2024

Inference and synthesis of temporal logic properties for autonomous systems

BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Dissertation

INFERENCE AND SYNTHESIS OF TEMPORAL LOGIC PROPERTIES FOR AUTONOMOUS SYSTEMS

by

ERFAN AASI

B.S., Sharif University of Technology, 2018
M.S., Boston University, 2022

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2024

Approved by

First Reader

Calin A. Belta, PhD
Professor of Mechanical Engineering
Professor of Electrical and Computer Engineering
Professor of Systems Engineering

Second Reader

Roberto Tron, PhD
Associate Professor of Mechanical Engineering
Associate Professor of Systems Engineering

Third Reader

Sean B. Andersson, PhD
Professor and Chair of Mechanical Engineering
Professor of Systems Engineering

Fourth Reader

Alyssa Pierson, PhD
Assistant Professor of Mechanical Engineering
Assistant Professor of Systems Engineering

Learn the alchemy true human beings know.
The moment you accept what troubles you've been given,
the door will open. Rumi

Acknowledgments

The completion of this dissertation owes its credit to the assistance and support provided by many people. Primarily, I want to express my sincerest gratitude to my PhD advisor, Prof. Calin Belta, for his support, motivation and guidance that helped me become a better researcher. His illuminating mentorship and encouragement throughout my PhD journey will be eternally cherished. I also want to thank my thesis committee members – Prof. Roberto Tron, Prof. Sean Andersson, and Prof. Alyssa Pierson – for dedicating their time to assess my thesis and providing insightful comments that contributed to enhancing this dissertation.

I wish to extend my deepest gratitude to Prof. Cristian Ioan Vasile, who served as a co-advisor during my PhD, consistently offering invaluable feedback and unwavering support. I also want to appreciate my collaborators at Boston University, Lehigh University, Georgia Institute of Technology, and Massachusetts Institute of Technology, for their constructive feedbacks and our fruitful discussions.

I want to thank all of the current and past members of the Robotics Lab, Mechanical Engineering Department and CISE community who made my experience at Boston University exceptionally wonderful. I would like to especially thank my roommate, Saeed Mohammadzadeh, for all of his help and support on various occasions. I am also thankful for all of my amazing friends in Boston and other places, who have truly enriched my PhD experience.

Finally, yet of utmost significance, I wish to express my heartfelt gratitude to my family. My incredible and caring mother Tahereh, my remarkable and affectionate father Aliakbar, and my cherished and supportive brother Aref, deserve special mention, for their unconditional love and support throughout the years. The sacrifices they've willingly undertaken have played a fundamental role in my attainment of this milestone, for which I will remain eternally grateful.

Erfan Aasi

Ph.D. Student

ME Department

INFERENCE AND SYNTHESIS OF TEMPORAL LOGIC PROPERTIES FOR AUTONOMOUS SYSTEMS

ERFAN AASI

Boston University, College of Engineering, 2024

Major Professor: Calin A. Belta, PhD
Professor of Mechanical Engineering
Professor of Electrical and Computer Engineering
Professor of Systems Engineering

ABSTRACT

Recently, formal methods have gained significant traction for describing, checking, and synthesizing the behaviors of cyber-physical systems. Among these methods, temporal logics stand out as they offer concise mathematical formulas to express desired system properties. In this thesis, our focus revolves around two primary applications of temporal logics in describing the behavior of autonomous system. The first involves integrating temporal logics with machine learning techniques to deduce a temporal logic specification based on the system's execution traces. The second application concerns using temporal logics to define traffic rules and develop a control scheme that guarantees compliance with these rules for autonomous vehicles. Ultimately, our objective is to combine these approaches, infer a specification that characterizes the desired behaviors of autonomous vehicles, and ensure that these behaviors are upheld during runtime.

In the first study of this thesis, our focus is on learning Signal Temporal Logic (STL) specifications from system execution traces. Our approach involves two main phases. Initially, we address an offline supervised learning problem, leveraging the

availability of system traces and their corresponding labels. Subsequently, we introduce a time-incremental learning framework. This framework is designed for a dataset containing labeled signal traces with a common time horizon. It provides a method to predict the label of a signal as it is received incrementally over time. To tackle both problems, we propose two decision tree-based approaches, with the aim of enhancing the interpretability and classification performance of existing methods. The simulation results demonstrate the efficiency of our proposed approaches.

In the next study, we address the challenge of guaranteeing compliance with traffic rules expressed as STL specifications within the domain of autonomous driving. Our focus is on developing control frameworks for a fully autonomous vehicle operating in a deterministic or stochastic environment. Our frameworks effectively translate the traffic rules into high-level decisions and accomplish low-level vehicle control with good real-time performance. Compared to existing literature, our approaches demonstrate significant enhancements in terms of runtime performance.

Contents

List of Tables

List of Figures

List of Abbreviations

BCDT	.....	Boosted Concise Decision Tree
CDT	.....	Concise Decision Tree
IMCR	.....	Incremental MisClassification Rate
LTL	.....	Linear Temporal Logic
MCR	.....	MisClassification Rate
MG	.....	Misclassification Gain
MILC	.....	Mixed-Integer Linear Constraint
MIQP	.....	Mixed-Integer Quadratic Programming
MPC	.....	Model Predictive Control
NMPC	.....	Nonlinear Model Predictive Control
PSO	.....	Particle Swarm Optimization
PSTL	.....	Parametric Signal Temporal Logic
STL	.....	Signal Temporal Logic
TMCR	.....	Timepoint MisClassification Rate
wSTL	.....	Weighted Signal Temporal Logic

Chapter 1

Introduction

1.1 Motivation

Dynamic systems are utilized extensively across diverse domains, including but not limited to robotics, medical sciences, transportation, and biology. Such systems are essential elements in addressing the grand challenges for engineering in the 21st century. Modeling and control of dynamic systems represent a crucial research field, which entails the development of novel tools and methodologies for comprehending, forecasting, and controlling autonomous systems, and it has numerous benefits and implications.

Formal methods is a powerful mathematical tool that enables us to specify and verify system behaviors (Baier and Katoen, 2008). In the last decades, there has been a great interest in integrating formal methods with control algorithms, to synthesize controllers that satisfy high-level objectives (Belta et al., 2017), and with machine learning approaches, to describe the system behaviors with interpretable specifications (Bombara et al., 2016). *Temporal logics* (Clarke et al., 1986), as a branch of formal methods, is a precise formal language to describe time-series properties by formulas that are easy to understand by humans. Motivated by the readability and interpretability of temporal logic formulae, they are widely used to describe time-series behaviors of dynamic systems.

Despite recent advances in employing temporal logics with control algorithms, existing approaches generally are computationally expensive and they do not satisfy

real-time performance requirements, specifically in control of autonomous vehicles. On the other side, most existing approaches that integrate temporal logics with machine learning algorithms either do not achieve good classification performance for real-world applications, or they do not provide interpretability of their output formulae: they generate long and complicated specifications.

The motivation of this dissertation is to address the following questions:

1. Can we infer time-series characteristics of dynamic systems through interpretable specifications that are easy to understand by humans, while maintaining accuracy in classifying the system traces?

2. Can we develop control algorithms, specifically for self-driving cars, that adhere to traffic rules, meet the real-world runtime performance requirement, and also fulfill the user's high-level objectives?

Since the main focus of this thesis revolves around studying the time-series behavior of dynamic systems, we look into answering these questions using *Signal Temporal Logic* (STL) specifications (Maler and Nickovic, 2004), which is a formal language to describe the temporal properties over real-valued signals with dense-time intervals.

We proceed to answer the first question by integrating STL specifications into two machine learning frameworks: offline learning and time-incremental learning. For each framework, we develop decision tree-based approaches (Breiman et al., 1984) with precise classification performance and interpretable outputs.

For the second question, first we tackle the problem in a deterministic environment and then we extend the scope of our studies to stochastic environments. We use STL specifications to describe traffic rules and we propose *Model Predictive Control (MPC)*-based algorithms (Camacho and Alba, 2013) to satisfy these specifications, the high-level objectives of user, and the runtime performance requirements.

1.2 Contributions

The contributions of this dissertation are in two following areas:

1. **Inference of temporal properties using STL classifiers:** We focus on a so-called *two-class (binary) classification problem*, aiming to infer an STL formula that can differentiate between system traces belonging to two distinct classes. The dataset comprises a finite collection of *signal-label* pairs, where each signal represents a system trace, and the corresponding label indicates whether the trace demonstrates a desired system behavior, such as the correct functioning of an engine (referred to as *supervised learning*). We address this problem in two machine learning frameworks:

   (a) **Offline learning:** In this context, our focus lies on the traditional *offline* supervised learning scenario, where the objective is to construct a classifier for assigning labels to the system traces. Inspired by the straightforward relation between decision trees and STL formulas (Bombara et al., 2016), and motivated by the classification accuracy improvement of boosting algorithms (Freund and Schapire, 1997), we propose a *boosted decision tree* algorithm to construct STL classifiers with better classification performance compared to existing works (Aasi et al., 2022). In addition, we empower our approach by a set of techniques, called *conciseness techniques*, to generate simpler STL formulae and improve the interpretability of the final output.

   (b) **Time-incremental learning:** We introduce the *time-incremental learning framework* to predict the labels of signals that are received incrementally over time, referred to as *prefix signals*. These signals are being observed as they are generated, and their time lengths are shorter than their

corresponding time horizons. Inspired by our proposed algorithm for the offline learning problem, we present an innovative method, based on decision trees, to infer a finite number of STL formulas from a provided dataset. Subsequently, we employ these formulas to construct a *predictor*. The predictor is developed by assigning *time-variant weights* to the STL formulas, signifying their respective classification impacts. These weights are determined through neural network training, aiming to minimize misclassification rates when categorizing prefix signals of varying time lengths (Aasi et al., 2023).

2. **Autonomous vehicle control with safe and real-time performance:** We consider the problem of controlling an autonomous vehicle, which we refer to as *ego*, along a given reference path, while satisfying traffic rules and avoiding collisions with other traffic participants. We consider a finite set of standard urban traffic rules, such as speed limit and traffic light, and express them using STL formulas. The high-level objective of ego is to adhere closely to the reference path, ensuring minimal energy consumption, and maximizing passenger comfort. We address this problem in two urban-driving frameworks:

   (a) **Deterministic environment:** Within this framework, we assume that ego's sensors provide deterministic sensor-measurements from the state of other traffic participants. Our approach involves a two-level control algorithm designed to navigate ego from its initial state to the desired goal. At the high-level, we formulate an MPC problem using simple representations of ego's dynamic model and its environment. Subsequently, at the low-level, we assess the correctness of the MPC's optimal control by comparing it against complex models, and apply small changes to it if needed (Aasi et al., 2021).

(b) **Stochastic environment:** In this context, we assume ego's sensor measurements from the state of other traffic participants are noisy. From ego's point of view, we consider the uncertainty in predicting the future motion of other vehicles in the form of posterior normal distributions, parameterized by their mean and covariance. Inspired by our solution to the deterministic control problem, we formulate a two-level MPC approach that uses simple models in the high-level and evaluates the correctness of MPC's output in the low-level against detailed models. In the case of violation of any traffic rule, a set of constraints, called *correction constraints*, are extracted and imposed on the MPC problem at the high-level to be solved again.

1.3 Organization

This thesis is organized as following:

Chapter 2: Preliminaries

This chapter provides a comprehensive overview of commonly used notations and a brief description of mathematical concepts that are used in this dissertation. In addition, it introduces the necessary background on signal temporal logic, its qualitative and quantitative semantics, and two of its extensions that are needed for formulating our algorithms.

Chapter 3: Offline Learning of Temporal Logic Classifiers

In this chapter, first we outline our motivation for the offline inference of STL specifications and present an overview of related studies in the literature. Subsequently, we mathematically define the learning problem and elucidate our approach to tackling it. Finally, we assess the performance of our method by comparing it to two existing works in the field.

Chapter 4: Time-Incremental Learning of Temporal Logic Classifiers

This chapter emphasizes the significance of time-incremental learning for STL formulae and present a survey of related literature. Then, the time-incremental learning problem is introduced in a formal manner, along with our approach to address it, which is inspired by our solution to the offline learning problem in Chapter 3. Subsequently, we evaluate the effectiveness of our algorithm by comparing it against two baseline methods.

Chapter 5: Autonomous Vehicle Control in Deterministic Environments

In this chapter, we delve into the realm of autonomous vehicle control. First, we provide an overview of the research conducted in this field and its corresponding challenges. Later, we mathematically define the problem of controlling a self-driving car along a reference path in a deterministic environment, while adhering to traffic rules and runtime constraints. Our approach to address this problem is thoroughly investigated, and we demonstrate its efficacy through four urban-driving simulation scenarios.

Chapter 6: Autonomous Vehicle Control in Stochastic Environments

This chapter broadens the focus of our research discussed in Chapter 5 from deterministic settings to stochastic ones. The expansion involves conducting a literature review of prior research in this domain and presenting a mathematical formulation of the problem. Subsequently, a comprehensive explanation of our algorithm is provided, along with demonstrations of its effectiveness through simulations based on real-world scenarios.

Chapter 7: Conclusions

This chapter summarizes the findings of this thesis, along with an overview of potential directions for future research.

Chapter 2

Preliminaries

In this chapter, we provide an overview of the notations used throughout this dissertation, in addition to a necessary background on signal temporal logic formulas.

2.1 Notations

Let $\mathbb{R}, \mathbb{R}_{\geq 0}, \mathbb{Z}$ and $\mathbb{Z}_{\geq 0}$ represent the sets of real, non-negative real, integer, and non-negative integer numbers, respectively. With slight abuse of notation, given $a, b \in \mathbb{Z}_{\geq 0}$, we use $[a, b] = \{t \in \mathbb{Z}_{\geq 0} \,|\, a \leq t \leq b\}$ to denote the inclusive set of non-negative integers between a and b. The cardinality of a set S is denoted by $|S|$, and the sign function is denoted by $\text{sign} : \mathbb{R} \to \{-1, 0, 1\}$. We show the empty set by $\varnothing$, the vector of zeros by $\emptyset$, and the vector of ones by $\mathbb{I}$ (the dimension should be clear from the context). We denote the Euclidean norm of a vector x by $||x||$, and its transpose by $x^\top$. The first and second order derivatives of a variable v with respect to time are denoted by $\dot{v}$ and $\ddot{v}$, respectively. We represent a diagonal matrix with *diag*.

A discrete-time signal s with time horizon $T \in \mathbb{Z}_{\geq 0}$ is a function $s : [0, T] \to \mathbb{R}^n$ that maps each discrete timepoint $t \in [0, T]$ to an n-dimensional vector $s(t)$ of real values. Each component of signal s is denoted by $s_j, j \in [1, n]$. Let $\ell \in C = \{C_p, C_n\}$ denote the label of a signal s, where C_p and C_n are the labels for the positive and negative classes, respectively. A labeled dataset with N signals is shown by $S = \{(s^i, \ell^i)\}_{i=1}^N$, where s^i and ℓ^i are the i^{th} signal and its label, respectively. The *prefix* of the signal s with time horizon t is the partial signal from the starting time point

0 up to time t, denoted by $s[0:t]$. A *prefix dataset* $S[0:t]$ with horizon t, is the dataset consisting of prefix signals with horizon t and their labels, denoted by $\{(s^i[0:t], \ell^i)\}_{i=1}^N$.

2.2 Signal Temporal Logic

Signal Temporal Logic (STL) was introduced in (Maler and Nickovic, 2004) to express temporal properties over real-valued signals with dense-time intervals, commonly used to describe behaviors of systems. The syntax of STL is defined as:

$$\phi ::= \top \mid \mu \mid \neg \phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathcal{U}_{[t_1,t_2]} \phi_2, \tag{2.1}$$

where ϕ, ϕ_1, and ϕ_2 are STL formulae, $\top$ is the logical *True* value, μ is a linear *predicate* over signal components in the form of $s_j \sim \pi$, where $\pi \in \mathbb{R}$ is the threshold and $\sim \in \{\geq, <\}$, $\neg$ and $\wedge$ are the Boolean *negation* and *conjunction* operators, respectively, and $\mathcal{U}_{[t_1,t_2]}$ is the temporal *until* operator, defined over the time interval $[t_1, t_2], t_1, t_2 \in \mathbb{Z}_{\geq 0}$ with $t_2 \geq t_1$. The other Boolean operators, e.g., *disjunction* $\vee$, *implication* $\longrightarrow$, and temporal operators, in particular *eventually* $\Diamond_{[t_1,t_2]}$ and *always* $\Box_{[t_1,t_2]}$, can be derived as (Maler and Nickovic, 2004):

$$\begin{aligned} \phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2), \quad \phi_1 \longrightarrow \phi_2 \equiv \neg\phi_1 \vee \phi_2 \\ \Diamond_{[t_1,t_2]} \phi \equiv \top \mathcal{U}_{[t_1,t_2]} \phi, \quad \Box_{[t_1,t_2]} \phi = \neg \Diamond_{[t_1,t_2]} \neg\phi \end{aligned} \tag{2.2}$$

The semantics of STL is defined over signals recursively. The satisfaction of an STL formula ϕ with respect to signal s at time t is denoted by $s[t] \models \phi$ and defined inductively as follows (Maler and Nickovic, 2004):

Definition 1 (STL Qualitative Semantics) *The Boolean semantics of STL for-*

mulas are defined as:

$$
\begin{aligned}
&s[t] \models \mu \equiv s_j[t] \sim \pi, \\
&s[t] \models \neg\phi \equiv s[t] \not\models \phi. \\
&s[t] \models \phi_1 \wedge \phi_2 \equiv s[t] \models \phi_1 \wedge s[t] \models \phi_2, \\
&s[t] \models \phi_1 \vee \phi_2 \equiv s[t] \models \phi_1 \vee s[t] \models \phi_2, \\
&s[t] \models \phi_1 \mathcal{U}_{[t_1,t_2]} \phi_2 \equiv \exists t' \in t + [t_1, t_2] : \ s[t'] \models \phi_2 \wedge \forall t'' \in [t_1, t'] : \ s[t''] \models \phi_1, \\
&s[t] \models \Diamond_{[t_1,t_2]} \phi \equiv \exists t' \in t + [t_1, t_2] : \ s[t'] \models \phi, \\
&s[t] \models \Box_{[t_1,t_2]} \phi \equiv \forall t' \in t + [t_1, t_2] : \ s[t'] \models \phi,
\end{aligned}
\tag{2.3}
$$

where $\not\models$ *denotes violation, and* $t + [t_1, t_2] \equiv [t + t_1, t + t_2]$.

In (Fainekos and Pappas, 2009), they equipped temporal logic with quantitative semantics, by replacing the binary satisfaction with a robustness degree function, while preserving the original syntax of the specification language. In particular, the robustness function gives a real value that shows how far a signal is from satisfying/violating a formula.

Definition 2 (STL Quantitative Semantics) *The quantitative semantics or the robustness score of an STL formula* ϕ *with respect to signal* s *at time* t *is recursively computed as (Fainekos and Pappas, 2009):*

$$
\begin{aligned}
&\rho(\top, s, t) := \rho_\top, \\
&\rho(\mu, s, t) := (-1)^i \left(s_j[t] - \pi\right), \\
&\rho(\neg\phi, s, t) := -\rho(\phi, s, t), \\
&\rho(\phi_1 \wedge \phi_2, s, t) := \min\left(\rho(\phi_1, s, t),\ \rho(\phi_2, s, t)\right), \\
&\rho(\phi_1 \vee \phi_2, s, t) := \max\left(\rho(\phi_1, s, t),\ \rho(\phi_2, s, t)\right), \\
&\rho(\phi_1 \mathcal{U}_{[t_1,t_2]} \phi_2, s, t) := \sup_{t' \in t + [t_1,t_2]} (\min(\rho(\phi_2, s, t'), \inf_{t'' \in [t_1,t']} \rho(\phi_1, s, t''))), \\
&\rho(\Diamond_{[t_1,t_2]} \phi, s, t) := \sup_{t' \in t + [t_1,t_2]} \rho(\phi, s, t'), \\
&\rho(\Box_{[t_1,t_2]} \phi, s, t) := \inf_{t' \in t + [t_1,t_2]} \rho(\phi, s, t'),
\end{aligned}
\tag{2.4}
$$

where $\rho_\top$ *is the maximum possible robustness, and* $i = 0$ *if* $\mu = s_j[t] \geq \pi$*, else* $i = 1$.

Theorem 1 (Soundness of Robustness) *The robustness score of STL is sound, meaning that* $\rho(\phi, s, t) > 0$ *implies that signal* s *satisfies formula* ϕ *at time* t*, and* $\rho(\phi, s, t) < 0$ *implies the violation of* ϕ *by* s *at time* t*.*

For simplicity of notation, we use $s \models \phi$ as short for $s[0] \models \phi$, and $\rho(\phi, s)$ as short for $\rho(\phi, s, 0)$, to denote that the signal s satisfies the formula ϕ at time 0. The minimum amount of time required to decide the satisfaction of a STL formula ϕ is called its horizon, and is denoted by $hrz(\phi)$. For example, the horizons of the two example formulas $\phi_1 = \Box_{[2,5]} s_1 < 4$ and $\phi_2 = \Diamond_{[3,10]} s_2 > 4$ are 5 and 10, respectively.

2.2.1 Parametric STL

Parametric STL (PSTL) (Asarin et al., 2011) is an extension of STL, where the endpoints t_1, t_2 of the time intervals in the temporal operators and the thresholds π in the predicates are parameters. The set of all possible valuations of all parameters in a PSTL formula ϑ is called the parameter space, and is denoted by Θ. A particular valuation of PSTL formula ϑ is denoted by $\theta \in \Theta$, and the corresponding STL formula is denoted by ϑ_θ.

2.2.2 Weighted STL

Weighted STL (wSTL) (Mehdipour et al., 2020) is another extension of STL with the same qualitative semantics as STL, but its robustness degree is modulated by the weights associated with the Boolean and temporal operators. In this thesis, we focus on a fragment of wSTL, with weights on conjunction operators only. For example, for the wSTL formula $\phi_1 \wedge^\alpha \phi_2$ with weights $\alpha = \{\alpha_1, \alpha_2\} \in \mathbb{R}^2_{\geq 0}$, its robustness is computed as $\min(\alpha_1 \,.\, \rho(\phi_1, s),\ \alpha_2 \,.\, \rho(\phi_2, s))$, where the weights capture the importance of each formula in computing the robustness.

Chapter 3

Offline Learning of Temporal Logic Classifiers

In this chapter, we investigate the inference of temporal logic specifications in a two-class classification problem, and present an algorithm for learning STL formulae in an offline machine learning framework.

3.1 Introduction

Machine learning involves creating algorithms capable of learning from data. These algorithms function by constructing classifiers using training data, which enables them to make precise predictions on unseen data (Ripley, 2007). In order to tackle the complexity of robotic tasks, machine learning algorithms have been adopted to grasp their temporal and logical structure from time-series data. One of the key challenges in the field of machine learning is the two-class classification problem, which involves constructing a classifier aimed at distinguishing the desired system behaviors from the unfavorable ones. The significance of this problem lies in its ability to address the broader multi-class classification problem. Additionally, it can be applied directly to tackle anomaly detection, which involves identifying anomalous patterns in data that deviate from the expected, target behaviors. In this chapter, we focus on an offline (supervised) learning framework, where a set of finite time series data, called signals or traces, and their corresponding labels are available, and the true class for each trace (either positive or negative) can be used during the training phase of the

machine learning algorithm.

Conventional machine learning algorithms primarily concentrate on constructing the classifiers, but they frequently lack interpretability and are challenging for humans to comprehend. Moreover, these algorithms tend to solely focus on creating the classifier itself, providing no insights into the underlying system (Bombara et al., 2016). Inspired by the understandable nature of temporal logic formulae (Clarke et al., 1986), there has been a significant interest in the application of formal methods to machine learning in recent years (Asarin et al., 2011; Kong et al., 2016). Signal Temporal Logic (STL) (Maler and Nickovic, 2004) is a specification language used for expressing temporal properties of real-valued signals. In this chapter, we use STL to generate specifications of the time-series system behaviors of dynamic systems.

In this context, efforts have been made to infer the temporal characteristics of time-series data. Nonetheless, the majority of current algorithms designed to learn STL formulae suffer from one of the following limitations:

1. The scope of their outputs is restricted, as they focus on finding the optimal parameters for a given formula structure;

2. They do not attain satisfactory classification performance in real-world applications;

3. They lack interpretability in their output formulae, often generating long and complicated specifications.

To address these concerns, motivated by the straightforward relation between decision trees and STL formulas (Bombara et al., 2016), and the accuracy enhancement of boosting algorithms (Freund and Schapire, 1997), we propose a new method called *Boosted Concise Decision Trees* (BCDTs) to infer STL formulae from time-series data (Aasi et al., 2022). The boosting method in machine learning involves com-

bining multiple models with weak classification power to construct a single, strong classifier. In our approach, we utilize bounded-depth decision trees, referred to as *Concise Decision Trees* (CDTs), as the weak learning models. Each CDT is a decision tree (Breiman et al., 1984), empowered by a set of techniques called conciseness techniques, to generate simpler formulae and improve the interpretability of the final output. Additionally, we employ a heuristic method within the BCDT algorithm to prune the ensemble of trees, further contributing to the interpretability of the output formulae. To make a connection between STL and BCDTs, we establish a relationship between boosted trees and weighted STL (wSTL) formulae, where wSTL involves weights associated with its Boolean and temporal operators (Mehdipour et al., 2020). Through two case studies involving naval surveillance and urban driving scenarios, we demonstrate the effectiveness and improved interpretability of our method compared to existing algorithms in the literature.

The main contributions of our study in this chapter are:

1. **Novel inference algorithm based on boosted decision trees**: The proposed algorithm builds an ensemble of decision trees, where each tree is trained to correct the errors of the previous trees. This approach has been shown to be effective for classification problems, and it achieves better performance than related approaches.

2. **Heuristic techniques to generate simple STL formulae**: The proposed algorithm uses a set of heuristic techniques to generate simple STL formulae from decision trees. These formulae are easier to understand and interpret than the decision trees themselves, which makes them more useful for practitioners.

3. **Case studies in naval surveillance and urban-driving**: The proposed algorithm is evaluated on two case studies in naval surveillance and urban-driving and compared with two of the existing works. The results show that

our algorithm achieves better classification performance on both datasets, and it is also able to generate simpler and more interpretable STL formulae.

3.2 Related Works

Motivated by the readability and interpretability of temporal logic formulae (Clarke et al., 1986), there has been a great interest in applying formal methods to machine learning algorithms in recent years (Ketenci and Gol, 2019; Yan and Julius, 2021; Kong et al., 2016; Bartocci et al., 2015; Yang et al., 2012). Initially, data-driven techniques for extracting temporal properties primarily emphasized parameter synthesis for a given formula structure (Asarin et al., 2011; Jin et al., 2015; Hoxha et al., 2018; Bakhirkin et al., 2018). The provided formula's structure demonstrates the designer's significant domain knowledge about the system and the specific properties they wish to investigate. This methodology, however, lacks the capacity to gain fresh insights about the system, as it necessitates the designer to precisely specify the form of the system properties under examination.

A general supervised learning framework is introduced in (Kong et al., 2014; Jones et al., 2014), which aims to infer both the structure and parameters of a formula. The authors defined a specific fragment of STL, known as inference parametric signal temporal logic. They demonstrated that this fragment allows for a partial order among formulae in two aspects: (1) concerning language inclusion and (2) in terms of the robustness degree. Their approach involves optimizing the formula structure first and then performing parameter synthesis through a nonlinear optimization problem. However, this generality comes at the cost of inefficiency in performance and computational resources. In (Bartocci et al., 2014; Bufo et al., 2014), they also addressed the task of two-class classification to deduce temporal logic formulas. Their method involves two distinct stages: initially, the formula structure is created using either

heuristics (Bartocci et al., 2014) or a genetic algorithm (Bufo et al., 2014), followed by an exploration of the parameter space using statistical model checking. However, this approach has some drawbacks. First and foremost, it necessitates the construction of models for the system being analyzed, demanding both a domain expert's involvement and a considerable amount of data. Second, the reported case studies required significant intervention from a designer to steer the process and achieve a satisfactory formula.

Using an efficient decision tree-based framework to learn STL formulae is explored in (Bombara et al., 2016; Bombara and Belta, 2021), where the nodes of the tree contain simple formulae that are tuned optimally from a predefined set of primitives, in a greedy growing procedure. The impurity measure is employed at each node to evaluate the optimality of its primitive, by assessing how well the primitive splits the signals in the training data. The algorithm they developed was among the pioneering efforts to integrate STL formulas with decision trees, and our current research in this chapter draws inspiration from their work. Despite achieving promising classification accuracy, the output formulae generated by their approach are lengthy and lack interpretability for humans.

The method introduced by (Mohammadinejad et al., 2020b) utilizes systematic enumeration to acquire concise and easily interpretable STL formulae. However, the achieved simplicity in their resulting formulae comes at the cost of increased misclassification rates.

Other works regarding the inference of temporal logic formulae in different machine learning frameworks include learning from positive examples only (Jha et al., 2019), clustering (Vazquez-Chanlatte et al., 2017), active learning (Linard and Tumova, 2020), and automata-based methods for untimed formulae (Neider and Gavran, 2018; Xu et al., 2019).

3.3 Offline Learning Problem

We want to find an STL formula that can distinguish between traces of a system that exhibit a desired property, such as behaving normally, and traces of the same system that do not exhibit this property. Let signal s^i denote the i^{th} trace of the system, and let $\ell^i \in C$ be its labels, where $C = \{C_p, C_n\}$ is the set of classes, with C_p for the positive class and C_n for the negative class. We consider the following problem:

Problem 1 (Offline Learning) *Given a labeled data set $S = \{(s^i, \ell^i)\}_{i=1}^N$, find an STL formula ϕ such that MisClassification Rate $MCR(\phi, S)$ defined below is minimized:*

$$MCR(\phi, S) = \frac{|\{s^i \mid (s^i \models \phi \wedge \ell^i = C_n) \vee (s^i \not\models \phi \wedge \ell^i = C_p)\}|}{N} \tag{3.1}$$

3.4 Offline Inference Approach

We propose a solution to Pb. 1 based on Boosted Concise Decision Tree (BCDT) method, presented in Sec. 3.4.1. The BCDT algorithm grows multiple binary Concise Decision Trees (CDTs), inspired by AdaBoost (Freund and Schapire, 1997) technique, where each CDT is a decision tree empowered by a set of conciseness techniques to generate simpler formulae. The construction method for a single CDT is inspired by the work in (Bombara et al., 2016) and is explained in Sec. 3.4.2. We describe the meta-parameters used in the construction of CDTs in Sec. 3.4.3, and in Sec. 3.4.4 we explain the conciseness techniques and their connection with interpretability of the inferred STL formulae.

3.4.1 Boosted Concise Decision Tree Algorithm

The BCDT algorithm described in Alg.1 draws inspiration from the AdaBoost technique (Shalev-Shwartz and Ben-David, 2014). AdaBoost combines weak classifiers with simple models, trained on weighted data samples. Weights of the data represent

the difficulty of correctly classifying them. Following the training of a weak classifier, the weights of correctly classified samples are reduced, while the weights of misclassified samples are increased. This algorithm requires three main inputs: the labeled dataset S, the number of learners K, and the weak learning model $\mathcal{E}$. In our proposed method, the weak learning models are the CDTs (explained in Alg.2). These CDTs, or binary decision trees, employ primitives as node formulas (see Sec. 3.4.3), with general rectangular predicates μ in the form of $As \leq b$. Here, $A = [\mathbb{I}_{n_1}\ , -\mathbb{I}_{n_2}]^T$, $b \in \mathbb{R}^{n_1+n_2}$, and $\mathbb{I}_n$ represents the $n \times n$ identity matrix, with $n_1, n_2 \in [0, n]$.

Algorithm 1 Boosted Concise Decision Tree (BCDT)

1: **Input:** $S = \{(s^i, \ell^i)\}_{i=1}^N$, K, $\mathcal{E}$
2: **Output:** final classifier $f_{BCDT}(\cdot)$
3: **Initialize:** $\forall\,(s^i, \ell^i) \in S:\ D_1(s^i) = 1/|S|$
4: **for** k = 1, ..., K:
5: classifier $f_{CDT}^k(\cdot) \leftarrow \mathcal{E}(S, D_k)$
6: $\epsilon_k \leftarrow \sum_{(s^i,\ell^i)\in S} D_k(s^i)\ \cdot\ \mathbf{1}[\ell^i \neq f_{CDT}^k(s^i)]$
7: $\alpha_k \leftarrow \begin{cases} \frac{1}{2}\ln\left(\frac{1}{\epsilon_k} - 1\right) & 0 < \epsilon_k \leq 1/2 \\ M & \epsilon_k = 0 \end{cases}$
8: $D_{k+1}(s^i) \propto D_k(s^i)\exp\left(-\alpha_k \cdot \ell^i \cdot f_{CDT}^k(s^i)\right)$
9: $f_{BCDT}(\cdot) \leftarrow \begin{cases} \text{sign}(\sum_{k=1}^K \alpha_k \cdot f_{CDT}^k(\cdot)) & \alpha_k < M,\ \forall k \\ f_{CDT}^{k^*}(\cdot) & \text{otherwise} \end{cases}$
10: **return** $f_{BCDT}(\cdot)$

In Alg. 1, the weights of all data samples are initialized equally (line 3). The algorithm iterates over the number of trees (line 4). At each iteration, a single CDT $f_{CDT}^k(\cdot)$ is constructed, using the weak learning algorithm $\mathcal{E}$ and based on the data set S and current samples' weights D_k (line 5). Next, the misclassification error of the built tree ϵ_k is computed (line 6). If the current tree has a weak classification performance better than random guessing ($0 < \epsilon_k \leq 1/2$), its weight is computed based on the original AdaBoost method, and if it has a perfect classification perfor-

mance and classifies all signals correctly ($\epsilon_k = 0$), a big, positive weight M is assigned to it (line 7). At the end of each iteration, the samples' weights are updated and normalized (denoted by $\propto$) based on the performance of the current tree, to focus on the misclassified signals in the next trees (line 8).

In order to determine the final output of the algorithm, we adopt a heuristic technique to trim the collection of trees, resulting in more straightforward formulas and enhanced interpretability. This technique draws inspiration from the heuristic methods used for pruning an ensemble of decision trees in prior works such as (Breiman et al., 1984; Kulkarni and Sinha, 2012). We compute the final output of the BCDT algorithm $f_{BCDT}(\cdot)$ (line 9) as following:

(a) if the weights of all trees are less than M, the final output is determined through a weighted majority vote among all the CDTs, following a similar approach to the AdaBoost method;

(b) if there exist one or more trees with weight M, the final output is derived from the tree with weight M that possesses the simplest STL formula, denoted as $f^{k^*}_{CDT}(\cdot)$.

The number of Boolean and temporal operators is considered as a metric for comparing the simplicity of formulae. This pruning technique is beneficial in reducing the generalization error during the test phase and generating more straightforward formulae. The empirical results supporting its advantages are presented in Sec.3.5.

The final output $f_{BCDT}(\cdot)$ assigns a label to each data sample. For simplicity, we conveniently represent $C_p = 1$ and $C_n = -1$, such that $f^k_{CDT}(\cdot) \in \{-1, 1\}$ for all $k \in [1, K]$. It's important to note that a key assumption in boosting methods is that each weak learner performs slightly better than random guessing (similar to coin tossing). Therefore, in Alg.1, if any newly generated tree performs worse than

random guessing ($\epsilon_k > 0.5$), we discard it and generate another tree. An illustration of Alg. 1 is shown in Fig. 3·1.

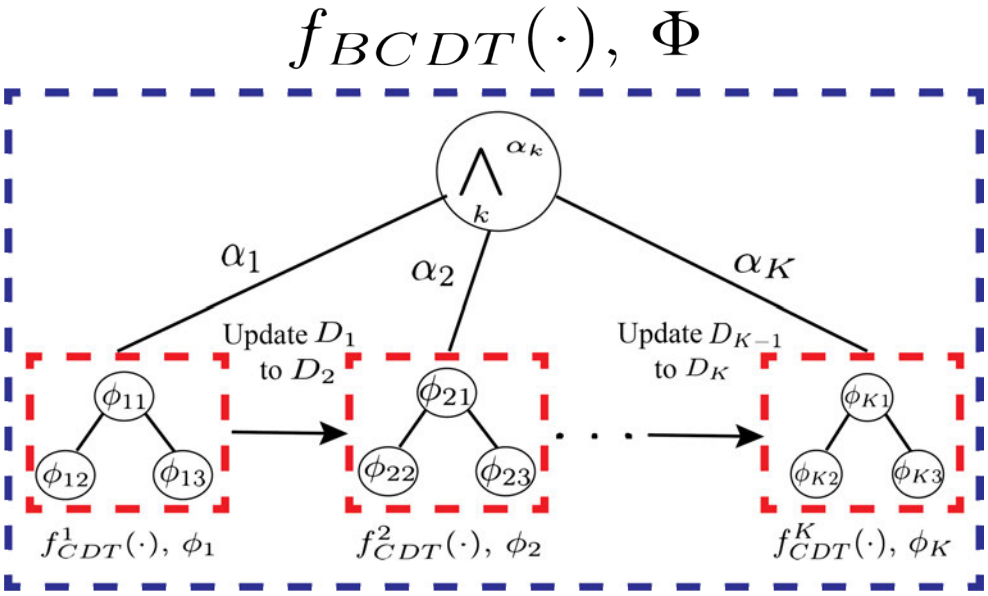


Figure 3·1: Illustration of BCDT Alg. 1 (Aasi et al., 2022). The CDTs and their weights are used in construction of the final classifier $f_{BCDT}(.)$, and its corresponding formula Φ. Here we have assumed $\forall\, k \in [0, K] : \alpha_k < M$.

We use the method from (Bombara et al., 2016) to convert each CDT $f^k_{CDT}(\cdot)$ to a corresponding STL formula ϕ_k. The algorithm is invoked from the root, and builds the formula that captures all the branches that end in leaves marked C_p. The output of the BCDT method is translated to a set of formulae and associated weights $\{(\phi_k, \alpha_k)\}_{k=1}^K$. The STL formula $\Phi = \bigwedge_k \phi_k$ is the overall output formula; however, using wSTL (Mehdipour et al., 2020) we express $\Phi = \bigwedge_k{}^{\alpha_k} \phi_k$, to capture the classification performance of each CDT.

3.4.2 Construction of Concise Decision Trees

Decision trees (Breiman et al., 1984; Ripley, 2007) are sequential decision models with hierarchical structures. In our approach, decision trees operate on the signals

with the goal of predicting their labels. Inspired by (Bombara and Belta, 2021), we present the Concise Decision Tree (CDT) method $\mathcal{E}$ in Alg. 2, which extends their decision tree construction algorithm to CDTs, by applying conciseness techniques to generate simpler formulae (detailed in Sec. 3.4.4).

Algorithm 2 Concise Decision Tree (CDT) method $\mathcal{E}$

Meta-Parameters: $\mathcal{P}, \mathcal{J}, stop$
Input: S, ϕ^{path}, h, ϕ^{c}_{parent}
Output: sub-tree $\mathcal{T}$
if $stop(\phi^{path}, h, S)$ **then**
    $c = \mathcal{O}(S, \phi^{path}, \mathcal{P}, h)$
    return $leaf(c)$
$\mathcal{T} \leftarrow non\_terminal(\phi^{c}_{parent})$
$\phi^{new} = \phi^{path} \wedge \phi^{c}_{parent}$
$S_{\top}, S_{\bot} \leftarrow partition(S, \phi^{new})$
for $\otimes \in \{\top, \bot\}$ **do**
    $\phi^{c}_{\otimes} = \mathcal{O}(S_{\otimes}, \phi^{new}, \mathcal{P}, h+1)$
    $\phi^{\otimes} = \mathcal{C}(\phi^{c}_{parent}, \phi^{c}_{\otimes}, S, \phi^{path}, h)$
    if $\phi^{path} \wedge \phi^{\otimes} \succeq^{\mathcal{J}} \phi^{new}$:
        return $\mathcal{E}(S, \phi^{path}, h, \phi^{\otimes})$
$\mathcal{T}.left \leftarrow \mathcal{E}(S_{\top}, \phi^{new}, h+1, \phi^{c}_{\top})$
$\mathcal{T}.right \leftarrow \mathcal{E}(S_{\bot}, \phi^{new}, h+1, \phi^{c}_{\bot})$
return $\mathcal{T}$

In order to control the complexity of CDTs in Alg. 2, three meta-parameters are taken into account: (1) PSTL primitives $\mathcal{P}$ which define the various ways to partition the data at each node, (2) impurity measures $\mathcal{J}$ utilized to determine the optimal primitive for each node, and (3) stop conditions *stop* that restrict the growth of the CDTs. These meta-parameters are explained in details in Sec. 3.4.3.

Alg. 2 is recursive, and takes as input (1) the set of labeled signals S at the current node, referred to as *parent* node, (2) the path formula ϕ^{path} from the root to the parent node, (3) the depth h from the root to the node, and (4) the candidate formula ϕ^{c}_{parent}

for the node. At the beginning, the algorithm evaluates the stop conditions *stop* (line 4). Should these conditions be met (lines 5-6), a singular leaf marked with label c is returned, using the primitive optimization method described in Alg. 3. In the event that the conditions are not fulfilled, a non-terminal node is generated and associated with the candidate formula ϕ^c_{parent} (line 7). The formula ϕ^{new} denotes the updated path formula from the root, incorporating the candidate primitive ϕ^c_{parent} of the parent node (line 8). Subsequently, the data set S is partitioned based on the new formula (line 9), resulting in $S_\top$ and $S_\bot$, representing the sets of signals that satisfy and violate ϕ^{new}, respectively.

Following the logical structure of the tree, initially for the left child of the node ($\otimes = \top$) and subsequently for the right child ($\otimes = \bot$), we follow these steps (line 10): firstly, the candidate primitive $\phi^c_\otimes$ for the child is computed from the set $\mathcal{P}$ (line 11). Next, we utilize the conciseness method $\mathcal{C}$ (explained in Sec. 3.4.4) on the combination of the parent's candidate formula ϕ^c_{parent} and the child's candidate primitive $\phi^c_\otimes$ to obtain a new formula $\phi^\otimes$ (line 12) which serves as a new candidate for the parent node. In line 13, the notation $\succeq^{\mathcal{J}}$ is utilized to compare two formulae, considering the impurity measure $\mathcal{J}$. If the impurity reduction of the new candidate formula $\phi^\otimes$ is more than the previous candidate ϕ^c_{parent}, the algorithm is repeated for the parent node, with ϕ^c_{parent} replaced by $\phi^\otimes$ (line 14). It is worth noting that the decision tree approach presented in (Bombara and Belta, 2021) is built on the concept of incremental impurity reduction at each node of the tree. Following the same concept, we contend that by applying the conciseness techniques at each node, if the impurity reduction achieved by the new candidate formula outperforms the previous one, the new candidate results in a more robust classifier with a simpler specification.

Ultimately, once the conciseness method can no longer be applied to the parent

node, we continue building the tree for the left and right children (lines 15-16) and then the sub-tree for the parent is returned (line 17).

Algorithm 3 Parameterized Primitive Optimization $\mathcal{O}$

1: **Meta-Parameters:** $\mathcal{J}, stop$
2: **Input:** S, ϕ^{path}, $prim$, h
3: **Output:** optimal primitive ϕ^*
4: **if** $stop(\phi^{path}, h, S)$ **then**
5: $\quad \phi^* \leftarrow arg\max_{c \in C}\{p(S, c; \phi^{path})\}$
6: **else**
7: $\quad \phi^* = \underset{\vartheta \in prim,\, \theta \in \Theta}{\operatorname{argmax}} \mathcal{J}(S, partition(S, \phi^{path} \wedge \vartheta_\theta))$
8: **return** ϕ^*

The algorithm $\mathcal{O}$ for parameterized primitive optimization, as presented in Alg.3, is utilized to discover the best primitive with the optimal evaluation from the input primitive set $prim$. This method has similar meta parameters as Alg. 2 and takes the following inputs: (1) the set of labeled signals S at the current node, (2) the path formula ϕ^{path} from the root to the current node, (3) a set of input primitives denoted by $prim$, and (4) the depth h from the root to the node. If the stop conditions are met (line 4), a label $c^* \in C$ is computed (line 5) based on the best classification quality using the partition weight $p(S, c; \phi^{path})$ of the impurity measure (see Sec. 3.4.3). Otherwise, if the stop conditions are not satisfied, the algorithm computes the best primitive from the primitive set $prim$ by solving an optimization problem that relies on the impurity measure $\mathcal{J}$.

3.4.3 Meta Parameters

Here we explain the meta-parameters that are used in the construction of the CDTs in Alg. 2 and 3:

1. PSTL primitives $\mathcal{P}$: The primitives, which are simple PSTL formulae, are utilized as splitting rules at each node (Bombara and Belta, 2021). Here we use

first-order primitives $\mathcal{P}$: $\{\Box_{[t_0,t_1]}(s_j \sim \pi), \Diamond_{[t_0,t_1]}(s_j \sim \pi)\}$, where the decision parameters are (t_0, t_1, π) and the space of parameters is $\{(a,b) \,|\, a < b,\, a, b \in \mathbb{R}_{\geq 0}\} \times \mathbb{R}$.

2. Impurity measure $\mathcal{J}$: At each node, we utilize the *Misclassification Gain* (MG) impurity measure (Breiman et al., 1984) as the criterion to determine the optimal primitive selection. Given a finite set of signals S, an STL formula ϕ, and $(S_\top, S_\bot)$ as the subsets of S that are partitioned based on the satisfaction of ϕ ($S_\top, S_\bot = \text{partition}(S, \phi)$), we have (Bombara et al., 2016):

$$\begin{aligned} MG(S, \{S_\top, S_\bot\}) &= MR(S) - \sum_{\otimes \in \{\top, \bot\}} p_\otimes \, MR(S_\otimes), \\ MR(S) &= \min(p(S, C_p; \phi)\,,\, p(S, C_n; \phi)), \end{aligned} \tag{3.2}$$

where the p parameters are the partition weights computed based on the signals' labels and satisfaction of ϕ:

$$p_\top = \frac{|S_\top|}{S}, \quad p_\bot = \frac{|S_\bot|}{S}, \quad p(S, c; \phi) = \frac{|\{(s^i, \ell^i) \,|\, \ell^i = c\}|}{|S|} \tag{3.3}$$

Specifically, $p_\top$ and $p_\bot$ denote the proportions of signals originating from S that are found in $S_\top$ and $S_\bot$, respectively. Moreover, $p(S, c; \phi)$ indicates the proportion of signals in S that are categorized under class $c \in C$.

In (Bombara et al., 2016), they extended the notion of impurity measure to include the robustness of satisfaction in the partition weights as below:

$$\begin{aligned} p_\top &= \frac{\sum_{s^i \in S_\top} \rho(\phi, s^i)}{\sum_{s^i \in S} |\rho(\phi, s^i)|}, \qquad p_\bot = -\frac{\sum_{s^i \in S_\bot} \rho(\phi, s^i)}{\sum_{s^i \in S} |\rho(\phi, s^i)|}, \\ p(S, c; \phi) &= \frac{\sum_{s^i \in S_c} |\rho(\phi, s^i)|}{\sum_{s^i \in S} |\rho(\phi, s^i)|}; \quad S_c = \{s^i \in S \,|\, \ell^i = c\} \end{aligned} \tag{3.4}$$

Here, we extend the robustness-based impurity measure in (3.4) to account for the sample weights D_k from the BCDT algorithm in Alg. 1. We define the

boosted impurity measure using the partition weights below:

$$
\begin{aligned}
p_{\otimes} &= \frac{\sum_{(s^i,\ell^i)\in S_{\otimes}} D_k(s^i)\cdot\rho(\phi,s^i)}{\sum_{(s^i,\ell^i)\in S} D_k(s^i)\cdot|\rho(\phi,s^i)|},\ \ \otimes\in\{\top,\bot\} \\
p(S,c;\phi) &= \frac{\sum_{(s^i,\ell^i)\in S,\,\ell^i=c} D_k(s^i)\cdot|\rho(\phi,s^i)|}{\sum_{(s^i,\ell^i)\in S} D_k(s^i)\cdot|\rho(\phi,s^i)|}
\end{aligned}
\tag{3.5}
$$

In this chapter, we use the MG boosted impurity measure in our calculations.

3. Stop conditions: Alg. 2 can be terminated based on various stopping conditions. Here we consider the growth of trees is halted either when they reach a specified depth, or when λ percent of the signals are classified into a single class. In our implementations, we set $\lambda = 95\%$.

3.4.4 Conciseness Techniques

We propose the conciseness method $\mathcal{C}$, presented in Alg. 4, aimed at enhancing the simplicity and interpretability of STL formulae. The algorithm takes as input (line 1) the candidate primitive ϕ^c_{parent} for the parent node, the candidate primitive for its child (either left or right child) $\phi^c_{\otimes}, \otimes \in \{\top,\bot\}$, the set of signals S, path formula ϕ^{path}, and depth h of the parent node. As the output, the algorithm produces a new candidate primitive, denoted as ϕ^c_{new}, for the parent node (line 2).

Algorithm 4 Conciseness Method $\mathcal{C}$

1: **Input:** $\phi^c_{parent}, \phi^c_{\otimes}, S,\ \phi^{path},\ h$
2: **Output:** new candidate primitive ϕ^c_{new}
3: $\phi_{parent} = \phi^c_{parent} \biguplus \phi^c_{\otimes}$
4: $\phi^c_{new} = \mathcal{O}(S, \phi^{path}, \phi_{parent}, h)$
5: **return** ϕ^c_{new}

Initially, the algorithm creates a new PSTL primitive represented as ϕ_{parent} for the parent node. This is achieved by combining the candidate primitives of both the parent and child nodes, using the combination operator $\biguplus$ (line 3). This process

involves examining various methods of combining two candidate primitives, for which we introduce two heuristic techniques. Subsequently, the optimal assessment of the resulting PSTL primitive is determined using the optimization method $\mathcal{O}$ and the path formula ϕ^{path} (line 4).

This is done by considering the possible ways to combine two candidate primitives, which we propose two heuristic techniques for it. Then, the optimal valuation of the new PSTL primitive is computed by using the optimization method $\mathcal{O}$ and the path formula ϕ^{path} (line 4).

The heuristic techniques to combine two primitives and generate shorter PSTL formulae are as following:

Combination of Always operators

If the candidate primitives of the parent and child nodes are as $\phi^c_{parent} = \Box_{[t_0,t_1]}\mu_{parent}$ and $\phi^c_{\otimes} = \Box_{[t_2,t_3]}(\mu_{child})$, respectively, we construct a new PSTL primitive $\phi_{parent} = \Box_{[t_4,t_5]}((\mu_{parent}) \wedge (\mu_{child}))$ for their combination. For example, given:

$$\phi^c_{parent} = \Box_{[t_0,t_1]}((s_1 > \pi_1) \wedge (s_2 \leq \pi_2)), \quad \phi^c_{\otimes} = \Box_{[t_2,t_3]}(s_2 > \pi_3),$$

their combined PSTL primitive would be $\phi_{parent} = \Box_{[t_4,t_5]}((s_1 > \pi_1) \wedge (\pi_3 < s_2 \leq \pi_2))$.

Combination of Eventually operators

Similar to the combination of always operators, if the candidate primitives of the parent and child nodes are as $\phi^c_{parent} = \Diamond_{[t_0,t_1]}\mu_{parent}$ and $\phi^c_{\otimes} = \Diamond_{[t_2,t_3]}(\mu_{child})$, respectively, we construct a new PSTL primitive as $\phi_{parent} = \Diamond_{[t_4,t_5]}((\mu_{parent}) \wedge (\mu_{child}))$.

Example 1 *Here we provide an example of how Alg. 4 works, using the illustration in Fig. 3·2. This example is based on the dataset of the naval surveillance case study in Sec. 3.5.1. Assume we have constructed a tree, shown on the left, where the candidate primitive for the parent node 1 is* $\phi^c_{parent} = \Box_{[18,58]}(y \leq 32.55)$*, and after partitioning the signals, the candidate primitive for the left child (node 2) is computed as* $\phi^c_{\top} =$

$\square_{[22,48]}(y > 23.33)$. *By applying the Alg. 4, the conciseness method* $\mathcal{C}$ *constructs a new candidate PSTL primitive* $\phi_{parent} = \square_{[t_0,t_1]}(\pi_0 < y \leq \pi_1)$ *for their combination, where its optimal valuation according to the primitive optimization method* $\mathcal{O}$ *would be* $\phi^{\top} = \square_{[17,60]}(23.45 < y \leq 33.66)$. *Due to the higher impurity reduction of* $\phi^{\top}$ *compared to* ϕ^{c}_{parent}, *the new candidate* $\phi^{\top}$ *is chosen as* ϕ^{c}_{parent} *for the parent node 1 on the right. The partitioning of the signals and the candidate primitives for the left and right children (nodes 2 and 3) are recomputed according to the new candidate primitive* ϕ^{c}_{parent}. *According to the conciseness technique* $\mathcal{C}$, *there is no more possibility of combining the candidate primitives of the parent node and its children, because the temporal operators of the parent and its children nodes are different. Therefore, the procedure of CDT construction is followed up from the left and right children.*

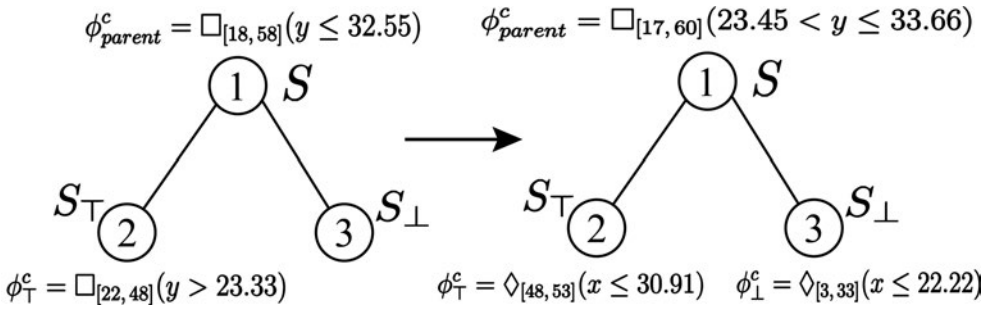


Figure 3·2: Example of applying the conciseness method $\mathcal{C}$ on the naval surveillance data set.

Remark 1 *There are multiple ways of combining the primitives to improve the interpretability of the final formula. For example, given the candidate primitive of the parent node as* $\phi^{c}_{parent} = \square_{[t_0,t_1]}(\mu_{parent})$, *and the candidate primitive of its child as* $\phi^{c}_{\otimes} = \Diamond_{[t_2,t_3]}(\mu_{child})$, *we can construct a new PSTL primitive for the parent node as* $\phi_{parent} = \Diamond_{[t_4,t_5]}((\mu_{parent})\,\mathcal{U}_{[0,t_6]}\,(\mu_{child}))$. *We will explore these other ways of combining the primitives in the future works (see Sec. 7.2).*

Remark 2 *Note that our heuristic method combines two primitives whenever their combination improves the impurity measure and classification performance. This leads to a larger set of primitives for constructing the trees, and it is more efficient compared to the naive approach of investigating all possible combinations of primitives.*

3.4.5 Complexity Analysis

We denote the lower and the two-sided asymptotic bounds for the complexity of the overall algorithm by $\Omega(.)$ and $\Theta(\cdot)$, respectively. The complexity of the BCDT algorithm (Alg. 1) is equivalent to the complexity of the AdaBoost method with K trees $O(K\,.\,C_{\mathcal{E}}(N))$, where $C_{\mathcal{E}}(N)$ is the complexity of constructing a CDT by Alg. 2, and N is the number of signals to be processed. Let $C_{\mathcal{O}}(N)$ be the complexity of the optimization method in Alg. 3. Clearly we have $C_{\mathcal{O}}(N) = \Omega(N)$, because the method must at least check the labels of all signals (Cormen et al., 2009). The worst-case complexity of Alg. 2 is obtained when at each node the optimal partition has size $(1, N-1)$, and we run the conciseness method (Alg. 4) for each child of the node, which leads to $2C_{\mathcal{O}}(N)$. Using the recursive nature of decision trees, the complexity analysis of (Bombara et al., 2016) and the Akra-Bazzi method (Cormen et al., 2009), for the worst-case and average-case complexity of $C_{\mathcal{E}}(N)$ we have $\Theta(N + 4\sum_{k=2}^{N} C_{\mathcal{O}}(k))$ and $\Theta(N \cdot (1 + 2\int_1^N \frac{C_{\mathcal{O}}(u)}{u^2})du)$, respectively.

3.5 Case Studies

We demonstrate the effectiveness and computational advantages of our method with two case studies. The first is the naval surveillance scenario from (Kong et al., 2016). The second is an urban-driving scenario, implemented in the simulator CARLA (Dosovitskiy et al., 2017). We use Particle Swarm Optimization (PSO) method (Kennedy and Eberhart, 1995) for solving the optimization problems in Alg. 3. The parameters of the PSO method are tuned empirically and we use $M = 100$ in our implementations. We run the case studies on a 3.70 GHz processor with 16 GB RAM.

3.5.1 Naval Surveillance

Consider the maritime surveillance scenario from (Kong et al., 2016; Bombara and Belta, 2021) (see Fig. 3·3). The goal is to detect anomalous vessel behaviors by looking at their trajectories. A vessel behaving normally approaches from the open sea and heads directly towards the harbor, while a vessel with anomalous behaviors either veers to the island and then heads to the harbor, or it approaches other vessels in the passage between the peninsula and the island and then returns to the open sea. In the scenario's dataset (Bombara and Belta, 2021), the signals are represented as 2-dimensional trajectories with planar coordinates $(x(t), y(t))$. The labels indicate the type of a vessel's behavior (normal or anomalous). The dataset of the scenario is composed of 2000 signals, with 1000 normal and 1000 anomalous trajectories, and each signal has 61 timepoints.

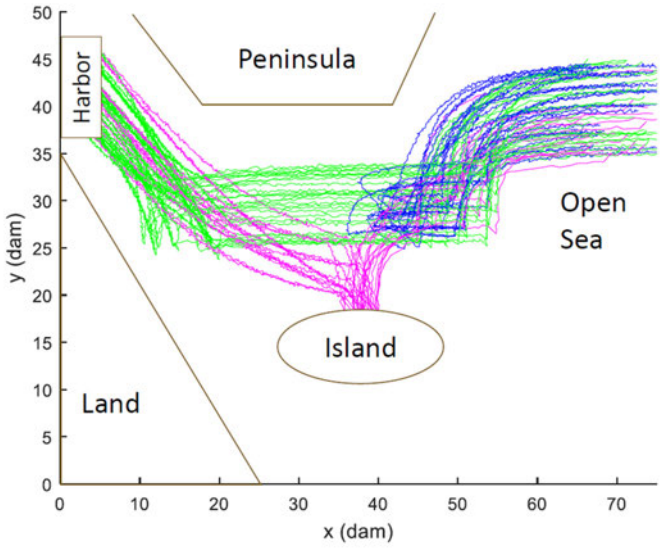


Figure 3·3: Naval surveillance scenario (Kong et al., 2016), where normal trajectories are shown in green, and anomalous signals are shown in blue and magenta.

We compare the performance of our inference algorithm with the methods from (Bombara and Belta, 2021) (the DTL4STL tool) and (Mohammadinejad et al., 2020b).

We evaluate our approach with 5-fold cross validation and maximum depth = 3 for the trees (as in (Bombara and Belta, 2021)). The results are provided in Table. 3.1 for different number of decision trees K in Alg. 1. In this table, TR-M(%) and TR-S(%) are the mean and standard deviation of the MCR in the training phase, respectively; TE-M(%) and TE-S(%) are the mean and standard deviation of the MCR in the test phase; R is the runtime, and CT is the number of times that by applying the conciseness method $\mathcal{C}$ during the construction of CDTs, a simpler formula is found.

Table 3.1: Classification performance of BCDT algorithm in Naval Surveillance case study

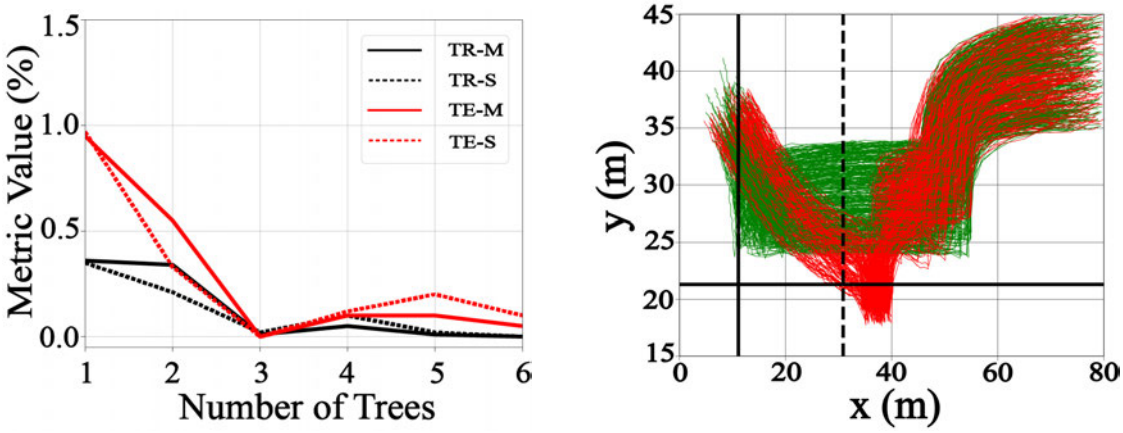
K	TR-M (%)	TR-S (%)	TE-M (%)	TE-S (%)	R	CT
1	0.36	0.35	0.95	0.97	11m 8s	4
2	0.34	0.21	0.55	0.33	30m 47s	14
3	0.01	0.02	0.00	0.00	33m 16s	10
4	0.05	0.10	0.10	0.12	61m 33s	29
5	0.01	0.02	0.10	0.20	81m 52s	33
6	0.00	0.00	0.05	0.10	85m 55s	38

An illustration of the classification performance of our method in Table. 3.1, with respect to the number of employed CDTs, is shown in Fig. 3·4 on left. Based on the results shown in Table. 3.1, it is evident that the optimal classification performance, achieved during both the training and test phases, occurs with $K = 3$. At this value, we find a set of concise trees that are capable of accurately classifying all signals during the test phase. It's worth noting that increasing the number of trees enhances the complexity of the framework and enables the capture of finer dataset details,

but it also introduces the risk of overfitting, as the TE-M increases for $K > 3$ (see Table 3.1). An example of the formula learned in one of the folds with $K = 3$ is:

$$\Phi^{Naval} = \phi_1 \wedge \phi_2 \ , \quad \phi_1 = \Diamond_{[28,53]}(x \leq 30.85), \\ \phi_2 = \Box_{[2,26]}((y > 21.31) \wedge (x > 11.10)) \tag{3.6}$$

and its thresholds are shown in Fig. 3·4 on right.



(a) (b)

Figure 3·4: (a) Classification performance of our method, with respect to the number of CDTs, (b) Examples of trajectories from the naval surveillance case study. The green and red trajectories belong to normal and anomalous behaviors, respectively. For formula Φ^{Naval}, the thresholds of the always and eventually operators are shown by solid and dashed black lines, respectively.

In (Bombara and Belta, 2021), using first-order primitives and maximum tree depth of 3, the authors achieved a mean MCR of 1.3% with a standard deviation of 0.28% for this data set. To ensure a fair comparison, we ran the algorithm from (Bombara and Belta, 2021) on the same computer that we used for our algorithm and for the same data set. By running their algorithm, we obtained a test-phase MCR with a mean of 1.5% and a standard deviation of 0.5%, with total runtime of 33 seconds. An example formula learned in one of the folds using the method from

(Bombara and Belta, 2021) is:

$$(\phi_1 \wedge \phi_2) \vee (\neg\phi_1 \wedge ((\phi_3 \wedge \phi_4) \vee (\neg\phi_3 \wedge \phi_5)))$$
$$\phi_1 = \square_{[39,60]}(x \leq 19.5), \quad \phi_2 = \Diamond_{[11,38]}(x > 41.2), \quad \phi_3 = \square_{[20,59]}(y < 32.3)$$
$$\phi_4 = \square_{[59,60]}(x \leq 39.2), \quad \phi_5 = \square_{[20,53]}(y > 29.7)$$

Compared to (Bombara and Belta, 2021), our algorithm achieves superior classification performance while also producing simpler and more interpretable formulas. However, it is worth noting that this comes at the expense of increased runtime due to the utilization of boosting and conciseness techniques.

In the work by (Mohammadinejad et al., 2020b), the authors achieved a mean MCR of 5% during the test phase, with a total runtime of 45 minutes. The formula they derived in their study is expressed as $(y \geq 19.74)\,\mathcal{U}_{[0,9.84]}\,(x \leq 24.86)$. In terms of interpretability, both their algorithm's learned formula and ours are straightforward and easily understandable. Moreover, both methods exhibit comparable runtimes. However, it is worth noting that our algorithm demonstrates significantly improved classification performance.

3.5.2 Urban Driving

Consider an autonomous vehicle (referred to as *ego*) navigating through an urban environment as depicted in Fig. 3·5. The scenario involves a pedestrian and another car, which is presumed to be driven by a law-abiding human. Ego and the other car occupy different but adjacent lanes, both moving in the same direction. Both cars are ascending in the $y - z$ plane of the coordinate frame, with their motion directed towards positive y and z coordinates, while remaining stationary in the lateral x direction. Both vehicles maintain constant acceleration, with ego exhibiting a slightly smaller acceleration compared to the other car.

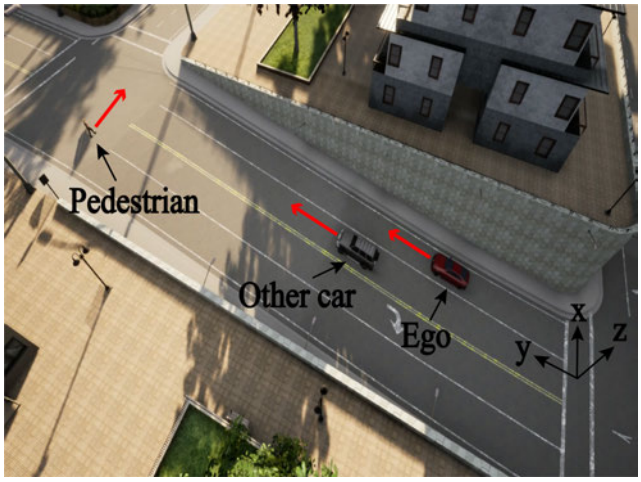


Figure 3·5: Urban driving scenario implemented in the simulator CARLA (Dosovitskiy et al., 2017).

The cars' positions and accelerations are initially set up in a way that the other car is consistently ahead of ego. Both vehicles are approaching an intersection with no traffic light. At the end of the road, just before the intersection, there is an unmarked crosswalk. If a pedestrian crosses the street at the crosswalk, the other car applies the brakes and comes to a stop before reaching the intersection. However, if the pedestrian doesn't cross, the other car continues its movement without reducing its speed. Ego's view of the pedestrian crossing at the intersection is obstructed due to the presence of the other car and the uphill shape of the road. The main objective is to devise a method that enables ego to deduce whether a pedestrian is crossing the street by observing the behavior of the other car, such as the relative position and velocity changes over time.

The simulation of the scenario concludes whenever the distance between the ego and the intersection becomes less than 8 meters. We make the assumption that the necessary labeled behaviors, including relative distances and velocities, are accessible. These labels indicate whether a pedestrian is currently crossing the street or not. Our dataset consists of 300 signals, each containing 500 uniformly sampled time-points

per trace. Among these signals, 150 of them include pedestrians crossing the street, while the other 150 do not. The dataset is accessible at (Aasi, 2022). To evaluate our algorithm, we employ 5-fold cross-validation and set the maximum depth of the trees to 2. The results for different values of K are presented in Table 3.2.

Table 3.2: Classification performance of BCDT method in Urban Driving scenario

K	TR-M (%)	TR-S (%)	TE-M (%)	TE-S (%)	R	CT
1	0.00	0.00	1.00	1.33	7m 10s	2
2	0.00	0.00	0.67	0.82	9m 57s	2
3	0.00	0.00	0.33	0.66	14m 52s	1
4	0.00	0.00	0.00	0.00	24m 40s	3
5	0.00	0.00	1.00	1.33	24m 49s	3
6	0.00	0.00	0.33	0.66	32m 52s	3

Based on the data presented in Table 3.2, it is evident that our method achieves optimal performance when $K = 4$. An illustrative instance of the formula acquired in one of the folds under this condition is as follows:

$$\Phi^{Urban} = \Diamond_{[370,485]}((y \leq 14.01) \wedge (v_y > 7.45))$$

and its thresholds are shown in Fig. 3·6.

Notice that the primary aim of this scenario is to deduce whether a pedestrian is crossing the street based on the behavior of the other car as it approaches the intersection. Consequently, we anticipate that the desired specifications should be concise, considering signals close to the end of the simulation time intervals. Our method produces straightforward and easily understandable output formulae. For

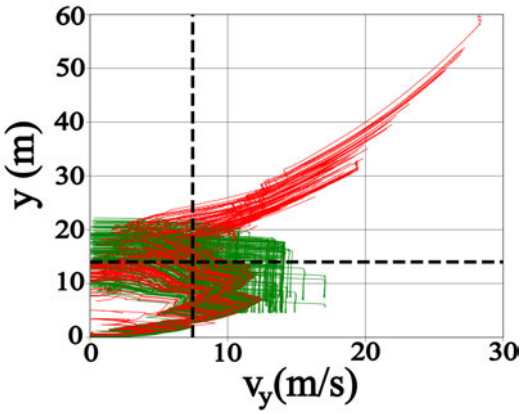


Figure 3·6: Examples of signals from the urban driving case study. The green and red signals belong to the cases when there is a pedestrian crossing the street and when there is no pedestrian crossing, respectively. The thresholds of the formula Φ^{Urban} are shown by dashed black lines.

instance, Φ^{Urban} indicates a pedestrian crossing if, "at some point within the time interval [370, 485], the distance between the vehicles in the $y-$direction is less than 14.01m, and the y component of ego's velocity exceeds the corresponding component of the other car by 7.45m/s." This effectively means that the other car comes to a halt at the intersection due to a pedestrian crossing, while ego approaches the other car, causing the relative distance in the $y-$direction to decrease, and ego's velocity to surpass that of the other car.

In order to ensure a fair comparison, we assessed the algorithm's performance from (Bombara and Belta, 2021) on the same dataset and computer. Employing first-order primitives, 5-fold cross-validation, and a maximum tree depth of 2 for the algorithm in (Bombara and Belta, 2021), we achieved a mean MCR of 1% with a standard deviation of 1.5% during the test phase, completing in a total runtime of 7.72 seconds. One illustrative formula learned in a fold using the method from (Bombara and Belta, 2021) is $\Diamond_{[474,499]}(z < 1.2) \wedge \Diamond_{[0,499]}(v_y > 8.97)$. Our findings reveal that the formulae inferred by our method exhibit a simpler structure compared

to those obtained by (Bombara and Belta, 2021). Additionally, our approach achieves superior classification performance than the algorithm presented in (Bombara and Belta, 2021), though it comes at the expense of increased execution time, again which is expected due to the utilization of boosting and conciseness techniques.

3.6 Summary

In this chapter, we introduced the BCDT approach designed for offline learning of two-class classifiers on time-series data. The BCDT method constructs an ensemble of decision trees, which are empowered by conciseness techniques to enhance the interpretability of output STL formulae. To assess the effectiveness of our algorithm, we conducted evaluations using two case studies, and compared its classification and interpretability benefits with two of existing algorithms in the literature.

Chapter 4

Time-Incremental Learning of Temporal Logic Classifiers

In this chapter, we broaden our attention beyond offline learning, as discussed in Chapter 3, and delve into the realm of time-incremental learning framework, and present an algorithm for inferring STL formulae within this framework.

4.1 Introduction

Real-time decision-making for robotic tasks poses a formidable challenge, as it often involves making predictions about potential outcomes incrementally, relying on partial signals available over time. The accuracy of these step-by-step predictions is crucial in efficiently preventing unwanted behaviors from occurring. To illustrate, consider the naval surveillance scenario, described in Sec. 3.5.1, from (Kong et al., 2016). In this scenario, a normally behaving vessel follows a direct course from the open sea towards the harbor. In contrast, an anomalously behaving vessel may exhibit two different patterns: either veering towards the island before heading to the harbor or approaching other vessels in the passage between the peninsula and the island before returning to the open sea (see Fig. 3·3).

Predicting vessel behavior labels in real-time is of interest for mitigating abnormal behaviors and reducing potential casualties. Assume the time horizon of the scenario is T, and a dataset $S = \{(s^i, \ell^i)\}_{i=1}^N$ is provided. The dataset consists of N signals s^i of duration T each, along with their corresponding labels ℓ^i. These signals represent

recorded vessel trajectories over time (e.g., $x(t)$ and $y(t)$), while the labels indicate normal or anomalous behaviors. In this chapter, we extend our studies in Chapter 3 from the offline learning framework, and introduce the *time-incremental learning* framework. In this setting, the objective is to develop a method that can classify (predict) the real-time behavior of a vessel at each time point t, using its prefix signal $s[0:t]$, where $1 \leq t \leq T$. To address the interpretability and readability concern of the final outputs, we use STL formulas (Maler and Nickovic, 2004) as classifiers.

A vast body of literature exists concerning the fusion of formal methods and machine learning (ML) to represent time-series data classifiers as temporal logic formulas (Bartocci et al., 2014; Jin et al., 2015; Kong et al., 2016; Bombara et al., 2016). However, none of these existing works focus on predicting the label of prefix signals. It should be noted that one possible approach to tackle this problem is by utilizing an offline supervised learning technique on the provided dataset. This involves learning an STL formula and creating a monitor based on that (Maler and Nickovic, 2004). Nevertheless, it is important to consider that the monitor's Boolean or quantitative output might not provide conclusive results for prefix signals with time lengths shorter than the horizon of the learned STL formula.

We introduce an innovative framework designed to predict the labels of prefix signals with various time lengths (Aasi et al., 2023). The framework comprises three key components. Firstly, we conduct signal analysis and employ a heuristic approach to identify specific time points, termed "decision times," which potentially hold valuable information for classifying prefix signals into two categories. Subsequently, for each decision time, we create a decision tree-based classifier (Breiman et al., 1984; Ripley, 2007) and describe it using an STL specification. Each STL formula captures the temporal characteristics of the prefix signals within time horizons shorter than the corresponding decision time. Lastly, we employ neural networks to assign a time-

variant weight distribution to the learned STL formulas. The weighted conjunction of these STL formulas is considered as a weighted STL (wSTL) formula (Mehdipour et al., 2020), which forms the basis for constructing our predictor. The predictor is then used to predict the label of prefix signals. We assess the efficiency and classification performance of our approach in comparison to two baselines by conducting two case studies, which involve naval surveillance case study from Sec. 3.5.1 and an autonomous driving scenario.

The primary contributions of our study in this chapter are as following:

1. We introduce the time-incremental learning problem, which involves predicting the labels of prefix signals received incrementally over time. Given a dataset of signals and labels, our goal is to develop a method for accurate predictions of the label of prefix signals.

2. We propose an interpretable classification solution to address the time-incremental learning problem, leveraging STL, decision trees, and neural networks in our approach.

3. To evaluate the effectiveness of our framework, we conduct case studies on autonomous driving and naval-surveillance scenarios, examining the classification and prediction performance of our approach compared to two baselines.

4.2 Related Works

Some of the existing machine learning algorithms, such as deep neural networks, long short-term memory models (Hochreiter and Schmidhuber, 1997), and transformers (Wolf et al., 2020), have been primarily dedicated to the classification of time-series data, which can be used in a time-incremental framework as well. However, these complex models focus mainly on accurate classification of the data, without giving

much attention to the interpretability of their output model.

There has been a significant interest in integrating formal methods with machine learning algorithms, to introduce interpretability into inference problems through expressing the classifiers of time-series data as temporal logic formulae (Mohammadinejad et al., 2020b; Ketenci and Gol, 2019; Bombara et al., 2016; Li et al., 2023; Linard et al., 2022).

Early efforts in this domain centered around discovering the optimal parameters for predetermined formula structures (Bartocci et al., 2015; Asarin et al., 2011; Hoxha et al., 2018; Jin et al., 2015). Recently, some supervised classification methods have tackled the challenge of learning both formula structures and their parameters. For instance, in (Kong et al., 2016) they employed a lattice search technique, in (Bombara et al., 2016) they utilized decision trees, and in our research discussed in chapter 3 we used boosted decision trees (Aasi et al., 2022). The other related approaches in this field are in different inference frameworks (Yan and Julius, 2021; Vazquez-Chanlatte et al., 2017; Bombara and Belta, 2017; Baharisangari et al., 2021; Yan et al., 2019; Mohammadinejad et al., 2020a; Linard and Tumova, 2020). However, none of these existing works focus on predicting the label of prefix signals in a time-incremental framework, using interpretable classifiers.

4.3 Time-Incremental Learning Problem

First, we provide some definitions that are used in the formulation of the time-incremental learning problem:

Definition 3 (Predictor) *The predictor $P_{\Phi,t}(s[0{:}t]) = \tilde{\ell}(t) \in C$ is a mapping that assigns label $\tilde{\ell}(t)$ to the prefix signal $s[0{:}t]$, which predicts whether the STL formula Φ is satisfied by the prefix signal $s[0{:}t]$ at time t. In more details: $P_{\Phi,t}(s[0{:}t]) = C_p$ if $s[0{:}t] \models \Phi$; otherwise, $P_{\Phi,t}(s[0{:}t]) = C_n$.*

Definition 4 (Timepoint MisClassification Rate (TMCR)) *Consider the predictor* $P_{\Phi,t}(s^i[0{:}t]) = \tilde{\ell}^i(t)$. *The Timepoint MisClassification Rate* $TMCR(P_{\Phi,t}, t)$ *of predictor* $P_{\Phi,t}$ *at time step* t *is defined as:*

$$\frac{1}{N}\left|\{s^i[0:t] \mid (\tilde{\ell}^i(t) = C_p \wedge \ell^i = C_n) \vee (\tilde{\ell}^i(t) = C_n \wedge \ell^i = C_p)\}\right| \tag{4.1}$$

In other words, TMCR accounts for the number of misclassified prefix signals with time horizon t*, divided by the total number of signals* N*.*

Definition 5 (Incremental MisClassification Rate (IMCR)) *We define the Incremental MisClassification Rate of predictor* $P_{\Phi,t}$*, denoted by* $IMCR(P_{\Phi,t})$*, as the vector of TMCR values over the time horizon* T*:*

$$IMCR(P_{\Phi,t}) = [TMCR(P_{\Phi,t}, 0),\, TMCR(P_{\Phi,t}, 1), ...,\, TMCR(P_{\Phi,t}, T)] \tag{4.2}$$

Now we have the necessary materials to formulate the main problem of this chapter as below:

Problem 2 (Time-Incremental Learning) *Given a labeled data set* $S = \{(s^i, \ell^i)\}_{i=1}^N$*, find an STL formula* Φ *and its corresponding predictor* $P_{\Phi,t}$*, such that* $IMCR(P_{\Phi,t})$ *defined in* (4.2) *is minimized.*

Remark 3 *It is clear that a predictor* $P_{\Phi,t}$ *minimizes the IMCR vector, if it minimizes the* $TMCR(P_{\Phi,t}, t)$*,* $\forall t \in [0, T]$*.*

4.4 Time-Incremental Inference Approach

Our approach to Pb. 2 is visualized in Fig. 4·1. It comprises three key components:

(i) *Signal Analysis*, as described in Section 4.4.1, applies a heuristic method to analyze the signals in the given dataset. The goal is to identify a finite number of potentially informative timepoints known as decision times and denoted by the set $\mathcal{T} = \{t_k\}_{k=1}^K$;

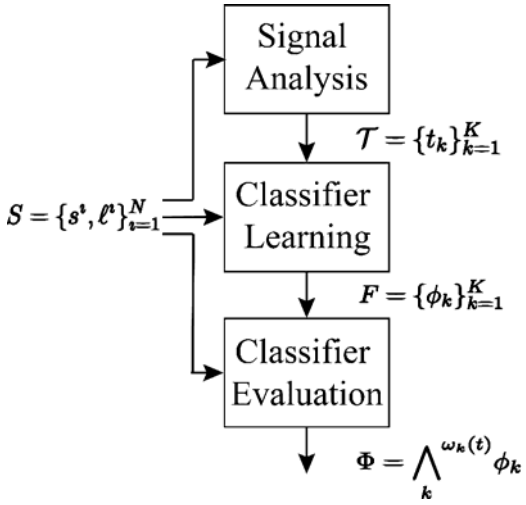


Figure 4·1: Schematic of our proposed framework for the time-incremental learning problem(Aasi et al., 2023)

(ii) *Classifier Learning*, explained in Section 4.4.2, utilizes a decision tree method to generate an STL formula for each decision time t_k in $\mathcal{T}$. This process involves learning both the structure and parameters of the STL formula. The collection of generated STL formulas is denoted by $F = \{\phi_k\}_{k=1}^{K}$;

(iii) *Classifier Evaluation*, detailed in Section 4.4.3, employs neural networks to assign a time-dependent weight distribution to the STL formulas in F. These weights represent the prediction performance of each STL formula over time. The final output of this stage is the weighted STL formula $\Phi = \bigwedge_k^{\omega_k(t)} \phi_k$ along with a predictor $P_{\Phi,t}$ based on that formula.

In the following, we explain each of these components in details.

4.4.1 Signal Analysis

The Signal Analysis component's objective is to examine the signals of the dataset S and identify specific timepoints called *decision times* that carry crucial information over time. These decision times, represented as t_k, are critical for classifying the

prefix dataset $S[0:t_k]$ into two distinct classes, making them suitable candidates for generating classifiers. To accomplish this, we present a metric that formulates an optimization problem for determining these decision times.

The *Euclidean distance* is frequently employed as the distance metric when dealing with multi-dimensional time-series data (Bombara and Belta, 2017). The Euclidean distance between two signals s^1 and s^2 is defined as:

$$d^2\left(s^1, s^2\right) = \sum_{j=1}^{n} \sum_{t=0}^{T} \left(s_j^1(t) - s_j^2(t)\right)^2 \tag{4.3}$$

Here, we extend the Euclidean distance (4.3) to define the *time-dependent Euclidean distance* as:

$$d^2\left(s^1, s^2, t\right) = \sum_{j=1}^{n} \left(s_j^1(t) - s_j^2(t)\right)^2, \; \forall\, t \in \{0, ..., T\} \tag{4.4}$$

In particular, the time-dependent Euclidean distance evaluates the Euclidean distance between two signals s^1 and s^2 at a specific time point t. Given a dataset S with N pairs of (signal, label), let us consider the set of signals with positive labels are indexed by $h \in \{1, ..., N_p\}$ and the set of signals with negative labels are indexed by $g \in \{1, ..., N_n\}$, such that $N_p + N_n = N$. Using the metric in (4.4), we define the *positive-negative distance* as:

$$d_{pn}^2(t) = \sum_{j=1}^{n} \sum_{h=1}^{N_p} \sum_{g=1}^{N_n} \left(s_j^h(t) - s_j^g(t)\right)^2, \; \forall t \in \{0, ..., T\} \tag{4.5}$$

In other words, the positive-negative distance calculates the sum of time-dependent Euclidean distances between each pair of signals with different labels.

We use the positive-negative distance as an optimization metric to assess the distinction between positive and negative labeled prefix signals. We compute the decision times as the timepoints at which the first- or second-order discrete derivatives

of d^2_{pn} in (4.5) become zero. The underlying idea is that at these specific timepoints, the signals with positive and negative labels are either maximally distant or closest to each other at a local level (first-order derivatives). Alternatively, they represent the time points when the positive-negative distance transitions most significantly over time (second-order derivatives). We also consider the horizon T of the signals as a decision time, so we can generate a classifier when we have the whole traces of the signals. The set of decision times is denoted by $\mathcal{T} = \{t_k\}_{k=1}^{K}$, where $\forall k \in \{1, ..., K\}$: $t_k \in \mathbb{Z}_{\geq 0}$, $1 \leq t_k \leq T$.

Remark 4 *Note that a trivial approach to Pb.2 is to create classifiers for each time-point throughout the signal horizon. However, this method is clearly inefficient and computationally costly. In Sec.4.5, we assess the efficiency and prediction accuracy of this trivial solution in comparison to our approach.*

Example 2 *Fig. 4·2 illustrates the positive-negative distance over time for the naval surveillance case study (see Sec. 3.5.1, and 4.5.1). The Signal Analysis part of our algorithm identifies 8 decision times denoted as* $\mathcal{T} = \{t_k\}_{k=1}^{8}$, *where* t_1, t_3, t_5 *and* t_7 *correspond to the time points when the first-order discrete derivatives equal zero,* t_2, t_4 *and* t_6 *represent the time points when the second-order discrete derivatives are zero, and* t_8 *denotes the horizon* T *of the signals.*

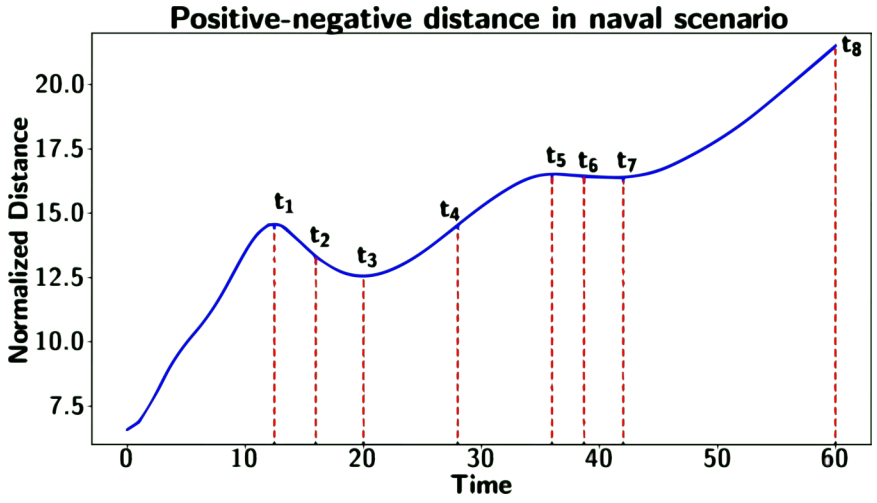


Figure 4·2: Representation of positive-negative distance over time, in the naval surveillance scenario

4.4.2 Classifier Learning

The subsequent module of our algorithm, Classifier Learning, receives two inputs: the set of decision times $\mathcal{T} = \{t_k\}_{k=1}^{K}$ obtained from the Signal Analysis part, and the dataset S. The primary objective of Classifier Learning is to generate classifiers for each decision time t_k, using the prefix dataset $S[0 : t_k]$. To include interpretability into our framework, we draw inspiration from (Bombara et al., 2016) and our CDT algorithm in Sec. 3.4.2, and employ a decision tree method $\mathcal{D}$ described in Alg. 5 to construct the classifiers. At each decision time t_k in the set $\mathcal{T}$, we employ Alg. 5 to grow a decision tree, denoted by Υ_k, alongside an associated STL formula, denoted by ϕ_k.

Algorithm 5 Decision Tree Construction Method $\mathcal{D}$

Meta-Parameters: $\mathcal{P}$, $\mathcal{J}$, *stop*
Input: $S[0 : t_k]$, ϕ^{path}, h
Output: sub-tree Υ_k
if $stop(\phi^{path}, h, S[0 : t_k])$ **then**
    $c^* = \text{argmax}_{c \in C}\{p(S[0 : t_k], c; \phi^{path})\}$
    return $leaf(c^*)$
$\phi^* = \text{argmax}_{\vartheta \in \mathcal{P}, \theta \in \Theta} \mathcal{J}(S[0 : t_k], partition(S[0 : t_k], \phi_\theta \wedge \phi^{path}))$
$\Upsilon_k \leftarrow non\_terminal(\phi^*)$
$S_\top[0 : t_k], S_\bot[0 : t_k] \leftarrow partition(S[0 : t_k], \phi^{path} \wedge \phi^*)$
$\Upsilon_k.left \leftarrow \mathcal{D}(S_\top[0 : t_k], \phi^{path} \wedge \phi^*, h + 1)$
$\Upsilon_k.right \leftarrow \mathcal{D}(S_\bot[0 : t_k], \phi^{path} \wedge \neg\phi^*, h + 1)$
return Υ_k

Similar to the CDT algorithm in Sec. 3.4.2 and following the same notations from Sec. 3.4.3, Alg. 5 involves three meta parameters:

1. PSTL primitives $\mathcal{P}$: These are simple PSTL formulae, termed primitives (Bombara et al., 2016), which serve as splitting rules at each node. For this purpose,

we employ first-order primitives described as:

$$\mathcal{P} : \{\Box_{[t_0,t_1]}(s_j \sim \pi), \Diamond_{[t_0,t_1]}(s_j \sim \pi)\}$$

2. Impurity measure $\mathcal{J}$: To select the best primitive at each node, we utilize the extended misclassification gain impurity measure proposed in (Bombara et al., 2016), detailed in (3.4).

3. Stopping conditions *stop*: We halt the growth of the decision trees when they reach a specified maximum depth.

Alg. 5 operates recursively and takes the following inputs: (1) the prefix dataset $S[0{:}t_k]$ at the current node, (2) the path formula ϕ^{path} from the root to the current node, and (3) the current depth level h. Initially, the algorithm checks the stopping conditions (line 4), and if they are met, it returns a single leaf node, labeled by $c^* \in C$ (lines 5-6). The function $p(S[0{:}t_k], c; \phi^{path})$ determines the classification quality based on the impurity measure $\mathcal{J}$.

If the stopping conditions are not fulfilled, it identifies an optimal STL formula, denoted as ϕ^*, among all possible valuations of the first-order primitives (line 7). This formula is then assigned to a new non-terminal node in the tree (line 8). Subsequently, the prefix dataset $S[0{:}t_k]$ is divided into satisfying and violating prefix datasets, $S_\top[0{:}t_k]$ and $S_\bot[0{:}t_k]$ respectively, according to the optimal formula from the root to the newly generated node, denoted by $\phi^{path} \wedge \phi^*$ (line 9). The tree construction continues recursively on the left and right subtrees of the new node (lines 10-11).

The output of the Classifier Learning module comprises a set of STL formulas, denoted by $F = \{\phi_k\}_{k=1}^K$, where each decision tree Υ_k is translated into a corresponding STL formula ϕ_k using the translation approach described in (Bombara et al., 2016).

Note that for all formulas ϕ_k in set F, it holds that $hrz(\phi_k) \leq t_k$, since each tree Υ_k is generated using the prefix dataset $S[0:t_k]$.

Example 3 *Following up on the Ex. 2 and for the naval surveillance scenario, the output of the Signal Analysis part is* $\mathcal{T} = \{t_k\}_{k=1}^{8}$. *The Classifier Learning module generates a classifier for each decision time* t_k, *using the prefix dataset* $S[0:t_k]$, *and the output of this module would be denoted by* $F = \{\phi_k\}_{k=1}^{8}$. *For example, with a 3-fold cross validation and a maximum depth of 2 for the trees, the STL formulae learned for the third and fourth decision times are:*

(i) $t_3 = 20 : \phi_3 = (\phi_{31} \wedge \phi_{32}) \vee (\neg\, \phi_{31} \wedge \phi_{33})$, *where:*

$$\phi_{31} = \square_{[11,16]}(y > 23.33), \quad \phi_{32} = \Diamond_{[15,18]}(y \leq 33.83), \quad \phi_{33} = \square_{[4,13]}(x > 42.69)$$

(ii) $t_4 = 28 : \phi_4 = (\phi_{41} \wedge \phi_{42}) \vee (\neg\, \phi_{41} \wedge \phi_{43})$, *where:*

$$\phi_{41} = \square_{[7,24]}(y > 20.88), \quad \phi_{42} = \Diamond_{[7,27]}(x \leq 44.71), \quad \phi_{43} = \square_{[7,27]}(x > 29.02)$$

Note that $hrz(\phi_3) = 18 \leq t_3$ *and* $hrz(\phi_4) = 27 \leq t_4$. *The trajectories of the vessels over time and the learned formulas are shown in Fig. 4·3.*

4.4.3 Classifier Evaluation

Our framework's final component, Classifier Evaluation, evaluates the given dataset S along with the set of generated STL formulas $F = \{\phi_k\}_{k=1}^{K}$. The aim of Classifier Evaluation is to assign time-variant weight distributions to the formulas, denoted by $\omega(t) = \{\omega_k(t)\}_{k=1}^{K}$. These weights adjust the classification performance of the formulas when applied to prefix signals with different time lengths. For convenience, we represent the weights at time t as a column vector denoted by $\omega(t)$, which has dimensions of $K \times 1$. To determine the weights $\omega(t)$ over time, we present a method in Alg. 6, where our objective is to find the $K \times T$ dimensional matrix $W = [\omega(0), , \omega(1), ..., , \omega(T)]$, encompassing the weight vectors for all time points within the signal horizon T.

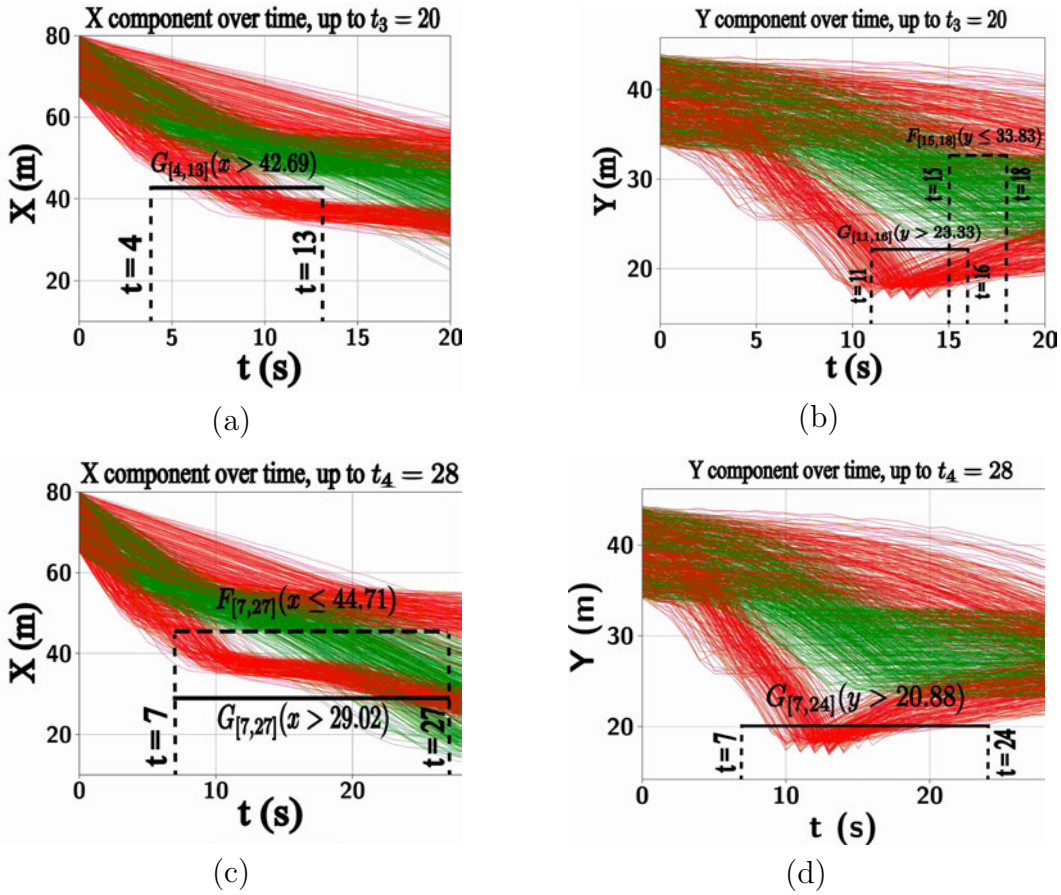


Figure 4·3: x and y components of naval surveillance trajectories over time, up to (a), (b) decision time $t_3 = 20$, (c), (d) decision time $t_4 = 28$, respectively. The learned formulas ϕ_3 and ϕ_4 are illustrated, where the thresholds of the always and eventually operators are shown by solid and dashed lines, respectively (Aasi et al., 2023).

First, Let us introduce some notations. For the given set of formulas $F = \{\phi_k\}_{k=1}^{K}$ and every time step t, we use $F_t^{\leq}$ to represent the subset of formulas in F that have a horizon less than or equal to t, and we use $F_t^{>}$ to represent the remaining formulas with a higher horizon. It should be clear that $F_t^{\leq} \cap F_t^{>} = \varnothing$ and $F_t^{\leq} \cup F_t^{>} = F$. The reason behind this partitioning is that given a specific time step t and the prefix signal $s^i[0{:}t]$, the formulas in $F_t^{\leq}$ can predict a label for $s^i[0{:}t]$ based on their satisfaction or violation concerning the prefix signal. However, the set of formulas in $F_t^{>}$ may not be

able to conclusively predict a label, as they require more instances of the prefix signal over time. Note that at $t = 0 : F_0^{\leq} = \varnothing, F_0^{>} = F$, and at $t = T : F_T^{\leq} = F, F_T^{>} = \varnothing$.

Algorithm 6 Learning the weights of the STL classifiers

1: **Input:** $S = \{(s^i, \ell^i)\}_{i=1}^N$, $F = \{\phi_k\}_{k=1}^K$
2: **Output:** matrix of the weights Ω
3: **Initialize:** $F_0^{\leq} \leftarrow \varnothing$, $F_0^{>} \leftarrow F$, $\omega(0) \leftarrow \emptyset$
4: **For** $t = 1, ..., T$:
5: $\quad F_t^{\leq}, F_t^{>} \leftarrow partition\_formulas(F, t)$
6: $\quad$ **if** $F_t^{\leq} = F_{t-1}^{\leq}$ **then**
7: $\quad\quad \omega(t) = \omega(t-1)$
8: $\quad$ **Otherwise**
9: $\quad\quad S[0:t] = \{s^i[0:t], \ell^i\}_{i=1}^N$
10: $\quad\quad \mathrm{R}_t^{\leq} \leftarrow compute\_robustness(F_t^{\leq}, S[0:t])$
11: $\quad\quad \omega^{\leq}(t) \leftarrow learn\_weights(R_t^{\leq}, \{\ell^i\}_{i=1}^N)$
12: $\quad\quad \omega^{>}(t) = \emptyset$
13: $\quad\quad \omega(t) = [\omega^{\leq}(t), \omega^{>}(t)]^{\top}$
14: **return** $W = [\omega(0), \omega(1), ..., \omega(T)]$

Alg. 6 receives the dataset $S = \{(s^i, \ell^i)\}_{i=1}^N$ and the set of STL formulas $F = \{\phi_k\}_{k=1}^K$ learned by the Classifier Learning module as input. The subsets $F_0^{\leq}$ and $F_0^{>}$, along with the weight vector $\omega(0)$, are initialized as $\varnothing$, F, and the zero vector $\emptyset$, respectively (line 3). At each time step t within the horizon T of signals (line 4), the subsets of formulas $F_t^{\leq}$ and $F_t^{>}$ are computed using the $partition\_formulas$ function (line 5). This function compares the horizons of the formulas in F with the current time step t and divides them into $F_t^{\leq}$ and $F_t^{>}$. If there are no updates in the subset $F_t^{\leq}$ compared to the previous time step (line 6), the same weight vector from the previous time step is retained for the current time (line 7).

However, if the set $F_t^{\leq}$ is updated (line 8), we first construct the prefix dataset $S[0:t]$ (line 9). Subsequently, the robustness of the prefix signals in $S[0{:}t]$ with

respect to the formulas in $F_t^{\leq}$ is computed using the *compute_robustness* function, and the results are stored in the robustness matrix $R_t^{\leq}$ (line 10). The dimensions of $R_t^{\leq}$ are $N \times |F_t^{\leq}|$, with the i^{th} row representing the robustness of prefix signal $s^i[0{:}t]$ with respect to the formulas in $F_t^{\leq}$. The robustness matrix $R_t^{\leq}$ and the labels of the signals $\{\ell^i\}_{i=1}^N$ are used to learn the weights of the formulas in $F_t^{\leq}$ at time t, denoted by the weight vector $\omega^{\leq}(t)$ (line 11). The function *learn_weights* creates a neural network aimed at learning the weights of the formulas in $F_t^{\leq}$. Prior research in (Yan and Julius, 2021) has already investigated the use of neural networks for learning the weights of wSTL formulas. Taking inspiration from (Cuturi and Blondel, 2017), our custom-designed neural network employs a differentiable loss function to quantify the disparity between the predicted label of the prefix signal $s^i[0:t]$ and its actual label ℓ^i from the dataset S. Since the formulas in $F_t^{>}$ are not capable of predicting a label for prefix dataset $S[0:t]$, their weights $\omega^{>}(t)$ are set to zero (line 12). Finally, the column vector $\omega(t)$ is constructed as the weight vector at time t (line 13), and gets appended to the weight matrix W (line 14).

The result of the Classifier Evaluation module is considered as the weighted conjunction of the STL formulas in F, represented by the wSTL formula $\Phi = \bigwedge_k {}^{\omega_k(t)}\phi_k$. Using formula Φ, we construct the predictor $P_{\Phi,t}$ to predict the label of a prefix signal $s^i[0:t]$ as below:

$$P_{\Phi,t}(s^i[0:t]) = \tilde{\ell}^i(t) = \text{sign}\,(\sum_{k=1}^{K} \omega_k(t)\,.\,\rho(\phi_k,\, s^i[0:t])) \tag{4.6}$$

Note that the predictor we propose calculates the robustness of the prefix signal differently from the methods presented in (Mehdipour et al., 2020; Yan and Julius, 2021), which involve monitoring the wSTL formula Φ and computing its robustness. Our approach involves computing the robustness of each STL formula in Φ and then taking a weighted sum of these individual robustness values. The weights used in this

computation are time-dependent and are adjusted based on the classification performance of the STL formulas and the time length of the prefix signals. In Sec. 4.5, we highlight the significance of considering the weight distributions of the STL formulas by demonstrating the predictor's performance with and without taking these weight distributions into account.

Example 4 *We illustrate how our Classifier Evaluation part operates by examining the examples Ex. 2 and Ex. 3, from the naval surveillance case study. Specifically, we focus on the time window* $t \in [20, 28]$*, which represents the period between the third decision time* $t_3 = 20$ *and the fourth decision time* $t_4 = 28$ *as mentioned in Ex. 3. The same approach is applicable to all other timepoints along the signals' horizon.*

Using the Classifier Evaluation module, at time $t = 20$ *we have two sets of formulas (classifiers) to be considered:* $F^{\leq}_{20}$ *and* $F^{>}_{20}$*. The former set, denoted by* $F^{\leq}_{20} = \phi_1, \phi_2, \phi_3$*, consists of formulas with a horizon less than or equal to 20. On the other hand,* $F^{>}_{20}$ *comprises formulas with a longer horizon and is represented as* $F^{>}_{20} = \phi_4, ...\phi_8$*. To predict the label* $\hat{\ell}^i(20)$ *of the prefix signal* $s^i[0:20]$*, the weighted sum of its robustness value with respect to the formulas in* $F^{\leq}_{20}$ *is utilized. A neural network is constructed to determine the optimal weights* $\omega^{\leq}(20) = [\omega_1(20), \omega_2(20), \omega_3(20)]$ *for the formulas in* $F^{\leq}_{20}$ *(see Fig. 4·4.(a)). The objective is to minimize the misclassification rate of the prefix signals in prefix dataset* $S[0:20]$*. Since the formulas in* $F^{>}_{20}$ *might not be conclusive in predicting the label of a prefix signal* $s^i[0:20]$*, their weight vector* $\omega^{>}(20) = [\omega_4(20), ..., \omega_8(20)]$ *is set to zero. Ultimately, for the weight vector at* $t = 20$ *we have* $\omega(20) = [\omega^{\leq}(20),\, \omega^{>}(20)]^{\top}$*.*

Between timepoints $t = 21$ *and* $t = 27$*, there are no updates in the set* $F^{\leq}_t$*. In other words,* $F^{\leq}_t$ *remains unchanged compared to* $F^{\leq}_{t-1}$*, and the robustness of the prefix signals with respect to the formulas in* $F^{\leq}_t$ *is the same as* $t = 20$*. Thus, for all* t *from 21 to 27, the weight vector* $\omega(t)$ *is the same as* $\omega(20)$*.*

When $t = 28$*, the formula* ϕ_4 *becomes part of the set* $F^{\leq}_t$*. As a result,* $F^{\leq}_t$ *differs from* $F^{\leq}_{t-1}$*, prompting the construction of a new neural network to calculate the weights* $\omega^{\leq}(t) = [\omega^{\leq}_1(t), ..., \omega^{\leq}_4(t)]$ *based on the prefix signals* $s^i[0:28]$ *(see Fig. 4·4.(b)). Meanwhile,* $\omega^{>}(t) = [\omega_5(t), ..., \omega_8(t)]$ *is set to zero. Eventually, for the weight vector at* $t = 28$ *we have* $\omega(28) = [\omega^{\leq}(28),\, \omega^{>}(28)]^{\top}$*.*

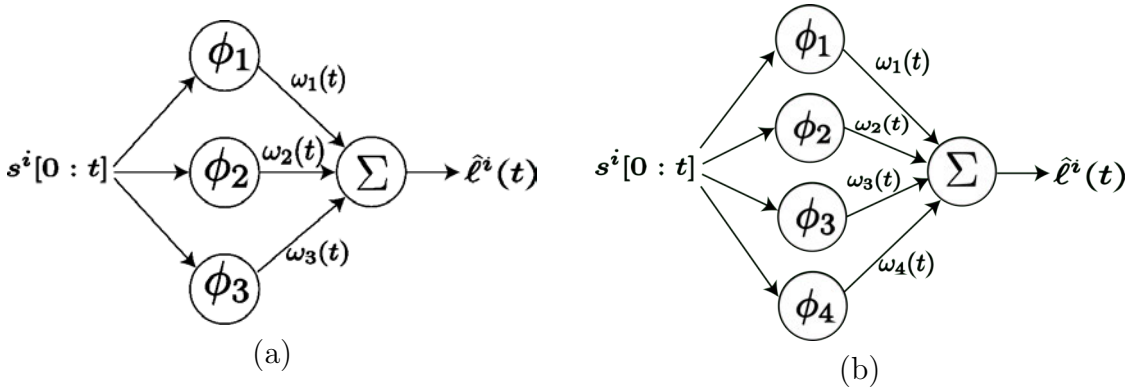


Figure 4·4: Neural networks constructed by the Classifier Evaluation part of our framework for the naval surveillance scenario, at decision times (a) $t_3 = 20$ and (b) $t_4 = 28$.

4.5 Case Studies

Our approach's efficacy and classification performance are showcased in two case studies: the naval surveillance scenario which was introduced originally in Sec. 3.5.1 and here will be followed up in Sec. 4.5.1, and an autonomous driving scenario, detailed in Sec. 4.5.2 and implemented in the simulator CARLA (Dosovitskiy et al., 2017). To assess the effectiveness of our framework, we compare its performance with two baseline methods:

(i) **Uniform-weights:** The primary objective of the uniform-weights baseline is to highlight the significance of incorporating time-variant weight distributions in the predictor function. In this baseline approach, we adhere to the Signal Analysis and Classifier Learning aspects of our framework precisely. If we use subscript $(\cdot)_u$ to denote the properties of the uniform-weights baseline, for its set of decision times and set of classifiers we have $\mathcal{T}_u = \mathcal{T} = \{t_k\}_{k=1}^K$ and $F_u = F = \{\phi_k\}_{k=1}^K$, respectively. However, in this baseline instead of assigning time-varying weights to the formulas, we assign a uniform weight distribution to them and we omit the Classifier Evaluation stage of our framework. Therefore,

for all formula indices $k \in \{1, ..., K\}$ and all time points $t \in [1, T]$, we have: $\omega_{k,u}(t) = \mathbb{I}$. The final output of the uniform-weights baseline is then represented as $\Phi_u = \bigwedge_{k=1}^{K} \phi_k$.

(ii) **All-times:** The main role of the all-times baseline is to demonstrate the importance of selecting a finite number of decision times. However, instead of opting for a finite set of decision times, we construct classifiers at every time-point within the horizon T of signals. If we use subscript $()_a$ to denote the properties of the all-times baseline, its set of decision times can be represented by $\mathcal{T}_a = \{1, 2, ..., T\}$, and the Signal Analysis part of our framework is skipped. The rest of the all-times baseline is similar to our approach: at each decision time $t_{k,a} \in \mathcal{T}_a$ and for each prefix dataset $S[0 : t_{k,a}]$, the Classifier Learning and Classifier Evaluation components are employed to acquire an STL formula $\phi_{k,a}$ and weight distribution $\omega_{k,a}(t)$. The final outcome of the all-times baseline is denoted as $\Phi_a = \bigwedge_{k=1}^{T} {}^{\omega_{k,a}(t)} \phi_{k,a}$.

Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995) is employed to address the optimization problems presented in Classifier Learning part and Alg.5, while the PyTorch (Paszke et al., 2019) library is utilized for training the neural networks in Classifier Evaluation part and Alg. 6. Parameters of both PSO and neural network methods are tuned empirically. Our approach and the two baselines share the same hyperparameter initialization. For both case studies, the evaluation of our framework and the baselines involves decision trees with a maximum depth of 2 and adopts 3-fold cross-validation. Execution times are measured using the system's clock. All computations are conducted in Python 2, running on an Ubuntu 18.04 system, with an Intel Core i7 processor clocked at 3.70 GHz and 16 GB RAM.

4.5.1 Naval Surveillance

Consider the naval surveillance scenario from (Kong et al., 2016), which we elaborated on previously in Sec. 3.5.1. The dataset for this case study comprises 2000 signals, consisting of 1000 normal and 1000 anomalous trajectories. These signals represent 2-dimensional trajectories with planar coordinates $(x(t), y(t))$ and span 61 timepoints. The labels are used to indicate the vessel's behavior type, whether it is normal or anomalous.

By applying our framework, the Signal Analysis component identifies 8 decision times (refer to Fig. 4·2), represented by $\mathcal{T}_{Naval} = \{t_k\}_{k=1}^{8} = \{12, 16, 20, 28, 36, 38, 41, 60\}$. Examples of the formulas acquired through the Classifier Learning process are shown in Ex. 3. The ultimate wSTL formula learned by our framework is denoted as $\Phi_{Naval} = \bigwedge_{k=1}^{8} {}^{\omega_k(t)}\phi_k$, where $\omega_k(t)$ signifies the time-dependent weight distribution obtained through the Classifier Evaluation stage. The total execution time of our method amounts to $739\,s$.

The uniform-weights baseline shares the same structure and formulas as our framework. However, in this baseline, the formulas have a uniform weight distribution: $\Phi_{Naval,u} = \bigwedge_{k=1}^{8} \phi_k$. The total runtime of the uniform-weights baseline is $727\,s$. The difference in runtime between this baseline and our approach can be attributed to the execution of the Classifier Evaluation part for our method, which takes approximately $12\,s$. Regarding the all-times baseline, the decision times encompass the entire signal horizon: $\mathcal{T}_a = \{1, 2, ..., 60\}$. For each decision time, we learn an STL formula and then assign a weight distribution, resulting in the final formula $\Phi_{Naval,a} = \bigwedge_{k=1}^{60} {}^{\omega_{k,a}(t)}\phi_{k,a}$. The runtime of the all-times baseline is $9747\,s$, significantly larger than our framework. This difference arises because the all-times baseline involves learning 60 STL formulas and computing their corresponding weight distributions over time.

Fig. 4·5 illustrates the IMCR performance of our framework in comparison to

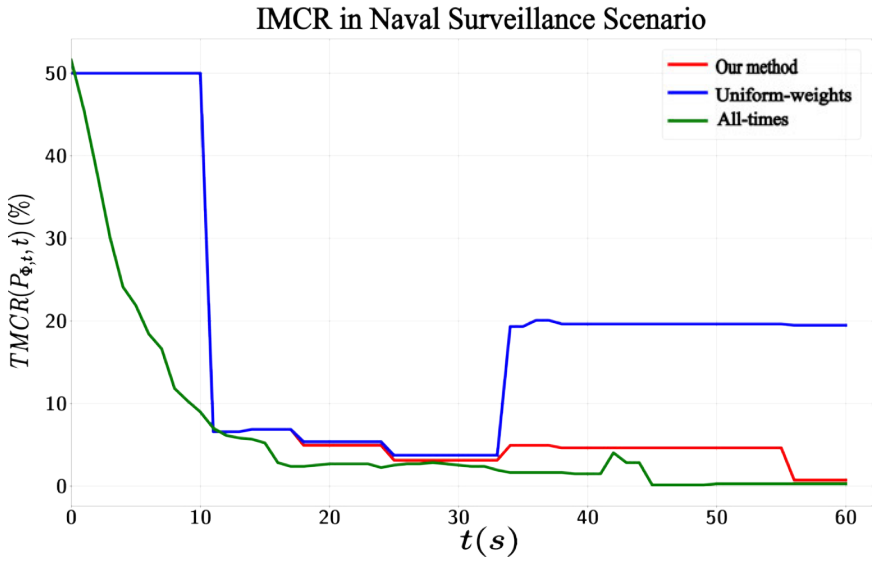


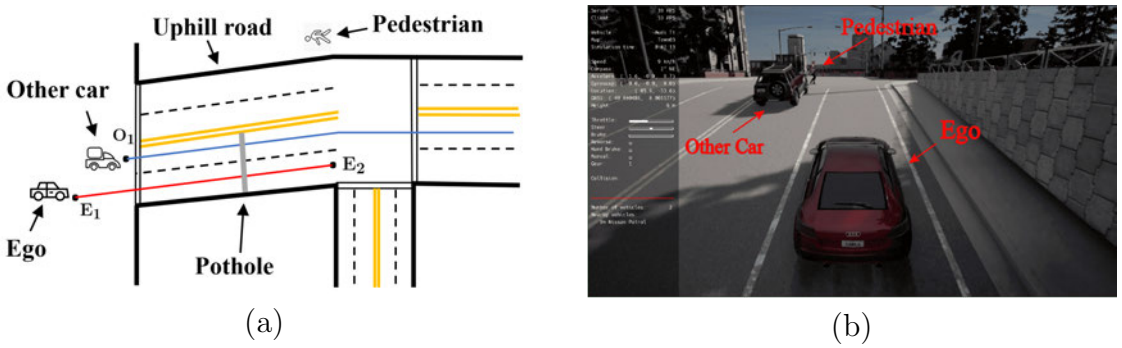
Figure 4·5: IMCR comparison of our framework with the baseline methods, in the naval surveillance scenario.

the two baselines. Our approach outperforms the uniform-weights baseline at all timepoints, demonstrating the importance of incorporating time-dependent weights to adjust the classification power over time. The all-times baseline exhibits better IMCR performance compared to our approach. However, this comes at the expense of significantly longer runtime ($9747\,s$) and higher memory consumption, as it involves learning 60 STL formulas and computing their weight distributions. In contrast, our method achieves comparable results with a much shorter runtime of $739\,s$, utilizing only 8 STL formulas and their corresponding weight distributions. These findings align with our expectations, indicating that the all-times baseline is generally better in terms of IMCR but at the cost of considerable execution time and memory usage.

4.5.2 Autonomous Driving

Consider the autonomous driving case study depicted in Fig. 4·6. The scenario comprises three elements: an autonomous vehicle referred to as ego, a pedestrian, and a car operated by a human driver. Both ego and the other car occupy separate, adjacent

lanes, proceeding in the same direction on an incline, heading towards an intersection without a traffic light. The vehicles maintain constant accelerations, though ego's acceleration is lower than that of the other car. The initial positioning ensures that the other car is consistently ahead of ego. Additionally, there is an unmarked cross-walk at the end of the road and a pothole situated in the middle of the ramp-shaped path.



(a) (b)

Figure 4·6: (a) Schematic of the autonomous driving scenario. The initial points of ego and other car are denoted by E_1 and O_1 and the goal point for ego is denoted by E_2 (Aasi et al., 2023); (b) Snapshot of the autonomous driving scenario in simulator CARLA (Dosovitskiy et al., 2017).

In this scenario, we encounter two distinct behavior types for the other car: aggressive and safe. An aggressive driver maintains constant acceleration within their lane, while a safe driver applies slight braking when encountering a pothole and full braking if a pedestrian crosses the street at an intersection. Accurately predicting the behavior label of the human driver in the other car becomes particularly valuable when ego lacks a clear line-of-sight to the pedestrian. To address this, we utilize the CARLA simulator (Dosovitskiy et al., 2017) to create this scenario. The simulation concludes once ego approaches its destination point within a 2-meter distance.

It is assumed that ego has the capability to estimate the relative position and velocity of the other car. The dataset of this case study comprises 300 signals, each containing 477 uniform time-samples per trace. Of these, 150 signals represent an

aggressive driver, while the remaining 150 signify a safe driver. The signals are presented in a 4-dimensional format, encompassing the relative position and velocity of the other car along the y and z axes.

The Signal Analysis component of our framework identifies 11 decision times in the dataset of this scenario, denoted by:

$$\mathcal{T}_{Urban} = \{t_k\}_{k=1}^{11} = \{97, 131, 166, 186, 240, 289, 334, 394, 420, 440, 476\}$$

The positive-negative distance metric for this case study, together with the decision times, are shown in Fig. 4·7.

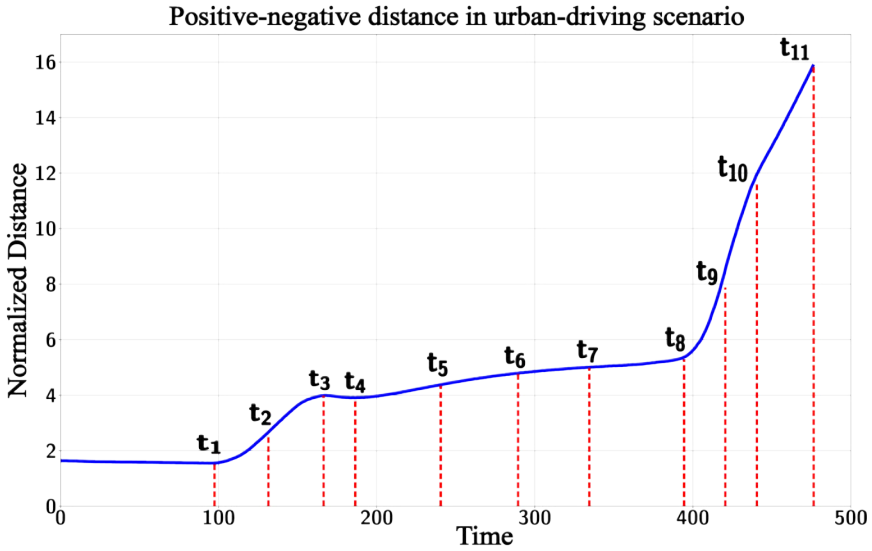


Figure 4·7: Representation of positive-negative distance over time, in the autonomous driving scenario.

An illustrative example of the formula acquired by the Classifier Learning module of our approach for decision time $t_2 = 131$ is as follows:

$$\phi_2 = (\phi_{21} \wedge \phi_{22}) \vee (\neg\phi_{21} \wedge \phi_{23})$$

$$\phi_{21} = \Diamond_{[122,130]}(v_y \leq 1.11), \quad \phi_{22} = \Diamond_{[108,124]}(y \leq 12.58), \quad \phi_{23} = \Diamond_{[110,121]}(y \leq 6.81)$$

To elaborate, ϕ_{21} indicates that the relative velocity of the other car in the y-axis is

less than or equal to 1.11 m/s at some timepoint between 122 to 130. Importantly, it should be noted that $hrz(\phi_2) = 130 \leq t_2$. The final wSTL formula obtained through our approach is denoted as $\Phi_{Urban} = \bigwedge_{k=1}^{11} {}^{\omega_k(t)}\phi_k$, where the weight distribution $\omega_k(t)$ is determined by the Classifier Evaluation component. The entire runtime of our framework for this case study amounts to 812 s.

The uniform-weights baseline employs identical formulas as our approach, with a uniform weight distribution: $\Phi_{Naval,u} = \bigwedge_{k=1}^{11} \phi_k$. Its total runtime amounts to 805 s. In contrast, the all-times baseline adopts the set of decision times $\mathcal{T}_a = \{1, 2, ..., 477\}$. Consequently, its final wSTL formula becomes $\Phi_{Naval,a} = \bigwedge_{k=1}^{477} {}^{\omega_{k,a}(t)}\phi_{k,a}$. However, the all-time baseline exhibits a significantly longer runtime of 33158 s, as it necessitates learning 477 STL formulas and their respective weight distributions over time. This exponential increase in runtime sets it apart from our framework.

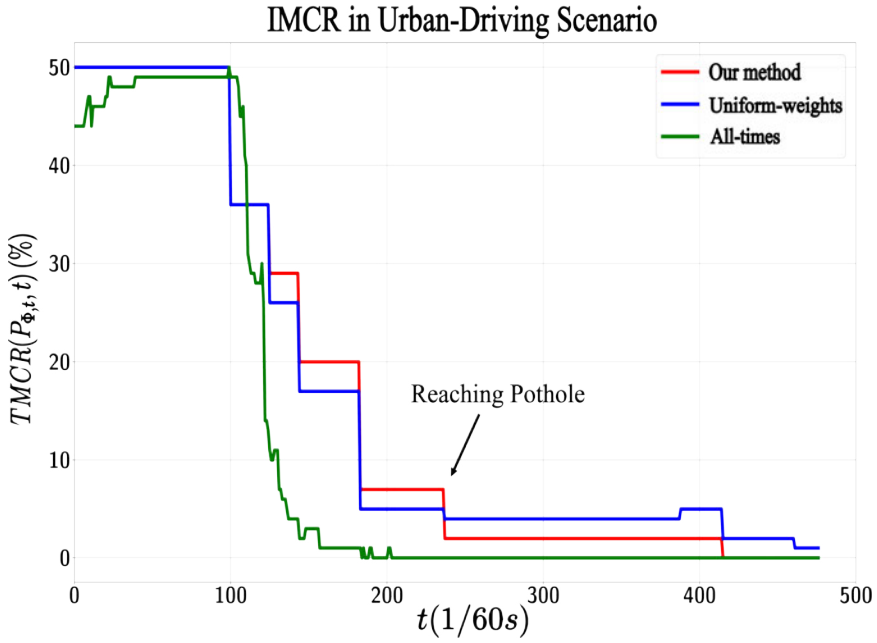


Figure 4·8: IMCR comparison of our framework with the baseline methods, in the autonomous driving scenario.

Fig. 4·8 illustrates the IMCR evaluation of our method compared to the baseline approaches. Our approach achieves superior classification accuracy compared to the

uniform-weights as we proceed in time and more parts of the signals become available. However, the all-times baseline generally outperforms our framework in terms of IMCR. This improvement comes at the expense of significantly longer execution time ($33,158\,s$) and higher memory consumption due to the need to learn 477 STL formulas and their weight distributions. Consequently, this renders it inefficient for handling large datasets with extended time horizons.

4.6 Summary

In this chapter, the time-incremental learning framework was presented. This framework addresses the prediction of labels for prefix signals over time, utilizing a labeled dataset. The approach combines decision trees and neural networks to create a predictor using STL formulae for effective classification of the prefix signals. To assess its performance, we conducted two case studies and compared the results against two baseline methods.

Chapter 5

Autonomous Vehicle Control in Deterministic Environments

In this chapter, we redirect our focus from inference problems, which were addressed in Chapter 3 and 4, to explore temporal logic-based control methods for dynamic systems, particularly for autonomous vehicles. Within this framework, we introduce an algorithm that enables the safe navigation of an autonomous vehicle along a given reference path. The algorithm ensures compliance with traffic rules and the fulfillment of runtime performance criteria.

5.1 Introduction

In the past few years, significant progress in computational software and electronic devices has spurred the creation of numerous automated safety functionalities in modern vehicles. Examples of such features include Adaptive Cruise Control (Vahidi and Eskandarian, 2003) and lane-keeping systems (Liu et al., 2007). Nevertheless, the constantly evolving urban driving environments and the need to adhere to a wide array of traffic regulations impose constraints on the effectiveness of these safety systems, especially in handling complex driving scenarios (see Fig. 5·1).

Various control and planning algorithms have been suggested to address this challenge, some of which rely on Model Predictive Control (MPC) algorithm(Camacho and Alba, 2013). In MPC, the autonomous vehicle's (referred to as ego) model, along with models of traffic and the environment at the current state, are utilized to foresee

Figure 5·1: Snapshot of an urban driving environment in the simulator CARLA (Dosovitskiy et al., 2017). Ego (black car) is subject to various traffic rules, including collision avoidance and stopping before red traffic light.

the future behavior of traffic participants within a finite horizon set by the MPC. This predictive capability, combined with the ability to systematically manage constraints, has made MPC a favored control approach for self-driving cars (Li et al., 2019; Anderson et al., 2010; Faulwasser et al., 2009; Tika et al., 2023). However, the majority of MPC-based approaches are based on Nonlinear MPC (NMPC) formulations and they suffer from high computational costs. The complexity of models (including ego's dynamics, traffic, environment, and uncertainty) leads to significant runtime expansion, rendering these methods impractical for handling real traffic scenarios.

Recently, there has been a growing trend in addressing traffic complexities that involve human drivers (Sadigh et al., 2016), and compliance with rules of the road (Vasile et al., 2017). For the later, temporal logics (Baier and Katoen, 2008) have been used in some of the existing works to formalize the traffic rules. Previous studies have explored MPC problems incorporating different types of temporal logic specifications. However, despite relatively fast solvability of these problems, existing

approaches often fail to meet real-time performance requirements.

In this chapter, we consider the problem of controlling ego, along a given reference path, while satisfying traffic rules such as collision avoidance and speed limit. The high-level goal of ego is to closely follow the reference path, aiming to minimize the energy usage while maximizing passenger comfort. Inspired by the readability of STL formulas, we describe the traffic rules using STL specifications. In order to reduce the computational load related to controlling systems with complex dynamics, models, and constraints, we employ runtime monitoring (Bartocci et al., 2018), (Deshmukh et al., 2017), which is a branch of formal methods that offers accurate insights into the runtime behavior of the systems.

Within this framework, we assume that ego has access to deterministic sensor measurements obtained from the state of other traffic participants on the road. Moreover, our primary emphasis lies in fulfilling the runtime performance requirements for real-world applications. It is essential to recognize that the issue we are addressing remains challenging, even in the deterministic setting, owing to the computational complexities arising from the extensive number of constraints. To tackle this, we introduce a two-level control scheme (Aasi et al., 2021) (see Fig. 5·3). At the high level of our approach, we solve the control problem using simple models, while at the low level, we validate the solutions against more complex models and modify them if needed.

At the high level of our framework, we develop an MPC problem using a simplified bicycle model and environment representation. The framework utilizes a quadratic objective function to satisfy the high-level objectives of ego. STL specifications are translated into Mixed-Integer Linear Constraints (MILCs) and integrated into the MPC's optimization problem, resulting in a Mixed-Integer Quadratic Programming (MIQP) problem. The optimal controller obtained from this process is then employed in the low level controller to determine the best control inputs for adhering to traffic

rules in a more detailed context, by applying runtime monitoring techniques to ego's trajectories in real-time.

Our proposed method is evaluated through computational experiments conducted in the urban traffic simulator CARLA (Dosovitskiy et al., 2017). The results demonstrate significant improvements in terms of runtime and solution quality, compared to existing NMPC-based approaches.

5.2 Related Works

Due to the ability of effectively managing constraints and accommodating extensive operating regions, MPC (Camacho and Alba, 2013) has emerged as a highly successful method for controlling autonomous systems (Falcone et al., 2007; Faulwasser et al., 2009; Erlien et al., 2015; Weiskircher et al., 2017; Li et al., 2019; Wu et al., 2022; Tika et al., 2023). Some of the early works in this domain focus on collision avoidance for semi-autonomous vehicles using single control inputs, such as active front steering (Falcone et al., 2007; Keviczky et al., 2006). In (Faulwasser et al., 2009), they propose an MPC approach for constrained nonlinear systems in a corridor path-following problem. In (Erlien et al., 2015), they present an MPC problem that focuses on ensuring safe driving while minimizing the need for human driver control intervention. In (Weiskircher et al., 2017), the authors address the issue of ensuring the safety of the ego vehicle in public traffic, by incorporating constraints related to road conditions and the behavior of other vehicles into a nonlinear MPC problem.

A dynamic trajectory planning and tracking controller for ensuring collision avoidance is designed in (Li et al., 2019), based on NMPC formulation. In (Wu et al., 2022), they propose a dynamic NMPC approach for trajectory tracking and collision avoidance requirements, which predicts the state of other traffic participants within the predictive step length of their MPC method. A path following MPC algorithm for mo-

bile robots introduced in (Tika et al., 2023), aimed to pursue a parametrized geometric path at its highest attainable speed. The authors employ a Bayesian optimization-driven method to automatically determine the optimal parameters that maximize the speed while minimizing the tracking error.

Most of the mentioned approaches either neglect the traffic rules when dealing with urban-driving situations, or they are computationally expensive. As the complexity of the models grows, their runtime increases significantly, making them unsuitable for real traffic scenarios. Therefore, the majority of these methods are unable to effectively handle actual traffic scenarios.

Recently, there has been an increasing inclination to take into account the traffic complexities, including the human driver behaviors (Sadigh et al., 2016; Schwarting et al., 2019), as well as traffic rules (Tumova et al., 2013; Vasile et al., 2017; Karlsson et al., 2018; Censi et al., 2019). To formulate these traffic rules in a concise mathematical manner, temporal logics (Baier and Katoen, 2008) provide us powerful tools, such as Linear Temporal Logic (LTL) (Clarke, 1997) and STL formulas (Maler and Nickovic, 2004). In (Ding et al., 2014; Wongpiromsarn et al., 2012), they explore MPC problems subject to LTL specifications, while in the works by (Raman et al., 2014; Sadraddini and Belta, 2015), they propose methods for incorporating STL formulas into MPC frameworks. However, despite relatively fast solvability of such problems, the current approaches generally fail to meet the real-time performance requirements.

5.3 Deterministic Control Problem

In this section, first we present the mathematical definition of the components required for formulating the control problem. The dynamics of the vehicles in the environment are represented using discrete-time continuous-space models. It is assumed that the discretization intervals, denoted by Δt, are uniform, meaning each

time step t corresponds to an integer multiple of Δt, e.g., $t = k\,\Delta t, k \in \mathbb{Z}_{\geq 0}$. Ego is required to adhere to a predefined reference path $\mathcal{R}$, starting from an initial point and reaching a goal point. The MPC's finite horizon consists of H time steps, and the total finite number of time steps of ego's operation is denoted by T, which is unknown.

5.3.1 Dynamic Model of Ego

We consider the dynamic model of ego by the general form:

$$\zeta_{t+1} = f(\zeta_t, u_t), \tag{5.1}$$

where $\zeta_t = [x_t, y_t, \psi_t, v_t]^\top \in \mathbb{R}^4$ is the state vector of ego, and $u_t = [\delta_t, \gamma_t]^\top \in \mathbb{R}^2$ is its control input, at time t. The state vector of ego consists of ego's position (x_t, y_t) in the absolute coordinate frame of the environment, its heading (yaw angle) ψ_t, and its velocity magnitude $v_t = ||(\dot{x}_t, \dot{y}_t)||$. The control input of ego consists of its steering angle $\delta_t \in [\delta_{min}, \delta_{max}]$, and its throttle (or braking) $\gamma_t \in [-1, 1]$, in their physical ranges, respectively. We elaborate on the form of function f in the Sec. 5.5.1 and 5.6.1.

Remark 5 *At each time step t, it is assumed that the state vector of the ego ζ_t is obtained through its onboard sensors.*

5.3.2 Model of Environment

Ego's environment *env* comprises various elements, including lanes, traffic signs, markings, ego, and other traffic participants like cars, pedestrians, and bicyclists. To maintain simplicity, we treat all participants as vehicles and utilize the same dynamic model (5.1) for all of them, while clarifying that our approach is applicable for diverse urban driving scenarios with various traffic participants. Notably, there are no predefined paths for the other vehicles, but we assume they adhere to all traffic

regulations, such as reducing speed and exercising caution at T-junctions. Within this environment, there are N other vehicles present, each identified by an index $i \in I = \{1, ..., N\} \subset \mathbb{Z}$. We consider that ego's sensors provide processed data about road geometry, traffic signs, and other nearby vehicles within a specific radius denoted as r_{near}. The set of indices corresponding to vehicles within this radius at time t is represented by $I_{near,\,t} \subseteq I$.

5.3.3 Traffic Rules

Traffic rules are represented using STL formulae, and their semantics are defined over a trajectory of ego's state vector ζ_t. We use the notation $\zeta_t \models \phi$ to show the satisfaction of formulae ϕ at time t with respect to ego's state vector ζ_t. We consider a finite number of traffic rules in the environment, denoted by the set Φ. At each time step t, the rules within the detection range of ego's sensors, r_{near}, are considered as *active rules*. We denote the set of active rules at time t by $\Phi_{active,\,t} \subseteq \Phi$.

Here, we present concrete examples of how to formulate typical traffic regulations using STL formulas:

(a) **Speed Limit:** When driving in an urban environment, it is essential for ego to adhere to the maximum speed limit of the lanes, denoted by v_{lim}, at all times. This rule can be described as follows:

$$\square_{[0,H-1]}\,(v \leq v_{lim}), \tag{5.2}$$

which states that at all time steps within a given interval H, ego is supposed to maintain a speed that does not exceed the speed limit of the lanes.

(b) **Stop Sign:** When a stop sign is detected, the legal obligation is to come to a complete stop prior to the sign and subsequently proceed with caution. This

rule can be expressed as follows:

$$(stop\_sign) \longrightarrow \Diamond_{[0,\, t_s]}(v = 0), \tag{5.3}$$

where the predicate $(stop\_sign)$ is true when ego detects a stop sign. The variable t_s denotes the time interval between the detection of the stop sign (at a distance r_{near}) and ego coming to a complete stop. As stop signs are commonly found in areas with low speed limits (approximately $30\,km/h$), we empirically adjust the value of t_s to account for the time required to stop the vehicle at a fixed deceleration rate within the braking distance of r_{near}, considering an initial velocity of $30\,km/h$.

(c) **Collision Avoidance:** According to the standard urban-driving regulations, it is essential for ego to consistently avoid collisions with other vehicles in its surroundings. We formulate this rule over a time interval H by:

$$\Box_{[0,\, H-1]}\,(collision\_avoidance), \tag{5.4}$$

where $(collision\_avoidance)$ refers to the predicate that denotes the absence of any collision between ego and other vehicles in the surrounding environment. In Sec. 5.5.2 and 5.6.2, we elaborate on how this predicate is evaluated.

(d) **Traffic Light:** Formulating mathematically the appropriate behavior at a traffic light proves to be a complex task. Traffic lights possess three states: red, yellow, and green, with the green light allowing unrestricted motion for the ego. In this dissertation, to simplify notation, we treat the behavior concerning yellow and red lights as identical, necessitating ego to come to a complete stop behind the light. This behavior is expressed as follows:

$$(reached\_red\_light) \longrightarrow \Diamond_{[0,t_l]}(v = 0), \tag{5.5}$$

where predicate ($reached\_red\_light$) becomes true when ego arrives at a red light. The value t_l represents the time horizon required for ego to come to a complete stop, and this value is determined through empirical tuning (similarly to the tuning of t_s in the stop sign rule). After coming to a stop, ego is expected to remain stationary and wait for the traffic light to change to green, which can be expressed as follows:

$$(red\_light) \longrightarrow (v = 0)\,\mathcal{U}_{[0,t_r]}\,(green\_light) \tag{5.6}$$

Here, predicate (red_light) is true when ego comes to a stop behind a red light and the light remains red, and predicate ($green\_light$) is true when the traffic light changes from red to green. The value of t_r, representing the maximum duration of a red light before it transitions to green, is determined through empirical experimentation.

5.3.4 Objective Function

The high-level objective of ego is to closely adhere to a given reference path and, along with obeying traffic rules, minimize the energy consumption and maximize the comfort of the passengers.

We consider the given reference path $\mathcal{R}$ as a finite sequence of R reference waypoints $\mathcal{R} = \{P_r\}_{r=1}^{R}$. These waypoints are evenly distributed along the path and positioned in the center of their respective lanes. Each reference waypoint $P_r = [x_r, y_r, \psi_r]^\top$ comprises both its position and heading relative to the absolute coordinate frame of the environment (see Fig. 5·2). At each time step t, we represent the tracking error of ego with respect to the reference path $\mathcal{R}$ as $e_t = E(\mathcal{R}, \zeta_t)$, where the function E computes the Euclidean distance between ego's current state and the closest reference waypoint in $\mathcal{R}$.

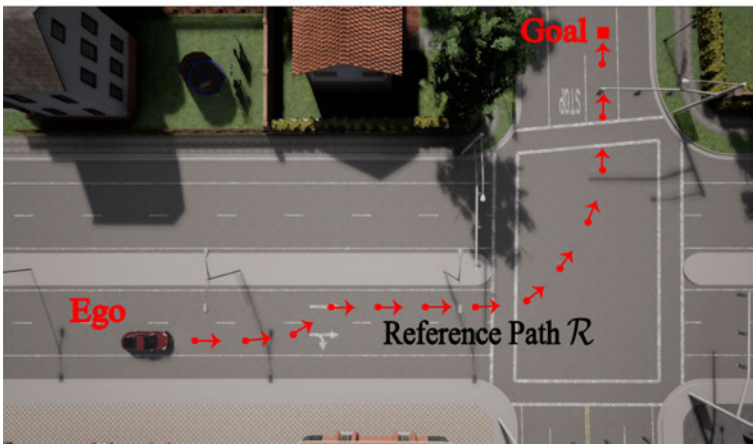


Figure 5·2: Illustration of a reference path in the simulator CARLA (Dosovitskiy et al., 2017), where the arrows indicate the heading of the reference waypoints.

By aiming to minimize the control input (u_t) and the differences in control commands between consecutive time points ($u_t - u_{t-1}$), we take into account both energy minimization and comfort maximization concerns of ego. We consider a quadratic cost function as the objective function of ego at time t, denoted by $\mathcal{F}_{obj}(u_t, u_{t-1}, e_t)$:

$$\mathcal{F}_{obj}(u_t, u_{t-1}, e_t) = ||u_t||^2 + ||u_t - u_{t-1}||^2 + e_t^2 \tag{5.7}$$

Note that generally the objective function would be the weighted summation of its objective components. Here, for simplicity of notation, we assume the objective terms are of the same importance to the user and hence they are weighted equally.

The deterministic control problem can now be formulated as following:

Problem 3 (Deterministic Control Problem) *Given the dynamic model of ego* (5.1)*, its environment env, the set of traffic rules* Φ*, the reference path* $\mathcal{R}$*, and the*

objective function (5.7)*, find a control sequence* $u^*_{1:T}$ *such that:*

$$\begin{aligned} \mathbf{u}^*_{1:T} = \arg\min_{\mathbf{u}_{1:T}} & \sum_{t=1}^{T} \mathcal{F}_{obj}(u_t, u_{t-1}, e_t) \\ s.t. \quad & \forall\, t \in \{1, ..., T\} : \\ & \zeta_{t+1} = f(\zeta_t, u_t), \\ & e_t = E(\mathcal{R}, \zeta_t), \\ & \zeta_t \models \phi \qquad \forall \phi \in \Phi_{active,\,t} \end{aligned} \tag{5.8}$$

5.4 Two-Level Control Approach

We propose a two-level controller designed to navigate ego from its initial state to the goal (Fig. 5·3), while ensuring compliance with runtime performance requirements. At the high level of our approach, we create an MPC problem, utilizing a simplified representation of ego's dynamic model and its surrounding environment, to calculate the optimal control input at each time step. To satisfy the traffic rules, we impose them as Mixed-Integer Linear Constraints (MILCs) into the MPC. The resulting optimal control generated by the high-level controller is then utilized in the lower level, to simulate ego's trajectory using detailed models for ego's dynamic and its environment. STL-based runtime monitoring is implemented in low-level controller to verify the correctness of ego's behavior, and adjust it if needed. In the following sections, we elaborate on each layer of the controller.

5.5 High-Level Controller

At the higher level of our approach (Fig. 5·3), we employ a simple bicycle dynamic model for ego and simple representations for other traffic participants. We formulate an MPC problem with planning horizon H, along with a quadratic cost function to address ego's objective function. Inspired by the works in (Raman et al., 2014; Sadraddini and Belta, 2015), we translate the traffic rules into MILCs, and at each

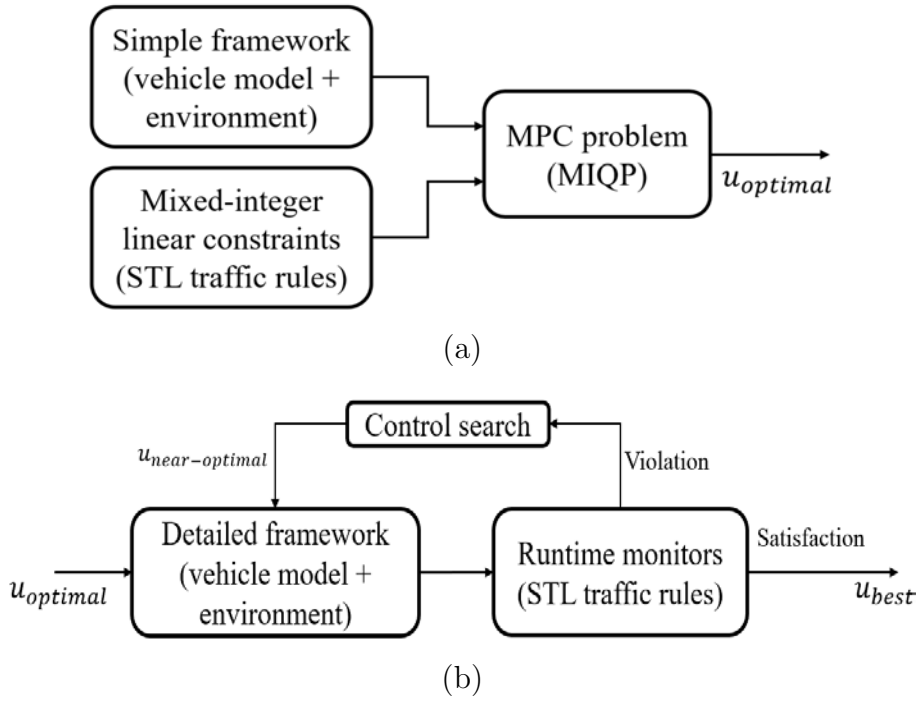


Figure 5·3: (a) Overview of the proposed method (Aasi et al., 2021): (a) high-level controller based on MPC approach, (b) low-level controller based on runtime monitoring

time step the set of translated active rules are integrated into the MPC problem, resulting in a Mixed-Integer Quadratic Programming (MIQP) optimization problem.

5.5.1 Simplified Dynamic Model of Ego

As a simplified dynamic model for ego, we consider a kinematic bicycle model as:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \psi_{t+1} \\ v_{t+1} \end{bmatrix} = \begin{bmatrix} v_t \cos\psi_t \\ v_t \sin\psi_t \\ \frac{v_t}{l_f}\tan\delta_t \\ \beta_t \end{bmatrix} \Delta t + \begin{bmatrix} x_t \\ y_t \\ \psi_t \\ v_t \end{bmatrix}, \tag{5.9}$$

where l_f is the distance from the center of the mass to the front axle, and β_t is ego's acceleration (or deceleration) (see Fig. 5·4). We assume the model is driven by the front-wheel steering. Therefore, if we denote the parameters of front and rear wheels by subscripts $(\cdot)_f$ and $(\cdot)_r$, respectively, we have: $\delta_f = \delta, \delta_r = 0$.

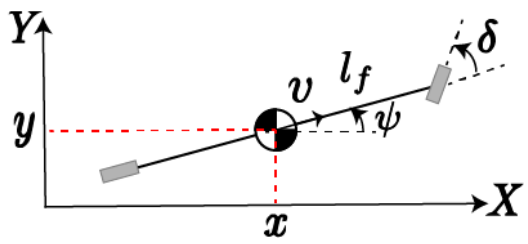


Figure 5·4: Representation of bicycle model as the simplified dynamic model of ego, where we assume the model is driven by the front-wheel steering

Remark 6 *We use benchmark data from simulator CARLA to estimate the mapping between ego's throttle (or braking) γ_t and its acceleration (or deceleration) β_t.*

Since the bicycle model (5.9) is nonlinear, we use its linearized version in the MPC problem of the high-level controller. We denote the linearized bicycle model by:

$$\zeta_{t+1} = f_s(\zeta_t, u_t) \tag{5.10}$$

In App. A, we provide exact formulation of the linearized bicycle model (5.10).

5.5.2 Simplified Model of Environment

We assume at each time step t, ego's sensors are able to provide deterministic information about the velocity and heading of the nearby vehicles in $I_{near,t}$. Using the bicycle model (5.9), we predict the trajectories of nearby vehicles over the time interval $[t, t+H]$. This involves assuming that nearby vehicles maintain a constant velocity and heading throughout the MPC horizon H. Furthermore, to simplify the representation of traffic participants, we regard all vehicles in the environment as point masses.

If we denote the position of the i^{th} vehicle at time t by $(x_{i,t}, y_{i,t})$, the set of nearby vehicles can be considered as $I_{near,t} = \{i \in I \mid \|(x_t, y_t) - (x_{i,t}, y_{i,t})\| \leq r_{near}\}$. To ensure collision avoidance between the ego vehicle and nearby vehicles, a common

approach is to maintain a minimum safety distance d_{safe} between them, measured using the Euclidean distance. However, imposing quadratic non-convex constraints can result in longer algorithm runtimes. To tackle this issue, we propose an approximation by using the 1-norm distance instead of the Euclidean norm. We verify collision avoidance requirement through the predicate (*collision_avoidance*), which is satisfied if $\forall i \in I_{near,t} : |x_t - x_{i,t}| + |y_t - y_{i,t}| \geq d_{safe}$. It's worth noting that the 1-norm distance serves as an upper bound for the Euclidean distance, making it a more conservative collision avoidance requirement, applied as a linear constraint in the optimization of MPC.

5.5.3 Rules of the Road

At each time step t, given the set of active traffic rules $\Phi_{active,t}$, we employ the methods from (Raman et al., 2014; Sadraddini and Belta, 2015) to translate the STL formulae of these rules into MILCs. We present the corresponding MILCs that ensure the fulfillment of rules in a concise manner, using the notation: $\forall \phi_a \in \Phi_{active,t} : z_{\phi_a} = 1 \implies \zeta_t \models \phi_a$. The binary variable $z_{\phi_a} \in \{0, 1\}$ represents whether the rule ϕ_a is satisfied or violated, where $z_{\phi_a} = 0$ indicates violation, and $z_{\phi_a} = 1$ indicates satisfaction.

It is important to note that certain traffic rules, such as speed limit and collision avoidance, must be adhered to at all time steps within the horizon H (see Sec. 5.3.3). Additionally, if any active rule ϕ_a has a time horizon $hrz(\phi_a)$ that extends beyond the MPC's horizon H, we keep track of ego's states over the horizon $hrz(\phi_a)$ to accurately evaluate rule satisfaction within the receding horizon framework.

5.5.4 MPC Formulation

To formulate the MPC optimization problem, a timed sequence of waypoints is required to define the desired trajectory over the MPC horizon. However, the existing

reference path $\mathcal{R}$ only provides an untimed sequence of reference waypoints for navigation in the environment. To overcome this issue, at each time step t, the current state of ego is considered along with the dynamic model (5.9) and the reference path $\mathcal{R}$ to estimate ego's motion over the MPC horizon. Subsequently, a desired trajectory is extracted for the time interval $[t, t+H-1]$ in the form of H desired waypoints $[w_t, w_{t+1}, ..., w_{t+H-1}]$. Each desired waypoint $w_t = [x_{des,t}, y_{des,t}, \psi_{des,t}, v_{des,t}]^\top$ comprises its position $(x_{des,t}, y_{des,t})$ and heading $\psi_{des,t}$, which corresponds to one of the reference waypoints P_r from the reference path. Additionally, each desired waypoint is associated with a desired speed $v_{des,t}$ (see Fig. 5·5).

The value of the desired speed is adjusted based on various factors, including the user's preferences, the speed limit of the lanes, and the ego vehicle's current situation with respect to the environment. For instance, $v_{des,t}$ may be set to the maximum speed limit of the lanes to enable the vehicle to move as fast as possible. On the other hand, when approaching a red light, $v_{des,t}$ is set to zero to ensure the vehicle comes to a complete stop.

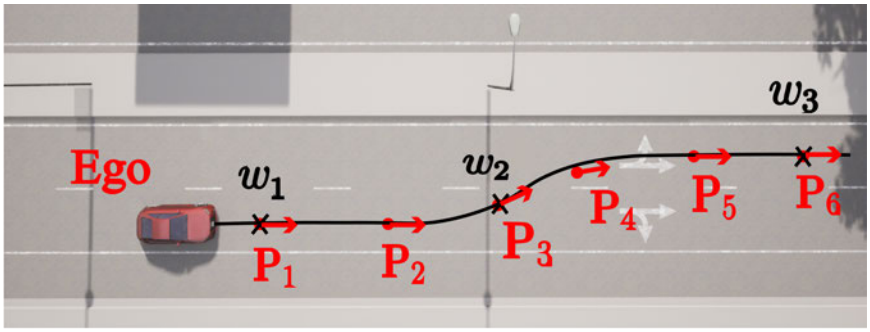


Figure 5·5: Representation of the desired trajectory, obtained from the reference path $\mathcal{R}$. The reference waypoints P_1, P_2, and P_3 of the reference path are associated with the positions and headings of the desired waypoints w_1, w_2, and w_3, respectively. These desired waypoints have been set to a speed of 30, km/h in accordance with the velocity limit of the lane. The interpolation between these waypoints is illustrated by the black curve.

Remark 7 *It should be noted that the desired trajectory $w_{t:t+H-1}$ is computed based*

on the simplified bicycle model of the ego. As a result, it may not always be dynamically feasible to follow this trajectory at every time step along MPC horizon, especially when considering the precise and intricate dynamic model of the ego.

In order to satisfy the objective function of ego $\mathcal{F}_{obj,t}(u_t, u_{t-1}, e_t)$, we formulate the MPC's cost function as:

$$\mathcal{F}_{cost,t}(u_t, u_{t-1}, e_t) = ||u_t||^2 + ||u_t - u_{t-1}||^2 + ||\zeta_t - w_t||^2, \tag{5.11}$$

where the expression $||\zeta_t - w_t||^2$ takes into account the tracking error concerning the reference path $\mathcal{R}$. Consequently, we can formulate the MPC problem as following:

$$\begin{aligned} u^*_{t:\,t+H-1} = \underset{u_{t:\,t+H-1}}{\arg\min} & \sum_{k=t}^{t+H-1} \mathcal{F}_{cost,\,k}(u_t, u_{t-1}, e_t) \\ \text{s.t.} \quad & \forall k \in \{t, ..., t+H-1\} : \\ & \zeta_{k+1} = f_s(\zeta_k, u_k), \\ & z_{\phi_a} = 1, \quad \forall\, \phi_a \in \Phi_{active,\,k}, \\ & \zeta_{min} \leq \zeta_k \leq \zeta_{max}, \\ & u_{min} \leq u_k \leq u_{max} \end{aligned} \tag{5.12}$$

where $u^*_{t:t+H-1} = [u^*_t, ..., u^*_{t+H-1}]$ represents the optimal control sequence over the MPC horizon H, and the expressions $\zeta_{min} \leq \zeta_k \leq \zeta_{max}$ and $u_{min} \leq u_k \leq u_{max}$ represent the constraints imposed by the physical limitations of ego in terms of its state and control input, respectively.

Remark 8 *In our approach, lane keeping is not explicitly regarded as a traffic rule. Instead, we assume this requirement is fulfilled when ego follows the reference path (through optimizing the objective function), which is made of reference waypoints positioned in the center of the lanes.*

Discussion 1 *It should be noted that addressing the recursive infeasibility of MPC problems has been a persistent issue extensively studied in the literature (Kousik et al.,*

2020; Koller et al., 2018; Brüdigam et al., 2023). In case our MPC problem in (5.12) *becomes infeasible at a specific time step t, we pass the optimal solution from the previous step $t-1$ to the lower level of our framework. This decision is motivated by the assumption that the environment and ego states undergo slight changes within a single time step, especially with small time steps ($\Delta t \leq 0.1\,s$). As a result, the optimal solution from the previous time step is likely to be near-optimal at the current time step and complies with the rules.*

5.6 Low-Level Controller

The main objective of the low-level controller is to enforce correctness into ego's behavior and compensate for the mismatch between simple models used in the high-level controller and the complex, real-world models. In the low level of our approach (Fig. 5·3), we use the optimal solution from the high-level controller to forward-simulate the ego's trajectory over the MPC's planning horizon H. The trajectory is forward-simulated using a detailed dynamic model for ego (Sec. 5.6.1) and a detailed representation of other vehicles (Sec. 5.6.2). STL runtime monitors (Bartocci et al., 2018) are constructed from the active traffic rules, serving to confirm the adherence of ego's behavior to the active traffic rules (Sec. 5.6.3). If the ego's simulated trajectory accurately adheres to all traffic rules, the first control element is applied to ego. Alternatively, if the trajectory violates one or more of the active rules, we randomly sample the control space within a small region around the optimal control. This process aims to find a near-optimal control that ensures compliance with all traffic rules.

5.6.1 Detailed Dynamic Model of Ego

We employ an extended bicycle model, similar to the one taken in (Turri et al., 2013), to serve as the detailed dynamic model of ego. This model incorporates the non-linear Pacejka tire force model for accurate computation of tire forces (Pacejka, 2005). The

comprehensive formulation of the dynamic model is provided in App. B. We denote the discrete-time formulation of this model by:

$$\zeta_{t+1} = f_d(\zeta_t, u_t) \tag{5.13}$$

5.6.2 Detailed Model of Environment

In the detailed model of the environment, we consider the axis-aligned bounding boxes of vehicles for verifying the satisfaction of collision avoidance constraint. The bounding box of the i^{th} vehicle at time step t is denoted by $\mathcal{BB}_{i,t}$, and we use $\mathcal{BB}_{near,\,t} = \bigcup_{i \in I_{near,\,t}} \mathcal{BB}_{i,t}$ to represent the combined bounding boxes of nearby vehicles. The ego's bounding box is identified as $\mathcal{BB}_t$. The fulfillment of collision avoidance constraints is assessed through the intersection analysis of these bounding boxes. Therefore, within the low-level controller, we consider the predicate $(collision\_avoidance)$ is true at time t if $\mathcal{BB}_t \cap \mathcal{BB}_{near,\,t} = \varnothing$.

5.6.3 Runtime Monitoring of Traffic Rules

We utilize runtime monitoring techniques as described in (Ulus, 2019) to validate the adherence of ego's trajectory to active traffic rules, considering the detailed dynamic model of ego and the detailed representation of the environment. Initially, we conduct a forward simulation of ego's trajectory over the predictive horizon H, utilizing the optimal control input u^*. Should this trajectory conform to all active traffic rules, its initial element is applied on ego. In cases where the trajectory violates one or more traffic rules, we undertake a random sampling of the control space within an r-ball region denoted as C_{u^*}, encompassing the optimal control. This region is defined as $C_{u^*} = \{u \,|\, ||u - u^*|| < r\}$, where r is a small positive value. Subsequently, we proceed with a forward simulation of ego's trajectory using the newly sampled control input and re-evaluate the compliance with active traffic rules. This iterative process is

repeated, for a maximum number of N_{iter} iterations, until a suitable control input denoted as u_{best} is identified, ensuring the fulfillment of all active traffic rules (see Fig. 5·6).

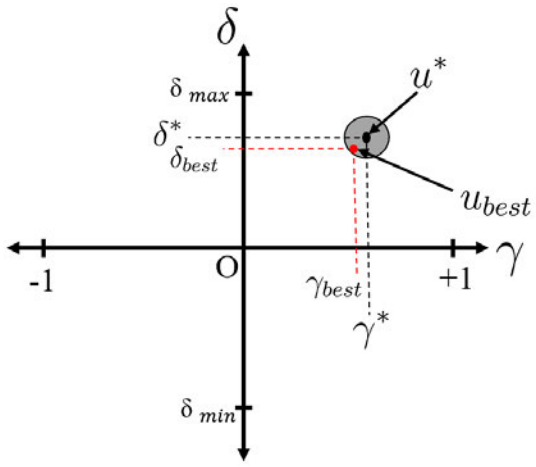


Figure 5·6: Representation of the control search process in low-level controller. The gray area C_{u^*} around the optimal control u^* is the narrow region that we sample randomly to find the best control input u_{best} that satisfies all traffic rules.

If the low-level controller cannot find the best control input within the sampling threshold N_{iter}, we apply full braking on ego. Handling infeasibility is a big concern in the field of self-driving cars. We choose full-breaking, because we are interested in urban driving setting, and the recommended course of action by law in most of the urban driving scenarios is to decrease speed and perform an emergency stop.

5.7 Case Studies

The performance and capabilities of our proposed control method are validated through four urban-driving simulation case studies:

(i) **Comparison Scenario:** Since most of the existing works are based on NMPC formulation, in this case study we implement a naive NMPC method and com-

pare its performance as a baseline with our framework (Sec. 5.7.1). This scenario aims to underscore the enhanced solution quality and expedited runtime achievements of our approach in relation to an NMPC method.

(ii) **Traffic Rules Scenario:** This scenario presents a comprehensive evaluation of our method in a typical urban driving setting (Sec. 5.7.2). The case study demonstrates how well our proposed framework tracks the reference path and adheres to traffic rules, while maintaining real-time performance efficiency.

(iii) **Safety Scenario:** In this scenario we emphasize the importance of our low-level controller and how it compensates for the mismatch between simple models used in our high-level controller, and the complex, real-world models (Sec. 5.7.3).

(iv) **Scalability Scenario:** This scenario examines how the performance of our proposed framework scales with the number of nearby vehicles (Sec. 5.7.4).

In all scenarios, ego must track the reference path as close as possible, while also ensuring compliance with a set of traffic rules, which we detail for each scenario. In the schematics of all scenarios, the reference path is shown by a red line that connects the starting point A to the destination point B.

We use the CARLA simulator (Dosovitskiy et al., 2017) for implementing the scenarios, and Gurobi package (Gurobi Optimization, 2020) for solving the MPC problem in the high-level controller. The runtime speed evaluation of our algorithm and the NMPC baseline is based on the system's clock frequency, which indicates how often the control loop and environment are updated. This frequency is denoted in Frames Per Second (FPS) in our figures. In all scenarios, we use the empirically-tuned hyper-parameters in Table 5.1.

Table 5.1: Hyperparameters used in the simulation scenarios of our deterministic control approach

Parameter	Value	Parameter	Value	Parameter	Value
H	10	Δt	$0.1\,s$	l_f	$2.1\,m$
r_{near}	$10\,m$	d_{safe}	$4.5\,m$		

5.7.1 Comparison Scenario

In order to demonstrate the effectiveness of our control algorithm in comparison to prior studies utilizing NMPC, we have designed a scenario where ego has to follow a reference path along a curved road, featuring varying speed limit signs along the route (see Fig. 5·7).

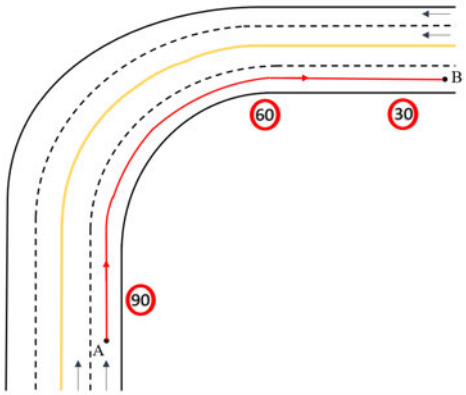


Figure 5·7: Schematic of the comparison scenario, with three different speed limits along the reference path of ego (Aasi et al., 2021).

We initially developed the NMPC baseline using the detailed dynamic model employed in our algorithm's low-level controller (Sec. 5.6.1). However, this approach exhibited an average runtime performance of under $5\,Hz$, which contributed to controller instability. Consequently, to ensure the comparability of the results, we redefined the NMPC baseline utilizing the simple bicycle model from the high-level controller (Sec. 5.5.1), and here we present the results of this approach. It is important

to highlight that both our control method and the NMPC baseline were executed with identical planning horizons, reference paths, and cost functions for the MPC problem. Illustrated in Figure 5·8, our proposed method demonstrates significantly enhanced runtime speed compared to the NMPC baseline.

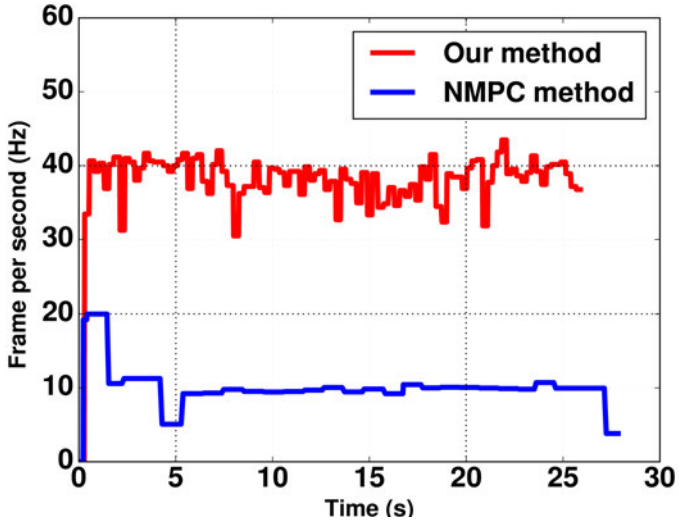
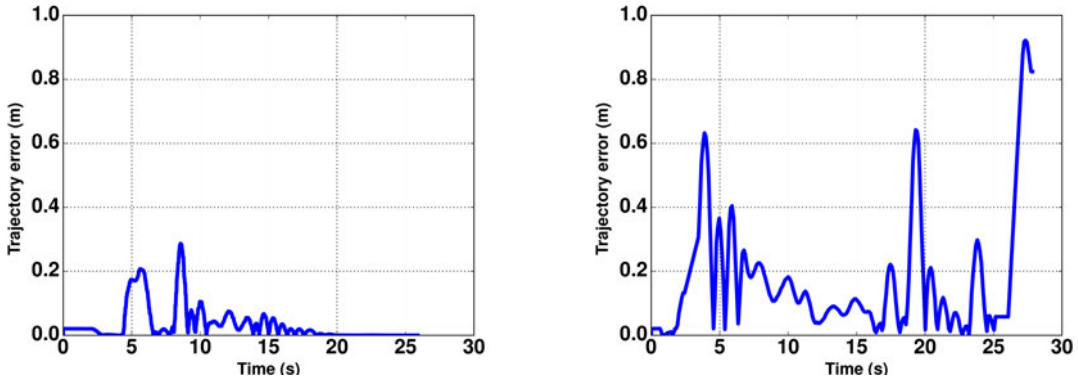


Figure 5·8: Runtime performance of our proposed approach compared to the NMPC baseline (Aasi et al., 2021).

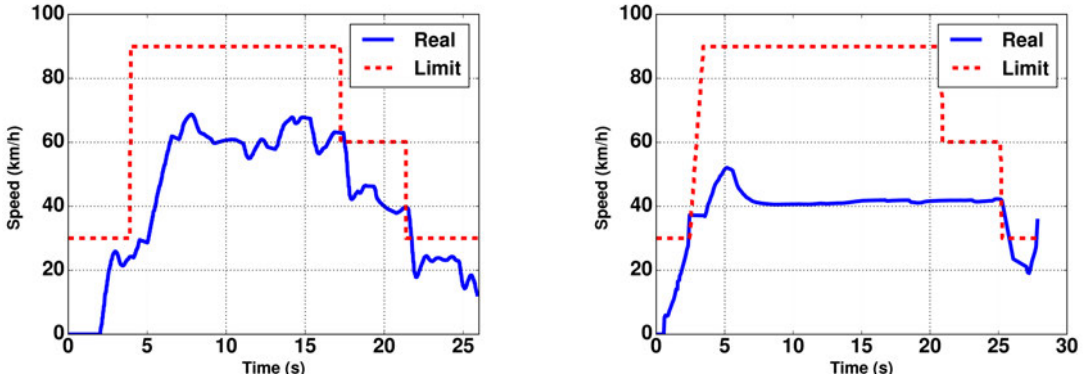
In Fig. 5·9 the tracking error of our algorithm compared to the NMPC baseline, is illustrated over time. Our algorithm showcases a superior performance, demonstrating smaller average tracking error compared to the NMPC baseline. Our algorithm exhibits a maximum tracking error of $0.29\,m$, a significant improvement over the NMPC baseline's value of $0.93\,m$. Moving on to Fig. 5·10, we present the speed profile of our algorithm compared to the NMPC baseline, where our algorithm achieves higher legal speeds on average than the NMPC implementation.

Overall, our assessment suggests that the NMPC problem is more susceptible to becoming trapped in locally-optimal solutions, which negatively impacts the overall performance of the NMPC baseline. This contrasts with our MIQP formulation of the MPC problem, which appears to mitigate such issues effectively.



(a) Our method's tracking error (b) NMPC baseline's tracking error

Figure 5·9: Tracking error of our algorithm, compared to the NMPC baseline (Aasi et al., 2021). The blue line shows the real speed of ego, and the red dashed line shows the speed limit of the lanes.

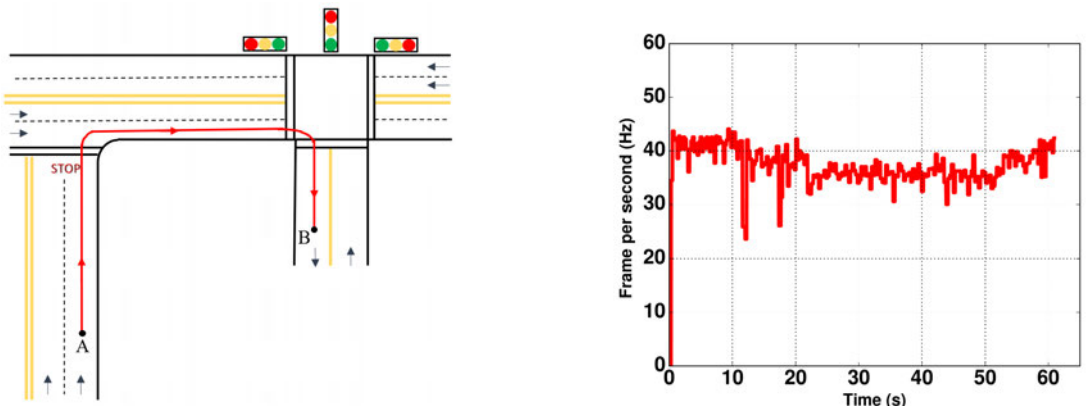


(a) Our method's speed profile (b) NMPC baseline's speed profile

Figure 5·10: Speed profile of our algorithm, compared to the NMPC baseline (Aasi et al., 2021)

5.7.2 Traffic Rules Scenario

This case study demonstrates the effectiveness of our algorithm in adhering to traffic rules. In this scenario, as ego tracks the reference path, it encounters a stop sign and a traffic light (see Fig. 5·11.(a)). In Fig. 5·11.(b), we illustrated the runtime performance of our method, which demonstrates that, overall our framework maintains a high-speed runtime performance throughout its functioning.



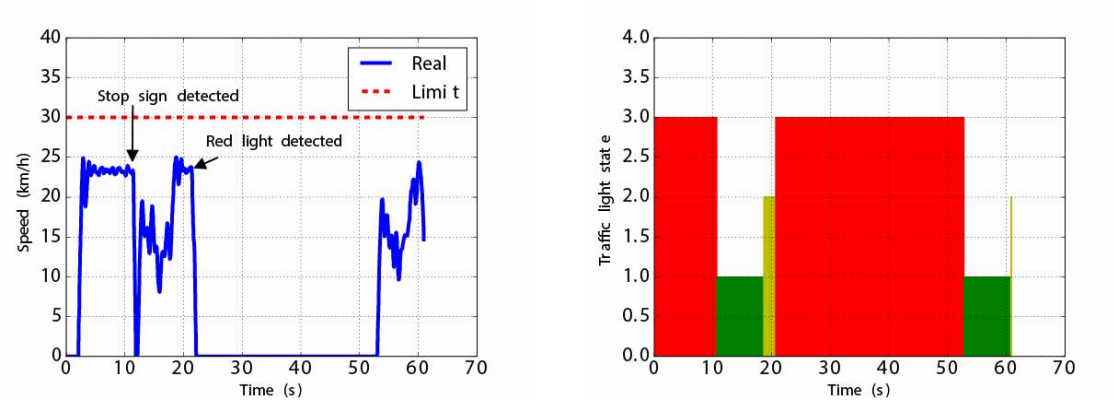
(a) Schematic of scenario (b) Our method's runtime speed

Figure 5·11: (a) Schematic of traffic rules scenario, where ego is subject to a default 30 km/h speed limit rule, and at some point the stop sign and traffic light rules get activated (Aasi et al., 2021); (b) Runtime performance of our proposed framework (Aasi et al., 2021).

In Fig. 5·12.(a), ego's speed profile is illustrated, accompanied by the corresponding state of the traffic light as depicted in Fig. 5·12.(b). In Fig. 5·12.(a), ego detects a stop sign at 11.2 s, prompting a gradual reduction in its speed to come to a halt close to the stop sign by 11.9 s. Subsequently, it recognizes a red light at 21.7 s, leading to a complete stop by 22.1 s. Ego remains stationary until it detects a green light at 52.8 s, at which point it starts moving again. The figures highlight the rapid responsiveness of ego to the dynamic events occurring within its surroundings.

5.7.3 Safety Scenario

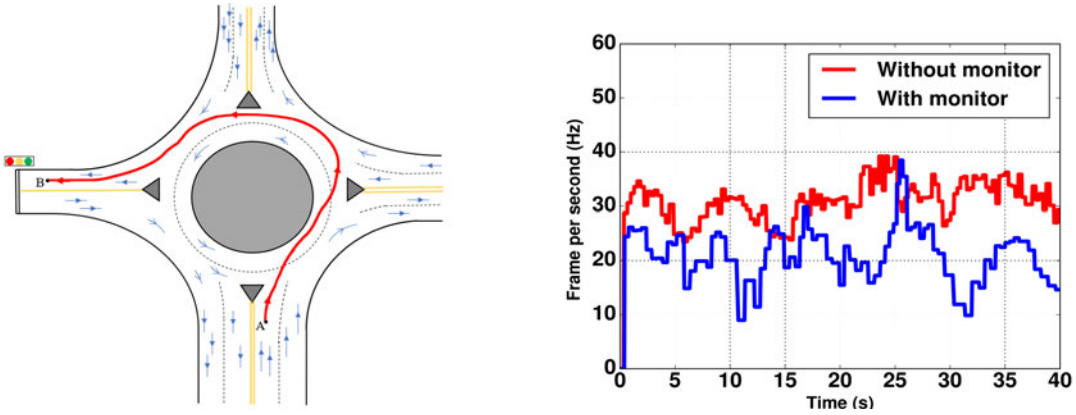
This scenario depicts how the low-level controller in our framework addresses the mismatch between simplified models utilized in the high-level controller of our approach, and the complicated real-world models. Schematic of the scenario is presented in Fig. 5·13.(a), involving 160 other vehicles in the environment. In Fig. 5·13.(b), the runtime speed of our controller is displayed, both with and without applying the runtime monitoring at the low-level controller. It is evident that the average runtime



(a) Our method's speed profile (b) Traffic light status

Figure 5·12: (a) Speed profile of ego, where the activation times of stop sign and traffic light rules are indicated (Aasi et al., 2021); (b) The color-status of the traffic light over time (Aasi et al., 2021).

speed is lower in the "with monitor" case, compared to the "without monitor" configuration. This discrepancy is justifiable, as the lower-level controller takes into account detailed models with additional constraints, resulting in a reduction of runtime speed in the "with monitor" implementation, on average.



(a) Schematic of scenario (b) Our method's runtime speed

Figure 5·13: (a) Schematic of safety scenario, where the small blue arrows show the direction of motion of other vehicles in the environment (Aasi et al., 2021); (b) Runtime performance of our proposed framework, with and without applying runtime monitoring in the low-level controller (Aasi et al., 2021).

The primary objective of this scenario is to demonstrate the crucial function of the low-level controller in ensuring safety. This concept is visually explained in Fig. 5·14. The diagram depicts ego's minimum distance from nearby vehicles over a period of time, including implementations with and without runtime monitoring in the low-level controller.

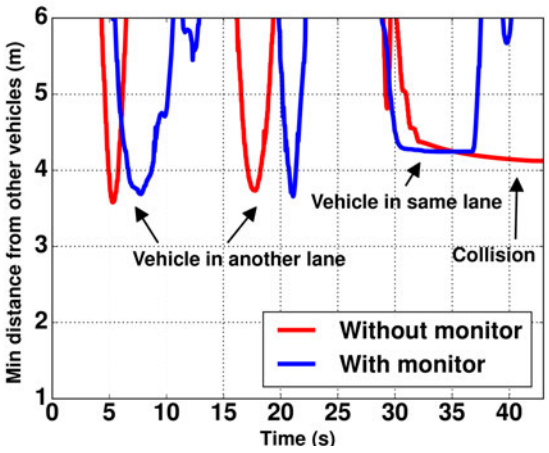


Figure 5·14: Representation of minimum distance between ego and its nearby vehicles, over time (Aasi et al., 2021).

All vehicles in the environment are represented by bounding boxes with uniform dimensions: length of $4.2\,m$ and width of $1.8\,m$. Consequently, if two vehicles in the same lane come within a distance of less than $4.2\,m$, or two vehicles in parallel lanes get closer than $1.8\,m$ to each other, a collision is deemed to have occurred. In the presented scenario, in both of "with monitor" and "without monitor" implementations, at time instances $\sim 6\,s$ and $\sim 19\,s$, ego detects the nearest nearby vehicle in the parallel lane, and the distance between them is bigger than $3.5\,m$, indicating the absence of a collision.

At time $\sim 32\,s$, ego detects a stationary vehicle in its lane. Ego decelerates and comes to a stop behind the stationary vehicle. Notably, in the "without monitor" case, the distance between vehicles gradually reduces to less than $4.2\,m$, resulting

in a collision. Conversely, in the "with monitor" implementation, ego maintains a distance greater than $4.2\,m$, showcasing how the runtime monitoring at the low-level controller contributes to ensuring ego's safety.

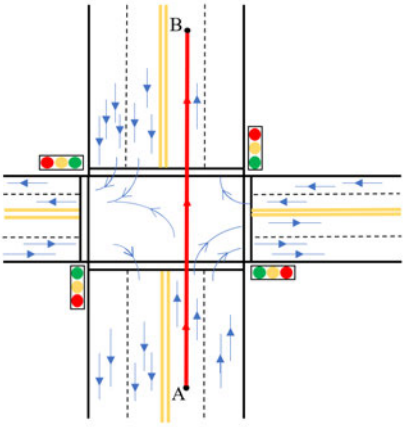
5.7.4 Scalability Scenario

This scenario demonstrates the scalability of our framework in relation to the number of nearby vehicles. To address this concern, we increase the detection range of ego's sensors r_{near} from $10\,m$ to $40\,m$. This adjustment allows for the incorporation of a larger number of other vehicles into our calculations. A schematic of the scenario is provided in Fig. 5·15.(a), involving 121 other vehicles in the environment.

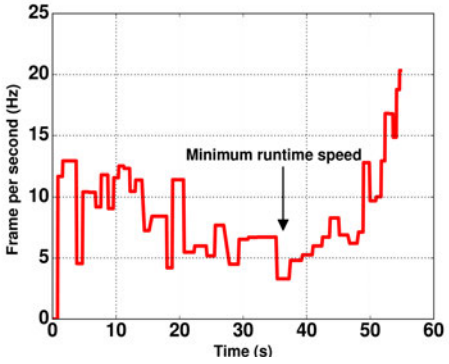
In Fig. 5·15.(b) and 5·15.(d), the runtime speed and the number of nearby vehicles detected by ego, are plotted over time. Throughout most of the scenario's runtime, ego detects on average more than 6 nearby vehicles, and close to the time $36\,s$, it detects 18 nearby vehicles and predicts their future trajectories in the detailed framework, which drops the runtime speed to its minimum value of $4\,Hz$. Fig. 5·15.(c) displays ego's minimum distance from nearby vehicles over time, consistently affirming its adherence to maintain a safe distance from nearby vehicles.

5.8 Summary

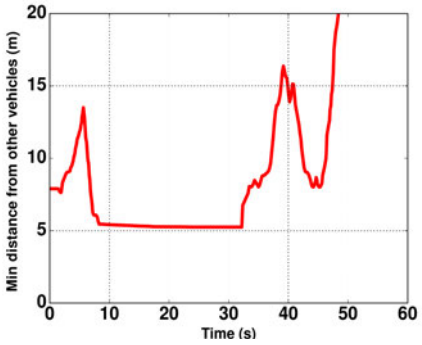
In this chapter, we presented a two-level control architecture for navigating a fully-autonomous vehicle along a given reference path, while adhering to traffic rules and runtime performance requirements. We evaluate the performance of our framework, in terms of solution quality and runtime, through various urban-driving simulation scenarios, and demonstrate its advantages over an NMPC baseline.



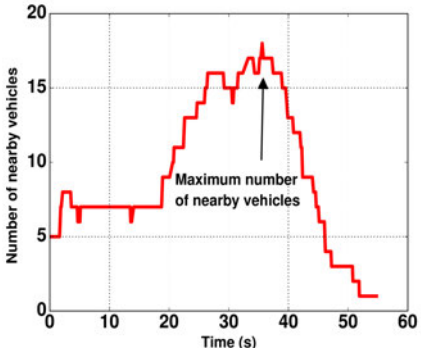
(a) Schematic of scenario



(b) Our method's runtime speed



(c) Minimum distance from nearby vehicles



(d) Number of nearby vehicles

Figure 5·15: (a) Schematic of safety scenario, where the small blue arrows show the direction of motion of other vehicles in the environment (Aasi et al., 2021); (b) Runtime performance of our method, with indication of its minimum value (Aasi et al., 2021); (c) Minimum distance from nearby vehicles (Aasi et al., 2021); (d) Number of detected nearby vehicles, with indication of its maximum value (Aasi et al., 2021).

Chapter 6

Autonomous Vehicle Control in Stochastic Environments

In this chapter, we extend the problem setting of the last chapter, from deterministic environments to stochastic ones. Inspired by our proposed method in Chapter 5, we develop a two-level control architecture to navigate ego safely along a given reference path, in the presence of uncertainty in predicting the future trajectories of other vehicles. Our algorithm achieves adherence to traffic rules and obtains a promising performance in terms of runtime speed and solution quality.

6.1 Introduction

Autonomous navigation in complex urban environments, avoiding collisions with other traffic participants, and adhering to traffic regulations constitute essential challenges for self-driving vehicles. Moreover, the movement of autonomous vehicles must optimize factors like energy efficiency, distance covered, and tracking error concerning a given reference path. Coping with the escalating challenges of urban driving scenarios while upholding diverse traffic rules presents demanding issues in achieving real-time, safe planning, and control for autonomous vehicles (Albrecht et al., 2021). This task is notably complicated given the array of uncertainties originating from sources such as localization, or estimation of the behavior of other traffic participants. Such challenges have received substantial attention in recent years, with a majority of solutions founded on MPC methodologies (Suh et al., 2018; Carvalho et al., 2014; Gray

et al., 2013; Kabzan et al., 2019; Lenz et al., 2015; Khaitan et al., 2021). Nevertheless, a significant proportion of existing control algorithms for autonomous vehicles lack convergence guarantees, primarily due to their reliance on nonlinear, nonconvex optimization methods, rendering them computationally expensive. Consequently, most of the existing approaches cannot handle real-time performance requirements within urban scenarios.

In this chapter, we expand the scope of the problem investigated in the previous chapter, transitioning from deterministic urban-driving environments to stochastic ones. Specifically, we focus on navigating ego along a given reference path. This involves satisfying a set of high-level objectives, like energy minimization and comfort maximization, while also abiding by common traffic rules, such as stop signs and traffic lights, expressed by STL specifications. From ego's perspective, we take into account the uncertainty associated with predicting the future states of other traffic participants, in the form of a posterior normal distribution, characterized by its mean and covariance. We assume that this distribution is available for ego, through an inference technique it executes.

In order to tackle this problem, we present a two-level, MPC-based control approach (see Fig. 6·1). Inspired by our method in Chapter 5, the top-level of our algorithm employs simplified models of ego's dynamics and its surrounding environment. This is combined with MILC translations of STL traffic rules, to formulate an MIQP optimization problem.

Subsequently, the top-level's optimal solution undergoes forward simulation in the lower level of our framework. In this level, a detailed dynamic model for ego is utilized, and the correctness of the simulated trajectory is assessed relative to a detailed representation of the environment. If the optimal control satisfies all traffic regulations in the detailed setting, the first control element is applied on ego. Otherwise, a

collection of MILCs, referred to as *correction constraints*, concerning ego's state and control input, is derived. These constraints are imposed upon the MPC problem at the top-level of our framework, to be solved again.

We establish the efficacy of our method by conducting simulations within a collection of complex urban-driving scenarios, using the CARLA simulator (Dosovitskiy et al., 2017). Moreover, we illustrate that our approach surpasses the runtime performance and solution quality achieved by two state-of-the-art baseline methods.

6.2 Related Works

In the previous chapter, we delved into the effectiveness of MPC approach (Camacho and Alba, 2013), as a prominent method for controlling self-driving cars in deterministic environments. In this section, we explore the literature concerning MPC-based control of autonomous vehicles, within stochastic urban-driving environments.

As the earlier works in this area, the work in (Carvalho et al., 2014) combines environment modeling with stochastic MPC. Their method involves an interacting multiple model Kalman filter for target vehicle position estimation and prediction, which contributes to a chance-constrained MPC formulation. The work in (Lenz et al., 2015) introduces the use of stochastic model predictive control with chance constraints for autonomous driving, by employing a condensed formulation of a linearized vehicle model to create a solvable quadratic program with nonlinear chance constraints, thus enabling application of standard optimization algorithms. The uncertainty in the human driver's behavior is investigated in the MPC-based approach presented in (Liu et al., 2015). The authors of (Zhang et al., 2016) focus on developing an MPC strategy for enhancing the autonomous mobility-on-demand system, by optimizing the route of ego.

Later, in the work (Jardine and Givigi, 2018), they introduce a robust MPC algo-

rithm for determining optimal motion plans in the presence of bounded uncertainty, where the obstacles are depicted as sets of linear inequality constraints. The work in (Suh et al., 2018) outlines a method for lane change motion planning in complex driving scenarios for autonomous vehicles, incorporating both probabilistic and deterministic prediction techniques. In (Wang et al., 2018), they introduce a relaxed, stochastic MPC framework that combines discrete maneuvering choices and motion planning within uncertain environments. The work in (Rick et al., 2019) highlights the successful application of nonlinear MPC combined with multi-sensor fusion and uncertainty representation in achieving safety, and proposes an algorithm based on this methodology.

The study described in (Kabzan et al., 2019) combines the integration of learning and predictive control to achieve autonomous racing using complex nominal vehicle models. In the work by (Liniger and Van Gool, 2020), a safe algorithm for autonomous driving in the presence of uncertain terminal conditions is introduced, achieved by integrating game-theoretic constraints into an MPC framework. In (Eiras et al., 2021), a two-stage receding horizon control technique for autonomous driving is introduced, which addresses safety and rule adherence in the first stage through solving a mixed-integer linear programming problem. Subsequently, in the second stage, a non-linear programming problem is employed to enhance the solution's feasibility and smoothness. In the work by (Khaitan et al., 2021), an obstacle avoidance framework addressing uncertainties in both ego's movement and the predictions of dynamic obstacles is presented.

6.3 Stochastic Control Problem

Since we are extending the scope of problem investigated in Chapter 5 from a deterministic context to a stochastic scenario, most of the requisite definitions for problem

formulation remain consistent with those outlined in Section 5.3. The primary distinction lies in the introduction of uncertainty when forecasting the trajectories of other vehicles. To tackle this issue, we assume that ego employs an inference procedure that yields a posterior Gaussian distribution, describing the future states of nearby vehicles across a time span. Given the collection of N other vehicles in the environment, indexed by $i \in I = \{1, ..., N\}$, we denote the uncertainty associated with the i^{th} vehicle by the Gaussian distribution $\mathcal{N}(\zeta_{i,t}, \sigma_{i,t})$. This distribution utilizes the vehicle's state $\zeta_{i,t}$ as its mean, and its covariance is symbolized as $\sigma_{i,t}$, which signifies the degree of estimation uncertainty. In Section 6.6.1, we delve further into a comprehensive examination of this distribution.

Hence, we can formulate the stochastic control problem as below, using the same notations as deterministic control problem in Pb. 3:

Problem 4 (Stochastic Control Problem) *Given the dynamic model of ego* (5.1)*, its environment env including N other vehicles with uncertainty distributions $\mathcal{N}(\zeta_{i,t}$, $\sigma_{i,t})$, the set of traffic rules Φ, the reference path $\mathcal{R}$, and the objective function* (5.7)*, find a control sequence $u^*_{1:T}$ such that:*

$$
\begin{aligned}
\mathbf{u}^*_{1:T} = \underset{\mathbf{u}_{1:T}}{\arg\min} & \sum_{t=1}^{T} \mathcal{F}_{obj}(u_t, u_{t-1}, e_t) \\
s.t. \quad & \forall\, t \in \{1, ..., T\} : \\
& \zeta_{t+1} = f(\zeta_t, u_t), \\
& e_t = E(\mathcal{R}, \zeta_t), \\
& \zeta_t \models \phi \qquad \forall \phi \in \Phi_{active,\,t}
\end{aligned}
\tag{6.1}
$$

6.4 Two-Level Control Architecture

We present a two-level control architecture (Fig. 6·1) to guide ego from its starting position to the destination point, while adhering to traffic rules and meeting the runtime performance criteria. In the top-level of our framework (Sec. 6.5), we ignore the uncertainty in predicting the future trajectories of other vehicles, and use the

deterministic part of ego's sensor measurements in our calculations. In this regard, we design the top-level of our approach exactly similar to the high-level controller of our approach for the deterministic problem (see Sec. 5.5), to formulate an MIQP optimization problem. The optimal output of the top-level is denoted by u^*.

In the bottom-level of our framework (Sec. 6.6), we forward simulate ego's trajectory, under u^* from the top-level, and evaluate its correctness with respect to detailed representations of ego's dynamic and its environment. In the case of satisfaction of all traffic rules, the first element of u^* is applied on ego. Otherwise, we extract a set of MILCs, called correction constraints and denoted by $\mathcal{Q}$, to compensate for the mismatch between simple models in the top-level and the real-world models. Correction constraints are imposed on the MPC problem in the top-level to be solved again.

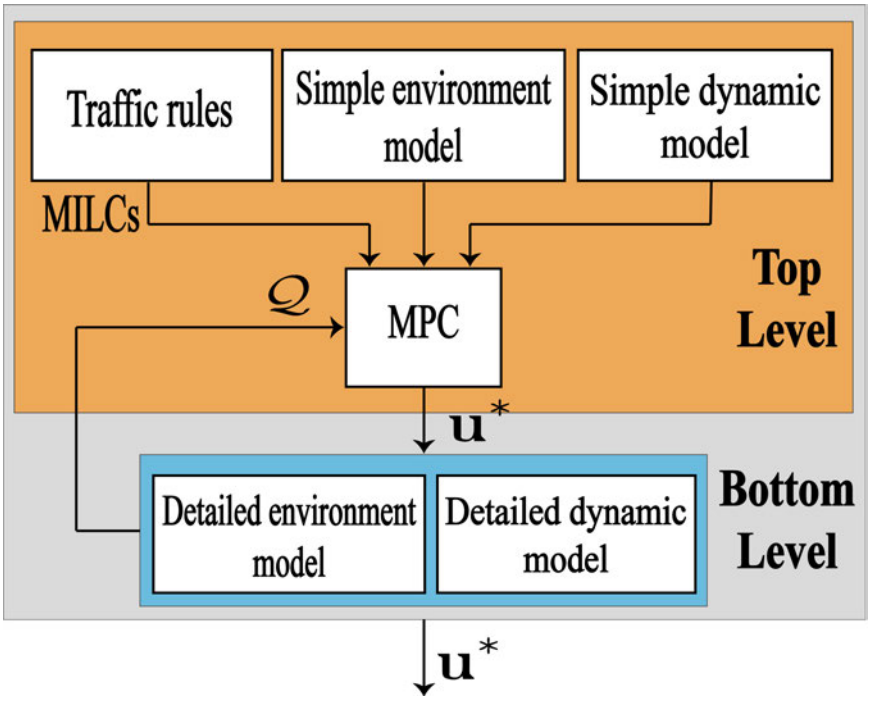


Figure 6·1: Representation of our proposed approach to the stochastic control problem in Pb. 4.

In the following we explain about each of the control levels.

6.5 Top-Level of The Control Architecture

At the top-level of our control architecture, we employ simplified models for both ego's dynamics and its surrounding environment. We utilize translated MILC representations of the STL traffic regulations to construct an MIQP optimization problem. The structure of the top-level of our control framework is closely similar to the high-level controller (see Sec. 5.5) employed in our approach to address the deterministic control problem.

To elaborate further, we adopt the kinematic bicycle model in Sec. 5.5.1 as a simplified dynamic model for ego. In our simplified environment model, we assume the absence of uncertainty when estimating the state vectors of other vehicles. We employ the deterministic aspect of ego's sensor measurements to predict the trajectories of nearby vehicles $I_{near,\,t}$, over the MPC horizon. Consequently, we apply the identical assumptions and methodologies as discussed in Sec. 5.5.2 to establish a simplified environmental model. Additionally, we employ the technique outlined in Sec. 5.5.3 to translate the STL traffic rules into MILC representations.

Utilizing the aforementioned information, we establish the same MIQP optimization problem as described in (5.12), to define our MPC problem with horizon H. At every time step t, we solve the MIQP problem to determine the optimal control sequence, denoted as $u^*_{t:\,t+H-1}$.

6.6 Bottom-Level of The Control Architecture

At the bottom-level of our control strategy, we forward-simulate ego's trajectory, under the MPC's optimal solution $u^*_{t:\,t+H-1}$ and by using detailed models for ego's dynamics and its environment. For the detailed dynamic model of ego, we opt for the extended bicycle model as outlined in Sec. 5.6.1, which was previously employed in the low-level controller of our approach to address the deterministic control prob-

lem. Within the detailed representation of environment (detailed in Sec. 6.6.1), we consider the uncertainty of predicting the trajectories of other vehicles, which is used in evaluating the satisfaction of traffic rules in the detailed framework.

In cases where ego's simulated trajectory adheres to all traffic regulations, the first control element is applied on ego. In instances of non-compliance, a set of correction constraints, denoted as $\mathcal{Q}$, are identified. These constraints pertain to the state and control inputs of ego and are subsequently integrated into the MPC problem at the top-level to be solved again (see Sec. 6.6.2). The primary intent of these constraints is to enforce correction on the MPC problem, compensating for the mismatch between the simplified models employed at the top-level and the complex, real-world models. The new output of the top-level undergoes another round of forward-simulation at the bottom-level, and if any traffic rule violations are detected, a backup controller takes over the vehicle, which applies full braking on ego. Empirical evidences presented in Sec. 6.7 demonstrate the efficacy of the correction constraints in consistently achieving accurate and optimal control for ego, thereby ensuring that our backup controller remains inactive across various scenarios.

6.6.1 Detailed Representation of Environment

In the detailed environment model, we consider the uncertainty in predicting the trajectories of other vehicles. Moreover, we use vehicles' bounding boxes to check collision avoidance between ego and other cars. Generally, we can check the collision avoidance requirement between ego and the i^{th} nearby vehicle at time t with the chance constraint:

$$p(collision \,|\, \zeta_t,\, \zeta_{i,t}) < 1 - \epsilon\,, \tag{6.2}$$

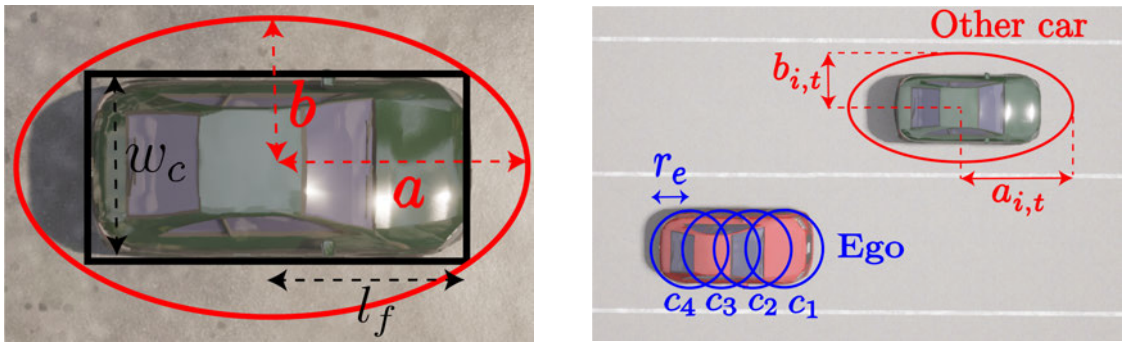
where *collision* is a predicate that is true in the case of collision between ego and the i^{th} nearby vehicle. Equation (6.2) ensures collision avoidance between ego and the

other vehicle, with probability higher than ϵ. With the assumption of conditional independence between ego and other vehicles, the inequality constraint (6.2) transforms into:

$$\mathcal{BB}_t \cap \mathcal{BB}_{i,t}(\zeta_{i,t}, \sigma_{i,t}, \epsilon) = \varnothing\,, \tag{6.3}$$

where $\mathcal{BB}_t$ denotes the bounding box of ego, and $\mathcal{BB}_{i,t}(\zeta_{i,t}, \sigma_{i,t}, \epsilon)$ represents the bounding box of the i^{th} nearby vehicle, with probability higher than ϵ. The bounding box for each vehicle is defined by the corner positions of the enclosing rectangle (see Fig. 6·2.(a)). To simplify, we assume that both ego and other cars share a uniform shape: a rectangular bounding box measuring $2\,l_f$ in length and w_c in width.

Inspired by (Schwarting et al., 2017), we adopt a conservative approach for assessing collision avoidance at the bottom-level of our framework, employing the footprint encompassing ellipse created by other vehicles' footprints. For the i^{th} vehicle, given its rectangular bounding box with orientation $\psi_{i,t}$, we can easily demonstrate that its footprint encompassing ellipse $E(a, b)$ exhibits longitudinal and lateral semi-major axes of $a = 2\sqrt{2}\,l_f$ and $b = \sqrt{2}\,w_c$ respectively. This elliptical boundary maintains the same orientation as $\psi_{i,t}$ (see Fig. 6·2.(b)).



(a) (b)

Figure 6·2: (a) Rectangular bounding box of a vehicle (black) and its footprint encompassing ellipse (red), (b) Representation of the encompassing ellipse of the i^{th} other vehicle at time t, and the estimation of ego's footprint bounding box by disks.

Without the loss of generality, we adopt an identical uncertainty growth rate denoted by $\sigma = diag[\sigma_a^2, \sigma_b^2]$ for all other vehicles in the environment. We depict the uncertainty growth for the i^{th} nearby vehicle at time t using the linear expression $\sigma_{i,t+1} = \sigma\,\Delta\,t + \sigma_{i,t}$, where $\sigma_{i,t} = diag[\sigma_{i,t,a}^2, \sigma_{i,t,b}^2]$ signifies the uncertainty, or covariance, pertaining to the vehicle's position. This covariance is roughly aligned with the axes of its enclosing ellipse.

Lemma 1 *Let's consider the Gaussian distribution $\mathcal{N}(0, \boldsymbol{\sigma}_{i,t})$, which characterizes the position uncertainty of the i^{th} vehicle at time t. This distribution is defined by the covariance $\sigma_{i,t} = diag[\sigma_{i,t,a}^2, \sigma_{i,t,b}^2]$. The contour lines of this distribution correspond to ellipses $E(a_{i,t}^u, b_{i,t}^u)$, denoted as uncertainty-ellipses, and are characterized by the semi-major axes:*

$$\begin{bmatrix} a_{i,t}^u \\ b_{i,t}^u \end{bmatrix} = \begin{bmatrix} \sigma_{i,t,a} \\ \sigma_{i,t,b} \end{bmatrix} (-2\log(2\,\pi\,\epsilon\,\sigma_{i,t,a}\,\sigma_{i,t,b}))^{0.5} \tag{6.4}$$

By utilizing Lemma 1, we estimate the bounding box of the i^{th} vehicle at time t through the Minkowski sum of its footprint encompassing ellipse $E(a, b)$ and its uncertainty-ellipse $E(a_{i,t}^u, b_{i,t}^u)$. This results in the encompassing ellipse of $E(a_{i,t}, b_{i,t})$ (see Fig. 6·2.(b)) with:

$$\begin{bmatrix} a_{i,t} \\ b_{i,t} \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} + \begin{bmatrix} a_{i,t}^u \\ b_{i,t}^u \end{bmatrix} \tag{6.5}$$

On the other side, in order to offer a conservative approximation of ego's footprint bounding box, and to calculate the collision avoidance requirement between ego and other cars with a closed-form expression as mentioned in (Schwarting et al., 2017), we estimate ego's bounding box using four disks denoted as c_ℓ, $\ell \in \{1, .., 4\}$, each possessing a radius of r_e (see Fig. 6·2.(b)). Details of computing the radius r_e and the center of the disks are provided in App. C.

We can now assess the fulfillment of the collision avoidance rule by employing the Minkowski sum between ego's disks and the encompassing ellipses of other cars. In

other words, the predicate (*collision_avoidance*) holds true when:

$$\begin{gathered}\forall i \in I_{near,\,t},\ \forall \ell \in \{1, ..., 4\}: \\ \begin{bmatrix} x_{i\ell,t} \\ y_{i\ell,t} \end{bmatrix}^{\top} Rot(\psi_{i,t})^{\top} \begin{bmatrix} \frac{1}{a_t^2} & 0 \\ 0 & \frac{1}{b_t^2} \end{bmatrix} Rot(\psi_{i,t}) \begin{bmatrix} x_{i\ell,t} \\ y_{i\ell,t} \end{bmatrix} > 1,\end{gathered} \tag{6.6}$$

where $(x_{i\ell,t}, y_{i\ell,t})$ represent the relative distance between centers of encompassing ellipse of i^{th} nearby vehicle and the ℓ^{th} disk of ego. The variable $Rot(\psi_{i,t})$ corresponds to the rotation matrix of the heading of i^{th} nearby vehicle. Additionally, we have $a_t = a_{i,t}^u + r_e$ and $b_t = b_{i,t}^u + r_e$. Since we have conservatively estimated the bounding boxes of ego ($\mathcal{BB}_t \subset \bigcup_{\ell=1}^4 c_\ell$) and other vehicles ($\mathcal{BB}_{i,t}(\zeta_{i,t}, \sigma_{i,t}, \epsilon) \subset E(a_{i,t}, b_{i,t})$), Inequality (6.6) guarantees collision avoidance between ego and other vehicles with a probability exceeding ϵ.

6.6.2 Correction Constraints

Utilizing the detailed dynamic model of ego and the detailed representation of the environment, we conduct a forward simulation of ego's trajectory over the MPC horizon. This simulation employs the optimal control output from the top-level of our approach. Should the trajectory of ego adhere to all traffic regulations, the first element of the control sequence is applied on ego.

Conversely, if any traffic regulation is violated, we define the set of violated rules at time t as $\Phi_{violated,\,t} \subseteq \Phi_{active,\,t}$, with the finite B_t number of rules: $\Phi_{violated,t} = \{\phi_b\}_{b=1}^{B_t}$. For each $\phi_b \in \Phi_{violated,\,t}$, we generate a set of MILCs $\mathcal{Q}_b$ as the correction constraints. These constraints are aimed to ensure the satisfaction of violated rule ϕ_b and impose correctness into ego's motion. We obtain the correction constraints by translating the STL formulae of violated rules, similar to the technique in Sec. 5.5.3, with the main difference that their predicates are evaluated using the linearized version of detailed dynamic model of ego and the detailed collision avoidance requirement (6.6).

We concisely represent the necessary correction constraints $\mathcal{Q}_b$ for the fulfillment of violated rule ϕ_b as $z_{\phi_b} = 1$.

The collection of correction constraints at time t is symbolized as $\mathcal{Q}_t = \{\mathcal{Q}_b\}_{b=1}^{B_t}$ and is passed to the top-level of our control structure, where it is applied to the MPC problem to be solved again. The formulation for the updated MPC problem can be articulated as follows:

$$
\begin{aligned}
u^*_{t:\,t+H-1} = \underset{u_{t:\,t+H-1}}{\arg\min} & \sum_{k=t}^{t+H-1} \mathcal{F}_{cost,\,k}(u_t, u_{t-1}, e_t) \\
\text{s.t.} \quad & \forall k \in \{t, ..., t+H-1\}: \\
& \zeta_{k+1} = f_s(\zeta_k, u_k), \\
& z_{\phi_a} = 1, \quad \forall\, \phi_a \in \Phi_{active,\,k}, \\
& \zeta_{min} \leq \zeta_k \leq \zeta_{max}, \\
& u_{min} \leq u_k \leq u_{max}, \\
& z_{\phi_b} = 1, \quad \forall\, \phi_b \in \Phi_{violated,\,k}
\end{aligned}
\tag{6.7}
$$

We solve the MPC problem in (6.7) and verify its correctness at the bottom-level using the detailed models. If its optimal solution violates any traffic rules, a backup controller enforces full-braking on ego.

Discussion 2 *In urban driving situations, according to regulations recommended by law, backup controllers are advised to reduce speed and perform an emergency stop in the majority of cases. This approach is favored because: 1) minimizing the velocity of ego to nearly zero reduces the impact force and potential harm in the event of a collision, and 2) attempting to predict the optimal control response during an emergency could be time-consuming and may not yield a practical solution due to the limited time available for reaction. Therefore, our backup controller opts to fully apply the brakes to ego, as this is the most rational action for urban driving scenarios. While this chapter does not delve into alternative backup controllers, various alternate strategies for autonomous vehicles during emergencies can be explored in (Isele, 2019; Cai et al., 2020).*

6.7 Case Studies

The validation of our algorithm's efficacy and runtime performance is demonstrated by conducting simulations based on three real-world scenarios:

(i) **General Performance**: presents an overview of our method's overall effectiveness within a typical urban driving setting. It illustrates the method's capability to adhere to the reference path, abide by traffic regulations, and maintain real-time performance.

(ii) **Comparison with Deterministic Approach**: we assess the effectiveness of our technique in contrast to the deterministic solution presented in Chapter 5. We remark on the proficiency of our stochastic method in managing the uncertainty associated with estimating the trajectories of other vehicles.

(iii) **Comparison with NMPC Baseline**: Since numerous pieces of literature rely on NMPC formulation, we execute a basic NMPC approach in our final case study. This allows us to establish a baseline and compare its performance with our framework. The objective is to highlight the superior solution quality and runtime efficiency of our approach in comparison to an NMPC method.

In all scenarios, the primary objective of ego is to closely adhere to the provided reference path, while simultaneously adhering to the active traffic rules at each time step. In the schematics of the case studies (Fig. 6·3, Fig. 6·5, and Fig. 6·7) the reference path is illustrated by a red line. This path originates from ego's initial position (marked as S) and extends to the ego's destination point (labeled as G). To enhance clarity in visualization, we depict the reference path by interpolating between its designated reference points. Across all scenarios, the environment accommodates a total of 100 other vehicles, and their positions are arbitrarily depicted. In all scenarios,

we employ the hyperparameters outlined in Table 6.1, which have been fine-tuned through empirical results. We set the duration of a time step $\Delta t = 0.1\,s\,(10\,Hz)$, which is a standard rate for low-speed urban driving.

Table 6.1: Hyperparameters used in the simulation scenarios of our stochastic control approach

Parameter	Value	Parameter	Value	Parameter	Value
H	10	Δt	$0.1\,s$	l_f	$2.1\,m$
r_{near}	$10\,m$	d_{safe}	$4.5\,m$	ϵ	0.95
σ_a	$0.3\,m$	σ_b	$0.1\,m$		

We implement the scenarios in the simulator CARLA (Dosovitskiy et al., 2017), and we employ Gurobi package (Gurobi Optimization, 2020) for solving the MPC optimization problem. We run all the simulations on a $3.70\,GHz$ processor with $16\,GB$ RAM. In order to assess the real-time performance of our algorithm and the baseline methods, we use the system clock frequency, which signifies how often the control loop and environment are updated. This parameter is denoted as Frames Per Second (Hz) in our results.

6.7.1 General Performance

This scenario highlights the efficacy of our approach in adhering to traffic rules and closely tracking the given reference path. Ego must accurately follow the reference path while minimizing errors and simultaneously obeying speed limit, traffic light, and collision avoidance rules. The schematic of the scenario is shown in Fig. 6·3, which shows ego has to change its lane three times along the reference path.

The results of our method's execution in this scenario are depicted in Fig. 6·4. These results encompass several aspects, including the runtime speed as illustrated in Fig. 6·4.(a), the speed profile of ego and road speed limits visualized in Fig. 6·4.(b),

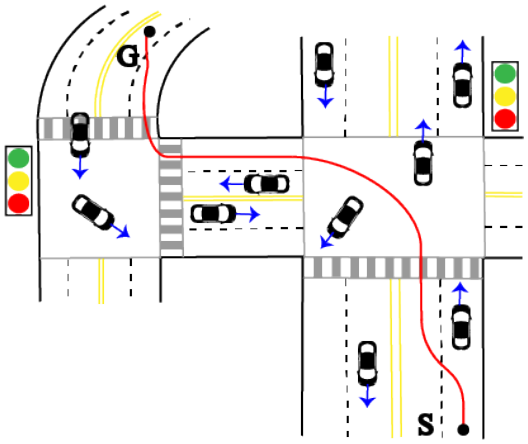
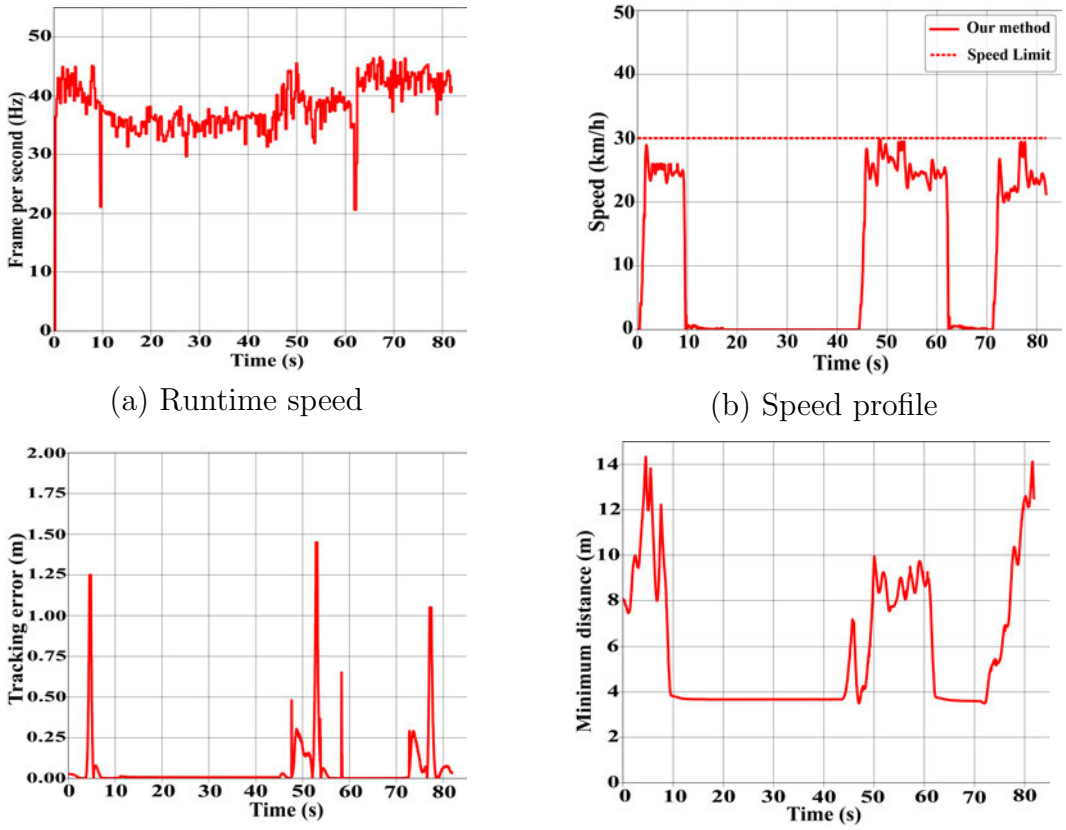


Figure 6·3: Schematic of the general performance scenario.

the tracking error with respect to reference path demonstrated in Fig. 6·4.(c), and the safety performance denoted by the minimum distance maintained from nearby vehicles, presented in Fig. 6·4.(d). Throughout the scenario, ego undertakes three lane changes, occurring at time instances $t = 5\,s$, $t = 53\,s$, and $t = 78\,s$, all throughout the reference path. Additionally, ego comes to a stop behind the initial red traffic light from $t = 10\,s$ to $t = 44\,s$, and subsequently halts behind the second red light from $t = 62\,s$ to $t = 72\,s$.

The primary slowdowns in the runtime speed of our method depicted in Fig. 6·4.(a) occur during instances when ego approaches another vehicle ($t = 10\,s$) and halts at a red light ($t = 62\,s$). This observation is also evident from Fig. 6·4.(d). Additionally, the bottom-level of our controller is engaged for a brief duration during these times to manage the uncertainty associated with predicting the trajectory of the other car. Conversely, the peaks in the tracking error profile of ego in Fig. 6·4.(c) ($t = 5\,s, t = 53\,s$, and $t = 78\,s$) coincide with moments when ego changes lanes to closely follow the reference path, while adhering to the lane's speed limit. This lane change leads to a brief spike in the tracking error.

In this scenario, the bottom-level of our algorithm is activated 14 times, with the



(a) Runtime speed

(b) Speed profile

(c) Tracking error

(d) Minimum distance from nearby vehicles

Figure 6·4: Results from running our method in the general performance scenario.

backup controller never being engaged. This indicates that in all these instances, the bottom-level component effectively enhances the accuracy of ego's actions. The data presented in Fig.6·4 illustrates our algorithm's promising real-time capabilities, maintaining an average runtime speed of $30 \sim 40\,Hz$ (as depicted in Fig.6·4.(a)). Furthermore, it consistently adheres to legal speed limits (see Fig.6·4.(b)), demonstrating minimal average tracking errors (as shown in Fig.6·4.(c)), and consistently ensures a safe distance from nearby vehicles (refer to Fig. 6·4.(d)).

6.7.2 Comparison with Deterministic Approach

In this scenario, we compare the effectiveness of our stochastic technique with the deterministic methodology outlined in Chapter 5. We underscore the significance of the bottom-level of our stochastic approach in managing uncertainty. The primary objective of ego is this scenario is to follow the reference path with minimum error while satisfying a stop sign, traffic light, speed limit, and collision avoidance rules (Fig. 6·5).

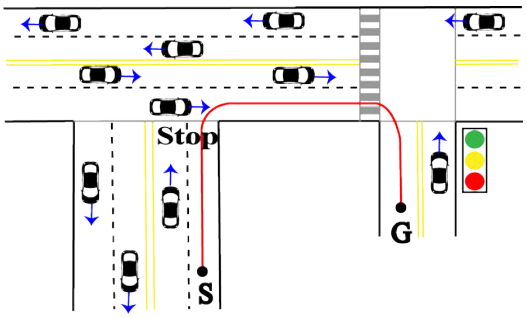
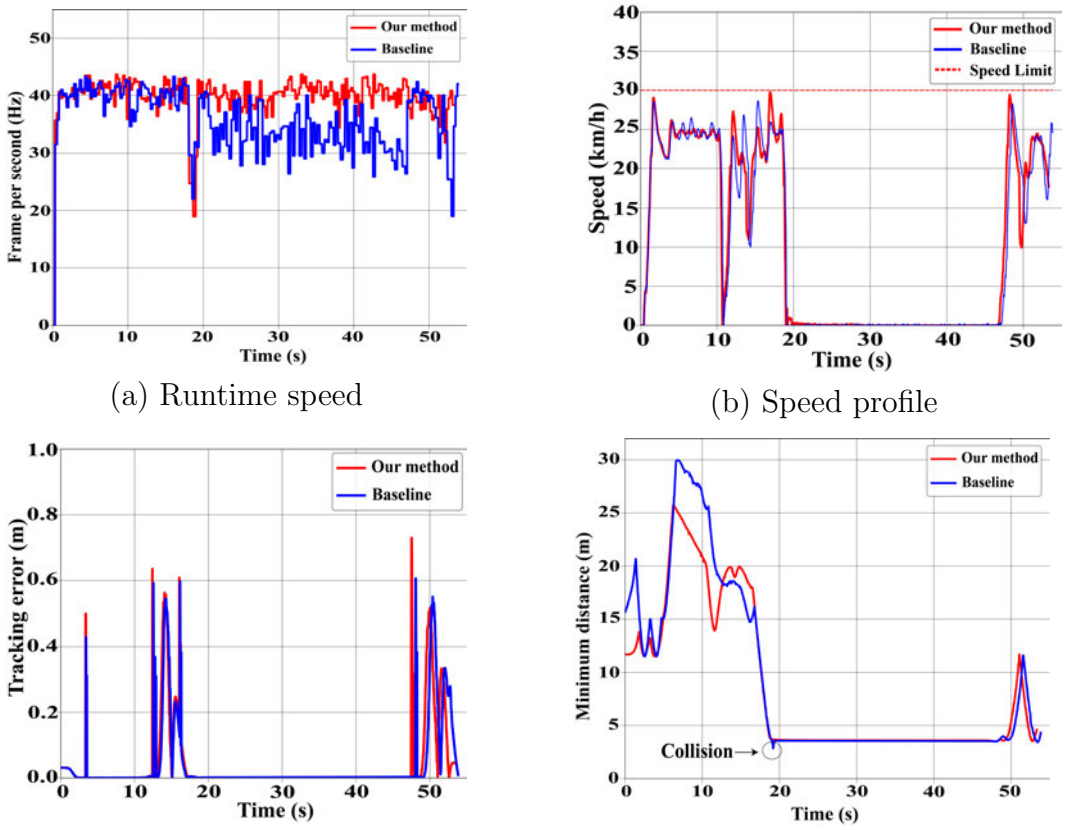


Figure 6·5: Schematic of the scenario for comparison with our deterministic approach.

In Fig. 6·6, we display the comprehensive performance of our framework, compared with the deterministic approach. The runtime speed of the methods is depicted in Fig. 6·6.(a), while Fig. 6·6.(b) illustrates both the speed profiles and the scenario's speed limit. The tracking error with respect to the reference path is visualized in Fig. 6·6.(c), whereas Fig. 6·6.(d) showcases the minimum distance between ego and nearby cars. In both simulations, ego reaches the stop sign at $t = 10\,s$, and subsequently at $t = 18\,s$ gets close to another vehicle ahead of it, which has stopped at the red light. As indicated in Fig. 6·6.(b), ego momentarily halts behind the stop sign at $t = 10\,s$, and then remains stationary behind the preceding vehicle at the red light from $t = 20\,s$ until the light turns green at $t = 47\,s$.

The data presented in Fig. 6·6 illustrates the similarity between the speed profile and tracking error of both our stochastic method and the deterministic baseline.



(a) Runtime speed

(b) Speed profile

(c) Tracking error

(d) Minimum distance from nearby vehicles

Figure 6·6: Results from comparing our stochastic approach with the deterministic baseline from Chapter 5.

However, our stochastic approach has a faster runtime speed on average and maintains a safe distance from nearby vehicles. In contrast, the deterministic baseline leads to a collision between ego and the preceding car at time $t = 18\,s$. In this scenario, our stochastic framework's bottom-level is invoked five times, imposing correction constraints on the MPC problem in top-level, while the backup controller remains unused. On the other side, the deterministic baseline's low-level controller is activated four times, successfully identifying near-optimal control solutions, but its backup controller is triggered 12 times, indicating the lower-level controller's failure to ascertain near-optimal control during those instances, compelling the backup controller to engage the emergency brake. The deterministic baseline's backup controller

comes into play predominantly as ego approaches the red light while getting close to the other vehicle ahead of it, exposing the low-level controller's inability to handle the uncertainty in predicting the trajectory of other vehicles and resulting in collision. This is also evident in the runtime speed depicted in Fig. 6·6.(a), where the baseline's runtime speed dips below that of our stochastic method between time points $20\,s$ and $47\,s$. This disparity arises because the deterministic baseline's low-level controller explores the control space randomly, failing to find the collision-avoiding near-optimal controls. To summarize, our stochastic approach offers faster runtime speed and safe behavior in contrast to the deterministic baseline from Chapter 5.

6.7.3 Comparison with NMPC Baseline

In this scenario, we demonstrate the merits of our framework in comparison to the existing research on NMPC. We achieve this by evaluating the efficacy of our proposed controller against an NMPC baseline. The foundation of this baseline involves utilizing the extended bicycle model (Sec. 5.6.1) to characterize ego dynamics, the detailed representation of the environment (Sec. 6.6.1), and integrating constraints to satisfy traffic rules. In this scenario, the primary objective of ego is to closely adhering to the reference path and ensuring compliance with speed limit, traffic light, and collision avoidance rules (Fig. 6·7). There are three different speed limits along the reference path.

Fig. 6·8 illustrates a comparative analysis between our controller's performance and the NMPC baseline. This comparison encompasses various aspects: the runtime speed of the approaches (depicted in Fig. 6·8.(a)), speed profiles along with speed limits of the roads (shown in Fig. 6·8.(b)), tracking error with respect to reference path (displayed in Fig. 6·8.(c)), and the minimum distance from nearby vehicles (represented in Fig. 6·8.(d)). In both simulations, ego reaches the initial speed limit of $60\,km/h$ at $t = 10\,s$, the subsequent limit of $30\,km/h$ at $t = 18\,s$, and the third

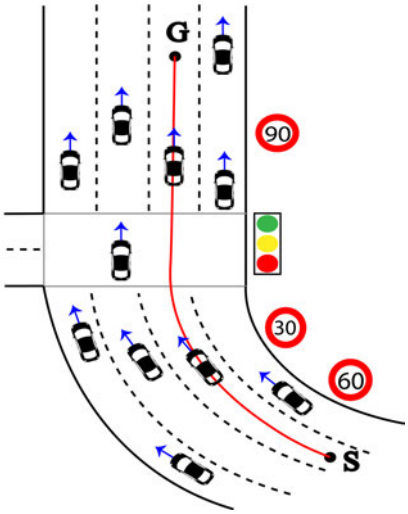
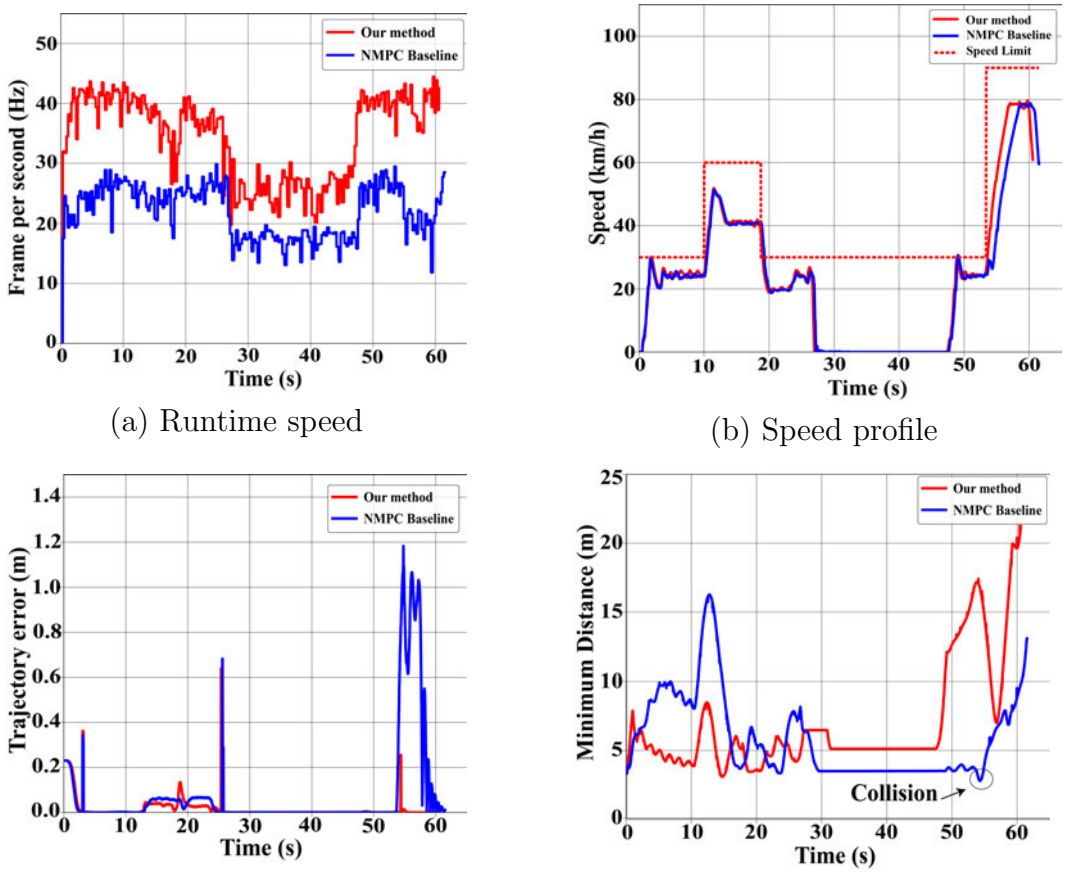


Figure 6·7: Schematic of the scenario for comparison with the NMPC baseline.

speed limit of $90\,km/h$ around $t \approx 54\,s$.

By comparing the simulation results, it is evident that our approach and the NMPC baseline exhibit similar velocity profiles (Fig. 6·8.(b)). However, our framework achieves faster runtime speeds than the baseline (Fig. 6·8.(a)) and maintains a lower tracking error on average (Fig.6·8.(c)). Notably, our controller consistently maintains a safe distance from nearby vehicles. Conversely, the NMPC baseline leads to a collision between ego and another car in the adjacent lane at $t = 54\,s$ (Fig. 6·8.(d)). The collision happens when ego tries to increase its speed to drive faster, with respect to the $90\,km/h$ speed limit of the lane, and manage the uncertainty in predicting the trajectory of the nearby vehicle. However, it fails to find an optimal control strategy that satisfies both requirements. Throughout this scenario, our controller's bottom-level is activated eight times, added correction constraints to the MPC problem in the top-level, and the backup controller remains inactive. Overall, our approach outperforms the NMPC baseline in terms of faster runtime performance and superior solution quality.



(a) Runtime speed

(b) Speed profile

(c) Tracking error

(d) Minimum distance from nearby vehicles

Figure 6·8: Comparing the performance of our method with the NMPC baseline.

6.8 Summary

In this chapter, we presented a two-level control framework designed for autonomous vehicles navigating through urban driving environments. At the top-level of our approach, we formulate an MIQP optimization problem using a simplified dynamic model for ego and a simplified environment model. It incorporates MILCs based on active traffic regulations. Meanwhile, the bottom-level of our framework assesses ego's trajectory based on the optimal control output from the top-level. This assessment is carried out in relation to more detailed models for ego's dynamics and its environment.

In instances where traffic rules are violated, correction constraints are imposed to the top-level MPC problem. These constraints enhance the accuracy of ego's performance. Through simulations within three urban driving scenarios, we demonstrated that our algorithm achieves fast runtime speed and exhibits promising performance, compared with two baselines from the literature.

Chapter 7

Conclusions

7.1 Summary of the Dissertation

In this thesis, our emphasis was on employing temporal logic for inference and control of autonomous systems. The selection of temporal logic formulae stems from their capacity to enhance interpretability and readability in addressing these subjects. More precisely, we utilized STL specifications to express the temporal characteristics of real-valued signals within dynamic systems.

In the first part of the thesis, we explored the inference of temporal properties of dynamic systems using STL specifications, within an offline learning framework. In this framework, which shares similarities with traditional offline learning problems in machine learning, we are given a dataset containing signals and their corresponding labels. We specifically focused on binary classification problems, where the labels can be either positive or negative. The primary goal is to develop an interpretable classifier that can accurately classify signals. Our contribution involves the introduction of the BCDT algorithm. This algorithm leverages the boosting technique, commonly employed in machine learning to establish a classifier with strong classification performance, built from an ensemble of models with weak classification performance. The weak classifiers of our algorithm are CDTs, which are decision trees that are empowered by conciseness techniques, to generate simpler STL formulae. We established connections between CDTs and STL formulae, as well as between the BCDT approach and wSTL formulae. These connections serve to incorporate interpretability into our

framework. Through the use of two illustrative case studies, we demonstrated the classification efficacy and interpretability of our BCDT algorithm. Furthermore, we showcased its superiority over two existing methods in the literature.

In the second part of the thesis, inspired by the offline learning problem in the first part, we presented the concept of time-incremental learning framework. This scenario involves a dataset comprising signals and corresponding labels, similar to the offline learning framework. However, the objective here is to develop a predictor capable of forecasting labels for prefix signals over time. Our proposed algorithm is structured into three key components. The initial phase, Signal Analysis, delves into the temporal properties of the signals, identifying a set of decision times that are potentially informative for effectively classifying prefix signals. Subsequently, in the Classifier Learning phase, we construct decision trees for each of these decision times, drawing upon insights from the CDT method used in the offline learning problem. Finally, the Classifier Evaluation phase assigns time-variant weight distributions to the set of classifiers, resulting in developing the predictor. To validate the effectiveness of our algorithm, we conducted two case studies, and compared its performance against two baseline methods.

In the next part of the dissertation, our attention shifted from inference problems to the control of autonomous systems, particularly focusing on autonomous vehicles. In this part, we explored the problem of guiding a self-driving car, referred to as ego, along a given path within a deterministic environment. This involves adhering to traffic rules and fulfilling real-time performance criteria. To tackle this problem, we introduced a two-level control architecture. The high-level controller formulates an MPC problem, utilizing simplified models to represent ego's dynamics and its surrounding environment. Additionally, it translates traffic rules into MILCs and impose them to the MPC problem. The MPC's optimal control output is then utilized

by the low-level controller to simulate ego's trajectory, using detailed models for ego's dynamics and its environment. We employed runtime monitoring techniques to assess compliance with traffic rules within the detailed framework, making necessary adjustments to the control inputs as needed. Through the analysis of four urban-driving simulation scenarios, we demonstrated the quality of our solution and its runtime performance, showcasing its superiority over an NMPC baseline.

In the last part of the thesis, we extended the problem setting in the previous part from deterministic environments to stochastic ones, particularly addressing uncertainty in predicting the trajectories of other vehicles using a posterior Gaussian distribution. We presented a two-level control algorithm for driving ego along the given reference path, while adhering to traffic rules and runtime performance requirements. In the top-level of our approach, we ignored the uncertainty in estimating the states of other vehicles, and followed the same structure as the high-level controller in our deterministic framework: used simplified models and MILC translations of traffic rules to formulate an MIQP problem. In the bottom-level of our approach, we forward-simulated ego's trajectory, under the optimal control of top-level MPC, and evaluated its correctness with respect to detailed models. In this level, we considered the effect of uncertainty in evaluating the satisfaction of traffic rules. In the case of violation of any rule, a set of MILC correction constraints were extracted and imposed on top-level MPC problem to be solved again. Our algorithm's performance was assessed against our deterministic framework and an NMPC baseline, in various urban-driving simulations.

7.2 Future Directions

The ideas introduced in this thesis can be extended in multiple directions. Here we discuss some of the areas that can be further investigated:

1. Regarding offline inference of STL specifications, potential future research could explore alternative methods for balancing the simplicity of the resulting STL formulae with their classification performance, while also aiming for faster runtime speeds. Another area open for enhancement involves examining the inference of STL formulae from signals with varying time lengths, thereby increasing the applicability of the inference framework.

2. Regarding the time-incremental inference of STL formulae, potential directions for future investigation involve delving into advanced signal analysis methods to identify decision times. Another area of improvement is the examination of the efficacy of alternative machine learning approaches, such as recurrent neural networks and transformers, and comparing their performance with the decision tree approach.

3. With respect to the control algorithms for autonomous vehicles in deterministic environments, there is potential for upcoming research to explore broadening the range of traffic rules described by STL specifications. Additionally, one may investigate the effect of including the robustness of satisfying the STL traffic rules in formulating the MPC problem.

4. Regarding the controlling of autonomous vehicles in stochastic environments, future research could explore the impact of other sources of uncertainty, such as uncertainty in detecting traffic signs, and strategies for addressing these challenges within the control framework. Additionally, one could delve into scenarios involving various driving behaviors exhibited by human operators of other vehicles, aiming to develop methods for ensuring safe motion in relation to these diverse driving models.

Appendix A

Linearized Bicycle Model

The continuous-time formulation of a general dynamic model can be considered as:

$$\dot{\zeta} = f(\zeta, u) \tag{A.1}$$

Hence, for the continuous-time formulation of kinematic bicycle model we have:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v\cos\psi \\ v\sin\psi \\ \frac{v}{l_f}\tan\delta \\ \beta \end{bmatrix} \tag{A.2}$$

In order to linearize a general dynamic model as (A.1), we can use the Taylor expansion (Abramowitz, 1965) about the operating points $\bar{\zeta}$ and $\bar{u}$, which leads to:

$$\dot{\zeta} \approx f(\bar{\zeta}, \bar{u}) + \frac{\partial f(\zeta, u)}{\partial \zeta}\Big|_{\bar{\zeta},\bar{u}}(\zeta - \bar{\zeta}) + \frac{\partial f(\zeta, u)}{\partial u}\Big|_{\bar{\zeta},\bar{u}}(u - \bar{u}) \tag{A.3}$$

By applying the Taylor expansion (A.3) to kinematic bicycle model (A.2) we have:

$$\dot{\zeta} = \begin{bmatrix} \bar{v}\cos\bar{\psi} \\ \bar{v}\sin\bar{\psi} \\ \frac{\bar{v}}{l_f}\tan\bar{\delta} \\ \bar{\beta} \end{bmatrix} + \begin{bmatrix} 0 & 0 & -\bar{v}\sin\bar{\psi} & \cos\bar{\psi} \\ 0 & 0 & \bar{v}\cos\bar{\psi} & \sin\bar{\psi} \\ 0 & 0 & 0 & \frac{\tan\bar{\delta}}{l_f} \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x - \bar{x} \\ y - \bar{y} \\ \psi - \bar{\psi} \\ v - \bar{v} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{\bar{v}}{l_f\cos^2\bar{\delta}} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \delta - \bar{\delta} \\ \beta - \bar{\beta} \end{bmatrix}$$

which can be simplified to:

$$\dot{\zeta} = \begin{bmatrix} v\cos\bar{\psi} - \bar{v}\sin\bar{\psi}\psi + \bar{v}\sin\bar{\psi}\bar{\psi} \\ v\sin\bar{\psi} + \bar{v}\cos\bar{\psi}\psi - \bar{v}\cos\bar{\psi}\bar{\psi} \\ \frac{v}{l_f}\tan\bar{\delta} + \frac{\bar{v}}{l_f\cos^2\bar{\delta}}\delta - \frac{\bar{v}}{l_f\cos^2\bar{\delta}}\bar{\delta} \\ \beta \end{bmatrix} \tag{A.4}$$

Appendix B

Extended Bicycle Model

In the extended bicycle model (see Fig. B·1), the equations of motion are described as (Turri et al., 2013):

$$\begin{aligned} m\ddot{x'} &= F_{x_{f1}} + F_{x_{f2}} + F_{x_{r1}} + F_{x_{r2}} + m\dot{y'}\dot{\psi}, \\ m\ddot{y'} &= F_{y_{f1}} + F_{y_{f2}} + F_{y_{r1}} + F_{y_{r2}} - m\dot{x'}\dot{\psi}, \\ \mathcal{I}\ddot{\psi} &= l_f(F_{y_{f1}} + F_{y_{f2}}) - l_r(F_{y_{r1}} + F_{y_{r2}}), \end{aligned} \tag{B.1}$$

where m, $\mathcal{I}$ and l_r are the mass, mass moment of inertia, and distance from center of mass to rear axle of the vehicle, $\dot{x'}$ and $\dot{y'}$ are the longitudinal and lateral speed of the vehicle, the subscripts $()_{f1}, ()_{f2}, ()_{r1}$, and $()_{r2}$ are used to refer to the front-right, front-left, rear-right and rear-left tires, and F_x and F_y are the tire forces acting along the vehicle's longitudinal and lateral axes relative to each wheel, respectively.

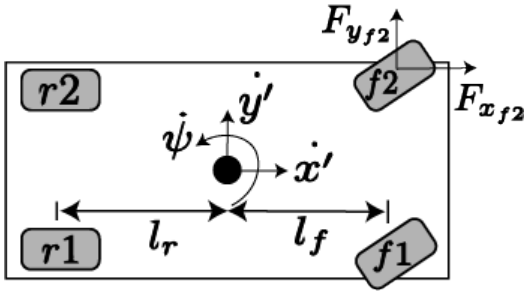


Figure B·1: Representation of the extended bicycle model.

For the longitudinal and lateral tire forces we have:

$$\begin{aligned} F_{x_{f\star}} &= f_{x_{f\star}} \cos(\delta) - f_{y_{f\star}} \sin(\delta), \\ F_{y_{f\star}} &= f_{x_{f\star}} \sin(\delta) + f_{y_{f\star}} \cos(\delta), \\ F_{x_{r\star}} &= f_{x_{r\star}}, \quad F_{y_{r\star}} = f_{y_{r\star}}, \end{aligned} \tag{B.2}$$

where f_x and f_y are the forces acting along the wheel longitudinal and lateral axes, respectively, and $\star \in \{1, 2\}$. By following the same assumptions from (Turri et al., 2013), we have:

$$f_{x_{\bullet\star}} = F_{z_{\bullet\star}}\, \kappa\, \gamma, \quad f_{y_{\bullet\star}} = \kappa\, F_{z_{\bullet\star}} \sqrt{1-\gamma^2}\, \sin(\xi\, \arctan \tau_\bullet\, \alpha_\bullet), \tag{B.3}$$

where $\bullet \in \{f, r\}, \star \in \{1, 2\}$, F_z is the force acting along the vertical axis of each wheel, κ is the friction coefficient between the tires and the road surface, ξ and τ are tire parameters that are calibrated using experimental data.

In (B.3), α is the slip angle of the wheels, which is computed as:

$$\alpha_f = \frac{\dot{y} + l_f\, \dot{\psi}}{\dot{x}} - \delta, \quad \alpha_r = \frac{\dot{y} - l_r\, \dot{\psi}}{\dot{x}} \tag{B.4}$$

Moreover, the distribution of vertical forces are considered as:

$$\begin{aligned} F_{z_{f1}} &= \frac{l_r\, F_l - h_c\, F_x}{2\,(l_f + l_r)} - \frac{h_c\, F_y}{2\, w_c}, \quad F_{z_{f2}} = \frac{l_r\, F_l - h_c\, F_x}{2\,(l_f + l_r)} + \frac{h_c\, F_y}{2\, w_c}, \\ F_{z_{r1}} &= \frac{l_f\, F_l + h_c\, F_x}{2\,(l_f + l_r)} - \frac{h_c\, F_y}{2\, w_c}, \quad F_{z_{r2}} = \frac{l_f\, F_l + h_c\, F_x}{2\,(l_f + l_r)} + \frac{h_c\, F_y}{2\, w_c}, \end{aligned} \tag{B.5}$$

where the gravity force applied on the vehicle is shown by $F_l = m\, g$ (g is the gravitational acceleration), the width and height of the vehicle's center of gravity are denoted by w_c and h_c, and the cumulative longitudinal and lateral forces are shown by $F_x = \sum_{\bullet\in\{f,r\},\star\in\{1,2\}} F_{x_{\bullet\star}}$ and $F_y = \sum_{\bullet\in\{f,r\},\star\in\{1,2\}} F_{y_{\bullet\star}}$, respectively.

It is important to note that the longitudinal and lateral speed of ego are related to

its velocity components in the absolute coordinate frame of the environment, through:

$$\dot{x} = \dot{x'}\cos\psi - \dot{y'}\sin\psi, \quad \dot{y} = \dot{x'}\sin\psi + \dot{y'}\cos\psi.$$

The hyperparameters of the extended bicycle model are tuned empirically in our implementations and they are provided in Table B.1.

Table B.1: Hyperparameters of the extended bicycle model

Parameter	Value	Parameter	Value	Parameter	Value
m	$1200.0\,kg$	$l_r = l_f$	$2.1\,m$	κ	0.95
$\mathcal{I}$	$2100.0\,kg \cdot m^2$	g	$9.8\,m/s^2$	ξ	0.5
τ_f, τ_r	$-10.5, -12.7$	w_c	$1.8\,m$	h_c	$0.7\,m$

Appendix C

Estimating Bounding Box of Ego

By using 4 disks to estimate the bounding box of ego (see Fig. 6·2.(b)), it is easy to show that the radius r_e of the disks can be computed as

$$r_e = \sqrt{\frac{l_f^2 + 4w_c^2}{16}} \tag{C.1}$$

Moreover, by having the position (x_t, y_t) and heading ψ_t of ego at time t, we can compute the position of the center of the disks c_ℓ, $\ell \in \{1, ..., 4\}$ by:

$$\begin{aligned}
x_{c_1,t} &= x_t + \frac{3l_f}{4}\cos\psi_t, \quad & y_{c_1,t} &= y_t + \frac{3l_f}{4}\sin\psi_t \\
x_{c_2,t} &= x_t + \frac{l_f}{4}\cos\psi_t, \quad & y_{c_2,t} &= y_t + \frac{l_f}{4}\sin\psi_t \\
x_{c_3,t} &= x_t - \frac{l_f}{4}\cos\psi_t, \quad & y_{c_3,t} &= y_t - \frac{l_f}{4}\sin\psi_t \\
x_{c_4,t} &= x_t - \frac{3l_f}{4}\cos\psi_t, \quad & y_{c_4,t} &= y_t - \frac{3l_f}{4}\sin\psi_t
\end{aligned} \tag{C.2}$$

References

Aasi, E. (2022). Unmarked crosswalk carla scenario. `https://github.com/erfanaasi/unmarked_crosswalk_carla_scenario`. [Online; accessed 1-March-2022].

Aasi, E., Cai, M., Vasile, C. I., and Belta, C. (2023). Time-incremental learning of temporal logic classifiers using decision trees. In *Learning for Dynamics and Control Conference*, pages 547–559. PMLR.

Aasi, E., Vasile, C. I., Bahreinian, M., and Belta, C. (2022). Classification of time-series data using boosted decision trees. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1263–1268. IEEE.

Aasi, E., Vasile, C. I., and Belta, C. (2021). A control architecture for provably-correct autonomous driving. In *2021 American Control Conference (ACC)*, pages 2913–2918. IEEE.

Abramowitz, M. (1965). *Handbook of mathematical functions, with formulas, graphs, and mathematical tables,*. Dover Publications, New York.

Albrecht, S. V., Brewitt, C., Wilhelm, J., Gyevnar, B., Eiras, F., Dobre, M., and Ramamoorthy, S. (2021). Interpretable goal-based prediction and planning for autonomous driving. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1043–1049. IEEE.

Anderson, S. J., Peters, S. C., Pilutti, T. E., and Iagnemma, K. (2010). An optimal-control-based framework for trajectory planning, threat assessment, and semi-autonomous control of passenger vehicles in hazard avoidance scenarios. *International Journal of Vehicle Autonomous Systems*, 8(2-4):190–216.

Asarin, E., Donzé, A., Maler, O., and Nickovic, D. (2011). Parametric identification of temporal properties. In *International Conference on Runtime Verification*, pages 147–160. Springer.

Baharisangari, N., Gaglione, J.-R., Neider, D., Topcu, U., and Xu, Z. (2021). Uncertainty-aware signal temporal logic. *arXiv preprint arXiv:2105.11545*.

Baier, C. and Katoen, J.-P. (2008). *Principles of model checking*. MIT press.

Bakhirkin, A., Ferrère, T., and Maler, O. (2018). Efficient parametric identification for stl. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*, pages 177–186.

Bartocci, E., Bortolussi, L., Nenzi, L., and Sanguinetti, G. (2015). System design of stochastic models using robustness of temporal properties. *Theoretical Computer Science*, 587:3–25.

Bartocci, E., Bortolussi, L., and Sanguinetti, G. (2014). Data-driven statistical learning of temporal logic properties. In *International conference on formal modeling and analysis of timed systems*, pages 23–37. Springer.

Bartocci, E., Deshmukh, J., Donzé, A., Fainekos, G., Maler, O., Ničković, D., and Sankaranarayanan, S. (2018). Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications. *Lectures on Runtime Verification: Introductory and Advanced Topics*, pages 135–175.

Belta, C., Yordanov, B., and Gol, E. A. (2017). *Formal methods for discrete-time dynamical systems*, volume 89. Springer.

Bombara, G. and Belta, C. (2017). Signal clustering using temporal logics. In *International Conference on Runtime Verification*, pages 121–137. Springer.

Bombara, G. and Belta, C. (2021). Offline and online learning of signal temporal logic formulae using decision trees. *ACM Transactions on Cyber-Physical Systems*, 5(3):1–23.

Bombara, G., Vasile, C.-I., Penedo, F., Yasuoka, H., and Belta, C. (2016). A decision tree approach to data classification using signal temporal logic. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, pages 1–10.

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and regression trees*. CRC press.

Brüdigam, T., Olbrich, M., Wollherr, D., and Leibold, M. (2023). Stochastic model predictive control with a safety guarantee for automated driving. *IEEE Transactions on Intelligent Vehicles*, 8(1):22–36.

Bufo, S., Bartocci, E., Sanguinetti, G., Borelli, M., Lucangelo, U., and Bortolussi, L. (2014). Temporal logic based monitoring of assisted ventilation in intensive care patients. In *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications: 6th International Symposium, ISoLA 2014, Imperial, Corfu, Greece, October 8-11, 2014, Proceedings, Part II 6*, pages 391–403. Springer.

Cai, K. X., Phan-Minh, T., Chung, S.-J., and Murray, R. M. (2020). Rules of the road: Safety and liveness guarantees for autonomous vehicles. *arXiv preprint arXiv:2011.14148*.

Camacho, E. F. and Alba, C. B. (2013). *Model predictive control*. Springer Science & Business Media.

Carvalho, A., Gao, Y., Lefevre, S., and Borrelli, F. (2014). Stochastic predictive control of autonomous vehicles in uncertain environments. In *12th International Symposium on Advanced Vehicle Control*, pages 712–719.

Censi, A., Slutsky, K., Wongpiromsarn, T., Yershov, D., Pendleton, S., Fu, J., and Frazzoli, E. (2019). Liability, ethics, and culture-aware behavior specification using rulebooks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8536–8542. IEEE.

Clarke, E. M. (1997). Model checking. In *Foundations of Software Technology and Theoretical Computer Science: 17th Conference Kharagpur, India, December 18–20, 1997 Proceedings 17*, pages 54–56. Springer.

Clarke, E. M., Emerson, E. A., and Sistla, A. P. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*. MIT press.

Cuturi, M. and Blondel, M. (2017). Soft-dtw: a differentiable loss function for time-series. In *International Conference on Machine Learning*, pages 894–903. PMLR.

Deshmukh, J. V., Donzé, A., Ghosh, S., Jin, X., Juniwal, G., and Seshia, S. A. (2017). Robust online monitoring of signal temporal logic. *Formal Methods in System Design*, 51(1):5–30.

Ding, X., Lazar, M., and Belta, C. (2014). Ltl receding horizon control for finite deterministic systems. *Automatica*, 50(2):399–408.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). Carla: An open urban driving simulator. *arXiv preprint:1711.03938*.

Eiras, F., Hawasly, M., Albrecht, S. V., and Ramamoorthy, S. (2021). A two-stage optimization-based motion planner for safe urban driving. *IEEE Transactions on Robotics*, 38(2):822–834.

Erlien, S. M., Fujita, S., and Gerdes, J. C. (2015). Shared steering control using safe envelopes for obstacle avoidance and vehicle stability. *IEEE Transactions on Intelligent Transportation Systems*, 17(2):441–451.

Fainekos, G. E. and Pappas, G. J. (2009). Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291.

Falcone, P., Borrelli, F., Asgari, J., Tseng, H. E., and Hrovat, D. (2007). Predictive active steering control for autonomous vehicle systems. *IEEE Transactions on control systems technology*, 15(3):566–580.

Faulwasser, T., Kern, B., and Findeisen, R. (2009). Model predictive path-following for constrained nonlinear systems. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 8642–8647. IEEE.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139.

Gray, A., Gao, Y., Lin, T., Hedrick, J. K., and Borrelli, F. (2013). Stochastic predictive control for semi-autonomous vehicles with an uncertain driver model. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 2329–2334. IEEE.

Gurobi Optimization, L. L. C. (2020). Gurobi optimizer reference manual (2020).

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hoxha, B., Dokhanchi, A., and Fainekos, G. (2018). Mining parametric temporal logic properties in model-based design for cyber-physical systems. *International Journal on Software Tools for Technology Transfer*, 20(1):79–93.

Isele, D. (2019). Interactive decision making for autonomous vehicles in dense traffic. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3981–3986. IEEE.

Jardine, P. T. and Givigi, S. N. (2018). A robust model-predictive guidance system for autonomous vehicles in cluttered environments. *IEEE Systems Journal*, 13(2):2034–2045.

Jha, S., Tiwari, A., Seshia, S. A., Sahai, T., and Shankar, N. (2019). Telex: learning signal temporal logic from positive examples using tightness metric. *Formal Methods in System Design*, 54(3):364–387.

Jin, X., Donzé, A., Deshmukh, J. V., and Seshia, S. A. (2015). Mining requirements from closed-loop control models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(11):1704–1717.

Jones, A., Kong, Z., and Belta, C. (2014). Anomaly detection in cyber-physical systems: A formal methods approach. In *53rd IEEE Conference on Decision and Control*, pages 848–853. IEEE.

Kabzan, J., Hewing, L., Liniger, A., and Zeilinger, M. N. (2019). Learning-based model predictive control for autonomous racing. *IEEE Robotics and Automation Letters*, 4(4):3363–3370.

Karlsson, J., Vasile, C.-I., Tumova, J., Karaman, S., and Rus, D. (2018). Multi-vehicle motion planning for social optimal mobility-on-demand. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7298–7305. IEEE.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE.

Ketenci, A. and Gol, E. A. (2019). Synthesis of monitoring rules via data mining. In *American Control Conference*, pages 1684–1689.

Keviczky, T., Falcone, P., Borrelli, F., Asgari, J., and Hrovat, D. (2006). Predictive control approach to autonomous vehicle steering. In *2006 American control conference*, pages 6–pp. IEEE.

Khaitan, S., Lin, Q., and Dolan, J. M. (2021). Safe planning and control under uncertainty for self-driving. *IEEE Transactions on Vehicular Technology*, 70(10):9826–9837.

Koller, T., Berkenkamp, F., Turchetta, M., and Krause, A. (2018). Learning-based model predictive control for safe exploration. In *2018 IEEE conference on decision and control (CDC)*, pages 6059–6066. IEEE.

Kong, Z., Jones, A., and Belta, C. (2016). Temporal logics for learning and detection of anomalous behavior. *IEEE Transactions on Automatic Control*, 62(3):1210–1222.

Kong, Z., Jones, A., Medina Ayala, A., Aydin Gol, E., and Belta, C. (2014). Temporal logic inference for classification and prediction from data. In *Proceedings of the 17th international conference on Hybrid systems: computation and control*, pages 273–282.

Kousik, S., Vaskov, S., Bu, F., Johnson-Roberson, M., and Vasudevan, R. (2020). Bridging the gap between safety and real-time performance in receding-horizon trajectory design for mobile robots. *The International Journal of Robotics Research*, 39(12):1419–1469.

Kulkarni, V. Y. and Sinha, P. K. (2012). Pruning of random forest classifiers: A survey and future directions. In *2012 International Conference on Data Science & Engineering (ICDSE)*, pages 64–68. IEEE.

Lenz, D., Kessler, T., and Knoll, A. (2015). Stochastic model predictive controller with chance constraints for comfortable and safe driving behavior of autonomous vehicles. In *2015 IEEE intelligent vehicles symposium (IV)*, pages 292–297. IEEE.

Li, D., Cai, M., Vasile, C.-I., and Tron, R. (2023). Learning signal temporal logic through neural network for interpretable classification. In *2023 American Control Conference (ACC)*, pages 1907–1914. IEEE.

Li, S., Li, Z., Yu, Z., Zhang, B., and Zhang, N. (2019). Dynamic trajectory planning and tracking for autonomous vehicle with obstacle avoidance based on model predictive control. *Ieee Access*, 7:132074–132086.

Linard, A., Torre, I., Leite, I., and Tumova, J. (2022). Inference of multi-class stl specifications for multi-label human-robot encounters. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1305–1311. IEEE.

Linard, A. and Tumova, J. (2020). Active learning of signal temporal logic specifications. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pages 779–785. IEEE.

Liniger, A. and Van Gool, L. (2020). Safe motion planning for autonomous driving using an adversarial road model. *arXiv preprint arXiv:2005.07691*.

Liu, C., Carvalho, A., Schildbach, G., and Hedrick, J. K. (2015). Stochastic predictive control for lane keeping assistance systems using a linear time-varying model. In *2015 American Control Conference (ACC)*, pages 3355–3360. IEEE.

Liu, J.-F., Wu, J.-H., and Su, Y.-F. (2007). Development of an interactive lane keeping control system for vehicle. In *2007 IEEE vehicle power and propulsion conference*, pages 702–706. IEEE.

Maler, O. and Nickovic, D. (2004). Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer.

Mehdipour, N., Vasile, C.-I., and Belta, C. (2020). Specifying user preferences using weighted signal temporal logic. *IEEE Control Systems Letters*, 5(6):2006–2011.

Mohammadinejad, S., Deshmukh, J. V., and Puranic, A. G. (2020a). Mining environment assumptions for cyber-physical system models. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*, pages 87–97. IEEE.

Mohammadinejad, S., Deshmukh, J. V., Puranic, A. G., Vazquez-Chanlatte, M., and Donzé, A. (2020b). Interpretable classification of time-series data using efficient enumerative techniques. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, pages 1–10.

Neider, D. and Gavran, I. (2018). Learning linear temporal properties. In *Formal Methods in Computer Aided Design*, pages 1–10. IEEE.

Pacejka, H. (2005). *Tire and vehicle dynamics*. Elsevier.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Raman, V., Donzé, A., Maasoumy, M., Murray, R. M., Sangiovanni-Vincentelli, A., and Seshia, S. A. (2014). Model predictive control with signal temporal logic specifications. In *53rd IEEE Conference on Decision and Control*, pages 81–87.

Rick, M., Clemens, J., Sommer, L., Folkers, A., Schill, K., and Büskens, C. (2019). Autonomous driving based on nonlinear model predictive control and multi-sensor fusion. *IFAC-PapersOnLine*, 52(8):182–187.

Ripley, B. D. (2007). *Pattern recognition and neural networks*. Cambridge university press.

Sadigh, D., Sastry, S., Seshia, S. A., and Dragan, A. D. (2016). Planning for autonomous cars that leverage effects on human actions. In *Robotics: Science and Systems*, volume 2. Ann Arbor, MI, USA.

Sadraddini, S. and Belta, C. (2015). Robust temporal logic model predictive control. In *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 772–779. IEEE.

Schwarting, W., Alonso-Mora, J., Paull, L., Karaman, S., and Rus, D. (2017). Safe nonlinear trajectory generation for parallel autonomy with a dynamic vehicle model. *IEEE Transactions on Intelligent Transportation Systems*, 19(9):2994–3008.

Schwarting, W., Pierson, A., Alonso-Mora, J., Karaman, S., and Rus, D. (2019). Social behavior for autonomous vehicles. *Proceedings of the National Academy of Sciences*, 116(50):24972–24978.

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press.

Suh, J., Chae, H., and Yi, K. (2018). Stochastic model-predictive control for lane change decision of automated driving vehicles. *IEEE Transactions on Vehicular Technology*, 67(6):4771–4782.

Tika, A., Hiremath, S., and Bajcinca, N. (2023). Predictive path following control for mobile robots with automatic parameter tuning. In *2023 European Control Conference (ECC)*, pages 1–8. IEEE.

Tumova, J., Hall, G. C., Karaman, S., Frazzoli, E., and Rus, D. (2013). Least-violating control strategy synthesis with safety rules. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 1–10.

Turri, V., Carvalho, A., Tseng, H. E., Johansson, K. H., and Borrelli, F. (2013). Linear model predictive control for lane keeping and obstacle avoidance on low curvature roads. In *16th international IEEE conference on intelligent transportation systems (ITSC 2013)*, pages 378–383. IEEE.

Ulus, D. (2019). Online monitoring of metric temporal logic using sequential networks. *arXiv preprint arXiv:1901.00175*.

Vahidi, A. and Eskandarian, A. (2003). Research advances in intelligent collision avoidance and adaptive cruise control. *IEEE transactions on intelligent transportation systems*, 4(3):143–153.

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

Vazquez-Chanlatte, M., Deshmukh, J. V., Jin, X., and Seshia, S. A. (2017). Logical clustering and learning for time-series data. In *International Conference on Computer Aided Verification*, pages 305–325.

Wang, Q., Ayalew, B., and Weiskircher, T. (2018). Predictive maneuver planning for an autonomous vehicle in public highway traffic. *IEEE Transactions on Intelligent Transportation Systems*, 20(4):1303–1315.

Weiskircher, T., Wang, Q., and Ayalew, B. (2017). Predictive guidance and control framework for (semi-) autonomous vehicles in public traffic. *IEEE Transactions on control systems technology*, 25(6):2034–2046.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.

Wongpiromsarn, T., Topcu, U., and Murray, R. M. (2012). Receding horizon temporal logic planning. *IEEE Transactions on Automatic Control*, 57(11):2817–2830.

Wu, Y., Li, S., Zhang, Q., Sun-Woo, K., and Yan, L. (2022). Route planning and tracking control of an intelligent automatic unmanned transportation system based on dynamic nonlinear model predictive control. *IEEE Transactions on Intelligent Transportation Systems*, 23(9):16576–16589.

Xu, Z., Ornik, M., Julius, A. A., and Topcu, U. (2019). Information-guided temporal logic inference with prior knowledge. In *American Control Conference*, pages 1891–1897.

Yan, R. and Julius, A. (2021). Neural network for weighted signal temporal logic. *arXiv preprint arXiv:2104.05435*.

Yan, R., Xu, Z., and Julius, A. (2019). Swarm signal temporal logic inference for swarm behavior analysis. *IEEE Robotics and Automation Letters*, 4(3):3021–3028.

Yang, H., Hoxha, B., and Fainekos, G. (2012). Querying parametric temporal logic properties on embedded systems. In *Testing Software and Systems: 24th IFIP WG 6.1 International Conference, ICTSS 2012, Aalborg, Denmark, November 19-21, 2012. Proceedings 24*, pages 136–151. Springer.

Zhang, R., Rossi, F., and Pavone, M. (2016). Model predictive control of autonomous mobility-on-demand systems. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1382–1389. IEEE.

CURRICULUM VITAE