

**Titre:** Explainability and reliability for automated dynamic decision systems  
Title:

**Auteur:** Tzu-Yi Chiu  
Author:

**Date:** 2022

**Type:** Mémoire ou thèse / Dissertation or Thesis

**Référence:** Chiu, T.-Y. (2022). Explainability and reliability for automated dynamic decision systems [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie.  
Citation: <https://publications.polymtl.ca/10537/>

 **Document en libre accès dans PolyPublie**  
Open Access document in PolyPublie

**URL de PolyPublie:** <https://publications.polymtl.ca/10537/>  
PolyPublie URL:

**Directeurs de recherche:** Jérôme Le Ny, & Jean Pierre David  
Advisors:

**Programme:** Génie électrique  
Program:

**POLYTECHNIQUE MONTRÉAL**  
affiliée à l'Université de Montréal

**Explainability and reliability for automated dynamic decision systems**

**TZU-YI CHIU**  
Département de génie électrique

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*  
Génie électrique

Août 2022

**POLYTECHNIQUE MONTRÉAL**

affiliée à l'Université de Montréal

Ce mémoire intitulé :

**Explainability and reliability for automated dynamic decision systems**

présenté par **Tzu-Yi CHIU**

en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

a été dûment accepté par le jury d'examen constitué de :

**Richard GOURDEAU**, président

**Jérôme LE NY**, membre et directeur de recherche

**Jean Pierre DAVID**, membre et codirecteur de recherche

**Sofiane ACHICHE**, membre

## ACKNOWLEDGEMENTS

First and foremost, I would like to thank my research advisor Mr. Jérôme Le Ny, Associate Professor in the Department of Electrical Engineering at Polytechnique Montreal, for having accepted me in his research group in March 2020, a tough period when my original internship was cancelled due to the pandemic, so that I can proceed with a 6-month research internship with him. Since then, until the acquisition of my Master of Science degree in August 2022, he has always been responsive to messages and steered me in the right direction by providing inspiring feedback during weekly meetings. He also granted me opportunities to accumulate professional experiences as a teaching assistant to present lab sessions in the robotics course that he offered in the Fall of 2021.

I would also like to acknowledge my co-advisor Mr. Jean Pierre David, Professor in the Department of Electrical Engineering at Polytechnique Montreal, for providing valuable and insightful comments and suggestions on my research work. Additionally, I am sincerely grateful to NSERC, the Natural Sciences and Engineering Research Council, for the funding of scholarship, which made my life in Montreal much easier. I address my sincere thanks to Mr. Jean-Pierre David again for being the grant holder. My gratitude extends to Polytechnique Montreal and its Department of Electrical Engineering for having me admitted as a master student in double degree.

Last but not least, I must express my very profound gratitude to my parents, my girlfriend, my old friends, my roommates and new friends encountered in Montreal for providing me with unfailing support and continuous encouragement throughout the study, the researching and the writing of this thesis. Without them, this accomplishment would have been impossible.

## RÉSUMÉ

Pour de nombreuses tâches automatisées de perception et de décision, les meilleures performances sont actuellement atteintes grâce à des algorithmes trop complexes pour que leur comportement soit entièrement prédictible et compréhensible par les utilisateurs, parce qu'ils déploient de grands modèles d'apprentissage automatique par exemple. Pour déployer ces algorithmes dans des systèmes de décision et de contrôle critiques à la sécurité, il est particulièrement important de développer des méthodes afin de promouvoir la confiance dans leurs décisions et d'aider à explorer leur mode de défaillance. Ce mémoire comprend un article et un travail de recherche complémentaire, où nous cherchons à aborder le problème de certification pour permettre une meilleure compréhension de leurs comportements sous deux aspects différents : explicabilité (générer des explications) et fiabilité (fournir une mesure de qualité) de leurs comportements. Pour l'explicabilité, nous présentons une méthode qui permet de générer des explications pour les comportements de n'importe quel système dynamique de décision traitant des signaux, tandis que pour la fiabilité nous nous concentrons sur un type particulier de système dynamique de décision — système de suivi multi-objets en ligne, où nous proposons d'apprendre une mesure de qualité interprétable pour chaque trajectoire d'objet, ou "tracklet", pour que l'on puisse réagir en conséquence lors de l'exécution.

Plus précisément, dans l'article nous combinons la méthodologie d'« anchors » avec Monte Carlo Tree Search (MCTS) pour générer des explications locales pour un comportement donné d'un modèle dit boîte-noire, qui est censé prendre des décisions de manière dynamique en traitant des signaux qui varient dans le temps. En effet, des systèmes de contrôle tels qu'un système de direction basé sur la perception dans un véhicule autonome sont fondamentalement des systèmes dynamiques entrée-sortie traitant des signaux. Anchors est une méthode modèle-agnostique qui peut s'appliquer à tout type de modèle d'entrée-sortie que l'on cherche à analyser, indépendamment de son architecture interne (e.g., réseau de neurones, système à base de règles, etc.). L'approche proposée cherche des explications très descriptives pour ces décisions sous la forme de propriétés des signaux d'entrée, exprimées en Signal Temporal Logic (STL), qui sont les plus susceptibles de reproduire le comportement observé. STL est une logique populaire qui sert à capturer les propriétés temporelles des signaux, et qui fournit des descriptions riches du comportement des séries temporelles facilement interprétables. Pour illustrer la méthodologie, nous l'appliquons en simulation à l'analyse d'un système de transmission automatique basé sur des règles et d'un système anticollision d'aéronef sans pilote (ACAS Xu) implémenté avec un réseau de neurones.

Dans le travail de recherche complémentaire, nous nous concentrons plus spécifiquement sur les systèmes de suivi multi-objets (MOT) en ligne, dont le but est d'estimer la trajectoire des objets dans l'environnement, tout en maintenant l'identité de chaque objet, dans le cadre du « suivi-par-détection ». Plus précisément, les signaux d'entrée sont des détections 3D sous forme de boîtes englobantes, fournies par des détecteurs 3D prêts à utiliser et qui traitent des nuages de points LiDAR bruts avec un modèle complexe d'apprentissage automatique. Pour mieux comprendre ces systèmes, il est intéressant de disposer d'une mesure de qualité indiquant à tout moment la fiabilité de chaque tracklet. Cette mesure prédit la valeur d'Intersection Over Union (IoU) entre le tracklet et un objet réel potentiel en temps réel. Elle peut s'appliquer par exemple à la gestion du cycle de vie des tracklets, permettant ainsi d'éliminer les faux positifs le plus tôt possible et de réduire le nombre d'objets ratés dû à l'occlusion. En effet, des tracklets non pertinents peuvent être créés par accident à cause des fausses détections ; un objet suivi peut avoir déjà quitté le champ de vue du capteur ; tandis que ceux qui sont simplement occlus pendant une courte durée doivent continuer à être suivis. Nous proposons une approche d'apprentissage automatique surveillé pour apprendre cette mesure de qualité en entraînant un réseau de neurones récurrents Long-Short Term Memory (LSTM), capable de prendre en compte des caractéristiques de tout le passé à partir de la toute première détection de l'objet. Enfin, nous faisons des expériences sur les benchmarks KITTI et nuScenes pour illustrer le système proposé et le comparer avec la littérature. Un problème existant dans l'approche moderne d'évaluation des performances d'un système de suivi multi-objets en ligne est également identifié et discuté.

## ABSTRACT

For many automated perception and decision tasks, state-of-the-art performance may be obtained by algorithms that are too complex for their behavior to be completely understandable or predictable by human users, e.g., because they employ large machine learning models. To integrate these algorithms into safety-critical decision and control systems, it is particularly important to develop methods that can promote trust into their decisions and help explore their failure modes. This thesis includes an article and a complementary research work, in which we try to tackle the problem of certification to provide a better understanding of the behaviors from two different aspects: explainability (generating explanations) and reliability (estimate a quality measure) of the behaviors. For explainability, we introduce a method to generate explanations for behaviors of any dynamic decision system processing signals, while for reliability, we focus on a particular type of dynamic decision system — online 3D Multi Object Tracking (MOT) system, for which we propose to learn an interpretable quality measure for each estimated object trajectory, or tracklet, so that human users can react upon accordingly at runtime.

More precisely, in the article we combine the “anchors” methodology with Monte Carlo Tree Search (MCTS) to provide local explanations for a given behavior of a black-box model, supposed to make decisions dynamically by processing time-varying signals. Indeed, control systems such as a perception-driven steering system for an autonomous car are fundamentally input-output dynamical systems processing signals. Anchors is a model-agnostic method which can be applied to any type of input-output model that one tries to analyze, irrespective of its internal architecture (e.g., neural network, rule-based system, etc.). The proposed approach searches for highly descriptive explanations for these decisions in the form of properties of the input signals, expressed in Signal Temporal Logic (STL), which are most susceptible to reproduce the observed behavior. STL is a popular logic used to capture temporal properties of signals, which provides rich descriptions of the behavior of time series that are easily interpretable by humans. To illustrate the methodology, we apply it in simulations to the analysis of a rule-based automatic transmission system and a collision avoidance system for unmanned aircraft (ACAS Xu) implemented with a neural network.

In the complementary research, we focus more specifically on perception-driven online MOT systems, whose goal is to estimate the trajectory of objects in the surrounding environment based on the “tracking-by-detection” framework, while keeping track of the identity of each object. More specifically, input signals are 3D bounding-box detections, provided by off-the-

shelf 3D detectors based on complex machine learning models processing raw LiDAR point clouds. To better understand these systems, it is beneficial to be informed of a quality measure indicating each tracklet’s reliability at all times. This measure predicts the Intersection Over Union (IoU) value between the tracklet and a potential true instance at runtime. It can be applied for example to life cycle management, allowing to eliminate false positives as early as possible and reducing misses due to occlusion. Indeed, irrelevant tracklets may be accidentally created due to false detections; an object being tracked may have already left the sensor’s field of view; while those which are simply occluded for a short period should keep being tracked. We propose a data-driven approach to learn this quality measure by training a Long-Short Term Memory (LSTM) recurrent neural network, capable of processing features of the entire history starting from the very first detection of the object. Finally, we conduct experiments on benchmarks KITTI and nuScenes to illustrate the proposed system and compare it with previous work. A problem existing with the modern evaluation approach of online MOT system performance is also identified and discussed.



## TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . . .	iii
RÉSUMÉ . . . . .	iv
ABSTRACT . . . . .	vi
TABLE OF CONTENTS . . . . .	viii
LIST OF TABLES . . . . .	xi
LIST OF FIGURES . . . . .	xii
LIST OF SYMBOLS AND ABBREVIATIONS . . . . .	xiv
CHAPTER 1 INTRODUCTION . . . . .	1
1.1 Certification . . . . .	1
1.1.1 Verification: test the system and find decision errors . . . . .	3
1.1.2 Explainability: generate explanations . . . . .	4
1.1.3 Reliability: estimate a quality measure . . . . .	6
1.2 Research objectives . . . . .	7
1.3 Thesis organization . . . . .	8
CHAPTER 2 LITERATURE REVIEW . . . . .	9
2.1 Explainability . . . . .	9
2.1.1 Interpretable Machine Learning . . . . .	9
2.1.2 Signal Temporal Logic . . . . .	10
2.1.3 Monte Carlo Tree Search . . . . .	11
2.2 Reliability . . . . .	12
2.2.1 Object tracking . . . . .	13
2.2.2 Confidence measure . . . . .	14
2.2.3 RNN and LSTM . . . . .	16
CHAPTER 3 RESEARCH APPROACH AND THESIS ORGANIZATION . . . . .	17

CHAPTER 4 ARTICLE 1: TEMPORAL LOGIC EXPLANATIONS FOR DYNAMIC  
DECISION SYSTEMS USING ANCHORS AND MONTE CARLO TREE SEARCH

.....	18
4.1 Introduction .....	18
4.2 Preliminaries and problem statement .....	20
4.2.1 Anchors .....	20
4.2.2 Signal Temporal Logic .....	23
4.2.3 Formal problem statement .....	25
4.3 Computing anchors using Monte Carlo Tree Search (MCTS) .....	26
4.3.1 Primitive candidates .....	26
4.3.2 MCTS algorithm .....	28
4.3.3 Thermostat: an illustrative example .....	32
4.4 Case study: automatic transmission .....	34
4.4.1 Background .....	34
4.4.2 Explaining an STL-based monitoring system .....	39
4.4.3 Explaining the transmission system during a passing maneuver .....	40
4.5 Case study: ACAS Xu .....	45
4.5.1 Background .....	45
4.5.2 Explaining an advisory change .....	46
4.6 Related work .....	48
4.6.1 Interpretable Machine Learning .....	48
4.6.2 Temporal Logic-based inference .....	49
4.6.3 Monte Carlo Tree Search .....	50
4.7 Conclusion and future work .....	51

CHAPTER 5 COMPLEMENTARY RESEARCH WORK: TRACKLET RELIABIL-  
ITY IN ONLINE 3D MULTI OBJECT TRACKING .....

.....	52
5.1 Introduction .....	52
5.1.1 Context .....	52
5.1.2 Contribution .....	53
5.1.3 Outline .....	53
5.2 Literature review .....	53
5.2.1 Online 3D Multi Object Tracking .....	53
5.2.2 Tracklet reliability .....	55
5.3 Online 3D MOT algorithm .....	55
5.3.1 Detector .....	56

5.3.2	Association . . . . .	56
5.3.3	Motion model . . . . .	57
5.3.4	Life cycle management . . . . .	57
5.4	Evaluation of online MOT systems . . . . .	60
5.4.1	CLEAR MOT metrics . . . . .	60
5.4.2	Reflection on integral metrics . . . . .	61
5.5	Training . . . . .	62
5.5.1	Raw data preprocessing . . . . .	62
5.5.2	Neural network structure and hyper-parameters . . . . .	64
5.6	Experiments . . . . .	64
5.6.1	Datasets . . . . .	64
5.6.2	3D detectors . . . . .	65
5.6.3	Hyper-parameters . . . . .	66
5.6.4	Training . . . . .	67
5.6.5	Qualitative results . . . . .	68
5.6.6	Quantitative results . . . . .	68
5.7	Conclusion and future work . . . . .	70
CHAPTER 6 GENERAL DISCUSSION . . . . .		74
CHAPTER 7 CONCLUSION AND RECOMMENDATIONS . . . . .		75
7.1	Synthesis . . . . .	75
7.2	Future work . . . . .	75
REFERENCES . . . . .		76

## LIST OF TABLES

Table 4.1	Hyper-parameters for explaining the STL-based monitoring system using each of the five benchmarks . . . . .	42
Table 4.2	Explanations returned by our algorithm for the STL-based monitoring system using each of the five benchmarks . . . . .	42
Table 4.3	Intermediate results for the STL-based monitoring systems using $\varphi_3, \varphi_4$ and $\varphi_5$ . . . . .	42
Table 4.4	Hyper-parameters for explaining the engagement of the 3 <sup>rd</sup> gear during the passing maneuver . . . . .	44
Table 4.5	The STL formulas after each move to explain the engagement of the 3 <sup>rd</sup> gear during the passing maneuver . . . . .	44
Table 4.6	Hyper-parameters for explaining the switch from SRT to WRT . . . . .	48
Table 4.7	STL formulas after each move for explaining the the switch from SRT to WRT . . . . .	48
Table 5.1	Important hyper-parameters in the association module . . . . .	67
Table 5.2	Validation results of AB3DMOT (KITTI) . . . . .	71
Table 5.3	Validation results of CBMOT (KITTI) . . . . .	71
Table 5.4	Validation results of our system (KITTI) . . . . .	71
Table 5.5	Validation results of AB3DMOT (nuScenes) . . . . .	72
Table 5.6	Validation results of CBMOT (nuScenes) . . . . .	72
Table 5.7	Validation results of our system (nuScenes) . . . . .	72

## LIST OF FIGURES

Figure 1.1	Accidents: (a) Uber’s fatal crash into a pedestrian (b) Tesla’s autopilot failing to stop in front of a truck overturned on a highway . . . . .	2
Figure 1.2	Adversarial examples: (a) an image of “panda” wrongly classified into a “gibbon” by adding some random noise (b) a lane faked by three patched stickers using Tesla Autopilot . . . . .	4
Figure 1.3	Example of a decision tree . . . . .	5
Figure 1.4	Example of YOLO detections of a cat and a dog . . . . .	6
Figure 1.5	Example of objects tracked by an MOT system . . . . .	7
Figure 2.1	The 4 steps involved in one roll-out of MCTS . . . . .	12
Figure 2.2	General architecture of a 3D online MOT system . . . . .	15
Figure 2.3	A graphical representation of an LSTM . . . . .	16
Figure 4.1	Illustration of some example explanations for an input instance $x_0$ and their respective precision and coverage . . . . .	22
Figure 4.2	After roll-outs 0 to 2 . . . . .	35
Figure 4.3	After roll-outs 3 to 5 . . . . .	36
Figure 4.4	After roll-outs 6 to 8 . . . . .	37
Figure 4.5	After roll-outs 9, 10 and 15 . . . . .	38
Figure 4.6	Automatic transmission shift points . . . . .	39
Figure 4.7	Signals violating $\varphi_1$ to $\varphi_5$ . . . . .	41
Figure 4.8	Simulation of the automatic transmission system during a passing maneuver . . . . .	43
Figure 4.9	Geometry for the ACAS Xu horizontal logic table . . . . .	46
Figure 4.10	Simulation of the ACAS Xu system . . . . .	47
Figure 5.1	Diagram of the life cycle management . . . . .	58
Figure 5.2	Example of a training sequence . . . . .	63
Figure 5.3	Structure of our network . . . . .	65
Figure 5.4	Distributions for both true and false detections . . . . .	66
Figure 5.5	Training and validation loss . . . . .	67
Figure 5.6	Example of a tracklet with its two input features and the predicted IoU	68
Figure 5.7	LSTM score distributions before and after each non-detection . . . . .	69

## LIST OF SYMBOLS AND ABBREVIATIONS

**ACAS** Airborne Collision Avoidance System.

**ADAS** Advanced Driving Assistance System.

**AMOTA** Average Multi Object Tracking Accuracy.

**AMOTP** Average Multi Object Tracking Precision.

**CNN** Convolutional Neural Network.

**COC** Clear Of Conflict.

**CPS** Cyber-Physical System.

**DAG** Directed Acyclic Graph.

**FN** False Negative.

**FP** False Positive.

**GIoU** Generalized Intersection Over Union.

**GPS** Global Positioning System.

**GT** Ground Truth.

**IDS** Identity Switch.

**ILP** Inductive Logic Programming.

**IMU** Inertial Measurement Unit.

**IoU** Intersection Over Union.

**LiDAR** Light Detection and Ranging.

**LSTM** Long-Short Term Memory.

**mAP** mean Average Precision.

**MCTS** Monte Carlo Tree Search.

**ML** Machine Learning.

**MOT** Multi Object Tracking.

**MOTA** Multi Object Tracking Accuracy.

**MOTP** Multi Object Tracking Precision.

**PR curve** Precision-Recall curve.

**PSTL** Parametric Signal Temporal Logic.

**PtSTL** Past Time Signal Temporal Logic.

**RaDAR** Radio Detection and Ranging.

**RGB** Red Green Blue.

**RNN** Recurrent Neural Network.

**SLT** Strong Left Turn.

**SRT** Strong Right Turn.

**STL** Signal Temporal Logic.

**SVM** Support Vector Machine.

**TP** True Positive.

**WLT** Weak Left Turn.

**WRT** Weak Right Turn.

## CHAPTER 1 INTRODUCTION

### 1.1 Certification

Self-driving cars are ushering the world into an entire new era, with the goal of improving traffic problems by providing safety and efficiency, especially in urban areas. Unavoidably, due to short concentration time and slow human reaction, vehicles accelerating at different times and drivers not obeying traffic rules are some of the main causes of road problems. Advanced Driving Assistance System (ADAS) is designed to avoid accidents and collisions by alerting the driver, or by implementing safeguards to take over control when a human makes a mistake. Such system may use sensors like camera, LiDAR or even RaDAR to assimilate visual object recognition [1] in human brains, by analyzing important features to recognize the surrounding obstacles. Compared to ADAS, more advanced perception-based autonomous driving systems can be categorized into two major paradigms [2]: mediated perception approaches and behavior reflex approaches. Mediated perception approaches analyze the entire scenario to identify surrounding objects such as lanes, lights, vehicles and other obstacles, and apply rules to decide on the control strategies; while behavior reflex approaches directly mimic human reactions by mapping input scenes to driving actions, using a trained neural network regressor.

In a similar spirit, Airborne Collision Avoidance System (ACAS) is an aviation system which provides maneuver guidance so that horizontal and vertical separation can be maintained between two aircraft. It allows to reduce the risk of mid-air collision and works independently of the Air Traffic Control (ATC) ground systems and other navigation systems. Sensors are used to measure and provide information such as distance, relative position and heading direction of any other nearby aircraft so as to capture the potential risk of collision. Both the ACAS system integrated in an aircraft and the automated control system in a self-driving car are examples of a Cyber-Physical System (CPS), which are designed to connect separate interacting computer components with physical inputs and outputs, including the aforementioned sensors and processors, to intelligently monitor or control specific mechanisms. They are not only intensively used in transportation, but also in many other domains such as robotics, civil engineering, medical monitoring, chemical processes, etc.

For these decision tasks, state-of-the-art performance is usually obtained by algorithms which are too complex for their behavior to be completely understandable or predictable by human users, e.g., because they deploy large Machine Learning (ML) models or are designed with complicated rules. In safety-critical applications, there is an increase need of formal



certification approaches to ensure the system behaves correctly to meet users' expectations, especially in scenarios without human intervention. These requirements may be less important in certain applications, such as recommender systems in movie streaming websites or online shopping platforms, where a suggestion error can often be tolerated. In safety-critical automated CPS however, such as self-driving cars, automated trains or ACAS, an unexpected behavior would cause real danger to human lives or serious damage to the system itself. It is thus desirable to develop certification methods that can promote trust into their decisions, help detect anomalies and explore their failure modes.

Unfortunately, the lack of safety guarantees still prevents people from having faith in *every* decision made by a perception-driven control system. It has been demonstrated that the output may sometimes be sensitive to small input variations, known as adversarial examples [3], a sort of intentional attack designed to lead the model to make a wrong decision. There have already been several severe accidents, including Uber's fatal crash where a self-driving car ran into a pedestrian riding a bicycle across the road [4] in Tempe, Arizona, in 2018, see Figure 1.1(a); and Tesla's autopilot failing to stop in front of a truck overturned on a highway [5], in Taiwan, in 2020, see Figure 1.1(b).

In the following sections we introduce three system certification methods:

1. **Verification:** test the system to find decision errors;
2. **Explainability:** generate explanations for decisions;
3. **Reliability:** provide a quality measure for decisions.



(a) Source: NISTB [4]



(b) Source: Taiwan English News [5]

Figure 1.1 Accidents: (a) Uber's fatal crash into a pedestrian (b) Tesla's autopilot failing to stop in front of a truck overturned on a highway

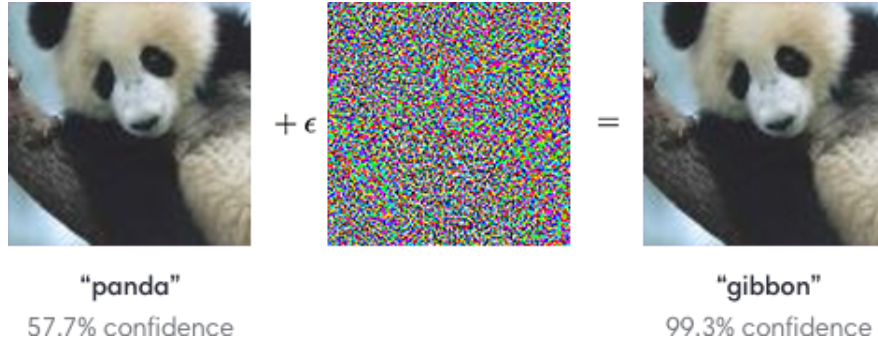
### 1.1.1 Verification: test the system and find decision errors

It has been shown in [3] that even linear behaviors of a simple model in a high-dimensional space can produce adversarial examples, i.e., a slight difference applied on a correctly classified sample may cause the system to make a wrong classification decision. A classic example is shown in Figure 1.2(a), where an image of “panda” is wrongly classified with a 99.3% confidence into a “gibbon”, just by adding some random noise. The Tesla Autopilot system has also been shown by Tencent Keen Security Lab [6] to be vulnerable to adversarial examples which can disable the lane detection function, or even fake a lane with only three patched stickers, see Figure 1.2(b). While researchers keep seeking for corner cases to identify perturbations that may lead to unexpected behaviors, it should be noticed that, as pointed out in [7], it is intractable to verify *every* possible scenario or *every* single part of a CPS, such as a modern microprocessor containing billions of transistors, or a neural network containing millions of parameters.

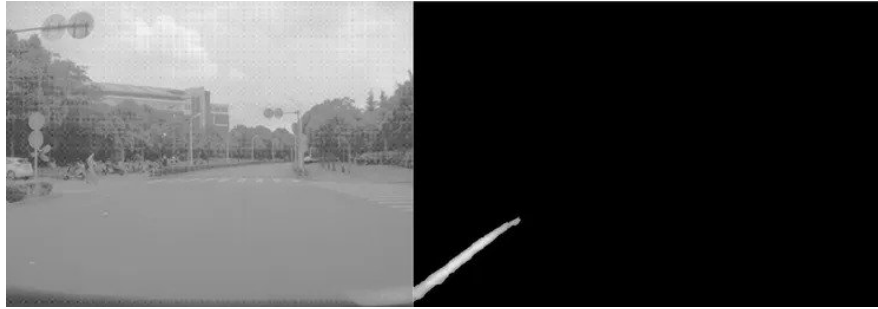
Nonetheless, formal specifications for safe monitoring can still be imposed at runtime to identify errors immediately. For example, some temporal logic requirements are proposed as benchmark problems in [8] for hybrid system verification, such as the Matlab/Simulink automatic transmission model described in [9]:

1.  $\mathbf{G}_{[0,10]}(\text{espd} < 4750)$
2.  $\mathbf{G}_{[0,20]}(\text{vspd} < 120)$
3.  $\mathbf{G}_{[0,30]}(\text{espd} < 3000) \Rightarrow \mathbf{G}_{[0,4]}(\text{vspd} < 35)$
4.  $\mathbf{G}_{[0,30]}(\text{espd} < 3000) \Rightarrow \mathbf{G}_{[0,8]}(\text{vspd} < 50)$
5.  $\mathbf{G}_{[0,30]}(\text{espd} < 3000) \Rightarrow \mathbf{G}_{[0,20]}(\text{vspd} < 65)$

where  $\mathbf{G}_{[t_1,t_2]}$  stands for: “for all times between  $t_1$  and  $t_2$ ” (expressed in seconds), while  $\text{espd}$  and  $\text{vspd}$  denote engine speed (expressed in rpm) and vehicle speed (expressed in mph) respectively. More specifically, the safety monitoring system alerts the engineers when one of these requirements is violated. For instance, the first requirement says: “during the first 10 seconds, the engine speed should never exceed 4750 rpm”; while the third one says: “if in the first 30 seconds the engine speed stays below 3000 rpm, then the vehicle should remain slower than 35 mph in the first 4 seconds.” In some falsification tasks, such as those proposed in the competition of the ARCH workshop [10], participants are asked to find initial conditions or time-varying inputs which lead the system to a violation of an imposed safety requirement, including the aforementioned temporal logic specifications for the hybrid automatic transmission system.



(a) Source: [3]



(b) Source: [6]

Figure 1.2 Adversarial examples: (a) an image of “panda” wrongly classified into a “gibbon” by adding some random noise (b) a lane faked by three patched stickers using Tesla Autopilot

### 1.1.2 Explainability: generate explanations

Explainability of ML models draws continuous attention in the research community of Interpretable ML [11], due to the curiosity and the necessity of knowing *why* a model may behave unexpectedly. For example, a robot vacuum stuck in a rug which is able to provide a viable reason is strongly preferred to one that has a better overall performance but shuts down silently without giving any explanation about its failure. The failure can be anything like low-battery level, wrong configuration, plenty of dust in the inner bag or canister, insufficiency of motor power, etc, which are all beneficial for its user to know about if explained. Explainability plays an even more important role in high-risk environment, where safety-critical mistakes may result in severe consequences, such as a self-driving car running through red lights, or worst, into pedestrians, or an ACAS system providing wrong advisories and resulting in collision of two aircraft.

Explainability also promotes human-machine interaction, as humans tend to ignore commands without knowing *why*, due to the lack of trust. For instance, when the ACAS system suggests that the aircraft slightly change direction, it would be more interesting for the pilot

to know more about the approaching aircraft, including its speed, its heading direction and the distance between the two aircraft, in order to judge the necessity of obeying the system's suggestion. In case of wrong decisions, explanations also provide engineers insight into the algorithms' behavior, so as to identify and remediate a problem more quickly. Also, simply optimizing a loss function to train a model could make certain decisions highly biased in situations of rare occurrence. In some context where discrimination is undesirable, model bias and missing features should be identified and incorporated into the problem formulation.

Some simple models are directly *interpretable*, meaning that the parameters or the model structure itself can be directly given an understandable, or interpretable, meaning. Classic examples of interpretable models include linear regression models and decision trees. Suppose the price of an apartment  $y$  can be predicted via a linear regression model, comprising the following features: the apartment size,  $x_1$ , the number of rooms,  $x_2$ , and the population of the city where the apartment is located,  $x_3$ , with the learnt parameters  $\beta_1, \beta_2, \beta_3 \in \mathbb{R}$ :

$$\hat{y} = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

and suppose the apartment size  $x_1$  increases by one unit while the others remain constant, then  $\beta_1$  simply suggests by how much the price  $y$  would increase. The same thing for the other parameters, which also have easily interpretable meanings with the corresponding feature. A decision tree such as the one illustrated in Figure 1.3 is even more interpretable. The model makes a decision simply based on the value of some features in a greedy manner.

However, some models are too large and complex for each parameter to be interpretable, such as a neural network or a decision system based on sophisticated rules. To interpret the behaviors of these models, *model-agnostic* methods have been proposed by simply treating them as *black-box* models. These methods would generate explanations completely separated

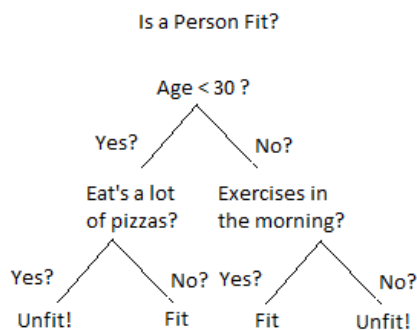


Figure 1.3 Example of a decision tree

from the model itself and are thus flexible as they can be applied to any model. Model-agnostic methods can be further categorized as either global or local. Global methods seek to characterize the average behavior of the model, while local ones focus specifically on one scenario and one decision by exploiting interesting properties of that particular scenario. In Chapter 4, we focus on a specific type of local model-agnostic method, *anchors* [12], to generate local explanations for a decision made by an automated dynamic decision system.

### 1.1.3 Reliability: estimate a quality measure

To improve certification, it is also interesting to provide decisions with an interpretable quality measure, or confidence score, at runtime. This measure informs about their reliability, so that users can further exploit this measure to react accordingly, depending on the application. For instance, in medical diagnosis, instead of simply informing the doctor of the existence of a cancer, the detection’s reliability would also highly influence the doctor’s analysis. In object detection, YOLO [13] chooses to train the confidence score of a bounding box with the Intersection Over Union (IoU) between the box and the true object if the latter is indeed present, and push it to zero in case no object is in the box. An example of YOLO detections is shown in Figure 1.4. Here, the measure IoU between two bounding boxes quantifies their overlap ratio between the intersection area and the union area:

$$\text{IoU}_{B_1, B_2} := \frac{|B_1 \cap B_2|}{|B_1 \cup B_2|} \in [0, 1]$$

where  $|\cdot|$  denotes the area (or volume in 3D),  $\cap$  denotes intersection while  $\cup$  denotes union.

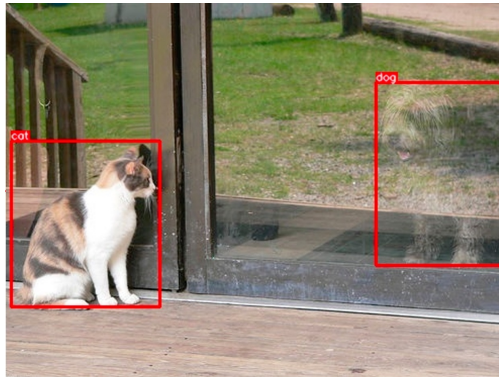
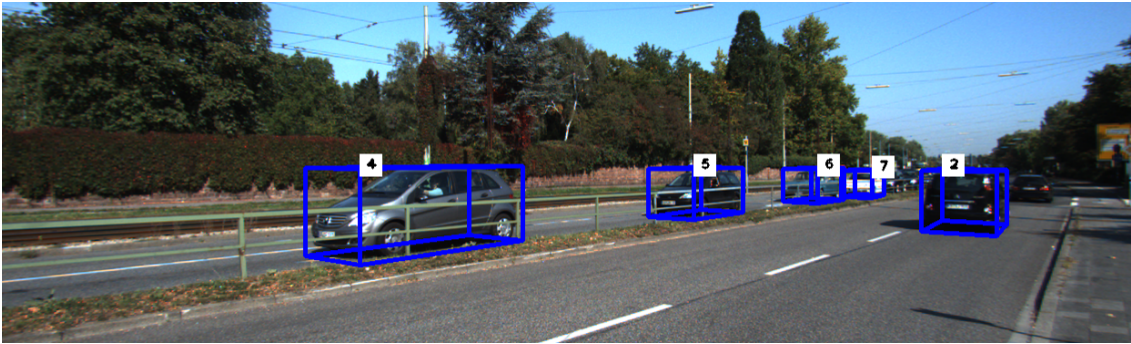


Figure 1.4 Example of YOLO detections of a cat and a dog

These confidence scores, despite the absence of a proper definition, are largely used in non-maximum suppression and performance evaluation. Non-maximum suppression means that if a number of overlapping boxes correspond to the same object, only the box with the best confidence score is considered; in performance evaluation, on the other hand, thresholding on the confidence score can change the number of output detections thus allows to compute some average metrics. In Chapter 5, we focus on a specific dynamic decision system — online 3D Multi Object Tracking (MOT) system using LiDAR, where multiple objects are detected and tracked, as shown in Figure 1.5 below. For better visualization, 3D bounding-box detections are projected onto a 2D image. Tracking systems allow to continuously estimate the bounding box information based on past observations, while remaining a constant identity for each object, despite a possible short period of occlusion, but these systems still lack an interpretable quality measure for each estimated bounding box. For this reason, we introduce in Chapter 5 the use of a recurrent neural network to predict such a measure in an online fashion.



KITTI dataset [14]. Sequence 0004, 26<sup>th</sup> frame. Detector: PointRCNN [15]

Figure 1.5 Example of objects tracked by an MOT system

## 1.2 Research objectives

Among the three certification methods discussed previously, in this thesis we propose methods for the latter two: explainability and reliability. Due to the high dimensionality of the raw input signals used to perceive complex environments (e.g., video signals), certification of control systems is becoming more challenging. In terms of explainability, we seek to explain a given decision of a system processing a given time-varying signal, by generating explanations with a human-friendly temporal logic. This involves designing an efficient algorithm to find the optimal explanation in a large search space. Then the method and the generated expla-

nations should be illustrated in simulation or with real-world data. In terms of reliability, we focus on a specific dynamic decision system — online 3D MOT system. We seek to provide decisions, which, in the case of MOT systems, are basically the estimated bounding boxes of the surrounding objects, with an interpretable quality measure. This measure can eventually improve the tracking system and should be illustrated with public real-world datasets.

### 1.3 Thesis organization

This thesis by article includes one article submitted in April 2022 (Chapter 4):

- Tzu-yi Chiu, Jérôme Le-Ny, and Jean-Pierre David, “Temporal Logic Explanations for Dynamic Decision Systems using Anchors and Monte Carlo Tree Search”, *The journal of Artificial Intelligence (AIJ)*, [under review] 2022

and a complementary research work (Chapter 5) titled “Tracklet Reliability in 3D Online Multi Object Tracking”. The research presented in this thesis was done in the mobile robotics and autonomous systems laboratory at Polytechnique Montreal, under supervision of Prof. Jérôme Le Ny.

The remainder of this thesis is organized as follows. Chapter 2 reviews some basic concepts and the related literature. Next, Chapter 3 briefly summarizes the research approaches to cope with the above research objectives. Chapter 4 and Chapter 5 contain the article and the complementary research work, respectively. Finally, Chapter 6 provides a general discussion of the results obtained in the previous two chapters, while Chapter 7 concludes the thesis with potential future work.

## CHAPTER 2 LITERATURE REVIEW

In this chapter, we review the related literature and present some basic concepts. We divide this chapter into Section 2.1 and Section 2.2, to discuss the two certification methods in the scope of our work: explainability (Chapter 4) and reliability (Chapter 5).

### 2.1 Explainability

We seek to generate explanations for a given behavior of an automated dynamic decision system. Section 2.1.1 presents the literature in the area of Interpretable Machine Learning, where we also present *anchors*, a framework allowing to quantify the relevance of explanations using the concept of *precision* and *coverage*. To involve time properties, the explanations will be expressed in the form of Signal Temporal Logic (STL) formulas, discussed in Section 2.1.2. Then, the conceived algorithm for searching the best formulas is based on Monte Carlo Tree Search (MCTS), which is finally presented in Section 2.1.3.

#### 2.1.1 Interpretable Machine Learning

Interpretation methods can be categorized as either model-specific or model-agnostic. Model-specific methods apply to particular types of ML models, whose structure can be exploited. For instance, in computer vision, class activation maps (CAM [16] and Grad-CAM [17]) identify the input pixels that most influence a classification decision made by a Convolutional Neural Network (CNN). In some neural networks that are too large and complex however, the huge number of parameters (weights and biases) included in a network prevents us from interpreting its behaviors from the model itself. Model-agnostic methods are more flexible, as they can be applied to *any* model, which can be treated as a black-box model. Instead of analyzing the model's inner structure, these methods rather seek to provide interesting features of the input data that would most probably result in a specific decision.

Global model-agnostic methods such as replacing the model by a small decision tree [18] characterize the average behavior of the model based on the input data distribution, but are usually not precise enough. On the other hand, local methods only focus on a specific region of the input space. In Chapter 4, we consider an important local method, *anchors* [12], based on which we try to explain specific behaviors of dynamic systems by incorporating temporal properties. In the framework of anchors, two metrics, *precision* and *coverage*, are introduced to measure the relevance of an explanation. Essentially, an explanation of high precision



provides an accurate sufficient condition on the input features such that the output decision remains the same. A larger coverage makes the explanation more general, i.e., applicable to more input instances, thus approaching a necessary condition for the decision. A more general explanation is also more comprehensible and user-friendly. Using the terminology of [19], these *rule-based* explanations define clear boundaries between the decision for which we request an explanation and a different one (see Section 4.2.1 for more details). Another local method, LIME [20], precedes the method of anchors, and locally approximates the model by a simpler interpretable model, e.g., a linear model or decision tree.

### 2.1.2 Signal Temporal Logic

STL [21] is a type of temporal logic [22] used to describe temporal and spatial properties of a signal. The most common STL formulas are constructed based on the two operators  $\varphi \mapsto \mathbf{F}_{[a,b]}\varphi$  and  $\varphi \mapsto \mathbf{G}_{[a,b]}\varphi$ , which mean that the predicate  $\varphi$  must be true at least once ( $\mathbf{F}$ ) or at all times ( $\mathbf{G}$ ) in the time interval  $[a, b]$ , starting from the current time  $t$ . Past Time Signal Temporal Logic (PtSTL) can also be used when it is more natural to generate explanations based on “past” observations. STL also admits quantitative semantics [23, 24], quantified by the “robustness degree”, capturing how far a signal is about to violate a formula.

Some special STL formulas are called *primitives*, which may have different structures. For example, we can define primitives of different levels, such as first-level primitives  $\mathcal{P}_1$  and second-level primitives  $\mathcal{P}_2$  adapted from [25]:

$$\begin{aligned} \mathcal{P}_1 &:= \left\{ \mathbf{F}_{[a,b]}(s \leq \mu), \mathbf{G}_{[a,b]}(s \leq \mu) \mid 0 \leq a \leq b, \mu \in \mathbb{R} \right\} \\ \mathcal{P}_2 &:= \left\{ \mathbf{F}_{[a,b]}\mathbf{G}_{[0,c]}(s \leq \mu), \mathbf{G}_{[a,b]}\mathbf{F}_{[0,c]}(s \leq \mu) \mid 0 \leq a \leq b, c \geq 0, \mu \in \mathbb{R} \right\} \end{aligned}$$

In words, at the current time  $t$ ,  $s \leq \mu$  intuitively means “the signal is [smaller/greater] than  $\mu$ ”, and the first-level primitives in  $\mathcal{P}_1$  mean “ $s \leq \mu$  is [once/always] true within the time interval  $[t + a, t + b]$ ”. For second-level primitives in  $\mathcal{P}_2$ ,  $\mathbf{F}_{[a,b]}\mathbf{G}_{[0,c]}(s \leq \mu)$  specifies “within  $[t + a, t + b]$ , there is a time after which  $s \leq \mu$  is true for  $c$  timestamps”, and  $\mathbf{G}_{[a,b]}\mathbf{F}_{[0,c]}(s \leq \mu)$  specifies “for all times in  $[t + a, t + b]$ ,  $s \leq \mu$  is true at least once within the next  $c$  timestamps”. The second-level primitives are richer than the first-level ones. Note that depending on the desired STL specifications, one is not limited to these two sets of primitives. Finally, longer STL formulas can be obtained via conjunction (and,  $\wedge$ ) or disjunction (or,  $\vee$ ) of these primitives.

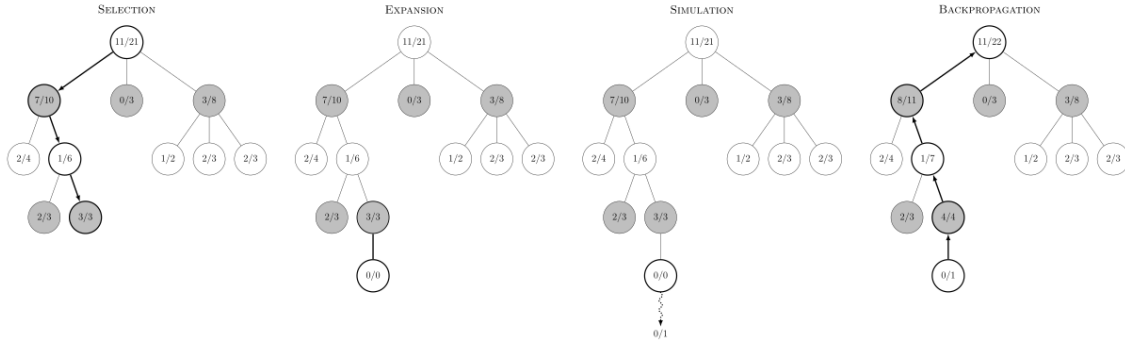
Related literature includes [26] that tries to classify signals into anomalous and normal signals using STL formulas. The search includes finding an appropriate formula structure, using Parametric Signal Temporal Logic (PSTL) [27], as well as the parameters in a given PSTL formula via optimization. Bombara *et al.* [28] use a decision-tree approach to build STL formulas incrementally to classify a given set of signals. These methods suppose that a dataset of signals is available to learn the formula. When only the model and a specific input-output pair are available, it would be inefficient to generate a large dataset to apply the method, without any guarantee that every special case could be included in the dataset and explored sufficiently. That is the reason why we use the “anchors” methodology: with some probability of error we can generate just enough signal samples to find the boundaries described in STL.

### 2.1.3 Monte Carlo Tree Search

MCTS [29] is a heuristic algorithm that employs a multi-step look-ahead strategy to make decisions sequentially, under uncertainty, in a large search space. Combined with deep learning, it was used in the algorithm of AlphaGo [30], a program developed by Google’s DeepMind to master the board game of Go. The data structure used in MCTS is a game tree, or a Directed Acyclic Graph (DAG), whose nodes represent states while edges represent actions. Given the current state, the goal is to choose the most promising action to pass to the next state among its children, by exploring multiple steps ahead. In contrast to the *Minimax* search algorithm, a basic depth-first adversarial search method, MCTS doesn’t explore *all* possible subsequent states, especially in a complex game like Go where hundreds of possible actions can be chosen at each state. Exploring all possible nodes is infeasible within a limited amount of time. Starting from the root state, the tree is built incrementally using guided random sampling, via simulation, to estimate the success rate of each possible action. MCTS is capable of finding the balance between exploration and exploitation using a bandit strategy [31,32], to concentrate on the most promising actions.

To briefly summarize the algorithm, before making every decision a large number of *roll-outs* are performed repeatedly until the imposed time limit is reached, starting from the root node corresponding to the current state. A roll-out consists of 4 steps, as illustrated in Figure 2.1:

1. **Selection.** A path is selected from the root to a leaf using a bandit strategy.
2. **Expansion.** If the leaf reached is not a terminal node, the leaf is expanded and its children are incorporated into the tree.
3. **Simulation.** Starting from the leaf, Monte-Carlo simulations are performed until a result is achieved, which is then used to update the success rate.



Source: Wikipedia

Figure 2.1 The 4 steps involved in one roll-out of MCTS

4. **Back-propagation.** Simulation scores are also back-propagated to every ancestor in the selected path to update their success rate.

To draw connection with our work in Chapter 4, each state corresponds to an STL formula, while its children are longer formulas obtained via conjunction with an STL *primitive*. Interestingly, the empirical score computed for each state can be related to the notion of “precision” defined in anchors [12]. Originally, the search algorithm proposed in [12] consists of a Multi-Armed Bandit (MAB)-based sampling strategy, KL-LUCB [33], to greedily optimize the empirical precision. Inspired by [34, 35] where MCTS is used for feature selection, we propose to improve the efficiency with MCTS to search for high-precision explanations.

## 2.2 Reliability

To be informed of the reliability of decisions, we seek an interpretable quality measure at runtime. In the complementary research work (Chapter 5), we focus specifically on MOT systems. In Section 2.2.1, tracking systems are introduced and the related literature is reviewed. Next, we show in Section 2.2.2 the use of confidence measures in ML in general, and in object tracking more specifically. Finally, in Section 2.2.3 we introduce Recurrent Neural Network (RNN), and a particular type of RNN — Long-Short Term Memory (LSTM), which are capable of making predictions that take into account past information. We make use of an LSTM to learn such a measure, which can provide interpretable meaning.

### 2.2.1 Object tracking

Object tracking [36] is an important task in computer vision applied in a variety of domains: video surveillance, traffic monitoring, logistics, robotics, sports analysis, etc. Different modalities such as camera, LiDAR and RaDAR can be used to detect surrounding objects. With camera, a 2D object detector takes a static image as input and outputs a set of 2D detections, described with bounding boxes or segmentation masks for localization, and may encode their RGB information together. Stereo cameras can complement monocular cameras with depth information and improve accuracy. LiDAR and RaDAR, on the other hand, estimate the distance of objects, using the *time-of-flight* distance measurement of light pulses and radio waves, respectively. The scanner can thus incrementally encode necessary information about the detected physical surfaces to the point cloud (3D data points), which is then processed by a 3D detector to output a set of 3D detections. See Figure 1.5 for illustration. These 3D bounding-box detections are projected on a 2D image for better visualization.

In Visual (Single) Object Tracking, only the initial location of the object is provided, and the goal is to identify its location in every subsequent frame. Relevant features are compared to effectively discriminate the object from the background. Depending on the way the object is described, methods can be classified as either point-based or contour-based. In point-based methods such as [37], objects are described using feature points (or keypoints). However, feature points don't explicitly provide precise location of the object. In contour-based methods however, the object is described with a bounding box or a segmentation mask. For example, [38] surveys the most prominent tracking methods: Discriminative Correlation Filters and Siamese Networks. In Multi Object Tracking (MOT) [39], multiple objects have to be discovered and localized in each frame and a same object should maintain a constant identity across frames. The estimated trajectory of an object across frames is called a tracklet. With the rapid development of state-of-the-art deep learning models trained for object detection in real-time speed, a "tracking-by-detection" framework has been used in recently proposed algorithms.

Methods can also be categorized as either online or offline, depending on the way sequences are processed. In offline tracking, a video of 2D images or 3D point clouds is processed as a whole to obtain a global optimal solution, thus future information can be used to determine an object's current location on a given frame. In online tracking however, only up-to-time observations can be used and decisions should be made online at each frame.

According to [40], an online MOT system can be decomposed into four individual modules: detector, association, motion model and life cycle management. The general architecture is shown in Figure 2.2. To briefly summarize the architecture, at each frame, the detector pro-

vides a set of object detections, which are new measurements used to update tracklets, while their movement is modelled and predicted with the motion model. The association module matches tracklets with new detections, while the life cycle management decides whether tracklets should be kept or deleted.

Recent literature on 3D online MOT includes a simple baseline method, AB3DMOT [41], which uses a 3D detector that processes LiDAR point clouds to provide 3D bounding boxes, then uses a Kalman Filter [42] as the motion model, the Hungarian algorithm [43] for association, and finally Intersection Over Union (IoU) as the matching metric. Among its followers that seek to improve the association module, Chiu *et al.* [44] replaces IoU by Mahalanobis distance [45] to take into account the tracklet’s state uncertainty; CenterPoint [46] considers objects as points and replaces IoU by their euclidean distance; [40] generalizes IoU to GIoU [47]; FANTrack [48] trains a deep CNN for data association; finally, [49–51] add 2D camera data to provide additional position and RGB information. Different to previous works, CBMOT [52] concentrates on improving life cycle management. They design score-update functions and a score decay mechanism to assign confidence scores to tracklets, and distinguish their “confidence-based” methods from “count-based” rules proposed in [41], based on which the number of frames of (non-)detection is used to determine the life cycle of each tracklet.

### 2.2.2 Confidence measure

As explained previously, it is difficult to properly define what the confidence score of a prediction or a decision made by an ML model is. In a neural network that performs image classification, the confidence score of “an image associated to a class” is often considered as the output corresponding to that class after the sigmoid or softmax normalization layer. In an SVM classifier, the margin can be used as a proxy of confidence measure. However, as pointed out by [53], these scores are only useful to compare reliability between two predictions, but do not usually provide meaningful interpretation. A reliable interpretation of such a measure can enable human users to react to the result accordingly. In object detection, YOLO [13] trains the confidence score of a bounding box with the IoU between the box and the true object, and can be used in non-maximum suppression to choose the best bounding box if there are multiple of them corresponding to a same object.

It can also be used in performance evaluation via thresholding on this score. The two major evaluation metrics of a system in ML are *recall* and *precision*: recall is defined as the ratio of correct objects output by the system among all true objects (ground-truth), while precision is the ratio of correct objects output by the system among all objects output by the system. In

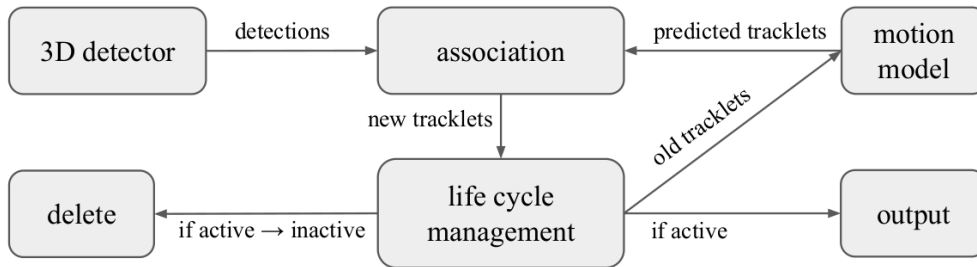


Figure 2.2 General architecture of a 3D online MOT system

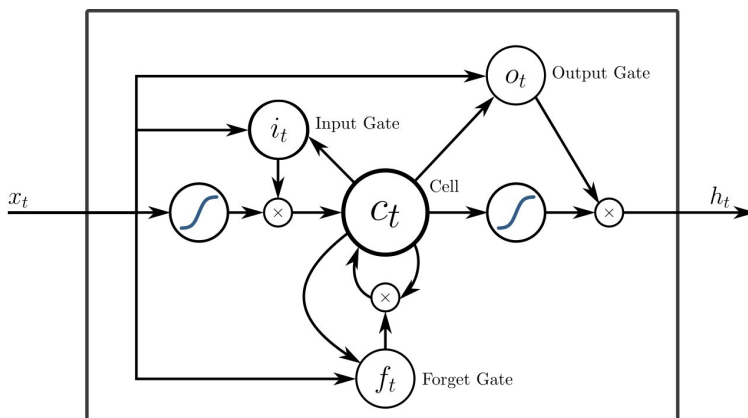
general, depending on the application, there is a trade-off to find between recall and precision, because outputting everything is equivalent to setting the threshold to 0, while increasing the threshold can decrease the recall and improve the precision. Thus, thresholding on the confidence score allows to find the best threshold to balance the two metrics. To evaluate a system, certain metrics such as mean Average Precision (mAP) require that the model performs correctly at each threshold by computing the area below the Precision-Recall curve (PR curve).

Concerning the confidence scores used in object tracking specifically, a Bayesian model is used in [54] to estimate the probability of a binary hypothesis (presence, absence) or the density function of the object’s state (and presence) based on previous observations. [55] designs heuristic functions to assign confidence scores to a tracklet using past information such as its length, the number of frames of non-detection and the affinity with matched detections. In 3D tracking, [52,56] design heuristic score-update functions to compute confidence scores for tracklets. In a similar spirit of mAP, AB3DMOT [41] proposes two integral metrics AMOTA and AMOTP for evaluation by thresholding on the tracklet’s overall confidence score, defined as the average of its confidence scores across all frames. However, these scores can neither be obtained at runtime nor necessarily have an interpretable meaning. In our complementary research work (Chapter 5), we propose to provide each tracklet with an interpretable quality measure by training an LSTM [57], introduced in the next section, to predict the IoU between the tracklet and a true instance, as in YOLO [13], with the goal of informing users of each tracklet’s reliability at runtime.

### 2.2.3 RNN and LSTM

Recurrent Neural Network (RNN) addresses the issue of stateless neural networks incapable of making predictions based on past observations. While stateless neural networks may take as input a window of past observations of fixed size, this would result in high dimensional inputs and intermediate layers. RNNs can take inputs of variable length without being too large and complex, by including loops in their internal structure. More specifically, memory can be stored in hidden states and propagated forward to the next time step. With a new input, the model updates the hidden states, to take into account past information. The hidden states are then used to output predictions, usually by simply applying a linear (fully connected) layer at the end.

Unfortunately, classical RNNs are known for their short-term memory and unable to memorize previous information from long time ago [58]. LSTM [57] is a special type of RNN, carefully designed to keep track of long-term dependencies from the past. In addition to the hidden states present in every RNN, it also holds cell states that contain “gated” information flow for long-term memory. Figure 2.3 shows a graphical representation of an LSTM, where three of these gates are present: the forget gate, the input gate and the output gate. A gate is a sigmoid layer, which outputs a number between 0 and 1, indicating how much information can actually get through, followed by a pointwise multiplication operation. The forget gate controls which bit of information in the long-term memory has less weight and should be forgotten. The input gate controls the flow of new information from the new input and hidden states into the long-term memory. Finally, the output gate decides how the hidden state should be updated, depending on the new cell states, the new input and the previous hidden states.



Source: Wikipedia

Figure 2.3 A graphical representation of an LSTM

### CHAPTER 3 RESEARCH APPROACH AND THESIS ORGANIZATION

To allow a better understanding of automated dynamic decision systems, we propose in this thesis methods to improve certification of these systems from two aspects: explainability and reliability, as discussed respectively in the submitted article included in Chapter 4 and the complementary research work presented in Chapter 5.

On the one hand, to explain automated decisions (Chapter 4), we introduce a model-agnostic method to generate user-friendly local explanations for a given behavior of *any* dynamic decision system, treated as black-box model. To include features of time series in dynamic scenarios, we search for explanations expressed in STL [21], a highly descriptive logic capable of capturing temporal properties of input signals, which are easily understandable by human users. Indeed, the aim is that such explanations should be usable by human designers to better understand the failure modes of a given system or gain confidence in its capabilities. The search in such a high-dimensional space under uncertainty requires the use of heuristic algorithms such as MCTS [29], to gradually optimize the “precision” of an explanation in the framework of *anchors* [12]. Properties of STL can also be leveraged to find formulas that maximize the “coverage” among previously found high-precision explanations (anchors). Finally, we illustrate our method in simulations with the analysis of an automatic transmission system implemented in a vehicle, and a collision avoidance system for unmanned aircraft (ACAS Xu) implemented with a neural network.

On the other hand, for reliability, we seek to provide an interpretable quality measure for every decision made by the system. In the complementary research work of Chapter 5, we focus on a particular type of perception-driven dynamic decision system — online 3D MOT system. Recent papers generally design these systems based on the “tracking-by-detection” paradigm. Upon these systems, we propose a data-driven approach using LSTM [57] recurrent neural networks to learn a quality measure for each tracklet at runtime, predicting the IoU measure between the tracklet and a potential true instance. We show that this quality measure can also help manage each tracklet’s life cycle by conducting extensive experiments on two benchmarks: KITTI [14] and nuScenes [59].

The remainder of this thesis is composed of the submitted article about explainability (Chapter 4), the complementary research work about the reliability of MOT systems (Chapter 5), a general discussion about the obtained results (Chapter 6) and a conclusion with potential future work (Chapter 7).



# CHAPTER 4    ARTICLE 1:    TEMPORAL LOGIC EXPLANATIONS FOR DYNAMIC DECISION SYSTEMS USING ANCHORS AND MONTE CARLO TREE SEARCH

Tzu-yi Chiu, Jérôme Le Ny, Jean Pierre David

*The journal of Artificial Intelligence (AIJ)*

*Submission date: April 11, 2022*

## 4.1 Introduction

Progress in Machine Learning (ML) in recent years is motivating efforts to use data-driven design methodologies in the development of automated decision and control systems. For example, neural networks trained on large image datasets can be used for perception in self-driving cars or automated trains, and reinforcement-learning based controllers could be used for online planning. Unfortunately, modern machine learning models and architectures, such as deep neural networks used for object recognition and image interpretation, are often very large and complex, containing millions of parameters in a nonlinear architecture. Thus, assuring that these models will behave as expected in real-world situations is generally hard, which constitutes a barrier to their adoption.

To integrate and deploy modern ML-based systems into safety-critical control systems, one needs to understand, trust and even certify their behavior for environments that could differ from a given training dataset. It has been shown that the outputs of ML models can be in some cases very sensitive to small variations in their inputs (adversarial examples [3]). High performance on a test dataset could also be due to the model learning spurious correlations due to deficiencies in the data collection process and as a result might not be representative of real-world performance [20]. Addressing these issues requires developing methods to assist designers of ML-based systems in justifying, explaining or even certifying to third parties to which extent a given system is likely to work as intended. Among existing approaches [11], we focus here on *model-agnostic* explanation methods, which can be applied to any type of input-output model that one tries to analyze, irrespective of its internal architecture (e.g., neural network, rule-based system, etc.). These methods can then be used, e.g., to compare the behaviors of different types of models trained to solve the same given task.

Control systems, e.g., a perception-driven steering system for an autonomous car, are fundamentally input-output dynamical systems processing time-varying signals. Increasingly, the high-dimensionality of the raw input signals and the complexity of the environments in

which these systems are deployed require new decision-making architectures together with the development of new validation methodologies that complement traditional model-based formal analysis or simulation-based methods. In this chapter, we develop a framework to provide local model-agnostic explanations for the behaviors of a dynamic decision or control system. These explanations are expressed in Signal Temporal Logic (STL) [21], a popular logic used to capture temporal properties of dense-time real-valued signals, which provides rich descriptions of the behavior of time series that are easily interpretable by humans. Indeed, the aim is that such explanations should be usable by human designers for example to better understand the failure modes of a given system or gain confidence in its capabilities.

In this work, a simulator or implementation of the control system under study is assumed to be available, which consists of a module computing decisions based on simulated or physical input signals. Given a behavior of interest, i.e., an input signal and corresponding decision, our algorithm searches efficiently for precise STL formulas that attempt to describe the most relevant properties of the input signal that lead to this decision. Depending on the context, it is also possible to use a variant of STL called Past Time Signal Temporal Logic (PtSTL) [60], since an action performed by a controller is based on its past measurements. Following the general *anchors* framework of Ribeiro et al. [12], for a formula to provide a good explanation of a behavior, any input signal verifying the formula should lead the system to reproduce this behavior with high probability. This probability is called the *precision* of the formula. We describe an algorithm to search for descriptive STL formulas using Monte Carlo Tree Search (MCTS) [29], an efficient heuristic algorithm allowing us to concentrate the search on the most promising formulas (those expected to have higher precision) within a large possible space, using guided random sampling. Another important measure for the relevance of an explanation is the notion of *coverage*, which captures the generality of an explanation, i.e., whether it can be applied to more input instances. Indeed, a more general explanation is argued to be more comprehensible and user-friendly. We describe how to leverage specific features of STL to gradually maximize the coverage among high-precision formulas.

The rest of the paper is organized as follows. We start by providing the necessary background on the *anchors* methodology in Section 4.2.1 and on STL in Section 4.2.2, in order to formally state in Section 4.2.3 the problem we address. Section 4.3 describes the proposed algorithm in detail, more specifically, how MCTS is adapted to generate STL-based explanations (anchors) efficiently, with an illustrative example allowing to walk through the algorithm step-by-step. Then, two case studies are discussed. In Section 4.4, we consider a hybrid (continuous-discrete) control system implementing an automatic transmission in a vehicle [9]. We start by evaluating and validating our algorithm with a monitoring system for which the real explanations are known [61], and then illustrate the method by analyzing

a simulated behavior of the transmission system. In Section 4.5, we apply the methodology to the ACAS Xu system [62], which uses a neural network to issue guidance instructions to avoid collisions between aircraft. Finally, related work is discussed in Section 4.6 before concluding the paper with future work in Section 4.7.

## 4.2 Preliminaries and problem statement

### 4.2.1 Anchors

We are concerned with describing formally certain behaviors of critical systems whose structure might be either partially unknown or too complex to be fully characterized. This work builds on the *anchors* methodology of Ribeiro *et al.* [12], which provides *local* model-agnostic explanations for specific behaviors (input-output pairs) produced by a black-box input-output system. Formally, let  $f : X \rightarrow \{0, 1\}$  be a black-box model and  $x_0 \in X$  be a given input instance for which we want to explain the model’s output  $f(x_0)$ . An anchor can be viewed as a logic formula describing via a set of rules (predicates) a neighborhood  $A_{x_0} \subset X$  of  $x_0$ , such that inputs sampled from  $A_{x_0}$  lead to the same output  $f(x_0)$  with high probability. Since this concept of explanation is very general, algorithms to find good anchors for different types of systems (e.g., for text analysis, image classification, or, in our case, dynamic decision-making systems) can benefit from leveraging domain-specific tools.

For our problems of interest,  $f$  defines the behavior of a dynamic controller or decision-making module, for which we seek a local explanation. The input  $x_0$  is an observable signal to be analyzed, and  $A_{x_0}$  corresponds to a monitoring rule on signals, which we find convenient to formulate using STL. Hence, the term *rule* in the following can refer both to a subset of  $X$  and to a logical formula describing this subset. We assume that we are given a “perturbation distribution” [12], i.e., a fixed probability distribution  $\mathcal{D}_{x_0}$  on  $X$  (which depends on  $x_0$ , see Remark 4.2.1 below) that we use to sample input signals, by perturbing  $x_0$ . All probabilities in the following are defined by considering  $X$  as a probability space equipped with  $\mathcal{D}_{x_0}$ .

**Definition 4.2.1.** Let  $\tau \in [0, 1]$ . A rule  $A \ni x_0$  (satisfied by  $x_0$ ) is said to be an *anchor* for  $x_0$  if

$$p_A \geq \tau \tag{4.2.1}$$

where  $p_A$  denotes the *precision* of  $A$ , defined as follows:

$$p_A := \mathbb{P}(f(z) = f(x_0) \mid z \in A). \tag{4.2.2}$$

Therefore, an anchor  $A$  is a rule verified by  $x_0$  whose precision is above a certain threshold  $\tau$ , where the precision of  $A$  is defined as the probability that the black-box model’s output is the same for a random input  $z$  satisfying  $A$  as for  $x_0$ . Hence,  $A$  provides some conditions that are sufficient to predict the output of the model with high probability (under the perturbation distribution  $\mathcal{D}_{x_0}$ ). Another important notion introduced in [12] is the notion of *coverage*.

**Definition 4.2.2.** The coverage of a rule  $A$  is defined as  $\text{cov}(A) := \mathbb{P}(z \in A)$ .

Hence, the coverage of  $A$  is the probability that a random input (according to  $\mathcal{D}_{x_0}$ ) satisfies  $A$ . Although only the precision is involved in Definition 4.2.1 to define *anchors*, rules that have broader coverage, i.e., rules that are satisfied by more input instances, are intuitively preferable. Essentially, an explanation of high precision approximates an accurate sufficient condition on the input features such that the output remains the same, while a larger coverage makes the explanation more general, thus approaching a necessary condition. Therefore, [12] proposes to maximize the *coverage* among all rules satisfying (4.2.1) to find the best explanations among anchors, i.e., among those which have sufficient precision. This anchor with maximized coverage allows to locally approximate the border separating the output from the rest. Illustrative examples of precision and coverage are given in Figure 4.1: the curved border separates the two decisions  $f(x) = 1$  from  $f(x) = 0$ . In these examples, suppose that  $\tau$  defined in Definition 4.2.1 is fixed at 99%. Then  $A_1$  is not a valid anchor due to its precision, while  $A_2$  and  $A_3$  are. Finally,  $A_3$  is preferred to  $A_2$  because of its coverage.

As explained in [12], for most problems of interest it is intractable to compute the precision and the coverage exactly, and these probabilities have to be estimated by drawing independent samples from  $\mathcal{D}_{x_0}$ . We denote the *empirical precision* estimated from such samples by  $\hat{p}_A$ . On the other hand, we show in Section 4.2.2 that the *robustness degree* of an STL formula, whose definition is recalled in the next section, captures a notion related to coverage and can be computed exactly. We also show in Section 4.3.1 that coverage can be gradually maximized using logical implications between STL formulas, without the need to be accurately estimated.

**Remark 4.2.1.** How to choose  $\mathcal{D}_{x_0}$  is problem-specific and left essentially unspecified by the anchors methodology [12]. In practice,  $\mathcal{D}_{x_0}$  should be used to generate a sufficiently rich set of inputs around  $x_0$ . Indeed, if the all the generated samples are very close to  $x_0$ , many candidate anchors could have a precision close to 100% and the method would not be able to discriminate well among them. On the other hand, if the support of  $\mathcal{D}_{x_0}$  is so large that the behavior being explained is very rarely reproduced, it could take too long for a promising candidate anchor  $A$  to collect enough samples for  $\hat{p}_A$  to approach  $p_A$ . Since the region of inputs producing the same output  $f(x_0)$  is initially unknown, it may be delicate to choose

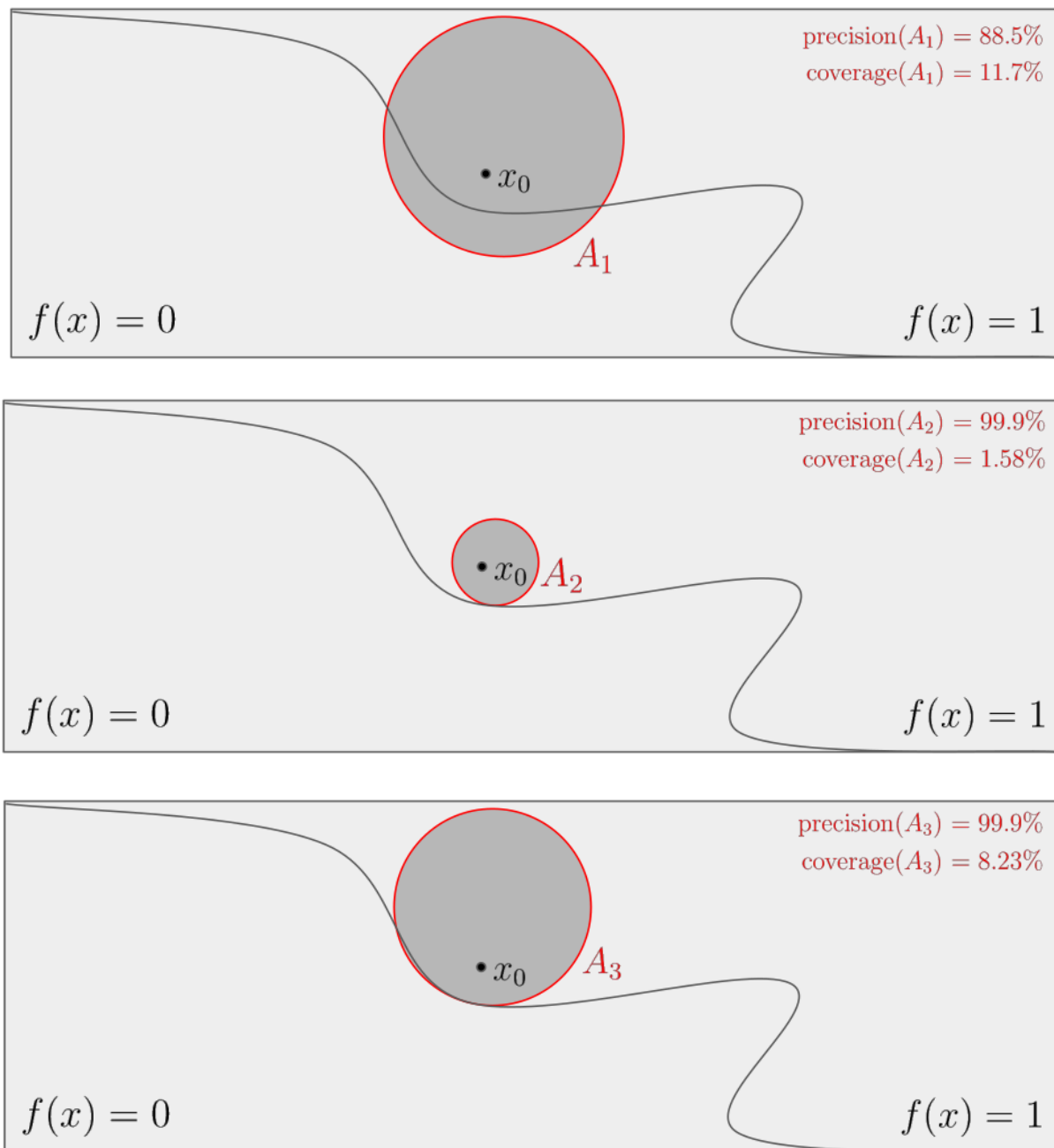


Figure 4.1 Illustration of some example explanations for an input instance  $x_0$  and their respective precision and coverage

$\mathcal{D}_{x_0}$  appropriately. In this paper however, we generate simple random perturbations of the input signal  $x_0$ , see Sections 4.4 and 4.5, and leave for future work a study of the dependence of the generated anchors with respect to the choice of  $\mathcal{D}_{x_0}$ .

### 4.2.2 Signal Temporal Logic

We consider signals  $\mathbf{s} : \mathbb{R}_+ \rightarrow \mathcal{S}_1 \times \dots \times \mathcal{S}_d$  where  $d$  denotes the signal dimension. For  $i \in \llbracket 1, d \rrbracket := \{1, \dots, d\}$ , we assume that  $\mathcal{S}_i$  is a bounded interval in  $\mathbb{R}$ , containing all real values lying between a minimum value  $\min_i$  and a maximum value  $\max_i$ . The components of a signal  $\mathbf{s}$  are denoted by  $s_i$ . The grammar of STL is recursively defined as follows:

$$\phi := \top \mid s_i > \mu \mid s_i < \mu \mid \neg\phi \mid \phi \wedge \phi \mid \phi \mathbf{U}_{[a,b]}\phi,$$

where  $\top$  represents the Boolean *true* constant;  $s_i > \mu$  and  $s_i < \mu$  are Boolean predicates with  $\mu \in \mathcal{S}_i$ ;  $\neg$  is the Boolean negation and  $\wedge$  is the Boolean conjunction; finally  $\mathbf{U}_{[a,b]}$  is the bounded temporal operator *Until* on the time interval  $[a, b]$ . Let  $\mathbf{s}$  be a signal and  $t$  a timestamp. The semantics of STL is defined recursively as follows, where  $\mathbf{s}(t) \models \varphi$  means at time  $t$ , the signal  $\mathbf{s}$  satisfies the condition  $\varphi$ :

$$\begin{aligned} \mathbf{s}(t) \models \top &\quad \Leftrightarrow \top \\ \mathbf{s}(t) \models (s_i > \mu) &\quad \Leftrightarrow \mathbf{s}_i(t) > \mu \\ \mathbf{s}(t) \models (s_i < \mu) &\quad \Leftrightarrow \mathbf{s}_i(t) < \mu \\ \mathbf{s}(t) \models \neg\varphi &\quad \Leftrightarrow \neg(\mathbf{s}(t) \models \varphi) \\ \mathbf{s}(t) \models \varphi_1 \wedge \varphi_2 &\quad \Leftrightarrow (\mathbf{s}(t) \models \varphi_1) \wedge (\mathbf{s}(t) \models \varphi_2) \\ \mathbf{s}(t) \models \varphi_1 \mathbf{U}_{[a,b]}\varphi_2 &\quad \Leftrightarrow \exists t' \in [t + a, t + b] \text{ s.t. } (\mathbf{s}(t') \models \varphi_2) \\ &\quad \wedge (\forall t'' \in [t, t'], \mathbf{s}(t'') \models \varphi_1) \end{aligned}$$

In addition, we use the two temporal operators (*Eventually*, *Globally*), which can be defined from *Until*:

$$\mathbf{F}_{[a,b]}\phi \equiv \top \mathbf{U}_{[a,b]}\phi ; \quad \mathbf{G}_{[a,b]}\phi \equiv \neg \mathbf{F}_{[a,b]}\neg\phi ,$$

whose semantics can be easily deduced:

$$\begin{aligned} \mathbf{s}(t) \models \mathbf{F}_{[a,b]}\varphi &\quad \Leftrightarrow \exists t' \in [t + a, t + b], \mathbf{s}(t') \models \varphi \\ \mathbf{s}(t) \models \mathbf{G}_{[a,b]}\varphi &\quad \Leftrightarrow \forall t' \in [t + a, t + b], \mathbf{s}(t') \models \varphi. \end{aligned}$$

In words,  $\mathbf{s}(t) \models \mathbf{F}_{[a,b]}\varphi$  (resp.  $\mathbf{s}(t) \models \mathbf{G}_{[a,b]}\varphi$ ) says that the predicate  $\varphi$  must be true at least once (resp. at all times) during  $[t+a, t+b]$  for the signal  $\mathbf{s}$ .

STL also admits *quantitative semantics* [23, 24], quantified by the *robustness degree*  $\rho$  of a formula  $\varphi$  over a signal  $\mathbf{s}$  at time  $t$ . Due to different scales between signal components, we normalize the robustness degree so that it is always contained within  $[-1, 1]$ . The semantics is then recursively defined as follows:

$$\begin{aligned} \rho(\top, \mathbf{s}, t) &= 1 \\ \rho(s_i \leq \mu, \mathbf{s}, t) &= \mp(\mathbf{s}_i(t) - \mu) / (\max_i - \min_i) \\ \rho(\neg\varphi, \mathbf{s}, t) &= -\rho(\varphi, \mathbf{s}, t) \\ \rho(\varphi_1 \wedge \varphi_2, \mathbf{s}, t) &= \min(\rho(\varphi_1, \mathbf{s}, t), \rho(\varphi_2, \mathbf{s}, t)) \\ \rho(\varphi_1 \mathbf{U}_{[a,b]}\varphi_2, \mathbf{s}, t) &= \max_{t' \in [t+a, t+b]} \min \left( \rho(\varphi_2, \mathbf{s}, t'), \min_{t'' \in [t, t']} \rho(\varphi_1, \mathbf{s}, t'') \right) \end{aligned}$$

and thus extends to:

$$\begin{aligned} \rho(\mathbf{F}_{[a,b]}\varphi, \mathbf{s}, t) &= \max_{t' \in [t+a, t+b]} \rho(\varphi, \mathbf{s}, t') \\ \rho(\mathbf{G}_{[a,b]}\varphi, \mathbf{s}, t) &= \min_{t' \in [t+a, t+b]} \rho(\varphi, \mathbf{s}, t') \end{aligned}$$

The robustness degree captures how far a signal is from violating a formula, as illustrated by the two following properties explained in [63]:

**Property 4.2.1.** (Soundness) Let  $\mathbf{s} \in \mathcal{S}$ ,  $\varphi$  an STL formula and  $t \in \mathbb{R}_+$ . If  $\rho(\varphi, \mathbf{s}, t) > 0$ , then  $\mathbf{s}(t) \models \varphi$ . If  $\rho(\varphi, \mathbf{s}, t) < 0$ , then  $\mathbf{s}(t) \not\models \varphi$ .

**Property 4.2.2.** (Correctness) Let  $\mathbf{s}, \mathbf{s}' \in \mathcal{S}$ ,  $\varphi$  an STL formula and  $t \in \mathbb{R}_+$ . Suppose  $\rho(\varphi, \mathbf{s}, t) > 0$ . If we have  $\|\mathbf{s}_i - \mathbf{s}'_i\|_\infty < (\max_i - \min_i)\rho(\varphi, \mathbf{s}, t)$  for all  $i \in \llbracket 1, d \rrbracket$ , then  $\mathbf{s}'(t) \models \varphi$ .

In Property 4.2.2, the notation  $\|u\|_\infty$  refers to the sup-norm of a scalar signal  $u$ . This property shows that the robustness degree captures a notion related to the *coverage* of Definition 4.2.2. If a rule applies to an input signal, then its robustness degree ensures that it also applies to signals that are sufficiently close, within a distance proportional to the robustness degree.

## Parametric STL and primitives

Parametric Signal Temporal Logic (PSTL) [27] replaces the parameters in STL formulas with indeterminates (to be evaluated) and thus represents the structure of the formula itself. For example, given a PSTL formula  $\psi = \mathbf{F}_{[a,b]}(s_1 > \mu)$ , the evaluation of the parameters

$\theta = \{a : 0, b : 1, \mu : 2\}$  results in the STL formula  $\psi(\theta) = \mathbf{F}_{[0,1]}(s_1 > 2)$ . Depending on the kind of system specifications one is interested in, different types of *primitives* can be used to build formulas incrementally, such as the following ones adapted from [25].

**Definition 4.2.3.** The set of first-level primitives is defined as

$$\mathcal{P}_1 := \bigcup_{i \in \llbracket 1, d \rrbracket} \left\{ \mathbf{F}_{[a,b]}(s_i \leq \mu_i), \mathbf{G}_{[a,b]}(s_i \leq \mu_i) \mid 0 \leq a \leq b, \mu_i \in \mathcal{S}_i \right\}$$

**Definition 4.2.4.** The set of second-level primitives is defined as

$$\mathcal{P}_2 := \bigcup_{i \in \llbracket 1, d \rrbracket} \left\{ \mathbf{F}_{[a,b]} \mathbf{G}_{[0,c]}(s_i \leq \mu_i), \mathbf{G}_{[a,b]} \mathbf{F}_{[0,c]}(s_i \leq \mu_i) \mid 0 \leq a \leq b, c \geq 0, \mu_i \in \mathcal{S}_i \right\}$$

In words, given that  $s_i \leq \mu$  means “the  $i$ -th signal is [smaller/greater] than  $\mu$  at the time of decision  $t$ ”, the first-level primitives provide explanations of the form: “ $s_i \leq \mu$  is [once/always] true within the time interval  $[t+a, t+b]$ ”, while for the second-level primitives,  $\mathbf{F}_{[a,b]} \mathbf{G}_{[0,c]}(s_i \leq \mu)$  specifies “within  $[t+a, t+b]$ , there is a time after which  $s_i \leq \mu$  is true for  $c$  timestamps” and  $\mathbf{G}_{[a,b]} \mathbf{F}_{[0,c]}(s_i \leq \mu)$  specifies “for all times in  $[t+a, t+b]$ ,  $s_i \leq \mu$  is true at least once within the next  $c$  timestamps”. These primitives are useful to capture certain safety, reachability and liveness properties for example.

**Remark 4.2.2.** The second-level primitives are richer than the first-level ones because we can show that  $\mathcal{P}_1 \subset \mathcal{P}_2$ . However, having more parameters increases the computation time when searching for anchors. Note also that, depending on the desired STL specifications, one is not limited to these two sets of primitives. In our current implementation and experiments, we consider only first-level primitives.

**Remark 4.2.3.** Past time STL can be used instead of standard STL when it is more natural to have explanations based on past observations. In this case, it suffices to replace the interval  $[t+a, t+b]$  with  $[t-b, t-a]$  in the above definitions.

### 4.2.3 Formal problem statement

Given an input signal  $\mathbf{s}_0 \in \mathcal{S}$ , a black-box system and a perturbation distribution  $\mathcal{D}_{\mathbf{s}_0}$ , our goal is to compute the *best* possible anchor for  $\mathbf{s}_0$  expressed in STL. As it would either be too long or impossible to generate random samples from explanations whose coverage is too small, we also impose that the robustness degree of the formulas be above a certain threshold  $\rho \geq 0$  (cf. Property 4.2.2). Finally, if multiple anchors are found, the one with the largest coverage is considered as the best anchor to be returned. More formally, given:



- $\mathcal{P}$  a set of STL primitives,
- $\underline{\rho} \in [0, 1]$  close to 0,
- $\tau \in [0, 1]$  close to 1,

identify  $\varphi = \psi_1 \wedge \dots \wedge \psi_n$ , with  $\psi_k \in \mathcal{P}$  for all  $k$ , such that at the time of decision  $t$ :

$$\rho(\varphi, \mathbf{s}_0, t) \geq \underline{\rho} \quad (4.2.3a)$$

$$\wedge \quad p_\varphi \geq \tau \quad (4.2.3b)$$

$$\wedge \quad \varphi \in \operatorname{argmax}_{p_{\varphi'} \geq \tau} \operatorname{cov}(\varphi'). \quad (4.2.3c)$$

### 4.3 Computing anchors using Monte Carlo Tree Search (MCTS)

The richness of temporal logic specifications makes it difficult to analyze every possible formula to find anchors expressed in STL. When adding a new predicate to a formula by conjunction ( $\wedge$ ), we reduce the coverage in order to increase the precision of the formula, so it is natural to proceed incrementally. In this paper, we develop a search algorithm for anchors using MCTS (see Section 4.6.3) in a Directed Acyclic Graph (DAG) whose nodes are STL formulas, where the children of an STL formula  $\varphi$  consist of its conjunction with primitive candidates  $\{\varphi \wedge \psi \mid \psi \in \mathcal{P}\}$ . As such, the moves in MCTS are primitive candidates  $\psi$ . However, some moves do not need to be considered, when the conjunction has no effect on the formula itself, e.g.,  $\mathbf{F}_{[0,1]}(s < 0) \wedge \mathbf{F}_{[0,1]}(s < 1) \equiv \mathbf{F}_{[0,1]}(s < 0)$ . We discuss in Section 4.3.1 how to effectively choose the eligible primitive candidates to conjoin with an STL formula  $\varphi$ , and we give in Section 4.3.2 the details of the MCTS algorithm, which explores and exploits formulas using a multi-step bandit sampling strategy before deciding on the best move, and recursively builds the formula until its empirical precision increases above the imposed threshold  $\tau \in [0, 1]$ . Finally, an illustrative and visualized example showing the evolution of a DAG built with MCTS is given further in Section 4.3.3.

#### 4.3.1 Primitive candidates

First, to generate a finite set of STL primitive candidates  $\psi$  for consideration at each move, we discretize the parameters  $\mu$  for signal components. Let us introduce the following notation. Let  $m \in \mathbb{N}^*$  denote a quantization stepsize. For  $x < y \in \mathbb{R}$ :

$$\operatorname{linspace}(x, y, m) := \left\{ x + k \frac{y - x}{m}, k \in \llbracket 0, m \rrbracket \right\}. \quad (4.3.1)$$

then only values in  $\operatorname{linspace}(\min_i, \max_i, m)$  are considered for  $\mu$ .

Secondly, in order to satisfy (4.2.3a), the STL primitive candidates should be robust enough, so we consider the following set of primitive candidates, which are generated once when the program starts:

$$\mathbf{cands} := \{\psi \in \mathcal{P} \mid \rho(\psi, \mathbf{s}_0, t) \geq \underline{\rho}\} \quad (4.3.2)$$

Property 4.2.1 ensures that every STL primitive  $\phi \in \mathbf{cands}$  is satisfied by the signal  $\mathbf{s}_0$ , while Property 4.2.2 ensures a minimum coverage of  $\phi$ , as it is also satisfied by the neighborhood of  $\mathbf{s}_0$  defined by  $\underline{\rho}$ . Meanwhile, the property “robustness degree greater than  $\underline{\rho}$ ” is preserved by conjunction, because

$$\rho(\phi_1 \wedge \phi_2, \mathbf{s}_0, t) = \min(\rho(\phi_1, \mathbf{s}_0, t), \rho(\phi_2, \mathbf{s}_0, t)) \geq \underline{\rho},$$

so the computed anchors will certainly verify (4.2.3a).

Finally, the children of a formula  $\varphi$  consist of its conjunction with the elements of  $\mathbf{cands}$ . However, some simplification can be made when the conjunction is redundant, due to  $\mathbf{cands}$  being partially ordered.

**Definition 4.3.1.** For two primitive candidates  $\phi_1, \phi_2 \in \mathbf{cands}$ , we say that “ $\phi_1$  implies  $\phi_2$ ” ( $\phi_1 \Rightarrow \phi_2$ ) if for all signals  $\mathbf{s}$  and all timestamps  $t$ :

$$\mathbf{s}(t) \models \phi_1 \Rightarrow \mathbf{s}(t) \models \phi_2.$$

For example, we have the following implications:

- $\mathbf{G}_{[0,3]}(s < 0) \Rightarrow \mathbf{F}_{[0,3]}(s < 0)$
- $\mathbf{G}_{[0,3]}(s < 0) \Rightarrow \mathbf{G}_{[1,2]}(s < 0)$
- $\mathbf{G}_{[0,3]}(s < 0) \Rightarrow \mathbf{G}_{[0,3]}(s < 1)$
- $\mathbf{F}_{[0,3]}(s > 1) \Rightarrow \mathbf{F}_{[1,2]}(s > 0)$

**Property 4.3.1.** If  $\phi_1, \phi_2 \in \mathbf{cands}$  verify  $\phi_2 \Rightarrow \phi_1$ , then:

- $\text{cov}(\phi_2) \leq \text{cov}(\phi_1)$
- $\phi_1 \wedge \phi_2 \equiv \phi_2$

Definition 4.3.1 and Property 4.3.1 can be naturally extended to STL formulas.

**Property 4.3.2.** Let  $\varphi$  be an STL formula and  $\phi_1, \phi_2 \in \mathbf{cands}$  such that  $\phi_2 \Rightarrow \phi_1$ . We have:

- $\varphi \wedge \phi_1 \Rightarrow \varphi$
- $\varphi \wedge \phi_2 \Rightarrow \varphi \wedge \phi_1$

- $\text{cov}(\varphi \wedge \phi_2) \leq \text{cov}(\varphi \wedge \phi_1) \leq \text{cov}(\varphi)$
- $\varphi \wedge \phi_1 \wedge \phi_2 \equiv \varphi \wedge \phi_2$

Notice from these properties that  $\varphi \wedge \phi_2$  may be a child of  $\varphi \wedge \phi_1$  in the DAG, even if they have the same number of primitives. Moreover, if  $\phi_1, \dots, \phi_n \in \mathbf{cands}$ , according to the previous properties, the children of  $\varphi = \phi_1 \wedge \dots \wedge \phi_n$  should only be its conjunction with the following primitive candidates:

$$\mathbf{cands} \setminus \bigcup_{k=1}^n \{\phi \in \mathbf{cands} \text{ s.t. } \phi_k \Rightarrow \phi\}. \quad (4.3.3)$$

### 4.3.2 MCTS algorithm

Since (4.2.3a) is necessarily verified by any conjunction of primitive candidates of  $\mathbf{cands}$ , we attempt to find an STL formula  $\varphi$  that satisfies (4.2.3b), by selecting the *best* primitive candidate incrementally using MCTS. In other words, we attempt to increase the empirical precision every time a candidate is selected, so that a precision of  $\tau$  can be eventually achieved. By building a DAG consisting of STL formulas, MCTS allows us to explore a significant number of formulas while using a bandit strategy to concentrate on the most promising ones before deciding on each move. When exploiting a node corresponding to a formula  $\varphi$ , the empirical precision is estimated by drawing random signal samples from  $\mathcal{D}$ :

$$\hat{p}_\varphi = \begin{cases} S_\varphi/N_\varphi, & \text{if } N_\varphi \neq 0, \\ -\infty, & \text{if } N_\varphi = 0, \end{cases} \quad (4.3.4)$$

where  $N_\varphi$  is the total number of sampled signals verifying  $\varphi$  and  $S_\varphi$  is the total score, i.e., counting the number of input signal samples verifying  $\varphi$  and yielding the same output as  $\mathbf{s}_0$  with the black-box system.

Algorithm 1 shows the outline of the MCTS algorithm. At each stage of the algorithm, we have a current *root node*, which we denote here  $\varphi^*$ . We start with the trivial STL formula  $\varphi^* = \top$ , whose children are exactly the primitive candidates in (4.3.2). In order to identify the next *move* from  $\varphi^*$ , i.e., the child of the root that maximizes the precision, a (typically large) number of *roll-outs* are performed to expand the DAG below  $\varphi^*$  and estimate the precision of each node. A roll-out, starting from the current root  $\varphi^*$ , consists of 4 steps, each of which is described in detail in the following: selection, expansion, simulation and back-propagation.

---

**Algorithm 1** Outlines of the MCTS algorithm
 

---

```

 $\varphi^* \leftarrow \top$ ,  $\varepsilon \leftarrow$  error threshold,  $N \leftarrow$  batch size
while  $\hat{p}_{\varphi^*} < \tau$  do
  error  $\leftarrow +\infty$ 
  while error  $> \varepsilon$  do  $\triangleright$  roll-out from  $\varphi$ 
    select a path from  $\varphi^*$  to a leaf  $L$  according to (4.3.5) at each node
    expand the leaf  $L$  with its children
    run simulations to obtain  $N$  signals and their scores (0 or 1)
    if none of the  $N$  signals verifies  $L$  then
      prune  $L$  from the tree
    end if
    for all node  $\varphi$  in path from  $\varphi^*$  to  $L$  do
      update  $N_\varphi$  and  $S_\varphi$   $\triangleright$  back-propagation
    end for
    error  $\leftarrow \varepsilon_\varphi$  where  $\varphi$  is the selected child of  $\varphi^*$  according to (4.3.5)
  end while
   $\varphi^* \leftarrow$  child of  $\varphi$  chosen according to (4.3.8)  $\triangleright$  next move
   $\hat{p}_\varphi \leftarrow S_\varphi/N_\varphi$ 
end while
while  $\varphi$  has a parent whose precision  $\geq \tau$  do  $\triangleright$  termination
   $\varphi \leftarrow$  parent of  $\varphi$  whose precision  $\geq \tau$ 
end while
return  $\varphi$ ,  $\hat{p}_\varphi$ 

```

---

## Selection

A *path* from  $\varphi^*$  to a leaf<sup>1</sup>  $L$  is selected by successively choosing child nodes based on a bandit strategy, in order to maintain the balance between exploration (nodes with few simulation) and exploitation (nodes with high empirical precision). The most classical strategy in MCTS is called UCT (Upper Confidence Bound 1 applied to trees) [31,32]. Based on [64, Chapter 7], the child of some node  $\varphi$  in the path is chosen by maximizing the following upper confidence bound:

$$\operatorname{argmax}_{\varphi' \text{ child of } \varphi} \hat{p}_{\varphi'} + \varepsilon_{\varphi'} \quad (4.3.5)$$

with the confidence width defined as

$$\varepsilon_{\varphi'} := \begin{cases} \sqrt{c_e \ln(N_\varphi)/N_{\varphi'}} & \text{if } N_{\varphi'} \neq 0, \\ +\infty & \text{if } N_{\varphi'} = 0, \end{cases} \quad (4.3.6)$$

---

<sup>1</sup>A leaf in MCTS is a node for which no simulation has ever been made.

where  $c_e$  is an exploration parameter to be tuned, usually taken equal to 2. In our implementation, we found that another strategy, called UCB1-tuned [32], tends to perform better, by setting

$$\varepsilon_{\varphi'} := \begin{cases} \sqrt{\frac{c_e \ln(N_{\varphi})}{N_{\varphi'}} \min\left(\frac{1}{4}, \hat{\sigma}_{\varphi'}^2 + \sqrt{\frac{2 \ln(N_{\varphi})}{N_{\varphi'}}}\right)} & \text{if } N_{\varphi'} \neq 0, \\ +\infty & \text{if } N_{\varphi'} = 0, \end{cases} \quad (4.3.7)$$

with  $\hat{\sigma}_{\varphi'}^2 = \hat{p}_{\varphi'}(1 - \hat{p}_{\varphi'})$ , an estimate of the precision's variance.

## Expansion

When a leaf  $L$  is reached by following the above node selection strategy, the children of  $L$ , defined by its conjunction with the primitives in (4.3.3), are incorporated into the tree, thus expanding the tree. We note that the memory requirements can be reduced by expanding the tree only when necessary during the *Selection* phase, as some nodes may never be visited twice, making it unnecessary to incorporate their children into the tree.

## Simulation

At leaf  $L$ , we sample from  $\mathcal{D}$  a fixed number of signals, say  $N$ . In practice, to define  $\mathcal{D}$ , the signals are generated around the signal  $\mathbf{s}_0$  by running simulations using random but realistic initial states and/or control signals. Each signal  $\mathbf{s}$  is stored and associated with a score  $f(\mathbf{s}) = 1$  if it produces the same output as  $\mathbf{s}_0$ , and  $f(\mathbf{s}) = 0$  otherwise. In order to ensure a sufficient coverage for  $L$ , at least one signal from the batch should satisfy  $L$ , otherwise we simply prune the leaf. The probability that none of the  $N$  signals verifies  $L$  is  $(1 - \text{cov}(L))^N$  which is upper bounded by  $(1 - c)^N$  if  $\text{cov}(L) \geq c$ . To ensure that this probability is small enough, say less than  $\gamma$ , the batch size should then satisfy  $N \geq \log(\gamma)/\log(1 - c)$ . For example, if  $\gamma = 10^{-3}$  and  $c = 0.03$ , then  $N > 226$ . In our implementation, we typically choose  $N = 256$ . Then every leaf  $L$  remaining in the tree verifies  $\mathbb{P}(\text{cov}(L) \geq c) \geq 1 - \gamma$ .

## Back-propagation

The path from  $\varphi^*$  to  $L$  is of the form  $\varphi^* =: \varphi_0 \rightarrow \varphi_1 \rightarrow \dots \rightarrow \varphi_m := L$ . For each signal  $\mathbf{s}_j$  generated by simulation,  $1 \leq j \leq N$ , we notice that if  $\mathbf{s}_j$  satisfies some  $\varphi_i$ , it necessarily satisfies all  $\varphi_l$  such that  $0 \leq l \leq i$ , because of the relationship  $\varphi_m \Rightarrow \varphi_{m-1} \Rightarrow \dots \Rightarrow \varphi_0$

(Property 4.3.2). Thus we can naturally use binary search<sup>2</sup> to find  $i^*$  such that  $\mathbf{s}_j$  satisfies all  $\varphi_l$ , for  $0 \leq l \leq i^*$ , and otherwise for all  $l > i^*$ . For  $0 \leq l \leq i^*$ , we increment all  $N_{\varphi_l}$  by 1 and  $S_{\varphi_l}$  by the score  $f(\mathbf{s}_j) \in \{0, 1\}$ . For  $l > i^*$ , we simply leave  $N_{\varphi_l}$  and  $S_{\varphi_l}$  unchanged.

### Next move

After a number of roll-outs, chosen as explained in Remark 4.3.2 below, the next move is decided among the children of the current root  $\varphi^*$ , by maximizing the lower confidence bound of their empirical precision:

$$\operatorname{argmax}_{\varphi' \text{ child of } \varphi^*} \widehat{p}_{\varphi'} - \varepsilon_{\varphi'} \quad (4.3.8)$$

where  $\varepsilon_{\varphi'}$  is the error term in either (4.3.6) or (4.3.7), depending on the chosen strategy. The reason not to choose the child node simply maximizing the empirical precision is that we want to prioritize formulas whose estimated precision is more accurate, with  $\varepsilon_{\varphi'}$  capturing this accuracy. Finally, the selected node becomes the root  $\varphi^*$  from which we restart a number of roll-outs in the same DAG, and so on, until the empirical precision of some node exceeds the imposed threshold  $\tau$ . In addition, we multiply the batch size by the number of moves, as the coverage becomes gradually smaller, making it more difficult to sample signals verifying the formula.

### Termination (maximizing coverage)

The search terminates when we find a formula after a move with empirical precision exceeding  $\tau$ . Denote such a formula  $\varphi \wedge \phi_1$ , where  $\phi_1$  corresponds to the last move of the algorithm. This formula is an anchor, but it is possible that other anchors be present among the other children of  $\varphi$ . To maximize coverage, see (4.2.3c), we start from  $\varphi \wedge \phi_1$  and iteratively search for another child  $\varphi \wedge \phi_2$  whose precision exceeds  $\tau$  (anchor) and which is a parent of  $\varphi \wedge \phi_1$  (hence has greater coverage, see Property 4.3.2), and so on, until it is no longer possible. The last found formula  $\varphi \wedge \phi_k$  is then returned by our algorithm.

**Remark 4.3.1.** In this basic setting, it is possible that the algorithm never terminates, which means that it isn't able to find an STL formula that explains the behavior of the system with sufficiently high precision. This may occur due to the primitives not being rich enough, for example, the *actual* explanation may involve primitives of  $\mathcal{P}_2$  (cf. Definition 4.2.4) but only  $\mathcal{P}_1$  (cf. Definition 4.2.3) is used. To give a stopping criterion for the algorithm, we impose a maximum depth for the tree, `max_depth`, typically 4 or 5, in order to maintain relatively short explanations, which are more understandable by humans, and limit the memory requirements.

---

<sup>2</sup>Also known as bisection search, half-interval search, logarithmic search, or binary chop.

If the explanation is already a conjunction of `max_depth` primitives (after simplification) and its empirical precision remains smaller than  $\tau$ , then the algorithm returns the best explanation found with its empirical precision.

**Remark 4.3.2.** The number of roll-outs in many applications of MCTS depends on an imposed time limit to make a decision for the next move. As the algorithm performs more roll-outs, the precision of the STL formulas is estimated more accurately. Since in our case such a time limit is not present, we instead impose a threshold  $\varepsilon$  on the confidence width  $\varepsilon_{\varphi'}$  defined in (4.3.6) or (4.3.7) to ensure that the estimated precision of the child chosen as the next move is accurate enough. A maximum number of roll-outs, `max_iter`, can also be imposed if the error converges too slowly.

### 4.3.3 Thermostat: an illustrative example

In this section, we provide a simple example to illustrate the evolution of a DAG built with the proposed algorithm using MCTS. Consider an automated thermostat which turns itself off whenever the detected temperature is once greater than  $20^\circ\text{C}$  within the past two seconds. Suppose that this mechanism is unknown to our algorithm but that we can perform as many simulations of this thermostat as we wish. The sample rate is supposed to be 1 Hz and a signal  $\mathbf{s}_0 := [19^\circ\text{C}, 21^\circ\text{C}]$  has been measured during the last two seconds<sup>3</sup>. The thermostat is thus turned off automatically, the decision “off” corresponding to the observed output for which we seek to provide an explanation. Starting from the trivial STL formula  $\top$ , we perform 15 roll-outs and explain the evolution with figures in the following. The batch size is fixed at 256.

#### 0<sup>th</sup> roll-out (Figure 4.2(a))

Starting from the trivial formula  $\top$  as the root node, we run simulations to sample 256 signals, which clearly verify  $\top$ . Among these samples, only 170 signals result in the same output as  $\mathbf{s}_0$ , i.e., make the thermostat turn off automatically. At this stage,  $S_\top = 170$ ,  $N_\top = 256$  and the empirical precision of  $\top$  is 66.41%.

#### 1<sup>st</sup> roll-out (Figure 4.2(b))

The DAG is, from the root node  $\top$ , expanded with the 8 primitive candidates, and we run simulations to obtain another batch of 256 signals. For the formula  $\varphi := \mathbf{F}_{[0,1]}(s_1 > 20^\circ\text{C})$ , where  $s_1$  corresponds to the temperature, only 252 signals verify  $\varphi$ , among which 166 produces

---

<sup>3</sup>This signal is clearly unrealistic but this example is only for illustration.

the same output as  $\mathbf{s}_0$ . Thus  $S_\varphi = 166, N_\varphi = 252$  and the empirical precision of  $\varphi$  is 65.87%.  $S_\varphi$  and  $N_\varphi$  are then back-propagated to  $\top$  to increment  $S_\top = 170 + 166 = 336$  and  $N_\top = 256 + 252 = 508$ . The other 4 signals clearly verify  $\top$ , but none of them produce the same output, thus only  $N_\top$  is incremented by 4 to reach 512.

### 2<sup>nd</sup> to 8<sup>th</sup> roll-out (Figures 4.2(c) to 4.4(c))

Note that Equation (4.3.5) prioritizes nodes for which no simulation has ever been made. Thus we successively run simulations for the remaining nodes and update the empirical precision accordingly. We can observe that, at the end of the 8<sup>th</sup> roll-out, the most promising formulas are  $\mathbf{F}_{[0,1]}(s_1 > 20^\circ\text{C})$  and  $\mathbf{F}_{[1,1]}(s_1 > 20^\circ\text{C})$ , whose empirical precision stays at 100%.

### 9<sup>th</sup> roll-out (Figure 4.5(a))

Now that we have sampled for every child of  $\top$ , the selection strategy described in Equation (4.3.5) suggests us to select again  $\mathbf{F}_{[1,1]}(s_1 > 20^\circ\text{C})$  and expand the DAG with its children, i.e., its conjunction with the 8 same primitive candidates. Compared to the other promising formula, i.e.,  $\mathbf{F}_{[0,1]}(s_1 > 20^\circ\text{C})$ , it is selected basically due to its fewer samples, thus greater uncertainty (error), in order to promote exploration. We then run simulations for a newly incorporated node, update its empirical precision and that of its ancestors, i.e.,  $\mathbf{F}_{[1,1]}(s_1 > 20^\circ\text{C})$  and  $\top$ .

### 10<sup>th</sup> roll-out (Figure 4.5(b))

Now the formula  $\mathbf{F}_{[0,1]}(s_1 > 20^\circ\text{C})$  is selected and the DAG is expanded with its conjunction with the 8 primitive candidates. These children, interestingly, also include  $\mathbf{F}_{[1,1]}(s_1 > 20^\circ\text{C})$  simply because  $\mathbf{F}_{[0,1]}(s_1 > 20^\circ\text{C}) \wedge \mathbf{F}_{[1,1]}(s_1 > 20^\circ\text{C}) = \mathbf{F}_{[1,1]}(s_1 > 20^\circ\text{C})$ . We run simulations for a new node again and update its empirical precision and that of its ancestors accordingly.

### 15<sup>th</sup> roll-out, next move (Figure 4.5(c))

Suppose at the end of the 15<sup>th</sup> roll-out, the error of the last selected child, i.e.,  $\mathbf{F}_{[1,1]}(s_1 > 20^\circ\text{C})$ , is small enough that we have enough confidence about its empirical precision. At this stage, we stop the roll-outs and choose the next move among the children of the current root node, i.e.,  $\top$ . Among all children of  $\top$ , we choose the one that maximizes the lower confidence bound for the next move, as described in Equation (4.3.8). Here, the next move corresponds to  $\varphi^* := \mathbf{F}_{[0,1]}(s_1 > 20^\circ\text{C})$ . It has the highest lower bound, principally due to



its empirical precision and its large number of samples. As its empirical precision reaches 100%, making the node an anchor for any value of  $\tau$  (which is supposed to be fixed at the beginning, say 95% or 99%), we can terminate the search and proceed to the termination step. In a more general case, if none of the nodes had a precision higher than  $\tau$ , i.e., if no anchor were found, we would transfer the current root node  $\top$  to the chosen child  $\varphi^*$ , and start again a number of roll-outs from this new root node.

### Termination (Figure 4.5(c))

The termination step consists of maximizing the coverage by climbing up the DAG from the new root node while remaining an anchor. There is nothing to do in this example, because the only ancestor of  $\mathbf{F}_{[0,1]}(s_1 > 20^\circ\text{C})$  is  $\top$  which is not an anchor due to its empirical precision. Had, for instance,  $\mathbf{F}_{[1,1]}(s_1 > 20^\circ\text{C})$  been the chosen node with a 100% empirical precision, we would have climbed up the DAG and found the other anchor  $\mathbf{F}_{[0,1]}(s_1 > 20^\circ\text{C})$ . It has a greater coverage due to the logical implication between them, according to Property 4.3.2. The algorithm ends and returns  $\mathbf{F}_{[0,1]}(s_1 > 20^\circ\text{C})$  with its 100% empirical precision. In words, this explanation says: “the thermostat is off because the temperature is above  $20^\circ\text{C}$  once between the 0<sup>th</sup> second and the 1<sup>st</sup> second”, corresponding exactly to how the thermostat works (unknown to the algorithm).


## 4.4 Case study: automatic transmission

In this section and the following, where we present simulated case studies, we assume that the simulated model is totally unknown (black-box) to our algorithm. The experiments were run on Linux with an Intel i7-7700K CPU. The code is developed in Python 3.8 and not optimized for efficiency.

### 4.4.1 Background

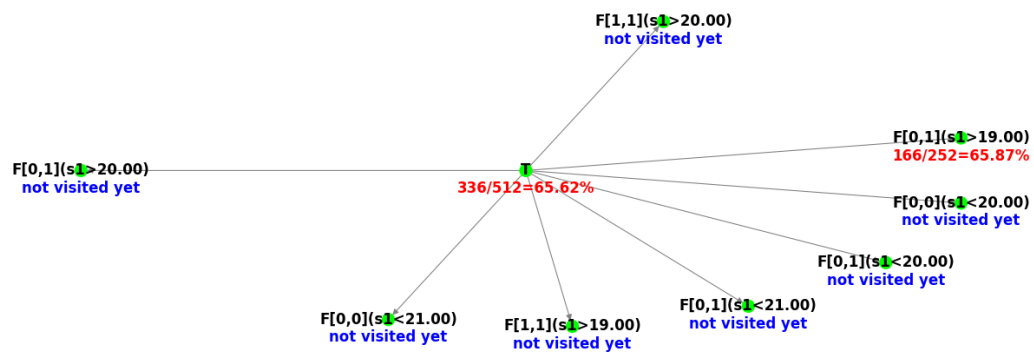
We consider an automotive automatic transmission model proposed in [8] as a benchmark for hybrid system verification and widely used as case study, see, e.g., [65, 66]. This model includes one discrete state variable,  $\text{gear} \in \{1, 2, 3, 4\}$ , and three continuous state variables: the engine speed ( $\text{espd}$ , in rpm), the vehicle speed ( $\text{vspd}$ , in mph) and the throttle opening percentage ( $\text{throttle} \in [0, 1]$ ). The closed-loop vehicle model includes an automatic transmission controller that shifts the gear based on the current gear setting, vehicle speed and throttle opening. The controller we implemented is an approximation of that proposed in [9], according to the schedule shown in Figure 4.6. The engine speed depends on the gear and

0 rollout

  
**170/256=66.41%**

(a) After the 0<sup>th</sup> roll-out

1 rollout

(b) After the 1<sup>st</sup> roll-out

2 rollouts

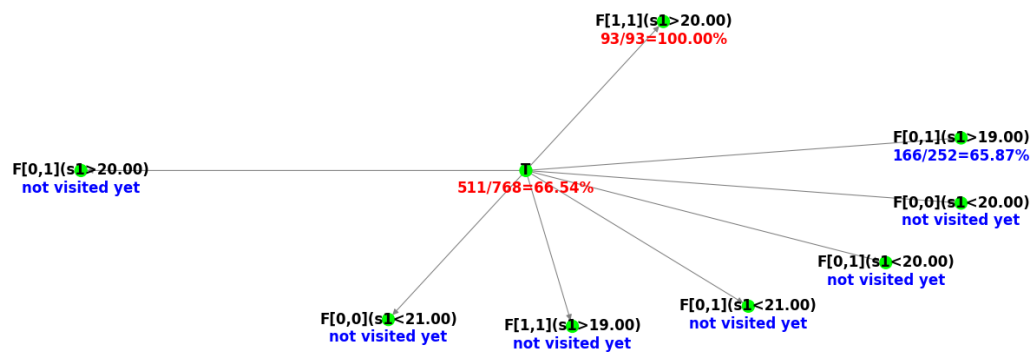
(c) After the 2<sup>nd</sup> roll-out

Figure 4.2 After roll-outs 0 to 2

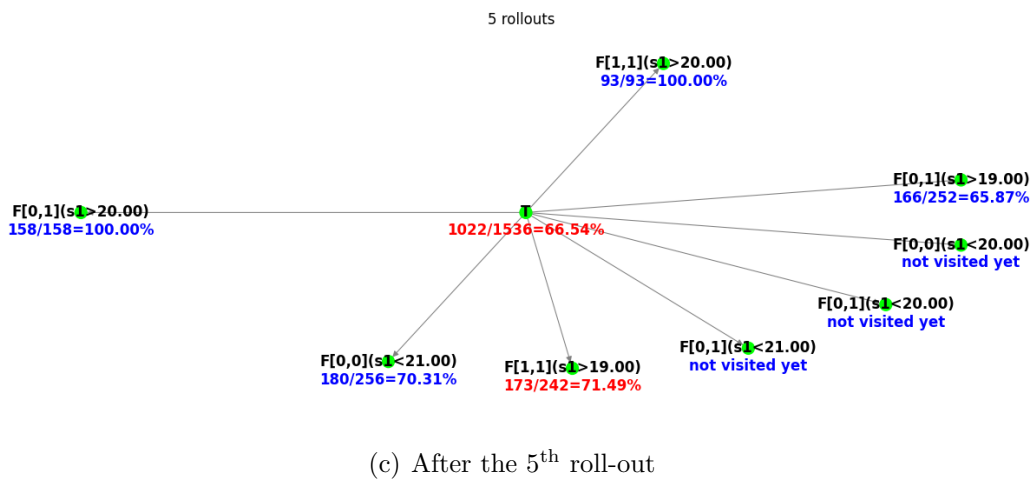
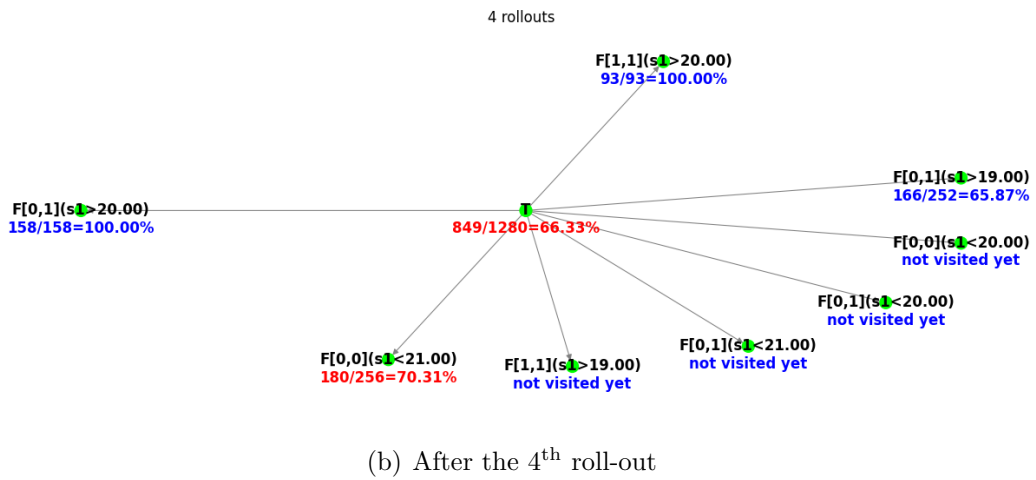
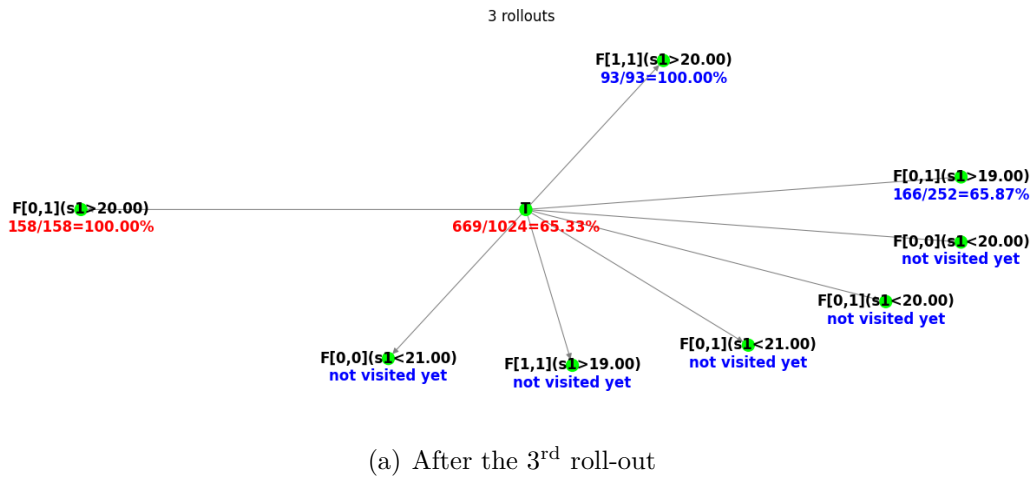


Figure 4.3 After roll-outs 3 to 5

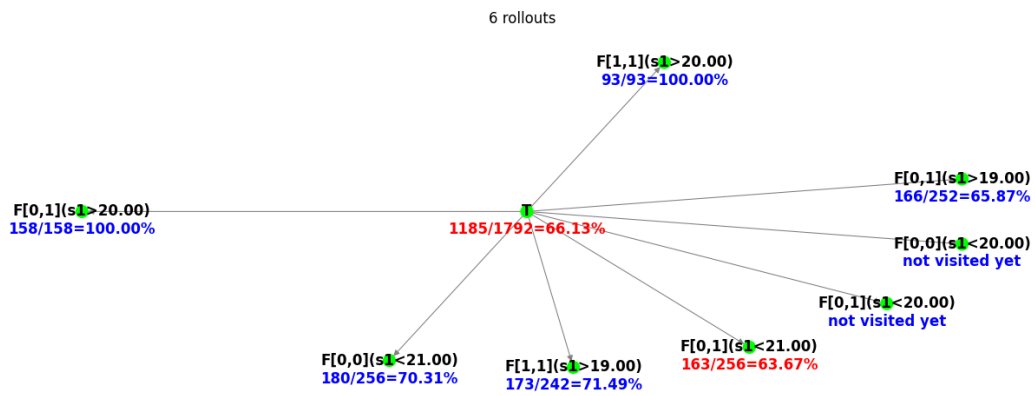
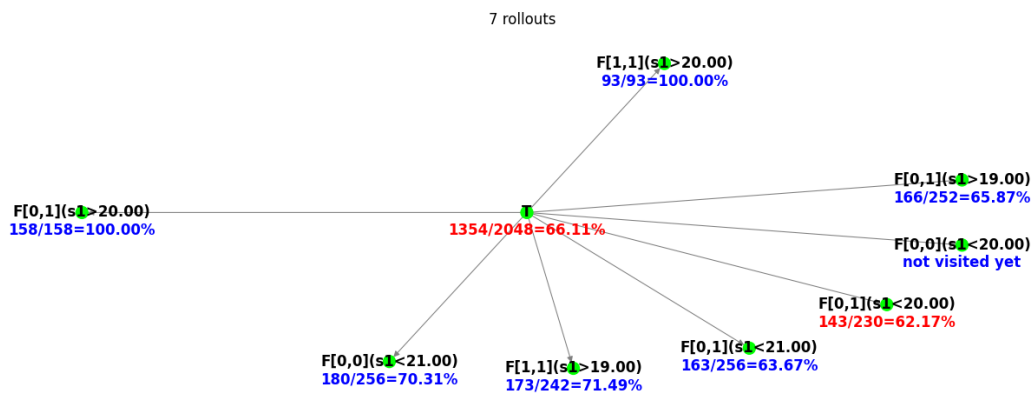
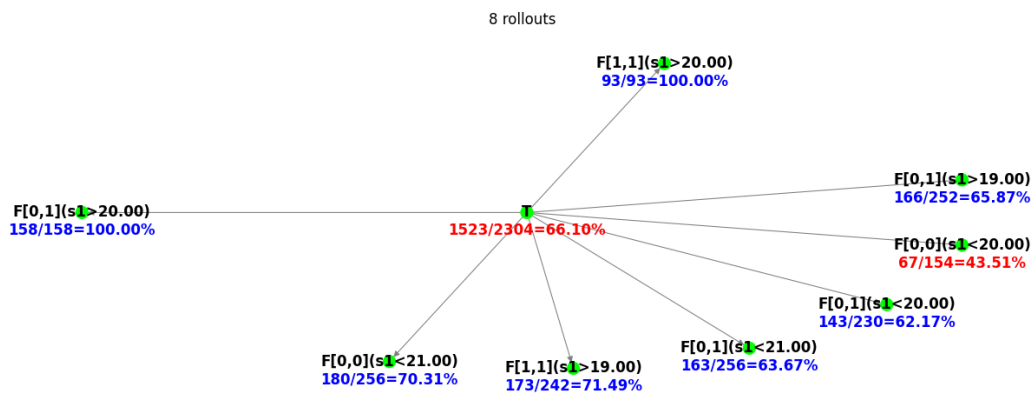
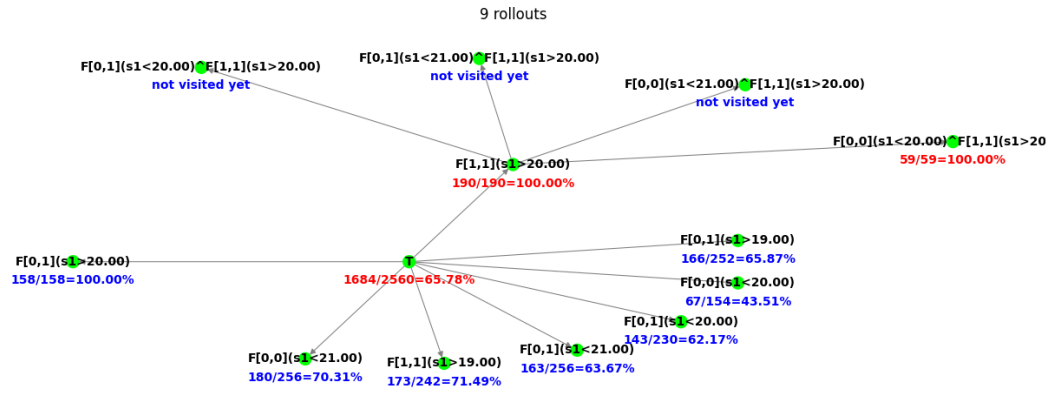
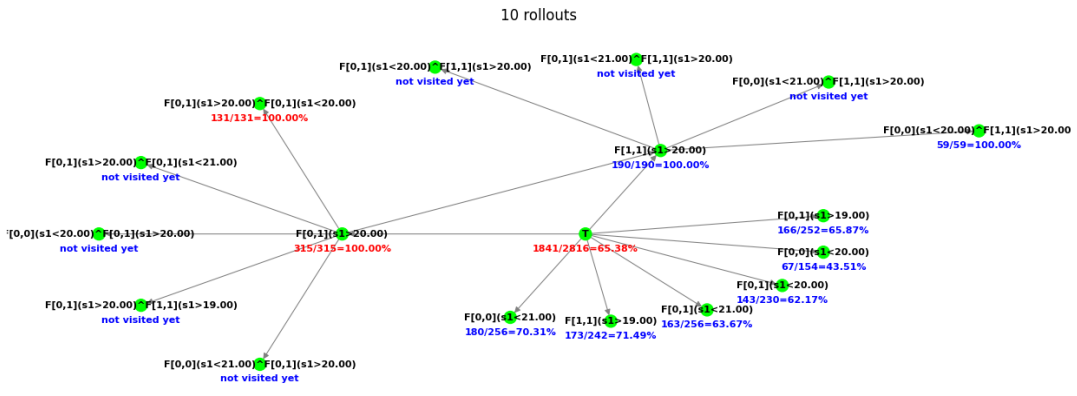
(a) After the 6<sup>th</sup> roll-out(b) After the 7<sup>th</sup> roll-out(c) After the 8<sup>th</sup> roll-out

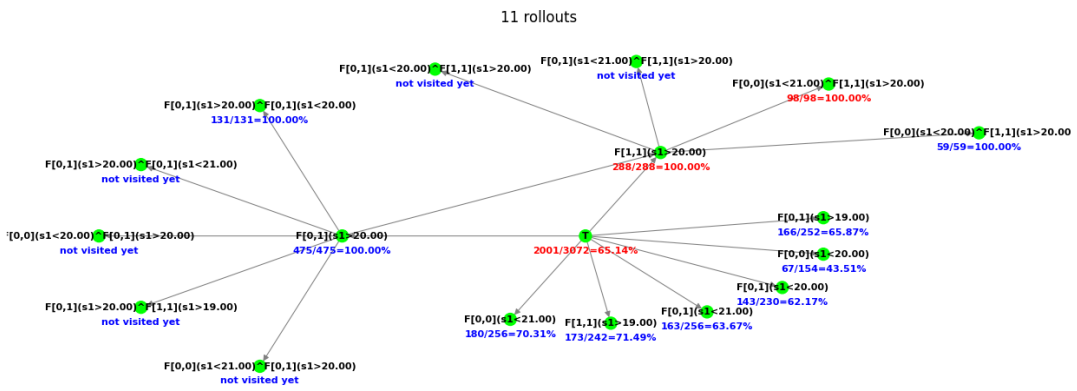
Figure 4.4 After roll-outs 6 to 8



(a) After the 9<sup>th</sup> roll-out



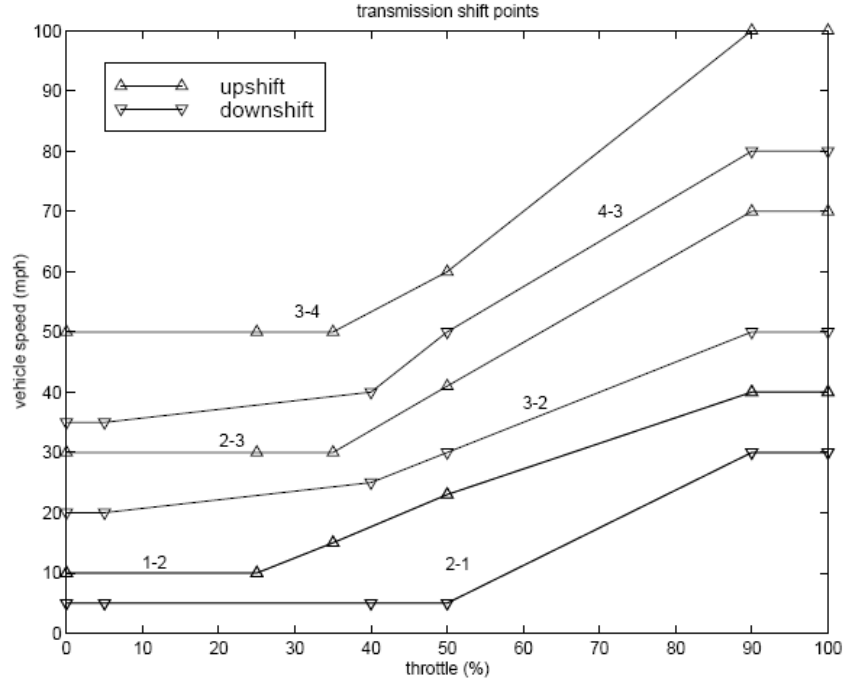
(b) After the 10<sup>th</sup> roll-out



(c) After the 15<sup>th</sup> roll-out: next move & termination

Figure 4.5 After roll-outs 9, 10 and 15

the vehicle speed, while the vehicle speed changes according to an acceleration signal that depends on the engine torque, the gear, the throttle opening, the road slope, the rolling friction and the aerodynamic drag. In this case study, the road is assumed flat and the vehicle is initially stationary with the first gear engaged.



Source: [9]

Figure 4.6 Automatic transmission shift points

#### 4.4.2 Explaining an STL-based monitoring system

To evaluate and validate the proposed algorithm, we consider five of the requirements on the transmission system proposed in [8], expressed in STL:

- $\varphi_1 = \mathbf{G}_{[0,10]}(\text{espd} < 4750)$
- $\varphi_2 = \mathbf{G}_{[0,20]}(\text{vspd} < 120)$
- $\varphi_3 = (\mathbf{G}_{[0,30]}(\text{espd} < 3000) \rightarrow \mathbf{G}_{[0,4]}(\text{vspd} < 35))$
- $\varphi_4 = (\mathbf{G}_{[0,30]}(\text{espd} < 3000) \rightarrow \mathbf{G}_{[0,8]}(\text{vspd} < 50))$
- $\varphi_5 = (\mathbf{G}_{[0,30]}(\text{espd} < 3000) \rightarrow \mathbf{G}_{[0,20]}(\text{vspd} < 65))$

Suppose that a monitoring system triggers an alarm when a requirement is violated. We consider each of these monitoring systems as a black-box model and aim to *recover* the formulas defining them. In other words, we try to explain why the alarm has been triggered

just by observing a signal  $\mathbf{s}_0$ . The report [10] provides some signals that violate the above requirements. Figure 4.7(a) shows a signal violating  $\varphi_1$ , as the engine speed exceeds 4750 rpm within 10 seconds. Similarly, Figure 4.7(b) shows a signal violating  $\varphi_2$ , as the vehicle speed exceeds 120 mph within 20 seconds. Finally, the signal shown in Figure 4.7(c) violates simultaneously  $\varphi_3, \varphi_4$  and  $\varphi_5$ . These signals were generated by adjusting the throttle opening over time.

## Results and Discussion

For the five benchmarks, the chosen hyper-parameters for our algorithm are shown in Table 4.1. The perturbation distribution  $\mathcal{D}$  is specified by changing the throttle randomly every second as specified in the table. The initial state is kept unchanged since the vehicle is supposed to be initially stationary. We choose the precision threshold to be  $\tau = 100\%$  because this case study is meant to validate our algorithm, and we know *a priori* that the monitoring system is based on an STL formula, which moreover can be recovered exactly using the primitives  $\mathcal{P}_1$ . The results are shown in Table 4.2, where `1inspace` is defined in (4.3.1). For the five benchmarks, the anchors returned by our algorithm correspond exactly to their negation, since the alarm is triggered when the requirements are violated, except for  $\varphi_3$  where  $\mathbf{G}_{[2,30]}(\text{espd} < 3000)$  is almost equivalent to  $\mathbf{G}_{[0,30]}(\text{espd} < 3000)$ , as the engine speed may not surpass 3000 rpm in 2 seconds. This demonstrates the capability of the algorithm of retrieving the correct ground-truth formula, while exploring among conjunctions of thousands of primitives. Interestingly, if at the end of the algorithm we hadn't gradually increased the coverage by climbing up the DAG (see the *Termination* step of the MCTS algorithm description), we wouldn't have obtained the exact formula, as illustrated in Table 4.3 where the intermediate results for  $\varphi_3, \varphi_4$  and  $\varphi_5$  are given. Here, a number of moves are made until the empirical precision arrives at  $\tau = 100\%$  (before dashed line), and starting from the first chosen anchor, we gradually increase the coverage by climbing up the DAG, while ensuring that the precision stays at 100%.

### 4.4.3 Explaining the transmission system during a passing maneuver

We now focus on a scenario where the vehicle is performing a passing maneuver. Suppose that we observe the input signal  $\mathbf{s}_0$  given by the first 3 plots in Figure 4.8 and the output of the transmission system given by the last plot. Note that initially the vehicle is accelerating with the throttle linearly decreasing from 60% to 40%, up-shifting the vehicle to the 4<sup>th</sup> gear. At the 12<sup>th</sup> second, the throttle is suddenly pressed to 100%, making the transmission system down-shift to the 3<sup>rd</sup> gear. The shifting schedule of the transmission system, shown in Figure

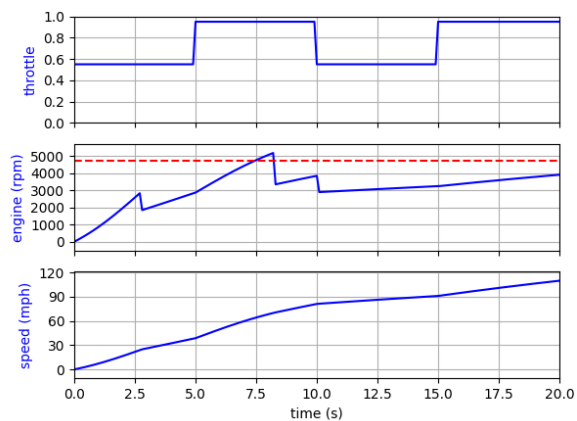
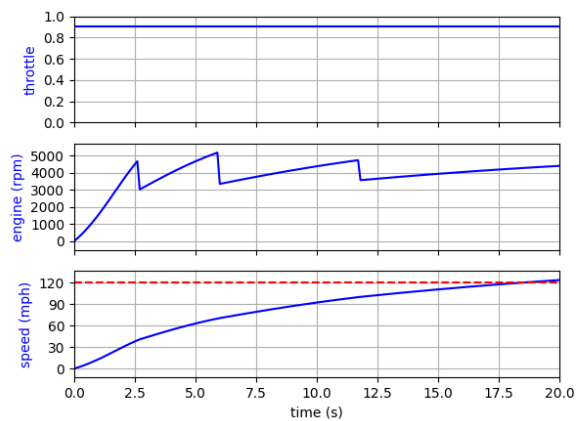
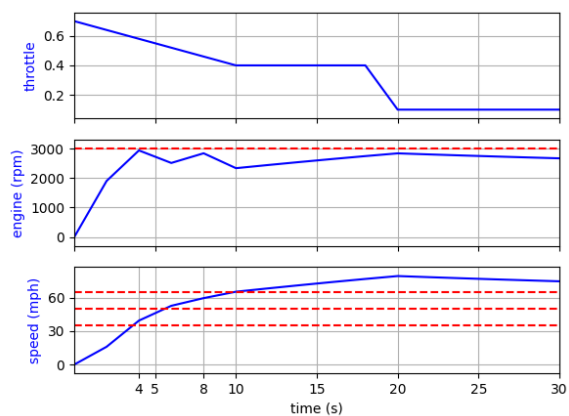
(a) Signal violating  $\varphi_1$ (b) Signal violating  $\varphi_2$ (c) Signal violating  $\varphi_3, \varphi_4$  and  $\varphi_5$ Figure 4.7 Signals violating  $\varphi_1$  to  $\varphi_5$



Table 4.1 Hyper-parameters for explaining the STL-based monitoring system using each of the five benchmarks

hyper-parameter	$\varphi_1$	$\varphi_2$	$\varphi_3, \varphi_4, \varphi_5$
$\tau$	100%	100%	100%
$\rho$	0	0	0
$\varepsilon$	2.0%	2.0%	1.0%
batch_size	256	256	256
max_depth	4	4	4
max_iter	40000	40000	40000
$\mu_{\text{espd}} \in$	linspace(0, 6000, 24)	linspace(0, 5000, 5)	linspace(0, 4000, 4)
$\mu_{\text{vspd}} \in$	linspace(0, 160, 8)	linspace(0, 160, 8)	linspace(0, 70, 14)
random throttle distribution	uniform between 0% and 100%	uniform between 70% and 100%	uniform $\pm 40\%$ from original

Table 4.2 Explanations returned by our algorithm for the STL-based monitoring system using each of the five benchmarks

STL	number of primitives	anchor	empirical precision	execution time
$\varphi_1$	7680	$\mathbf{F}_{[0,10]}(\text{espd} > 4750)$	100%	00:35:46
$\varphi_2$	4851	$\mathbf{F}_{[0,20]}(\text{vspd} > 120)$	100%	00:22:15
$\varphi_3$	4096	$\mathbf{G}_{[2,30]}(\text{espd} < 3000) \wedge \mathbf{F}_{[0,4]}(\text{vspd} > 35)$	100%	02:20:13
$\varphi_4$	4096	$\mathbf{G}_{[0,30]}(\text{espd} < 3000) \wedge \mathbf{F}_{[0,8]}(\text{vspd} > 50)$	100%	03:01:14
$\varphi_5$	4096	$\mathbf{G}_{[0,30]}(\text{espd} < 3000) \wedge \mathbf{F}_{[0,20]}(\text{vspd} > 65)$	100%	06:56:46

Table 4.3 Intermediate results for the STL-based monitoring systems using  $\varphi_3, \varphi_4$  and  $\varphi_5$

STL	Intermediate results	Remark	Precision
$\varphi_3$	$\top \rightsquigarrow \mathbf{G}_{[2,30]}(\text{espd} < 3000)$	1 <sup>st</sup> move	35.97%
	$\rightsquigarrow \mathbf{G}_{[2,30]}(\text{espd} < 3000) \wedge \mathbf{G}_{[4,26]}(\text{vspd} > 35)$	2 <sup>nd</sup> move	100.00%
	$\rightsquigarrow \mathbf{G}_{[2,30]}(\text{espd} < 3000) \wedge \mathbf{G}_{[4,18]}(\text{vspd} > 35)$	termination	
	$\rightsquigarrow \mathbf{G}_{[2,30]}(\text{espd} < 3000) \wedge \mathbf{G}_{[4,16]}(\text{vspd} > 35)$		
	$\rightsquigarrow \mathbf{G}_{[2,30]}(\text{espd} < 3000) \wedge \mathbf{G}_{[4,8]}(\text{vspd} > 35)$		
	$\rightsquigarrow \mathbf{G}_{[2,30]}(\text{espd} < 3000) \wedge \mathbf{F}_{[4,4]}(\text{vspd} > 35)$		
	$\rightsquigarrow \mathbf{G}_{[2,30]}(\text{espd} < 3000) \wedge \mathbf{F}_{[2,4]}(\text{vspd} > 35)$		
	$\rightsquigarrow \mathbf{G}_{[2,30]}(\text{espd} < 3000) \wedge \mathbf{F}_{[0,4]}(\text{vspd} > 35)$	final result	100.00%
$\varphi_4$	$\top \rightsquigarrow \mathbf{G}_{[0,30]}(\text{espd} < 3000)$	1 <sup>st</sup> move	34.52%
	$\rightsquigarrow \mathbf{G}_{[0,30]}(\text{espd} < 3000) \wedge \mathbf{F}_{[2,8]}(\text{vspd} > 50)$	2 <sup>nd</sup> move	100.00%
	$\rightsquigarrow \mathbf{G}_{[0,30]}(\text{espd} < 3000) \wedge \mathbf{F}_{[0,8]}(\text{vspd} > 50)$	final result	100.00%
$\varphi_5$	$\top \rightsquigarrow \mathbf{G}_{[0,30]}(\text{espd} < 3000)$	1 <sup>st</sup> move	61.53%
	$\rightsquigarrow \mathbf{G}_{[0,30]}(\text{espd} < 3000) \wedge \mathbf{F}_{[4,20]}(\text{vspd} > 65)$	2 <sup>nd</sup> move	100.00%
	$\rightsquigarrow \mathbf{G}_{[0,30]}(\text{espd} < 3000) \wedge \mathbf{F}_{[0,20]}(\text{vspd} > 65)$	final result	100.00%

4.6, is assumed unknown. We attempt to find automatically a (local) rule explaining why the system engaged the 3<sup>rd</sup> gear at the 12<sup>th</sup> second, by analyzing the throttle opening, the engine speed and the vehicle speed in the previous seconds, using PtSTL.

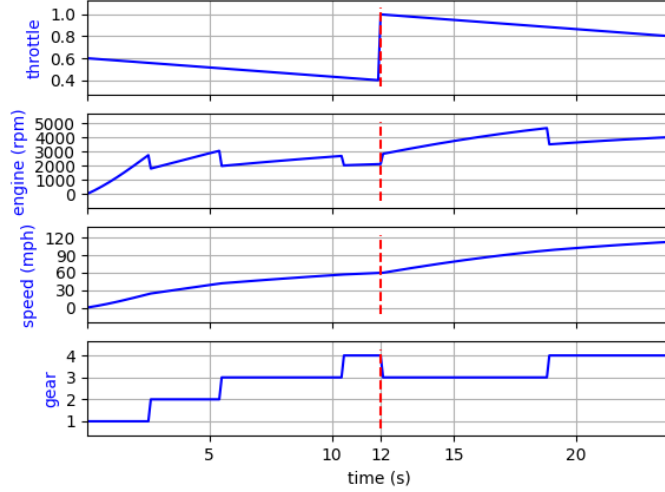


Figure 4.8 Simulation of the automatic transmission system during a passing maneuver

## Results and Discussion

To define the perturbation distribution  $\mathcal{D}$ , the throttle is changed to a random and uniform value between 0% and 100% every second, while the initial state remains unchanged. The chosen hyper-parameters for our algorithm are shown in Table 4.4. We use PtSTL and choose to analyze only the 4 previous seconds. As a result, 450 primitives were generated, and we obtained after 4 moves the following anchor:

$$\mathbf{F}_{[-1,-1]}(\mathbf{throttle} > 0.9) \wedge \mathbf{G}_{[-4,-2]}(\mathbf{vspd} > 45) \wedge \mathbf{F}_{[-4,-1]}(\mathbf{vspd} < 70) \quad (4.4.1)$$

achieving a precision of 99.41%. The formulas after each move along with their respective empirical precision, number of roll-outs and execution time are shown in Table 4.5, where the final formula after the dashed line is obtained by increasing the coverage by climbing up the DAG. In words, the predicates of the explanation (4.4.1) mean that:

- $\mathbf{F}_{[-1,-1]}(\mathbf{throttle} > 0.9)$ : the throttle was opened more than 90% in the last second, before down-shifting to the 3<sup>rd</sup> gear;
- $\mathbf{G}_{[-4,-2]}(\mathbf{vspd} > 45)$ : four to two seconds before down-shifting, the vehicle speed was greater than 45 mph;

- $\mathbf{F}_{[-4,-1]}(\text{vspd} < 70)$ : the vehicle speed has once been lower than 70 mph during the 4 previous seconds.

If we look at these conditions on the shifting schedule of Figure 4.6 (which is unknown to our algorithm), between the 8<sup>th</sup> and 12<sup>th</sup> second, the vehicle was moving faster than 45 mph and up-shifted from the 3<sup>rd</sup> gear to the 4<sup>th</sup> (see also Figure 4.8). Then, the throttle was opened to the maximum and thus the vehicle crossed the “4-3” line and down-shifted to the 3<sup>rd</sup> gear. However, if the speed had exceeded 70 mph, the 4<sup>th</sup> gear would have remained engaged, and if the speed had been lower than 45 mph, the system would have been directly down-shifted to the 2<sup>nd</sup> gear. Thus, although limited to local explanations, our algorithm is able to provide some insight on the boundaries separating the controller’s decisions.

Table 4.4 Hyper-parameters for explaining the engagement of the 3<sup>rd</sup> gear during the passing maneuver

hyper-parameter	value
$\tau$	99%
$\rho$	0.01
$\varepsilon$	0.75%
batch_size	256
max_depth	4
max_iter	50000
$\mu_{\text{espd}} \in$	<code>linspace(0, 5000, 5)</code>
$\mu_{\text{vspd}} \in$	<code>linspace(0, 80, 16)</code>
$\mu_{\text{throttle}} \in$	<code>linspace(0, 1, 10)</code>

Table 4.5 The STL formulas after each move to explain the engagement of the 3<sup>rd</sup> gear during the passing maneuver

move	STL formula	empirical precision	number of roll-outs	execution time
1	$\mathbf{F}_{[-1,-1]}(\text{throttle} > 0.9)$	91.18%	2376	00:03:22
2	$\mathbf{F}_{[-1,-1]}(\text{throttle} > 0.9) \wedge \mathbf{G}_{[-4,-2]}(\text{vspd} > 45)$	95.63%	5790	00:13:31
3	$\mathbf{F}_{[-1,-1]}(\text{throttle} > 0.9) \wedge \mathbf{G}_{[-4,-2]}(\text{vspd} > 45) \wedge \mathbf{F}_{[-4,-1]}(\text{vspd} < 65)$	99.99%	15912	00:56:47
termination	$\bar{\mathbf{F}}_{[-1,-1]}(\text{throttle} > 0.9) \wedge \mathbf{G}_{[-4,-2]}(\text{vspd} > 45) \wedge \mathbf{F}_{[-4,-1]}(\text{vspd} < 70)$	99.41%	–	–

## 4.5 Case study: ACAS Xu

### 4.5.1 Background

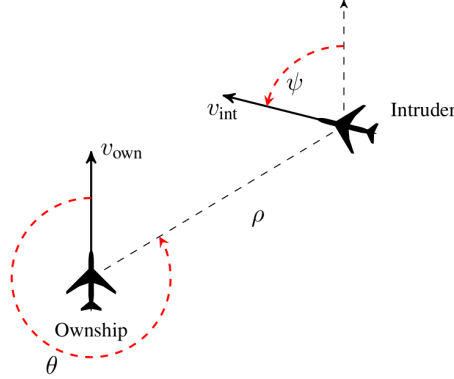
ACAS Xu [67] is a system implementing the decision making logic of an Airborne Collision Avoidance System (ACAS) specifically for unmanned aerial vehicles. It uses dynamic programming to provide maneuver guidance maintaining horizontal and vertical separation between two aircraft. The original version of ACAS Xu consists of a large lookup *score table* containing millions of discrete states, requiring several GB of floating point storage. Julian et al. [62] introduced a compression method relying on deep neural networks to approximate the original score table, trained using supervised learning, which requires only a few MB of memory. It stores only the weights of the neural network instead of the discrete states and is able to provide guidance instructions consistent with the ones recommended by the original table, without losing much performance. According to the description provided in [62, 68], the system takes as input the following 7-dimensional signal obtained from sensor measurements to issue one of 5 possible action advisories. Each advisory is assigned a score, with the lowest score corresponding to the best advisory.

- Inputs:**
1.  $\rho$  (m): distance from ownship to intruder;
  2.  $\theta$  (rad): angle to intruder relative to ownship heading direction;
  3.  $\psi$  (rad): heading angle of intruder relative to ownship heading direction;
  4.  $v_{own}$  (m/s): speed of ownship;
  5.  $v_{int}$  (m/s): speed of intruder;
  6.  $\chi$  (s): time until loss of vertical separation;
  7.  $a_{prev}$ : previous advisory.

- Outputs:**
1. Clear Of Conflict (COC):  $0^\circ/\text{s}$ ;
  2. Weak Left Turn (WLT):  $1.5^\circ/\text{s}$ ;
  3. Weak Right Turn (WRT):  $-1.5^\circ/\text{s}$ ;
  4. Strong Left Turn (SLT):  $3.0^\circ/\text{s}$ ;
  5. Strong Right Turn (SRT):  $-3.0^\circ/\text{s}$ .

The geometry of the input parameters is illustrated in Figure 4.9. The authors of [62] trained 45 deep neural networks for ACAS Xu by discretizing  $\chi$  (9 discrete values) and  $a_{prev}$  (5 advisories), each network receiving the 5 other inputs and producing the 5 scores assigned to the possible outputs. These networks are fully connected, use ReLU activation functions and have 6 hidden layers. The ACAS Xu neural networks have been used extensively

as a benchmark for *neural network verification* algorithms, as they contain a fairly small number (300) of neurons. For example, 200 network verification queries were proposed in [68], consisting of preconditions on the inputs and postconditions on the outputs.



Source: [62]

Figure 4.9 Geometry for the ACAS Xu horizontal logic table

In our framework, explanations for an output guidance advisory can be based on past values of the input signals, up to some maximum history length. In order to obtain realistic input data, we simulated scenarios involving 2 aircraft, assuming  $v_{own} = 300$  m/s and  $v_{int} = 100$  m/s were constant and  $\chi = 0$  for simplicity. Given the initial input data  $\rho, \theta, \psi$ , the simulator iterates at each timestamp by invoking the corresponding neural network and providing a new advisory, allowing the ownship to turn by the corresponding angle.

#### 4.5.2 Explaining an advisory change

In the scenario illustrated on Figure 4.10, the ownship aircraft is equipped with the ACAS Xu system, initially heading left. The red trajectory indicates a (continuous) SRT, the yellow indicates a WRT, while the green corresponds to COC. The initial state is the following:  $\rho = 5000$  (m),  $\theta = \pi/4$ ,  $\psi = -\pi/2$ ,  $v_{own} = 300$  (m/s),  $v_{int} = 100$  (m/s),  $\chi = 0$  and  $a_{prev} =$  SRT. The system issued an SRT advisory for the ownship from the very beginning during 10 seconds, and switched to WRT and finally COC when the two aircraft were no longer in danger of colliding with each other. In this case, we attempt to find an explanation, expressed in PtSTL, for why the advisory switched from SRT to WRT at the 10<sup>th</sup> second, by analyzing the signals  $\rho, \theta, \psi$  during the previous seconds.

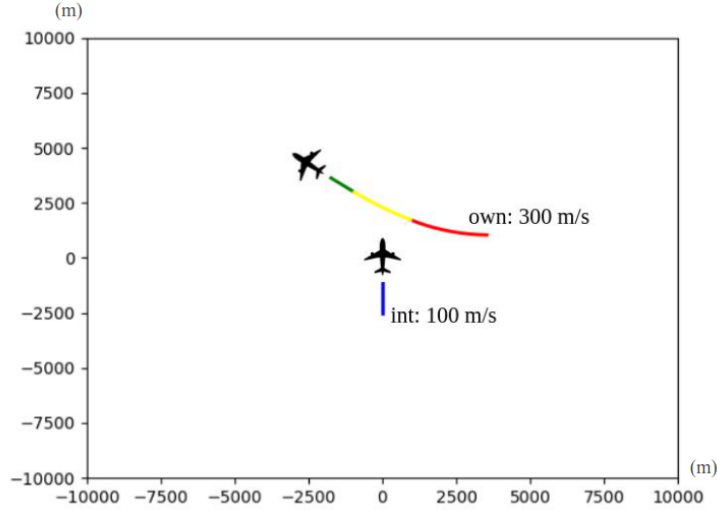


Figure 4.10 Simulation of the ACAS Xu system

## Results and Discussion

To define the perturbation distribution  $\mathcal{D}$ , we ran simulations by randomizing the initial state in the neighborhood of the original one. More precisely, the initial state components that are not assumed constant were uniformly sampled from the following intervals:  $\rho \in [2000, 8000]$  (m),  $\theta \in [0, \pi]$ ,  $\psi \in [-\pi, 0]$ . The chosen hyper-parameters for our algorithm are shown in Table 4.6. In our experiment, we chose to analyze only the 4 seconds before the advisory change. 448 primitives were generated and the anchor returned by the algorithm was the following:

$$\mathbf{F}_{[-4,-2]}(\theta > 1.57) \wedge \mathbf{F}_{[-4,-1]}(\rho > 3500) \quad (4.5.1)$$

achieving an empirical precision of 100.00% after 2 moves. Similarly to the previous case study, the formulas after each move and their respective empirical precision, number of roll-outs and execution time are shown in Table 4.7. Analyzing the formula (4.5.1) provides an intuitive explanation for a human user to understand the criteria (e.g., thresholds for certain inputs) according to which the neural network may consider a reduced danger level and issue a change of advisory from SRT to WRT:

- $\mathbf{F}_{[-4,-2]}(\theta > 1.57)$ : four to two seconds before the advisory change, the intruder was on the left side behind the ownship;
- $\mathbf{F}_{[-4,-1]}(\rho > 3500)$ : before issuing the WRT, the distance was once greater than 3500 meters.

Table 4.6 Hyper-parameters for explaining the switch from SRT to WRT

hyper-parameter	value
$\tau$	98%
$\rho$	0.01
$\varepsilon$	0.75%
batch_size	256
max_depth	4
max_iter	50000
$\mu_\rho \in$	<code>linspace(0, 8000, 8)</code>
$\mu_\theta \in$	<code>linspace(0, <math>\pi</math>, 8)</code>
$\mu_\psi \in$	<code>linspace(-<math>\pi</math>, 0, 8)</code>

Table 4.7 STL formulas after each move for explaining the the switch from SRT to WRT

move	STL formula	empirical precision	number of roll-outs	execution time
1	$\mathbf{F}_{[-4,-2]}(\theta > 1.57)$	91.23%	1762	00:10:05
2	$\mathbf{F}_{[-4,-2]}(\theta > 1.57)$ $\wedge \mathbf{F}_{[-4,-1]}(\rho > 3500)$	100.00%	33325	04:00:52

## 4.6 Related work

### 4.6.1 Interpretable Machine Learning

The monograph [11] surveys the area of *interpretable machine learning*, where methods can be categorized as either model-specific or model-agnostic. Model-specific methods apply to specific types of machine learning models. For example, class activation maps (CAM [16] and Grad-CAM [17]) provide *visual explanations* for the decisions made by Convolutional Neural Network (CNN) in computer vision tasks such as image classification and object detection, by identifying the input pixels that most influence a classification decision. For this, Grad-CAM requires the architecture of the specific CNN under study, in order to compute the gradient information flowing into the last convolutional layer.

Model-agnostic methods, on the other hand, offer additional flexibility since they can be applied to *any* model, which is treated as a black box transforming inputs to outputs. Global model-agnostic methods try to characterize the average behavior of the model, i.e., the expected output values based on the input data distribution, while local methods explain a model only in a certain region of the input space, e.g., to analyze individual predictions or decisions. This paper focuses on one important local method called *anchors* [12], which

we extend to explain the behavior of dynamic systems by incorporating temporal properties. Anchors aim to identify conditions on the input features under which a given prediction or decision, for which an explanation is requested, remains essentially the same. These *rule-based* explanations, using the terminology of [19], provide explicit decision boundaries between the inputs leading to a given prediction and those leading to a different one. More details on this method is provided in Section 4.2.1. Other local methods include, among others, LIME [20], which locally approximates a black-box model by a learned interpretable model, e.g., a small linear model or decision tree, and SHAP [69], which uses a game-theoretic approach to assign a (Shapley) value [70] to each input feature contributing to a particular prediction of the model.

#### 4.6.2 Temporal Logic-based inference

This paper focuses specifically on dynamic scenarios with black-box models processing signals over time. To specify desired or undesired properties of a dynamical system’s behavior in a way that is unambiguous yet easily interpretable by human operators, various types of *temporal logic* can be used [22], including Signal Temporal Logic (STL) [21]. Conversely, one can try to search for STL formulas that best classify observed behaviors into distinct groups, e.g., anomalous vs. normal signals [26]. This generally involves searching for an appropriate formula structure, usually expressed in Parametric Signal Temporal Logic (PSTL) [27], and for the best parameters contained in a given PSTL formula, by optimizing a certain classification criterion [26].

Among the work inferring both the formula structure and the parameters, Bombara *et al.* [28] draw connections between STL formulas and decision trees, and build the latter incrementally in order to classify a given set of signals using STL in an offline or online fashion. Kong *et al.* [26] further exploit the partial order existing between formula structures by constructing a Directed Acyclic Graph (DAG) and use a Support Vector Machine (SVM) for parameter optimization. In [71], complicated formulas including the *Until* operator are learnt by using a genetic algorithm to infer the formula structure and the GP-UCB algorithm [72] to estimate the parameters.

The aforementioned efforts are in the spirit of the literature on Inductive Logic Programming (ILP) [73], which aims to deduce logic programs from a set of positive and negative examples. Paraphrasing a remark from [12], while these methods suppose a given dataset of signals a priori available to infer formulas for ML tasks such as classification or clustering, to find anchors it would be too inefficient to generate a very large dataset to which one would try to apply such methods. Instead, we need to explore the black-box model to generate just enough



signal samples to find (with some probability of error) the boundary of the decision regions around a specific input signal, described by the most important features expressed in STL. Moreover, we exploit the logical implications (partial order) existing between STL formulas to further improve the efficiency. Another related work is [74], which, given a system model, aims to automatically discover ranges of parameters for which a given property specified by a parametrized temporal logic formula does not hold for the system. The approach actively samples behaviors of the system, but the structure of the temporal logic formula is fixed a priori.

### 4.6.3 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) [29] is a heuristic search method for making sequential decisions in a large search space under uncertainty. It has gained popularity in games and planning problems, notably with its success in the task of Computer Go. A tree (or typically a game tree) is a Directed Acyclic Graph (DAG) whose nodes represent states while edges represent actions, or moves, to pass from one state to another. Starting from the initial state, the tree is built in an incremental and asymmetric manner using guided random sampling to estimate the score, or the success rate, of each move. This score coincides with the definition of the precision in the framework of *anchors* [12], see Section 4.2.1.

The standard approach to search for *anchors* [12] relies on a Multi-Armed Bandit (MAB)-based sampling strategy (e.g., KL-LUCB [33]) to optimize, in a *greedy* manner, the empirical precision of formulas in propositional logic capturing the presence or absence of certain features in the input. Features are considered one after another. In contrast, as in [34, 35] for example where MCTS is used for feature selection, we argue that for our set-up it is more efficient to search for high-precision explanations in a tree encoding STL formulas using a multi-step look-ahead strategy. Indeed, MCTS is able to explore more formulas of different lengths before making a decision, while concentrating on the most promising ones. Furthermore, the *back-propagation* phase (cf. Section 4.3.2) allows to update the precision of shorter formulas without additional sampling effort. This is beneficial because the precision of shorter explanations, which are more interpretable to human users, is continuously updated, thus better estimated.

## 4.7 Conclusion and future work

In this paper, we present a framework providing precise model-agnostic explanations in the form of *anchors* expressed in Signal Temporal Logic (STL) to interpret specific behaviors of dynamic decision-making and control systems, considered as black-box models. Compared to recent work in the area of interpretable machine learning, which mostly focuses on static data, this work considers specifically black-box models processing time-varying signals, making decisions based on past observations. A new method is proposed to build the STL formulas defining anchors incrementally using a multi-step look-ahead strategy based on Monte Carlo Tree Search (MCTS). The logical implications existing between STL formulas were also leveraged to reduce the size of the search space.

Although our computational experiments could show in the case studies that scenarios of reasonable complexity could be analyzed, much work remains to be done to apply such techniques to realistic systems processing high-dimensional signals with high sampling rate, such as autonomous decision systems driven by machine perception algorithms. Moreover, our current implementation only considers first-level primitives. Future work can focus for example on alternative search strategies that avoid discretizing the parameter space for STL formulas and on the influence of the choice of hyper-parameters and perturbation distribution on the execution time and quality of the returned explanations.

## CHAPTER 5    COMPLEMENTARY RESEARCH WORK: TRACKLET RELIABILITY IN ONLINE 3D MULTI OBJECT TRACKING

### 5.1 Introduction

#### 5.1.1 Context

Modern online Multi Object Tracking (MOT) systems surveyed in [40] are generally built upon the “tracking-by-detection” framework, which employs directly an off-the-shelf detector to provide a set of bounding boxes describing the position and the dimension of surrounding objects at each frame. Different modalities such as camera, LiDAR or RaDAR can be used to serve the detection purpose. With camera for example, a 2D object detector takes a static image from a camera as input and outputs a set of 2D bounding boxes. In the same way, a 3D object detector takes a static point cloud as input, i.e., a set of 3D data points, and outputs a set of 3D bounding boxes. These detectors usually use large and complex neural networks to achieve reliable detection performance. In contrast to static inputs, online trackers process a sequence of detections up to the current frame and dynamically return a set of bounding boxes (localization) as well as their identity (identification), with the latter supposed to remain constant for the same object across frames. The estimated trajectory of an object is called a tracklet, which is essentially a sequence of bounding boxes with the same identity. In this work, we specifically focus on ground vehicle tracking, supposing that the tracked object are always placed on flat ground.

The three main failure modes considered in the CLEAR MOT metrics [75] consist of the number of false alarms (False Positives, or FP), misses (False Negatives, or FN), and mismatches (Identity Switches, or IDS). First, false tracklets could be accidentally created due to false detections. Also, not terminating the tracklet early enough even if it has already left the detector’s field of view would also result in false positives. Since most false detections are low-scored, one may try to reduce the number of false positives by simply ignoring them below a certain score threshold. However it is likely to introduce a large number of misses, because some of them are true objects but just poorly localized. These misses would then sacrifice the system’s recall, defined in machine learning as the ratio between the number of correctly identified objects and the total number of true objects. In general, the number of misses is largely determined by the detector’s performance, but can also be due to the following two reasons: an object may have already been detected but the tracking system waited before confirming its presence; an object may be occluded for a short period and the

system decides to terminate the tracklet too early. Finally, this latter issue is also one reason of identity switches: when the object reappears after being occluded, the system creates a new tracklet to track the same object. Two object trajectories crossing each other may also result in identity switches.

### 5.1.2 Contribution

To have a better control on each tracklet’s status, we believe it is important to provide each tracklet with a measure indicating its reliability at runtime, especially when the object suffers from non-detection. This confidence measure can then be used to improve, for example, the tracklet’s life cycle management. In this work we propose a data-driven approach to learn this measure by training a Long-Short Term Memory (LSTM) recurrent neural network, which is capable of processing past information of the entire history starting from the very first detection of the object. Experiments were conducted on two datasets, KITTI [14] and nuScenes [59], to illustrate the trained confidence measure. Furthermore, we also reflect on the definition of the integral metrics AMOTA and AMOTP proposed in [41], which are used as the official evaluation metrics by nuScenes [59] in their *online* tracking challenge. We also show that the CLEAR MOT metrics [75] such as MOTA or MOTP reported on the leaderboard of nuScenes actually correspond to some non-causal systems, due to the preprocessing step before evaluation.

### 5.1.3 Outline

The remainder of this chapter is organized as follows. We review related literature about online MOT systems in Section 5.2, and present in Section 5.3 their general architecture and details of each individual module. We then discuss in Section 5.4 about the recent evaluation approach of MOT systems and the aforementioned issues. Next, we explain the training process of our network in detail in Section 5.5. Finally, the results of experiments conducted on the KITTI and nuScenes datasets are presented in Section 5.6.

## 5.2 Literature review

### 5.2.1 Online 3D Multi Object Tracking

3D MOT systems using the “tracking-by-detection” paradigm in an online fashion include a simple and efficient baseline method AB3DMOT [41] and its followers [40, 44, 46, 52, 76]. As explained in [40], a 3D online MOT system can be decomposed into four individual modules:

3D detector, association, motion model and life cycle management. See Figure 2.2 for the general architecture. To briefly summarize, at each frame, the 3D detector provides a set of 3D bounding boxes as new detections, while the old tracklets are updated in the motion model module with their predicted new position. The association module then matches the predicted tracklets with the new detections. Matched tracklets are again updated with new detection measurements, while the unmatched detections are used to create new tracklets. These tracklets are then sent to the life cycle management module to decide on their active status, e.g., whether to be output or deleted.

Among the aforementioned literature, AB3DMOT [41] uses 3D Intersection Over Union (IoU) as the metric to associate new detections with predicted tracklets via the Hungarian algorithm, while the motion of tracklets is modelled with constant velocity in a Kalman Filter. Chiu *et al.* [44] replace 3D IoU by the Mahalanobis distance [45] to take into account the tracklet’s state uncertainty for data association, and trains the covariance matrices in the Kalman Filter from the detector statistics. CenterPoint [46] considers objects as points, trains a novel 3D detector of high quality and replaces 3D IoU by euclidean distance between the two objects’ center for association, while [40] generalizes 3D IoU to 3D GIoU [47], so as to capture the overlap ratio and the distance of two objects in case of non-overlap. Pöschmann *et al.* [76] represents the new detections as a Gaussian mixture model incorporated in a factor graph and solves data association implicitly via non-linear least-square optimization on the tracklets’ state estimation. FANTrack [48] solves data association directly via deep learning using a CNN.

While the above methods improve the association module, the authors of CBMOT [52] focus on the life cycle management: in contrast to [41] in which “count-based” rules are defined on the number of frames where the tracklet is detected or not, they design heuristic score-update functions and a score-decay mechanism to assign confidence scores to tracklets based on their previous score and the score of matched detections, and define “confidence-based” rules by thresholding on these scores. See Section 5.3.4 for more details. [40] also proposes a two-stage association approach for life cycle management to reduce identity switches. In other work, such as [49–51], 2D information is incorporated to provide additional position and RGB information which are then fused with 3D features to further improve association. Finally, data-driven approaches include [77], in which objects are jointly detected and tracked from a sequence of monocular images using a trained LSTM in an end-to-end manner, and [78], in which a graph neural network is designed to process detections and tracklets together to resolve data association, classify false detections and initialize new tracklets. In addition to learning feature fusion and data association, [49] also learns a binary classifier to decide whether a tracklet should be created from an unmatched detection.

### 5.2.2 Tracklet reliability

Having access to a confidence measure indicating the reliability of each tracklet facilitates the understanding of the tracking system’s decisions. Such a measure that is easily interpretable would also enable human users to react to the result accordingly. Previous work such as [54] uses Bayesian estimation to compute the probability of binary hypotheses (presence, absence) or the density function of the object’s state and existence, given the previous measurements. However, it would be impossible to continuously track an object when it starts to move away from the sensor, and the presence of such an object would be irrelevant in most application. In 2D MOT, [55] manually designs heuristic functions to assign a confidence score to a tracklet at each frame based on previous observations such as its length, the number of frames of non-detection and the affinity with matched detections. This confidence score can then be used to discriminate tracklets so that high-confidence ones are prioritized for data association. Notice that the designed confidence score can be naturally extended to 3D. The aforementioned CBMOT [52] also designs simple heuristic score-update functions to improve life cycle management, while [56] uses a similar score to guide data association.

However, it is unclear how to interpret these confidence scores. Fortunately, with the continuous development of benchmarks or datasets providing ground-truth annotations, such as [14, 59, 79], it becomes more relevant to say that a tracklet is present when it can be associated to a ground-truth object, via a predefined matching algorithm and distance metric, exactly like how systems are evaluated with CLEAR MOT. In object detection, YOLO [13] interprets its confidence score as the IoU between the predicted bounding box and the true object when it is indeed present while pushing it to zero in case no object is in the box. In our work, we also use the IoU between the tracklet and the true object to indicate how confident the system is about the tracklet’s presence, and train an LSTM to learn this measure frame-by-frame, hoping to inform users of each tracklet’s reliability at all times.

### 5.3 Online 3D MOT algorithm

The general architecture of a 3D online MOT system is given in Figure 2.2 and consists of the following modules: detector, association, motion model and life cycle management. The following subsections provide details of each individual module.

### 5.3.1 Detector

The 3D detector module provides at each frame a set of detected bounding boxes. In ground vehicle tracking, these boxes contain the following information:  $(x, y, z, \theta, l, w, h)$ , usually expressed in the sensor’s frame, where  $(x, y, z)$  are the coordinates of the box’s center;  $\theta$  is the heading or the yaw angle;  $(l, w, h)$  denote the length, width and height respectively. Notice the simplification made on the object’s orientation in ground vehicle tracking: the pitch and roll angles are not considered as they are usually not provided in the ground-truth annotations of many public datasets such as KITTI [14] and nuScenes [59]. Detection modules also provide a detection score for each 3D bounding box, which serves as an indicator of the detection quality.

### 5.3.2 Association

The association module is responsible for associating incoming detections with tracklets: a detection matched with a tracklet serves as a new measurement for the tracklet’s state estimation, while unmatched detections allow to create new tracklets. Assigning a detection to a tracklet naturally induces a cost: a wrong assignment should cost more than a correct one. Two popular matching algorithms for the assignment problem are the Hungarian algorithm [43], also known as the Kuhn–Munkres algorithm, and the greedy algorithm. The Hungarian algorithm solves the problem of minimizing the total assignment cost in polynomial time, while the greedy algorithm is an efficient heuristic that assigns gradually a detection to a tracklet by sorting the costs until either every detection or every tracklet has been assigned. Some possible metrics used as the assignment cost are, to list a few, Intersection Over Union (IoU), Generalized Intersection Over Union (GIoU) [47], 2D/3D euclidean distance between the two bounding box centers, or Mahalanobis distance [45]. We add a minus sign on IoU and GIoU when passing them into the cost matrix, because we want to maximize them, but the distances are to be minimized. If  $T$  denotes the bounding box of a tracklet and  $D$  denotes that of a detection, then:

- Intersection Over Union (IoU) of  $T$  and  $D$  is defined as:

$$\text{IoU}_{T,D} := \frac{|T \cap D|}{|T \cup D|} \quad (5.3.1)$$

where  $|\cdot|$  denotes the volume,  $\cap$  denotes intersection while  $\cup$  denotes union. By definition  $\text{IoU}_{T,D} \in [0, 1]$  and defines the overlap ratio.

- Generalized Intersection Over Union (GIoU) is defined as:

$$\text{GIoU}_{T,D} := \text{IoU}_{T,D} - \frac{|C_{T,D} \setminus (T \cup D)|}{|C_{T,D}|} \quad (5.3.2)$$

where  $C_{T,D}$  is the smallest convex hull enclosing both  $T$  and  $D$ , while  $\setminus$  is notation for relative complement or set difference. By definition  $\text{GIoU}_{T,D} \in (-1, 1]$  and generalizes IoU in case of non-overlap ( $\text{IoU} = 0$ ): the farther  $T$  and  $D$  are from each other, the more GIoU approaches  $-1$ .

At the end of data association, special care should be taken to exclude matching pairs inducing a cost higher than a certain threshold, to prevent unreasonable association. For example, if 3D IoU is used as the matching cost, then for any pair of  $(T, D)$  for which  $\text{IoU}_{T,D}$  doesn't reach 0.25,  $T$  should be considered as unmatched while  $D$  is used to create a new tracklet. The threshold for 3D GIoU is often fixed at  $-0.5$  or  $-0.2$ , and 3D distance at 2 meters.

### 5.3.3 Motion model

The motion model module uses a filtering algorithm for state estimation, such as a Kalman Filter. Though the state vector depends on the choice of the filter, which may naturally differ depending on the object category (**CAR**, **BICYCLE**, **PEDESTRIAN**, etc), it should contain the most basic information of a bounding box  $(x, y, z, \theta, l, w, h)$  defining our tracklet. For example, [41] uses a constant velocity model with the state vector  $(x, y, z, \theta, l, w, h, v_x, v_y, v_z)$ :

$$\dot{x} = v_x, \quad \dot{y} = v_y, \quad \dot{z} = v_z$$

while the rest of the parameters remain constant. Moreover, if information about sensor poses is also available at each frame, which can be obtained from GPS/IMU and calibration, then we can apply ego-motion compensation and express the predicted bounding box (from the previous frame) in the current sensor frame coordinates. The predicted bounding box can then be matched with the associated detection in the same sensor frame, allowing to perform measurement update on the first 7 variables  $(x, y, z, \theta, l, w, h)$ .

### 5.3.4 Life cycle management

The life cycle management module is the main focus of our work and is primordial to reduce false alarms (FP) and misses (FN) at the same time. To be more specific, a tracklet can take one of the four following active status in the state machine: **INIT**, **ACTIVE**, **PASSIVE** or **DEAD**. The status **ACTIVE** should only be set if we are confident enough of its real presence —



any tracklet created from a false detection or corresponding to an object which has already disappeared from the field of view should desirably be set to **DEAD** and deleted from the memory as soon as possible. Though it should still be kept **ACTIVE** if just occluded for a short period of time.

Figure 5.1 shows the state diagram of the life cycle management, and indicates the criteria to pass from a status to another, depending on two boolean variables **birth** and **death**. Their negation is noted with the exclamation mark (!). Upon creation of a tracklet, its status is originally set to **INIT**. It is then set to **ACTIVE** immediately if the **birth** criterion is met, and **PASSIVE** otherwise. The latter status corresponds to a warm-up period during which we are still not confident enough about the tracklet's presence: it will only be kept in memory but not eliminated right away. In the next frame, now that existing tracklets are either **ACTIVE** or **PASSIVE**, if the **death** criterion is met, the status is set to **DEAD** and the tracklet is immediately removed from the system's memory. The remaining tracklets in status **PASSIVE** will pass to **ACTIVE** when the **birth** criterion is finally met. Otherwise, the status just remains unchanged. Overall, the life cycle management provides a balance between false alarms and misses but requires tuning certain hyper-parameters to decide on the **birth** and **death** criteria.

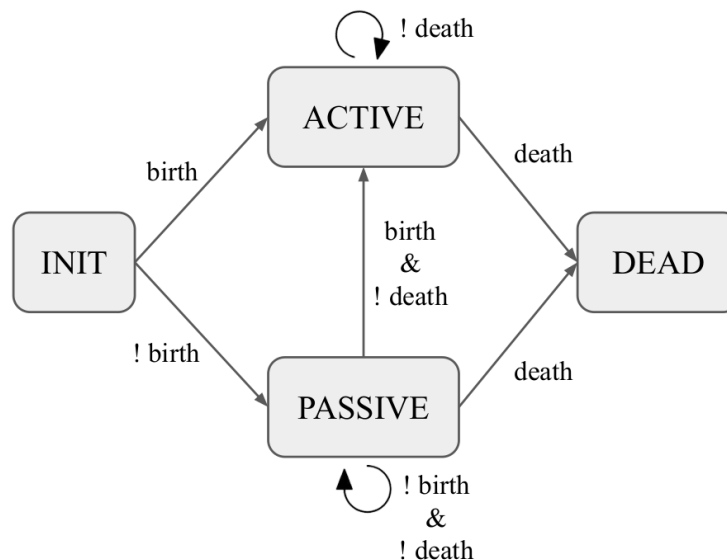


Figure 5.1 Diagram of the life cycle management

In *count-based* methods such as [41], the **birth** criterion is met when the tracklet has been matched to some detections for at least **min hits** frames, while **death** corresponds to being unmatched for more than **max age** “consecutive” frames. Notice however that these hyper-parameters are fixed and don't take into account the tracklet's actual length, thus a tracklet

which has been tracked for a long period of time, for which we should already be confident enough of its presence, would still be deleted after just a short occlusion. Also, a tracklet created with a detection producing a relatively high score would not be set to **ACTIVE** immediately.

On the contrary, in *confidence-based* methods [52], a tracklet is assigned a tracklet score  $s^{\text{trk}}$  at each frame, and two thresholds  $s^{\text{birth}}$  and  $s^{\text{death}}$  are used to define the **birth** and **death** criteria. More specifically, **birth** simply corresponds to  $s^{\text{trk}} \geq s^{\text{birth}}$ , while **death** corresponds to  $s^{\text{trk}} < s^{\text{death}}$ . Here we distinguish  $s^{\text{trk}}$  from detection scores  $s^{\text{det}}$ ; the latter comes directly with the bounding boxes from the 3D detector. For example, CBMOT [52] manually designs score-update functions to calculate  $s^{\text{trk}}$  at each frame based on the score of the matched detection  $s^{\text{det}}$ . More specifically, when a tracklet is created with an unmatched detection, its score is initialized with the corresponding  $s^{\text{det}}$ .  $s^{\text{trk}}$  then decays by a value  $\sigma$  (score decay) at each frame, since the presence of the tracklet is more susceptible to be unreliable as time goes by:

$$\hat{s}_{t+1}^{\text{trk}} = s_t^{\text{trk}} - \sigma.$$

In case the tracklet is detected, the authors argue that  $s_{t+1}^{\text{trk}}$  should be greater than both  $\hat{s}_{t+1}^{\text{trk}}$  and  $s^{\text{det}}$ , because we can be more confident of its presence. Among the score-update functions they propose, the one that experimentally achieves the best performance is

$$s_{t+1}^{\text{trk}} = \begin{cases} 1 - (1 - \hat{s}_{t+1}^{\text{trk}})(1 - s_{t+1}^{\text{det}}) & \text{if detected,} \\ \hat{s}_{t+1}^{\text{trk}} & \text{if undetected.} \end{cases}$$

In our work, we also propose a confidence-based method, but instead of manually designing heuristic functions to assign tracklet scores, we take a data-driven approach with an LSTM to learn a confidence score indicating the tracklet’s presence at each frame, predicting the IoU between the tracklet and a potential true object. While a value of 1 means a perfect tracklet, a value of 0 rejects the possibility of an overlap with any object. The use of an RNN such as LSTM is justified by the fact that whether the tracklet is actually present or not is highly dependent on its past. Interesting factors may include, among others, the continuity of matches, the reliability of matched detections, etc.

## 5.4 Evaluation of online MOT systems

In Section 5.4.1, we present the CLEAR MOT metrics used to evaluate an MOT system. In Section 5.4.2, we reflect on the definition of the integral metrics proposed in [41], which are then used as the official evaluation metrics by nuScenes [59] in their tracking challenge.

### 5.4.1 CLEAR MOT metrics

CLEAR MOT [75] defines standard metrics used to evaluate MOT algorithms nowadays. It includes the tracking accuracy, or MOTA, and the tracking precision, or MOTP, allowing to quantify the overall performance of an MOT system. The authors mention a few points expected from an ideal MOT system: the objects' trajectory should be precisely estimated and exactly one trajectory estimate (tracklet) be produced per object (ground-truth). More precisely, we compute at each frame:

1. valid correspondences between the output tracklets and the real objects;
2. error of position estimation for each correspondence;
3. correspondence errors: an unmatched object is counted as one miss (False Negative, or FN), an unmatched tracklet is counted as one false alarm (False Positive, or FP), and an identity change is counted as one mismatch (Identity Switch, or IDS) compared to the previous frame.

To determine whether a tracklet is a TP or a FP at each frame, we have to associate the output tracklets with the objects provided in the annotations, similar to the matching strategy described previously. In CLEAR MOT however, a correspondence already established in the previous frame is still considered valid in the current frame as long as the matching distance does not exceed a predefined threshold. This distance and the threshold cannot be generally defined and are task-specific. In 3D MOT, we use 3D euclidean distance with a threshold of 2 meters. Next, the tracklets and the objects that are not yet assigned are matched using the Hungarian algorithm [43] to minimize the total matching distance, followed by elimination of invalid correspondences, i.e., whose distance exceeds the threshold previously defined. Afterwards, if an object is matched to a different tracklet compared to the previous frame, a mismatch (Identity Switch, or IDS) is counted; the total number of valid correspondences is counted as True Positive (TP); unmatched objects are counted as False Negative (FN) and unmatched tracklets as False Positive (FP). Finally, MOTP is computed by averaging the distance for each valid correspondence, while MOTA is defined as follows:

$$\text{MOTA} := 1 - \frac{\text{FP} + \text{FN} + \text{IDS}}{\text{GT}} = \frac{\text{TP} - \text{FP} - \text{IDS}}{\text{TP} + \text{FN}} \quad (5.4.1)$$

where  $GT = TP + FN$  denotes the total number of ground-truth objects. We expect MOTA to be as high and near 1 as possible, and MOTP to be as low as possible.

#### 5.4.2 Reflection on integral metrics

To briefly introduce the integral metrics, [41] argues that the CLEAR MOT metrics don't explicitly consider objects' confidence score during evaluation, and that it should be acceptable if false tracklets have relatively low confidence scores. Inspired by the integral metrics such as mAP, which computes the area below the PR curve, the authors propose two integral metrics particularly for tracking systems, AMOTA and AMOTP, which average MOTA and MOTP respectively across different recall values (TP divided by GT) ranging from 0 to 1, to ensure that an outstanding MOT system should achieve high MOTA values at all possible recall values. To allow a better understanding of how the CLEAR MOT metrics (FP, FN, IDS, MOTA) behave across recall values, in the same spirit as the PR curve, we can render curves of these metrics over recall. While we compare different MOT systems, the corresponding curves (for a same metric) can be superposed to show at which threshold a system performs better than another.

As this practice starts to become a new standard for MOT evaluation, we would like to point out two major problems preventing it from reflecting the real performance of a system. Here we cite the definition of a tracklet's confidence score as follows [41]: “*We define the confidence score of an object trajectory as the average of its confidence scores across all frames*”; or in the official evaluation code of nuScenes [59]: “*We average over frame-level scores to compute the track-level score. The score is used to determine positive and negative tracks via thresholding.*”

On the one hand, this score has never been properly defined. A tracking system might not even assign any score to its output tracklets. Otherwise, in [41] for example, a tracklet's frame-level confidence score is directly assigned with the score of the matched detection, and remains constant if unmatched — its score is thus that of the last matched detection. Imagine a tracklet created with a high-scored detection at the first frame and remaining active for a period of time without being matched to any detection ever again before being killed. The average confidence of this tracklet across frames is exactly the score of the first detection, which is so high that it can beat other tracklets that may have been continuously tracked for a long period of time but with a slightly lower score.

On the other hand, we work with online MOT systems, which are supposed to be causal. In the official tracking evaluation code of nuScenes [59], a tracklet with missing scores is preprocessed so that its scores are interpolated across frames. [40] points out that the interpolation step uses future information and thus the evaluation is not a fully online one.

Furthermore, [40] also shows that we can explicitly assign very low scores to these tracklets at frames where scores are missing, and show a significant improvement on the system’s overall recall and AMOTA, without even changing the behavior and the decisions of the system. In addition, we argue that the non-causal aspect lies not only in the interpolation step. Even if no interpolation is performed (i.e., tracklets are assigned a confidence score at every frame), a user would not be allowed to wait until the end of a tracklet to compute its average confidence score before deciding to output it or not via thresholding. As a result, the thresholding step is also non-causal, so that the metrics computed for a certain recall value do not perfectly reflect the actual performance of an online MOT system on that particular recall. This is the case in particular for the highest MOTA and MOTP scores reported on the leaderboard of the nuScenes tracking challenge, in which “online” tracking is imposed. In consequence, we argue that these scores are misleading and the evaluation should be done directly on the output tracklets without any preprocessing.

In the experiments conducted in Section 5.6, the scores of some previous works are not the same as reported on the leaderboard, as we only compare different systems with the MOTA score (instead of AMOTA) of the output tracklets, without any preprocessing.

## 5.5 Training

In this section, we present how the network is trained in detail. Section 5.5.1 explains how the raw data of the training set are preprocessed to generate relevant input features and labels, while Section 5.5.2 provides details about the network structure and the choice of training hyper-parameters. Finally, during validation and testing, for each tracklet, the network will be able to output at runtime a prediction of the IoU between the tracklet and a potential true object.

### 5.5.1 Raw data preprocessing

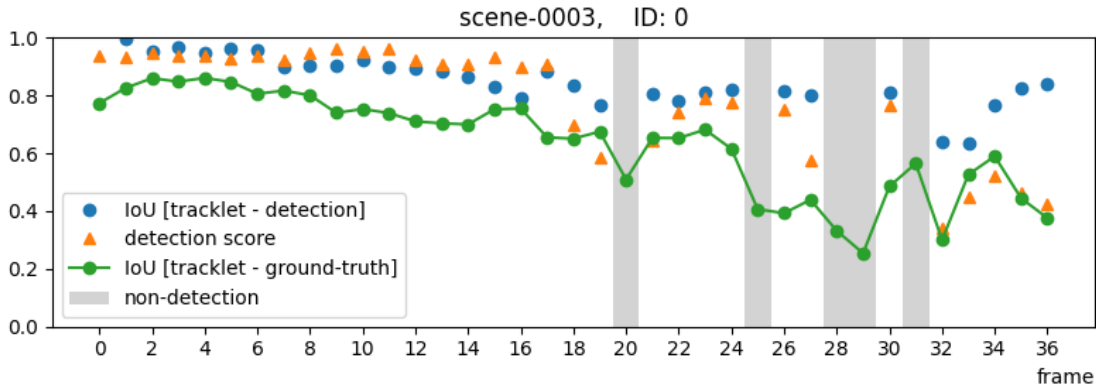
With the annotated bounding boxes (ground-truth) provided in the training set, we use a matching strategy similar to the one described previously to associate detections with the true objects, to obtain information about the presence, the true position and the identity of these detections. In our experiments, to label detections, we choose the 3D euclidean distance as the matching cost and apply the Hungarian algorithm for data association. If a detection is more than 2 meters away from the associated true object, then it should be excluded and considered as a false detection. To train the network, we make use of every true object provided in the training set, containing its real position and its identity. We create

tracklets from the “true” detections (detections associated with a true object) that newly appeared, and filter them with the sequence of detections labelled with the same identity, using the same filtering algorithm described in Section 5.3.3. A tracklet is terminated at the frame where the corresponding true object disappears in the annotations.

We manually select some interesting features to extract from a tracklet so that the network can learn the correlations between them across frames, and predict the IoU between the tracklet and the true object. Here we only use the following two features as inputs:

1. 3D IoU between the tracklet and the matched detection;
2. score of the matched detection.

We show an example of a training sequence in Figure 5.2. The network learns from sequences of these two features (blue circle and orange triangle), labelled with the IoU between the filtered tracklet and the true object (green line). We observe that whenever there is non-detection (area filled with light-gray, frame 20, 25, 28, 29, 31), the label tends to decrease, thus coinciding with the score decay mechanism designed in CBMOT [52], as desired. Indeed, the label corresponds to the IoU between the filtered tracklet and the true object. In case of non-detection, the estimated position of the tracklet would naturally derive from the true position, thus reducing our confidence of its presence.



Dataset: nuScenes [59]. Detector: Megvii [80]

Figure 5.2 Example of a training sequence

These features were inspired by [81], where the authors propose to perform sensor fusion (statically) between a 2D detection (camera) and a 3D detection (LiDAR) together to provide a more accurate detection and a more refined confidence score, using deep learning. They propose the following four features:

1. 2D IoU between the 2D detection and the projected 3D detection
2. 2D detection score
3. 3D detection score
4. normalized distance between the 3D detection and the LiDAR.

We make an analogy in our work, as if at each frame we were trying to fuse a tracklet with a new detection, but dynamically. However, notice that instead of repeatedly inputting the tracklet score (output from the previous frame), it is more suitable to use an RNN.

Special care should be taken during creation of a tracklet, on the one hand, and during non-detection, on the other hand. In the first case, the first feature is set to  $-2$ , to indicate to the network that the object just appeared; while the second feature is simply the score of the detection used to create the tracklet. In case of non-detection, i.e., when no detection is labelled with the tracklet’s identity, the two features are simply set to  $-1$  in order to distinguish from normal cases where the IoU and the detection score are within  $[0, 1]$ .

Finally, since the labels are contained within  $[0, 1]$ , they are passed into a logit function before being passed into the network, because we want to perform regression in the entire  $\mathbb{R}$ . We recall that the logit function is the inverse of the sigmoid function  $\sigma^{-1}(x) := \ln(x/(1-x))$ , for  $x \in [\epsilon, 1 - \epsilon]$  capped at  $\mp\sigma^{-1}(\epsilon)$  outside the interval, for some  $\epsilon$ , e.g.,  $10^{-6}$ . In consequence, the output of the network should inversely be passed into a sigmoid function to obtain a value in  $[0, 1]$  before making prediction during validation or testing.

### 5.5.2 Neural network structure and hyper-parameters

The inputs containing two features are sent one at a time into the LSTM, as we perform online tracking. The dimension of the hidden state is set to 8. A linear fully connected layer of shape  $8 \times 1$  is added after the LSTM. The structure is shown in Figure 5.3. In our experiments, we trained the network using the Mean Squared Error (MSE) loss function and the Adam optimizer [82] configured with a learning rate of  $10^{-3}$ .

## 5.6 Experiments

### 5.6.1 Datasets

We present in this section the two datasets, KITTI [14] and nuScenes [59], upon which our experiments were conducted. By comparing the leaderboard of these two datasets, nuScenes seems to be more challenging and difficult than KITTI.

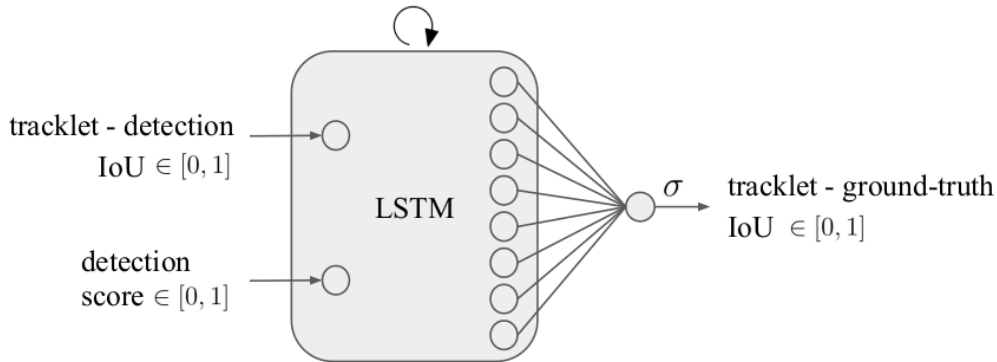


Figure 5.3 Structure of our network

**KITTI.** The raw data were collected in the city of Karlsruhe and include diverse driving scenarios in rural and urban areas. The sampled data were captured with a pair of grayscale stereo cameras, a pair of color stereo cameras, a Velodyne laserscanner, 4 varifocal lenses and GPS/IMU. The KITTI tracking benchmark proposes 21 training sequences and 29 test sequences. Synchronized at a frequency of 10 Hz, the samples were annotated with ground-truth 3D bounding boxes across 8 different classes. Among the training sequences, 11 sequences totalling 3908 frames were selected for validation in [41, 52]. In order not to train with sequences we evaluate on, we only use the remaining 10 sequences, totalling 4100 frames, to train our LSTM.

**nuScenes.** Inspired by the previous KITTI dataset, the authors of the nuScenes dataset collected data in Boston and Singapore, two cities known for their complicated traffic conditions, on a car fully equipped with six cameras, one LiDAR, five RaDARs and GPS/IMU, travelling through diverse driving conditions totalling a travel distance of 242 km. 1000 scenes were manually selected, each of which lasts for approximately 20 seconds. They were split into 700 scenes for training, 150 for validation and 150 for testing, and annotated with ground-truth 3D bounding boxes across 23 different object categories on keyframes sampled at 2 Hz. Among the 700 training scenes, 15 were blacklisted due to missing GPS/IMU data. In total, the training sequences contain 27533 frames, compared to 6019 frames for validation.

### 5.6.2 3D detectors

To fairly compare with previous works, we use the same 3D detections generated using the following detectors provided by [41]: PointRCNN [15] for KITTI and Megvii [80] for nuScenes. For simplicity, only the category **CAR** is considered, i.e., only cars (and vans in KITTI) are



detected and tracked. To get an idea of the detection scores for detection quality, we start by classifying the detections into true and false ones via data association with ground-truth annotations (Hungarian, 3D euclidean distance, threshold 2 m). For each detector in the corresponding dataset, we show in the first row of Figure 5.4 the distribution (histogram) of detection scores for both true and false ones. For the true ones in particular, we also show the distribution of IoU and 3D euclidean distance with the real objects for detection quality. For some reason, PointRCNN yields scores ranging in  $\mathbb{R}$ , between  $-2$  and  $16$ , thus we decided to apply a linear transformation to bring the scores within  $[0, 1]$ , in the preprocessing step. We can observe in the figure that although most false detections are low-scored, they are mixed with some low-scored true detections as well. It would thus be a bad idea to simply ignore detections whose score is under a certain threshold. Also, as a side note, PointRCNN applied to KITTI seems to have a better detection quality in terms of both IoU and 3D distance than Megvii applied to nuScenes.

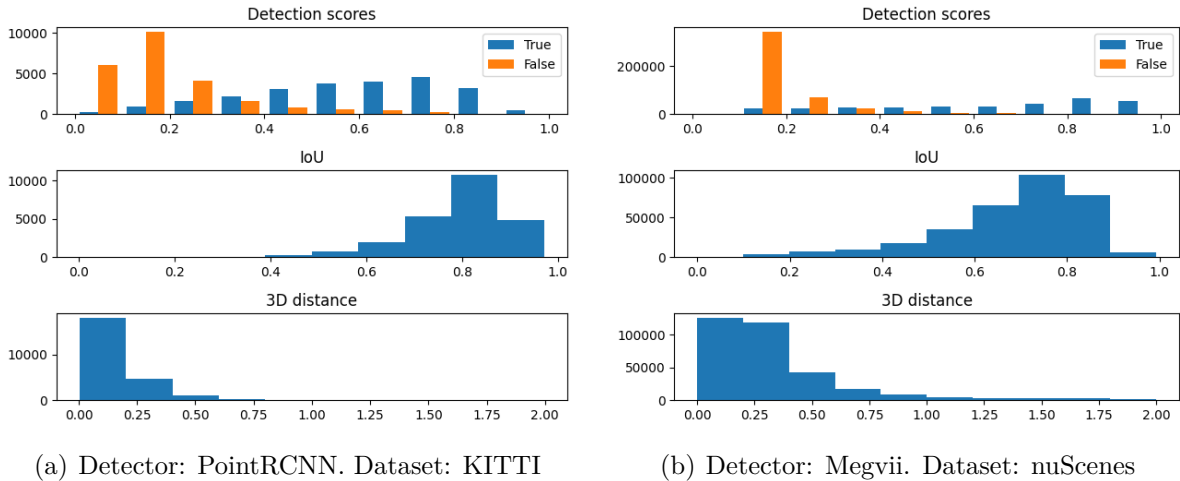


Figure 5.4 Distributions for both true and false detections

### 5.6.3 Hyper-parameters

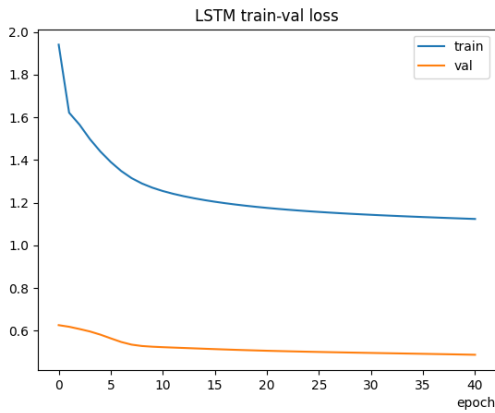
We present in Table 5.1 the hyper-parameters used in our experiments for the association module, the same as those proposed in [41]. The hyper-parameters used in the life cycle management module are tuned by evaluating on the validation set to optimize the MOTA metric, whose definition is given in Equation (5.4.1).

Table 5.1 Important hyper-parameters in the association module

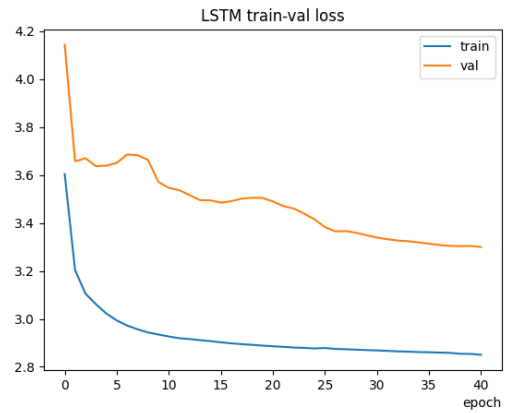
Dataset	algorithm	metric	threshold
<b>KITTI</b>	Hungarian	GIoU	-0.2
<b>nuScenes</b>	Greedy	GIoU	-0.5

### 5.6.4 Training

Before training the network, raw data should be preprocessed into tensors and labels as described in Section 5.5.1. Finally, with PointRCNN in KITTI, 420 labelled training sequences plus 209 labelled validation sequences were prepared for training; while 19280 labelled training sequences plus 3728 labelled validation sequences were prepared with Megvii in nuScenes. In Figure 5.5 we show the training and validation loss for both datasets. The model is only saved when the validation loss gets lower than each of the previous epochs. In both datasets, the model simply converges without overfitting after 40 epochs of training. Notice that in KITTI, the validation loss remains lower than the training loss, probably because the train-val split was not done properly, and it seems like the validation scenes are a lot easier than the training scenes.



(a) Detector: PointRCNN. Dataset: KITTI



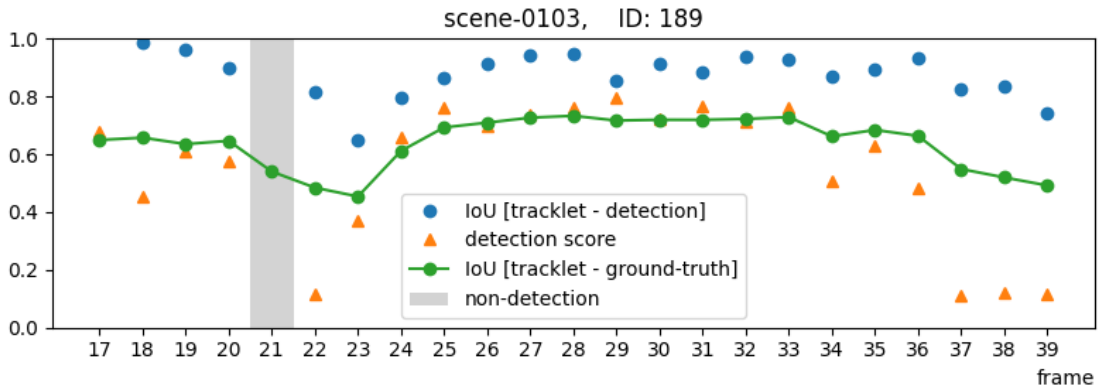
(b) Detector: Megvii. Dataset: nuScenes

Figure 5.5 Training and validation loss

### 5.6.5 Qualitative results

By applying the algorithm described in Section 5.3, we design an MOT system with a modification brought to the life cycle management module of AB3DMOT [41], using the confidence scores learnt with our LSTM. Figure 5.6 shows an example of a tracklet with the sequence of the two input features (blue circle and orange triangle) and the predicted confidence measure (green line). The confidence measure corresponds to the IoU between the tracklet and a potential true object. We observe that the trained LSTM tends to assimilate the effect of the score decay mechanism proposed in CBMOT: when tracklets suffer from non-detection (area filled with light-gray), the predicted score effectively decreases (frame 21), like in the training sequence shown in Figure 5.2. Moreover, when the detection scores are too low, the predicted confidence measure also tends to decrease (frame 22, 37-39) as desired.

We can also compare the distribution of the confidence score before and after each non-detection in the whole dataset. Figure 5.7 shows the respective score distributions. The overall scores are shifted to the left after each non-detection.



Dataset: nuScenes. Detector: Megvii

Figure 5.6 Example of a tracklet with its two input features and the predicted IoU

### 5.6.6 Quantitative results

We compare our evaluation results with AB3DMOT [41] and CBMOT [52] for which only the life cycle management is different. As discussed previously in Section 5.4.2, the submitted result should not be further processed to eliminate a small portion of tracklets in order to boost the metrics via a thresholding step that is non-causal. In the work of AB3DMOT and CBMOT, hyper-parameters were tuned to maximize AMOTA and/or MOTA via thresholding, which is questionable. As such, we manually fine-tune again the hyper-parameters, via

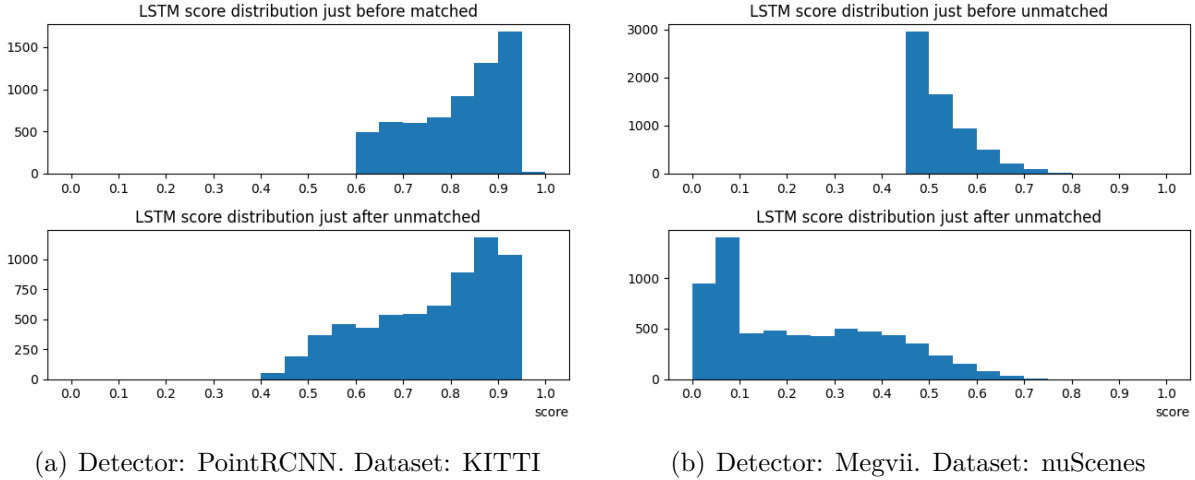


Figure 5.7 LSTM score distributions before and after each non-detection

grid search, such as **max age**, **min hits** for AB3DMOT; birth score  $s^{\text{birth}}$ , death score  $s^{\text{death}}$  for CBMOT and our system; and score decay  $\sigma$  for CBMOT. Since the annotations of the testing set weren't available for both datasets during our experiments, the results reported below were simply evaluated with the validation set. See Table 5.2, 5.3, 5.4 for KITTI, and Table 5.5, 5.6, 5.7 for nuScenes, where the tuned hyper-parameters and the corresponding MOTA score are marked in bold.

In both datasets, confidence-based methods including CBMOT and our system outperform the count-based method AB3DMOT. However, our system controlling the life cycle with a trained LSTM doesn't yield a better score than CBMOT which uses a heuristic score-update function and a score-decay mechanism, with a slight gap: 83.46% v.s. 85.09% in KITTI, and 56.64% v.s. 57.90% in nuScenes. Notice that we explicitly tuned the hyper-parameters on the validation set for both methods to pick the best ones during evaluation, thus it is uncertain that with the same hyper-parameters CBMOT would perform better than our method on the testing set.

Interestingly, the tuned hyper-parameters  $s^{\text{birth}}$  and  $s^{\text{death}}$  in our system are meaningful. Those maximizing the MOTA are the following:

$$(s^{\text{birth}}, s^{\text{death}}) = \begin{cases} (0.7, 0.6) & (\text{KITTI}) \\ (0.4, 0.45) & (\text{nuScenes}) \end{cases} \quad (5.6.1)$$

From the second row of Figure 5.4, we can notice that in KITTI the IoU between true detections and the real objects lies in majority above 0.6, while in nuScenes above 0.4. It is thus natural to set the status of a tracklet to **ACTIVE** when the predicted IoU exceeds this threshold, corresponding to both  $s^{\text{birth}}$  and  $s^{\text{death}}$  in Equation (5.6.1). Furthermore, this value could have been deduced directly from the training set and thus could have simplified the tuning effort of these hyper-parameters.

In conclusion, we successfully trained an LSTM capable of providing the tracklets with a more interpretable confidence measure at runtime, by predicting the IoU between each tracklet and a potential true object.

## 5.7 Conclusion and future work

In this chapter, we discussed about the general design of an online 3D Multi Object Tracking (MOT) system based on the “tracking-by-detection” framework, and introduced the use of a Long-Short Term Memory (LSTM) recurrent neural network, which can be trained to predict a confidence measure describing the IoU between the tracklet and a potential true object. This confidence measure not only improves robustness of such systems but can also be used in the life cycle management to achieve competitive performance. We also presented the CLEAR MOT metrics such as MOTA and MOTP, which facilitate evaluation of MOT systems, while questioning the usage of the recently proposed integral metrics such as AMOTA and AMOTP as well as the correctness of the MOTA and MOTP scores reported on the leaderboard of the nuScenes benchmark. We argue that the submitted tracklets should not be preprocessed before evaluation and the thresholding step across recall values contradicts the “online” principle. Finally, we conducted experiments of our proposed system on two public datasets, KITTI and nuScenes, and compared its performance with two similar previous works, by changing only the life cycle management module tested with different hyper-parameters.

Though we didn’t achieve the state-of-the-art performance, the main contribution of this work is an attempt to improve certification of online MOT systems. More extensive experiments could be conducted to optimize the design of the network structure and the features provided to the network. Due to the computational constraints, only the class **CAR** was used in our experiments, but without loss of generality, it can be easily replaced with other classes such as **PEDESTRIAN** and **BYCICLE**. Similarly, in each dataset the 3D detector is fixed during the whole experiment. It would be interesting to notice how detectors of different qualities would affect the training process and validation/testing results. Finally, the present work trains and applies LSTM on individual tracklets without considering their mutual interaction or interaction with the sensor, such as their relative position. We may try to improve the

Table 5.2 Validation results of AB3DMOT (KITTI)

min hits	1	1	2	2	<b>3</b>	3
max age	1	2	1	2	<b>1</b>	2
MOTA $\uparrow$	0.4236	0.1016	0.7526	0.6212	<b>0.8089</b>	0.7491
MOTP $\downarrow$	0.8203	0.8087	0.8310	0.8183	0.8365	0.8231
Recall $\uparrow$	0.9539	0.9689	0.9336	0.9567	0.9161	0.9455
TP $\uparrow$	9887	10218	9535	9979	9264	9786
FP $\downarrow$	4344	7194	1394	2719	752	1538
FN $\downarrow$	478	328	678	452	848	564
IDS $\downarrow$	8	6	1	3	1	0

Table 5.3 Validation results of CBMOT (KITTI)

$s^{\text{birth}}$	0.3	0.35	0.35	<b>0.35</b>	0.35	0.4
$s^{\text{death}}$	0.8	0.75	0.8	<b>0.8</b>	0.85	0.8
$\sigma$	0.15	0.15	0.1	<b>0.15</b>	0.15	0.15
MOTA $\uparrow$	0.8409	0.8502	0.8426	<b>0.8509</b>	0.8474	0.8493
MOTP $\downarrow$	0.8362	0.8389	0.8366	0.8388	0.8415	0.8410
Recall $\uparrow$	0.9222	0.9154	0.9230	0.9153	0.9058	0.9079
TP $\uparrow$	9383	9288	9413	9272	9148	9166
FP $\downarrow$	541	397	534	391	328	333
FN $\downarrow$	792	858	785	858	951	930
IDS $\downarrow$	0	0	0	0	0	0

Table 5.4 Validation results of our system (KITTI)

$s^{\text{birth}}$	0.65	0.7	<b>0.7</b>	0.7	0.75
$s^{\text{death}}$	0.6	0.55	<b>0.6</b>	0.65	0.6
MOTA $\uparrow$	0.7526	0.8258	<b>0.8346</b>	0.8266	0.8030
MOTP $\downarrow$	0.8228	0.8286	0.8314	0.8387	0.8454
Recall $\uparrow$	0.9524	0.9325	0.9307	0.9111	0.8766
TP $\uparrow$	9947	9692	9617	9317	8890
FP $\downarrow$	1573	757	669	543	399
FN $\downarrow$	497	702	716	909	1251
IDS $\downarrow$	3	1	1	1	1

Table 5.5 Validation results of AB3DMOT (nuScenes)

min hits	1	1	1	<b>2</b>	2	2
max age	1	2	3	<b>1</b>	2	3
MOTA $\uparrow$	0.4349	0.3236	0.2174	<b>0.5089</b>	0.4473	0.3836
MOTP $\downarrow$	0.5821	0.5621	0.5520	0.5971	0.5740	0.5629
Recall $\uparrow$	0.6974	0.7254	0.7391	0.6536	0.6903	0.7068
TP $\uparrow$	55791	58035	59127	52291	55226	56543
FP $\downarrow$	19537	30548	40008	10714	18380	24654
FN $\downarrow$	24213	21969	20877	27713	24778	23461
IDS $\downarrow$	1463	1600	1729	865	1057	1201

Table 5.6 Validation results of CBMOT (nuScenes)

$s^{\text{birth}}$	0.25	0.3	0.3	<b>0.3</b>	0.3	0.3	0.35
$s^{\text{death}}$	0.5	0.45	0.5	<b>0.5</b>	0.5	0.55	0.5
$\sigma$	0.15	0.15	0.1	<b>0.15</b>	0.2	0.15	0.15
MOTA $\uparrow$	0.5775	0.5790	0.5730	<b>0.5790</b>	0.5774	0.5786	0.5779
MOTP $\downarrow$	0.6013	0.6046	0.5927	0.6074	0.6168	0.6100	0.6119
Recall $\uparrow$	0.6677	0.6590	0.6776	0.6554	0.6392	0.6518	0.6444
TP $\uparrow$	53419	52720	54212	52437	51142	52143	51554
FP $\downarrow$	6399	5690	7570	5412	4354	5155	4761
FN $\downarrow$	26585	27284	25792	27567	28862	27861	28450
IDS $\downarrow$	819	705	796	699	593	694	556

Table 5.7 Validation results of our system (nuScenes)

$s^{\text{birth}}$	0.35	0.4	<b>0.4</b>	0.4	0.45
$s^{\text{death}}$	0.45	0.4	<b>0.45</b>	0.5	0.45
MOTA $\uparrow$	0.5650	0.5512	<b>0.5664</b>	0.5613	0.5634
MOTP $\downarrow$	0.6363	0.6222	0.6416	0.6638	0.6450
Recall $\uparrow$	0.6457	0.6584	0.6366	0.6048	0.6290
TP $\uparrow$	51660	52675	50931	48389	50324
FP $\downarrow$	5748	7767	4995	2994	4690
FN $\downarrow$	28344	27329	29073	31615	29680
IDS $\downarrow$	711	813	625	490	558

LSTM to learn to detect occlusion or truncation more accurately. For example, if one object is occluded by another in the sensor's viewing angle, the score of the occluded tracklet should not decrease even in case of non-detection.



## CHAPTER 6 GENERAL DISCUSSION

In the article (Chapter 4), we presented *Temporal Logic Explanations for Dynamic Decision Systems using Anchors and Monte Carlo Tree Search*, a framework that allows to interpret a given behavior of any black-box dynamic system. Recent literature in the field of interpretability includes model-specific models such as [16, 17], which provide visual explanations but are not flexible enough to be applied to other domains, while global methods such as replacing the model by a small decision tree [18] are not precise enough. Using the anchors methodology [12], we demonstrated that highly precise local explanations expressed in STL can be generated efficiently using MCTS, despite the high-dimensional search space where temporal properties are incorporated. We then applied the methodology to two case studies in simulations of reasonable complexity: an automatic transmission control system implemented in a vehicle and the ACAS Xu system that maneuvers an airplane to avoid collisions. We observed that the algorithm can produce interpretable time-dependant explanations without any prior knowledge of the inner mechanisms of the decision system.

In Chapter 5 (*Tracklet Reliability in Online 3D Multi Object Tracking*), we focused specifically on a type of dynamic decision system — online 3D MOT systems. We trained an LSTM to predict a quality measure for each tracklet describing the IoU between the tracklet and a potential true object at runtime. In contrast to previous works [52, 55, 56] that design heuristic confidence scores to improve association or life cycle management, our quality measure is more interpretable, allowing a better understanding of the tracking system’s decisions. Not only does it inform users of each tracklet’s reliability, but it can also be employed in life cycle management and achieve competitive performance in both KITTI [14] and nuScenes [59], compared to count-based methods such as AB3DMOT [41].

## CHAPTER 7 CONCLUSION AND RECOMMENDATIONS

### 7.1 Synthesis

In this thesis, we focused on dynamic decision systems, and included an article about generating local explanations for any black-box model, and a complementary research contribution on providing an interpretable quality measure for a tracking system. The behaviors of these systems are expected to be understood and reliable.

In the first article, we proposed to interpret specific behaviors of any black-box model by extracting features from time-varying input signals, expressed in Signal Temporal Logic (STL), to generate explanations that gradually maximize the precision and the coverage in the *anchors* framework. Since including past observations results in a high-dimensional search space, we proposed a new method to build STL formulas using a multi-step look-ahead strategy based on Monte Carlo Tree Search (MCTS).

In the complementary work, we focused on a specific type of dynamic decision system — online 3D Multi Object Tracking (MOT) system. To be aware of the reliability of each tracklet, we trained a Long-Short Term Memory (LSTM) recurrent neural network to predict a quality measure describing the IoU between the tracklet and a potential true object. It was also applied in the life cycle management to achieve competitive performance. Moreover, we raised issues about the usage of the recently proposed integral metrics such as AMOTA and AMOTP as well as the metrics reported on the nuScenes benchmark.

### 7.2 Future work

Extensive experiments were conducted to illustrate the proposed methods. In the first article, scenarios of reasonable complexity in simulation could be analyzed to generate explanations, but it would be interesting to apply the method to realistic perception-driven systems processing high-dimensional signals with high sampling rate, e.g., the online MOT system proposed in the complementary work. In such scenarios, much work remains to be done to optimize the algorithm complexity and to extend the search space of STL formulas, e.g., by considering second-level primitives. In the complementary research work, future work can include the optimization of the network structure and the choice of features provided to the network. More extensive experiments may also be conducted on different detectors and object categories. Finally, mutual interaction between tracklets may also be considered to detect occlusion and truncation and make the predicted quality measure even more accurate.

## REFERENCES

- [1] N. K. Logothetis and D. L. Sheinberg, “Visual object recognition,” *Annual Review of Neuroscience*, vol. 19, no. 1, pp. 577–621, 1996, pMID: 8833455. [Online]. Available: <https://doi.org/10.1146/annurev.ne.19.030196.003045>
- [2] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, “Deepdriving: Learning affordance for direct perception in autonomous driving,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2722–2730.
- [3] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” 2014.
- [4] NHTSA, “Collision between vehicle controlled by developmental automated driving system and pedestrian,” *NHTSA News and Events*, 2019. [Online]. Available: <https://www.nhtsa.gov/news/events/Pages/2019-HWY18MH010-BMG.aspx>
- [5] P. Charlier, “Tesla on autopilot crashes into overturned truck,” *Taiwan English News*, 2020. [Online]. Available: <https://taiwanenglishnews.com/tesla-on-autopilot-crashes-into-overturned-truck/>
- [6] Tencent Keen Security Lab, “Experimental security research of tesla autopilot,” pp. 30–35, 2019.
- [7] W. Xiang, P. Musau, A. A. Wild, D. M. Lopez, N. Hamilton, X. Yang, J. Rosenfeld, and T. T. Johnson, “Verification for machine learning, autonomy, and neural networks survey,” 2018. [Online]. Available: <https://arxiv.org/abs/1810.01989>
- [8] B. Hoxha, H. Abbas, and G. Fainekos, “Benchmarks for temporal logic requirements for automotive systems,” in *ARCH14-15. 1st and 2nd International Workshop on Applied Verification for Continuous and Hybrid Systems*, ser. EPiC Series in Computing, vol. 34. EasyChair, 2015, pp. 25–30.
- [9] MathWorks, “Modeling an automatic transmission controller,” online. Accessed on 10 Dec. 2021. [Online]. Available: <https://www.mathworks.com/help/simulink/slref/modeling-an-automatic-transmission-controller.html>

- [10] G. Ernst, P. Arcaini, I. Bennani, A. Donze, G. Fainekos, G. Frehse, L. Mathesen, C. Menghi, G. Pedrielli, M. Pouzet, S. Yaghoubi, Y. Yamagata, and Z. Zhang, “ARCH-COMP 2020 category report: Falsification,” in *ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20)*, vol. 74. EasyChair, 2020, pp. 140–152.
- [11] C. Molnar, “Interpretable machine learning: A guide for making black box models explainable,” <https://christophm.github.io/interpretable-ml-book/>, online. Accessed on 10 Dec. 2021.
- [12] T. M. Ribeiro, S. Singh, and C. Guestrin, “Anchors: High-precision model-agnostic explanations,” *AAAI*, 2018.
- [13] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, real-time object detection,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.
- [14] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the KITTI vision benchmark suite,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.
- [15] S. Shi, X. Wang, and H. Li, “PointRCNN: 3D object proposal generation and detection from point cloud,” 2018. [Online]. Available: <https://arxiv.org/abs/1812.04244>
- [16] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, “Learning deep features for discriminative localization,” *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [17] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-CAM: Visual explanations from deep networks via gradient-based localization,” *International Journal of Computer Vision*, vol. 128, no. 2, p. 336–359, 2019.
- [18] M. W. Craven and J. W. Shavlik, “Extracting tree-structured representations of trained networks,” in *NIPS*, 1995.
- [19] J. Van Der Waa, E. Nieuwburg, A. Cremers, and M. Neerincx, “Evaluating XAI: A comparison of rule-based and example-based explanations,” *Artificial Intelligence*, vol. 291, p. 103404, 2021.
- [20] M. T. Ribeiro, S. Singh, and C. Guestrin, “Why should I trust you?” *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

- [21] O. Maler and D. Nickovic, “Monitoring temporal properties of continuous signals,” in *FORMATS/FTRTFT*, ser. Lecture Notes in Computer Science, vol. 3253. Springer, 2004, pp. 152–166.
- [22] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.
- [23] A. Donzé and O. Maler, “Robust satisfaction of temporal logic over real-valued signals,” in *Formal Modeling and Analysis of Timed Systems*. Springer Berlin Heidelberg, 2010, pp. 92–106.
- [24] G. Fainekos and G. Pappas, “Robustness of temporal logic specifications for continuous-time signals,” *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262–4291, 2009.
- [25] G. Bombara, C.-I. Vasile, F. Penedo, H. Yasuoka, and C. Belta, “A decision tree approach to data classification using signal temporal logic,” in *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC. Association for Computing Machinery, 2016, p. 1–10.
- [26] Z. Kong, A. Jones, and C. Belta, “Temporal logics for learning and detection of anomalous behavior,” *IEEE Transactions on Automatic Control*, vol. 62, no. 3, pp. 1210–1222, 2017.
- [27] E. Asarin, A. Donzé, O. Maler, and D. Nickovic, “Parametric identification of temporal properties,” in *Proceedings of the Second International Conference on Runtime Verification*. Springer-Verlag, 2011, p. 147–160.
- [28] G. Bombara and C. Belta, “Offline and online learning of signal temporal logic formulae using decision trees,” *ACM Trans. Cyber-Phys. Syst.*, vol. 5, no. 3, 2021.
- [29] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, “A survey of Monte Carlo tree search methods,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.
- [30] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

- [31] L. Kocsis and C. Szepesvári, “Bandit based Monte-Carlo planning,” in *Machine Learning: ECML 2006*. Springer Berlin Heidelberg, 2006, pp. 282–293.
- [32] P. Auer, N. Cesa-Bianchi, and P. Fischer, “Finite-time analysis of the multiarmed bandit problem,” *Machine Learning*, vol. 47, no. 2–3, p. 235–256, 2002.
- [33] E. Kaufmann and S. Kalyanakrishnan, “Information complexity in bandit subset selection,” in *Proceedings of the Twenty-sixth annual Conference on Learning Theory (COLT 2013)*, ser. JMLR Workshop and Conference Proceedings, vol. 30. JMLR, 2013, pp. 228–251.
- [34] M. U. Chaudhry and J.-H. Lee, “MOTiFS: Monte Carlo tree search based feature selection,” *Entropy*, vol. 20, no. 5, 2018.
- [35] R. Gaudel and M. Sebag, “Feature selection as a one-player game,” in *Proceedings of the 27th International Conference on Machine Learning*. Omnipress, 2010, p. 359–366.
- [36] A. Yilmaz, O. Javed, and M. Shah, “Object tracking: A survey,” *ACM Comput. Surv.*, vol. 38, no. 4, p. 13–es, dec 2006. [Online]. Available: <https://doi.org/10.1145/1177352.1177355>
- [37] W. Bouachir and G.-A. Bilodeau, “Exploiting structural constraints for visual object tracking,” *Image and Vision Computing*, vol. 43, pp. 39–49, 2015. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0262885615001080>
- [38] S. Javed, M. Danelljan, F. S. Khan, M. H. Khan, M. Felsberg, and J. Matas, “Visual object tracking with discriminative filters and siamese networks: A survey and outlook,” 2021. [Online]. Available: <https://arxiv.org/abs/2112.02838>
- [39] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, and T.-K. Kim, “Multiple object tracking: A literature review,” *Artificial Intelligence*, vol. 293, p. 103448, 4 2021. [Online]. Available: <https://doi.org/10.1016%2Fj.artint.2020.103448>
- [40] Z. Pang, Z. Li, and N. Wang, “SimpleTrack: Understanding and rethinking 3D multi-object tracking,” 2021. [Online]. Available: <https://arxiv.org/abs/2111.09621>
- [41] X. Weng, J. Wang, D. Held, and K. Kitani, “3D Multi-Object Tracking: A Baseline and New Evaluation Metrics,” *IROS*, 2020.
- [42] G. Welch and G. Bishop, “An introduction to the Kalman Filter,” 1995.

- [43] H. W. Kuhn and B. Yaw, “The Hungarian method for the assignment problem,” *Naval Res. Logist. Quart.*, pp. 83–97, 1955.
- [44] H.-k. Chiu, A. Prioletti, J. Li, and J. Bohg, “Probabilistic 3D multi-object tracking for autonomous driving,” 2020. [Online]. Available: <https://arxiv.org/abs/2001.05673>
- [45] P. C. Mahalanobis, “On the generalized distance in statistics,” *Proceedings of the National Institute of Sciences (Calcutta)*, vol. 2, pp. 49–55, 1936.
- [46] T. Yin, X. Zhou, and P. Krähenbühl, “Center-based 3D object detection and tracking,” *CVPR*, 2021.
- [47] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, “Generalized Intersection over Union: A metric and a loss for bounding box regression,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 658–666.
- [48] E. Baser, V. Balasubramanian, P. Bhattacharyya, and K. Czarnecki, “FANTrack: 3D multi-object tracking with feature association network,” 2019. [Online]. Available: <https://arxiv.org/abs/1905.02843>
- [49] H.-k. Chiu, J. Li, R. Ambrus, and J. Bohg, “Probabilistic 3D multi-modal, multi-object tracking for autonomous driving,” 2020. [Online]. Available: <https://arxiv.org/abs/2012.13755>
- [50] A. Kim, A. Ošep, and L. Leal-Taixé, “EagerMOT: 3D multi-object tracking via sensor fusion,” 2021. [Online]. Available: <https://arxiv.org/abs/2104.14682>
- [51] X. Weng, Y. Wang, Y. Man, and K. Kitani, “GNN3DMOT: Graph neural network for 3D multi-object tracking with multi-feature learning,” 2020. [Online]. Available: <https://arxiv.org/abs/2006.07327>
- [52] N. Benbarka, J. Schröder, and A. Zell, “Score refinement for confidence-based 3D multi-object tracking,” *arXiv preprint arXiv:2107.04327*, 2021.
- [53] A. Mandelbaum and D. Weinshall, “Distance-based confidence score for neural network classifiers,” 2017. [Online]. Available: <https://arxiv.org/abs/1709.09844>
- [54] R. Altendorfer and S. Matzka, “A confidence measure for vehicle tracking based on a generalization of bayes estimation,” in *2010 IEEE Intelligent Vehicles Symposium*, 2010, pp. 766–772.

- [55] S.-H. Bae and K.-J. Yoon, “Robust online multi-object tracking based on tracklet confidence and online discriminative appearance learning,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1218–1225.
- [56] H. Wu, W. Han, C. Wen, X. Li, and C. Wang, “3D multi-object tracking in point clouds based on prediction confidence-guided data association,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 5668–5677, 2022.
- [57] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [58] C. Gallicchio, “Short-term memory of deep rnn,” 2018. [Online]. Available: <https://arxiv.org/abs/1802.00748>
- [59] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, “nuScenes: A multimodal dataset for autonomous driving,” *arXiv preprint arXiv:1903.11027*, 2019.
- [60] A. Ketenci and E. A. Gol, “Synthesis of monitoring rules via data mining,” in *2019 American Control Conference (ACC)*, 2019, pp. 1684–1689.
- [61] R. Guidotti, “Evaluating local explanation methods on ground truth,” *Artificial Intelligence*, vol. 291, p. 103428, 2021.
- [62] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer, “Policy compression for aircraft collision avoidance systems,” in *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, 2016, pp. 1–10.
- [63] A. Donzé, T. Ferrère, and O. Maler, “Efficient robust monitoring for STL,” in *Computer Aided Verification*. Springer Berlin Heidelberg, 2013, pp. 264–279.
- [64] T. Lattimore and C. Szepesvári, *Bandit Algorithms*. Cambridge University Press, 2020.
- [65] G. Bombara and C. Belta, “Signal clustering using temporal logics,” in *RV, 09 2017*, pp. 121–137.
- [66] G. E. Fainekos, S. Sankaranarayanan, K. Ueda, and H. Yazarel, “Verification of automotive control applications using S-TaLiRo,” in *2012 American Control Conference (ACC)*, 2012, pp. 3567–3572.
- [67] M. P. Owen, A. Panken, R. Moss, L. Alvarez, and C. Leeper, “ACAS Xu: Integrated collision avoidance and detect and avoid capability for UAS,” in *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*, 2019, pp. 1–10.



- [68] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer, “Reluplex: An efficient SMT solver for verifying deep neural networks,” 2017.
- [69] S. M. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” in *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 4765–4774.
- [70] E. Štrumbelj and I. Kononenko, “Explaining prediction models and individual predictions with feature contributions,” *Knowledge and Information Systems*, vol. 41, pp. 647–665, 12 2013.
- [71] L. Nenzi, S. Silveti, E. Bartocci, and L. Bortolussi, “A robust genetic algorithm for learning temporal specifications from data,” 2018.
- [72] N. Srinivas, A. Krause, S. Kakade, and M. Seeger, “Gaussian process optimization in the bandit setting: No regret and experimental design,” in *Proceedings of the 27th International Conference on International Conference on Machine Learning (ICML)*, 2010, p. 1015–1022.
- [73] L. De Raedt and K. Kersting, “Probabilistic inductive logic programming,” in *Probabilistic Inductive Logic Programming*. Springer, 2008, pp. 1–27.
- [74] B. Hoxha, A. Dokhanchi, and G. Fainekos, “Mining parametric temporal logic properties in model-based design for cyber-physical systems,” *International Journal on Software Tools for Technology Transfer*, vol. 20, pp. 79–93, 2018.
- [75] K. Bernardin and R. Stiefelhagen, “Evaluating multiple object tracking performance: The CLEAR MOT metrics,” *J. Image Video Process.*, vol. 2008, 2008. [Online]. Available: <https://doi.org/10.1155/2008/246309>
- [76] J. Pöschmann, T. Pfeifer, and P. Protzel, “Factor graph based 3D multi-object tracking in point clouds,” 2020. [Online]. Available: <https://arxiv.org/abs/2008.05309>
- [77] H.-N. Hu, Q.-Z. Cai, D. Wang, J. Lin, M. Sun, P. Krähenbühl, T. Darrell, and F. Yu, “Joint monocular 3D vehicle detection and tracking,” 2018. [Online]. Available: <https://arxiv.org/abs/1811.10742>
- [78] J.-N. Zaech, D. Dai, A. Liniger, M. Danelljan, and L. Van Gool, “Learnable online graph representations for 3D multi-object tracking,” 2021. [Online]. Available: <https://arxiv.org/abs/2104.11747>

- [79] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov, “Scalability in perception for autonomous driving: Waymo open dataset,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.
- [80] B. Zhu, Z. Jiang, X. Zhou, Z. Li, and G. Yu, “Class-balanced grouping and sampling for point cloud 3D object detection,” 2019. [Online]. Available: <https://arxiv.org/abs/1908.09492>
- [81] S. Pang, D. Morris, and H. Radha, “CLOCs: Camera-LiDAR object candidates fusion for 3D object detection,” 2020. [Online]. Available: <https://arxiv.org/abs/2009.00784>
- [82] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014. [Online]. Available: <https://arxiv.org/abs/1412.6980>