The set $\mathcal{D}_k$ is then formed by the union of these axis-aligned rectangles, collectively known as the *drivable area*. This drivable area contains all collision-free positions that the ego vehicle can reach and is essentially the projection of the reachable set onto the



(a) $\mathcal{P}_{s,k}^{(i)}$ (b) $\mathcal{P}_{d,k}^{(i)}$ (c) $\mathcal{D}_{k}^{(i)}$

Figure 2.3.: Polytopes and drivable area of a base set $\mathcal{R}_k^{(i)}$.

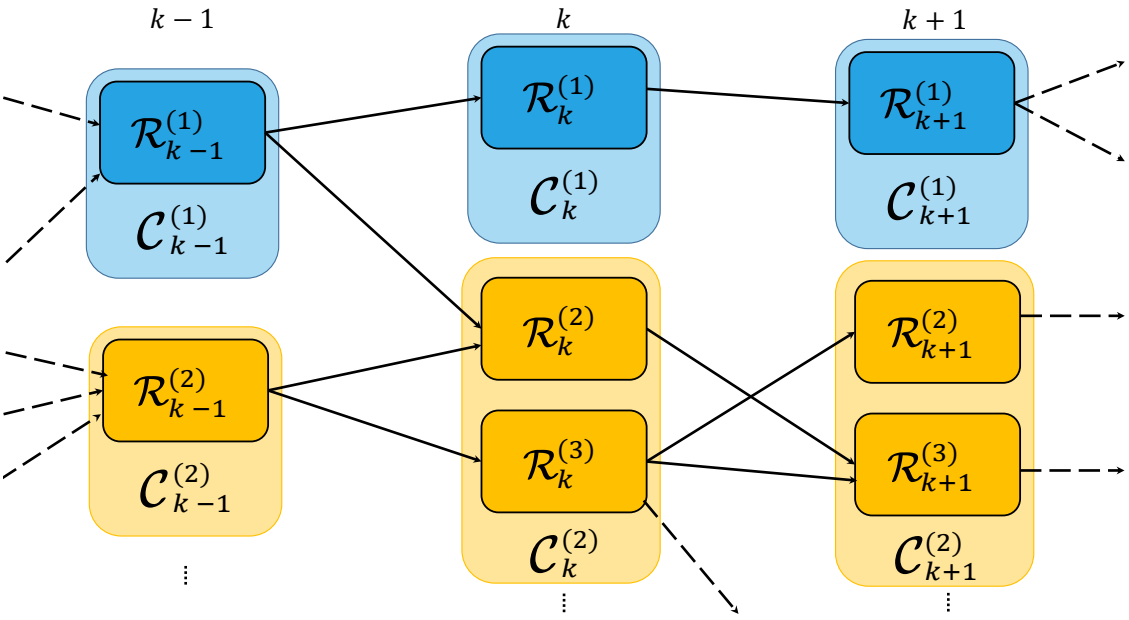


Figure 2.4.: Example of a reachability graph $\mathcal{G}_{\mathcal{R}}$, where each color represents a different driving corridor.

position domain [2]. By extension, the collection of $\mathcal{D}_k^{(i)}$ is simply represented as $\mathcal{D}_k$, offering a compact and convenient notation.

To represent the spatio-temporal relationships among the base sets $\mathcal{R}_k^{(i)}$, we utilize a directed, acyclic graph denoted as $\mathcal{G}_\mathcal{R}$. Figure 2.4 shows an example of a reachability graph, where nodes of identical colors possess interconnected drivable areas within a single step and are part of the same driving corridor. Each node in this graph corresponds to one base set $\mathcal{R}_k^{(i)}$, and connecting edges between two base sets $\mathcal{R}_k^{(i)}$ and $\mathcal{R}_{k+1}^{(j)}$ indicate that the latter base set $\mathcal{R}_{k+1}^{(j)}$ is reachable from the former base set $\mathcal{R}_k^{(i)}$ after performing a single step. The graph structure provides an efficient way to store and visualize the temporal evolution of the base sets, facilitating a clear representation of their interconnections and accessibility throughout the time steps.

2.4. Driving Corridors

Many scenarios can often consist of obstacles that may cause possible fragmentation in the drivable area. This setting can eventually lead to disconnected reachable sets. In order to address this issue, we leverage the definition of *connected sets* [5] which we denote by $\mathcal{C}_k^{(n)} \subseteq \mathcal{D}_k$, where $n \in \mathbb{N}_0$. These connected sets represent distinct regions within the drivable area at each discretized time step k.

A driving corridor is defined as a temporal sequence of connected sets $\mathcal{C}_k^{(n)} \subseteq \mathcal{D}_k$ over steps k_0 to k_f [2], i.e.,

$$\mathcal{C}(\cdot) = (\mathcal{C}_{k_0}^{(m)}, \ldots, \mathcal{C}_{k_f}^{(n)}), \text{ where } \mathcal{C}_k^{(n)} \subseteq \mathcal{D}_k \tag{2.5}$$

At every time step, the drivable area can contain several connected sets, leading to the existence of multiple driving corridors. The reachability graph example depicted in Fig. 2.4 clearly reveals the presence of two distinct driving corridors. These driving corridors are visually represented in Fig. 2.5.

Driving corridors can correspond to different homotopy classes. Homotopic classes are sets of trajectories which join the same initial and end states, and that could be smoothly deformed into one another without intersecting obstacles [53]. This classification consequently provides an insight into the various maneuvers that the ego vehicle can perform in response to the presence of obstacles.

Evading or braking in front of the given obstacles can be possible maneuvers exhibited by the driving corridors [5]. In order to illustrate this case, let us consider an example shown in Fig. 2.5. This example demonstrates two driving corridors that can represent the braking (shown on the top part) and evasive maneuver (shown on the bottom part). This manner of representation of the driving corridors allows us to explore and evaluate different possible trajectories for the ego vehicle. Consequently it can be argued that we could gain valuable knowledge about the range of possible safe motions that can be achieved in a given scenario. Therefore this can ensure that our trajectory planning is adaptable and robust, due to the diverse set of considered maneuvers.

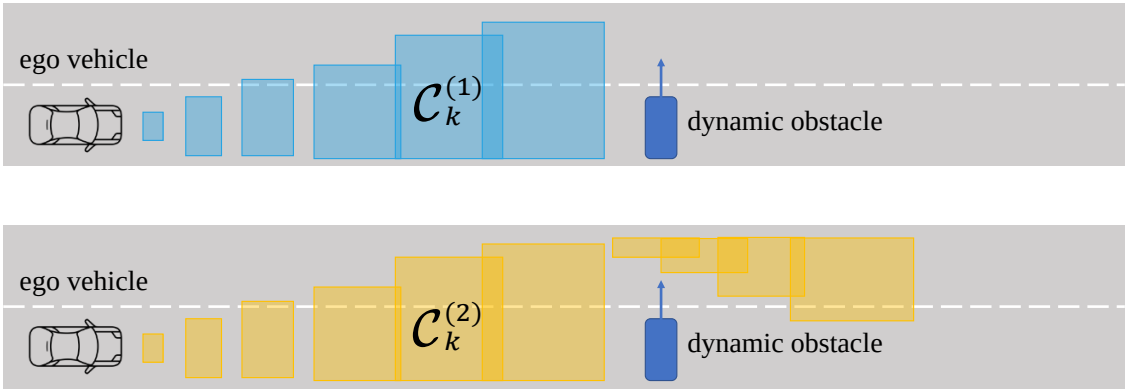


Figure 2.5.: Two driving corridors identified from the reachability graph in Fig. 2.4.

Driving corridors can be categorized into longitudinal and lateral driving corridors. This distinction is particularly of high importance for motion planners that independently plan the longitudinal (along the road axis) and lateral (perpendicular to the road axis) motions of a vehicle [5]. Given a longitudinal driving corridor, we could construct a lateral driving corridor by following these steps. First, we consider the positions $p_{s,k}$ of a planned trajectory. Second, we identify the connected sets within the longitudinal driving corridor which contains these positions $p_{s,k}$ for all time steps k in the range $\{k_0, \ldots, k_f\}$. The completion of these steps concludes the formation of the lateral driving corridor [5].

The upper mentioned process guarantees that the lateral driving corridor aligns with the planned trajectory. As a result, the motion planner can account for possible lateral positions (of the ego vehicle) at each time step along the longitudinal trajectory. The partitioning of driving corridors enables the independent optimization of longitudinal and lateral motions. Consequently this distinction can impact the strategies used for planning trajectories which are not more effective but also highly flexible. Additionally this approach carries importance in scenarios where the movement of the vehicle requires careful coordination to guarantee secure and efficient navigation, around obstacles and other vehicles, on the road.

2.5. Signal Temporal Logic

In this section, we present Signal Temporal Logic (STL) through the introduction of its syntax, semantics and robustness measurement. STL has recently gained recognition as an increasingly prominent formalism that excels in the specification and verification of behaviors exhibited by dynamic systems. It extends linear temporal logic (LTL) by incorporating signal functions that operate over continuous time intervals. This can eventually spark the possibility for formalizing descriptions of system properties over time in dense-time domains [41]. One can make the case that when compared to discrete

time logics, like LTL or MTL, STL proves to be especially beneficial for representing and studying systems that involve real-valued signals. Such systems can encompass the diverse applications where STL is leveraged, such are cyber-physical systems or autonomous vehicles.

The composition of formulas with temporal and non-temporal operators lies at the core of STL. This composition can represent the change of a system over time, resulting in the ability to express a broad spectrum of specifications. Formalism in STL can vary from simple temporal relations, to the expression of more complicated behaviors that can be exhibited by a dynamic system. Specifications can encompass signal patterns, constraints, as well as robustness measurements.

Thanks to its expressiveness and flexibility, STL is highly beneficial in capturing real-world requirements. For this reason, it can enable rigorous verification and synthesis of complex systems, as outlined in [54].

2.5.1. Syntax of STL

In this section we present the syntax of STL. An STL formula ϕ is constructed using the following grammar:

$$\phi := \pi | \neg\phi | \phi_1 \wedge \phi_2 | \phi_1 \vee \phi_2 | \phi_1 U_I \phi_2, \tag{2.6}$$

where π represents an atomic proposition, and ϕ_1 and ϕ_2 are STL formulas. The grammar contains the negation ($\neg$), conjunction ($\wedge$), and disjunction ($\vee$) operators, which are used to compose the STL formulas. Additionally, the grammar includes the temporally bounded *until* operator denoted by U_I, which plays a crucial role in expressing temporal properties in STL. This operator represents a temporal relation between two propositions, stating that one proposition is true until another proposition is satisfied within the given time interval I.

The given grammar allows for the straightforward construction of various other temporal operators. For instance, the *future* operator can be obtained by using the following equivalence: $F_I\phi = \top U_I \phi$. This operator requires the expression ϕ to hold at some time within the specified time interval. Similarly, the *globally* operator can be achieved through the following equivalence: $G_I\phi = \neg(\top U_I \neg\phi)$, which requires the satisfaction of the formula ϕ at every time step within the designated time interval.

In this thesis, we adopt an interpretation of the temporally bounded operators that is relative to the current time step, rather than relying on absolute time intervals, similarly to [55]. Nevertheless, our implementation allows for seamless switching between relative and absolute reasoning concerning the time intervals. This flexibility provides us with the ability to perform precise temporal analysis while accommodating different approaches to suit specific scenarios or requirements.

2.5.2. Semantics of STL

Consider an STL formula ϕ defined over the state space $\mathcal{X}$, and let $\zeta : \mathbb{R}^{\geq 0} \to \mathcal{X}$ be a trajectory also within the same state space. We denote that the trajectory ζ satisfies the

formula ϕ starting at time k as $\zeta, k \models \phi$. This notation indicates that the trajectory ζ adheres to the specified conditions expressed by the formula ϕ beginning at the given time k. The semantics for the given operators is defined as follows:

$$\begin{aligned}
\zeta, k \models \pi &\iff \pi(\zeta(k)) = \text{true} \\
\zeta, k \models \neg\phi &\iff \zeta, k \not\models \phi \\
\zeta, k \models \phi_1 \wedge \phi_2 &\iff \zeta, k \models \phi_1 \text{ and } \zeta, k \models \phi_2 \\
\zeta, k \models \phi_1 \vee \phi_2 &\iff \zeta, k \models \phi_1 \text{ or } \zeta, k \models \phi_2 \\
\zeta, k \models \phi_1 U_{[0,M]} \phi_2 &\iff \exists k' \in [k, k+M] : \zeta, k' \models \phi_2,\ \forall k'' \in [k, k'] : \zeta, k'' \models \phi_1
\end{aligned}$$

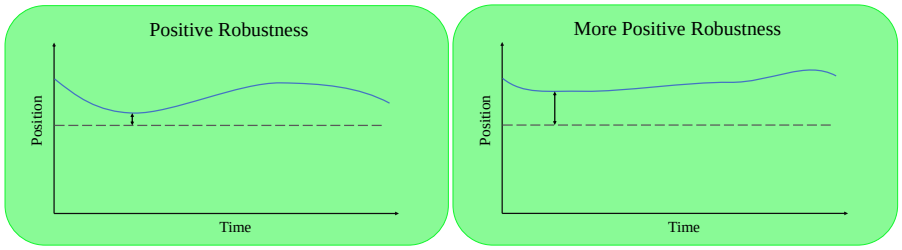
The logic primitives $\top$ (true) and $\bot$ (false) can be utilized as expected. The constant $\top$ represents a tautology, signifying that the corresponding formula is universally true, while $\bot$ denotes unsatisfiability, indicating that the formula is always false regardless of the assignment. The *future* operator $F_I\phi$ indicates that the formula ϕ must eventually hold during the time interval I, allowing for the expression of future-oriented temporal properties. On the other hand, the *globally* operator $G_I\phi$ requires that the formula ϕ holds at every discrete time step within the interval I, expressing the property's persistence over time.

2.5.3. STL Robustness

The robustness measurement is an essential feature of STL that allows the assessment of the satisfaction or violation of a formula over a trajectory. The ability to quantify robustness is critical in understanding the extent to which a trajectory adheres to a specific behavior. Not only does this measurement provide a satisfaction degree, but it can also offer insights into the capability of a system to withstand disturbances under the conditions of uncertainty [41]. Therefore, one can easily argue that robustness is a powerful tool in providing us a picture of the system's compliance over time. As a result, STL and its robustness measurement has been proven to be favorable to supporting system verification, motion planning and control aspects.

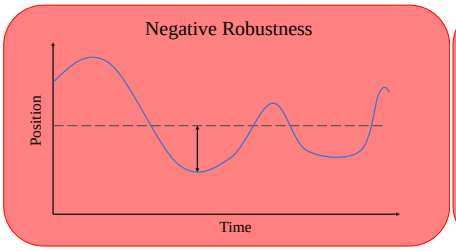
Given a specification ϕ and a trajectory ζ, the robustness score $\rho(\phi, \zeta, k)$ at time k is recursively computed as [56]:

$$\begin{aligned}
\rho(\top, \zeta, k) &:= \rho_\top, \\
\rho(\bot, \zeta, k) &:= -\rho_\top, \\
\rho(\pi(\zeta) \geq c, \zeta, k) &:= \pi(\zeta_k) - c, \\
\rho(\neg\phi, \zeta, k) &:= -\rho(\phi, \zeta, k), \\
\rho(\phi_1 \wedge \phi_2, \zeta, k) &:= \min\{\rho(\phi_1, \zeta, k), \rho(\phi_2, \zeta, k)\}, \\
\rho(\phi_1 \vee \phi_2, \zeta, k) &:= \max\{\rho(\phi_1, \zeta, k), \rho(\phi_2, \zeta, k)\}, \\
\rho(\phi_1 U_{[0,M]} \phi_2, \zeta, k) &:= \max_{k' \in [k, k+M]} \{\min\{\rho(\phi_1, \zeta, k'), \min_{k'' \in [k, k']} \rho(\phi_2, \zeta, k'')\}\}
\end{aligned}$$

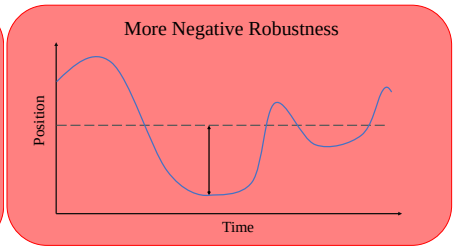


(a) Positive robustness

(b) Higher satisfaction degree



(c) Negative robustness



(d) Higher violation degree

Figure 2.6.: Example showcasing positive and negative robustness values for specific given signals.

In Fig. 2.6, we present an illustrative example that demonstrates the variance of the robustness measurement for STL. Four examples showcase the position signal's behavior over time, with each signal subjected to the simple STL formula $\text{position}(\zeta) \geq c$, where ζ represents the trajectory and c is a given constant. The dashed line in the figures represents the threshold value specified by the formula. In Figure 2.6a, the signal exhibits a positive robustness since it consistently maintains a position higher than the specified threshold. Conversely, in Figure 2.6c, the signal fails to preserve the required deviation from the threshold, resulting in a negative robustness.

Further analysis of the examples reveals other insights. By comparing Figures 2.6a and 2.6b, we observe that the latter case demonstrates higher robustness since it sustains a larger deviation from the threshold. Similarly, Figures 2.6c and 2.6d highlight the difference in robustness, with the latter signal exhibiting a greater negative robustness due to its more significant deviation from the threshold limit. This example emphasizes the importance of robustness measurement in quantifying the ability of signals to adhere to specified conditions, aiding in the assessment and comparison of different system behaviors over time.

The robustness measurement mentioned above represents the traditional approach for quantifying STL robustness. As discussed in Section 1.2.3, this conventional method has its inherent limitations, making it unsuitable for certain applications. Our primary objective is to enhance the quantification of STL robustness and, ultimately, to assess the robustness of driving corridors based on a given specification. Through our work, we aim to overcome the drawbacks of the traditional approach and pave the way for more effective and accurate robustness analysis.

3. Algorithmic Framework

This chapter introduces the algorithmic framework for the representation of a given STL specification. In temporal logic, many temporal operators exhibit the likelihood for the existence of many satisfaction possibilities. One typical example is the consideration of the *Until* operator. Depending on the length of the interval on which this operator is defined, it can be satisfied in different ways, as noted in Sec. 2.5.2. For example, two possibilities for the satisfaction of the simple specification $\phi_1 U_{[0,M]} \phi_2$ can be the following:

1. satisfaction of ϕ_1 from time steps 0 to $M-1$ and satisfaction of ϕ_2 at time step M.
2. the direct satisfaction of ϕ_2 at time step 0.

Therefore, it becomes crucial to consider the numerous possibilities that encompass the alternatives for satisfying a temporal operator. In this thesis, we will refer to these possibilities as *sequences* (denoted by *seq*). Eventually, a *sequence* corresponds to a representation of a satisfaction alternative to the temporal operator. Additionally, the collection of these unique sequences concludes the whole satisfaction space to which we will refer as the *solution* (denoted by *sol*).

In this chapter, we will consider *sequences* as a collection of pairs comprised of a time step and the corresponding polygon associated with a proposition. In Sec. 2.3.2, we already explored that a reachable set is defined by two polytopes on the longitudinal and lateral dimensions. Additionally, these polytopes represent the reachable positions and velocities. In this thesis, we aim a similar polygon-based representation for a given specification, defined on the position and velocity axes.

This approach offers several advantages as we leverage the combined information from both position and velocity, leading to a more thorough and accurate assessment of the system's behavior. First, it simplifies the process of specifying temporal formalisms in the domain of reachable sets. Additionally, it enables a comprehensive evaluation of the system by capturing interdependencies between variables, thereby ensuring consistency by avoiding conflicts in results.

The rest of this chapter is organized as follows. Sec. 3.1 presents the solution formulation. In Sec. 3.2 auxiliary notations and operations are introduced. Lastly, in Sec. 3.3 we introduce the interface EVAL and generate the solution representation for temporal and non-temporal operators.

3.1. Formulation

The solution of a temporal specification is given by

$$\langle \text{seq}_1; \text{seq}_2; \ldots; \text{seq}_m \rangle,\ m \in \mathbb{N}$$

where each seq_i represents a sequence and $i \in \mathbb{N}$. Additionally, the sequence itself can be formalized as following:

$$\text{seq}_i = \langle S_{k_s}, \ldots, S_{k_e} \rangle,$$

where each S_j is a polygon or a multi-polygon representing a specification at time j in the position-velocity domain and $k_s \leq j \leq k_e$. The index k_s represents the starting time step and k_e the final step of the considered time horizon. Given the above formalization, the solution can be represented as a 2-dimensional matrix, where columns are indexed by $k_s, k_s + 1, \ldots, k_e$, and rows are indexed by $1, \ldots, m$.

$$\begin{array}{rcccc} & k_s & k_s+1 & \ldots & k_e \\ \text{seq}_1: & S^1_{k_s} & S^1_{k_s+1} & \ldots & S^1_{k_e} \\ \text{seq}_2: & S^2_{k_s} & S^2_{k_s+1} & \ldots & S^2_{k_e} \\ \ldots & & & & \\ \text{seq}_m: & S^m_{k_s} & S^m_{k_s+1} & \ldots & S^m_{k_e} \end{array}$$

The notation $S^i_{k_j}$ refers to a polygon or multi-polygon of sequence i at time k_j. This representation allows us to uniquely identify and analyze various satisfaction possibilities throughout the solution space. An illustration associated with this definition is shown in Fig. 3.1. Two formulas ϕ_1 and ϕ_2 are chosen and their respective sequences are listed. These sequences enumerate all the possible ways to represent the underlying logical formulas within the domain defining the driving corridors for the considered time range. Each specification polygon within a sequence is numbered in order to provide information regarding the evolution of the specification throughout the considered time horizon.

3.2. Auxiliary notations and operations

In order to enhance the clarity and expressiveness of the following terminology and representations, we introduce auxiliary notations and operations. We leverage the notations in 3.2.1 through the rest of the thesis and the operations in 3.2.2 as the main backbone for supporting the various solutions to logical operators.

We will additionally implement a function $\text{EVAL}(\phi, \mathcal{C}, k_s, k_e)$ that evaluates the traces in the corridor $\mathcal{C}$ from time k_s to time k_e with respect to ϕ. The function will compute the solution representation of ϕ according to the structure that the formula represents. We note that the evaluation of the function happens at time k_s and k_e is the look-ahead horizon, which determines the span of time required to evaluate a temporal specification.

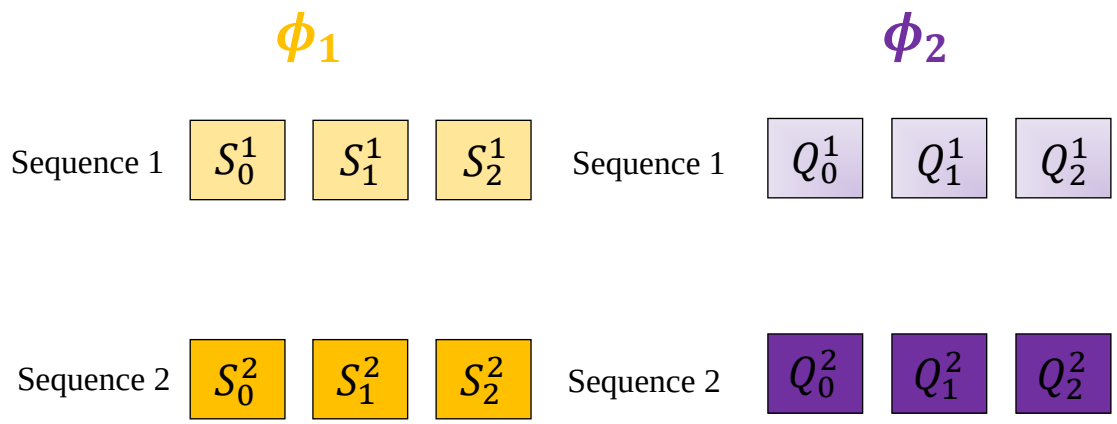


Figure 3.1.: Example illustrating two sequences for formulas ϕ_1 and ϕ_2. The rectangles show the specification polygons for the considered time horizon of $[0, 2]$ on different sequences.

3.2.1. Auxiliary notations

- sol.seq_i denotes the i-th sequence (i-th row) of the solution sol.
- sol.col_i denotes the i-th column of the solution.
- $\text{seq}[k]$ represents a polygon or multi-polygon from a sequence at the k-th time step.
- $\mathcal{C}(k)$ denotes the reachable sets contained in the driving corridor $\mathcal{C}$ at time k.
- $|\text{sol}|$ is the number of sequences in a solution.
- Given two sequences seq_1 and seq_2, and a binary operator f (e.g., intersection, union), then the result of the operation is another sequence computed as

$$\langle f(\text{seq}_1[k_s], \text{seq}_2[k_s]]), \ldots, f(\text{seq}_1[k_e], \text{seq}_2[k_e])\rangle.$$

3.2.2. Auxiliary operations

The collection of the basic operations are presented algorithmically in Alg. 1, which provide procedures related to the intersection and appending of solutions. In addition various helper functions are used in order to access specific properties or elements from the sequences and solutions:

- sol.AddSequence(seq) denotes the function appending a sequence seq to the solution sol, if seq $\notin$ sol.
- sol.GetStartTime return the starting time of a solution.

- sol.GetEndTime return the end time of the considered time horizon.
- seq.Append($\langle k, P \rangle$) appends the pair $< k, P >$ to the sequence, where k is the time step and P a polygon.
- sol.InsertColumn(j, P) adds in the j-th column of solution sol the polygon P.

The procedure AppendSolution is responsible for enlarging the solution space of a solution by additionally considering the sequences of another solution. It is an essential procedure, especially in cases where more possibilities have to be considered. The *OR* ($\vee$) operator is most common to invoke this procedure since it needs to consider the satisfaction possibilities of both operands on which it performs.

The second procedure IntersectSolutions intersects two solutions sol_1 and sol_2. It first determines the common start and end times for both solutions, which are required to be similar in order to establish a meaningful basis for comparison and analysis. Then it performs an iteration through each sequence of sol_1 and sol_2, and intersects the corresponding sequences using IntersectSequences, in order to produce resulting sequences that encapsulate the temporal segments where both solutions exhibit mutual agreements. If the intersection is not empty, the resulting sequence is added to the first solution. Otherwise, an empty intersection provides no contribution to the enlargement of the solution space, thus making it unnecessary to be appended in the satisfaction scope of a solution.

The use of this procedure is mostly common in the *AND* ($\wedge$) operator. According to this operator, we should return those solutions that can satisfy both operands. Therefore, it is important to pair-wisely consider each sequence from both solutions and to conclude similarities in the satisfaction possibilities that they represent.

The third procedure IntersectSequences facilitates the intersection of two sequences seq_1 and seq_2 from time k_s to the end of the considered time horizon k_e, by aligning and synchronizing the shared temporal information over the time range. It iterates over each time step within the given range and computes the intersection of polygons at each step. If the intersection is not empty, the resulting pair $< k, P >$ is added to the sequence. On the other hand, an empty intersection denotes the possibility of traces that unexpectedly terminate and begin between time steps, thus making it meaningless for the sequence to even be considered a possible part of the solution space.

3.3. Implementation of EVAL

In this section, we will look at the various forms of representing the solution of a formula ϕ based on its structure. The function EVAL considers a formula ϕ and produces a solution according to the operators that have constituted the formula. This function performs its evaluation from a given time step k_s to the end of the time horizon k_e. In this way, the function EVAL behaves as an interface allowing for each subformula to be evaluated on its own time conditions. For example, a formula which is evaluated

Algorithm 1 Auxiliary operations

sol.AppendSolution(sol_2)

Require: start and end times for sol and sol_2 are the same

for $r_2 \leftarrow 0, \ldots, |\text{sol}_2| - 1$ **do**
    $\text{seq}_2 \leftarrow \text{sol}_2.\text{seq}_{r_2}$
    sol.AddSequence(seq_2)
end for

IntersectSolutions($\text{sol}_1, \text{sol}_2$)

Require: start and end times for both solutions are the same

$k_s \leftarrow \text{sol}_1$.GetStartTime()
$k_e \leftarrow \text{sol}_1$.GetEndTime()
$\text{sol} \leftarrow \emptyset$
for $r_1 \leftarrow 0, \ldots, |\text{sol}_1| - 1$ **do**
    $\text{seq}_1 \leftarrow \text{sol}_1.\text{seq}_{r_1}$
    for $r_2 \leftarrow 0, \ldots, |\text{sol}_2| - 1$ **do**
        $\text{seq}_2 \leftarrow \text{sol}_2.\text{seq}_{r_2}$
        $\text{seq} \leftarrow$ IntersectSequences($\text{seq}_1, \text{seq}_2, k_s, k_e$)
        if $\text{seq} \neq \emptyset$ **then** sol.AddSequence(seq)
    end for
end for
return sol

IntersectSequences($\text{seq}_1, \text{seq}_2, k_s, k_e$)

$\text{seq} \leftarrow \emptyset$
for $k \leftarrow k_s, \ldots, k_e$ **do**
    $p \leftarrow \text{seq}_1[k] \cap \text{seq}_2[k]$
    if $p = \emptyset$ **then return** $\emptyset$
    seq.Append($\langle k, p \rangle$)
end for
return seq

at times $k_s, \ldots, k_e$ can easily invoke the EVAL function to evaluate a subformula ϕ' at times $k_{s'}, \ldots, k_{e'}$.

3.3.1. Evaluation of True constant

A formula ϕ can be easily represented by a True constant, denoted as $\phi = \top$, which indicates its universal satisfiability. We commonly refer to such formulas as tautologies [57]. In our setting, this translates to accepting all traces that originate from time step k_s and continue up to the time k_e. Therefore, we can simply represent the solution to this formula as a sequence that contains all reachable sets in $\mathcal{C}$ limited to times from k_s to k_e,

formally given as:

$$\text{sol}_{\phi=\top} = \langle (k_s, \mathcal{C}(k_s)), (k_{s+1}, \mathcal{C}(k_{s+1})), \ldots, (k_e, \mathcal{C}(k_e)) \rangle \tag{3.1}$$

The outline of the algorithm responsible for generating the solution represented in (3.1), is introduced in Algorithm 2. The algorithm iteratively builds a sequence by appending the reachable sets on the given time range. As a result, the solution contains a single sequence comprised of reachable sets from time k_s to k_e.

Algorithm 2 Implementation of TRUE : EVAL$(\top, \mathcal{C}, k_s, k_e)$

```
1: seq ← ∅
2: sol ← ∅
3: for k ← k_s, ..., k_e do
4:     seq.APPEND(⟨k, C(k)⟩)
5: end for
6: sol.ADDSEQUENCE(seq)
7: return sol
```

3.3.2. Evaluation of False constant

Another simple case for the representation of a formula can be through the False constant, denoted as $\phi = \bot$. This constant indicates the unsatisfiability of a formula, in contrast to the upper-mentioned tautologies. In our setting, this is reflected by the rejection of all traces that can originate from time step k_s to time k_e. Hence, the representation of the solution can be easily achieved through the returning of an empty solution (that contains no sequence), which can be simply represented by the $\emptyset$ symbol. Formally, we present the solution of this case as:

$$\text{sol}_{\phi=\bot} = \emptyset \tag{3.2}$$

3.3.3. Evaluation of a proposition

The third case of the representation of a formula ϕ can be through a simple proposition π, denoted as $\phi = \pi$. This proposition can be a simple predicate, e.g., lateral_distance (obstacle) ≤ 5 . Let P denote the polygon associated with π. Polygon P is the solution extended by the remaining reachable sets. Since a proposition π is not a temporal expression (not associated with an interval), we only have to return the specification polygon P (representing π) at the required evaluation time k_s. The remainder of the solution for the following time steps can be the rest of the trace (reachable sets) since no restrictions have to be considered for times $k > k_s$. Hence, the solution is given as follows:

$$\text{sol}_{\phi=\pi} = \langle (k_s, P), (k_s + 1, \mathcal{C}(k_s + 1)), \ldots, (k_e, \mathcal{C}(k_e)) \rangle \tag{3.3}$$

Algorithm 3 Implementation of PROPOSITION : EVAL$(\pi, \mathcal{C}, k_s, k_e)$

1: $P \leftarrow$ polygon (or multi-polygon) associated with the proposition π
2: seq $\leftarrow \emptyset$
3: sol $\leftarrow \emptyset$
4: seq.APPEND$(\langle k_s, P \rangle)$
5: **for** $k \leftarrow k_s + 1, \ldots, k_e$ **do**
6: seq.APPEND$(\langle k, \mathcal{C}(k) \rangle)$
7: **end for**
8: sol.ADDSEQUENCE(seq)
9: **return** sol

The algorithm representing the generation of solution (3.3) is introduced in Algorithm 3. The algorithm starts by storing the associated polygon P with the proposition π and continues by appending the polygon together with the first time step k_s. The time horizon is iterated from k_{s+1} to k_e and at each iteration a pair consisting of the respective time step and the reachable sets at the given step is added to the sequence.

3.3.4. Evaluation of AND operator

A common binary operator used in the formalization of specifications is the *AND* operator. In our setting, this operator is translated into the intersection of solutions of both operands, as discussed in Section 3.2.2. Formally, let:

- $\text{sol}_1 = \text{EVAL}(\phi_1, \mathcal{C}, k_s, k_e)$, be the solution for ϕ_1 and
- $\text{sol}_2 = \text{EVAL}(\phi_2, \mathcal{C}, k_s, k_e)$, the solution for ϕ_2.

The solution for ϕ is then obtained by computing the intersection between each sequence in sol_1 with each sequence in sol_2, i.e.,

$$\text{sol}_{\phi_1 \wedge \phi_2} = \langle \textsc{Intersect}(\text{sol}_1.\text{seq}_i, \text{sol}_2.\text{seq}_j) : i \in \{1, \ldots, |\text{sol}_1|\}, j \in \{1, \ldots, |\text{sol}_2|\} \rangle, \quad (3.4)$$

where $\textsc{Intersect}(\text{seq}_i, \text{seq}_j)$ is the function returning the intersection of two sequences seq_i and seq_j (Alg. 1).

The algorithm for generating the solution as outlined in (3.4) is introduced in Algorithm 4. Lines 1 and 2 are responsible for storing the solution spaces from the two operands ϕ_1 and ϕ_2. The algorithm concludes in line 3 by performing the intersection of the two solutions.

Algorithm 4 Implementation of AND : EVAL$(\phi_1 \wedge \phi_2, \mathcal{C}, k_s, k_e)$

1: $\text{sol}_1 \leftarrow$ EVAL$(\phi_1, \mathcal{C}, k_s, k_e)$
2: $\text{sol}_2 \leftarrow$ EVAL$(\phi_2, \mathcal{C}, k_s, k_e)$
3: **return** INTERSECTSOLUTIONS$(\text{sol}_1, \text{sol}_2)$

Considering the example illustrated in Fig. 3.1, we could visualize the *AND* operation, as shown in Fig. 3.2. The four new generated sequences come as a result of the pair-wise intersection between the sequences of the formulas ϕ_1 and ϕ_2. The green rectangles represent the new polygons of the generated sequences, which are a result of the intersection of the polygons included in the sequences of the solutions for the formulas ϕ_1 and ϕ_2. Moreover, the intersection of the polygons between sequences is realized at every time step, meaning that the pairs $\langle k, P_1 \rangle$ from the sequences of solution for ϕ_1 intersect with each pair $\langle k, P_2 \rangle$ from the sequences of solution for ϕ_2.

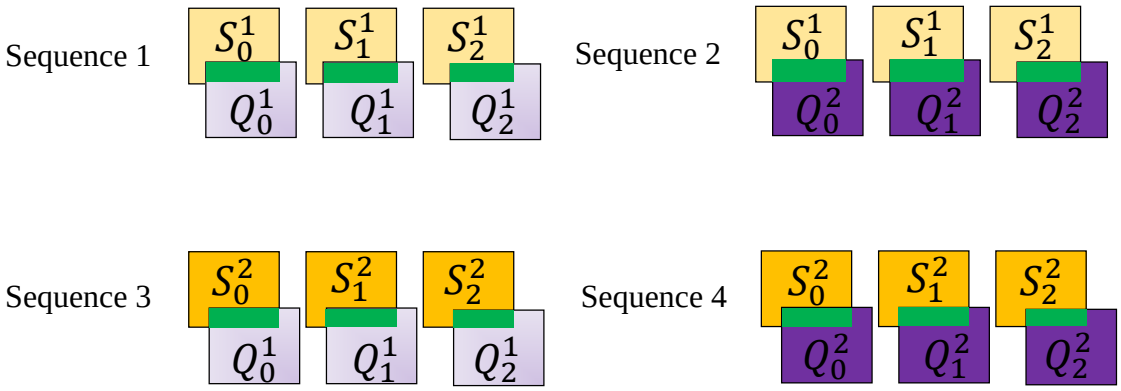


Figure 3.2.: Example illustrating $\phi_1 \wedge \phi_2$. The green rectangles represent the intersection between two polygons.

3.3.5. Evaluation of OR operator

Another common binary operator used in the formalization of specifications is the *OR* operator. It helps in the reasoning of the satisfaction of at least one of the two given operands. In our setting, this is reflected into the appending of solutions of both operands, as discussed in Section 3.2.2. Formally, let:

- $\text{sol}_1 = \text{EVAL}(\phi_1, \mathcal{C}, k_s, k_e)$, be the solution for ϕ_1 and
- $\text{sol}_2 = \text{EVAL}(\phi_2, \mathcal{C}, k_s, k_e)$, the solution for ϕ_2.

The solution for ϕ is then obtained by the expansion of the solution space of the first operand, with the solution space of the second operand, i.e.,

$$\text{sol}_{\phi_1 \vee \phi_2} = \langle \text{seq}_i \mid \text{seq}_i \in \text{sol}_1 \vee \text{seq}_i \in \text{sol}_2 \rangle \tag{3.5}$$

The algorithm for computing the solution as described in (3.5) is presented in Algorithm 5. The algorithm starts by storing the solutions of the two operands, as outlined in lines 1-2. In line 3, the solution space of the first operand is expanded through the appending of the sequences from the solution for the second operand. The algorithm terminates in line 4 by returning the newly expanded solution space.

It should be noted that the solution for the *OR* operator can be further improved. In the implementation of the Algorithm 5, we can additionally check for the existence of a sequence in the solution space. This trick can reduce the solution space quite considerably, especially in cases when the two operands provide highly similar sequences in their solutions.

Algorithm 5 Implementation of OR : EVAL$(\phi_1 \vee \phi_2, \mathcal{C}, k_s, k_e)$

1: $\text{sol}_1 \leftarrow \text{EVAL}(\phi_1, \mathcal{C}, k_s, k_e)$
2: $\text{sol}_2 \leftarrow \text{EVAL}(\phi_2, \mathcal{C}, k_s, k_e)$
3: sol_1.APPENDSOLUTION(sol_2)
4: **return** sol_1

To illustrate the generation of the solution for the *OR* operator, let's revisit the example presented in Fig. 3.1. The solution to this operation is demonstrated in Fig. 3.3, where all possible sequences from ϕ_1 and ϕ_2 are listed and indexed accordingly. The compilation of sequences arises naturally and straightforwardly for the *OR* operation since each sequence fundamentally represents a potential solution to a specification. Therefore, when we already have the solutions for the two operands beforehand, combining them is a matter of simply appending them together. The listing of solutions inherently implies a disjunction between these operands.

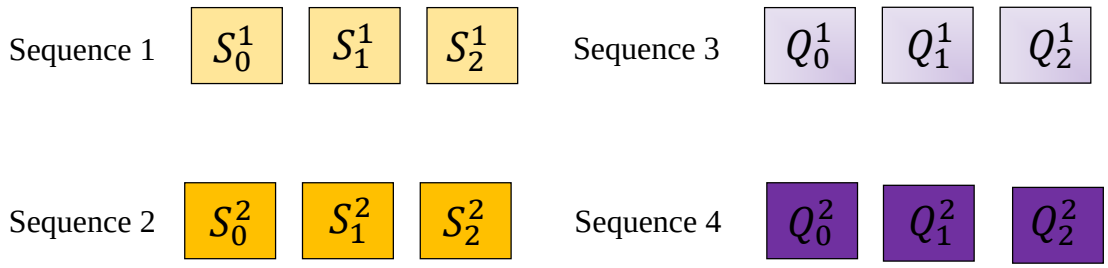


Figure 3.3.: Example illustrating $\phi_1 \vee \phi_2$. The listed sequences show the possible solutions for the OR operator.

3.3.6. Evaluation of UNTIL operator

The solution representation for the *Until* operator is essential not only because it is the first temporal operator we explore but also because it serves as the building block of other temporal operators, e.g., *Future* or *Globally*. This binary temporal operator ensures that the first operand should hold true until the second operand evaluates to true. Before we investigate the representation of the solution for this operator, we define the following notations as:

- $\text{sol}_1 = \text{EVAL}(\phi_1, \mathcal{C}, k_s, k_e)$, i.e., solution for ϕ_1

- $\text{sol}_2 = \text{EVAL}(\phi_2, \mathcal{C}, k_s, k_e)$, i.e., solution for ϕ_2
- I the time interval

Without loss of generality, we assume that the time interval I is relative with respect to the start time k_s. We also assume that I starts from 0. Additionally, we denote by $|I|$ the end time of the interval I.

Let us consider the formula $\phi = \phi_1 U_I \phi_2$. The obtainment of the solution for ϕ can vary according to the following possibilities:

- ϕ_2 is satisfied at time $k_s + 0$; *or*
- ϕ_2 is not satisfied at time $k_s + 0$, ϕ_1 is satisfied at time $k_s + 0$, and ϕ_2 is satisfied at time $k_s + 1$; *or*
- ...; *or*
- ϕ_2 is not satisfied at any time from $k_s + 0$ to $k_s + k - 1$, ϕ_1 is satisfied at every time from $k_s + 0$ to $k_s + k - 1$, and ϕ_2 is satisfied at time $k_s + k$; *or*
- ...; *or*
- ϕ_2 is not satisfied at any time from $k_s + 0$ to $k_s + |I| - 1$, ϕ_1 is satisfied at every time from $k_s + 0$ to $k_s + |I| - 1$, and ϕ_2 is satisfied at time $k_s + |I|$.

Let us take into account a time step k with $0 \leq k \leq |I|$. We can compute the solution set for "ϕ_2 is satisfied at time $k_s + k$" as

$$\text{part}_2^k \leftarrow \text{EVAL}(\phi_2, \mathcal{C}, k_s + k, k_e).$$

We can further compute the solution set for "ϕ_2 is not satisfied at time $k_s + k$ and ϕ_1 is satisfied at time $k_s + k$" as

$$\text{part}_1^k \leftarrow \text{EVAL}(\phi_1 \wedge \neg\phi_2, \mathcal{C}, k_s + t, k_e).$$

We can put these components together to obtain the solution for ϕ at time k:

$$\text{sol}^k \leftarrow \text{part}_1^0.\text{col}_{k_s+0} \circ \text{part}_1^1.\text{col}_{k_s+1} \circ \ldots \circ \text{part}_1^{k-1}.\text{col}_{k_s+t} \circ \text{part}_2^k \tag{3.6}$$

Therefore, the overall solution for ϕ is

$$\text{sol}_{\phi_1 U_I \phi_2} \leftarrow \langle \text{sol}^0; \ldots; \text{sol}^{|I|} \rangle \tag{3.7}$$

To illustrate the above approach let us consider the example presented in Fig. 3.4. This example introduces two formulas ϕ_1 and ϕ_2 and their respective sequences from time steps 0 to 4. The formulas are evaluated at $t_s = 0$. The sequences contain the polygon (shown in gold and purple for each formula) associated with a proposition

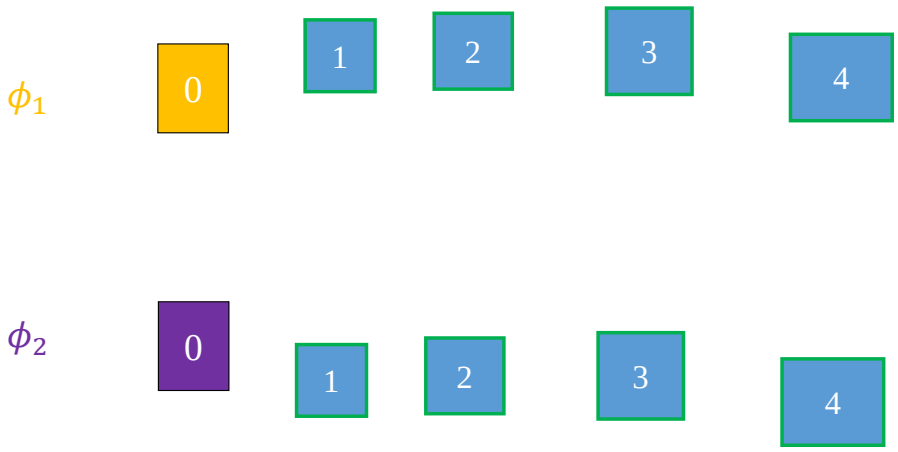


Figure 3.4.: Example illustrating two formulas ϕ_1 and ϕ_2 with their respective sequences from time steps 0 to 4.

at time 0 and the highlighted blue rectangles represent the reachable sets, which are considered part of the solution, as discussed in (3.3).

Based on the illustration shown in Fig. 3.4, we present the following example in Fig. 3.5 which illustrates the different sequences that can be generated from the expression $\phi_1 U_{[0,2]} \phi_2$. The blue rectangles show the reachable sets from time step 0 to time step 4. The specification polygons are drawn in gold for $\phi_1 \wedge \neg\phi_2$ and purple for ϕ_2. The highlighted rectangles in green represent the reachable sets that can be considered as part of the solution.

The first sequence represents the case, when the specification ϕ_2 is already satisfied at the first time step (denoted by part_2^k). Thus, the solution is represented as the polygon for ϕ_2 together with the rest of the reachable sets.

The second and third cases assume that ϕ_2 was not satisfied at the early time steps (time step 0 in the second sequence and time steps 0 and 1 in the third sequence). This means that $\phi_1 \wedge \neg\phi_2$ was satisfied instead, since the left-hand side of the *Until* expression has to hold until the satisfaction of ϕ_2 (denoted by part_1^k). Hence, the solution is represented as the sequences which assume the early satisfaction of $\phi_1 \wedge \neg\phi_2$ (shown in gold), followed by the specification polygon for ϕ_2, together with the rest of the reachable sets.

In order to further explain the generalized approach discussed above, we outline the procedure for the generation of the solution for the *Until* operator in Algorithm 6. The algorithm starts by defying two empty sets, namely parts1 and parts2, which will store the solutions at every time step $k \in \{0, \ldots, min(M, k_e - k_s + 1)\}$. It should be noted

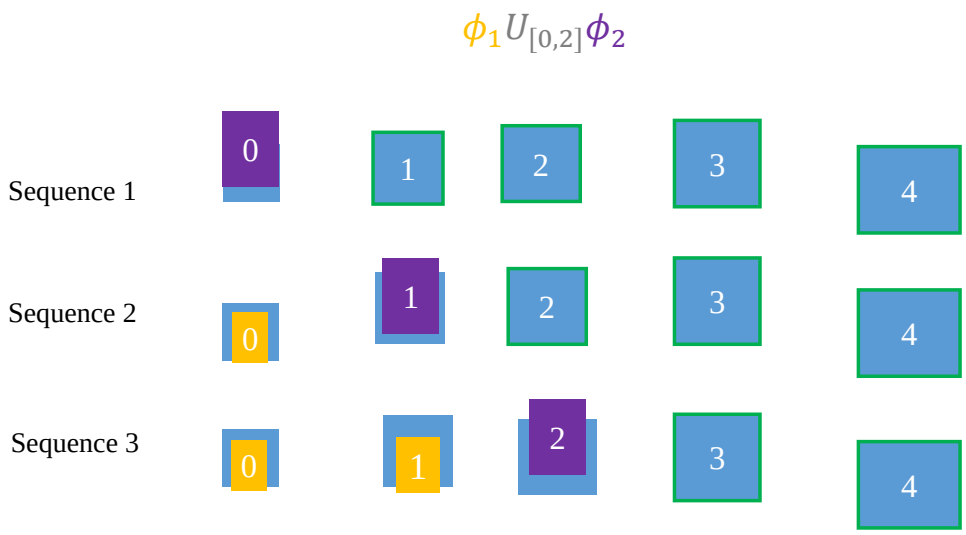


Figure 3.5.: Example illustrating $\phi_1 U_{[0,2]} \phi_2$.

that the *Until* operator can contain other temporal operators as its operands. Therefore, the recursive calls can invoke the EVAL function outside the considered time horizon. However, we are interested only in the time range from k_s to k_e. For this reason, the k index iterates at most $min(M, k_e - k_s + 1)$ in order to account for any recursive calls beyond the scope of the considered time frame.

The algorithm continues in lines 6-9 by inserting a column with the respective reachable sets at previous time steps. This further guarantees that all solutions are of the same length, as well as start and end time. In line 10, we intersect the solution of part1 with the solution of a previous step, if $k > 0$. This intersection is important since we want in the current part1 solution (which is evaluated at $k_s + k$), the result of the previous part1 solution (which is evaluated at $k_s + k - 1$). Both computed solutions (part1 and part2) are added into the sets parts1 and parts2.

The lines 14-20 correspond to the generation of sequences (or solutions) which will constitute the solution. First, the solution of parts2[0] is appended. For the rest of the time steps $k \in \{1, \ldots, min(M, k_e - k_s + 1)\}$, we intersect the solutions of parts2[k] and parts[k - 1], as noted in (3.6). If the solution produced is empty, then we omit it from contributing to the solution space. Finally, the appended solutions are returned in line 20, as described in (3.7).

3.3.7. Evaluation of NOT operator

One of the most common unary operators is the *Not* operator. This operator allows for producing the opposite truth value of a given formula. The necessity for defying

Algorithm 6 Implementation of UNTIL : EVAL$(\phi_1 U_{[0,M]} \phi_2, \mathcal{C}, k_s, k_e)$

```
parts1 ← ∅
parts2 ← ∅
for k ← 0, ..., min(M, k_e − k_s + 1) do
    part1 ← EVAL(φ_1 ∧ ¬φ_2, C, k_s + k, k_e)
    part2 ← EVAL(φ_2, C, k_s + k, k_e)
    for ℓ ← 0, ..., k − 1 do
        part1.INSERTCOLUMN(ℓ, C(ℓ))
        part2.INSERTCOLUMN(ℓ, C(ℓ))
    end for
    if k > 0 then part1 ← INTERSECTSOLUTIONS(part1, parts1[k − 1])
    parts1.ADDELEMENT(part1)
    parts2.ADDELEMENT(part2)
end for
sol ← ∅
sol.APPENDSOLUTION(parts2[0])
for k ← 1, ..., min(M, k_e − k_s + 1) do
    tmp ← INTERSECTSOLUTIONS(parts1[k − 1], parts2[k])
    if tmp ≠ ∅ then sol.APPENDSOLUTION(tmp)
end for
return sol
```

the representation of the solution for the *Not* operator also lies in the fact that many temporal and non-temporal operators incorporate *Not* in their logical equivalences. For example, the *Globally* operator can be expressed as $G_I\phi = \neg(\top U_I \neg\phi)$, as noted in Section 2.5.1.

Before we generalize the representation for the solution, let us investigate a few illustrative examples:

Example 1: Let us assume that the solution for ϕ consists of only one sequence, say $\langle \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle$. Then, the solution for $\neg\phi$ consists of the following sequences:

- $\langle \mathcal{C}(1) - \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle$ (remove if $\mathcal{C}(1) - \mathcal{X}_1 = \emptyset$)
- $\langle \mathcal{X}_1, \mathcal{C}(2) - \mathcal{X}_2, \mathcal{X}_3 \rangle$ (remove if $\mathcal{C}(2) - \mathcal{X}_2 = \emptyset$)
- $\langle \mathcal{X}_1, \mathcal{X}_2, \mathcal{C}(3) - \mathcal{X}_3 \rangle$ (remove if $\mathcal{C}(3) - \mathcal{X}_3 = \emptyset$)

where $\mathcal{C}(i)$ is the corridor (reachable sets) at time i. These are all the possible ways of how we can avoid $\langle \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle$. A sequence can be removed from the solution when $\mathcal{C}(i) - \mathcal{X}_i = \emptyset$.

Example 2: Let us now consider another case, when the solution for ϕ consists of two sequences, say sol $= \langle \langle \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle, \langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle \rangle$. We can start the same procedure as before and construct:

- $\langle \mathcal{C}(1) - \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle$ (remove if $\mathcal{C}(1) - \mathcal{X}_1 = \emptyset$)
- $\langle \mathcal{X}_1, \mathcal{C}(2) - \mathcal{X}_2, \mathcal{X}_3 \rangle$ (remove if $\mathcal{C}(2) - \mathcal{X}_2 = \emptyset$)
- $\langle \mathcal{X}_1, \mathcal{X}_2, \mathcal{C}(3) - \mathcal{X}_3 \rangle$ (remove if $\mathcal{C}(3) - \mathcal{X}_3 = \emptyset$)
- $\langle \mathcal{C}(1) - \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle$ (remove if $\mathcal{C}(1) - \mathcal{Y}_1 = \emptyset$)
- $\langle \mathcal{Y}_1, \mathcal{C}(2) - \mathcal{Y}_2, \mathcal{Y}_3 \rangle$ (remove if $\mathcal{C}(2) - \mathcal{Y}_2 = \emptyset$)
- $\langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{C}(3) - \mathcal{Y}_3 \rangle$ (remove if $\mathcal{C}(3) - \mathcal{Y}_3 = \emptyset$)

However, here we might be overcounting. For example $\langle \mathcal{C}(1) - \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle$ could contain some portion from $\langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle$. Therefore, we will have to remove this portion. In other words, we need to express all the ways to go through $\langle \mathcal{C}(1) - \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle$ without going through $\langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle$.

RemoveSeqFromSeq($\langle \mathcal{C}(1) - \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle, \langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle$):

- $\langle \mathcal{C}(1) - \mathcal{X}_1 - \mathcal{Y}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle$
- $\langle \mathcal{C}(1) - \mathcal{X}_1, \mathcal{X}_2 - \mathcal{Y}_2, \mathcal{X}_3 \rangle$
- $\langle \mathcal{C}(1) - \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 - \mathcal{Y}_3 \rangle$

Consequently, we will have to perform the above steps for each combination. Therefore the solution for $\neg\phi$ is given by:

- RemoveSeqFromSeq($\langle \mathcal{C}(1) - \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle, \langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle$)
- RemoveSeqFromSeq($\langle \mathcal{X}_1, \mathcal{C}(2) - \mathcal{X}_2, \mathcal{X}_3 \rangle, \langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle$)
- RemoveSeqFromSeq($\langle \mathcal{X}_1, \mathcal{X}_2, \mathcal{C}(3) - \mathcal{X}_3 \rangle, \langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle$)
- RemoveSeqFromSeq($\langle \mathcal{C}(1) - \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle, \langle \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle$)
- RemoveSeqFromSeq($\langle \mathcal{Y}_1, \mathcal{C}(2) - \mathcal{Y}_2, \mathcal{Y}_3 \rangle, \langle \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle$)
- RemoveSeqFromSeq($\langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{C}(3) - \mathcal{Y}_3 \rangle, \langle \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle$)

Example 3: Let us now consider another case, when the solution for ϕ consists of three sequences, say sol $= \langle \langle \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle, \langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle, \langle \mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3 \rangle \rangle$. We can start as before and construct:

- $\langle \mathcal{C}(1) - \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle$ (remove if $\mathcal{C}(1) - \mathcal{X}_1 = \emptyset$)
- $\langle \mathcal{X}_1, \mathcal{C}(2) - \mathcal{X}_2, \mathcal{X}_3 \rangle$ (remove if $\mathcal{C}(2) - \mathcal{X}_2 = \emptyset$)
- $\langle \mathcal{X}_1, \mathcal{X}_2, \mathcal{C}(3) - \mathcal{X}_3 \rangle$ (remove if $\mathcal{C}(3) - \mathcal{X}_3 = \emptyset$)
- $\langle \mathcal{C}(1) - \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle$ (remove if $\mathcal{C}(1) - \mathcal{Y}_1 = \emptyset$)

- $\langle \mathcal{Y}_1, \mathcal{C}(2) - \mathcal{Y}_2, \mathcal{Y}_3 \rangle$ (remove if $\mathcal{C}(2) - \mathcal{Y}_2 = \emptyset$)
- $\langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{C}(3) - \mathcal{Y}_3 \rangle$ (remove if $\mathcal{C}(3) - \mathcal{Y}_3 = \emptyset$)
- $\langle \mathcal{C}(1) - \mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3 \rangle$ (remove if $\mathcal{C}(1) - \mathcal{Z}_1 = \emptyset$)
- $\langle \mathcal{Z}_1, \mathcal{C}(2) - \mathcal{Z}_2, \mathcal{Z}_3 \rangle$ (remove if $\mathcal{C}(2) - \mathcal{Z}_2 = \emptyset$)
- $\langle \mathcal{Z}_1, \mathcal{Z}_2, \mathcal{C}(3) - \mathcal{Z}_3 \rangle$ (remove if $\mathcal{C}(3) - \mathcal{Z}_3 = \emptyset$)

However, here we might be overcounting. For example $\langle \mathcal{C}(1) - \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle$ could contain some portion from $\langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle$ or $\langle \mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3 \rangle$. Therefore, we will have to remove this portion. In other words, we need to express all the ways to go through $\langle \mathcal{C}(1) - \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle$ without going through $\langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle$ or $\langle \mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3 \rangle$. In order to tackle this problem, we can first compute all the sequences that go through $\langle \mathcal{C}(1) - \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle$ without going through $\langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle$ as we did in example 2. Then, for each of those sequences we remove the $\langle \mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3 \rangle$ portion.

```
RemoveSeqsFromSeq(⟨C(1) − X1, X2, X3⟩, ⟨Y1, Y2, Y3⟩, ⟨Z1, Z2, Z3⟩):
    all = ∅
    seqs = RemoveSeqFromSeq(⟨C(1) − X1, X2, X3⟩, ⟨Y1, Y2, Y3⟩)
    for each seq ∈ seqs do
        tmp = RemoveSeqFromSeq(seq, ⟨Z1, Z2, Z3⟩)
        add each sequence in tmp to all
    return all
```

Thus, we will have to do perform the above step for each combination. Therefore the solution for $\neg\phi$ is given by:

- RemoveSeqsFromSeq($\langle \mathcal{C}(1) - \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle, \langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle, \langle \mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3 \rangle$)
- RemoveSeqsFromSeq($\langle \mathcal{X}_1, \mathcal{C}(2) - \mathcal{X}_2, \mathcal{X}_3 \rangle, \langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle, \langle \mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3 \rangle$)
- RemoveSeqsFromSeq($\langle \mathcal{X}_1, \mathcal{X}_2, \mathcal{C}(3) - \mathcal{X}_3 \rangle, \langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle, \langle \mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3 \rangle$)
- RemoveSeqsFromSeq($\langle \mathcal{C}(1) - \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle, \langle \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle, \langle \mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3 \rangle$)
- RemoveSeqsFromSeq($\langle \mathcal{Y}_1, \mathcal{C}(2) - \mathcal{Y}_2, \mathcal{Y}_3 \rangle, \langle \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle, \langle \mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3 \rangle$)
- RemoveSeqsFromSeq($\langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{C}(3) - \mathcal{Y}_3 \rangle, \langle \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle, \langle \mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3 \rangle$)
- RemoveSeqsFromSeq($\langle \mathcal{C}(1) - \mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3 \rangle, \langle \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle, \langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle$)
- RemoveSeqsFromSeq($\langle \mathcal{Z}_1, \mathcal{C}(2) - \mathcal{Z}_2, \mathcal{Z}_3 \rangle, \langle \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle, \langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle$)
- RemoveSeqsFromSeq($\langle \mathcal{Z}_1, \mathcal{Z}_2, \mathcal{C}(3) - \mathcal{Z}_3 \rangle, \langle \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 \rangle, \langle \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3 \rangle$)

Algorithm 7 Implementation of NOT : EVAL$(\neg\phi, \mathcal{C}, k_s, k_e)$

```
1: sol_1 ← EVAL(φ, C, k_s, k_e)
2: sol ← ∅
3: for r ← 0, ..., |sol_1| − 1 do
4:     negseqs ← NEGATESEQ(sol_1.seq_r, C, k_s, k_e)
5:     for ℓ ← 0, ..., |sol_1| − 1 do
6:         if r ≠ ℓ then
7:             negseqs ← REMOVESEQFROMSOL(negseqs, sol_1.seq_ℓ, k_s, k_e)
8:         end if
9:     end for
10:    sol.APPENDSOL(negseqs)
11: end for
12: return sol
```

NEGATESEQ$(\text{seq}, \mathcal{C}, k_s, k_e)$

```
1: sol ← ∅
2: for k ← k_s, ..., k_e do
3:     p ← POLYGONDIFFERENCE(C(k), seq[k])
4:     if p ≠ ∅ then
5:         tmp ← EMPTYSEQ(C, k_s, k_e)
6:         tmp.SETPOLYGON(k, p)
7:         sol.ADDSEQUENCE(tmp)
8:     end if
9: end for
10: return sol
```

REMOVESEQFROMSEQ$(\text{seqMain}, \text{seqRem}, k_s, k_e)$

```
1: sol ← ∅
2: for k ← k_s, ..., k_e do
3:     p ← POLYGONDIFFERENCE(seqMain[k], seqRem[k])
4:     if p ≠ ∅ then
5:         seq ← seqMain
6:         seq.SETPOLYGON(k, p)
7:         sol.ADDSEQUENCE(seq)
8:     end if
9: end for
10: return sol
```

REMOVESEQFROMSOL$(\text{solMain}, \text{seqRem}, k_s, k_e)$

```
1: sol ← ∅
2: for ℓ ← 0, ..., |solMain| − 1 do
3:     tmp ← REMOVESEQFROMSEQ(solMain.seq_ℓ, seqRem, k_s, k_e)
4:     if tmp ≠ ∅ then sol.APPENDSOL(tmp)
5: end for
6: return sol
```

We can generalize the solution for the *Not* operator, as outlined in Algorithm 7. First, we consider the algorithm NegateSeq which is responsible for the negation of a sequence. An empty sequence is first created $\langle \mathcal{C}(k_s), \ldots, \mathcal{C}(k_e) \rangle$ using EmptySeq subroutine. For the given time horizon, from time step k_s to time k_e, we calculate the difference of the reachable set at a specific time k with the polygon at the same time from the given sequence. For each non-empty difference, we append it to the solution space, as denoted in example 1. However, if the difference is empty then we do not consider it as part of the solution.

The second algorithm RemoveSeqFromSeq iterates again through the given time horizon and performs at each time step the difference of the respective polygons from both sequences. Similarly, if the difference is non-empty then we consider it in the construction of the new sequence for the solution. On the other hand, an empty difference plays no contribution in the solution space, as discussed in the above examples. Thus, we can simply ignore this case for the solution.

The third algorithm RemoveSeqFromSol is responsible for removing a sequence from a solution while considering the specified time range. The algorithm iterates over each sequence from *solMain* and calls the subroutine RemoveSeqFromSeq to remove the *seqRem* from it within the designated time interval. If the resulting modified sequence is not empty, it is appended to the solution scope. Finally, the algorithm returns the updated solution that excludes the sequences contained in *seqRem*.

We refer to lines 1-12 for the overall approach. First, we evaluate the formula and store its solution space to sol1. Afterwards, for each sequence in the solution, we perform its negation. Given the negation of each sequence, we remove each sequence from the solution scope of its negation. We perform these steps iteratively after we have successfully generated the negated sequences, which are appended to solution in line 10. The algorithm concludes with the return of the negation of the solution for the formula.

4. Predicates

In this chapter, we introduce the predicates used to compile the STL specifications. To better organize this chapter, we have classified the predicates into the following categories:

- Position-absolute: predicates that require a position to be within a specified longitudinal or lateral interval.
- Position-relative: predicates that demand a certain position relative to another object.
- Velocity-related: predicates that require longitudinal or lateral velocities to be within a particular range.

Category	**Predicate**
Position-Absolute	In_Lanelet Lat_Position_Less_Than Lat_Position_More_Than Lat_Position_Between
Position-Relative	Safe_Distance_To_Front_Obstacle Left_Safe_Lat_Distance Right_Safe_Lat_Distance Safe_Lat_Distance Within_Obstacle_Range
Velocity	Lon_Velocity_Less_Than Lon_Velocity_More_Than

Table 4.1.: Table listing predicates according to their category.

Within this thesis, we have devised a total of eleven predicates that are distributed across the categories mentioned above (see Table 4.1). The rest of this chapter continues introducing the position-absolute predicates in Section 4.1. Section 4.2 showcases the position-relative predicates. Finally, Section 4.3 presents predicates related to velocity.

The formulation of predicates is based on the principle of constructing the associated polygon, as discussed in Section 3.3. Since one might prefer to evaluate a specification according to the longitudinal or lateral dimension, defining the longitudinal or lateral polygon associated with a given predicate becomes crucial. The evaluation of the predicate is then performed as described again in Section 3.3, where a sequence is

constructed with the respective polygon for each dimension, followed by the rest of the driving corridor.

4.1. Position-absolute predicates

4.1.1. In Lanelet

A lanelet $\ell \in \mathcal{L}$ in CommonRoad [50] is defined as a basic navigable road segment bounded by left and right polylines [51], as illustrated in Figure 4.1. The total of these lanelets in a scenario comprise the lanelet network $\mathcal{L}$. Let the notation (x, y) denote the points of the two respective polylines in the global Cartesian frame. The left and right polylines are transformed into curvilinear coordinates in order to retrieve the minimum and maximum longitudinal and lateral coordinates. Therefore, the associated polygons $P^{lon}_{\text{In\_Lanelet}}$ and $P^{lat}_{\text{In\_Lanelet}}$ for the predicate In_Lanelet(ℓ) are defined as:

$$
\begin{aligned}
P^{lon}_{\text{In\_Lanelet}} = \{&(\min\{s' | \exists (x,y) \in \ell : \text{proj}_{(s,d)}(x,y) = (s',d')\}, v^{lon}_{min}),\\
&(\min\{s' | \exists (x,y) \in \ell : \text{proj}_{(s,d)}(x,y) = (s',d')\}, v^{lon}_{max}),\\
&(\max\{s' | \exists (x,y) \in \ell : \text{proj}_{(s,d)}(x,y) = (s',d')\}, v^{lon}_{max}),\\
&(\max\{s' | \exists (x,y) \in \ell : \text{proj}_{(s,d)}(x,y) = (s',d')\}, v^{lon}_{min})\}\\
P^{lat}_{\text{In\_Lanelet}} = \{&(\min\{d' | \exists (x,y) \in \ell : \text{proj}_{(s,d)}(x,y) = (s',d')\}, v^{lat}_{min}),\\
&(\min\{d' | \exists (x,y) \in \ell : \text{proj}_{(s,d)}(x,y) = (s',d')\}, v^{lat}_{max}),\\
&(\max\{d' | \exists (x,y) \in \ell : \text{proj}_{(s,d)}(x,y) = (s',d')\}, v^{lat}_{max}),\\
&(\max\{d' | \exists (x,y) \in \ell : \text{proj}_{(s,d)}(x,y) = (s',d')\}, v^{lat}_{min})\},
\end{aligned}
$$

where $v^{lon}_{min}, v^{lon}_{max}, v^{lat}_{min}, v^{lat}_{max}$ are the minimum and maximum velocities of the driving corridors in the longitudinal and lateral dimensions. To simplify the representation of more complex lanelets, we can decompose them into smaller rectangular lanelets, as depicted in Fig. 4.2. The same approach can then be used to generate the associated polygons of the smaller lanelets.

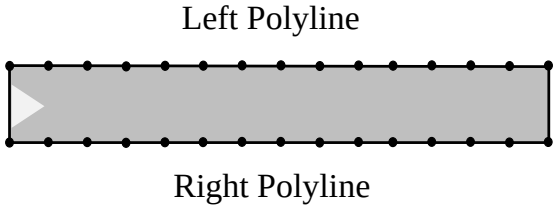


Figure 4.1.: Example of a lanelet bounded by the left and right polylines.

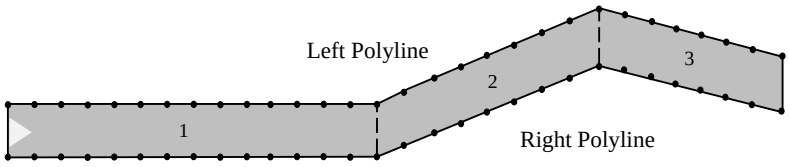


Figure 4.2.: Example of the decomposition of a lanelet into three smaller rectangular lanelets.

4.1.2. Lateral Position Less Than

This predicate puts a constraint on the lateral position of the ego vehicle. Let d^{lat}_{max} define the maximal boundary of the lateral position. The respective polygons for the predicate Lat_Position_Less_Than(d^{lat}_{max}) can then be constructed as follows:

$$P^{lon}_{\text{Lat\_Position\_Less\_Than}} = \{(p^{lon}_{min}, v^{lon}_{min}), (p^{lon}_{min}, v^{lon}_{max}), (p^{lon}_{max}, v^{lon}_{max}), (p^{lon}_{max}, v^{lon}_{min})\}$$
$$P^{lat}_{\text{Lat\_Position\_Less\_Than}} = \{(p^{lat}_{min}, v^{lat}_{min}), (p^{lat}_{min}, v^{lat}_{max}), (d^{lat}_{max}, v^{lat}_{max}), (d^{lat}_{max}, v^{lat}_{min})\},$$

where $p^{lon}_{min}, p^{lon}_{max}, p^{lat}_{min}, p^{lat}_{max}$ are the minimum and maximum positions of the driving corridors in the longitudinal and lateral dimensions.

4.1.3. Lateral Position More Than

This predicate is similar to the previous predicate. It constrains the lateral position by a minimal boundary denoted as d^{lat}_{min}. The associated polygons for the predicate Lat_Position_More_Than(d^{lat}_{min}) can then be formulated as follows:

$$P^{lon}_{\text{Lat\_Position\_More\_Than}} = \{(p^{lon}_{min}, v^{lon}_{min}), (p^{lon}_{min}, v^{lon}_{max}), (p^{lon}_{max}, v^{lon}_{max}), (p^{lon}_{max}, v^{lon}_{min})\}$$
$$P^{lat}_{\text{Lat\_Position\_More\_Than}} = \{(d^{lat}_{min}, v^{lat}_{min}), (d^{lat}_{min}, v^{lat}_{max}), (p^{lat}_{max}, v^{lat}_{max}), (p^{lat}_{max}, v^{lat}_{min})\}.$$

4.1.4. Lateral Position Between

This predicate requires the lateral position to be within a specified segment $[d^{lat}_{min}, d^{lat}_{max}]$. To generate the result of this predicate, we can easily leverage the outcomes of the two previous predicates, which also put constraints on the lateral position. Therefore, the predicate Lat_Position_Between($d^{lat}_{min}, d^{lat}_{max}$) can be expressed as Lat_Position_More_Than(d^{lat}_{min}) $\wedge$ Lat_Position_Less_Than(d^{lat}_{max}), with the corresponding polygons:

$$P^{lon}_{\text{Lat\_Position\_Between}} = P^{lon}_{\text{Lat\_Position\_More\_Than}} \cap P^{lon}_{\text{Lat\_Position\_Less\_Than}}$$
$$P^{lat}_{\text{Lat\_Position\_Between}} = P^{lat}_{\text{Lat\_Position\_More\_Than}} \cap P^{lat}_{\text{Lat\_Position\_Less\_Than}}.$$

4.2. Position-relative predicates

4.2.1. Safe Distance To Front Obstacle

The predicate Safe_Distance_To_Front_Obstacle(o, d) computes the region that is behind a preceeding obstacle o at a given distance d. Let o_k denote the state of the obstacle at a desired time step k for evaluation. We denote the center of the current position of the obstacle by (x_{o_k}, y_{o_k}) and ℓ_{o_k} the corresponding lanelet. We then construct the polygon $O_{o_k} = \{(x_{o_k} - d, y_{o_k} - \ell^{width}_{o_k}), (x_{o_k} - d, y_{o_k} + \ell^{width}_{o_k}), (x_{o_k} + d, y_{o_k} + \ell^{width}_{o_k}), (x_{o_k} + d, y_{o_k} - \ell^{width}_{o_k})\}$, which adds to the obstacle position the distance d and lanelet width $\ell^{width}_{o_k}$. The polygon is rotated to match the vehicle orientation, thus resulting in polygon O'_{o_k} (Figure 4.3). We then perform the set difference between the polygon representing the lanelet $O_{\ell_{o_k}}$ and O'_{o_k}.

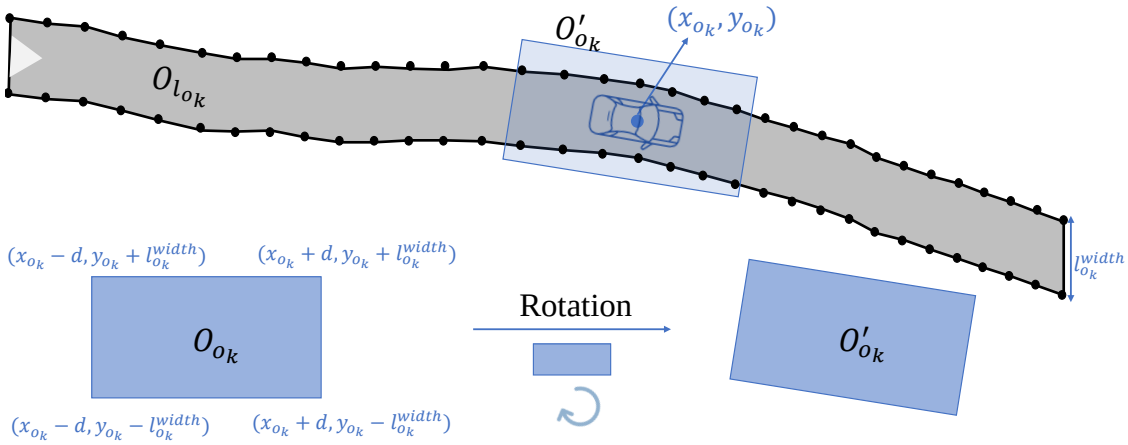


Figure 4.3.: Illustration presenting the construction of the polygon O'_{o_k}

The above difference $(O_{\ell_{o_k}} / O'_{o_k})$ can divide the lanelet into two parts namely $\ell_{o_{k_1}}$ and $\ell_{o_{k_2}}$, as shown in Figure 4.4. We select the lanelet that is the closest to the initial reachable set and denote it by $\ell_{o_{k_c}}$. The calculation of the associated polygons is performed similarly as in 4.1.1, by simply replacing the argument with $\ell_{o_{k_c}}$.

4.2.2. Left Safe Lateral Distance

The predicate Left_Safe_Lateral_Distance(o, d) generates a region with a minimum distance d to the left of the obstacle o. Let (x_o, y_o) be the center of the obstacle. We

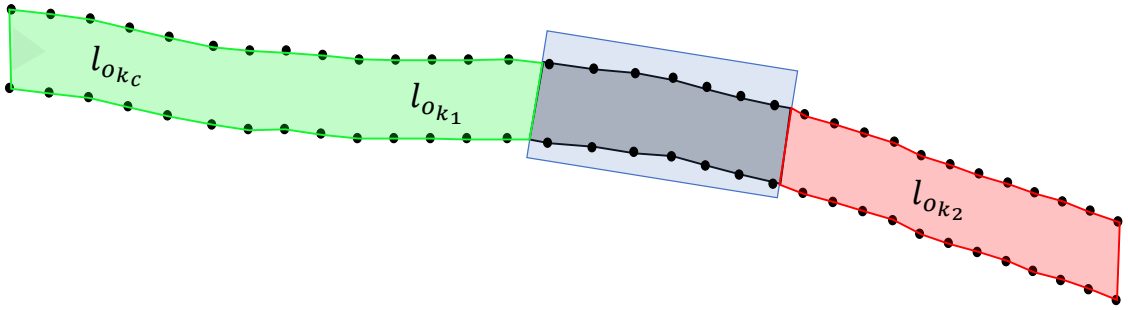


Figure 4.4.: Illustration presenting the difference $O_{\ell_{o_k}}/O'_{o_k}$ and the selection of portion $\ell_{o_{k_1}}$ as $\ell_{o_{k_c}}$

project these coordinates onto the curvilinear frame as $(s_o, d_o) = \text{proj}_{(s,d)}((x_o, y_o))$. We should ensure that the lateral deviance is at most $d_o - d$. The associated polygons are calculated as follows:

$$P^{lon}_{\text{Left\_Safe\_Lateral\_Distance}} = \{(p^{lon}_{min}, v^{lon}_{min}), (p^{lon}_{min}, v^{lon}_{max}), (p^{lon}_{max}, v^{lon}_{max}), (p^{lon}_{max}, v^{lon}_{min})\}$$
$$P^{lat}_{\text{Left\_Safe\_Lateral\_Distance}} = \{(p^{lat}_{min}, v^{lat}_{min}), (p^{lat}_{min}, v^{lat}_{max}), (d_o - d, v^{lat}_{max}), (d_o - d, v^{lat}_{min})\}.$$

4.2.3. Right Safe Lateral Distance

The predicate Right_Safe_Lateral_Distance(o, d) analogously computes a region with a minimum distance d on the right of the obstacle o. Again, let (x_o, y_o) denote the center position of the obstacle and $(s_o, d_o) = \text{proj}_{(s,d)}((x_o, y_o))$ the projection onto the curvilinear coordinate system. The associated polygons are generated as follows:

$$P^{lon}_{\text{Right\_Safe\_Lateral\_Distance}} = \{(p^{lon}_{min}, v^{lon}_{min}), (p^{lon}_{min}, v^{lon}_{max}), (p^{lon}_{max}, v^{lon}_{max}), (p^{lon}_{max}, v^{lon}_{min})\}$$
$$P^{lat}_{\text{Right\_Safe\_Lateral\_Distance}} = \{(d_o + d, v^{lat}_{min}), (d_o + d, v^{lat}_{max}), (p^{lat}_{max}, v^{lat}_{max}), (p^{lat}_{max}, v^{lat}_{min})\}.$$

4.2.4. Safe Lateral Distance

The computation of the predicate Safe_Lateral_Distance(o, d) can be easily achieved through the combination of the previous two predicates. A position is within the safe lateral distance if it is on the left safe lateral side of the obstacle or on the right safe lateral side of it. Therefore, the predicate Safe_Lateral_Distance(o, d) can be expressed as simply Left_Safe_Lateral_Distance$(o, d) \vee$ Right_Safe_Lateral_Distance(o, d), with the corresponding polygons:

$$P^{lon}_{\text{Safe\_Lateral\_Distance}} = P^{lon}_{\text{Left\_Safe\_Lateral\_Distance}} \cup P^{lon}_{\text{Right\_Safe\_Lateral\_Distance}}$$
$$P^{lat}_{\text{Safe\_Lateral\_Distance}} = P^{lat}_{\text{Left\_Safe\_Lateral\_Distance}} \cup P^{lat}_{\text{Right\_Safe\_Lateral\_Distance}}.$$

4.2.5. Within Obstacle Range

The predicate Within_Obstacle_Range(o, d) constitutes the area surrounding obstacle o with radius at most d. Let (x_o, y_o) denote the coordinates of the center of the obstacle o described in the global Cartesian frame. We project these coordinates into the curvilinear coordinate system $(s_o, d_o) = \text{proj}_{(s,d)}((x_o, y_o)$. Thus, the associated longitudinal and lateral polygons can be described in the subsequent manner:

$$P^{lon}_{\text{Within\_Obstacle\_Range}} = \{(s_o - d, v^{lon}_{min}), (s_o - d, v^{lon}_{max}), (s_o + d, v^{lon}_{max}), (s_o + d, v^{lon}_{min})\}$$
$$P^{lat}_{\text{Within\_Obstacle\_Range}} = \{(d_o - d, v^{lat}_{min}), (d_o - d, v^{lat}_{max}), (d_o + d, v^{lat}_{max}), (d_o + d, v^{lat}_{min})\}.$$

4.3. Velocity predicates

4.3.1. Longitudinal Velocity Less Than

The predicate Lon_Velocity_Less_Than(v_{max}) sets an upper bound to the longitudinal velocity equaling to v_{max}. Therefore, the associated polygons with this predicate can be defined as below:

$$P^{lon}_{\text{Lon\_Velocity\_Less\_Than}} = \{(p^{lon}_{min}, v^{lon}_{min}), (p^{lon}_{min}, v_{max}), (p^{lon}_{max}, v_{max}), (p^{lon}_{max}, v^{lon}_{min})\}$$
$$P^{lat}_{\text{Lon\_Velocity\_Less\_Than}} = \{(p^{lat}_{min}, v^{lat}_{min}), (p^{lat}_{min}, v^{lat}_{max}), (p^{lat}_{max}, v^{lat}_{max}), (p^{lat}_{max}, v^{lat}_{min})\}.$$

4.3.2. Longitudinal Velocity More Than

Analogously to the previous predicate, the predicate Lon_Velocity_More_Than(v_{min}) sets a lower bound to the longitudinal velocity equaling to v_{min}. Hence, the computation of the associated polygons with this predicate can be defined as follows:

$$P^{lon}_{\text{Lon\_Velocity\_More\_Than}} = \{(p^{lon}_{min}, v_{min}), (p^{lon}_{min}, v^{lon}_{max}), (p^{lon}_{max}, v^{lon}_{max}), (p^{lon}_{max}, v_{min})\}$$
$$P^{lat}_{\text{Lon\_Velocity\_More\_Than}} = \{(p^{lat}_{min}, v^{lat}_{min}), (p^{lat}_{min}, v^{lat}_{max}), (p^{lat}_{max}, v^{lat}_{max}), (p^{lat}_{max}, v^{lat}_{min})\}.$$

5. Robustness Measurement

In this chapter, we aim to explore different measurements for quantifying the satisfaction or violation of a given specification. The focus of this chapter is not only to capture the intricate dynamics of robustness but also to address possible limitations of existing metrics. We further introduce novel approaches for quantifying adherence to a specification while ensuring its soundness and monotonicity.

The geometric representation of a specification can facilitate the incorporation of geometric-based measurements. For example, in [58], authors leverage the signed version of Hausdorff distance to help reason about the satisfiability of predicates. While this is a common technique for measuring the distance between two given sets, it has limitations since it considers the maximum among the closest distances between points of two sets, which may need to be revised in order to also consider the proximity.

Another example showcasing the combination of geometry in model checking is the work presented in [59]. The authors propose a novel approach that introduces reachset temporal logic (RTL), defined by reachable sequences of states. During the model checking, the atomic predicates, represented as halfspace restrictions, are combined and expressed as polytopes. The satisfaction assessment is then performed by a simple inclusion check of the union of such polytopes.

As discussed in the previous chapters, we aim to transform the adherence to a given STL specification into a problem of capturing geometric dependencies between the reachable states of the system and the representation of a specification. Therefore the following sections explore different approaches to quantify the satisfaction or violation of a specification. First, Section 5.1 introduces the concept of minimum displacement as a measure of robustness. Section 5.2 presents a normalized weighted perimeter-based approach, which is further improved in Section 5.3. Finally, a joint optimization problem is introduced in Section 5.4 for combining the longitudinal and lateral robustness scores across all driving corridors.

5.1. Robustness based on displacement

Geometric-based measurements can provide numerous advantages while also presenting various limitations. Analysis based on the Hausdorff distance between sets can ignore the proximity between them, since it captures the maximum of the closest distances between the points of two sets. Analogously, the nearest distance (minimum of the closest distances between the points of two sets) can account for violation quantification but might miss capturing the adherence to a specification. Inclusion or intersection

checks can provide means to quantify the satisfiability by the number of points but may ignore the intersection depth or separation direction, which can provide stronger insights into the interdependencies between geometric objects.

In this section, we introduce a measurement for quantifying robustness based on the minimum displacement between two polygons. We leverage the concept of the Minimum-Translation Vector (MTV). This vector represents the minimum displacement required to separate two intersecting polygons.

In order to calculate the MTV between two polygons, we can first apply the separating axis theorem (SAT). This theorem states that for two non-intersecting convex objects, there exists an axis for which the projection of these geometries will not overlap [60]. Consequently, the intersection of two convex objects will result in the projection overlap of all axes.

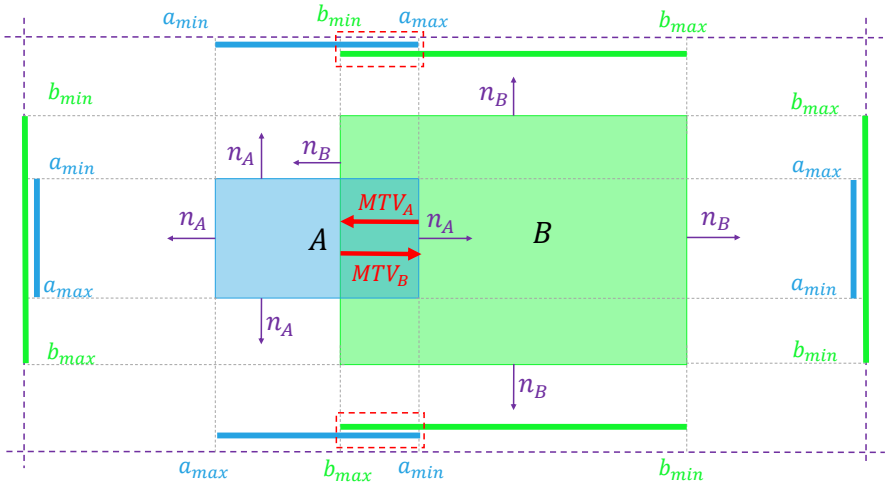


Figure 5.1.: Illustration presenting the calculation of the MTV. MTV_A is the minimum translation vector to separate polygon A and MTV_B is the minimum translation vector to separate polygon B.

An example of how to perform the SAT for two given polygons is illustrated in Fig. 5.1. For simplicity, we leverage an axis-aligned representation for the polygons, however our approach can be used for any convex polygons. First, the normals from each line of the polygons are extended, which will serve as projection axes. For each polygon A and B we denote by a_{min}, b_{min} and a_{max}, b_{max} the minimum and maximum projection points. The overlap is performed between segments $[a_{min}, a_{max}]$ and $[b_{min}, b_{max}]$. The calculation of MTV is simply realized through the selection of the axis, which contains the smallest overlap, that is, simultaneously, the magnitude of the vector (displacement).

$$P_{\mathcal{E}}^{lon} = \{(p_{min}^{lon}, v_{min}^{lon}), (p_{min}^{lon}, v_{max}^{lon}), (p_{max}^{lon}, v_{max}^{lon}), (p_{max}^{lon}, v_{min}^{lon})\} \tag{5.1}$$

$$P_{\mathcal{E}}^{lat} = \{(p_{min}^{lat}, v_{min}^{lat}), (p_{min}^{lat}, v_{max}^{lat}), (p_{max}^{lat}, v_{max}^{lat}), (p_{max}^{lat}, v_{min}^{lat})\}. \tag{5.2}$$

The calculation of the robustness measurement based on displacement is outlined in equation (5.3). In the intersection case between the two polygons, we return the magnitude of the MTV. On the other hand, if the polygons do not intersect, we return the minimum distance required to displace the polygons to an intersecting state, which is the minimum distance between the closest points.

The calculation of the robustness based on displacement is presented in Algorithm 8. The algorithm starts with the creation of the *Everything* corridor $\mathcal{E}$. This corridor contains at every time step the everything longitudinal polygon $P_{\mathcal{E}}^{lon}$ (5.1) and everything lateral polygon $P_{\mathcal{E}}^{lat}$ (5.2), in order to ensure fairness in the computation of the robustness across time steps and driving corridors. In simpler terms, since we retrieve the minimum robustness score from the elements of a sequence, we include *everything* polygons to avoid shadowing the elements that contain the associated polygons. However, this corridor can be customized according to preferences as it expresses only a way of representing a specification for times $k > k_s$ (see eq. (3.3)).

$$\rho_{Displacement}(p, c) = \begin{cases} \|MTV(p, c)\| & if\ p \cap c \neq \emptyset \\ -d_{min}(p, c) & if\ p \cap c = \emptyset \end{cases} \tag{5.3}$$

Algorithm 8 Implementation of robustness measure based on displacement

Measure from $-\infty$ to ∞ (negative failure, positive success).
For this measure, the solution should be computed by passing everything as the corridor.

RobustnessElement(P, c)

1: **return** $\min_{p \in P}(\rho_{Displacement}(p, c))$ ▷ Using equation 5.3

RobustnessSeq$(\text{seq}, \mathcal{C}, k_s, k_e)$

1: $a \leftarrow \infty$
2: **for** $k \leftarrow k_s, \ldots, k_e$ **do**
3: $\quad a \leftarrow \min(a, \text{RobustnessElement}(\text{seq}[k], \mathcal{C}(k)))$
4: **end for**
5: **return** a

RobustnessSol$(\text{sol}, \mathcal{C}, k_s, k_e)$

1: $a \leftarrow -\infty$
2: **for** $r \leftarrow 0, \ldots, |\text{sol}| - 1$ **do**
3: $\quad a \leftarrow \max(a, \text{RobustnessSeq}(\text{sol.seq}_r, \mathcal{C}, k_s, k_e))$
4: **end for**
5: **return** a

RobustnessFormula$(\phi, \mathcal{C}, k_s, k_e)$

1: $\mathcal{E} \leftarrow \text{CreateCorridor}(\text{everything}, k_s, k_e)$
2: $\text{sol} \leftarrow \text{eval}(\phi, \mathcal{E}, k_s, k_e)$
3: **return** RobustnessSol$(\text{sol}, \mathcal{C}, k_s, k_e)$

The subroutine RobustnessElement returns the robustness score for an element

according to equation (5.3). The minimum robustness is returned across a sequence from the function RobustnessSeq. This is because the elements within a sequence inherently imply a conjunction among them. In contrast, the function RobustnessSol returns the maximum robustness score among sequences since their ordering implicitly reflects a disjunction across the alternatives to satisfy a formula.

5.2. Robustness based on a normalized weighted perimeter

A problem with the robustness measurement presented in Section 5.1 stems from its inherent treatment of two polygons as singular points. In other words, we are only interested in the two closest or deepest points, thus disregarding the overall geometric shape of the given polygons. In order to tackle this issue, we leverage a normalized weighted perimeter-based approach, which considers the underlying geometry of the objects and weights their perimeter according to the distance between them.

The weighted perimeter can be conceptualized as a measurement of the boundary of a set that considers a weight function designating different values to different points [61]. Examples of such perimeters can be a Gaussian-weighted perimeter, which assigns weights to different parts of a perimeter based on a Gaussian distribution [62]. Points or segments closer to the center of the distribution are assigned higher weights than those farther away. In our approach, we customize the implementation of the weighted perimeter to offer a refined and accurate measurement of the robustness. To facilitate the implementation of this measurement, we separate the calculation of the weighted perimeter independently according to the intersection status between two polygons. We will refer to the non-intersecting case as the weighted outer perimeter; analogously, we will refer to the intersecting case as the weighted inner perimeter.

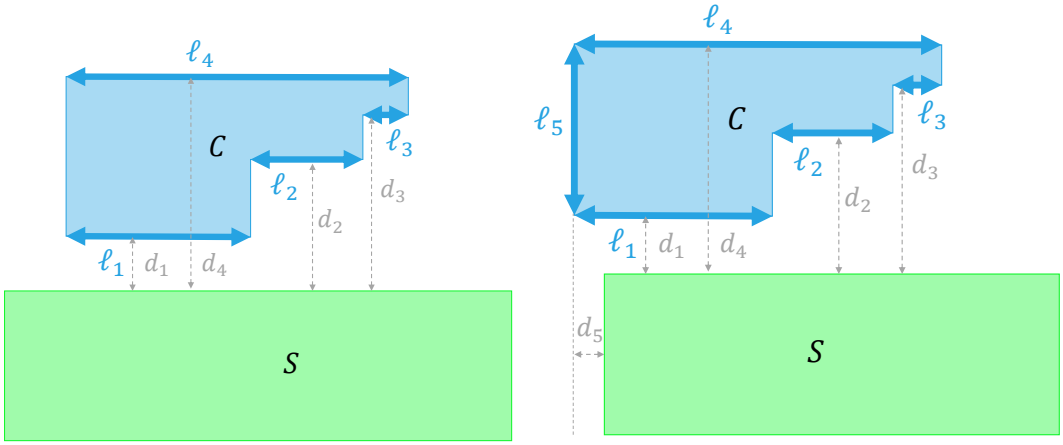
5.2.1. Weighted Outer Perimeter

The calculation of the robustness score using the weighted outer perimeter is introduced in equation (5.4). For each line segment $\ell \in c$, the score is determined by the ratio of the length of the line segment to the total perimeter of c, weighted by the minimum distance of the line segment to any polygon in P (see ShortestDistance in Algorithm 12). Therefore, we quantify the contribution of each line segment to the overall score based on the length and proximity to polygons in P.

The approach for calculating the normalized weighted outer perimeter is outlined in Algorithm 12. The function WeightedOuterPerimeter takes a polygon p, and a list of specification polygons $\mathcal{S}$ and iteratively calculates the perimeter. For every line $\ell_p \in p$, which is not between any two lines of the same orientation (horizontal or vertical) of any specification polygons (see IsBetweenLines in Algorithm 12 and IsBetween in Algorithm 9), we normalize it according to the perimeter of p and weight it according to the shortest distance to any of the provided specification polygons. Figure 5.2 illustrates the weighted outer perimeter, where the considered lines are indexed as ℓ_i with the

corresponding distances d_i ($i \in \mathbb{N}$). In the second scenario the line ℓ_5 is considered into the calculations since it does not lie within any two vertical lines from polygon S, in contrast to the first scenario. In an informal context, the reason for this distinction is that a line from polygon C between two lines of the same orientation from polygon S, plays no contribution in the violation of the specification (since they are contained within the boundaries of S).

$$\rho_{WOP}(c, P) = \sum_{\ell \in c} \left(\frac{\ell.\text{length}}{Perimeter(c)} \min_{p \in P} d_\ell(\ell, p) \right) \tag{5.4}$$



(a) First scenario

(b) Second scenario

Figure 5.2.: Illustration showcasing the weighted outer perimeter. The lines considered in the calculation are presented with double-headed arrows, together with their respective distances to the specification polygon.

Algorithm 9 Implementation of IsBetween function.

IsBetween(ℓ, ℓ_a, ℓ_b)

```
if ℓ.IsVertical then
    if (ℓ_a.start.x ≤ ℓ.start.x ≤ ℓ_b.start.x) or (ℓ_b.start.x ≤ ℓ.start.x ≤ ℓ_a.start.x) then
        return true
    else return false
    end if
else
    if (ℓ_a.start.y ≤ ℓ.start.y ≤ ℓ_b.start.y) or (ℓ_b.start.y ≤ ℓ.start.y ≤ ℓ_a.start.y) then
        return true
    else return false
    end if
end if
```

5.2.2. Weighted Inner Perimeter

The calculation of the robustness score using the weighted inner perimeter is introduced in equation (5.5). $\text{Inner}(c, P)$ and $\text{Outer}(c, P)$ contain the inner and outer line segments of polygon c with respect to polygons in P. The term $d_{MTV}(c, p, \ell)$ refers to the distance of line $\ell \in c$ to the line of polygon p inferred by the MTV between c and p (see InnerValues in Algorithm 13 and Fig. 5.3). Additionally, the distance $d_{MTV}(c, p, \ell) = 0$ if $c \cap p = \emptyset$. It should be noted that C can be dived during intersection into smaller polygons. For this reason, we introduce the outer sum iterating over all $c \in C$.

The approach for calculating the normalized weighted inner perimeter is outlined in Algorithm 13. We leverage an axis-aligned approach to simplify the computations to represent the polygons. In order to help better understand the algorithm, we illustrate the approach in Figure 5.3. The reachset-representing polygon intersects with the specification polygon, generating C_1 and C_2, which allows for the computation of the respective MTVs.

$$\rho_{WIP}(C, P) = \sum_{c \in C} \left(\sum_{\substack{\ell \in \text{Inner}(c,P) \\ \ell \in \text{Outer}(c,P)}} \frac{\ell.\text{length}}{Perimeter_{\Sigma}(C)} \max_{p \in P} d_{MTV}(c, p, \ell) \right), \tag{5.5}$$

where $Perimeter_{\Sigma}(C) = \sum_{c \in C} Perimeter(c)$.

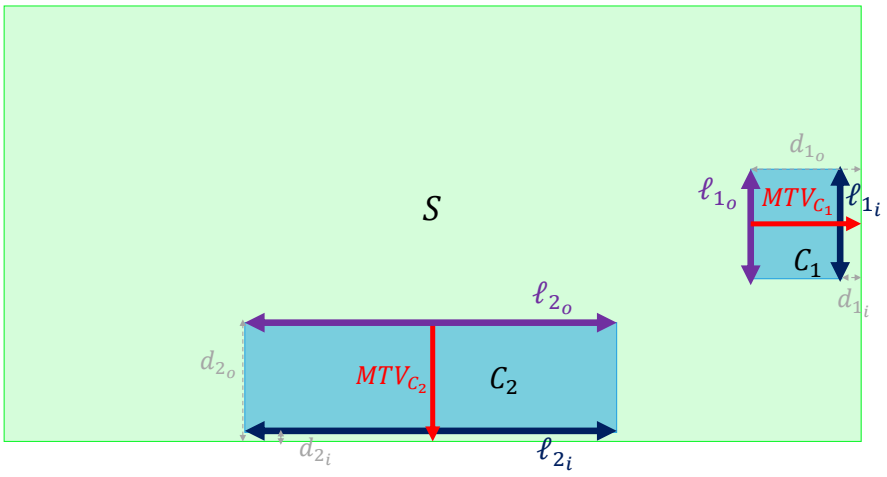


Figure 5.3.: Illustration presenting the weighted inner perimeter. The lines considered in the calculation are presented by a double-headed arrow. The respective MTVs for polygons C_1 and C_2 are denoted by MTV_{C_1} and MTV_{C_2}.

The function InnerValues returns for every polygon c, e.g., C_1 and C_2 in Figure 5.3, the so-called *inner values* with respect to a specification polygon s, i.e., S. The MTVs help us denote the inner and outer lines of a polygon. An inner line c_ℓ (presented by the lines ℓ_{1_i}, ℓ_{2_i} in dark blue in Fig. 5.3) is the perpendicular line to the MTV closest to

the specification polygon and an outer line f_ℓ (presented by the lines ℓ_{1_o}, ℓ_{2_o} in violet in Fig. 5.3) is the perpendicular line to the MTV furthest to the specification polygon. The algorithm InnerValues represents the length of the inner line by i_ℓ (denoted by the dark blue double-headed arrows) and its distance to the specification polygon by i_d (denoted by d_{1_i} and d_{2_i} in Fig. 5.3). Analogously, o_ℓ is the length of the outer line (denoted by the purple double-headed arrows) and o_d the distance to the specification polygon (denoted by d_{1_o} and d_{2_o} in Fig. 5.3).

The function WeightedInnerPerimeter adds for each polygon in line 8 the respective *inner values*. The total perimeter is calculated in line 13, which is then used to normalize every line as shown in lines 17 and 18. The lines are then weighted by their respective distances to the specification polygon. The algorithm concludes in line 20 by returning the normalized weighted inner perimeter P_{inner}. The rest of the function for reporting the robustness of a solution remain unchanged.

Algorithm 10 Implementation of robustness measure of an element based on 5.2

RobustnessElement(P, c)
  if $p \cap c \neq \emptyset$ **then**
    robustness $\leftarrow \rho_{WIP}(c, P)$ ▷ WeightedInnerPerimeter in Algorithm 13
  else
    robustness $\leftarrow -\rho_{WOP}(c, P)$ ▷ WeightedOuterPerimeter in Algorithm 12
  end if
  return robustness

The implementation of the overall robustness measurement is similar to Algorithm 8. We only have to change the implementation regarding the robustness of an element. Therefore, the function RobustnessElement is updated with the new function RobustnessElement of Algorithm 10.

5.3. Robustness based on a normalized weighted perimeter per side

The robustness measurement presented in the previous section normalizes the inner and outer perimeter according to the perimeters of the considered polygons. Consequently, this leads to considering lines that might not contribute to improving the satisfaction or worsening of a violation. For this reason, we would like to disregard such lines and only focus on the ones that can contribute to the adherence or violation of a specification.

$$\rho_{SWOP}(c, P) = \sum_{\ell \in C} \left(\frac{\ell.\text{length}}{\sum\limits_{\substack{\ell' \in C \\ \ell'.\text{orientation}=\ell.\text{orientation}}} \ell'.\text{length}} \min_{p \in P} d_\ell(\ell, p) \right) \tag{5.6}$$

The calculation of the robustness score for the normalized outer perimeter per side is presented in equation (5.6). The difference from the previous method is that we normalize according to the total length of line segments with the same orientation. Similarly, the equation (5.7) normalizes the inner and outer lines by half of the total length of the considered line segments. We consider the half sum since the total sum would be twice the total length of the inner or outer line segments.

$$\rho_{SWIP}(C,P) = \sum_{c\in C}\left(\sum_{\substack{\ell\in\text{Inner}(c,P)\\ \ell\in\text{Outer}(c,P)}}\left(\frac{\ell.\text{length}}{\frac{1}{2}\sum\limits_{\substack{\ell'\in\text{Inner}_C(C,P)\\ \ell'\in\text{Outer}_C(C,P)}}\ell'.\text{length}}\max_{p\in P} d_{MTV}(c,p,\ell)\right)\right), \quad (5.7)$$

where $\text{Inner}_C(C,P) = \bigcup\limits_{c\in C}\text{Inner}(c,P)$ and $\text{Outer}_C(C,P) = \bigcup\limits_{c\in C}\text{Outer}(c,P)$.

The implementation of the overall robustness measurement follows the outline presented in Algorithm 8. We solely modify the implementation concerning an element's robustness. Hence, the function RobustnessElement is updated with the new function RobustnessElement of Algorithm 11.

The function SidesWeightedInnerPerimeter calculates the inner perimeter as in Section. 5.2.2, but normalizes the inner and outer lines according to the total length of all inner or outer lines. Analogously, the function SidesWeightedOuterPerimeter computes the outer perimeter as in Section. 5.2.1, but uses the subroutine NormalizeLine for normalization. This subroutine returns the division of a line's length by half of the total sum of vertical or horizontal lines' lengths (depending on the line's orientation).

Algorithm 11 Implementation of robustness measure of an element based on 5.3

RobustnessElement(P, c)
  if $p \cap c \neq \emptyset$ **then**
    robustness $\leftarrow \rho_{SWIP}(c,P)$ ▷ SidesWeightedInnerPerimeter in Algorithm 14
  else
    robustness $\leftarrow -\rho_{SWOP}(c,P)$ ▷ SidesWeightedOuterPerimeter in Algorithm 14
  end if
  return robustness

5.4. Combined robustness

We propose a joint optimization problem for combining the robustness scores reported in the longitudinal and lateral dimensions. Given a total of n driving corridors with the respective scores denoted by $\boldsymbol{\rho_{lon}}$ and $\boldsymbol{\rho_{lat}}$, where $\boldsymbol{\rho_{lon_i}}$ and $\boldsymbol{\rho_{lat_i}}$ are the measurements for the i-th corridor, we aim to discover the optimal value for w^* in order to minimize

the collective Chebyshev distances across the given data:

$$w^* = \arg\min_{w \in [0,1]} \sum_{i=1}^{n} (\max(|\min(w \cdot \boldsymbol{\rho}_{lon_i}, (1-w) \cdot \boldsymbol{\rho}_{lat_i}) - \boldsymbol{\rho}_{lon_i}|) + \max(|\min(w \cdot \boldsymbol{\rho}_{lon_i}, (1-w) \cdot \boldsymbol{\rho}_{lat_i}) - \boldsymbol{\rho}_{lat_i}|)) \tag{5.8}$$

We use the Chebyshev distance since it primarily focuses on the most significant deviation from the objective function. The *soundness* property is still preserved since the $min(w^* \cdot \square, (1-w^*) \cdot \triangle)$ returns a negative value if at least one of its entries $(\square, \triangle)$ are negative. If both entries are positive, the function will return a positive value. The objective function is a sum of terms involving absolute differences between certain expressions. The expressions include a combination of w, $\boldsymbol{\rho}_{lon_i}$ and $\boldsymbol{\rho}_{lat_i}$. As w varies between 0 and 1, the terms involving w are linear. Due to the convex nature of the absolute value function, both the absolute differences and the summation preserve the convexity property, consequently implying *monotocity*, which is an important property to guarantee the stability and predictability of the optimization procedure.

Algorithm 12 Implementation of normalized weighted outer perimeter

Measure from 0 to ∞

WeightedOuterPerimeter$(p, \mathcal{S})$

$P_{\text{outer}} \leftarrow 0$
for each ℓ_p **in** p **do**
    $P_{\text{outer}} \leftarrow P_{\text{outer}} + (\frac{\ell_p.\text{length}}{p.\text{length}} \cdot \text{ShortestDistance}(\ell_p, \mathcal{S}))$
end for
return P_{outer}.

ShortestDistance$(\ell, \mathcal{S})$

$distance_{\text{shortest}} \leftarrow \infty$
$\mathcal{V}, \mathcal{H} \leftarrow \text{GetLines}(\mathcal{S})$
if ℓ.IsVertical **then** $\mathcal{L} \leftarrow \mathcal{V}$
else $\mathcal{L} \leftarrow \mathcal{H}$
end if
$distance \leftarrow \text{GetDistanceFromLines}(\ell, \mathcal{L})$
if not IsBetweenLines$(\ell, \mathcal{L})$ **and** $distance < distance_{\text{shortest}}$ **then**
    $distance_{\text{shortest}} \leftarrow distance$
    $consider \leftarrow True$
end if
if consider **then return** $distance_{\text{shortest}}$
else return 0
end if

GetDistanceFromLines$(\ell, \mathcal{L})$

$distance \leftarrow \infty$
for each $\ell_{\mathcal{L}}$ **in** $\mathcal{L}$ **do**
    $distance_{\ell} \leftarrow \ell.\text{distance}(\ell_{\mathcal{L}})$
    if $distance_{\ell} < distance$ **then**
        $distance \leftarrow distance_{\ell}$
    end if
end for
return $distance$

IsBetweenLines$(\ell, \mathcal{L})$

for $i \leftarrow 0, \ldots, |\mathcal{L}| - 2$ **do**
    for $j \leftarrow i + 1, \ldots, |\mathcal{L}| - 1$ **do**
        if IsBetween$(\ell, \mathcal{L}[i], \mathcal{L}[j])$ **then**
            return true
        end if
    end for
end for
return false

Algorithm 13 Implementation of normalized weighted inner perimeter

Measure from 0 to ∞

WeightedInnerPerimeter($\mathcal{P}, \mathcal{S}$)

$\mathcal{I}_\ell, \mathcal{I}_d, \mathcal{O}_\ell, \mathcal{O}_d, \mathcal{C}_\ell, \mathcal{F}_\ell \leftarrow \emptyset$
for each p **in** $\mathcal{P}$ **do**
  for each s **in** $\mathcal{S}$ **do**
    $c \leftarrow p$.Intersect(s)
    if $c \neq \emptyset$ **then**
      $\mu \leftarrow \text{MTV}(c, s)$
      $i_\ell, i_d, o_\ell, o_d, c_\ell, f_\ell \leftarrow$ InnerValues(c, s, μ)
      $\mathcal{I}_\ell$.APPEND(i_ℓ), $\mathcal{I}_d$.APPEND(i_d), $\mathcal{O}_\ell$.APPEND(o_ℓ), $\mathcal{O}_d$.APPEND(o_d), $\mathcal{C}_\ell$.APPEND(c_ℓ), $\mathcal{F}_\ell$.APPEND(f_ℓ),
    end if
  end for
end for
$P \leftarrow 0$
for $i \leftarrow 0, \ldots, |\mathcal{I}_\ell| - 1$ **do** $P \leftarrow P + (\mathcal{I}_\ell[i] + \mathcal{O}_\ell[i] + 2 \cdot \mathcal{C}_\ell[i]$.Distance($\mathcal{F}_\ell[i]$))
end for
$P_{\text{inner}} \leftarrow 0$
for $i \leftarrow 0, \ldots, |\mathcal{I}_\ell| - 1$ **do**
  $P_{\text{inner}} \leftarrow P_{\text{inner}} + (\frac{\mathcal{I}_\ell[i]}{P} \cdot \mathcal{I}_d[i])$
  $P_{\text{inner}} \leftarrow P_{\text{inner}} + (\frac{\mathcal{O}_\ell[i]}{P} \cdot \mathcal{O}_d[i])$
end for
return P_{inner}

InnerValues(c, s, μ)

if ABS($\mu[0]$) == 0 **then** $\mathcal{L}_c \leftarrow$ HorizontalLines(c), $\mathcal{L}_s \leftarrow$ HorizontalLines(s)
else $\mathcal{L}_c \leftarrow$ VerticalLines(c), $\mathcal{L}_s \leftarrow$ VerticalLines(s)
end if
$c_c, s_c \leftarrow NULL$
$distance_{min} \leftarrow \infty$
for each ℓ_c **in** c **do**
  for each ℓ_s **in** s **do**
    $distance_{s-c} \leftarrow \ell_c$.Distance($\ell_s$)
    if $distance_{s-c} < distance_{min}$ **then**
      $c_c \leftarrow \ell_c$, $s_c \leftarrow \ell_s$, $distance_{min} \leftarrow distance_{s-c}$
    end if
  end for
end for
$i_\ell \leftarrow c_c$.length, $i_d \leftarrow c_c$.Distance(s_c), $c_\ell \leftarrow c_c$, $f_\ell \leftarrow$ GetOtherLine($c_c, \mathcal{L}_c$), $o_\ell \leftarrow f_\ell$.length, $o_d \leftarrow f_\ell$.Distance(s_c)
return $i_\ell, i_d, o_\ell, o_d, c_\ell, f_\ell$

Algorithm 14 Implementation of sides normalized weighted outer and inner perimeter Measure from 0 to ∞

SidesWeightedOuterPerimeter$(p, \mathcal{S})$

$P_{\text{outer}} \leftarrow 0$
$\mathcal{V}, \mathcal{H} \leftarrow$ GetLines(p)
$\mathcal{V}_{\text{sum}} \leftarrow$ LengthsSummation$(\mathcal{V})$
$\mathcal{H}_{\text{sum}} \leftarrow$ LengthsSummation$(\mathcal{H})$
for each ℓ_p **in** p **do**
    $P_{\text{outer}} \leftarrow P_{\text{outer}} + (\text{NormalizeLine}(\ell_p, \mathcal{V}_{\text{sum}}, \mathcal{H}_{\text{sum}}) \cdot \text{ShortestDistance}(\ell_p, \mathcal{S}))$
end for
return P_{outer}.

SidesWeightedInnerPerimeter$(\mathcal{P}, \mathcal{S})$

$\mathcal{I}_\ell, \mathcal{I}_d, \mathcal{O}_\ell, \mathcal{O}_d \leftarrow \emptyset$
for each p **in** $\mathcal{P}$ **do**
    for each s **in** $\mathcal{S}$ **do**
        $c \leftarrow p$.Intersect(s)
        if $c \neq \emptyset$ **then**
            $\mu \leftarrow \text{MTV}(c, s)$
            $i_\ell, i_d, o_\ell, o_d, c_\ell, f_\ell \leftarrow$ InnerValues(c, s, μ)
            $\mathcal{I}_\ell$.APPEND(i_ℓ), $\mathcal{I}_d$.APPEND(i_d), $\mathcal{O}_\ell$.APPEND(o_ℓ), $\mathcal{O}_d$.APPEND(o_d),
        end if
    end for
end for
$I, O \leftarrow 0$
for $i \leftarrow 0, \ldots, |I_\ell| - 1$ **do** $I \leftarrow I + \mathcal{I}_\ell[i]$, $O \leftarrow O + \mathcal{O}_\ell[i]$
end for
$P_{\text{inner}} \leftarrow 0$
for $i \leftarrow 0, \ldots, |\mathcal{I}_\ell| - 1$ **do**
    $P_{\text{inner}} \leftarrow P_{\text{inner}} + (\frac{\mathcal{I}_\ell[i]}{I} \cdot \mathcal{I}_d[i])$
    $P_{\text{inner}} \leftarrow P_{\text{inner}} + (\frac{\mathcal{O}_\ell[i]}{O} \cdot \mathcal{O}_d[i])$
end for
return P_{inner}

6. Experimental Results

This chapter presents the experimental results of the considered approach. We start by exploring the correctness and the preservation of the monotonicity across the robustness measurements. To this end, we propose six specifications divided into non-temporal and temporal expressions. Additionally, we realize a ranking of the driving corridors according to their adherence to the given specification. Lastly, we discuss the computational running times of our approach as the time frame and formula size increase.

In order to help with reference to the various robustness measurements in the following plots, we employ the following keywords: *Displacement* - refers to the robustness based on displacement, *Normalized W-Perimeter* - refers to the robustness based on the normalized weighted perimeter, *Normalized S-W-Perimer* - refers to the robustness based on the normalized sides weighted perimeter.

The rest of this chapter is organized as follows. Section 6.1 presents the experimental results by considering non-temporal specifications. Section 6.2 advances by examining more complex temporal specifications. Finally, Section 6.3 discusses the computational efficiency of our approach.

6.1. Evaluation of non-temporal specifications

6.1.1. Ensuring a consistent lateral position

This particular requirement demands the consistent preservation of a predetermined lateral position. Initially, we calculate two driving corridors based on the scenario *ZAM_Intersection-1_1_T-1*. Subsequently, we assess the robustness measurement for these two designated corridors, necessitating the maintenance of a lateral position ranging from -2 to 0 concerning the provided reference path, formally expressed as $\phi = \mathit{lat\_position\_between}(-2, 0)$.

In this given scenario, we compute two driving corridors. Of particular interest is the trend of the robustness measurement at three distinct time steps: 1, 14, and 28, as illustrated in Fig. 6.1 together with the area where the lateral position ranges from -2 to 0 (shown in green). Concurrently, we present the recorded robustness measurements in Fig. 6.2. The observations show that the robustness trend remains consistent until time step 14 due to the equal lateral distance from the given reference path. As the driving corridors begin to differentiate, a notable change occurs from time step 15 onwards. At time step 28, as depicted in figures 6.1e and 6.1f, driving corridor one encompasses a more significant percentage of the specification region (highlighted in green), and

the reachable states (drawn in light blue) are generally close in comparison to driving corridor zero.

Consequently, higher robustness is recorded across all measurements for driving corridor one, as its reachable states are closer and cover a higher proportion of the specification area. Furthermore, it can be observed that the combined robustness has preserved the soundness and monotonicity across all three measurements. Since the lateral robustness is the predominating robustness due to the nature of the specification, we can see this aspect closely mirrored in the combined robustness.

6.1.2. Achieving high velocities

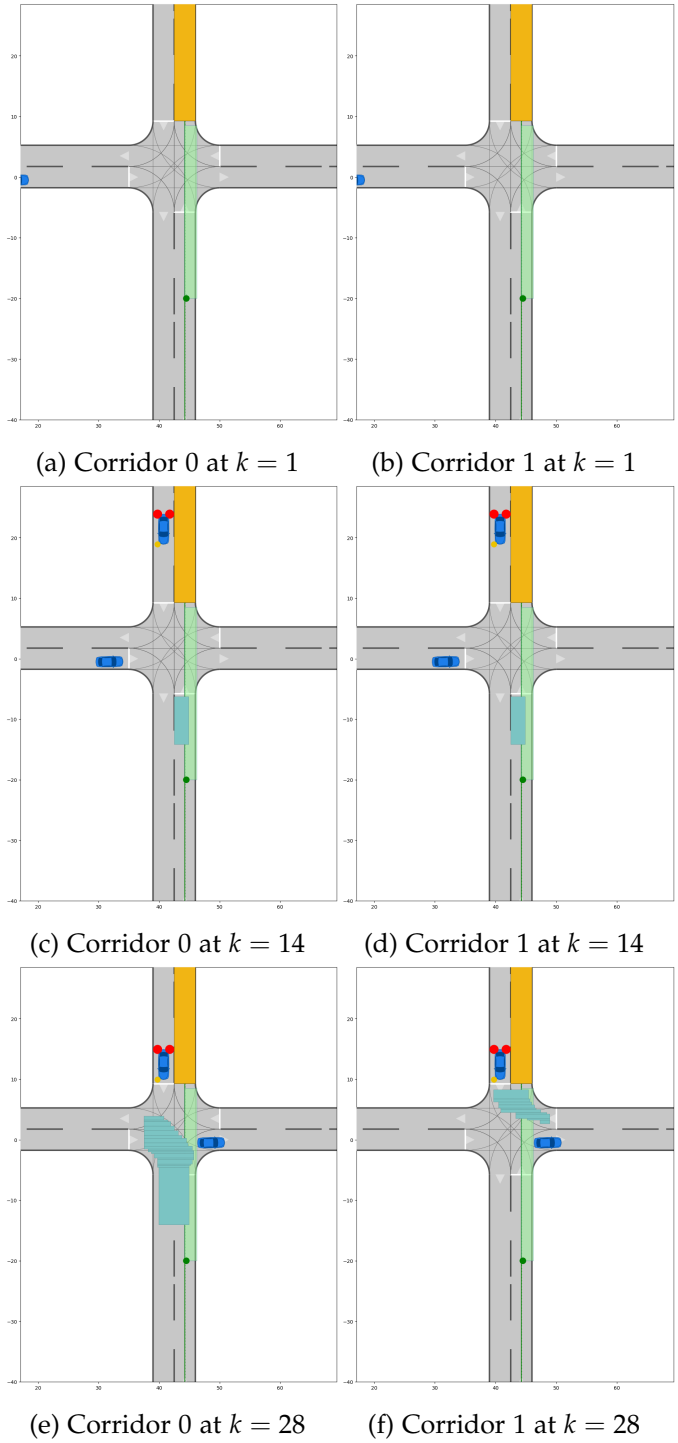
This specification captures the concept of a driving corridor achieving high velocities. Analyzing the previous example reveals that driving corridor zero exhibits a slight leftward deviation compared to driving corridor one. Based on this observation, a driving corridor deviating from a straight path might result in lower velocities for its reachable states, in contrast to a driving corridor that follows a straight path.

We formulate the specification $\phi = \textit{lon\_velocity\_more\_than(15)}$ demanding a velocity surpassing 15 m/s (54 km/h), which is considered sufficiently high given that the maximum attainable velocity in this scenario is approximately 15.8 m/s (56.88 km/h). With precision, we anticipate that the robustness measurements will yield a higher value for driving corridor one due to its adherence to a straightforward path. To visualize this aspect, we present in Fig. 6.3 the two driving corridors from scenario *ZAM_Intersection-1_1_T-1* across the position and velocity domains, at time steps 1, 20 and 28. Initially, both driving corridors exhibit small, similar reachable sets that are equally distant from meeting the velocity criteria. However, at time step 20, we observe a divergence between the driving corridors. The second driving corridor successfully meets the velocity criterion of 15 m/s, while the first driving corridor barely surpasses the limit at 12 m/s. In the final time steps, driving corridor zero displays reduced deviation and increasing velocity, demonstrated in the *Displacement* and *Normalized S-W-Perimeter* robustness measurements. Conversely, driving corridor one reaches a plateau and maintains the velocity limit without further escalation.

The *Normalized W-Perimeter* measurement fails to capture the slight increase in velocity at the last time steps for driving corridor zero due to its normalization approach. The *Displacement* measurement falsely pictures achieving a high velocity from the whole reachable sets. On the other hand, *Normalized S-W-Perimeter* captures the slight increase in velocity during the last time steps due to the small number of states that can achieve such high velocities.

6.1.3. Maintaining a distance from a moving obstacle

This requirement entails the maintenance of a fixed lateral distance from a dynamic obstacle. Consequently, we establish the specification $\phi = \textit{safe\_lat\_distance}(\textit{obstacle}, \textit{distance})$.



(a) Corridor 0 at $k = 1$ (b) Corridor 1 at $k = 1$

(c) Corridor 0 at $k = 14$ (d) Corridor 1 at $k = 14$

(e) Corridor 0 at $k = 28$ (f) Corridor 1 at $k = 28$

Figure 6.1.: Two driving corridors from scenario *ZAM_Intersection-1_1_T-1* shown at time steps 1, 14 and 28.

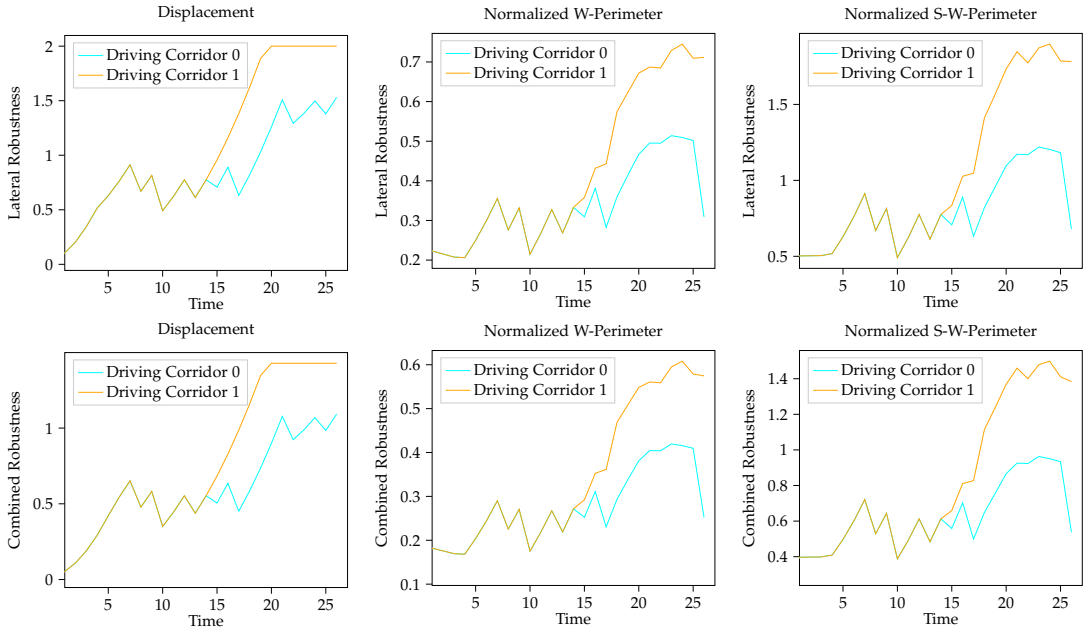
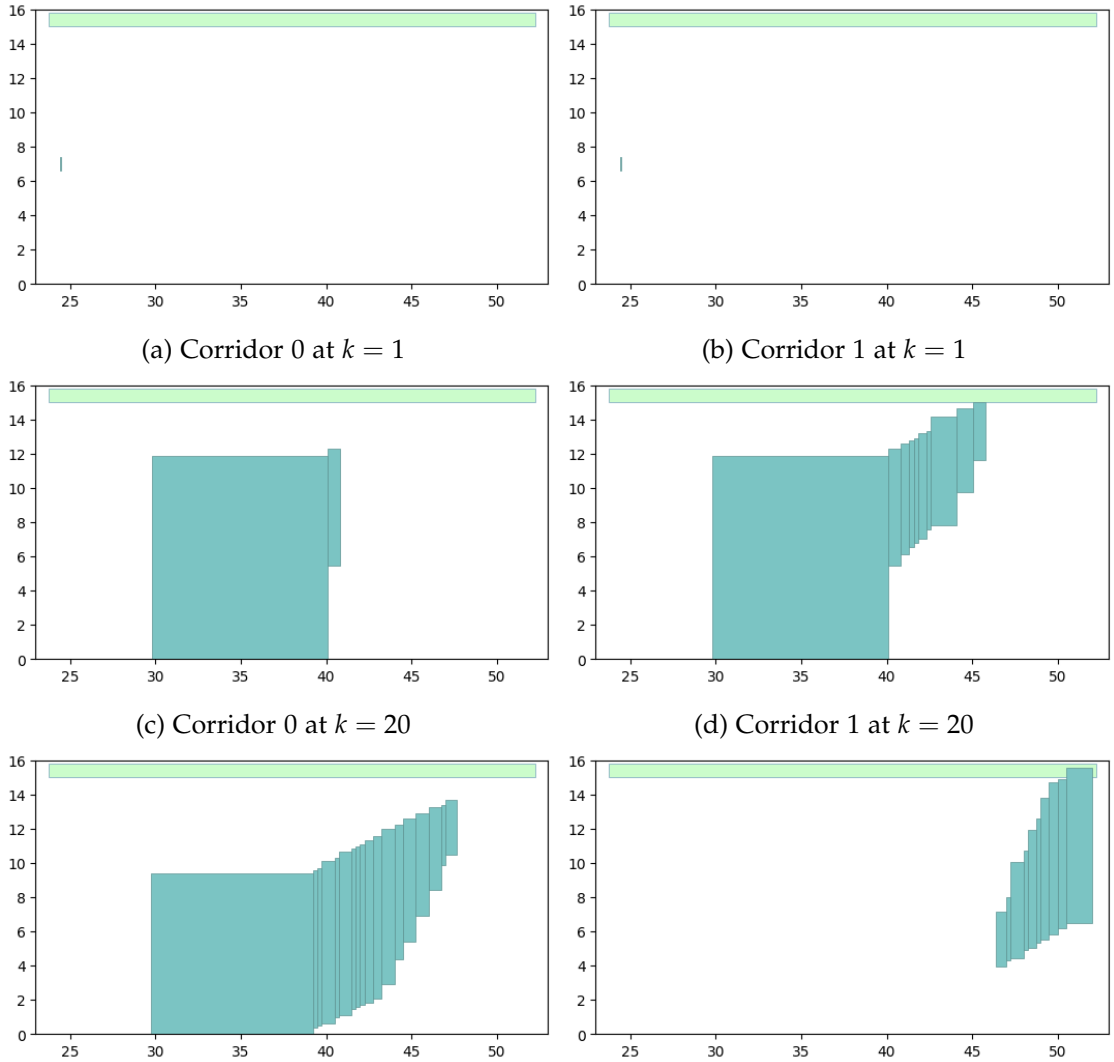


Figure 6.2.: Lateral and combined robustness shown among different measurements for specification ϕ = lat_position_between(-2,0).

This predicate will generate a specification polygon that ensures a lateral distance equal to the specified distance from the moving obstacle.

We consider scenario *DEU_Muc-2_2_T-1* to compute the aforementioned specification. A dynamic obstacle is randomly selected, identified by the number 203. The specification is invoked twice. In the first instance, we input a distance of 1 meter from the moving obstacle. Given this relatively short distance, we expect a significant portion of the driving corridor to align with the specification region. In the second case, we input a distance of 3 meters. With this increase in distance, we anticipate a smaller proportion of the driving corridor to meet the specification region, leading to a lower robustness score.

Figure 6.5 illustrates the application of the specification visually. The selected moving obstacle is highlighted in green, and the corresponding area where the specification is deemed true is also depicted in the same color. For each case, we capture the robustness values computed over a time horizon of 31 steps (Fig. 6.6). The selected corridor is also illustrated in Fig. 6.7a. All three measurements provide higher robustness scores for the lateral deviance of 1 meter compared to the 3 meters. In the last steps, the *Displacement* measurement still records a higher robustness score than the other measurements. That is because it simply regards the closest points to the specification polygon, whereas the other measurements also capture the nature of the reachable sets. In other words, while the *Displacement* measurement captures the shortest distance from the closest points, it fails to capture the increase in the distance from the rest of the reachable states, which is, in contrast, reflected by the two other robustness measurements (*Normalized W-Perimeter*



(a) Corridor 0 at $k = 1$

(b) Corridor 1 at $k = 1$

(c) Corridor 0 at $k = 20$

(d) Corridor 1 at $k = 20$

(e) Corridor 0 at $k = 28$

(f) Corridor 1 at $k = 28$

Figure 6.3.: Two driving corridors from scenario *ZAM_Intersection-1_1_T-1* shown at time steps 1, 20 and 28. The horizontal and vertical axes present the longitudinal position and velocity respectively.

and *Normalized S-W-Perimeter*).

6.2. Evaluation of temporal specifications

6.2.1. Slow velocity within an obstacle range

This specification requires the maintenance of a slow velocity inside the sensing range of an obstacle within a designated time frame. Based on this principle, we con-

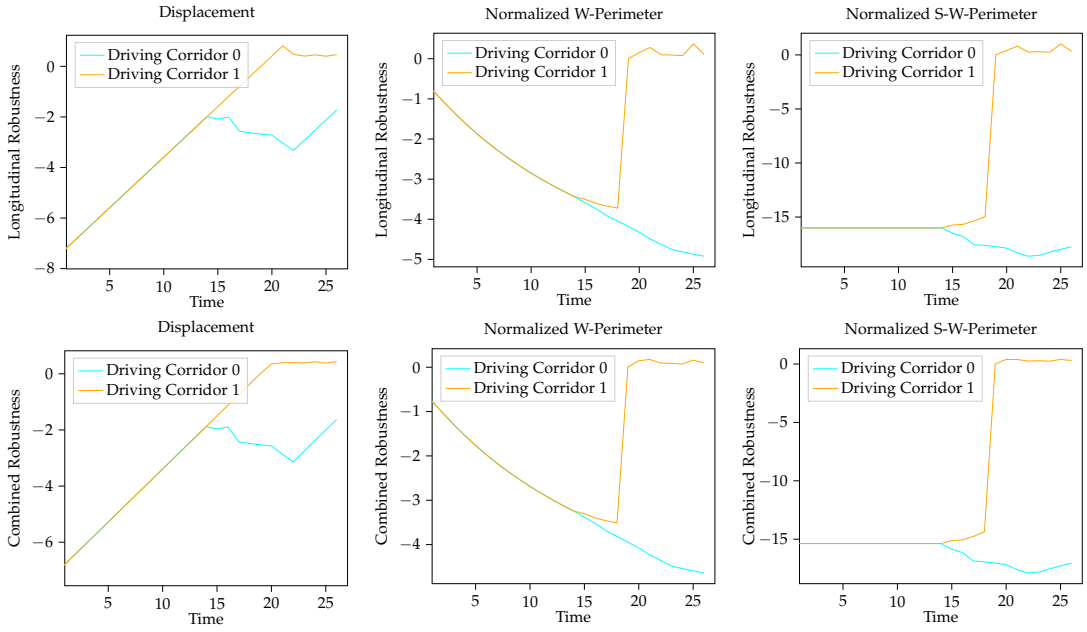
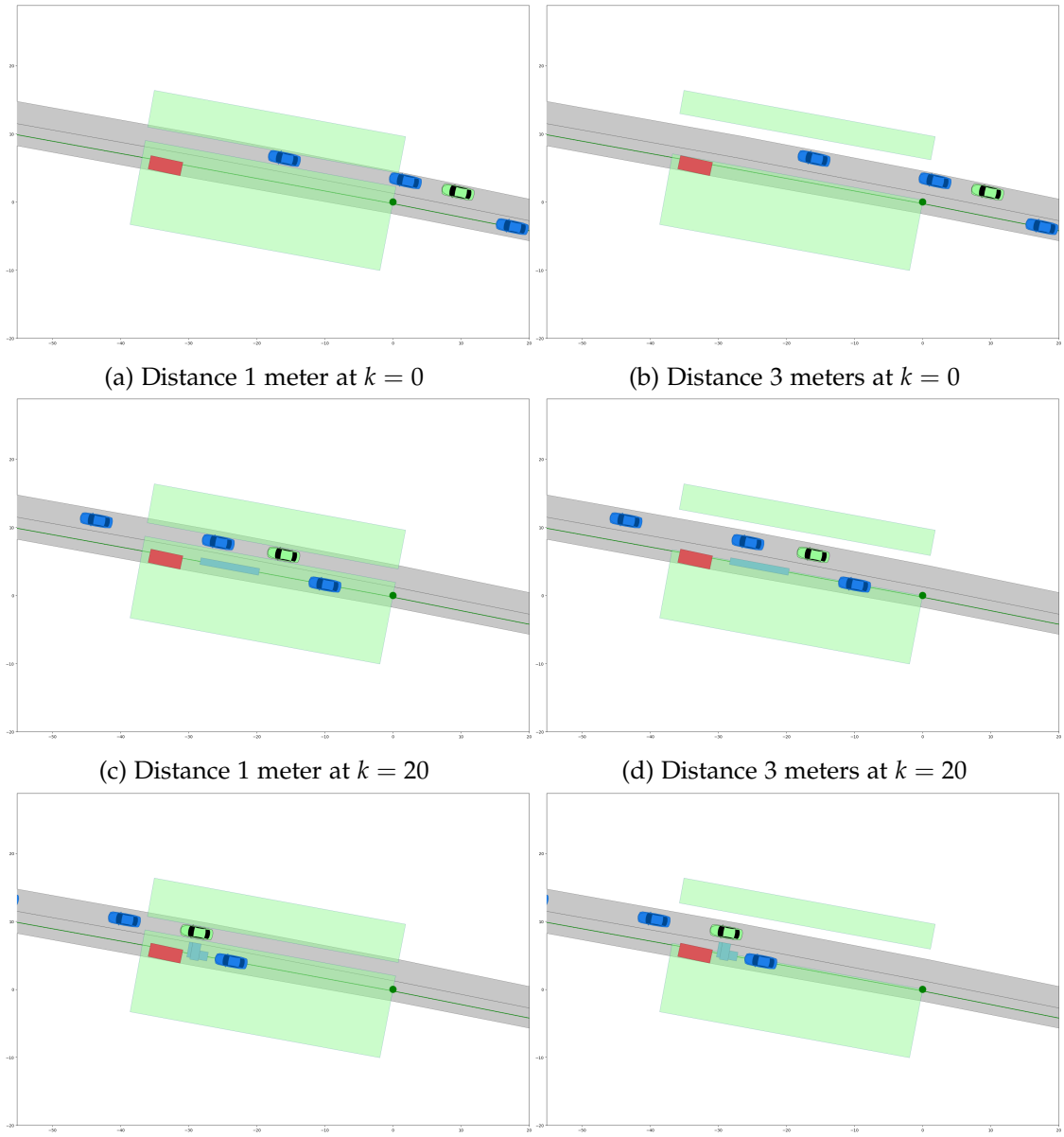


Figure 6.4.: Longitudinal and combined robustness shown among different measurements for specification $\phi = \text{lon\_velocity\_more\_than}(15)$.

struct the following temporal specification $\phi = \textit{within\_obstacle\_range}(\textit{obstacle}, 10)\ U_{[0,3]}\ \textit{lon\_velocity\_less\_than}(5)$. According to this specification a longitudinal velocity less than 5 m/s should be achieved within 3 time steps after being in the sensing range (of 10 meters) of the given *obstacle*.

Figure 6.7 showcases four driving corridors from the scenario *DEU_Muc-2_2_T-1*, where the given *obstacle* is illustrated in red. For the last time steps that fail to maintain a lower longitudinal velocity than 5 m/s, we have highlighted the corresponding reachable sets. Driving corridor one fails in the last 5 time steps, followed by corridor three with 3 time steps and slightly by corridor two with 2 time steps. On the other hand, corridor zero maintains a very low velocity due to the braking maneuver it represents. Consequently, the two last robustness measurements (*Normalized W-Perimeter* and *Normalized S-W-Perimeter*) favor driving corridors zero (highest) and two with higher robustness scores (along both longitudinal and combined scores), compared to driving corridors three and one (lowest) with the lowest robustness scores. On the other hand, the first robustness measurement (*Displacement*) fails to capture the correct ordering of the given driving corridors.

Another aspect to be noted is the robustness recorded during time steps 16 to 22 between driving corridors one and three. The average longitudinal velocities (see Fig. A.1) are considerably higher for the first driving corridor, thus making it harder to maintain a low-velocity limit. As a consequence, the two last robustness measurements (*Normalized W-Perimeter* and *Normalized S-W-Perimeter*) are able to capture this aspect and reflect a lower robustness score for driving corridor one.



(a) Distance 1 meter at $k = 0$

(b) Distance 3 meters at $k = 0$

(c) Distance 1 meter at $k = 20$

(d) Distance 3 meters at $k = 20$

(e) Distance 1 meter at $k = 30$

(f) Distance 3 meters at $k = 30$

Figure 6.5.: Two instances of the safe_lat_distance predicate with lateral distances of 1 meter and 3 meters, shown at time steps 0, 20 and 30.

6.2.2. Lane change with higher velocity

In this section, we consider the change to a new lanelet with higher velocity. For this reason, we construct the specification $\phi = (\mathit{In\_Lanelet}(52816) \vee \mathit{In\_Lanelet}(52756)) \wedge \mathit{Lon\_Velocity\_Less\_Than}(15)\ \mathcal{U}_{[0,2]}\ \mathit{Lon\_Velocity\_More\_Than}(15)$. We further test this specification in the *USA_Peach-1_1_T-1* scenario (see Fig 6.9).

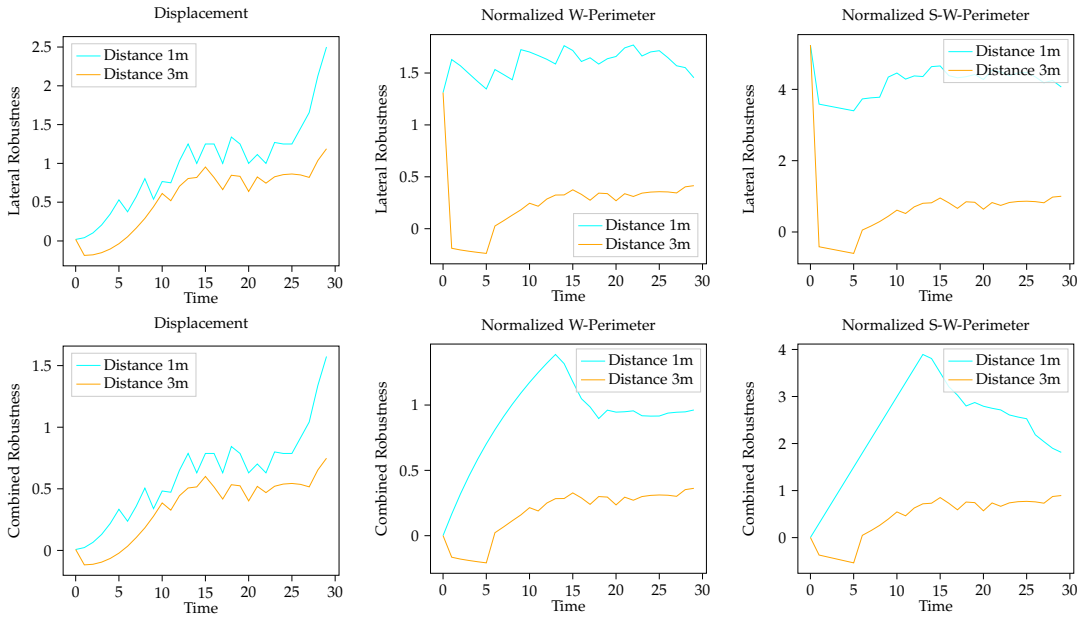


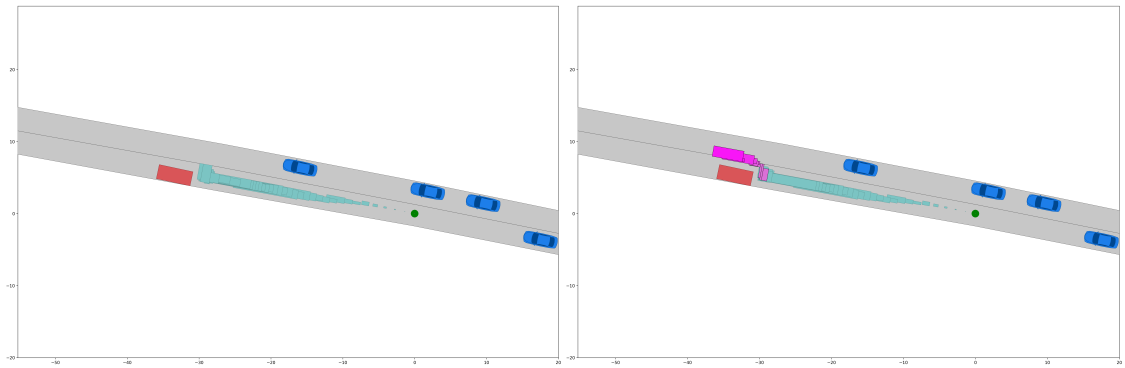
Figure 6.6.: Lateral and combined robustness shown among different measurements for the lateral deviation of 1 meter (blue) and 3 meters (orange).

The robustness scores are showcased in Fig 6.10. All three robustness measurements reflect increased robustness within the designated time frame. This is because we expect a higher velocity from the change of lanelets. From step 13 to around 17, the maximal longitudinal velocity plateaus, which is reflected in all three measurements. The reachable sets continue to expand until time $k = 26$, where suddenly the driving corridor brakes at time $k = 27$. The last two robustness measurements capture this increase and spontaneous decrease in velocity, as reflected in steps $17 - 22$ and $23 - 27$.

Another difference can be noted in the lateral robustness across the different measurements. The displacement-based measurement presents increasing robustness in the lateral dimension. In fact, the driving corridor expands considerably away from the two lanelets, which should, in turn, decrease the robustness, as reflected by the two other measurements. The reason for the increase in robustness by the first measurement is that it only considers the increase in the coverage of the current lanelets, which is significantly smaller than the coverage of other lanelets.

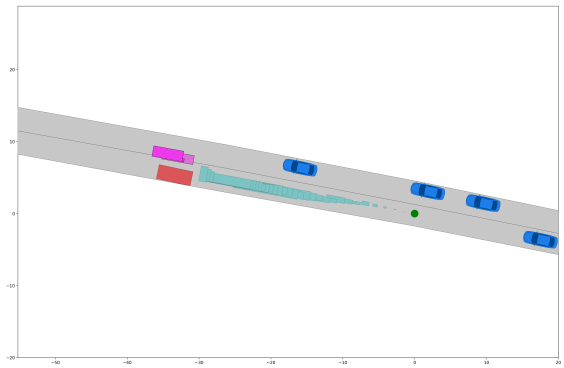
6.2.3. Lane change with minimal velocity

This specification requires the maintenance of a minimal velocity during a lane change within the designated time frame. Based on this principle, we construct the following expression $\phi = \neg \mathit{In\_Lanelet}(1552)\ U_{[0,2]}\ \mathit{Lon\_Velocity\_More\_Than}(10)$. According to this specification a longitudinal velocity of at least 10 m/s should be achieved within 3 seconds, from the moment when the lanelet is changed. The respective lanelet with the

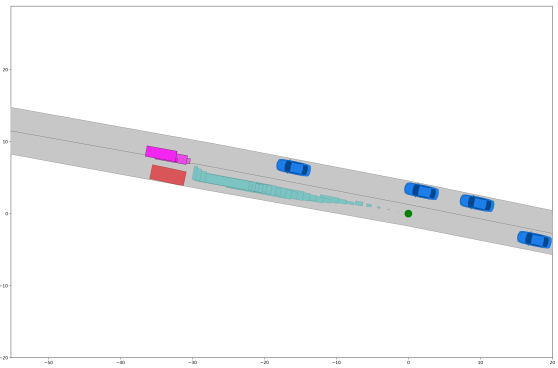


(a) Corridor 0

(b) Corridor 1



(c) Corridor 2



(d) Corridor 3

Figure 6.7.: Four driving corridors from scenario *DEU_Muc-2_2_T-1*. The violating reachable sets are highlighted with respective colors.

identification number of 1552 is highlighted in green and shown in Fig. 6.11a.

Figure 6.11 showcases two driving corridors (Fig. 6.11b and Fig. 6.11c) generated from the scenario *ITA_Empoli-2_5_T-1*. The reachable sets are illustrated with increasing brightness to reflect the change of the driving corridor over time. We are especially interested in the second half of the considered time horizon (presented in brighter colors), which corresponds to the divergence of the two driving corridors.

The computation of the robustness for this specification is presented in Fig. 6.12. As it can be observed from the scenario, driving corridor zero progresses with a straightforward path, thus indicating the possibility of achieving higher velocities. In contrast, the other driving corridor realizes a decelerating maneuver and brakes at the very end. Eventually, the attainable velocities for this corridor are lower than the first one, mirrored by a decrease in the robustness in all three measurements. On the other hand, driving corridor zero shows a positively increasing tendency, especially in the second half of the designated time horizon.

Considering the lateral robustness, the *Displacement* measurement fails to capture the lateral positioning of the two driving corridors. Since we are required not to be within

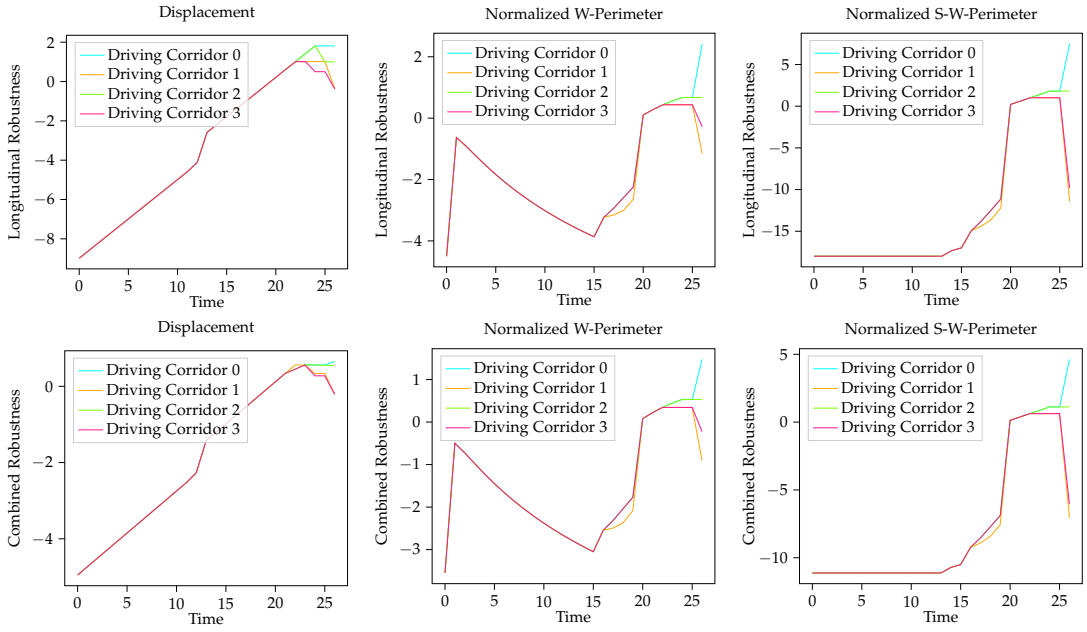
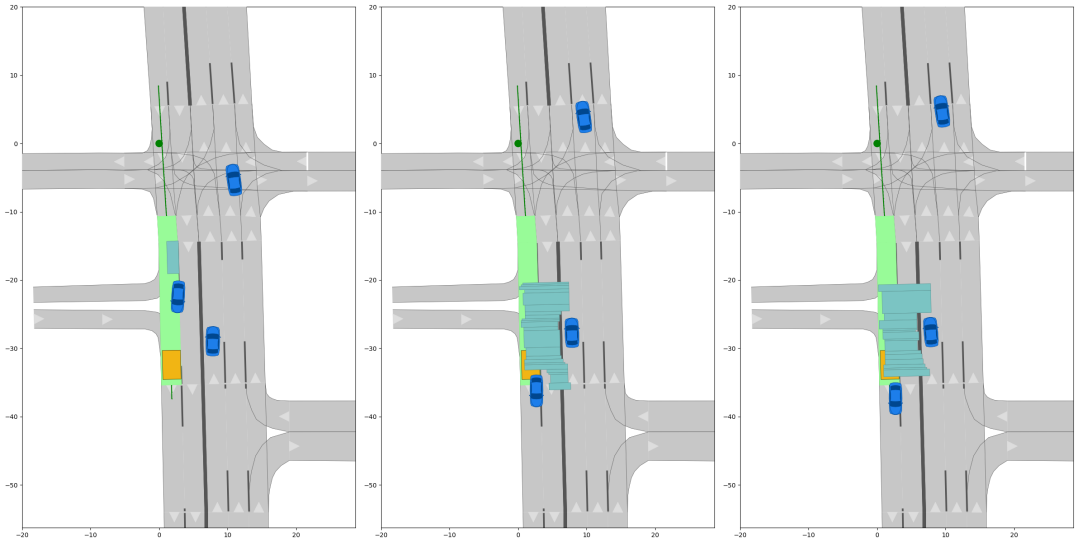


Figure 6.8.: Longitudinal and combined robustness shown among different driving corridors for specification $\phi = \text{within\_obstacle\_range}(\textit{obstacle}, 10)\ U_{0,3}$ lon_velocity_less_than(5).



(a) Corridor 0 at $k = 14$ (b) Corridor 0 at $k = 26$ (c) Corridor 0 at $k = 27$

Figure 6.9.: Scenario *USA_Peach-1_1_T-1* presented at time steps $k = 14$, $k = 26$, and $k = 27$, together with the highlighted lanelets.

the lanelet *1552*, driving corridor zero violates this specification more due to its minimal lateral deviance from the lanelet. In contrast, the other driving corridor is totally outside

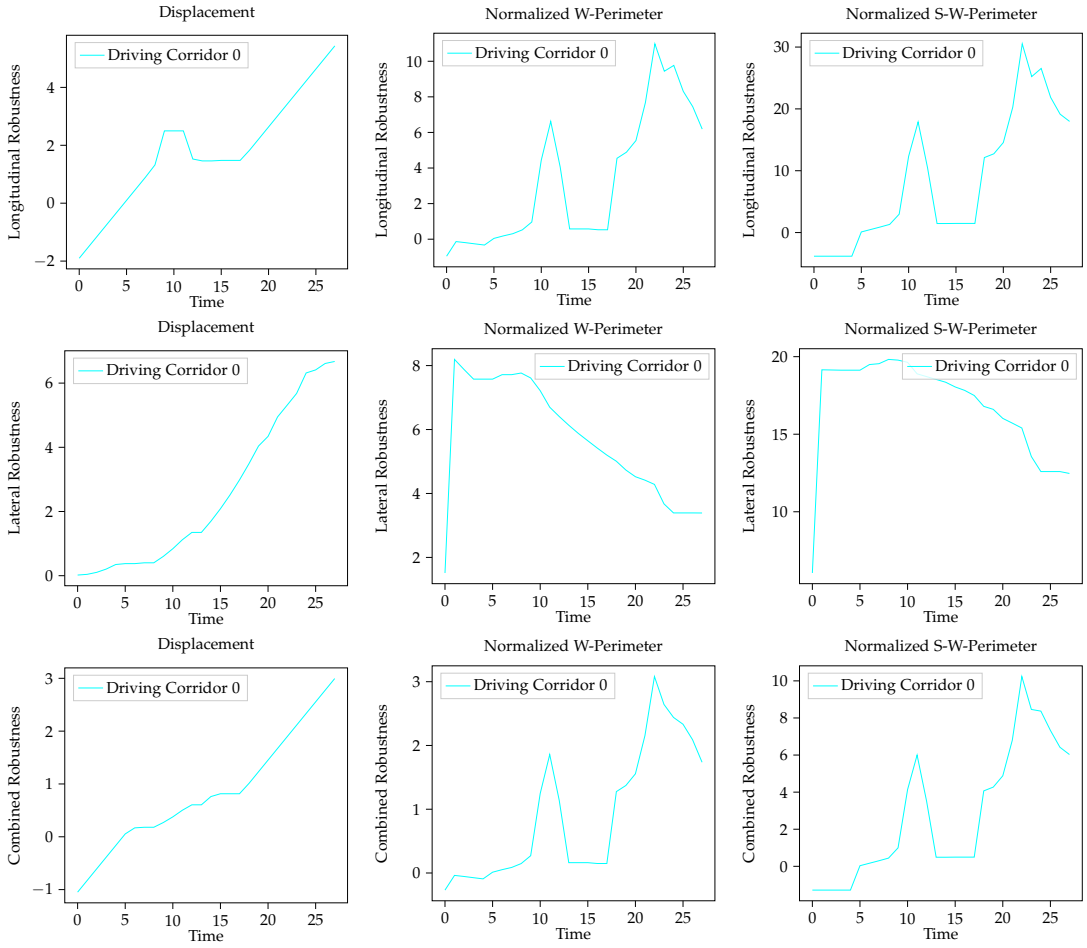
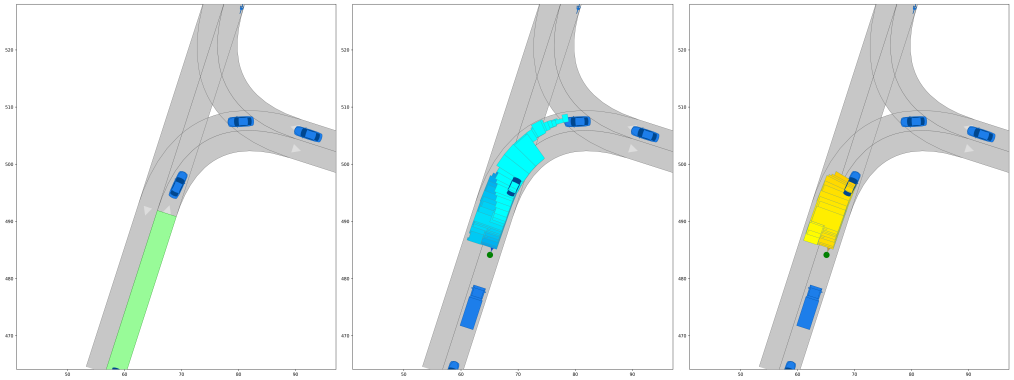


Figure 6.10.: Longitudinal, lateral and combined robustness for specification $\phi = (\text{In\_Lanelet}(52816) \vee \text{In\_Lanelet}(52756)) \wedge \text{Lon\_Velocity\_Less\_Than}(15)\ U_{[0,2]}\ \text{Lon\_Velocity\_More\_Than}(15)$.

the lateral scope of the given lanelet, thus resulting in higher robustness.

The combined robustness clearly captures the general tendency of the two driving corridors. As expected, driving corridor zero generally performs better than the other driving corridor due to the achievement of higher velocities. Because of the lower lateral robustness scores, the longitudinal robustness dominates and is reflected in all three combined measurements. Furthermore, the combined *Normalized S-W-Perimeter* measurement preserves the monotonicity to a greater extent than the combined *Normalized W-Perimeter* measurement, particularly evident in the time steps leading up to 15.



(a) Highlighted lanelet (b) Corridor 0 (c) Corridor 1

Figure 6.11.: Driving corridors and the highlighted lanelet *1552* from scenario *ITA_Empoli-2_5_T-1*.

6.3. Computational efficiency

In this section, we study the computational efficiency of our approach. The computational running times recorded across different time horizons and formula lengths are summarized in Fig. 6.13. The plot on the top-left shows the tendency of the running time across different robustness measurements. Three experiments have been conducted with changing time horizons ranging from 10, 20, and 30. The computational running times are averaged among the specifications regarded earlier in this chapter. The average computation times increase slightly for all three measurements as the time horizon expands. Additionally, the change in running times is also relatively small between the robustness measurements, especially in the last two.

The rest of the plots in Fig. 6.13 present the study of the individual computational running times for each robustness measurement as the size of the considered expression increases. We can observe very low running times and slight increments for the displacement-based measurement as the formula size varies. On the other hand, the two last robustness measurements show that the running times are both similar but relatively higher than the first measurement. This is because the approach used by these measurements relies heavily on the calculation and normalization of the weighted perimeters. Additionally, the change to the time horizon 30 exhibits longer computation times than the first measurement. However, even with the time frame's expansion, both measurements produce robustness scores under 1.2 seconds, which is a portion of the considered time horizon.

Another important aspect to consider is the scaling of the number of sequences as we vary the interval size of the temporal operators. To stress-test our approach, we consider the formula $\phi = (\phi_\alpha U_{[0,2]} \phi_\beta)\ U_{[0,n]}\ (\phi_\gamma U_{[0,2]} \phi_\delta)$, where n is the considered time horizon and the formulas ϕ_α, ϕ_β, ϕ_γ and ϕ_δ are atomic propositions. Figure 6.14 shows the number of sequences as we vary the time horizon n. In blue, we present the possible

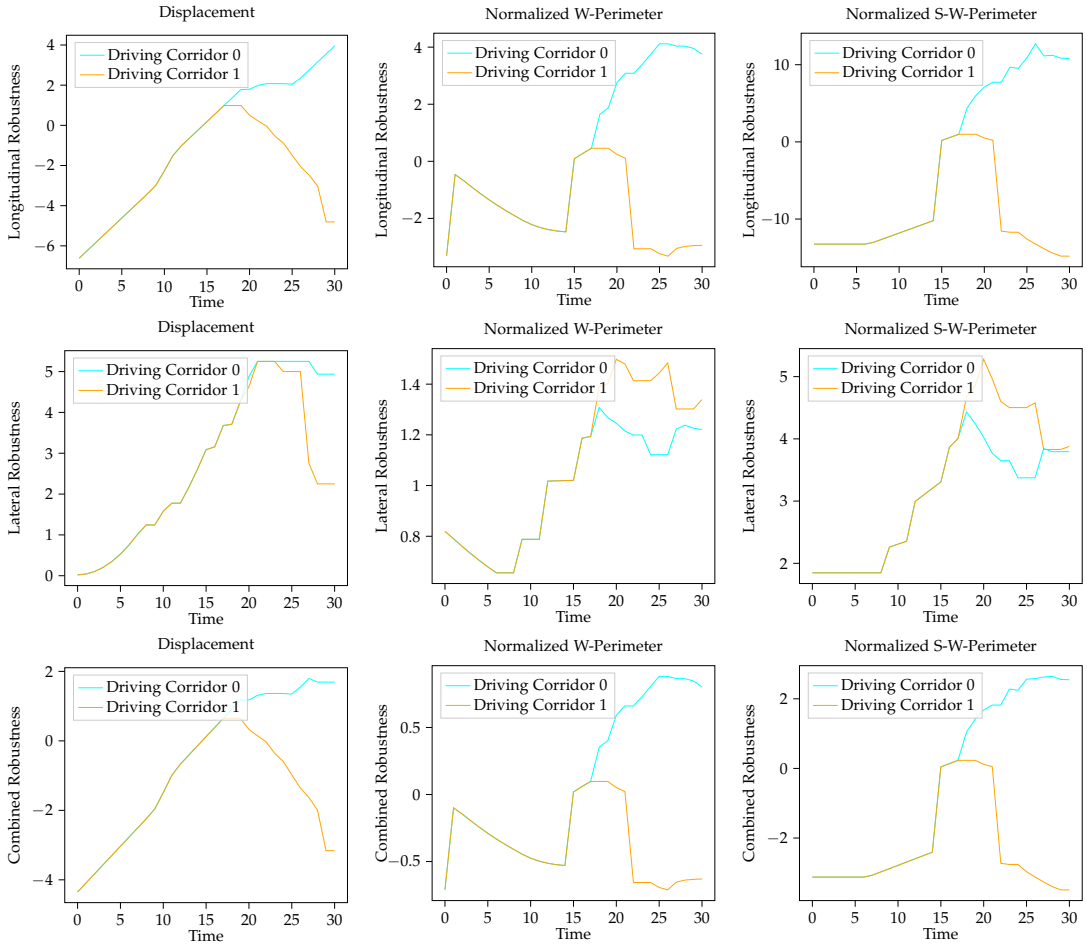


Figure 6.12.: Longitudinal, lateral and combined robustness for specification $\phi = \neg\text{In\_Lanelet}(1552)\ U_{[0,2]}\ \text{Lon\_Velocity\_More\_Than}(10)$.

solution space size, and in green, the optimized solution space resulting from the check of existing sequences as discussed in Chapter 3. It can be observed that the importance of the solution space's reduction becomes essential as the time horizon increases, thus necessitating the preservation of a minimal set of sequences.

It should be noted that the libraries chosen to establish the computation of the robustness measurements rely heavily on Python. This choice was influenced by the rapid nature of prototyping and ease of use. An implementation in a faster language, e.g., C++ could further improve the running time of our approach.

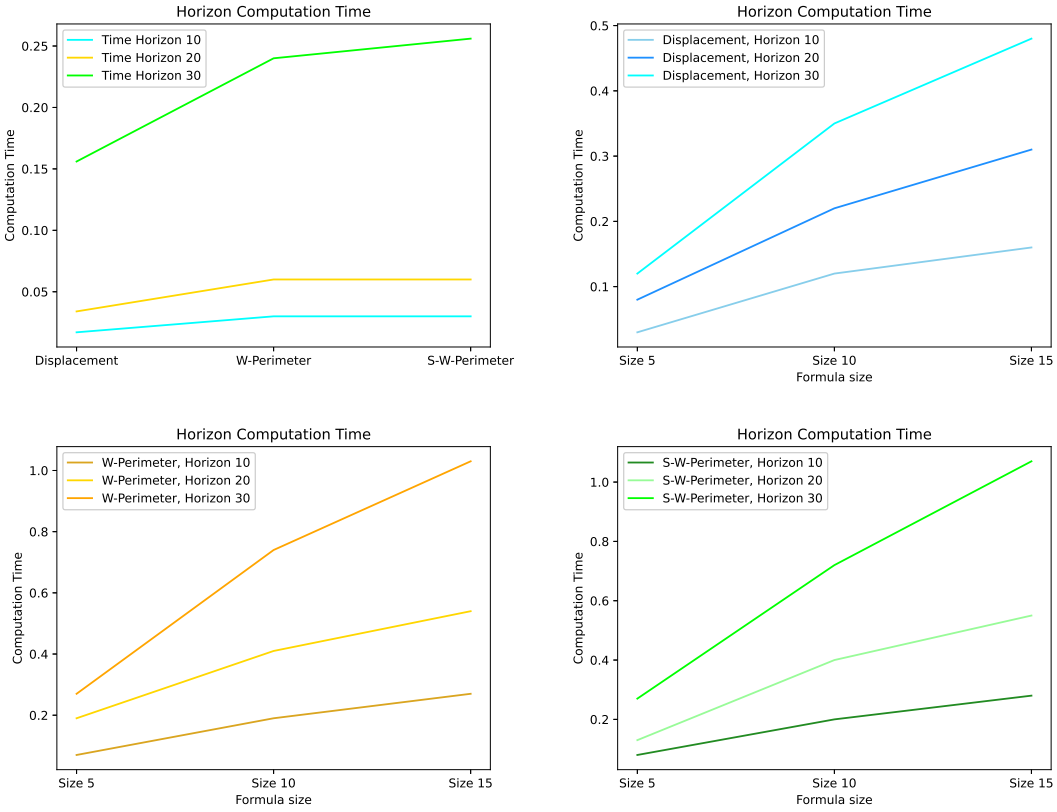


Figure 6.13.: Computational running times recorded across different time horizons and formula lengths.

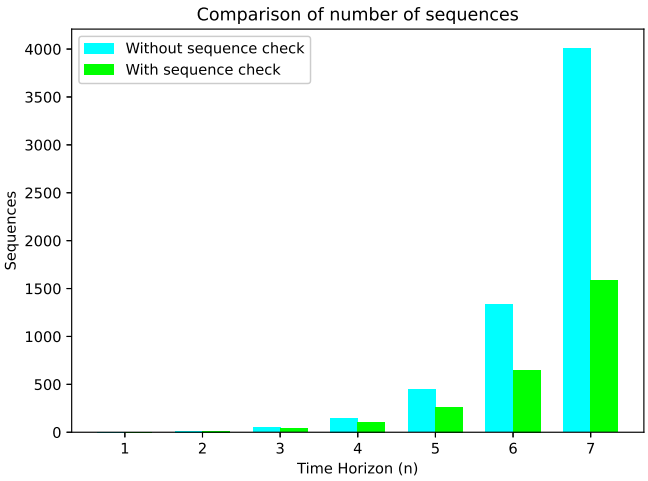


Figure 6.14.: Tendency of the number of sequences while increasing the interval length of temporal operators.

7. Conclusions

This chapter concludes the thesis. First, we describe the main conclusions in Section 7.1. Finally, Section 7.2 presents directions for future research.

7.1. Conclusions

In this thesis, we have tackled the problem of assessing driving corridors' robustness according to temporal logic's specifications. To this end, we have formulated an algorithmic framework for representing the STL specifications within the domains of reachable sets and developed novel robustness measurements. In our experimental settings, we have tested the adherence of driving corridors to non-temporal and temporal specifications, where we concluded the preservation of soundness and monotonicity. The experimental results, especially the measurement based on a normalized sides weighted perimeter approach, aligned with the initial predictions of adherence profiles and provided a fair ranking between driving corridors.

We enabled the generation of robustness scores according to the longitudinal and lateral domains. Since many curvilinear-based motion planners can plan independently on these domains, the robustness scores favor independent longitudinal or lateral planning strategies. Moreover, the proposed joint optimization problem allowed for combining robustness scores along the aforementioned dimensions across all driving corridors. Consequently, this enabled a fair comparison between driving corridors and the compilation of a comprehensive adherence profile to the given formalisms.

Due to the construction of our algorithm, we maintain a minimal solution space concerning the satisfaction possibilities of a specification. Our experimental results showed that the proposed formalisms could be solved in a reasonable time. Moreover, the number of sequences in these specifications remained low while we varied the interval lengths of the temporal operators. It should be noted that the efficiency of our approach can be further improved by switching to a faster language, e.g., C or C++.

7.2. Future Work

A direct use of our proposed approach can be the evaluation of the different maneuvers that could be performed by an ego vehicle. Since every driving corridor exhibits a specific maneuver, we can efficiently perform a ranking concerning a given specification. Moreover, our robustness measurement can be utilized to weight the reachable sets within the reachability graph. Consequently, this can lead to the creation of a weighted

reachability graph, which, in turn, can allow for the extraction of driving corridors according to their robustness scores.

Another research direction can be the maximal satisfaction of multi-agent systems with temporal logic specifications. Using our algorithmic framework, the solution spaces to the given specifications can be computed for the given agents, and the sequences with the highest robustness can be selected. As a result, we can derive the position intervals that maximally satisfy the given formalisms while avoiding collision with each other. To this end, extra caution should be paid, especially when assigning positions in a non-conflicting way.

A. Parameters and Velocity Tendency

The following tables present the values of parameters among the driving corridors for each presented scenario.

p^{lon}_{min}	p^{lon}_{max}	p^{lat}_{min}	p^{lat}_{max}
23.74	52.2325	-5.4953	7.5424
v^{lon}_{min}	v^{lon}_{max}	v^{lat}_{min}	v^{lat}_{max}
0.0	15.81	-4.0	4.0

p^{lon}_{min}	p^{lon}_{max}	p^{lat}_{min}	p^{lat}_{max}
128.443	165.8527	-10.0	10.0
v^{lon}_{min}	v^{lon}_{max}	v^{lat}_{min}	v^{lat}_{max}
3.18706	20.40706	-3.60245	2.33505

Table A.1.: Parameters' values for scenarios *ZAM_Intersection-1_1_T-1* (left) and *DEU_Muc-2_2_T-1* (right).

p^{lon}_{min}	p^{lon}_{max}	p^{lat}_{min}	p^{lat}_{max}
8.4143	54.04	-10.0	10.0
v^{lon}_{min}	v^{lon}_{max}	v^{lat}_{min}	v^{lat}_{max}
1.07795	20.8335	-3.0887	4.0

p^{lon}_{min}	p^{lon}_{max}	p^{lat}_{min}	p^{lat}_{max}
25.2439	51.1425	-0.92	5.275
v^{lon}_{min}	v^{lon}_{max}	v^{lat}_{min}	v^{lat}_{max}
0	14.3564	-3.112	4.0

Table A.2.: Parameters' values for scenarios *USA_Peach-1_1_T-1*(left) and *ITA_Empoli-2_5_T-1* (right).

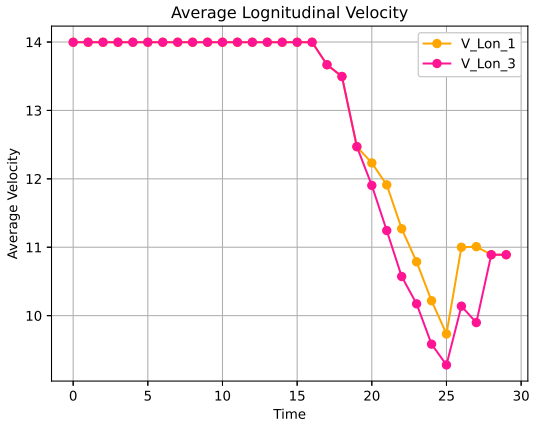


Figure A.1.: Average longitudinal velocity tendency for the first and third driving corridor from the scenario *DEU_Muc-2_2_T-1*.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] S. Söntges and M. Althoff. "Computing the Drivable Area of Autonomous Road Vehicles in Dynamic Road Scenes." In: *IEEE Transactions on Intelligent Transportation Systems* 19.6 (2018), pp. 1855–1866.

[2] S. Manzinger, C. Pek, and M. Althoff. "Using Reachable Sets for Trajectory Planning of Automated Vehicles." In: *IEEE Transactions on Intelligent Vehicles* PP (Aug. 2020), pp. 1–1.

[3] G. Würsching and M. Althoff. "Sampling-Based Optimal Trajectory Generation for Autonomous Vehicles Using Reachable Sets." In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. 2021, pp. 828–835.

[4] E. I. Liu and M. Althoff. "Computing Specification-Compliant Reachable Sets for Motion Planning of Automated Vehicles." In: *2021 IEEE Intelligent Vehicles Symposium (IV)*. 2021, pp. 1037–1044.

[5] E. I. Liu, G. Würsching, M. Klischat, and M. Althoff. "CommonRoad-Reach: A Toolbox for Reachability Analysis of Automated Vehicles." In: *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. 2022, pp. 2313–2320.

[6] J. Tumova, G. Hall, S. Karaman, E. Frazzoli, and D. Rus. "Least-violating control strategy synthesis with safety rules." In: Apr. 2013, pp. 1–10.

[7] M. Lahijanian, S. Almagor, D. Fried, L. Kavraki, and M. Vardi. "This Time the Robot Settles for a Cost: A Quantitative Approach to Temporal Logic Planning with Partial Satisfaction." In: Jan. 2015.

[8] C.-I. Vasile, J. Tumova, S. Karaman, C. Belta, and D. Rus. "Minimum-violation scLTL motion planning for mobility-on-demand." In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 1481–1488.

[9] J. Karlsson, J. Tumova, C.-I. Vasile, S. Karaman, and D. Rus. "Multi-Vehicle Motion Planning for Social Optimal Mobility-on-Demand." In: (May 2018).

[10] J. Karlsson and J. Tumova. "Intention-aware motion planning with road rules." In: *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*. 2020, pp. 526–532.

[11] M. Hekmatnejad, S. Yaghoubi, A. Dokhanchi, H. Amor, A. Shrivastava, L. Karam, and G. Fainekos. "Encoding and monitoring responsibility sensitive safety rules for automated vehicles in signal temporal logic." English (US). In: *MEMOCODE 2019 - 17th ACM-IEEE International Conference on Formal Methods and Models for System Design*. MEMOCODE 2019 - 17th ACM-IEEE International Conference on Formal Methods and Models for System Design. Association for Computing Machinery, Inc, Oct. 2019.

[12] A. Rizaldi, F. Immler, and M. Althoff. "A Formally Verified Checker of the Safe Distance Traffic Rules for Autonomous Vehicles." In: June 2016, pp. 175–190. ISBN: 978-3-319-40647-3.

[13] K. Esterle, V. Aravantinos, and A. Knoll. "From Specifications to Behavior: Maneuver Verification in a Semantic State Space." In: May 2019.

[14] R. Kohlhaas, T. Bittner, T. Schamm, and J. Zöllner. "Semantic state space for high-level maneuver planning in structured traffic scenes." In: Oct. 2014.

[15] A. Best, S. Narang, D. Barber, and D. Manocha. "AutonoVi: Autonomous vehicle planning with dynamic maneuvers and traffic constraints." In: Sept. 2017, pp. 2629–2636.

[16] S. Sontges and M. Althoff. "Computing possible driving corridors for automated vehicles." In: June 2017, pp. 160–166.

[17] K. Esterle, L. Gressenbuch, and A. Knoll. "Formalizing Traffic Rules for Machine Interpretability." In: *2020 IEEE 3rd Connected and Automated Vehicles Symposium (CAVS)*. 2020, pp. 1–7.

[18] S. Maierhofer, A.-K. Rettinger, E. C. Mayer, and M. Althoff. "Formalization of Interstate Traffic Rules in Temporal Logic." In: *2020 IEEE Intelligent Vehicles Symposium (IV)*. 2020, pp. 752–759.

[19] E. Irani Liu and M. Althoff. "Specification-Compliant Driving Corridors for Motion Planning of Automated Vehicles." In: *IEEE Transactions on Intelligent Vehicles* (2023), pp. 1–17.

[20] H. Schlüter, P. Schillinger, and M. Bürger. "On the Design of Penalty Structures for Minimum-Violation LTL Motion Planning." In: Dec. 2018, pp. 4153–4158.

[21] J. Karlsson, F. S. Barbosa, and J. Tumova. "Sampling-based Motion Planning with Temporal Logic Missions and Spatial Preferences**This work was supported by the EU H2020 Research and Innovation Programme under GA No. 731869 (Co4Robots), and by the Swedish Research Council (VR)." In: *IFAC-PapersOnLine* 53.2 (2020). 21st IFAC World Congress, pp. 15537–15543. ISSN: 2405-8963.

[22] J. Rong and N. Luan. "Safe Reinforcement Learning with Policy-Guided Planning for Autonomous Driving." In: *2020 IEEE International Conference on Mechatronics and Automation (ICMA)*. 2020, pp. 320–326.

[23] T. Wongpiromsarn, K. Slutsky, E. Frazzoli, and U. Topcu. "Minimum-Violation Planning for Autonomous Systems: Theoretical and Practical Considerations." In: *2021 American Control Conference (ACC)*. 2021, pp. 4866–4872.

[24] J. Karlsson, S. van Waveren, C. Pek, I. Torre, I. Leite, and J. Tumova. "Encoding Human Driving Styles in Motion Planning for Autonomous Vehicles." In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 1050–1056.

[25] J. Tumova, S. Karaman, C. Belta, and D. Rus. "Least-Violating Planning in Road Networks from Temporal Logic Specifications." In: Apr. 2016, pp. 1–9.

[26] L. Janiuk and J. Sjölén. *Probabilistic Least-violating Control Strategy Synthesis with Safety Rules*. 2018.

[27] L. Niu, J. Fu, and A. Clark. "Minimum Violation Control Synthesis on Cyber-Physical Systems under Attacks." In: *CoRR* abs/1809.00975 (2018). arXiv: 1809.00975.

[28] S. Andersson and D. V. Dimarogonas. "Human in the Loop Least Violating Robot Control Synthesis under Metric Interval Temporal Logic Specifications." In: *2018 European Control Conference (ECC)*. 2018, pp. 453–458.

[29] L. I. R. Castro, P. Chaudhari, J. Tumova, S. Karaman, E. Frazzoli, and D. Rus. "Incremental Sampling-based Algorithm for Minimum-violation Motion Planning." In: *CoRR* abs/1305.1102 (2013). arXiv: 1305.1102.

[30] S. Bharadwaj, T. Wongpiromsarn, N. Neogi, J. Muffoletto, and U. Topcu. "Minimum-Violation Traffic Management for Urban Air Mobility." In: May 2021, pp. 37–52. ISBN: 978-3-030-76383-1.

[31] W. Xiao, N. Mehdipour, A. Collin, A. Bin-Nun, E. Frazzoli, R. J. D. Tebbens, and C. Belta. "Rule-based Optimal Control for Autonomous Driving." In: *CoRR* abs/2101.05709 (2021). arXiv: 2101.05709.

[32] W. Xiao, N. Mehdipour, A. Collin, A. Y. Bin-Nun, E. Frazzoli, R. J. D. Tebbens, and C. Belta. "Rule-based Evaluation and Optimal Control for Autonomous Driving." In: *CoRR* abs/2107.07460 (2021). arXiv: 2107.07460.

[33] M. Srinivasan, S. Coogan, and M. Egerstedt. "Control of Multi-Agent Systems with Finite Time Control Barrier Certificates and Temporal Logic." In: *2018 IEEE Conference on Decision and Control (CDC)*. 2018, pp. 1991–1996.

[34] R. Cheng, M. J. Khojasteh, A. D. Ames, and J. W. Burdick. "Safe Multi-Agent Interaction through Robust Control Barrier Functions with Learned Uncertainties." In: *2020 59th IEEE Conference on Decision and Control (CDC)*. 2020, pp. 777–783.

[35] H.-D. Bui, Y. Lu, and E. Plaku. "Improving the Efficiency of Sampling-based Motion Planners via Runtime Predictions for Motion-Planning Problems with Dynamics." In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 4486–4491.

[36] A. Censi, K. Slutsky, T. Wongpiromsarn, D. S. Yershov, S. Pendleton, J. G. M. Fu, and E. Frazzoli. "Liability, Ethics, and Culture-Aware Behavior Specification using Rulebooks." In: *CoRR* abs/1902.09355 (2019). arXiv: 1902.09355.

[37] P. Chaudhari, T. Wongpiromsarny, and E. Frazzoli. "Incremental minimum-violation control synthesis for robots interacting with external agents." In: *2014 American Control Conference*. 2014, pp. 1761–1768.

[38] A. Zanardi, E. Mion, M. Bruschetta, S. Bolognani, A. Censi, and E. Frazzoli. "Urban Driving Games With Lexicographic Preferences and Socially Efficient Nash Equilibria." In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 4978–4985.

[39] A. Zanardi, G. Zardini, S. Srinivasan, S. Bolognani, A. Censi, F. Dörfler, and E. Frazzoli. "Posetal Games: Efficiency, Existence, and Refinement of Equilibria in Games With Prioritized Metrics." In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 1292–1299.

[40] T. Akazaki and I. Hasuo. "Time Robustness in MTL and Expressivity in Hybrid System Falsification (Extended Version)." In: vol. 9207. May 2015. ISBN: 978-3-319-21667-6.

[41] O. Maler and D. Nickovic. "Monitoring Temporal Properties of Continuous Signals." In: vol. 3253. Jan. 2004, pp. 152–166. ISBN: 978-3-540-23167-7.

[42] M. Ma, J. Gao, L. Feng, and J. Stankovic. "STLnet: Signal Temporal Logic Enforced Multivariate Recurrent Neural Networks." In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS'20. Vancouver, BC, Canada: Curran Associates Inc., 2020. ISBN: 9781713829546.

[43] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, and S. Sankaranarayanan. "Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications." In: *Lectures on Runtime Verification: Introductory and Advanced Topics*. Ed. by E. Bartocci and Y. Falcone. Cham: Springer International Publishing, 2018, pp. 135–175. ISBN: 978-3-319-75632-5.

[44] N. Mehdipour, C.-I. Vasile, and C. Belta. "Arithmetic-Geometric Mean Robustness for Control from Signal Temporal Logic Specifications." In: *2019 American Control Conference (ACC)*. 2019, pp. 1690–1695.

[45] P. Varnai and D. V. Dimarogonas. "On Robustness Metrics for Learning STL Tasks." In: *2020 American Control Conference (ACC)*. 2020, pp. 5394–5399.

[46] D. Zhao, Z. Zhou, Z. Cai, T. Long, S. Yangui, and X. Xue. "ASTL: Accumulative Signal Temporal Logic for IoT Service Monitoring." In: *2022 IEEE International Conference on Web Services (ICWS)*. 2022, pp. 256–265.

[47] N. Mehdipour, C.-I. Vasile, and C. Belta. "Specifying User Preferences Using Weighted Signal Temporal Logic." In: *IEEE Control Systems Letters* 5.6 (2021), pp. 2006–2011.

[48] N. Kochdumper and S. Bak. *Fully Automated Verification of Linear Time-Invariant Systems against Signal Temporal Logic Specifications via Reachability Analysis*. 2023. arXiv: 2306.04089 [cs.LO].

[49] M. Wetzlinger, N. Kochdumper, S. Bak, and M. Althoff. "Fully Automated Verification of Linear Systems Using Inner- and Outer-Approximations of Reachable Sets." In: *IEEE Transactions on Automatic Control* (2023), pp. 1–16.

[50] M. Althoff, M. Koschi, and S. Manzinger. "CommonRoad: Composable benchmarks for motion planning on roads." In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. 2017, pp. 719–726.

[51] P. Bender, J. Ziegler, and C. Stiller. "Lanelets: Efficient map representation for autonomous driving." In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*. 2014, pp. 420–425.

[52] E. Héry, S. Masi, P. Xu, and P. Bonnifait. "Map-based curvilinear coordinates for autonomous vehicles." In: Oct. 2017, pp. 1–7.

[53] S. Bhattacharya and R. Ghrist. "Path homotopy invariants and their application to optimal trajectory planning." In: *Annals of Mathematics and Artificial Intelligence* 84.3 (2018), pp. 139–160. ISSN: 1573-7470.

[54] G. Scher, S. Sadraddini, and H. Kress-Gazit. "Robustness-based Synthesis for Stochastic Systems under Signal Temporal Logic Tasks." In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 1269–1275.

[55] P. Thati and G. Roşu. "Monitoring Algorithms for Metric Temporal Logic Specifications." In: *Electronic Notes in Theoretical Computer Science* 113 (2005). Proceedings of the Fourth Workshop on Runtime Verification (RV 2004), pp. 145–162. ISSN: 1571-0661.

[56] A. Donzé and O. Maler. "Robust Satisfaction of Temporal Logic over Real-Valued Signals." In: *Formal Modeling and Analysis of Timed Systems*. Ed. by K. Chatterjee and T. A. Henzinger. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 92–106. ISBN: 978-3-642-15297-9.

[57] L. C. Paulson. *Logic and Proof*. Computer Science Tripos Part IB. Department of Computer Science and Technology, University of Cambridge, United Kingdom, 2021.

[58] L. Gressenbuch and M. Althoff. "Predictive Monitoring of Traffic Rules." In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2021.

[59] H. Roehm, J. Oehlerking, T. Heinz, and M. Althoff. "STL model checking of continuous and hybrid systems." In: *Automated Technology for Verification and Analysis: 14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings 14*. Springer. 2016, pp. 412–427.

[60] S. P. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. ISBN: 978-0-521-83378.

[61] X. Cabré, X. Ros-Oton, and J. Serra. "Sharp Isoperimetric Inequalities via the ABP Method." In: *Journal of the European Mathematical Society* 17.4 (2015), pp. 973–1000.

[62] E. Milman and J. Neeman. *The Gaussian Double-Bubble and Multi-Bubble Conjectures*. 2021. arXiv: `1805.10961 [math.DG]`.