New Attack Detection Methods for Connected and Automated Vehicles

by

Shuhao Bian

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2025

© Shuhao Bian 2025

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis consists of part of one manuscript written for publication.

Research presented in Chapter 3:

In collaboration with Prof. Nasser L. Azad and Dr. Milad Farsi, I led the investigation presented in this chapter, which has been published in [7]. My role encompassed designing the study, conducting the literature survey, developing the algorithms, implementing the code, and drafting and revising the manuscript. I received valuable guidance and feedback from my co-authors throughout this process. The material in this chapter is primarily derived from my own research.

Abstract

Ensuring the security of Connected and Automated Vehicles (CAVs) against adversarial threats remains a critical challenge in cyber-physical systems. This thesis investigates attack detection methodologies and presents novel dual-perspective detection frameworks to enhance CAVs resilience.

We first propose a vehicle dynamics-based attack detector that integrates the Unscented Kalman Filter (UKF) with machine learning techniques. This approach monitors physical system behaviour and identifies anomalies when sensor readings deviate from predicted states. Our enhanced model captures nonlinear vehicle dynamics while maintaining real-time performance, enabling the detection of sophisticated attacks that traditional linear models would miss.

We develop a complementary trajectory-based detection framework that analyzes driving behaviour rationality to address the limitations of purely physics-based detection. This system evaluates vehicle trajectories within their environmental context, incorporating road conditions, traffic signals, and surrounding vehicle data. By leveraging neural networks for trajectory prediction and evaluation, our approach can identify malicious interventions even when attackers manipulate vehicle behaviour within physically plausible limits.

Integrating these two detection perspectives—one based on vehicle dynamics modelling and the other on trajectory rationality analysis—provides a comprehensive security framework that significantly improves detection accuracy while reducing false positives. Experimental results demonstrate our system's effectiveness against various attack vectors, including false data injection, adversarial control perturbations, and sensor spoofing attacks.

Our research contributes to autonomous vehicle security by developing a holistic detection approach that considers both immediate physical anomalies and broader behavioural inconsistencies, enhancing system resilience against increasingly sophisticated cyber-physical threats.

Acknowledgements

First and foremost, I extend my deepest gratitude to my supervisor, Prof. Nasser L. Azad, for his unwavering guidance throughout this research. From refining the research direction to polishing the manuscript, your expertise and meticulous attention to detail have been invaluable. Your passion for science has profoundly shaped my academic journey.

I am equally indebted to my laboratory colleagues: Milad, Sajjad, Aditya, Faizan, Devon, Mahsa and others. Countless late nights spent debugging code and analyzing data together have forged not only progress in this project but also lasting camaraderie. Special thanks to Milad for your critical assistance in paper writing, without which this work would have been incomplete.

Lastly, to my family: my parents, who have always trusted my choices and supported me in every possible way.

This thesis stands as a humble tribute to all who have walked alongside me on this path.

Dedication

To my parents, whose unconditional love and support have been my foundation; And to every soul who perseveres in the pursuit of science—may curiosity forever light the path through the unknown.

Table of Contents

Author's Declaration ii

Statement of Contributions iii

Abstract iv

Acknowledgements v

Dedication vi

List of Figures x

List of Tables xii

List of Abbreviations xiii

1 Introduction 1

1.1 Motivation 1

1.2 Contributions 2

1.3 Organization 3

2 Background and Related Work 5

2.1 Overview of Attack Types 5

2.1.1 Denial of Service Attack . . . 5
2.1.2 Replay Attack . . . 6
2.1.3 False Data Injection Attack . . . 6
2.1.4 Summary for Attack Types . . . 7
2.2 Dynamic Model-Based Detector . . . 7
2.2.1 Kalman Filter Based Detector . . . 8
2.2.2 Observer-Based Detector . . . 10
2.2.3 Dynamic Watermarking-Based Detector . . . 11
2.3 Machine Learning-Based Detector . . . 11
2.3.1 Data-Based Detector . . . 11
2.4 System Health Management . . . 13
2.5 Path Planning for CAVs . . . 15
2.6 Summary . . . 17

3 Kalman Filter-Based Detector **18**
3.1 Problem Formulation . . . 18
3.1.1 Attack Model . . . 19
3.1.2 Vehicle Model . . . 20
3.2 Algorithm Framework . . . 20
3.2.1 Unscented Kalman Filter . . . 21
3.2.2 Machine Learning . . . 23
3.2.3 Model Update Method . . . 24
3.2.4 Attack Detection (AD) Method . . . 24
3.3 Environment Setup . . . 25
3.3.1 Hardware Setup . . . 25
3.3.2 Simulation Setup . . . 25
3.3.3 Data Preprocessing . . . 27
3.3.4 Attack Implementation . . . 28

3.3.5 Parameters . . . 28
3.4 Results . . . 30
3.5 Summary . . . 32

4 Trajectory-Based Detector **34**
4.1 Algorithm Framework . . . 34
4.1.1 Cubic Spline . . . 34
4.1.2 Frenet Frame . . . 39
4.1.3 Model Framework . . . 41
4.1.4 Data Prepocessing . . . 45
4.2 Environment Setup . . . 45
4.2.1 Simulation Software . . . 46
4.2.2 SQLite . . . 46
4.2.3 Model Parameters . . . 47
4.3 Results . . . 47
4.3.1 Prediction of Trajectory . . . 47
4.4 Summary . . . 48

5 Conclusion and Future Work **56**
5.1 Conclusion . . . 56
5.1.1 Kalman Filter-Based Detector . . . 56
5.1.2 Trajectory-Based Detector . . . 57
5.2 Future Work . . . 57
5.2.1 Kalman Filter-Based Detector . . . 57
5.2.2 Trajectory-Based Detector . . . 58
5.2.3 Integrated Detection Framework . . . 59

References **60**

Glossary **66**

List of Figures

1.1 An example of CAVs systems: the CAVs system consists of sensors (such as GPS receivers, LiDAR, and radar), Electronic Control Unit (ECU), onboard computing units and Vehicle-to-Vehicle (V2V) [22]. 2

2.1 An Example of Replay Attack. 6
2.2 The Process of Kalman filter by [50]. 9
2.3 System diagram proposed by [45]. 13
2.4 Overview of the steps involved in ReSonAte proposed by [18]. 14
2.5 Bow-Tie diagram for Autonomous Vehicles (AVs) example proposed by [18]. 15
2.6 VectNet proposed by [17]. 16
2.7 UniAD pipeline proposed by [20]. 17

3.1 Flowchart of the proposed method. 21
3.2 CARLA Environment. 27
3.3 The results of the attack detector are as follows: for the Kalman filter, $a1$ represents the residual and Cumulative Sum (CUSUM) of residuals in the sliding window, $a2$ denotes the standard error of residuals in the sliding window, and $a3$ indicates the attack state; for the proposed method, $b1$ signifies the residual and CUSUM of residuals in the sliding window, $b2$ refers to the standard error of residuals in the sliding window, and $b3$ denotes the attack state of the UKF. 29
3.4 The metrics of different detectors are defined as follows: true positive rate (TP), true negative rate (TN), false positive rate (FP), and false negative rate (FN). 30

3.5 F1 Score of Kalman Filter and the Proposed Method. 31
3.6 The metrics of different detectors are defined as follows: true positive rate (TP), true negative rate (TN), false positive rate (FP), and false negative rate (FN). 33

4.1 An example of a 2-dimensional cubic spline interpolation through a set of waypoints. 36
4.2 An example of Frenet frame. 40
4.3 Processed dataset. 49
4.4 LimSim simulation envrionment [51]. 51
4.5 Illustration of the simulation setup structure with SQLite. 52
4.6 Initial results after 10 training rounds. 54
4.7 Final results using parameters in Table 4.5. 55

List of Tables

2.1 Extensive Overview of Intrusion Detection Systems by [42]. 12

3.1 Parameters of Kalman filter 28
3.2 Parameters of machine learning model 28
3.3 Optimal Thresholds and Confusion Matrix Values for Kalman Filter and the Proposed Method 32

4.1 Road feature encoder structure 41
4.2 Agent feature encoder structure 42
4.3 Feature detector architecture 43
4.4 Feature data structure for simulations 50
4.5 Feature detector model parameters 53

List of Abbreviations

AD Attack Detection viii, 1–3, 11, 15–17, 24, 26, 31, 34, 48, 56, 59

AI Artificial Intelligence 59

AVs Autonomous Vehicles x, 5, 13, 15, 30, 35–37, 39, 40

BEV Bird's Eye View 16

BTDs Bow-Tie Diagrams 13, 14

CAN Controller Area Network 13

CAVs Connected and Automated Vehicles iv, viii, x, 1, 2, 5, 11, 14–18, 25, 34, 58, 59

CNN Convolutional Neural Network 13

CPS Cyber-Physical Systems 1, 2, 5

CUSUM Cumulative Sum x, 24, 26, 29

DOS Denial of Service 1, 3, 5–7, 10, 18, 19, 24, 25, 28, 47, 56

DW Dynamic Watermarking 11

ECU Electronic Control Unit x, 2

EKF Extended Kalman filter 6, 7

FDI False Data Injection 1, 5–7, 10, 57

GNNs Graph Neural Networks 34

LSTM Long Short-Term Memory 13

ML Machine Learning 1, 3, 18, 23, 26, 32

MLP Multi-Layer Perceptron 23, 28

PWM Pulse Width Modulation 19, 28

ReLU Rectified Linear Unit 23

UKF Unscented Kalman Filter iv, x, 3, 10, 17, 18, 20, 21, 23–25, 29–32, 56, 58

UT Unscented Transformation 21

V2I Vehicle-to-Infrasturcture 1, 13

V2V Vehicle-to-Vehicle x, 1, 2, 5, 11

Chapter 1

Introduction

1.1 Motivation

The rapid development of CAVs has revolutionized the transportation industry, promising increased safety, efficiency, and convenience. Unlike conventional vehicles, CAVs rely heavily on sensors, Machine Learning (ML) algorithms, and wireless communication networks, making them highly susceptible to cyber-attacks. Real-world incidents have underscored the severity of this threat. In 2015, security researchers remotely hacked a Jeep Cherokee, illustrating how an attacker could take full control of steering, acceleration, and braking [12]. Similarly, in 2016, a Tesla Model S was compromised by exploiting vulnerabilities in its autopilot system [1]. In 2019, adversarial road markings deceived a Tesla's autopilot into misinterpreting lane boundaries [2]. Such attacks endanger not only passengers but also pedestrians and other vehicles, highlighting the urgent need for robust AD mechanisms.

CAVs are a quintessential example of Cyber-Physical Systems (CPS) [28], tightly integrating real-time computing with physical components. This interconnectedness, coupled with V2V and Vehicle-to-Infrasturcture (V2I) communication, expands the potential attack surface. Figure 1.1 depicts an example of CAVs [22]. Cyber threats like Denial of Service (DOS) attacks, False Data Injection (FDI), and sensor spoofing can compromise critical functionalities, leading to erratic behaviour or even catastrophic failures. Consequently, government agencies and industry bodies (e.g., NHTSA, ENISA) are increasingly mandating stringent cybersecurity protocols. Meeting these evolving regulations will require advanced intrusion detection and mitigation strategies to identify and neutralize threats in real-time. Meanwhile, in contrast to conventional information security approaches, CPS require a more holistic perspective that encompasses both computational and physical

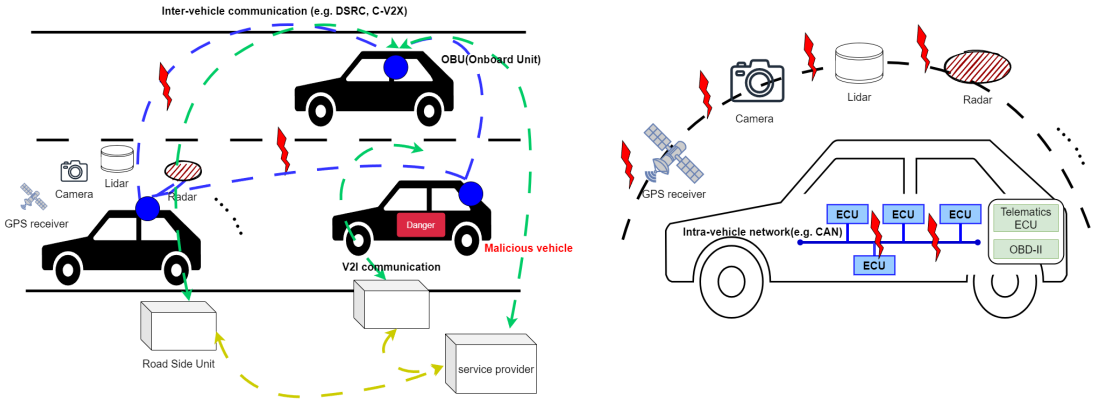


Figure 1.1: An example of CAVs systems: the CAVs system consists of sensors (such as GPS receivers, LiDAR, and radar), ECU, onboard computing units and V2V [22].

domains. By leveraging real-time feedback from sensors, actuators, and other physical components, CPS can detect and counteract malicious control inputs before they inflict tangible harm. This broader viewpoint not only enhances threat identification and mitigation but also bolsters the overall resilience and reliability of the system within real-world operational contexts.

This thesis proposes a novel AD method tailored to enhance the resilience of CAVs against cyber-attacks. By leveraging cutting-edge detection techniques, the approach aims to secure vital vehicle operations, safeguard passengers, and foster public trust in autonomous transportation.

1.2 Contributions

The contributions of our work are summarized as follows:

- **AD with Adaptive Algorithm:**
  - **Challenge in Literature:** Existing research predominantly uses linear vehicle models that neglect crucial nonlinearities and environmental context (road geometry, traffic regulations, other vehicles).
  - **Our Contribution:** We develop a refined adaptive AD algorithm to detect anomalies while maintaining real-time response.

- **Intent-Aware Security Framework:**
  - **Challenge in Literature:** Current detection mechanisms focus on immediate behavioural anomalies rather than the strategic objectives of attackers, missing sophisticated attacks designed to mimic normal behaviour.
  - **Our Contribution:** We introduce a novel detection framework that combines attack intent analysis with advanced path planning algorithms, a transformer-based architecture for modelling temporal-spatial relationships, and multi-horizon trajectory assessment to identify subtle, long-term attack strategies.
- **Context-Sensitive Anomaly Detection:**
  - **Challenge in Literature:** Existing systems operate in isolation from the environmental context, causing high false positive rates when legitimate responses to road conditions mimic attack patterns.
  - **Our Contribution:** Our system incorporates environmental awareness by processing road geometries, traffic patterns, and regulatory constraints to differentiate between genuine attacks and normal responses to complex driving conditions.

1.3 Organization

This thesis is organized as follows:

- **Chapter 2**: Reviews related works on autonomous vehicle security, categorizing AD methods into dynamic model-based approaches (such as Kalman filtering and observer-based methods), machine learning implementations, and traffic regulation-based analyses of vehicle pathway rationality. This chapter establishes the foundation for our two-pronged detection approach: a vehicle dynamics model utilizing UKF with machine learning and a neural network-based trajectory prediction system that incorporates traffic signals and historical path data.
- **Chapter 3**: Presents our Attack Detector based on the UKF and ML. This chapter details how we enhance traditional dynamic model monitoring to detect DOS attacks.
- **Chapter 4**: Introduces our machine learning approach that determines whether a vehicle is under attack by analyzing comprehensive contextual data, including ego vehicle information, road conditions, traffic signals, and surrounding vehicle data.

- **Chapter 5**: Summarizes conclusions and discusses potential future work, integrating insights from both detection approaches to create a more robust security framework for autonomous vehicles.

Chapter 2

Background and Related Work

This chapter provides an overview of various attack types, introduces the dynamic model for intrusion detection, and presents key concepts related to path planning for CAVs.

2.1 Overview of Attack Types

This section will introduce the attack types in CPS. In [13], attacks in CPS can be roughly divided into three parts: DOS attack, replay attack, and FDI attack. In this section, we will introduce these three types of attacks. The attack can target various devices, as shown in Figure 1.1, including ECU modules, sensors, and both inter- and intra-vehicle communications.

2.1.1 Denial of Service Attack

DOS[34] attacks are among the most widespread and damaging cyber threats, recognized since 2004. Saturating networks and systems with excessive traffic can degrade or completely disrupt service availability at minimal cost to the attacker. The widespread accessibility of powerful botnets and distributed resources has further intensified these attacks, making them a persistent challenge in the cybersecurity field.

In the context of AVs and CAVs, DOS attacks can cause severe operational disturbances, as these systems rely on uninterrupted communications for navigation, traffic updates, and V2V exchanges. The degradation of connectivity compromises safety-

critical functions and limits decision-making capabilities. Several recent works, such as [38, 32, 3, 5, 16], offer insights into attack classifications and defence mechanisms.

2.1.2 Replay Attack

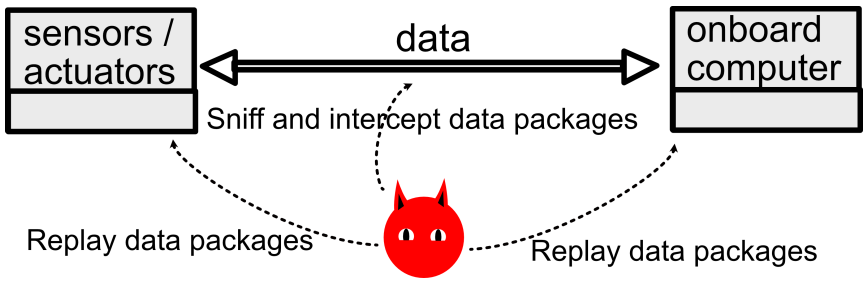


Figure 2.1: An Example of Replay Attack.

As shown in Figure 2.1, a replay attack, mentioned in [35, 45], is a specialized deception attack in which an adversary sniffs and intercepts legitimate sensor or actuator data. At a later time, the attacker substitutes the authentic, previously recorded data for the real-time data. By replaying signals that were once valid but are no longer current, the attacker deceives the system into believing that operations are proceeding as expected. This tactic can thwart security measures since the replayed data may pass basic integrity checks due to its original authenticity. As a result, replay attacks can significantly impact various applications, including industrial control systems and critical infrastructure, where accurate and timely data are crucial for safe and reliable operation.

2.1.3 False Data Injection Attack

The FDI[29, 37] attack is a deliberate and malicious manipulation of sensor measurements. In some cases, it also involves altering actuator signals. The goal is to mislead the control system or monitoring scheme. Unlike a DOS or replay attack, the attacker in an FDI attack has full knowledge of the system and greater access to its resources.

As discussed in [31], many detection methods fail to reliably detect FDI attacks:

The Extended Kalman filter (EKF) exhibits several vulnerabilities that sophisticated attackers can exploit. The filter inherently treats minor discrepancies as measurement noise, allowing attackers to inject minor false data that blends seamlessly with normal system uncertainties. Leveraging this characteristic, attackers can design stealthy attacks by

gradually modifying sensor data to align with the EKF's predictions, ensuring these modifications remain within expected uncertainty bounds. This vulnerability compounds over time, as the EKF incorporates false data into its state estimation process, allowing attackers to progressively increase their influence on the system. The threshold-based detection mechanism of EKF further exacerbates this weakness, as it only flags anomalies when residuals exceed predetermined limits. FDI attacks typically follow a calculated two-step approach: first introducing subtle, undetectable alterations, then gradually escalating the attack impact while remaining within the filter's tolerance thresholds, allowing malicious manipulations to grow in severity without triggering detection systems.

Due to these weaknesses, the system remains vulnerable. Attackers can always find ways to bypass detection. They can exploit its limitations indefinitely if they fully know the system. In such cases, **no detection method is entirely foolproof**.

2.1.4 Summary for Attack Types

The challenges posed by FDI attacks underscore the need for more robust detection mechanisms. However, as shown, even advanced filtering techniques, such as EKF, have fundamental limitations. This makes it challenging to ensure complete protection against such attacks.

Given these difficulties, this thesis focuses solely on the DOS attack. DOS attacks present a different type of security threat without full system knowledge. By studying DOS attacks, we aim to develop effective countermeasures that mitigate their impact.

2.2 Dynamic Model-Based Detector

Most existing methods primarily rely on network traffic analysis during training to identify patterns of normal operations, which are then used to detect anomalies in real-time. However, this approach often overlooks the dynamic aspects of vehicular subsystems. Vehicle dynamics, integral to vehicle design, enable the detection of anomalies through real-time comparison of observed behavior against model predictions. This dynamic model-based approach enhances intrusion detection by proactively identifying deviations and mitigating potential damages [23]. The accumulated difference or error detects the attack according to the Algorithm 1.

Input: Time series data $r_1, r_2, ..., r_T$, threshold h
Output: Change point detection alerts
Initialize $s \leftarrow 0$
for $t \leftarrow 1$ **to** T **do**
    $s \leftarrow \max(0, s + |r_t| - h)$ `// Non-negative accumulation`
    if $s \geq h$ **then**
        Trigger alert for change point detection
        $s \leftarrow 0$ `// Reset accumulator`
    end
end

Algorithm 1: CUSUM Change Point Detection Algorithm [36]

Integrating adaptable machine learning methods proves beneficial in accommodating changes in vehicle parameters, such as weight, tire conditions, and engine performance.

2.2.1 Kalman Filter Based Detector

As shown in Figure 2.2, the steps of the Kalman filtering can be divided into two parts: predict and update (in the figure, it's described as "correct"); in practice, we often add a detector based on the difference between the predicted value and the real value.

Predict The Kalman filter attempts to calculate the best estimate of a state $x \in \mathbf{R}^n$. Considering there is a discrete-time controlled process:

$$x_{k+1}^- = A_k \hat{x}_k + B u_k, \tag{2.1}$$

where A is the state transition matrix, x' is the prior estimation, and B is the control matrix. k denotes the discrete time step.

According to Equation 2.2, the covariance can be calculated by

$$P_{k+1}^- = A_k P_k A_k^T + Q_k, \tag{2.2}$$

where P is the covariance of the state, and Q is the process noise covariance.

Update We need measurement data to update x to make the estimation more precise. The Kalman Gain is computed by

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1}, \tag{2.3}$$

where H is the measurement matrix, R is the measurement noise. The estimate is updated by the sensor input $z_k \in \mathbf{R}^m$ according to

$$\hat{x}_k = \hat{x}_k^- + K(z_k - H_k \hat{x}_k^-). \tag{2.4}$$

Then, the error covariance is updated by

$$P_k = (I - K_k H_k) P_k^-. \tag{2.5}$$

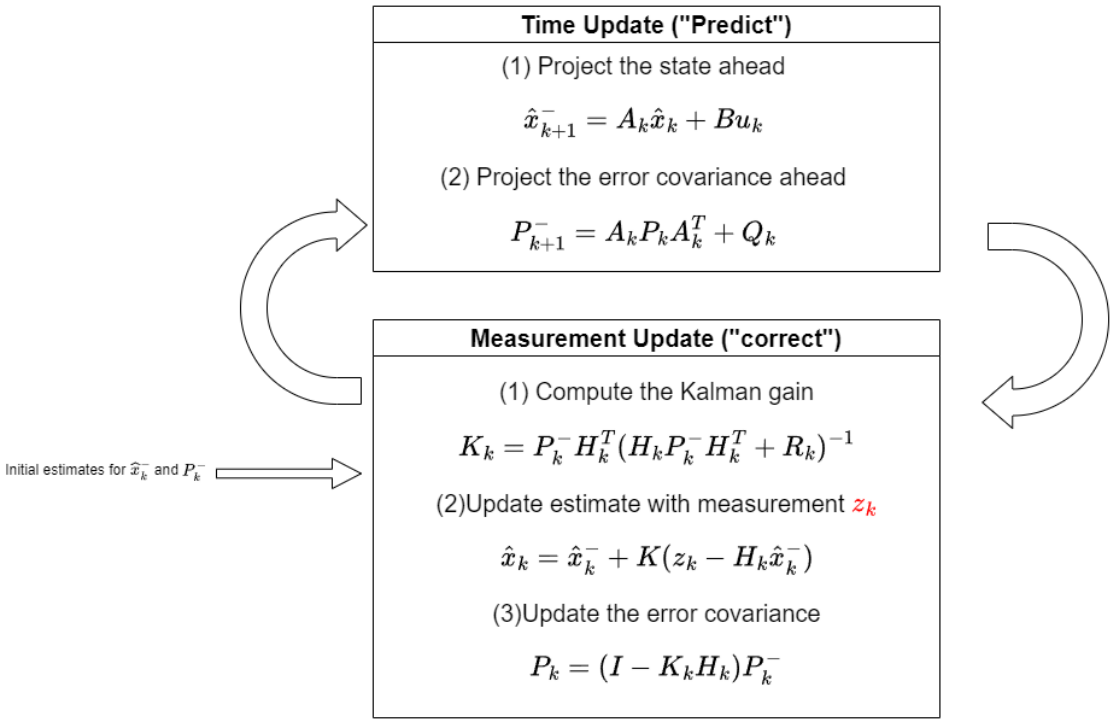


Figure 2.2: The Process of Kalman filter by [50].

The UKF approach, widely adopted in power and automotive systems, has also demonstrated robust anomaly detection capabilities. In power systems, the UKF ensures operational stability by accurately estimating grid states and identifying disruptions [15, 43]. In vehicular systems, the UKF significantly enhances resilience against DOS attacks by enabling precise real-time anomaly detection. For example, [56] employs the UKF to identify FDI attacks by comparing state estimates to those from weighted least squares-based techniques, noting significant deviations in attacked states.

Detection The detection mechanism is based on the approach described in Algorithm 1.

2.2.2 Observer-Based Detector

Before introducing the concept of an Observer-Based Detector, we will review how observers work in practical control applications. An observer provides estimates of system states that cannot be measured directly. It combines sensor measurements and mathematical models of the system. These estimates support feedback control, monitoring, or other tasks that require full state information.

Once we understand the basics of observers, we will explore how to integrate them with a detection mechanism. This combination forms an Observer-Based Detector. Specifically, the observer's state estimates can be combined with detection algorithms to enhance anomaly detection. This approach reveals abnormal system behaviours in real-time.

Observer Kordestani et al. (2021) [26] provide some details about the Observer-based Detector. In many practical control applications, measuring all states of a system directly is not feasible or cost-effective. An observer, also known as a state observer, is introduced to estimate unmeasured states using the system's inputs and measurable outputs. Consider a linear time-invariant (LTI) system:

$$\dot{x}_k = Ax_k + Bu_k, \quad y_k = Cx_k, \tag{2.6}$$

where $x_k \in \mathbf{R}^n$ is the state vector, $u_k \in \mathbf{R}^m$ is the control input, and $y_k \in \mathbf{R}^p$ is the measured output. The matrices A, B, C are constant and of appropriate dimensions.

An observer for this system can be designed as follows:

$$\dot{\hat{x}}_k = A\hat{x}_k + Bu_k + L\big(y_k - C\hat{x}_k\big), \tag{2.7}$$

where $\hat{x}_k \in \mathbf{R}^n$ denotes the estimated state, and L is the observer gain matrix. The difference $y_k - C\hat{x}_k$ is referred to as the estimation error signal, which drives the correction of $\hat{x}_k$.

Defining the estimation error $e_k = x_k - \hat{x}_k$, it can be shown that the choice of L affects the dynamics of e_k. For linear systems, if (A, C) is observable, there exists an L such that e_k converges to zero (or remains small under disturbances) as $t \to \infty$. Thus, the observer provides a reliable estimate of the full state vector, enabling state-feedback control or system monitoring even when direct measurement of all states is impossible.

Detection In this section, the AD component is akin to the Kalman filter-based detector, as they fully utilize the difference between the predicted signal and the observed value. According to formula 2.7, r can be calculated by

$$r_k = L\big(y_k - C\hat{x}_k\big). \tag{2.8}$$

Then, the attack is detected according to Algorithm 1.

2.2.3 Dynamic Watermarking-Based Detector

Among these various detection methods, Dynamic Watermarking (DW) is probably the most special detection method, which is an active AD method. In [45], the authors proposed a method of injecting a system with a sinusoidal signal of varying frequency to detect whether the output is compatible with the introduced signature, thereby judging whether the system is under a replay attack, as shown in Figure 2.3.

2.3 Machine Learning-Based Detector

In this section, we divide the anomaly detection methods into two parts. The first part relies on traditional data used in CAVs. The second part targets image-based data.

2.3.1 Data-Based Detector

Machine learning has found extensive applications in vehicular security, particularly anomaly detection in communication networks. One approach involves detecting anomalies in V2V

Approach	Learning Type	ML Technique	Count	Subtotal	Total
ID-Based	Unsupervised	Traditional ML	7	19	25
		Deep learning	9		
		Sequence learning	2		
		Hybrid	1		
	Supervised	Traditional ML	0	6	
		Deep learning	6		
		Sequence learning	0		
		Hybrid	0		
Payload-Based	Unsupervised	Traditional ML	2	21	29
		Deep learning	17		
		Sequence learning	1		
		Hybrid	1		
	Supervised	Traditional ML	1	8	
		Deep learning	7		
		Sequence learning	0		
		Hybrid	0		
CAN Frame-Based	Unsupervised	Traditional ML	2	12	46
		Deep learning	10		
		Sequence learning	0		
		Hybrid	0		
	Supervised	Traditional ML	12	34	
		Deep learning	19		
		Sequence learning	0		
		Hybrid	3		

Table 2.1: Extensive Overview of Intrusion Detection Systems by [42].

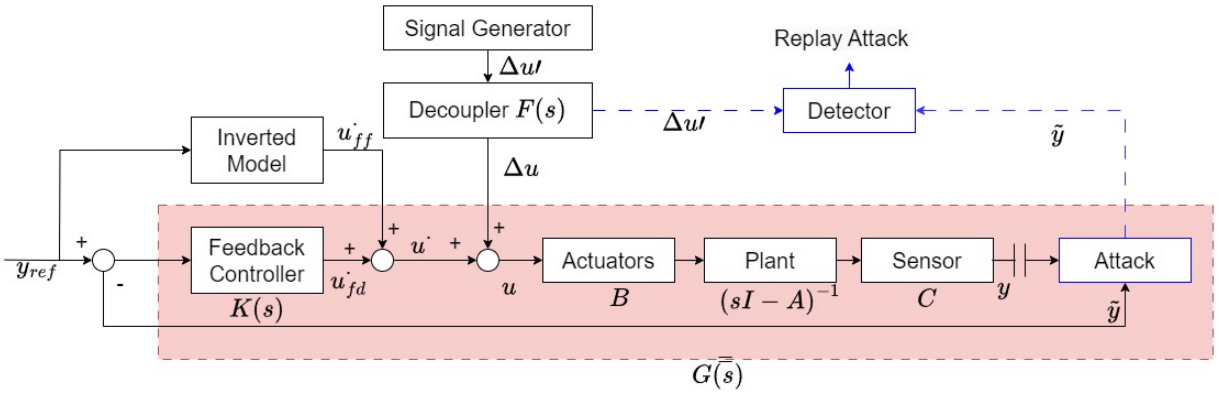


Figure 2.3: System diagram proposed by [45].

and V2I communications. For instance, [9] presents a machine learning-based detection system utilizing discriminative features trained on a collected dataset to evaluate various statistical and learning-based models. This method enables early detection, ensuring prompt countermeasures.

Intrusion detection within in-vehicle networks, such as the Controller Area Network (CAN) bus, has also garnered significant attention. Recent studies, including [4], employ deep learning techniques, such as Convolutional Neural Network (CNN)s and CNN-Long Short-Term Memory (LSTM) hybrids, to counter spoofing, flooding, and replay attacks. A comparative evaluation of these techniques, alongside traditional machine learning methods, is detailed in [6]. An extensive overview was conducted, as shown in Table 2.1.

2.4 System Health Management

A notable body of literature has addressed system health management for attack detection. For example, [18] presents the ReSonAte framework for dynamic risk assessment in AVs, Bow-Tie Diagrams (BTDs) to estimate and manage risks in real-time based on environmental conditions and system states. Traditional risk assessment techniques primarily rely on design-time methods such as hazard analysis, risk reduction, and assurance cases. However, these approaches do not adapt to dynamic operational contexts and runtime failures. Dynamic assurance approaches such as runtime certification, dynamic assurance cases, and modular safety certificates have been explored to address these limitations. These techniques extend static risk models by integrating real-time system monitoring to

dynamically update risk estimations. Nonetheless, several dynamic risk estimation methods experience state-space explosion, resulting in high computational costs and rendering them impractical for real-time use.

ReSonAte enhances previous methodologies by utilizing Bow-Tie Diagrams to model hazard propagation and control strategies while incorporating real-time system state monitoring and conditional probability modelling. By extending BTDs with attributes that capture dynamic environmental and system conditions, ReSonAte achieves adaptive risk estimation. Empirical evaluations in the CARLA simulator and underwater vehicle simulations demonstrate a strong correlation between estimated risk levels and actual system failures, supporting its effectiveness in CAVs safety management.

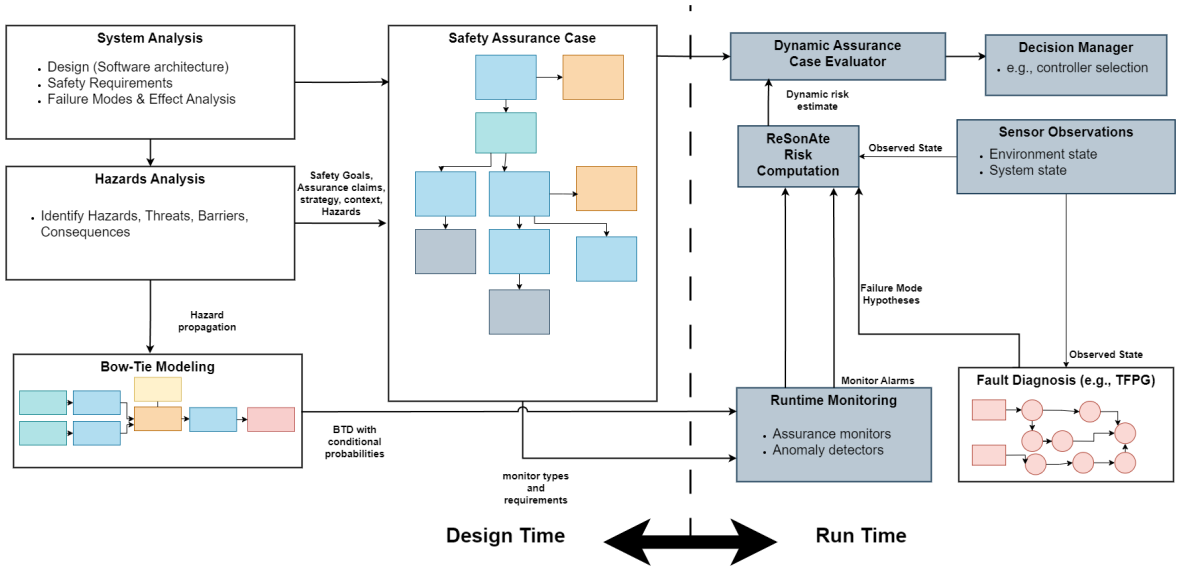


Figure 2.4: Overview of the steps involved in ReSonAte proposed by [18].

Figure 2.4 provides an overview of the ReSonAte framework, detailing its runtime risk assessment pipeline. Figure 2.5 illustrates the detailed implementation of the Bow-Tie Diagram, a key component in ReSonAte's hazard analysis methodology.

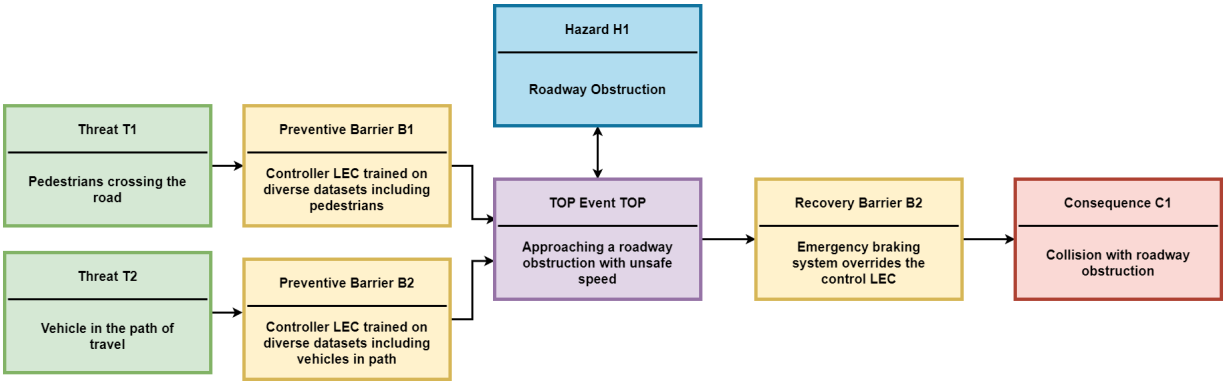


Figure 2.5: Bow-Tie diagram for AVs example proposed by [18].

2.5 Path Planning for CAVs

Chapter 3 presents a methodology for detecting vehicular attacks by evaluating the reasonableness of driving trajectories based on vehicle dynamics and traditional automotive models. This approach leverages established physical constraints and kinematic principles to identify anomalous behaviours that may indicate a compromised vehicle control system.

In **Chapter 4**, we introduce a novel detection mechanism that addresses a critical vulnerability in conventional approaches. Specifically, this chapter examines scenarios where attackers possess comprehensive knowledge of the vehicle's underlying models and can skillfully circumvent detection methods based on standard dynamic models. Our fundamental insight is that by focusing on the attacker's ultimate objectives rather than isolated behavioural anomalies, we can develop more robust detection frameworks that address the root cause of attacks. To our knowledge, there is relatively limited literature on AD methodologies incorporating path-planning considerations. Therefore, we approach this challenge from a trajectory prediction perspective, examining whether a vehicle has been compromised by analyzing its behaviour within specific time windows.

Before delving into the details of **Chapter 4**, it is pertinent to discuss the influential work presented in [17]. This research implements a sophisticated trajectory prediction framework that combines graph networks with Transformer architectures, as illustrated in Figure 2.6. The elegance of this approach lies in its exploitation of the inherent scalability of graph neural networks, enabling the seamless integration of traffic signals, surrounding road structures, and neighboring vehicle information to generate accurate trajectory predictions.

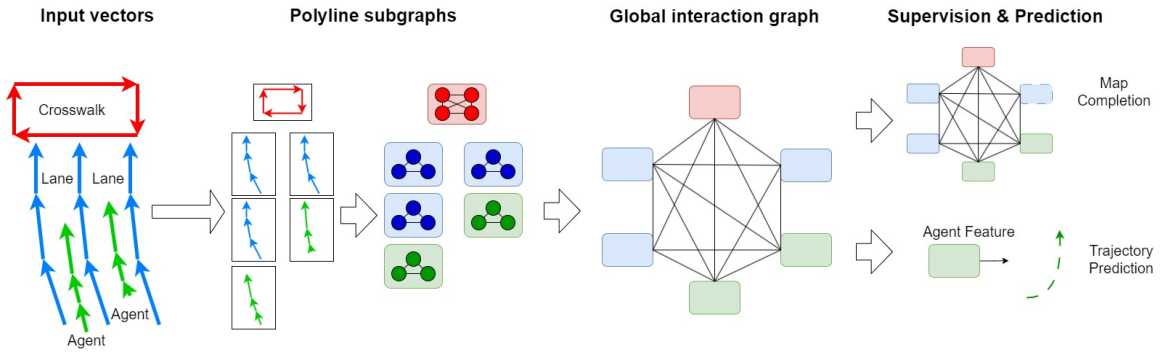


Figure 2.6: VectNet proposed by [17].

We have identified significant research potential in adapting this framework for AD purposes. With strategic modifications to its architecture, we believe this model can be repurposed to develop a robust path-planning-based AD system that overcomes the limitations of conventional methods.

In [20], the authors presented an innovative approach to vehicle trajectory planning by transforming input images into Bird's Eye View (BEV) representations. This transformation serves as the foundation for a comprehensive planning system that integrates multiple data streams, including predicted trajectories of surrounding vehicles, map processing, road structure analysis, and vehicle intent modeling. The system generates sophisticated path-planning solutions for CAVs through training on trajectory prediction tasks, as illustrated in Figure 2.7. The methodology extensively employs transformer model architectures to achieve high-fidelity trajectory predictions, leveraging the transformer's capacity to capture complex spatial-temporal relationships and contextual dependencies within the driving environment. The paper demonstrated how attention mechanisms can effectively weigh the relative importance of various environmental factors during the planning process, enabling more nuanced decision-making capabilities. This approach represents a significant advancement in unifying perception and planning frameworks, offering insights into how similar architectures could be adapted for anomaly detection in vehicle behaviour patterns.

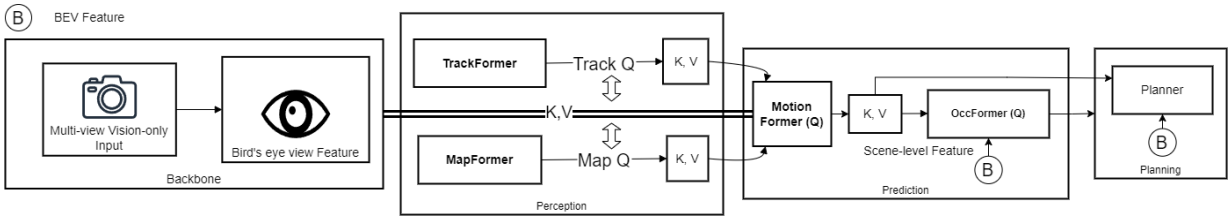


Figure 2.7: UniAD pipeline proposed by [20].

2.6 Summary

In this chapter, I focus on introducing several distinct types of attacks. According to the research background I have presented, AD methods can be categorized as follows: those based on dynamic models, such as Kalman filtering, observer-based approaches, and components of machine learning that implement AD through dynamic modelling. By integrating traffic regulations to analyze the rationality of autonomous vehicle pathways and actions, previous research has evaluated whether CAVs behaviour is reasonable based on historical trajectory data. Additionally, System Health Management approaches offer the potential for analyzing vehicle behavioural motivations to achieve effective AD.

Therefore, I propose AD methods in two distinct directions. The first approach is based on vehicle dynamics models, utilizing UKF combined with machine learning techniques to detect attacks. The second method utilizes neural networks to predict the trajectories of CAVs, incorporating traffic signals and historical vehicle path data to assess whether the current vehicle's path is reasonable. This assessment determines if the vehicle is potentially under attack, approaching the problem from the perspective of path rationality analysis.

Chapter 3

Kalman Filter-Based Detector

This chapter introduces a detection framework that utilizes the UKF combined with machine learning techniques to identify cyber-attacks in autonomous vehicles. By continuously monitoring vehicle dynamics against predicted states, the system can detect anomalies that may indicate malicious interference. The integration of ML enhances the detector's ability to distinguish between normal operational variations and actual attacks, providing robust protection for CAVs systems.

3.1 Problem Formulation

The following equations describe the system model

$$\begin{aligned} x^{k+1} &= f(x^k, u^k), \\ y^k &= h(x^k), \end{aligned} \tag{3.1}$$

where $x^k \in \mathbf{R}^n$, represents the state space of the vehicle, and $u^k \in \mathbf{R}^p$, denotes the control input. $y^k \in \mathbf{R}^m$ is the measurement vector. Moreover, $f : \mathbf{R}^n \times \mathbf{R}^p \to \mathbf{R}^n$ represents the state transition function that defines the next state x^{k+1} based on the current state x^k and the control input u^k. The function $h : \mathbf{R}^n \to \mathbf{R}^m$ denotes the measurement function that maps the current state x^k to the measurement vector y^k. k gives the time step. Next, we will model the DOS attack on various control loop elements.

3.1.1 Attack Model

Considering that the attacks are implemented in the cyber layer, we construct the cyber attack in a discrete space. Our attack model can be divided into two types: actuator attack and sensor attack. This paper simulates the attack-recovery-normal operation workflow by designing attack signatures in the form of Pulse Width Modulation (PWM). The attack pattern alternates between on and off states, creating a signature that oscillates between presence and absence of the attack signal.

Actuator Attack To model the effect of the attack, the dynamics in (3.1) are modified as

$$x^{k+1} = \begin{cases} f(x^k, a^k), & k \in \alpha, \\ f(x^k, u^k), & k \notin \alpha, \end{cases} \tag{3.2}$$

where we denote the set of time steps during which the attack is active using α. Moreover, we assume that DOS can arbitrarily affect a subset of components of u^k. This can be further described by

$$a^k[i] = \begin{cases} 0, & i \in \Gamma_a, \\ u^k[i], & \text{otherwise}, \end{cases} \tag{3.3}$$

where Γ_a denotes the set of indices corresponding to the actuators that are affected by the attack, and i ranges over the set $\Gamma_a \subseteq \{1, 2, \ldots, p\}$. Accordingly, a particular attack strategy can be represented by choosing nonempty sets α and Γ_a, which specify the time of the attack and the indices of targeted actuators, respectively.

Sensor Attack In this section, we define the sensor attack in a similar fashion. We modify the measurements relation given in Equation (3.1) to reflect the DOS attack on some specific set of sensors. The equation

$$y^k = \begin{cases} y_a^k, & k \in \beta, \\ h(x^k), & k \notin \beta, \end{cases} \tag{3.4}$$

switches the measurements to y_a^k so that the anomaly may be manipulated at different time steps specified by set β. Moreover, based on a particular attack pattern, a list of targeted sensors is given by Γ_s, with the attacked components being set to zero. This is shown below

$$y_a^k[i] = \begin{cases} 0, & i \in \Gamma_s, \\ y^k[i], & \text{otherwise}, \end{cases} \tag{3.5}$$

where $\Gamma_s \subseteq \{1, 2, \ldots, m\}$ denotes the set of indices corresponding to the components of y^k that are blocked by the attacker. Here, m refers to the dimension of the measurement vector.

3.1.2 Vehicle Model

To demonstrate the performance of the developed approach, we use the Kalman filter as a baseline approach for comparison in Section 3.4. We aim to obtain a sample-based technique, meaning that we treat the dynamics as a black box without using the analytical model. However, since the dynamic model is essential for running the Kalman Filter, we concisely present the equations of the vehicle in what follows.

According to [46], a vehicle dynamics model can be obtained as below. The longitudinal dynamics of the vehicle is given by

$$M_{vehicle}\dot{v}_h = M_{vehicle}a_f - r_{travel}, \tag{3.6}$$

where $M_{vehicle}$ is the mass of the vehicle, a_f is the traction force converted to acceleration and r_{travel} is the travel resistance. The model of r_{travel} can be described as

$$r_{travel} = r_{air}(v_h^2) + r_{roll}(v_h) + r_{accel}(\dot{v}_h) + r_{grad}(\theta), \tag{3.7}$$

where $r_{air} = \frac{1}{2}\rho C_d A v_h^2$ is the air drag, ρ is the air density, C_d is the drag coefficient, A is the frontal area of the vehicle, and v_h is the vehicle speed. The rolling resistance is given by $r_{roll} = C_r M_{vehicle} g v_h$, where C_r and g denote the rolling resistance coefficient and the acceleration due to gravity, respectively.

Then, the acceleration resistance is $r_{accel} = M_{vehicle}\dot{v}_h$, where $\dot{v}_h$ is the acceleration of the vehicle. Finally, $r_{grad} = M_{vehicle} g \sin(\theta)$, where θ is the slope angle of the road.

3.2 Algorithm Framework

Figure 3.1 presents the flow chart of UKF. In this section, we will explain how each part works respectively.

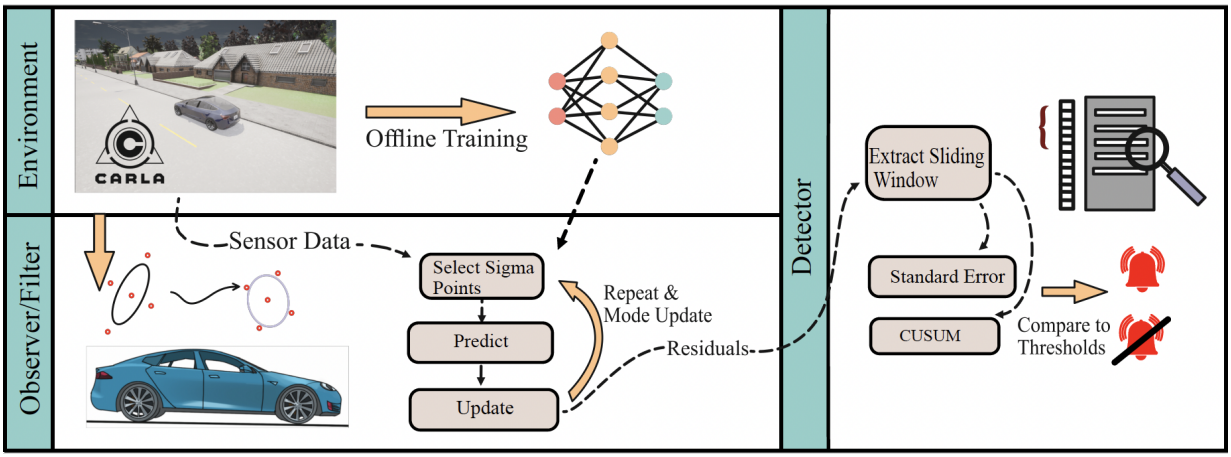


Figure 3.1: Flowchart of the proposed method.

3.2.1 Unscented Kalman Filter

Knowing the detailed model is unnecessary in the UKF. The UKF uses sigma points to sample the input and obtain the corresponding output. It is a method similar to the Monte Carlo approach but requires only a tiny number of sigma points. Next, we will briefly introduce the UKF.

Unscented Transformation

As described by [48], the Unscented Transformation (UT) is a technique developed for the generation of sigma points that are capable of undergoing nonlinear transformations expressed as $f(x^k, u^k)$. This approach is particularly valuable when dealing with a multitude of random vectors, each residing in an n-dimensional space ($x^k \in \mathbf{R}^n$), characterized by a mean $\bar{x}^k$ and a covariance matrix P^k.

Select Sigma Points

The sigma points χ_i^k are chosen to capture the true mean and covariance of the random variable x^k, enabling an accurate propagation through the nonlinear function $f(x^k, u^k)$. This selection ensures that the mean and covariance of the sigma points match $\bar{x}^k$ and P^k, respectively, providing an effective mechanism for estimating the statistical properties of x^{k+1}. The sigma points are selected according to

$$
\begin{cases}
\chi_0^k & = \bar{x}^k, \\
\chi_i^k & = \bar{x}^k + (\sqrt{(n+\lambda)P^k})_i \quad i = 1, ..., n, \\
\chi_i^k & = \bar{x}^k - (\sqrt{(n+\lambda)P^k})_i \quad i = n+1, ..., 2n, \\
W_0^{(m)} & = \frac{\lambda}{n+\lambda}, \\
W_0^{(c)} & = \frac{\lambda}{n+\lambda} + (1 - \phi^2 + \beta), \\
W_i^{(m)} & = W_i^{(c)} = \frac{1}{2(n+\lambda)} \qquad i = 1, ..., 2n.
\end{cases} \tag{3.8}
$$

According to [49], λ is a key parameter calculated using $\lambda = \phi^2(n+\kappa) - n$, ϕ determines the dispersion degree of σ points, and κ is typically $3 - n$. β is for integrating prior knowledge about x's distribution.

Predict

The predicted state and covariance are

$$
\begin{aligned}
\bar{x}^{k+1|k} &= \sum_{i=0}^{2n} W_i^{(m)} \chi_i^{k+1|k}, \\
P^{k+1|k} &= \sum_{i=0}^{2n} W_i^{(c)} \left(\chi_i^{k+1|k} - \bar{x}^{k+1|k} \right) \left(\chi_i^{k+1|k} - \bar{x}^{k+1|k} \right)^T, \\
\bar{y}^{k+1|k} &= \sum_{i=0}^{2n} W_i^{(m)} \mathcal{Y}_i^{k+1|k}, \\
P_{yy}^{k+1} &= \sum_{i=0}^{2n} W_i^{(c)} \left(\mathcal{Y}_i^{k+1|k} - \bar{y}^{k+1|k} \right) \left(\mathcal{Y}_i^{k+1|k} - \bar{y}^{k+1|k} \right)^T, \\
P_{xy}^{k+1} &= \sum_{i=0}^{2n} W_i^{(c)} \left(\chi_i^{k+1|k} - \bar{x}^{k+1|k} \right) \left(\mathcal{Y}_i^{k+1|k} - \bar{y}^{k+1|k} \right)^T,
\end{aligned} \tag{3.9}
$$

where $\chi_i^{k+1|k} = f(\chi_i^k, u^k)$, $\mathcal{Y}_i^{k+1|k} = h(\chi_i^k, u^k)$, for $i = 0, ..., 2n$.

Update

The update step can be defined as

$$\begin{aligned} K^{k+1} &= P_{xy}^{k+1} \left(P_{yy}^{k+1}\right)^{-1}, \\ r^{k+1} &= y^{k+1} - \bar{y}^{k+1|k}, \\ x^{k+1} &= \bar{x}^{k+1|k} + K^{k+1} r^{k+1}, \\ P^{k+1} &= P^{k+1|k} - K^{k+1} {P_{xy}^{k+1}}^{T}, \end{aligned} \tag{3.10}$$

where K is the Kalman gain, r is the residual. These values are updated when sensors provide new measurements.

3.2.2 Machine Learning

Obtaining the parameters of a vehicle might often be infeasible or economically prohibitive. In the update phase of the UKF, we employ ML techniques to predict the vehicle's subsequent state.

For this purpose, we selected the Multi-Layer Perceptron (MLP) network. This neural network receives control commands (u) — encompassing the throttle-brake and steering angle — as well as the vehicle's current state, which includes velocity, angular velocity, and acceleration, as its input. The network is designed to output the vehicle's acceleration for the next time step. Designed to predict the vehicle's acceleration at the next time step, it was necessary to account for the delay between the input commands and the resultant state changes. To address this, according to [54], we selected a model output that reflects the vehicle's acceleration 50 milliseconds after the input, ensuring the training data adequately captures the system's dynamics.

The MLP is structured with three distinct layers: an input layer comprising 5 units, a hidden layer containing 20 units, and a single-unit output layer. The activation function utilized within the hidden layer is the Rectified Linear Unit (ReLU).

With this predicted acceleration, we can determine the vehicle's upcoming velocity using Newton's second law, thereby integrating ML predictions seamlessly into the UKF's update mechanism for enhanced estimation accuracy.

3.2.3 Model Update Method

In order to make the detector adaptive, we deployed an algorithm that can make the detector update automatically. In the proposed detector, we updated the model of the vehicle when it got new sensor signals. To make sure the model is not updated when the attacker implements the DOS attack on the car model and is updated when the car is running normally, we update the learning rate according to (3.11).

$$l = 1 - \frac{1}{1 + e^{-S_{rate}\|r\|_2}}. \tag{3.11}$$

Here, l denotes the learning rate, r represents the residual of the car, S_{rate} is the scale of the residual, and $\|\cdot\|_2$ denotes the L_2 norm of a vector.

3.2.4 AD Method

As a widely used attack detector in many literature, CUSUM[30, 36] is selected as the detector. In [47], they also calculated a moving window average for the residual. In practice, there is often a gap between theoretical models and engineering applications. Therefore, we introduce W_{r1} and W_{r2} to fine-tune the target parameters. Additionally, we used two tests to detect whether the target was under attack.

Test 1 is shown by

$$s_1^k = \sum_{i=k-N+1}^{k} W_{r1} r^i, \tag{3.12}$$

where r is the residual updated by the UKF, $W_{r1}^T \in \mathbf{R}^m$, N is the length of the sliding window.

Test 2 is described as

$$s_2^k = \sum_{i=k-N+1}^{k} (r^i - \bar{r}) W_{r2} (r^i - \bar{r})^T, \tag{3.13}$$

where $\bar{r}$ is the average of the residuals within the sliding window, and $W_{r2} \in \mathbf{R}^{m\times m}$ is the weight matrix for the residual.

Then s_1 and s_2 will be compared with two thresholds, t_1 and t_2, respectively. Then, the alarm is triggered according to

$$A = \begin{cases} S_p, & s_1 > t_1 \text{ and } s_2 > t_2, \\ S_n, & s_1 \leq t_1 \text{ or } s_2 \leq t_2. \end{cases} \tag{3.14}$$

3.3 Environment Setup

3.3.1 Hardware Setup

All neural network training was conducted in Python on the Ubuntu operating system. The hardware configuration used included an AMD Ryzen 9 processor with 16 cores, clocked at 3.40 GHz, and 64GB of RAM. The Kalman filter and UKF were implemented using the FilterPy library [27], while the neural network was developed with PyTorch [40].

In the following subsection, we detail the data preprocessing methods, the implementation of attack scenarios, the simulation parameters and the resulting simulation outcomes.

3.3.2 Simulation Setup

Numerous software solutions have been created by companies and research groups to enhance CAVs research. This research aims to assess the effectiveness of adversarial attacks within a simulated setting, necessitating a photo-realistic 3D simulator.

CARLA[14] stands out as the most widely used open-source driving simulator. It is designed to support research and development in autonomous driving technologies, offering a realistic urban environment populated with various vehicles, pedestrians, and weather conditions. This setup enables the comprehensive testing of driving models in different scenarios.

Thanks to its flexible and customizable framework, CARLA has become an essential resource for researchers. It enables detailed evaluation of machine learning algorithms in areas such as perception, control, and adversarial robustness.

To demonstrate the efficacy of the proposed approach under DOS attacks, we chose it as our simulator, which can simulate real-world dynamics and generate sensor signals in real-time. This enables accurate and timely responses within the simulation environment. Moreover, to highlight the superiority of the proposed method, we compare it with the traditional Kalman filter.

Data: Sensor signal, Input signal
Result: Attack State: f_{ad} (0 for no attack, 1 for attack detected)
Initialisation: ;
Initialize the cumulative sum, $s_1, s_2 \leftarrow 0$;
Initialize the AD flag, $f_{ad} \leftarrow 0$;
Initialize the threshold value, *threshold* ;
Define the window size N ;
Initialize an empty list Q, to store the last N residuals ;
Detection Loop: ;
while *each new sensor signal input* y **do**
    Perform the selecting sigma points step;
    Perform the prediction step of UKF to estimate the next state ;
    Perform the update step of UKF with signal y to obtain residual r ;
    Append r to Q ;
    Update the UKF model with the ML algorithm `// using equation (3.11)` ;
    if *Size of* $Q > N$ **then**
        Remove the oldest residual from Q ;
    end
    Update cumulative sum using residuals within Q: $s_1 \leftarrow \sum Q$ `// using equation (3.12)` ;
    Update standard error using residuals within Q: $s_2 \leftarrow std(Q)$ `// using equation (3.13)` ;
    if $abs(s_1) > threshold_1$ ***and*** $s_2 > threshold_2$ **then**
        $f_{ad} \leftarrow 1$ `// attack detected immediately upon detection` ;
    else
        $f_{ad} \leftarrow 0$;
        Continue monitoring ;
    end
end

Algorithm 2: CUSUM algorithm with sliding window for AD

3.3.3 Data Preprocessing

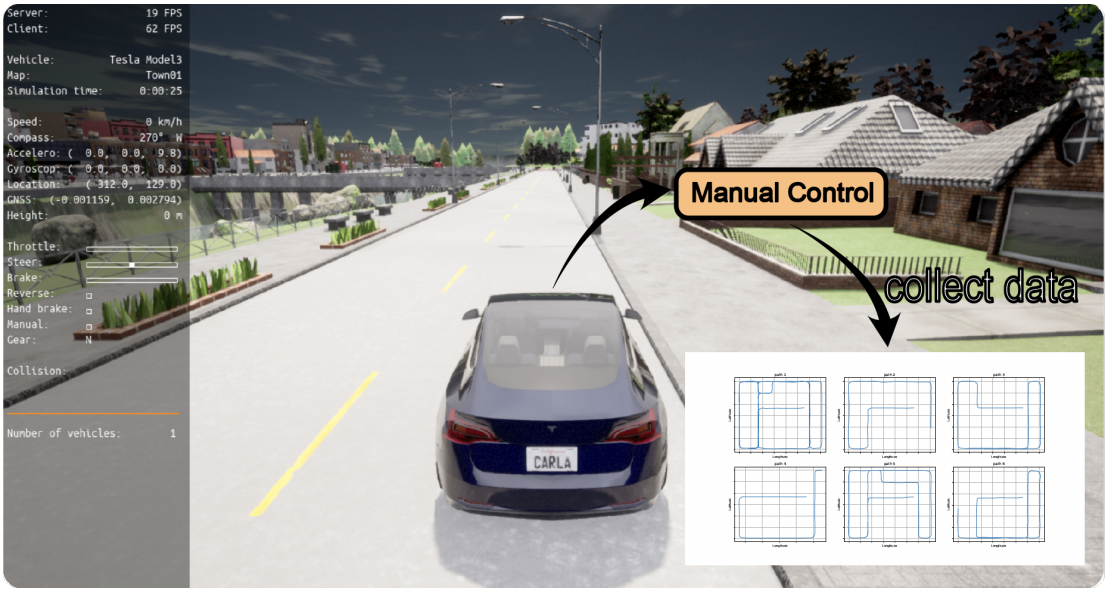


Figure 3.2: CARLA Environment.

Manual and autonomous driving data generate command signals (throttle, brake, and steering angle), which can be used as training data. We used manual control to generate data in the CARLA environment, obtaining representative data, as shown in Figure 3.2.

To achieve better training results, we need to ensure that the data are valid by removing all outliers.

The brake and throttle inputs are combined and normalized into a unified control signal ranging from 0 to 1 according to

$$u = \frac{T - B + 1}{2}, \tag{3.15}$$

where u is the unified control signal, $T \in [0, 1]$ is the throttle input, $B \in [0, 1]$ is the brake input, T_{max} is the maximum throttle value and B_{min} is the minimum brake value. When $u = 0$, it represents full braking; when $u = 1$, it represents full throttle.

3.3.4 Attack Implementation

We implemented the attack at 20 seconds in the simulation. To validate the algorithm, we chose a DOS attack signal as a PWM structure. When the signal is 1, it indicates that the monitor is blocked.

3.3.5 Parameters

To make a comparison, we chose a typical Kalman filter. The parameters of Equation (3.7) are shown as Table 3.1.

Table 3.1: Parameters of Kalman filter

Name	Value
r_{air}	68.9 N
r_{roll}	271.6 N
r_{grad}	0 N

If there is no slope, $r_{grad} = 0$. Therefore, the total travel resistance r_{travel} is given by $r_{travel} = r_{air} + r_{roll} + r_{grad} = 340.5N$. To make the detector more accurate, we have $W_{r1} = \begin{bmatrix} 1 & 0.01 & 0 \end{bmatrix}^T$, $W_{r2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0 \end{bmatrix}$, and we focus on the acceleration residual to detect the attack.

The learning parameters are detailed in Table 3.2. The optimizer used is Adam [25].

Table 3.2: Parameters of machine learning model

Name	Value
Learning rate	0.001
Train size/Total	0.8
Epochs	1000
MLP batch size	64
MLP epochs	1000

To improve the clarity of the results, we simplified the model; the value of t_2 is based on t_1, maintaining a fixed proportional relationship as shown by

$$t_2 = \gamma t_1, \tag{3.16}$$

where γ is a tuning parameter. In this simulation, γ is set to 0.04.

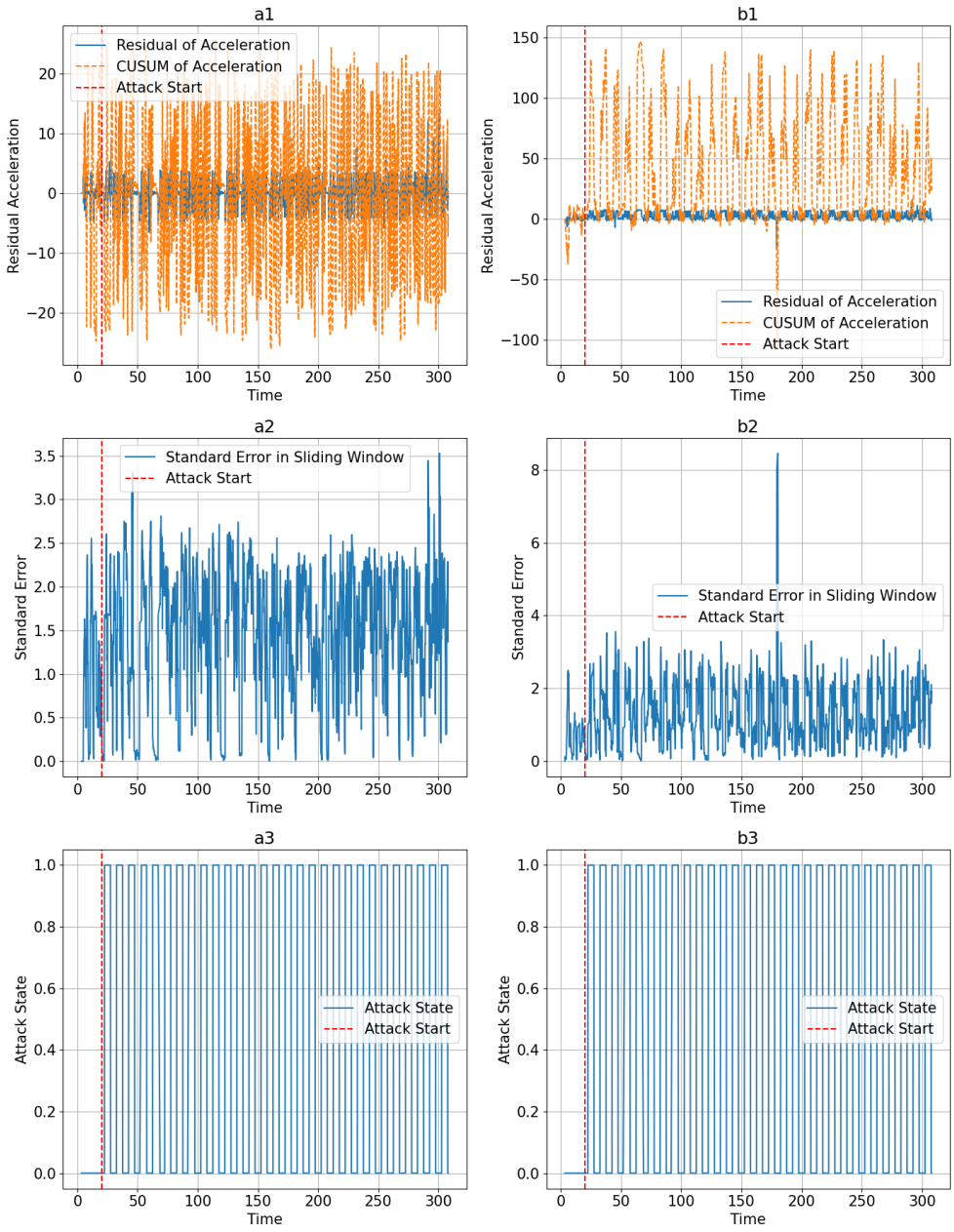


Figure 3.3: The results of the attack detector are as follows: for the Kalman filter, $a1$ represents the residual and CUSUM of residuals in the sliding window, $a2$ denotes the standard error of residuals in the sliding window, and $a3$ indicates the attack state; for the proposed method, $b1$ signifies the residual and CUSUM of residuals in the sliding window, $b2$ refers to the standard error of residuals in the sliding window, and $b3$ denotes the attack state of the UKF.

3.4 Results

This thesis evaluated the performances of the Kalman filter and UKF for detecting attacks in AVs. The experiments were conducted under two scenarios: actuator attack and sensor attack.

Actuator Attack

To validate the detector, we measure its performance using four metrics: false positive alarm rate, true positive alarm rate, false negative alarm rate, and true negative alarm rate.

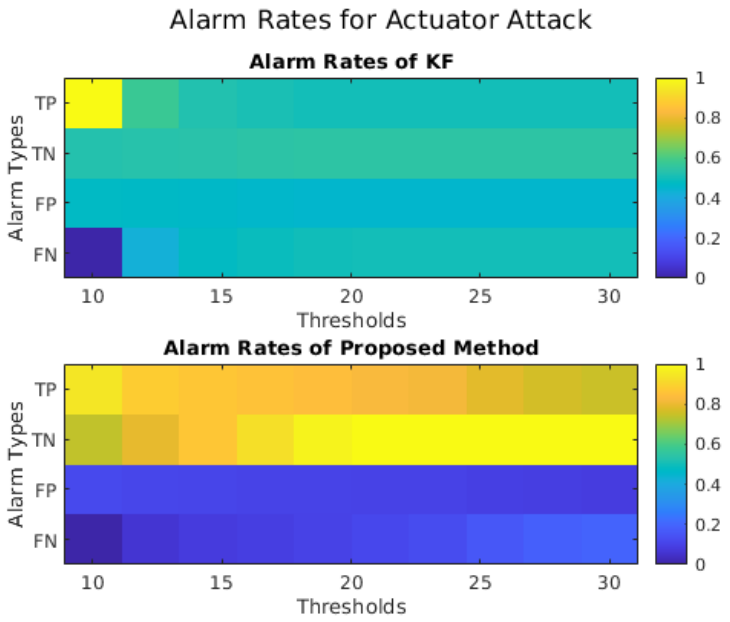


Figure 3.4: The metrics of different detectors are defined as follows: true positive rate (TP), true negative rate (TN), false positive rate (FP), and false negative rate (FN).

As shown in Figure 3.4, despite the threshold changes from 10 to 25, the proposed method's true positive and negative alarm rates are significantly higher than those of the Kalman filter when using the same threshold.

To achieve a more precise quantification of the research data, we utilized the F1-score [8]. As shown in Figure 3.5, the F1-score of the Kalman filter exhibits a decreasing trend,

whereas the F1-score of the UKF initially increases and then decreases. Based on this pattern, we selected the appropriate thresholds for the Kalman filter and the proposed method. The thresholds are 10 for the Kalman filter and 13.33 for the proposed method.

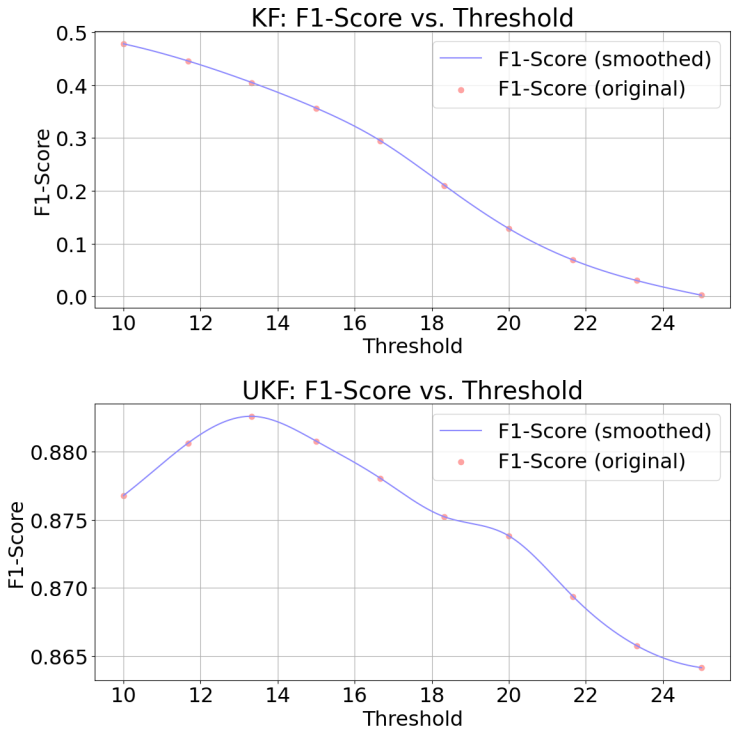


Figure 3.5: F1 Score of Kalman Filter and the Proposed Method.

Table 3.3 presents the optimal thresholds and corresponding values for the Kalman filter and the proposed method. In this table, all metrics of the proposed method significantly outperform those of the Kalman filter.

Sensor Attack

We obtained similar results for AD on the sensor, as shown in Figure 3.6. Both the true positive and true negative rates of the proposed UKF method are significantly higher than those of the Kalman filter when choosing the best F1-Score.

Table 3.3: Optimal Thresholds and Confusion Matrix Values for Kalman Filter and the Proposed Method

Metric	**Kalman filter**	**Ours**
Optimal Threshold	10.0	13.33
F1-Score	0.4784	0.8826
True Positives (TP)	42.88%	93.58%
False Positives (FP)	36.41%	18.48%
True Negatives (TN)	63.59%	81.52%
False Negatives (FN)	57.12%	6.42%

3.5 Summary

In this chapter, I introduced an attack detector based on the UKF and ML. Although the vehicle's dynamic model is repeatedly examined during operation, attackers can still evade detection methods when employing more stealthy attack behaviours. However, as long as the attacker's intention is to cause harm to drivers, pedestrians, or other vehicles on the road, we can similarly determine whether a vehicle is in a normal operational state by analyzing its driving trajectory. In the next chapter, I will introduce a machine learning method to determine whether the current vehicle is under attack by analyzing road conditions, ego vehicle information, and data from other vehicles on the road.

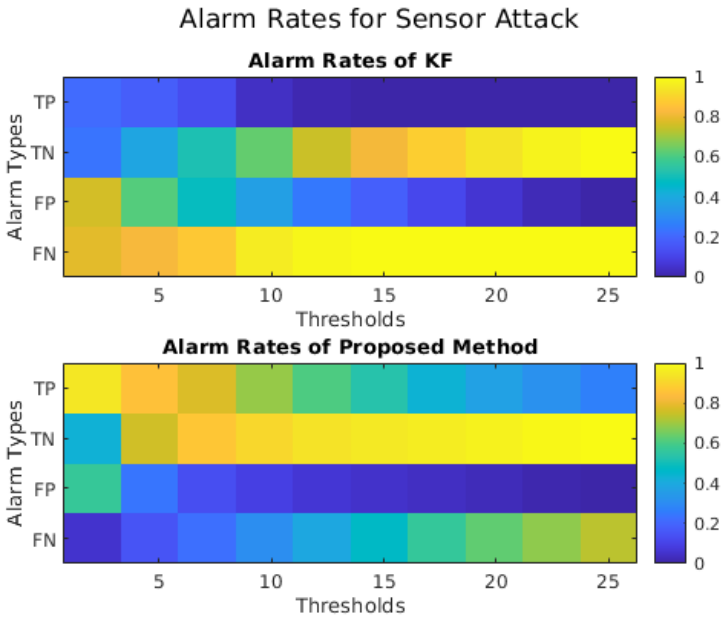


Figure 3.6: The metrics of different detectors are defined as follows: true positive rate (TP), true negative rate (TN), false positive rate (FP), and false negative rate (FN).

Chapter 4

Trajectory-Based Detector

4.1 Algorithm Framework

In this part, we present another new AD algorithm devised in this thesis. Because Graph Neural Networks (GNNs) are not ideally suited for conventional training methods, it was decided to abandon the use of GNNs in this experiment and instead employ a Transformer-based architecture. This decision was driven by the Transformer's proven ability to effectively capture long-range dependencies and handle sequential data, which aligns well with the standard training protocols used in this study. Moreover, the inherent scalability and robustness of the Transformer framework provided a more efficient and adaptable solution for the tasks at hand, ultimately enhancing both the training process and the overall model performance.

4.1.1 Cubic Spline

Cubic splines are widely used in numerical analysis and computer graphics to interpolate data points smoothly. This thesis employs cubic splines to generate continuous and smooth trajectories for CAVs. These splines are piecewise polynomial functions defined between consecutive data points, with each segment represented by a cubic polynomial.

According to [33], consider a one-dimensional spline set of $n+1$ points $(y_0, y_1, \ldots, y_n)$. A cubic spline can be described as:

$$S_i(t) = a_i + b_i t + c_i t^2 + d_i t^3, \tag{4.1}$$

where t is a parameter such that $t \in [0, 1]$ and $i = 0, \ldots, n-1$. The parameter t varies within each interval between consecutive data points. To ensure that the spline passes through each data point, the following boundary conditions must be satisfied:

$$\begin{aligned} S_i(0) &= y_i = a_i, \\ S_i(1) &= y_{i+1} = a_i + b_i + c_i + d_i. \end{aligned} \tag{4.2}$$

Continuity of the first and second derivatives at each interior knot is required to achieve smoothness across the entire spline. Specifically, the first derivative ensures tangential continuity, while the second derivative ensures curvature continuity. These conditions can be expressed as:

$$S_i'(1) = S_{i+1}'(0), \quad \text{for } i = 0, \ldots, n-2, \tag{4.3}$$

$$S_i''(1) = S_{i+1}''(0), \quad \text{for } i = 0, \ldots, n-2. \tag{4.4}$$

Additionally, natural cubic splines impose boundary conditions where the second derivatives at the endpoints are set to zero:

$$S_0''(0) = 0 \quad \text{and} \quad S_{n-1}''(1) = 0. \tag{4.5}$$

These conditions form a system of linear equations that can be solved to determine the coefficients a_i, b_i, c_i, and d_i for each spline segment. The natural cubic spline algorithm ensures that the resulting trajectory is smooth and optimally minimizes the curvature, which is crucial for AVs' stable and efficient operation.

While the one-dimensional cubic spline effectively interpolates data along a single axis, trajectory planning for AVs typically requires interpolation in two dimensions to account for the x and y coordinates. Extending the cubic spline to two dimensions involves independently applying the spline interpolation to each spatial dimension while ensuring parameter synchronization.

In a two-dimensional setting, the trajectory is parametrically represented as $(x(t), y(t))$, where both $x(t)$ and $y(t)$ are cubic spline parameterized by the same variable t. This ensures that the interpolation progresses consistently along both axes.

$$x(t) = a_i^x + b_i^x t + c_i^x t^2 + d_i^x t^3, \tag{4.6}$$

$$y(t) = a_i^y + b_i^y t + c_i^y t^2 + d_i^y t^3, \tag{4.7}$$

for $t \in [0, 1]$ and $i = 0, \ldots, n-1$.

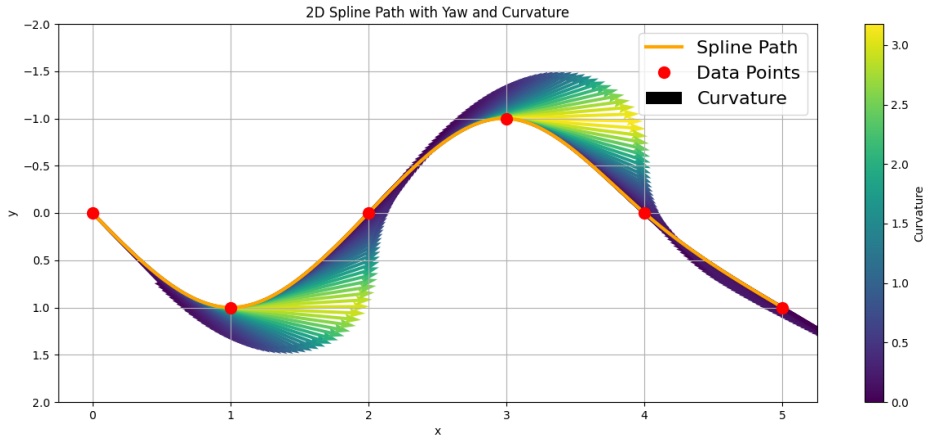


Figure 4.1: An example of a 2-dimensional cubic spline interpolation through a set of waypoints.

Figure 4.1 illustrates a 2-dimensional cubic spline interpolation through a set of waypoints, highlighting the smooth transitions and continuity properties that make cubic splines suitable for trajectory generation in AVs.

Algorithmic Implementation

The implementation of the natural cubic spline algorithm involves the following steps [10, 33]:

1. **Formulation of Equations:** Based on the boundary conditions and continuity requirements, formulate a system of linear equations to solve for the second derivatives at each waypoint.
2. **Solving the System:** Utilize efficient numerical methods, such as the tridiagonal matrix algorithm (Thomas algorithm), to solve the system for the second derivatives.
3. **Coefficient Calculation:** Once the second derivatives are known, compute the coefficients a_i, b_i, c_i, and d_i for each spline segment.

4. **Trajectory Generation:** Use the computed coefficients to evaluate the spline functions and generate the continuous trajectory for the AVs.

This structured approach ensures that the generated trajectories are both computationally efficient and meet the smoothness criteria necessary for high-performance autonomous navigation.

Matrix Formulation

The cubic spline problem can be converted into a matrix equation to solve for the spline coefficients systematically. This involves setting up a system of linear equations derived from the spline conditions and expressing them in matrix form.

Derivation of the System of Equations Consider the natural cubic spline conditions for $n+1$ data points $(y_0, y_1, \ldots, y_n)$. The primary goal is to determine the second derivatives $M_i = S''(x_i)$ at each waypoint x_i. The system of equations is derived from the continuity conditions of the first and second derivatives and the natural boundary conditions.

For each interval $i = 1, \ldots, n-1$, the second derivatives satisfy:

$$h_{i-1}M_{i-1} + 2(h_{i-1} + h_i)M_i + h_i M_{i+1} = 6\left(\frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}\right), \tag{4.8}$$

where $h_i = x_{i+1} - x_i$ is the interval length between consecutive data points.

Matrix Representation The above system can be expressed in matrix form as $\mathbf{AM} = \mathbf{B}$, where:

$$\mathbf{A} = \begin{bmatrix} 2h_0 & h_0 & 0 & \cdots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & \cdots & 0 \\ 0 & h_1 & 2(h_1+h_2) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & h_{n-2} \\ 0 & 0 & 0 & h_{n-2} & 2h_{n-2} \end{bmatrix}, \tag{4.9}$$

$$\mathbf{M} = \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_{n-1} \end{bmatrix}, \tag{4.10}$$

$$\mathbf{B} = \begin{bmatrix} 6\left(\frac{y_2-y_1}{h_1} - \frac{y_1-y_0}{h_0}\right) \\ 6\left(\frac{y_3-y_2}{h_2} - \frac{y_2-y_1}{h_1}\right) \\ \vdots \\ 6\left(\frac{y_n-y_{n-1}}{h_{n-1}} - \frac{y_{n-1}-y_{n-2}}{h_{n-2}}\right) \end{bmatrix}. \tag{4.11}$$

Here, $\mathbf{A}$ is a tridiagonal matrix, $\mathbf{M}$ is the vector of unknown second derivatives at the interior points, and $\mathbf{B}$ is the vector derived from the differences in the data points.

Given that $\mathbf{A}$ is a tridiagonal matrix, the system can be solved efficiently.

1. **Forward Elimination:** Modify the coefficients of the matrix $\mathbf{A}$ and the vector $\mathbf{B}$ to eliminate the lower diagonal elements.
2. **Backward Substitution:** Solve for the unknowns M_i starting from the last equation and substituting it in the previous equations.

Once the second derivatives $\mathbf{M}$ are determined, the coefficients c_i and d_i of the spline can be calculated as:

$$\begin{aligned} c_i &= \frac{M_{i+1} - M_i}{6h_i}, \\ d_i &= \frac{M_i}{2}, \\ b_i &= \frac{y_{i+1} - y_i}{h_i} - \frac{h_i}{6}(2M_i + M_{i+1}). \end{aligned} \tag{4.12}$$

The coefficient a_i is simply y_i, as established in Equation (4.2).

Overall, cubic splines are powerful tools for generating trajectories in AVs. They balance smoothness, flexibility, and computational efficiency.

4.1.2 Frenet Frame

The *Frenet frame* [53] is a curvilinear coordinate system that moves along a reference trajectory (or path) and is particularly useful in vehicle path planning and control. In this framework, a vehicle's position is expressed in terms of two coordinates:

- The longitudinal coordinate, s, which measures the arc length along the reference path.
- The lateral coordinate, d, which represents the perpendicular distance from the reference path.

This coordinate system offers several advantages:

- **Simplified trajectory tracking:** By representing the vehicle's motion relative to a smooth reference path, control algorithms can focus on minimizing lateral errors (d) and deviations in the longitudinal position (s).
- **Natural alignment with road geometry:** The Frenet frame inherently accommodates the curvature of roads, making it especially effective for modelling vehicle dynamics along curved paths.
- **Ease of derivative computation:** For trajectory planning, it is often necessary to compute the derivatives of the path, such as the velocity along s and the curvature. In the Frenet frame, these derivatives can be used to inform decisions regarding acceleration, braking, and steering.

The conversion between Cartesian coordinates (x, y) and Frenet coordinates (s, d) is typically performed using a reference path derived from methods such as cubic splines. Given a reference curve defined by the spline, the mapping is achieved through the following steps:

1. **Arc length calculation:** Compute the cumulative distance s along the reference curve.

2. **Projection:** For a given point (x, y) in the Cartesian frame, determine the point on the reference curve that minimizes the Euclidean distance to (x, y). The arc length corresponding to this closest point is taken as the longitudinal coordinate s.

3. **Lateral displacement:** The perpendicular distance from the point (x, y) to the reference curve at the corresponding s value is defined as the lateral coordinate d.

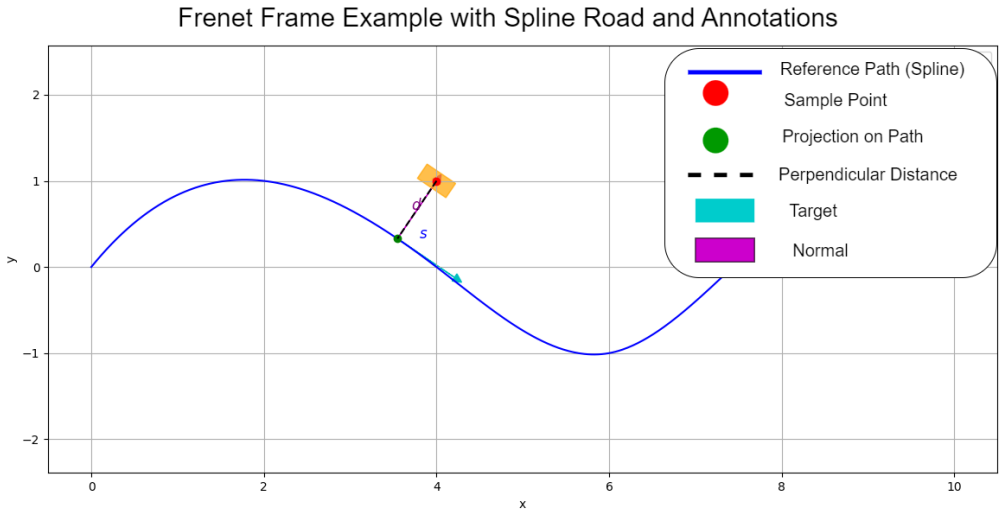


Figure 4.2: An example of Frenet frame.

Using the Frenet frame simplifies many aspects of trajectory generation and control. For example, when converting a planned trajectory expressed in Frenet coordinates back to the Cartesian frame, the reference curve's position, tangent (or heading), and curvature are used to reconstruct the (x, y) locations of the vehicle. This transformation can be expressed as:

$$\begin{aligned} x(s, d) &= x_{\text{ref}}(s) - d \sin \theta(s), \\ y(s, d) &= y_{\text{ref}}(s) + d \cos \theta(s), \end{aligned} \tag{4.13}$$

where $(x_{\text{ref}}(s), y_{\text{ref}}(s))$ is the point on the reference curve, and $\theta(s)$ is the tangent (yaw) angle of the reference curve at s.

In summary, the Frenet frame is an essential tool in AVs trajectory planning. It formulates control problems that are naturally aligned with road geometry and vehicle dynamics.

4.1.3 Model Framework

We propose a comprehensive, multi-modal, transformer-based vehicle trajectory prediction framework that encodes road features and agent interactions. Our model architecture effectively captures the complex dynamics of driving environments, incorporating both spatial and temporal dependencies to generate accurate multi-modal trajectory predictions. The model is also designed to detect potential adversarial scenarios or anomalous driving behaviours.

Road Feature Encoder

The Road Feature Encoder processes road geometry to understand the constraints and guidance provided by the road structure. Table 4.1 outlines the structure of this component.

Layer	**Description**
Road Embedding	Transforms raw road point coordinates into embedded representations with dimension $H/2$ followed by ReLU activation and projection to dimension H
Road Self-Attention	Multi-head attention mechanism (4 heads) that processes relationships within each road segment
Road Selection Attention	Multi-head attention mechanism (4 heads) that models relationships between different road segments
Final Transformer	Two-layer transformer encoder that further processes the selected important road features
Road Projection	Linear layer that assigns importance scores to each road segment for selective attention

Table 4.1: Road feature encoder structure

Agent Feature Encoder

The Agent Feature Encoder captures the behavioural patterns and interactions of surrounding vehicles. Table 4.2 describes this component.

Layer	Description
Feature Embedding	Linear projection of agent features (relative positions and orientation) to dimension H
Temporal Encoding	Learnable positional encoding for sequence time steps
Agent Encoding	Learnable positional encoding to differentiate between multiple agents
Temporal Transformer	Two-layer transformer encoder with 4 attention heads that processes temporal dependencies for each agent
Agent Transformer	Two-layer transformer encoder with 4 attention heads that models interactions between different agents

Table 4.2: Agent feature encoder structure

Feature Detector

The Feature Detector integrates road, agent, and vehicle-specific information to generate multi-modal trajectory predictions. Table 4.3 outlines this main component.

Component	Description
Feature Projections	Linear projections for different feature types: • Trajectory features: 2D positions $\rightarrow H$ dim • Dynamic features: velocity, acceleration, yaw $\rightarrow H$ dim • Traffic features: density, flow $\rightarrow H$ dim • Cost features: safety, efficiency metrics $\rightarrow H$ dim
Road and Agent Encoders	Previously described encoders integrated into the model
Road Curvature Extractor	Sequential layers that extract curvature information from road embeddings
Feature Projections	Non-linear projections with layer normalization for road and agent features
Encoder	Two-layer transformer encoder with 4 attention heads that fuses all feature representations
Decoder	Two-layer transformer decoder with 4 attention heads that generates trajectory features with temporal dependencies
Prediction Heads	Multiple Output Heads: • M trajectory heads: predict future 2D positions • M uncertainty heads: predict aleatoric uncertainty for each trajectory mode • Mode confidence: predicts probabilities for each trajectory mode • Attack classifier: identifies potential adversarial scenarios

Table 4.3: Feature detector architecture

Multi-Modal Trajectory Prediction

Our framework generates multiple potential future trajectories to account for the inherent uncertainty in driving scenarios. For each of the M trajectory modes, we predict:

- Future position coordinates (x, y) for F future time steps
- Position uncertainty estimates for each predicted point
- Confidence score indicating the likelihood of each trajectory mode

The model outputs are processed as follows:

$$\{\text{traj}_m, \text{uncert}_m, p_m\}_{m=1}^{M} = \Phi_\theta(X_{\text{hist}}, X_{\text{road}}, X_{\text{agent}}, X_{\text{dynamic}}), \tag{4.14}$$

where Φ_θ represents our model with parameters θ, X_{hist} is the historical trajectory, X_{road} contains road features, X_{agent} represents surrounding agent information, and X_{dynamic} captures vehicle dynamics.

Training Objective

The model is trained with a composite loss function that addresses multiple aspects of the prediction task:

$$\mathcal{L} = \mathcal{L}_{\text{traj}} + 0.5 \cdot \mathcal{L}_{\text{conf}} + 1.0 \cdot \mathcal{L}_{\text{attack}} + 0.1 \cdot \mathcal{L}_{\text{angle}} \tag{4.15}$$

where:

- $\mathcal{L}_{\text{traj}}$ is the trajectory prediction loss (MSE between predicted and ground truth positions)
- $\mathcal{L}_{\text{conf}}$ is the mode confidence loss (cross-entropy between predicted mode probabilities and best matching mode)
- $\mathcal{L}_{\text{attack}}$ is the attack classification loss (cross-entropy for anomaly detection)
- $\mathcal{L}_{\text{angle}}$ is the angle loss that ensures smooth trajectory predictions

4.1.4 Data Prepocessing

As shown in Table 4.4, I used simulation data that includes information about the ego vehicle, as well as the position information of surrounding vehicles, lane information, and the status of traffic lights on the ego vehicle's road. This information is used for machine learning training.

To better train the data, the vehicle's final point was used as a reference point and set as the origin of the coordinate system. Let's denote the final point as $P_n = (x_n, y_n)$ and the previous point as $P_{n-1} = (x_{n-1}, y_{n-1})$. The rotation vector can be defined as $\vec{v} = P_{n-1} - P_n = (x_{n-1} - x_n, y_{n-1} - y_n)$.

The rotation angle θ is calculated as

$$\theta = \arctan 2(\vec{v}_y, \vec{v}_x), \tag{4.16}$$

where x and y are the coordinate axes in the Cartesian coordinate system.

Using this angle, I constructed a rotation matrix R to transform all coordinate points:

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}. \tag{4.17}$$

For any point $P_i = (x_i, y_i)$ in the dataset (whether it belongs to other vehicles or lane markings), I first translated it relative to the reference point and then applied the rotation:

$$P_i' = R \cdot (P_i - P_n). \tag{4.18}$$

This transformation standardizes all data points in a consistent coordinate system where the ego vehicle's final position is at the origin $(0, 0)$, and its trajectory is aligned with a standard direction. This preprocessing step significantly improves the model's ability to learn patterns across different scenarios.

The processed data is shown as Figure 4.3.

4.2 Environment Setup

The hardware settings are the same as in Section 3.3.1. In addition to that, more setups will be introduced in this section.

4.2.1 Simulation Software

In addition to CARLA, we choose LimSim [52] as our decision-planning-oriented simulation software.

As shown in Figure 4.4, LimSim is a Long-term Interactive Multi-scenario Traffic Simulator, which is a comprehensive simulation system designed to model fine-grained dynamic traffic scenarios in urban environments. It addresses limitations in existing simulators by supporting long-term continuous simulations, diverse vehicle behaviours, and interactive multi-vehicle planning. LimSim incorporates modules for road network construction, multi-source traffic flow generation, joint decision-making and planning, and multi-dimensional scenario analysis. This enables realistic testing of autonomous driving algorithms and facilitates transportation planning. By integrating macroscopic and microscopic simulations, LimSim strikes a balance between real-time efficiency and simulation fidelity, making it a versatile tool for research and practical applications in traffic management and autonomous driving.

4.2.2 SQLite

SQLite is a powerful embedded relational database management system implemented as a compact C library by D. Richard Hipp. It supports a large subset of SQL-92, multiple tables and indexes, transactions, views, triggers, and numerous client interfaces and drivers. It's completely self-contained, requires fewer than 25,000 lines of ANSI C code, and is free for any purpose. It's fast, efficient, and highly portable, running on platforms ranging from ARM/Linux to SPARC/Solaris.

As shown in Figure 4.5, the architecture implements SQLite as an intermediary layer between the simulation engine and the data processing pipeline. The simulation engine writes event data, state transitions, and performance metrics directly to an in-memory SQLite database during execution. This database maintains a well-defined schema with indexed tables, enabling efficient querying and allowing for the rapid insertion of new data points without compromising the simulation's performance.

Offline processing tools can directly query the SQLite database using standard SQL interfaces available in most programming languages, thereby eliminating the need for custom parsing logic. This streamlined workflow enables rapid iteration between simulation runs and model training, allowing for the efficient evaluation of different approaches and parameters.

4.2.3 Model Parameters

Table 4.5 presents the parameters for my machine learning model, along with a brief description of the key parameters. The model architecture utilizes a transformer-based approach with 3 layers and 4 attention heads, operating on historical sequence data of 10 time steps to predict 10 future steps across 3 possible modes. The hidden dimension of 256 provides sufficient representational capacity while maintaining computational efficiency. The training was conducted over 1000 epochs with a learning rate of 0.001 and gradient clipping at 1.0 to ensure stable convergence. A validation split of 20% was used to monitor generalization performance throughout the training process.

4.3 Results

4.3.1 Prediction of Trajectory

Figure 4.6 presents a comparison between the predicted trajectories and the actual vehicle trajectories after the initial training rounds. As can be observed, the vehicle trajectory predictions are relatively inaccurate. This lack of precision is attributed to the insufficient number of training iterations.

In Figure 4.7, I increased the number of predicted paths to five and trained the network using the parameters listed in Table 3.2. The results demonstrate that vehicle trajectories can generally follow the predicted paths. In the figure, "Type 1" represents a DOS attack causing the vehicle to perform an "emergency stop," while "Type 2" represents a DOS attack preventing the vehicle from braking. When these attacks occur, the predicted probability of attack increases significantly.

In these examples, although the model often approximates the vehicle's trajectory reasonably well, it consistently assigns higher confidence to straight-line motion, reflecting the fact that, in most training cases, vehicles tend to drive in a straight line. At times, the model even ascribes a higher probability to an abrupt stop than to a genuine sudden-braking attack. This happens because the deceleration patterns leading up to a sudden stop can closely resemble those of a true sudden-braking event, causing the detector to lean toward the "attack" hypothesis.

However, given that this is an exploratory study, the vehicle trajectories often deviate from the actual paths and sometimes follow unrealistic routes. As a result, the attack detection accuracy has yet to reach the desired level.

4.4 Summary

In this chapter, I introduced a machine learning approach to determining whether the current vehicle is under attack by analyzing ego vehicle data, road information, traffic signals, and data from other vehicles. Combined with the previous chapter, I presented AD methods from two perspectives: vehicle dynamics modelling and trajectory rationality analysis.

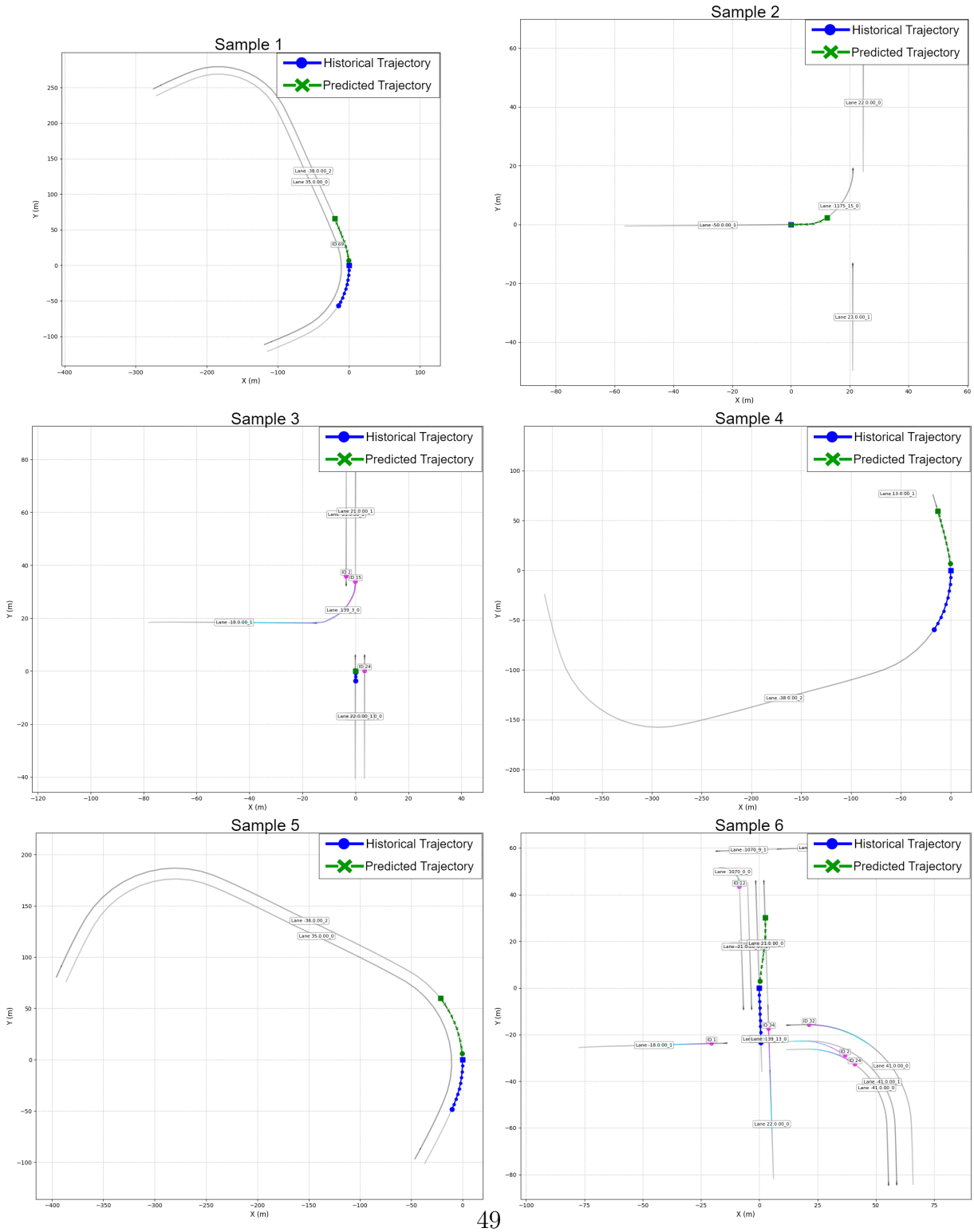


Figure 4.3: Processed dataset.

Table 4.4: Feature data structure for simulations

Category	**Feature**	**Dimensions**	**Description**
Historical Data	historical_trajectory	$[T_{obs}, 2]$	Ego vehicle's observed trajectory points
	historical_speed	$[T_{obs}]$	Ego vehicle's speed values
	historical_acceleration	$[T_{obs}]$	Ego vehicle's acceleration values
	historical_yaw	$[T_{obs}]$	Ego vehicle's yaw angles (rotation-adjusted)
Traffic Features	traffic_light_time	$[T_{obs}]$	Time elapsed since traffic light state change
	traffic_light_state	$[T_{obs}]$	State of traffic lights during observation
Agent Features	agent_id	[1]	Unique identifier for each agent
	time_steps	$[T_{obs}]$	Temporal features for each time step
	positions	$[T_{obs}, 2]$	Agent positions
	yaws	$[T_{obs}]$	Agent orientations
	lane_ids	$[T_{obs}]$	Lane identifiers for agent locations
	traffic_light_states	$[T_{obs}]$	Traffic light states affecting agent
	road_types	$[T_{obs}]$	Type of road segment for each time step
	current_relative_position	[2]	Final observed relative position
Road Features	lane_points	$[n, 2]$	Sequence of points representing lane centerline
Labels	trajectory	$[T_{pred}, 2]$	Future trajectory points (ground truth)
	attack_type	[1]	Classification of attack behavior

Note: T_{obs} represents the observation horizon (historical time steps), T_{pred} represents the prediction horizon, and n is the variable number of lane points.

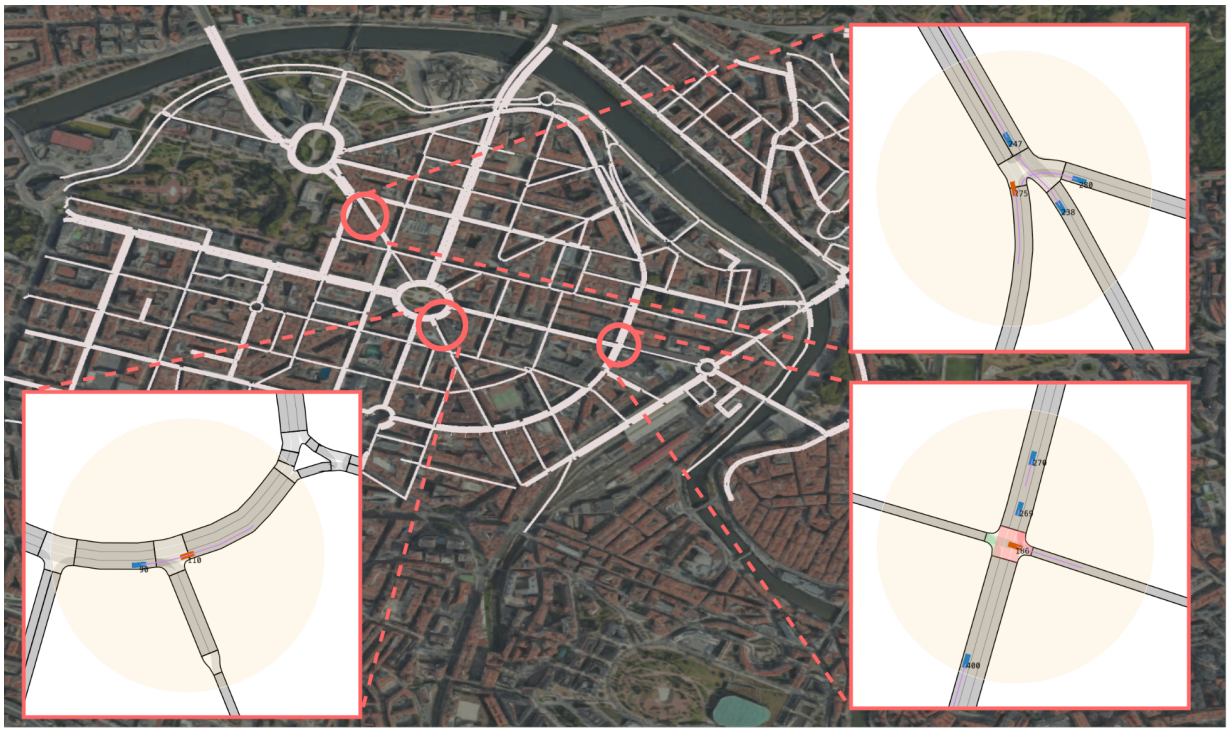


Figure 4.4: LimSim simulation envrionment [51].

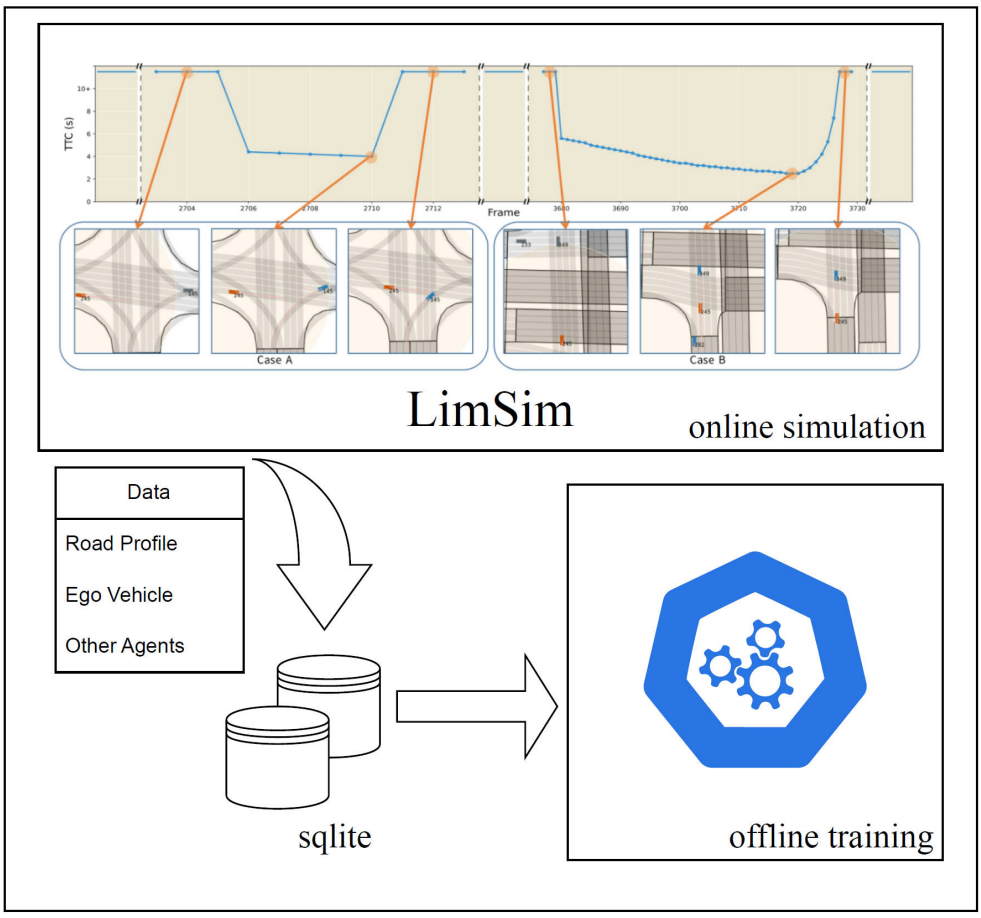


Figure 4.5: Illustration of the simulation setup structure with SQLite.

Table 4.5: Feature detector model parameters

Category	Parameter	Default Value	Description
Data	val_split	0.2	Proportion for validation
Model	hidden_dim	256	Hidden dimension
	num_heads	4	Number of attention heads
	num_layers	3	Number of transformer layers
	max_road_num	8	Maximum number of roads
	output_roads	2	Number of output roads
	max_agent_num	4	Maximum number of agents
	seq_len	10	Sequence length for historical data
	num_future_steps	10	Number of future prediction steps
	num_modes	5	Number of prediction modes
	num_attack_classes	3	Number of attack classes
	dropout	0.1	Dropout rate
Training	batch_size	8	Batch size for training
	epochs	1000	Number of training epochs
	lr	0.001	Learning rate
	weight_decay	1e-5	Weight decay for optimizer
	lr_scheduler	False	Use learning rate scheduler
	lr_step_size	10	Step size for StepLR scheduler
	lr_gamma	0.5	Gamma for StepLR scheduler
	clip_grad	1.0	Gradient clipping value
Other	seed	42	Random seed for reproducibility
	log_interval	10	Interval for logging training progress
	save_interval	5	Save interval (in epochs)
	no_cuda	False	Disable CUDA even if available
	plot	False	Plot training and validation losses
	resume	None	Path to checkpoint to resume training

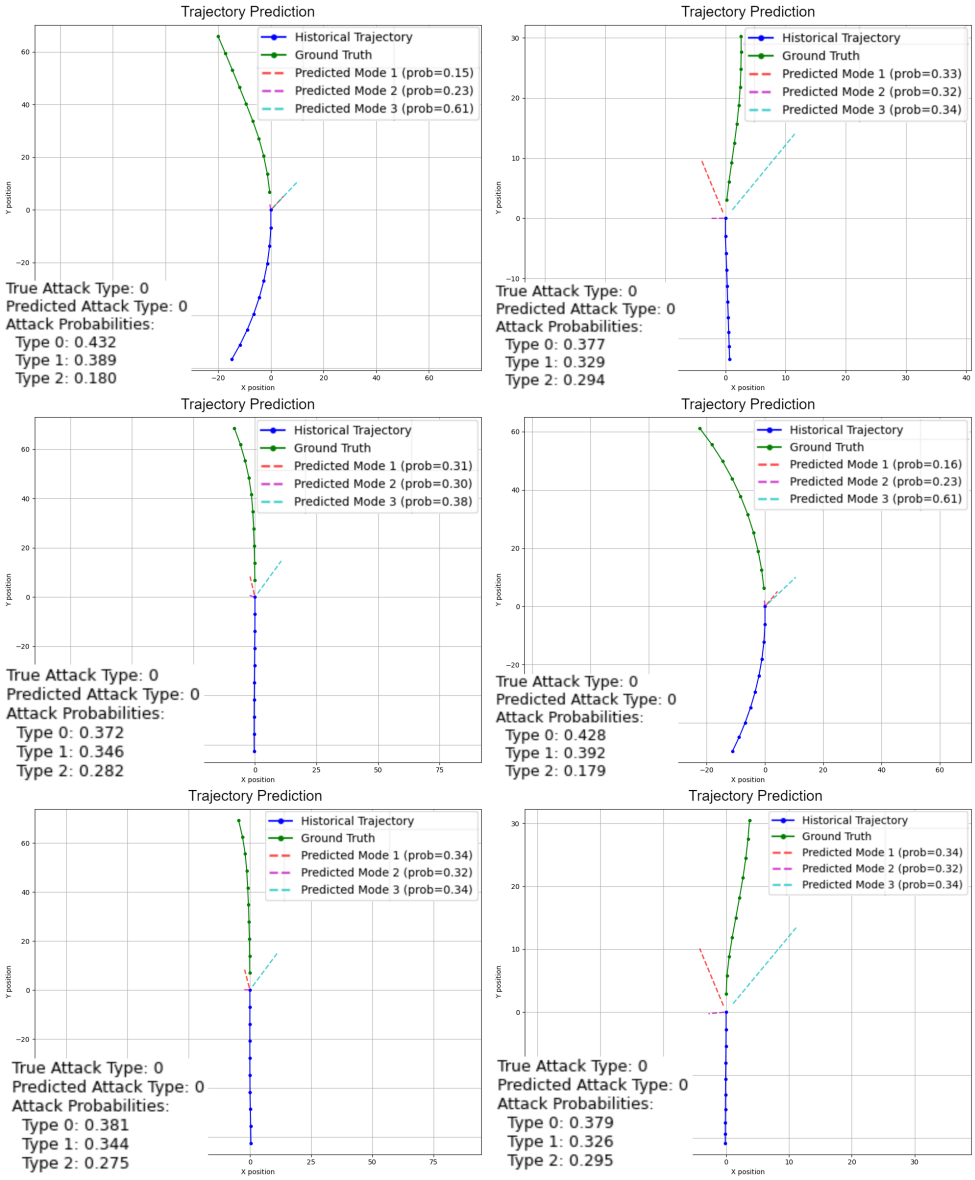


Figure 4.6: Initial results after 10 training rounds.

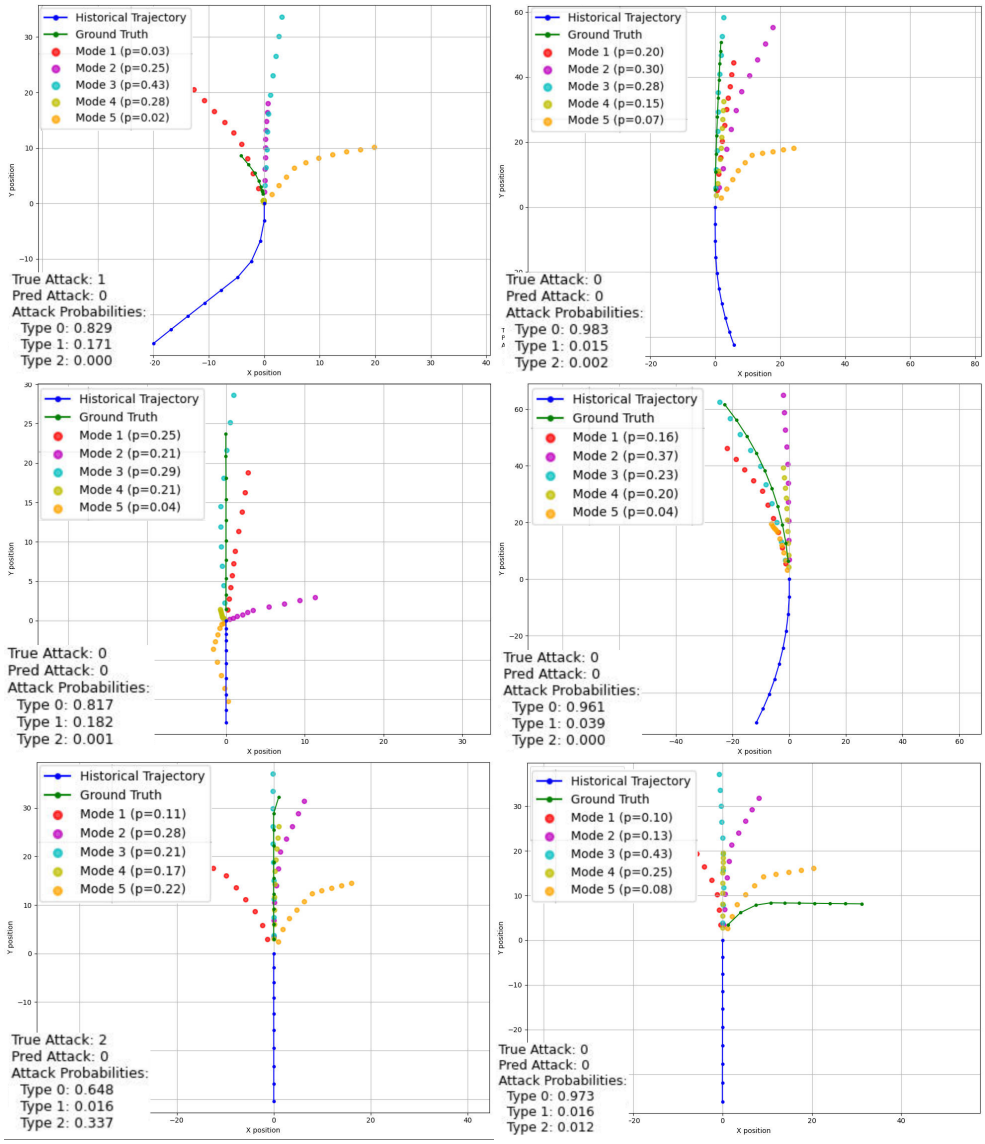


Figure 4.7: Final results using parameters in Table 4.5.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, I first introduced existing examples of automotive model AD in autonomous driving and analyzed potential works that could contribute to AD. I then proposed two approaches: a Kalman filter-based detector and a new detector based on path prediction.

5.1.1 Kalman Filter-Based Detector

The Kalman filter-based approach effectively incorporates vehicle dynamic information, continuously comparing expected vehicle states with observed measurements for real-time anomaly identification. We propose a novel framework for detecting attacks in vehicles with unknown systems, utilizing a learning-based model to predict and compare observations against expected behaviour.

Compared to traditional methods, our approach based on the UKF is capable of detecting DOS attacks from sensors and actuators without prior knowledge of system parameters. By exploiting UKF's capabilities in handling nonlinearity, our algorithm demonstrated significant advantages, outperforming traditional Kalman filters with substantially better results in both true positive and true negative alarm rates.

5.1.2 Trajectory-Based Detector

The trajectory-based detection approach is based on the premise that attackers aiming to cause maximum damage would ultimately seek to influence the vehicle's trajectory. This method evaluates vehicle states from a perspective that does not rely on vehicle dynamic models, potentially detecting sophisticated attacks that might bypass traditional systems.

For this detector, I employed a lane trajectory prediction approach based on the transformer architecture to predict the likelihood of vehicle cyberattacks. I utilized the LimSim platform, integrating vehicle trajectories, velocity, yaw angle, lane types, traffic signal types, and surrounding vehicles to predict both vehicle trajectories and the types and probabilities of network attacks. While ensuring acceptable training speeds, I achieved promising preliminary results.

5.2 Future Work

5.2.1 Kalman Filter-Based Detector

Enhancing the filtering design for vehicle incursion detection can be the primary focus of future research. By accurately simulating intricate dynamics, investigating robust particle filters may further improve anomaly identification and strengthen vehicle security. The presented framework can be extended to other types of attacks such as FDI.

Several specific directions for future research include:

- **Adaptive Threshold Optimization**: Developing adaptive thresholds that automatically adjust based on driving conditions, traffic patterns, and vehicle characteristics to reduce false alarms while maintaining high detection rates.
- **Sensor Fusion Approaches**: Integrating multiple sensor modalities (e.g., LIDAR, radar, cameras) to create a more robust detection system that can operate effectively even when certain sensors are compromised.
- **Resilient Control Strategies**: Designing control mechanisms to maintain safe vehicle operation even when attacks are detected, implementing graceful degradation protocols rather than complete system shutdowns.

- **Edge Computing Implementation**: Optimizing the algorithm for deployment on edge computing devices within vehicles to ensure real-time detection capabilities without relying on external communication networks that could themselves be vulnerable.
- **Adversarial Training**: Enhancing the detector's robustness by training it against sophisticated adversarial attacks that attempt to evade detection by mimicking normal vehicle behavior.
- **UKF Performance Evaluation**: Conducting comparative studies to specifically evaluate whether performance improvements observed are predominantly due to more accurate modelling by the machine learning system or the inherent advantages of the UKF. Future experiments could involve testing the UKF explicitly with a detailed dynamic vehicle model to isolate these effects.

5.2.2 Trajectory-Based Detector

Although this study uses program-generated vehicle trajectories to test model performance, these trajectories may not fully reflect the dynamic characteristics of real-world traffic scenarios. Future work could incorporate real-world traffic datasets (e.g., NGSIM[11]) to enhance the generalizability of the results.

There remains significant room for improvement in these results. Key areas for future development include:

- **Advanced Network Architectures**: Experiment with more sophisticated transformer variants, attention mechanisms, and hybrid models that combine the strengths of different neural network architectures to improve prediction accuracy.
- **Context-Aware Prediction**: Incorporating additional contextual information such as road infrastructure details, weather conditions, and traffic regulations to create more nuanced and accurate trajectory predictions.
- **Real-World Validation**: Transitioning from simulation to real-world testing by implementing the system on actual CAVs in controlled environments, gradually expanding to more complex traffic scenarios.
- **Attack Taxonomy Development**: Creating a comprehensive classification of potential cyberattacks based on their impact on trajectory, enabling more targeted detection and mitigation strategies for specific attack types.

- **Multi-Vehicle Coordination**: Extending the model to consider interactions between multiple vehicles in a connected environment, detecting coordinated attacks that might target multiple vehicles simultaneously.
- **Computational Efficiency**: Optimizing the model architecture and inference process to enable real-time operation on automotive-grade hardware with limited computational resources.
- **Transfer Learning Approaches**: Developing techniques to transfer knowledge from simulated environments to real-world scenarios, reducing the need for extensive real-world data collection while maintaining high detection accuracy.

5.2.3 Integrated Detection Framework

A promising direction for future research is the development of an integrated framework that combines both the Kalman filter-based and trajectory-based approaches:

- **Multi-Level Detection System**: Creating a hierarchical detection system that leverages the strengths of both approaches—using the Kalman filter for immediate, low-level anomaly detection and the trajectory predictor for higher-level behavioural analysis.
- **Confidence-Based Integration**: Developing algorithms that dynamically adjust the weighting between the two detection methods based on their confidence levels in different driving scenarios.
- **Explainable Artificial Intelligence (AI) Elements**: Incorporating explainable AI techniques to provide transparency into detection decisions, enabling system operators and passengers to understand the reasoning behind attack alerts.
- **Standardized Evaluation Framework**: Establishing comprehensive benchmarks and evaluation metrics that can assess the performance of different AD systems across various attack scenarios and driving conditions.

These future directions aim to address current limitations while advancing the field toward more robust, efficient, and practical automotive cybersecurity solutions capable of protecting increasingly CAVs from emerging cyber threats.

References

[1] Experimental security research of tesla autopilot. https://www.theguardian.com/technology/2016/sep/20/tesla-model-s-chinese-hack-remote-control-brakes, 2016. Accessed: 2025-01-29.

[2] Team of hackers take remote control of tesla model s from 12 miles away. https://www.technologyreview.com/2019/04/01/65915/hackers-trick-teslas-autopilot-into-veering-towards-oncoming-traffic/, 2016. Accessed: 2025-01-29.

[3] Omar Y Al-Jarrah, Carsten Maple, Mehrdad Dianati, David Oxtoby, and Alex Mouzakitis. Intrusion detection systems for intra-vehicle networks: A review. *Ieee Access*, 7:21266–21289, 2019.

[4] Theyazn HH Aldhyani and Hasan Alkahtani. Attacks to automatous vehicles: A deep learning algorithm for cybersecurity. *Sensors*, 22(1):360, 2022.

[5] Samira Tahajomi Banafshehvaragh and Amir Masoud Rahmani. Intrusion, anomaly, and attack detection in smart vehicles. *Microprocessors and Microsystems*, 96:104726, 2023.

[6] Ivo Berger, Roland Rieke, Maxim Kolomeets, Andrey Chechulin, and Igor Kotenko. Comparative study of machine learning methods for in-vehicle intrusion detection. In *International Workshop on Security and Privacy Requirements Engineering*, pages 85–101. Springer, 2018.

[7] Shuhao Bian, Milad Farsi, Nasser L. Azad, and Chris Hobbs. Data-driven intrusion detection in vehicles: Integrating unscented kalman filter (ukf) with machine learning.

In *Proceedings of the 21st International Conference on Informatics in Control, Automation and Robotics - Volume 1: ICINCO*, pages 714–723. INSTICC, SciTePress, 2024.

[8] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[9] Thanh Nguyen Canh and Xiem HoangVan. Machine learning-based malicious vehicle detection for security threats and attacks in vehicle ad-hoc network (vanet) communications. In *2023 RIVF International Conference on Computing and Communication Technologies (RIVF)*, pages 206–211. IEEE, 2023.

[10] AKB Chand and P Viswanathan. Cubic hermite and cubic spline fractal interpolation functions. In *AIP conference Proceedings*, volume 1479, pages 1467–1470. American Institute of Physics, 2012.

[11] Benjamin Coifman and Lizhe Li. A critical evaluation of the next generation simulation (ngsim) vehicle trajectory dataset. *Transportation Research Part B: Methodological*, 105:362–377, 2017.

[12] Cyber-Expert. The different cyber risks. https://blog.httpcs.com/en/autonomous-cars-cyber-risks/, 2024. Accessed: 2025-01-29.

[13] Seyed Mehran Dibaji, Mohammad Pirani, David Bezalel Flamholz, Anuradha M Annaswamy, Karl Henrik Johansson, and Aranya Chakrabortty. A systems and control perspective of cps security. *Annual reviews in control*, 47:394–411, 2019.

[14] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator, 2017.

[15] Dajun Du, Minggao Zhu, Xue Li, Minrui Fei, Siqi Bu, Lei Wu, and Kang Li. A review on cybersecurity analysis, attack detection, and attack defense methods in cyber-physical power systems. *Journal of Modern Power Systems and Clean Energy*, 11(3):727–743, 2022.

[16] Milad Farsi., Shuhao Bian., Nasser L. Azad., Xiaobing Shi., and Andrew Walenstein. An efficient resilient mpc scheme via constraint tightening against cyberattacks: Application to vehicle cruise control. In *Proceedings of the 20th International Conference on Informatics in Control, Automation and Robotics - Volume 1: ICINCO*, pages 674–682. INSTICC, SciTePress, 2023.

[17] Jiyang Gao, Chen Sun, Hang Zhao, Yi Shen, Dragomir Anguelov, Congcong Li, and Cordelia Schmid. Vectornet: Encoding hd maps and agent dynamics from vectorized representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11525–11533, 2020.

[18] Charles Hartsell, Shreyas Ramakrishna, Abhishek Dubey, Daniel Stojcsics, Nagabhushan Mahadevan, and Gabor Karsai. Resonate: A runtime risk assessment framework for autonomous systems. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 118–129. IEEE, 2021.

[19] Charles Hartsell, Shreyas Ramakrishna, Abhishek Dubey, Daniel Stojcsics, Nagabhushan Mahadevan, and Gabor Karsai. Resonate: A runtime risk assessment framework for autonomous systems. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 118–129, 2021.

[20] Yihan Hu, Jiazhi Yang, Li Chen, Keyu Li, Chonghao Sima, Xizhou Zhu, Siqi Chai, Senyao Du, Tianwei Lin, Wenhai Wang, et al. Planning-oriented autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 17853–17862, 2023.

[21] Abdulmalik Humayed, Jingqiang Lin, Fengjun Li, and Bo Luo. Cyber-physical systems security—a survey. *IEEE Internet of Things Journal*, 4(6):1802–1831, 2017.

[22] Zhiyang Ju, Hui Zhang, Xiang Li, Xiaoguang Chen, Jinpeng Han, and Manzhi Yang. A survey on attack detection and resilience for connected and automated vehicles: From vehicle dynamics and control perspective. *IEEE Transactions on Intelligent Vehicles*, 7(4):815–837, 2022.

[23] Zhiyang Ju, Hui Zhang, Xiang Li, Xiaoguang Chen, Jinpeng Han, and Manzhi Yang. A survey on attack detection and resilience for connected and automated vehicles: From vehicle dynamics and control perspective. *IEEE Transactions on Intelligent Vehicles*, 7(4):815–837, 2022.

[24] Masoud Khodarahmi and Vafa Maihami. A review on kalman filter models. *Archives of Computational Methods in Engineering*, 30(1):727–747, 2023.

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[26] Mojtaba Kordestani and Mehrdad Saif. Observer-based attack detection and mitigation for cyber-physical systems: A review. *IEEE Systems, Man, and Cybernetics Magazine*, 7(2):35–60, 2021.

[27] Roger Labbe. filterpy: Kalman filters and other optimal and non-optimal estimation filters in python. https://github.com/rlabbe/filterpy, 2024. Accessed: 20 June 2024.

[28] Edward A Lee. Cps foundations. In *Proceedings of the 47th design automation conference*, pages 737–742, 2010.

[29] Gaoqi Liang, Junhua Zhao, Fengji Luo, Steven R. Weller, and Zhao Yang Dong. A review of false data injection attacks against modern power systems. *IEEE Transactions on Smart Grid*, 8(4):1630–1638, 2017.

[30] Qipeng Liu, Yilin Mo, Xiaoyu Mo, Chen Lv, Ehsan Mihankhah, and Danwei Wang. Secure pose estimation for autonomous vehicles under cyber attacks. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 1583–1588. IEEE, 2019.

[31] Yifa Liu, Wenchao Xue, Shuping He, and Long Cheng. Stealthy false data injection attacks against extended kalman filter detection in power grids. In *2021 8th International Conference on Information, Cybernetics, and Computational Social Systems (ICCSS)*, pages 459–464, 2021.

[32] Pablo Marcillo, Diego Tamayo-Urgilés, Ángel Leonardo Valdivieso Caraguay, and Myriam Hernández-Álvarez. Security in v2i communications: a systematic literature review. *Sensors*, 22(23):9123, 2022.

[33] Sky McKinley and Megan Levine. Cubic spline interpolation. *College of the Redwoods*, 45(1):1049–1060, 1998.

[34] Jelena Mirkovic and Peter Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[35] Yilin Mo and Bruno Sinopoli. Secure control against replay attacks. In *2009 47th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 911–918, 2009.

[36] Carlos Murguia and Justin Ruths. Cusum and chi-squared attack detection of compromised sensors. In *2016 IEEE Conference on Control Applications (CCA)*, pages 474–480. IEEE, 2016.

[37] Ahmed S. Musleh, Guo Chen, and Zhao Yang Dong. A survey on the detection algorithms for false data injection attacks in smart grids. *IEEE Transactions on Smart Grid*, 11(3):2218–2234, 2020.

[38] Ila Naqvi, Alka Chaudhary, and Anil Kumar. A systematic review of the intrusion detection techniques in vanets. *TEM Journal*, 11(2):900, 2022.

[39] Fabio Pasqualetti, Florian Dörfler, and Francesco Bullo. Attack detection and identification in cyber-physical systems. *IEEE transactions on automatic control*, 58(11):2715–2729, 2013.

[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[41] Yashaswini S Pawar, Prasad Honnavalli, and Sivaraman Eswaran. Cyber attack detection on self-driving cars using machine learning techniques. In *2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT)*, pages 1–5. IEEE, 2022.

[42] Sampath Rajapaksha, Harsha Kalutarage, M Omar Al-Kadri, Andrei Petrovski, Garikayi Madzudzo, and Madeline Cheah. Ai-based intrusion detection systems for in-vehicle networks: A survey. *ACM Computing Surveys*, 55(11):1–40, 2023.

[43] Muhammad Rashed, Joarder Kamruzzaman, Iqbal Gondal, and Syed Islam. False data detection in a clustered smart grid using unscented kalman filter. *IEEE Access*, 10:78548–78556, 2022.

[44] Cian Ryan, Finbarr Murphy, and Martin Mullins. End-to-end autonomous driving risk analysis: A behavioural anomaly detection approach. *IEEE Transactions on Intelligent Transportation Systems*, 22(3):1650–1662, 2020.

[45] Helem Sabina Sánchez, Damiano Rotondo, Teresa Escobet, Vicenc Puig, Jordi Saludes, and Joseba Quevedo. Detection of replay attacks in cyber-physical systems using a frequency-based signature. *Journal of the Franklin Institute*, 356(5):2798–2824, 2019.

[46] Taku Takahama and Daisuke Akasaka. Model predictive control approach to design practical adaptive cruise control for traffic jam. *International Journal of Automotive Engineering*, 9(3):99–104, 2018.

[47] Laurens Van Eykeren, QP Chu, and JA Mulder. Sensor fault detection and isolation using adaptive extended kalman filter. *IFAC Proceedings Volumes*, 45(20):1155–1160, 2012.

[48] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for signal processing, communications, and Control Symposium (Cat. No. 00EX373)*, pages 153–158. Ieee, 2000.

[49] Chuan Wang, Long Chen, Lei Li, Yong-Hong Yan, Juan Sun, Chao Yu, Xin Deng, Chun-Ping Liang, Xue-Liang Zhang, and Wei-Ming Peng. Combining unscented kalman filter and wavelet neural network for anti-slug. *Petroleum Science*, 20(6):3752–3765, 2023.

[50] G Welch. An introduction to the kalman filter. 1995.

[51] Licheng Wen. Long-term interactive multi-scenario simulation, 2023.

[52] Licheng Wen, Daocheng Fu, Song Mao, Pinlong Cai, Min Dou, Yikang Li, and Yu Qiao. Limsim: A long-term interactive multi-scenario traffic simulator, 2023.

[53] Moritz Werling, Julius Ziegler, Sören Kammel, and Sebastian Thrun. Optimal trajectory generation for dynamic street scenarios in a frenét frame. In *2010 IEEE International Conference on Robotics and Automation*, pages 987–993, 2010.

[54] Jiaxuan Xu, Qi Luo, Kecheng Xu, Xiangquan Xiao, Siyang Yu, Jiangtao Hu, Jinghao Miao, and Jingao Wang. An automated learning-based procedure for large-scale vehicle dynamics modeling on baidu apollo platform. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5049–5056. IEEE, 2019.

[55] Dong Zhang, Chen Lv, Tianci Yang, and Peng Hang. Cyber-attack detection for autonomous driving using vehicle dynamic state estimation. *Automotive Innovation*, 4:262–273, 2021.

[56] Nemanja Živković and Andrija T Sarić. Detection of false data injection attacks using unscented kalman filter. *Journal of Modern Power Systems and Clean Energy*, 6(5):847–859, 2018.

Glossary

autopilot A system used in vehicles, aircraft, and ships that automatically controls the trajectory and operations of the vehicle without continuous manual input from the operator, often used in autonomous and semi-autonomous systems. 1

CARLA CARLA is an open-source simulator for autonomous driving research, allowing developers to train and validate self-driving algorithms in realistic urban environments. See https://carla.org/ for details. 25, 27

ECU Engine control unit, An engine control unit (ECU), also called an engine control module (ECM), is a device that controls various subsystems of an internal combustion engine. Systems commonly controlled by an ECU include the fuel injection and ignition systems. 5

ENISA European Union Agency for Cybersecurity, a center of expertise in cybersecurity within the EU, supports member states, EU institutions, and private stakeholders to improve overall cybersecurity across Europe. 1

NGSIM Next Generation Simulation, a project initiated by the Federal Highway Administration (FHWA) that provides high-fidelity traffic simulation data for research in traffic flow, transportation safety, and intelligent transportation systems. 58

NHTSA National Highway Traffic Safety Administration, a U.S. government agency responsible for ensuring road safety by setting and enforcing vehicle performance standards, conducting safety research, and overseeing vehicle recalls. 1

real-time In control theory, real-time refers to a system's ability to process inputs and produce outputs within strict time constraints, ensuring timely and deterministic responses. 1, 2, 6, 7, 10, 13, 14, 25