UNIVERSITY OF CALIFORNIA,
IRVINE

# Play the Imitation Game: Model Extraction Attack against Autonomous Driving Localization

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Electrical and Computer Engineering

by

Qifan Zhang

2022

# TABLE OF CONTENTS

# LIST OF FIGURES

Page

# LIST OF TABLES

# ACKNOWLEDGMENTS

I would like to thank my committee chair, also my advisor, Assistant Professor Zhou Li, for his gracious and constant support for my research.

I would also like to thank my committee members, Associate Professor Mohammad Abdullah Al Faruque, and Assistant Professor Qi Alfred Chen, for providing me collaboration and feedback. Without their help, I can not complete this thesis.

I would also like to thank my collaborators of this work, Junjie Shen, Mingtian Tan, Professor Zhe Zhou from Fudan University, and Professor Haipeng Zhang from ShanghaiTech University, for their collaboration and supervision in machine-learning-based model extraction attack against autonomous driving localization, and for their intellectual support with my research.

# ABSTRACT OF THE THESIS

Play the Imitation Game: Model Extraction Attack against Autonomous Driving Localization

By

Qifan Zhang

Master of Science in Electrical and Computer Engineering

University of California, Irvine, 2022

Assistant Professor Zhou Li, Chair

The security of the Autonomous Driving (AD) system has been gaining researchers' and public's attention recently. Given that AD companies have invested a huge amount of resources in developing their AD models, e.g., localization models, these models, especially their parameters, are important intellectual property and deserve strong protection.

In this work, we examine whether the confidentiality of production-grade Multi-Sensor Fusion (MSF) models, in particular, Error-State Kalman Filter (ESKF), can be stolen from an outside adversary. We propose a new model extraction attack called TaskMaster that can infer the secret ESKF parameters under black-box assumption. In essence, TaskMaster trains a substitutional ESKF model to recover the parameters, by observing the input and output to the targeted AD system. To precisely recover the parameters, we combine a set of techniques, like gradient-based optimization, search-space reduction and multi-stage optimization. The evaluation result on real-world vehicle sensor dataset shows that TaskMaster is practical. For example, with 25 seconds AD sensor data for training, the substitutional ESKF model reaches centimeter-level accuracy, comparing with the ground-truth model.

# Chapter 1

# Introduction

In the recent decades, the advancement of technologies in machine learning, sensing, and control has elevated autonomous vehicles (AV) from ideation to reality. A growing number of AV companies have emerged and some have pushed their products to public roads. For instance, Google and Baidu have been operating self-driving taxis [24, 82] for years. Among all the components inside AV, the Autonomous Driving (AD) system is the most important piece, acting as the AV's "brain". The AD system commands the actuators according to the prediction of perception models.

One key component in the pipeline of AD is *localization*, which computes the real-time vehicle position. Ensuring the accuracy of localization is fundamental to the safety of AV, for which most of the AV companies use a complex Multi-Sensor Fusion (MSF) model [106, 93] to fuse the readings of multiple sensors. Essentially, MSF takes input from sensors like GPS, IMU and LiDAR, and runs a state estimation model, *e.g.*, Kalman Filter (KF), to predict AV's state, including position, heading direction, velocity, etc. As a result, the prediction made by MSF is highly robust, even in bad weather conditions or when one sensor is under attack, like GPS spoofing [34]. Yet, Shen et al. [90] showed that the *integrity* of a MSF model can be violated, by demonstrating a successful attack on the production-grade AD system, *i.e.*, Baidu Apollo [8]. In the meantime, the *confidentiality* of a MSF has not been discussed, not to mention the demonstration of attacks and defense. Considering the importance of MSF, we study the MSF confidentiality issues in this thesis.

**Confidentiality of MSF models.** By examining the production-grade MSF implementation, *e.g.*, the one from Baidu Apollo, we found confidentiality is indeed a great concern. Although Apollo is an open-sourced project, the source code of MSF module is not released and cannot be decompiled into a readable format. In fact, based on our discussion with industrial partners, the parameters of MSF are considered as the project's top intellectual property, since they devoted years of hard work to tune the parameters and localization became the deciding factor for their product to outperform their competitors.

On the other hand, previous works in ML security has demonstrated *model extraction attacks* [101, 50, 19], which queries a blackbox ML models on a remote server (e.g., public cloud), can infer the secret parameters of ML models. Given that the input to and the output from a MSF model can be observed, a natural idea is to borrow such model extraction technique to attack MSF. Yet, a few challenges prevent the direct application of the existing model extraction attacks, including the physical-world constraints to attacker's observations, the complexity of MSF models, and its interaction with other controller components (detailed in Section 3.3).

**Our attack.** To tackle these challenges in extracting MSF models, we leverage the two main insights of the AD system: 1) Though MSF models differ from ML models, their parameters can be approximated through gradient-based optimization, as their equations are *derivable*. 2) When the input or output is inaccessible, in particular, when the MSF's output is only sent to AD controllers and the channel between them might not be interceptable, we can *emulate* derivable AD controllers and use their output for optimization. In fact, the emulated

AD controllers do not need to be the exact same implementations, and the only requirement is that their performance is comparable to the ones in the target AD system.

Based on the above insights, we propose TaskMaster[1], a new model extraction attack against MSF models, with a set of techniques, like training unrolling, search-space reduction, and multi-stage optimization. Though our approach can be classified as *system identification (SI)* [64], we found none of the existing approaches directly work in our setting due to the complexity of MSF in AV and the constraints on attacker's data access. We examine TaskMaster under three attack settings (intrusive in-AV attacker, non-intrusive in-AV attacker, and AV follower), and evaluate it on the real-world vehicle sensor traces (the KAIST Complex Urban sensor traces [52]). The evaluation shows that model extraction attack is a practical threat. As a highlight of our findings, by collecting data points within *25-second window* of a targeted AV, we can train an ESKF model (a variation of KF model used by AV companies) reaching *centimeter*-level accuracy to the ground-truth model. Starting from the extracted model, the cost of the adversary (e.g., unethical competitors) can be greatly reduced.

**Contributions.** We summarize the contributions of this work as follows:

- We present the *first* study about the confidentiality of the AD localization models.
- To address the new challenges posed by the unique structure of MSF, we develop a new model extraction approach, TaskMaster, comprising optimization techniques tailored to the control-theory models.
- We examine TaskMaster under three attack settings with the real-world sensor traces, and our result indicates model extraction attack is feasible and could benefit an unethical competitors.

[1]TaskMaster is a fictional character in Marvel Comics who can mimic any fighting style of a superhero.

- The implementations of TaskMaster will be open-sourced (also attached in the submission).

**Ethics and disclosure.** We disclosed our findings to developers of the Baidu Apollo team and are in discussion about the attack impact and potential fix.

# Chapter 2

# Background

In this section, we first overview the architecture of Autonomous Driving (AD) system of an Autonomous Vehicle (AV), focusing on the localization module and its design based on Multi-Sensor Fusion (MSF). Then, we describe the most popular MSF algorithm that is based on Kalman Filter. Finally, we introduce another AD component that is investigated in this work, AD controller.

## 2.1 AD Localization

Generally, an AD system is composed of 3 main modules: sensor/information collector, on-board computer and actuator/command executor. Their connections and sub-modules are illustrated in Figure 2.1, which is abstracted from AD systems like Baidu Apollo [8] and Autoware [59]. Specifically, the sensor/information collector takes input from sensors like GNSS (Global Navigation Satellite System) receiver, LiDAR, IMU (Inertial Measurement Unit), camera and communication devices like LTE/5G Antenna. The data is sent to the on-board computer to infer a model of the world. Based on the destination and route planning,

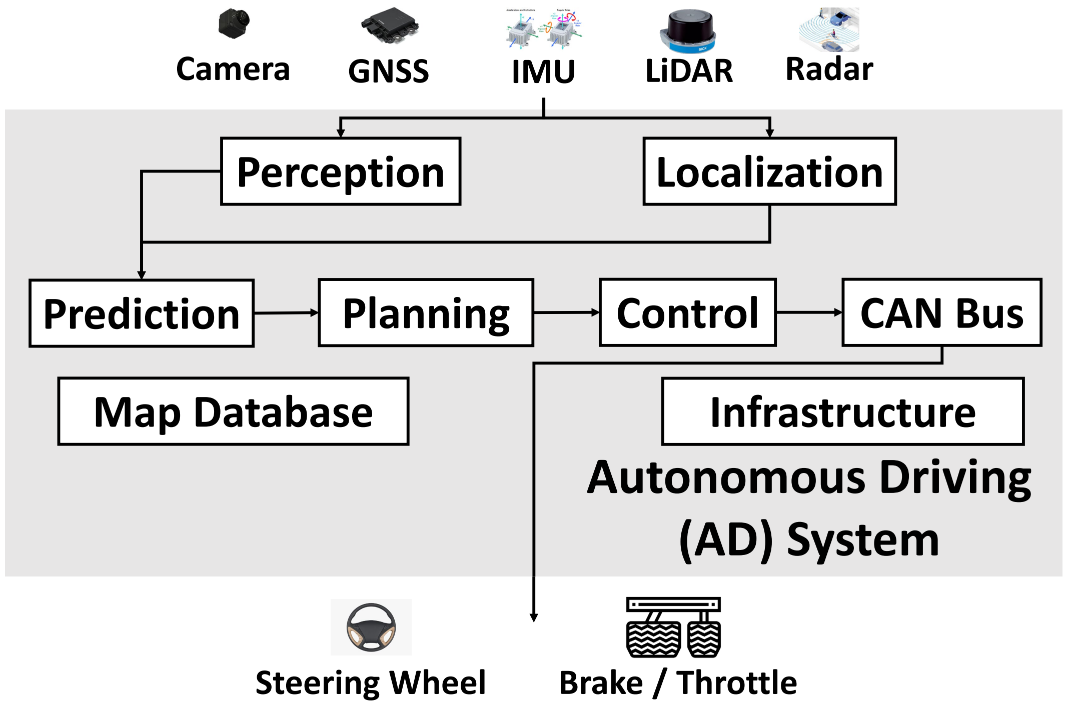

**Figure 2.1:** AD Architecture Example.

the controller inside the computer generates the control vector to direct the vehicle, including three parameters: steering control, throttle control and brake control. These 3 controlling parameters will be fed to the actuator/command executor to change the physical position of the vehicle and also affect the running state of the AD system.

In this work, we investigate the *localization* (or state estimation) component, which computes the real-time ego-vehicle position on the map. Localization is critical in ensuring the driving safety and correctness, requiring *centimeter-level* accuracy [85], robustness under the severe weather and road condition, and high-fidelity under cyber-attacks. A trivial solution for localization is to directly use the input from GNSS. However, GNSS signal significantly degrades due to atmosphere delays and multi-path effect [41]. Moreover, civilian GNSS lacks signal authentication and is vulnerable under spoofing attack [99], in which the attacker can override the authentic signal with stronger power. Using LiDAR, which measures the reflection of laser light, individually is also fragile for localization, especially under poor weather conditions like rain [34]. Hence, localization based on *Multi-Sensor Fusion (MSF)*, which fuses the input from multiple sensors like GPS, IMU and LiDAR, has become the optimal solution so far, as it delivers much more accurate and robust result, by addressing the weakness of individual sensors [106, 93].

### 2.1.1 MSF algorithms

Among the existing MSF algorithms, Kalman Filter (KF)-based MSF [67] has gained much broader adoption, compared to the others (e.g., Particle Filter [42]). According to the survey by Shen et al. [90], out of the 18 top-tier robotics papers for 2018-2019, 14 papers adopted KF-based MSF. Baidu Apollo [8], an open-source AD system that has gained prominent buy-in from AD industry [18] (e.g. being deployed in the self-driving taxi services in China [82]), also chooses KF-based MSF [106].

We focus on the confidentiality of KF-based MSF model, and the ESKF (Error-State Kalman Filter) model used by AV (e.g., Baidu Apollo ESKF) is our primary target, mainly because it reaches the highest localization accuracy among all papers surveyed by [90], and its implementation has been considered as a secret (detailed in Section 3.1). It is worth mentioning that our approach can be generalized to other KF-based MSF models, e.g., Extended Kalman filter (EKF).

## 2.2 Kalman Filter based Multi-Sensor Fusion

### 2.2.1 Additive White Guassian Noise and Kalman Filter

Additive White Guassian Noise (AWGN) is a basic and common noise model to imitate the random process in the nature. It follows normal distribution with zero mean. Each element in AWGN is independent of other elements. Therefore, suppose there is an AWGN $w \in \mathcal{R}^n$, its distribution is:

$$w \sim \mathcal{N}(0, R) \tag{2.1}$$

Where $R$ is a $n \times n$ diagonal matrix.

Kalman Filter (KF) [63], also known as linear quadratic estimation (LQE), uses prior state measurements to produce estimates of the posterior states. Equation 2.2 and 2.3 show the how a state in time $k$ is estimated from a state in time $k-1$. The advantage of KF is that it solves LQE with recursive methods. For each estimation, KF could make the prediction only with its previous state and its related state estimation. Since traditional KF adapts linear optimization methods with small amounts of computation, which is suitable for computers, KF is widely used in Multi-Sensor Fusion (MSF) algorithms.

$$\begin{aligned} x_k &= F x_{k-1} + B u_{k-1} \\ P_k &= F P_{k-1} F^\top + Q \end{aligned} \tag{2.2}$$

$$\begin{aligned} K' &= P_k H_k^\top (H_k P_k H_k^\top + R)^{-1} \\ \hat{x}_k &= x_k + K'(z_k - H_k x_k) \\ \hat{P}_k &= P_k - K' H_k P_k \end{aligned} \tag{2.3}$$

KF is a kind of Bayesian filter. It is also an algorithm to make accurate predictions on a series of uncertain states using time-series observations with noises. KF makes two basic assumptions:

- The whole system is a linear dynamic system.
- Each observation contains a noise, and that noise is an AGWN.

With the two assumptions held, KF optimizes observed states by minimizing Mean Square Error (MSE).

In particular, KF iteratively executes two phases: Prediction and Update. For Prediction (Equation 2.2), $x_k$ (the predicted state at $k$) and $P_k$ (the predicted covariance matrix measuring the confidence of $x_k$) are computed based on $x_{k-1}$, $P_{k-1}$ and $u_{k-1}$ (the measurement of kinetics). For Update (Equation 2.3), the observations of the real-world environment (e.g., through sensors), denoted as $z_k$, are used to refine $x_k$ and $P_k$ to $\hat{x}_k$ and $\hat{P}_k$, in order to reduce prediction errors. $H_k$ is used to map the true state space (where $x_k$ resides) into the observed space (where $z_k$ resides). $Q$ and $R$ are the covariance matrix of the process noise and the covariance matrix of the observation noise. $F$ and $B$ represent the state-transition model and the control-input model.

During execution, KF iterates its two phases (Prediction (Equation 2.2) and Update (Equation 2.3). This property ensures KF to make accurate estimations for a series of observations. Therefore, KF could make predictions without interfering sensors for observations, such as GNSS and LiDAR. KF itself will make predictions on the current state and its uncertainty, based on its previous state and its related uncertainty matrix, combined with current observations.

In update part, KF will minimize the error state, i.e., minimizing $E[(x_k - \hat{x_k})^2]$. In practice, KF reaches this goal by minimizing its state estimation and the trace of related covariance matrix. This is because the trace of a covariance matrix is the summation of the variances of its related state estimation. With the reduction of its trace, total estimation error will be minimized. Therefore, the whole state will be more accurate by minimizing the trace of covariance matrices.

### 2.2.2 Error-State Kalman Filter in AD system

We use the MSF implemented by Baidu Apollo to demonstrate how KF is applied for AD. Specifically, Baidu Apollo fuses the readings from IMU, LiDAR and GNSS with Error-State Kalman Filter (ESKF), a variant of KF [106]. IMU measures the acceleration ($accel_{k-1}$) and angular velocity ($omega_{k-1}$) of the AV, which are used to construct the control vector $u_{k-1} = (accel_{k-1}, omega_{k-1})^\intercal$, for Prediction phase. The predicted state $x_k$ is a vector consisting of 16 values. It is represented as $(pos_k, vel_k, quat_k, ba_k, bg_k)^\intercal$, where $pos_k$ represents the AV's current location (3 elements), $quat_k$ represents the heading direction in form of quaternion (4 elements), $vel_k$ represents velocity (3 elements), $ba_k$ represents accelerometer bias (3 elements), and $bg_k$ represents gyrometer bias (3 elements). $P_k$ is a $15 \times 15$ matrix. For Update phase, the position measurements from GNSS and the position measurements and car heading measurements from LiDAR are considered as the observations $z_k$ after data processing. When Update phase is finished, an Error-state Reset phase ($\hat{P}_k = G\hat{P}_k G^\intercal$) is introduced by ESKF to reset $\hat{P}_k$, to address the issue of observation drifting [106]. $G$ is defined in Equation 2.4.

$$G = \begin{pmatrix} I_6 & 0 & 0 \\ 0 & I_3 - [\frac{1}{2}\hat{\delta\theta}] & 0 \\ 0 & 0 & I_9 \end{pmatrix} \tag{2.4}$$

Where $I_n$ represents an $n \times n$ identity matrix and $\hat{\delta\theta}$ represents the error state of the AD car heading (in euler angles).

In addition, Baidu Apollo uses different $R$ and $H$ for GNSS and LiDAR. For GNSS, we assume $R_G$ (noise distribution injected into GNSS data) and $H_G$ are used to replace $R$ and $H$ in Equation 2.3, changing the Update phase to Equation 2.5. For LiDAR, the Update phase is separated into two sub-phases that uses the position sensing data and pose (or yaw of the AD car) sensing data separately. The output of the first sub-phase is fed to the second sub-phase. The Update phase is changed to Equation 2.6. To notice, Prediction phase happens whenever IMU sends new input, and Update phase happens whenever LiDAR or GNSS sends new input. Hence, Prediction and Update do not necessarily happen in turns. Figure 2.2 illustrates the workflow of ESKF.

$$
\begin{aligned}
K' &= P_k H_G^\top (H_G P_k H_G^\top + R_G)^{-1} \\
\hat{x_k} &= x_k + K'(z_k - H_G x_k) \\
\hat{P}_k &= P_k - K' H_G P_k
\end{aligned}
\tag{2.5}
$$

$$
\begin{aligned}
K' &= P_k (H_L^p)^\top (H_L^p P_k (H_L^p)^\top + R_L^p)^{-1} \\
x_k' &= x_k + K'(z_k^p - H_L^p x_k) \\
P_k' &= P_k - K' H_L^p P_k \\
K'' &= P_k (H_L^y)^\top (H_L^y P_k (H_L^y)^\top + R_L^y)^{-1} \\
\hat{x_k} &= x_k' + K''(z_k^y - H_L^y x_k) \\
\hat{P}_k &= P_k' - K'' H_L^y P_k
\end{aligned}
\tag{2.6}
$$

Where $R_L^p$,$H_L^p$,$z_k^p$ and $R_L^y$,$H_L^y$,$z_k^y$ are related to position and yaw observations separately.

## 2.3 AD Controller

After localization, the estimated vehicle status is combined with the output of path planning, collision avoidance, etc. in the navigation stack, and sent to the controller module to guide actuators/command executors. We briefly overview this module here, since it has to be emulated for the attack scenarios when the output to ESKF is not directly observable to the adversary (see Section 3.2).

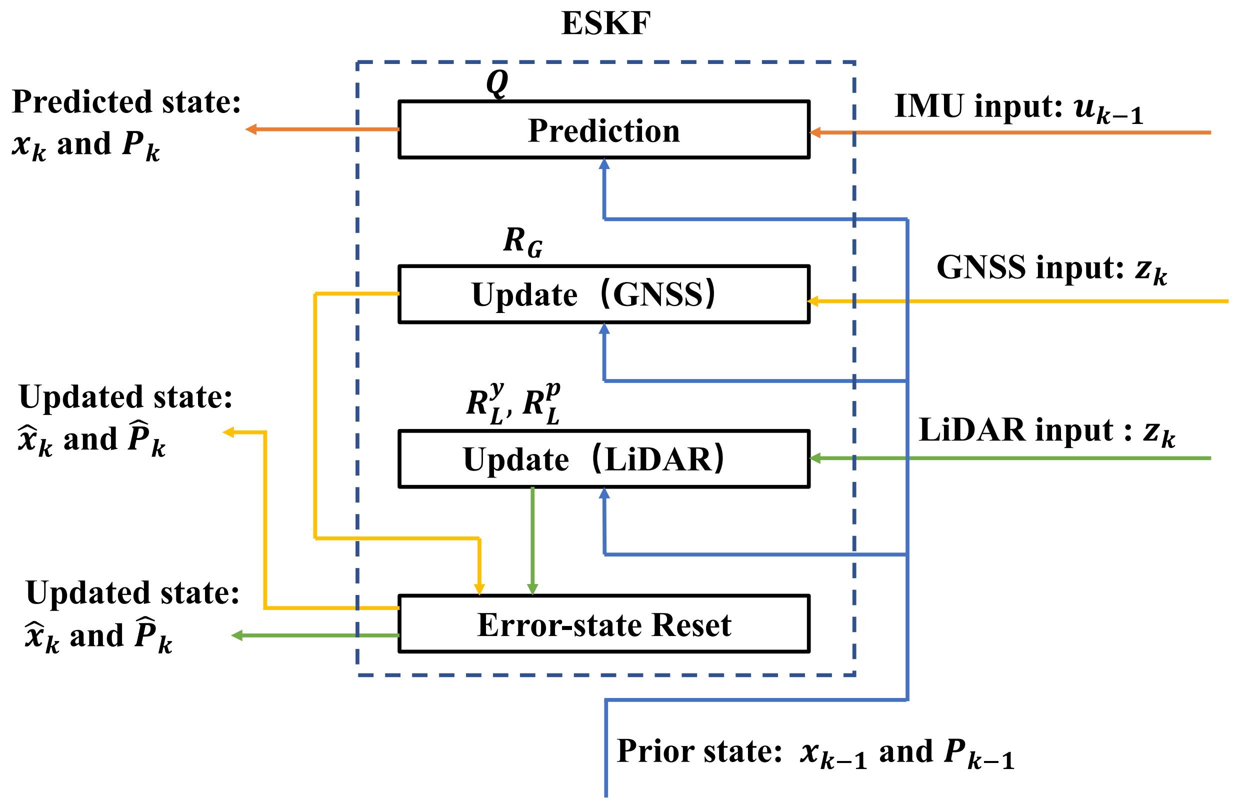
ESKF
Predicted state: $x_k$ and $P_k$
$Q$
Prediction
IMU input: $u_{k-1}$
$R_G$
Update（GNSS）
GNSS input: $z_k$
Updated state: $\hat{x}_k$ and $\hat{P}_k$
$R_L^y$, $R_L^p$
Update（LiDAR）
LiDAR input : $z_k$
Updated state: $\hat{x}_k$ and $\hat{P}_k$
Error-state Reset
Prior state: $x_{k-1}$ and $P_{k-1}$

**Figure 2.2:** Workflow of ESKF. The flows of GNSS and LiDAR both generate Update states, but the values could be different.

An AD controller is normally divided into 2 sub-components: lateral controller and longitudinal controller. The lateral controller makes decisions on the angular velocity change (i.e. steering) while the longitudinal controller makes decisions on the acceleration (i.e. throttle and brake). Both the lateral controller and the longitudinal controller share the same input data, and their output, including steering, throttle and brake, forms a complete control vector.

The AD controller solves the optimal control problem in a dynamic system. As such, it uses the existing classic control algorithms. For Baidu Apollo [9], the LQR (linear-quadratic regulator) algorithm [31] is used for lateral control while the PID (proportional-integral-derivative) algorithm [88] is used for longitudinal control. LQR controller uses a cost function defined by human, in the form of a sum of the deviations of key measurements, and finds the controller settings that minimizes the cost. PID controller calculates proportional, integral, and derivative responses after reading the sensors, and sums them to compute the actuator output. LQR controller produces better response compared to PID controller, as it aims to achieve optimal control states, though at the cost of higher complexity.

# Chapter 3

# Attack Overview

In this section, we first describe the motivation of our adversary. Then, we describe the three scenarios that the attack could happen, differentiated by attackers' capabilities. Finally, we overview the workflow of our attack, termed TaskMaster.

## 3.1 Adversary Motivation

Though the procedure and equation of KF (including ESKF) are known and it is expected that every AD system follows them in implementation, the parameters of KF can be varied among AD systems, resulting in different localization performance. According to [106], the production-grade implementation by Baidu Apollo achieves 0.054 meters accuracy, which outperforms the academic KF implementations by a large margin (1.17 meters for JS-MSF [92] and 1.91 meters for ETH-MSF [28]). A lot of driving data from a human driver needs to be collected and different tuning approaches have to be experimented by professional AV engineers [90]. In fact, we reached out to one author of Baidu Apollo ESKF [106], and learnt that it takes more than *6 months* for a specialized team to tune ESKF. As such, the parameters of KF are considered as “intellectual property”, and kept as secret by the AV companies (e.g., Baidu’s leadership decides to keep the current and future versions of ESKF close-source, as we learnt from the author of [106]).

### 3.1.1 Existing protections

Specifically, the GitHub repo [8] of Baidu Apollo embeds ESKF in a binary file `liblocalization_msf.so`, though other parts have source code. We have attempted to reverse engineer this binary file for 5 weeks but were unable to extract its ESKF parameters. First, we try to decompile this binary, and found SIMD vectorization [60] is heavily used, which makes the decompiled code (including the pre- and post-processing) less readable. We have used reverse-engineer tools including IDA Pro [37], Snowman [110], and McSema [100] (an LLVM IR lifting tool), and all failed. Though McSema claims that AVX instructions can be handled, SIMD Instructions still cannot be decompiled. Second, we also tried binary analysis tools like Intel Pin [49] and Frida [30] to recover the secret values at the runtime, but were also unsuccessful. Regarding the information learnt from the 5 weeks’ efforts, we roughly know the execution steps of ESKF, such as IMU prediction, measurement update, outlier detection, after reading the disassembled code. We also discovered that the ESKF prediction and updates occurred asynchronously with multi-threading, which increases the difficulty for reverse engineering.

### 3.1.2 Generalizability of the extracted model

This work focuses on the confidentiality of four covariance matrices $Q$, $R_G$, $R_L^p$ and $R_L^y$ (process noises, GNSS noises, position observation noises and yaw observation noises) of an ESKF model, which are explained in Section 2.2. $R$ (including $R_G$, $R_L^p$ and $R_L^y$) in particular depends on the sensors (e.g., GNSS and LiDAR). For two AVs, if their sensors are similar, their $R$ can be similar. As a supporting evidence, we collected a trace (*local08*) from KAIST Complex Urban [52] (described in Section 5.1), which contains the sensor input and localization results when the tested AV is driven by a human. We consider its localization results as the ground truth, and compare to the localization results generated by Baidu Apollo's ESKF (using the sensor input from *local08*), in order to assess how well the ESKF can adapt to different vehicles (the default vehicle supported by Baidu Apollo is different from the KAIST vehicle) with similar sensors (e.g., their LiDAR sensors are the same). We found that the root mean squared error (RMSE) between the ground truth and Apollo's output is only 0.074m, reaching cm-level error, suggesting the industry-grade MSF has good adaptability.

When the sensors are quite different, directly using the stolen $R$ parameters for attacker's ESKF model might yield sub-optimal result. We consider the re-tuning of $R$ in certain circumstances as limitation (also described in Section 6), but we expect heavy re-tuning is unnecessary in most case. In fact, a few sensor providers are very popular among the car manufacturers. For instance, (1) the default LiDAR sensors supported by Baidu Apollo were manufactured by Velodyne [105], which were also integrated by AVs from Google/Alphabet [3], Ford [86, 87, 103], Toyota [47], Mercedes-Benz [15], Hyundai Mobis [48], ThroDrive [104], etc; (2) most car manufacturers purchase GNSS/IMU sensors from Novatel [38].

Aside from $R$ and $Q$, the other matrices, including $F$, $B$, $H_G$, $H_L^p$ and $H_L^y$, are determined by the vehicle kinematics and sensor measurement models, which can be obtained from textbook or tutorials [92]. Since sensors do not have lots of measurement variations – they typically measure vehicle positions in global coordinate systems such as longitude/latitude/altitude, these matrices usually will not change for different sensors.

### 3.1.3 Hacking AV to extract KF parameters

Hacking into the AD system of the targeted AV and then stealing the KF model is another unethical approach for the same attacker's goal. A few recent works demonstrated it is feasible to exploit the vulnerabilities in Wi-Fi modules of Tesla, send messages through CAN (Controller Area Network) bus, and take control of the AD system remotely, e.g., by opening a Linux shell [76, 97]. However, due to the high investment into AV security by AV companies (e.g., Tesla puts US$1 million for bug bounty [22]), such vulnerabilities are very rare, and can be quickly patched. Moreover, even if the shell is obtained by an adversary who is interested in KF parameters, the files containing KF models are very likely to be protected (e.g., the AVs using Baidu Apollo), and the KF parameters stored in memory or CPU registers are very difficult to be inferred.

## 3.2 Adversary Model

We assume the adversary wants to steal the KF model of a competitor's AV and integrate it into her own AV products, to save the hard work for KF tuning. The vehicles with the similar kinematics or sensor installment are expected to be compatible to such KF models.

Instead of assuming "whitebox" access and directly extracting the model parameters (e.g., by reading the files/memory/registers containing the KF parameters), our adversary has *"blackbox"* access to a KF model, meaning she can observe the input and output, and use the information to *infer* the KF parameters of victim's AV[1]. As such, defending against our attack, or TaskMaster, is significantly more challenging as no software/hardware vulnerability is exploited. We assume 3 attack scenarios based on adversary's capabilities, which are also summarized in Figure 3.1.

[1]Similar as blackbox model extraction attacks against DNN [101], the attacker needs to know the structure of KF ahead.

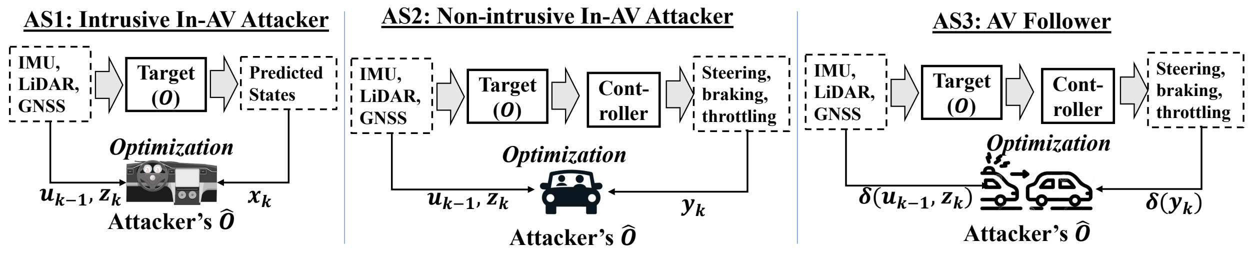

**Figure 3.1:** Adversary model. $\delta$ is the function adding noises.

### 3.2.1 AS1: Intrusive In-AV Attacker

We assume the attacker has exclusive physical access to the targeted AV, e.g., by purchasing, renting or borrowing the AV, and the attacker is able to sniff the data transmitted *within* the AD system, by inserting the sniffers directly onto the paths between ECUs (Electronic Control Units). As such, the attacker is able to observe the input to ESKF ($u_{k-1}$ and $z_k$), and the output from ESKF ($\hat{x}_k$). With such information, the attacker attempts to extract a victim AV's ESKF model.

### 3.2.2 AS2: Non-intrusive In-AV Attacker

We assume the attacker cannot sniff the data within the AD system, but she can plug in a CAN transceiver (e.g., TI VP232 CAN transceiver [98]) onto the AV's CAN bus, let the AV drive through a planned path, and use the transceiver to read the messages (which are unencrypted by CAN standard), which include the readings of sensors (IMU, LiDAR, and GNSS ) [43]. Alternatively, the attacker can bring her own sensors. For example, LiBackpack [36] integrate LiDAR and GNSS at the backpack size, and mobile devices usually have IMU sensors [95]. On the other hand, the attacker cannot observe the data between ESKF and controller, therefore she has no direct visibility into $\hat{x}_k$. The output of the controller, termed $y_k$, including steering, throttling, and braking, can be observed, by sniffing the command issued to those actuators. To notice, we assume the attacker does not know which controller is used by the AV or how it is designed, and we do not consider controller parameters as a secret.

### 3.2.3 AS3: AV Follower

This scenario has the most stringent attack condition that the attacker has to be outside of the AV. On the other hand, the attacker is able to drive another car and follow the AV in close vicinity. With high-resolution sensors on her AV, including GNSS, LiDAR and camera, the attacker collects the motion traces of the victim AV, and infers the sensor readings of the victim AV ($u_{k-1}$ and $z_k$) and the controller output ($y_k$), but the readings are inaccurate. We model the input to victim's KF as the combination of the sensor readings of attacker's AV and attacker's measurement noises. Shen et al. [90] adopts a similar approach to model the inaccurate attacker's readings when launching GPS spoofing against another AV on the move.

## 3.3 KF Model Extraction

At the high level, extracting KF model resembles *extracting machine-learning (ML) models*, of which the related works are surveyed in Section 8. In essence, model extraction against ML models also assumes blackbox access, though which the attacker uses the prediction APIs provided by the deployed model $O : \mathcal{X} \rightarrow \mathcal{Y}$ to issue queries (e.g., requesting classification of images) $X \subset \mathcal{X}$, and obtains the responses, including the labels $Y \subset \mathcal{Y}$, and optionally confidence scores $S_Y$ or logits $L_Y$. With $X$, $Y$ (together with $S_Y$ or $L_Y$ if available), the attacker runs an extraction algorithm $\mathcal{A}$ and obtains an extracted model $\hat{O}$. The extraction is considered successful, if $\hat{O}$ matches one of the criteria [51]: 1) Functional equivalent: $\forall x \in \mathcal{X}, \hat{O}(x) = O(x)$; 2) High fidelity: for a target distribution $\mathcal{D}_{\mathcal{F}}$ over $\mathcal{X}$, $Pr_{x \sim D_F}[S(\hat{O}(x), O(x))]$ is maximized, where $S$ is a similarity function; 3) High accuracy:

given a true task distribution $\mathcal{D}_{\mathcal{A}}$ over $\mathcal{X} \times \mathcal{Y}$, $Pr_{(x,y) \sim D_A}[\text{argmax}(\hat{O}(x)) = y]$ is maximized. $\hat{O}$ with high fidelity tries to replicate the decisions of $O$, including mis-classifications, while $\hat{O}$ with high accuracy aims to match or even exceed the accuracy of $O$.

Following the above terminology, we aim to recover a KF model, in particular $Q$, $R_G$, $R_L^p$ and $R_L^y$, at *high accuracy* or *high fidelity*, and we focus on ESKF in this work. A variety of learning-based approaches have been developed towards this goal [101, 50, 19]. For instance, Tramer et al. applies active learning to adaptively train $\hat{O}$ by selecting data points to query in each round [101]. Though none of the related works investigated control-theory models, we found the learning-based approaches hold promises in addressing our problem here. When considering ESKF in isolation, our task is similar as model extraction against *RNN* models, as both RNN and ESKF have feedback loop between output and input. As such, we can try to find the best parameters that minimize the error between the predicted states outputted by the targeted ESKF $O$ (ground-truth) and attacker's ESKF $\hat{O}$. Gradient-based optimization can be applied here because the ESKF functions are derivable.

A similar research direction is system identification [64], which aims to construct the mathematical models of dynamic systems from measured input-output data. Section 8 reviews the existing methods, but we found none of the are directly applicable to the complex MSF models, in particular ESKF, used by AVs.

### 3.3.1 Challenges

Yet, extracting the parameters from KF models, especially ESKF models encounter prominent challenges that cannot be addressed by the existing approaches. **1)** ESKF is complex, which takes the input generated by the heterogeneous sensors (GNSS, LiDAR and IMU) at vastly different pace. Applying the classical methods under system identification does not yield satisfactory result, as indicated by our evaluation in Section 5.6. **2)** AV is not always controlled by the attacker (e.g., under AS3), and the number of traces about the targeted AV might be small. Given that the search space of the secret is not small (e.g., $Q$ and $R$ are 15x15, 3x3 matrices), the attacker's search strategy has to be highly efficient. **3)** When the output of ESKF (i.e., $\hat{x}$ and $\hat{P}$) is not directly observable, e.g., under scenario AS2 and AS3, the data available to the model extraction is incomplete.

To address these challenges, we proposed a novel method for learning-based KF model extraction, termed TaskMaster, involving techniques like *multi-stage optimization*, *search-space reduction* and *controller simulation*. The details are described next.

# Chapter 4

# Attack Implementation

The goal of the attacker is to learn an ESKF model $\hat{O}$ that mimics the target model $O$. In this section, we first describe how we optimize the training procedure of ESKF to learning $\hat{O}$ in an efficient way when the ESKF output is available. Then, we describe how to train $\hat{O}$ without the ESKF output, by emulating controllers. We summarize the symbols in Table 4.1.

| Symbol | Description | Dim |
|---|---|---|
| $u_{k-1}$ | IMU measurement | $2 \times 1$ |
| $z_k^p$ | Position measurement | $3 \times 1$ |
| $z_k^y$ | Yaw measurement | $4 \times 1$ |
| $x_k$ | Predicted state | $16 \times 1$ |
| $P_k$ | Predicted cov | $15 \times 15$ |
| $y_k$ | Controller output | $4 \times 1$ |
| $p_k$ | Position control | $2 \times 1$ |
| $a_i$ | Acceleration control | $1 \times 1$ |
| $d_i$ | Yaw control | $1 \times 1$ |
| $*Q$ | Observation noise cov | $15 \times 15$ |
| $*R_G$ | GNSS noise cov | $3 \times 3$ |
| $*R_L^p$ | LiDAR position noise cov | $3 \times 3$ |
| $*R_L^y$ | LiDAR yaw noise cov | $3 \times 3$ |
| $O$ | Ground-truth model | - |
| $\hat{O}$ | Extracted model | - |

**Table 4.1:** Symbols used in Section 4. "Dim" is for Dimension. "cov" is for covariance matrix. "*" marks the secret to be inferred by TaskMaster.

## 4.1 Extracting ESKF Alone

Under AS1, TaskMaster uses $u_{k-1}, z_k^p, z_k^y$ (ESKF input, position measurement and yaw measurement) and $x_k$ (ESKF output) to train $\hat{O}$. The attacker can directly sniff IMU output to get $u_{k-1}$. By sniffing GNSS output, $z_k^p$ is obtained. By sniffing LiDAR locator output, $z_k^p$ and $z_k^y$ are obtained. In the end, the attacker obtains a time sequence $T = [t_1, ..., t_i, ...]$ as input, where $t_i$ is $u_{k-1}$, $z_k^p$ or $z_k^y$. For output, $\hat{x_k}$ can be intercepted from the wires between ECUs within AD system, which are produced after $t_i$ is processed by the ESKF. We train $\hat{O}$ in a recurrent way. Specifically, for each round $i$, $\hat{O}$ uses $t_i$ and the last state $P_{i-1}$ as input, and predicts a new state $x_i$ and its covariance $P_i$. The same input is sent to $O$ to generate the predicted state $x_i'$ and its covariance $P_i'$ . Notably, $O$ is treated as a blackbox here. The difference between the output of $O$ and $\hat{O}$ is leveraged to update $\hat{O}$.

We use an optimizer penalized by the logarithmic value of Mean Squared Error (MSE) (denoted as $L$) between $x_i$ and $x_i'$, as shown in Equation 4.1. The difference between $P_i$ and $P_i'$ is not integrated because $P_i'$ is an internal variable that cannot be obtained when $O$ is considered blackbox. We compute the logarithmic MSE to make the convergence process faster.

$$L(x, x') = \log(\frac{1}{N}\sum_{i=1}^{N} \|x_i - x_i'\|^2) \tag{4.1}$$

Training $\hat{O}$ is similar as training an LSTM model at the high level, where unrolling is performed on the ESKF model, as illustrated in Figure 4.1. All “ESKF”s in Figure 4.1 right refer to only one ESKF model, which is the unrolled version of the left figure. Therefore, when training, the feed-forward process that calculates the series $x_k'$ one by one (*i.e.*, unrolled) and then calculates the loss according to Equation 4.1. While in the back-propagation process, the parameters of the ESKF model are only updated once according to the gradient from the loss to each variable (as if not unrolled).

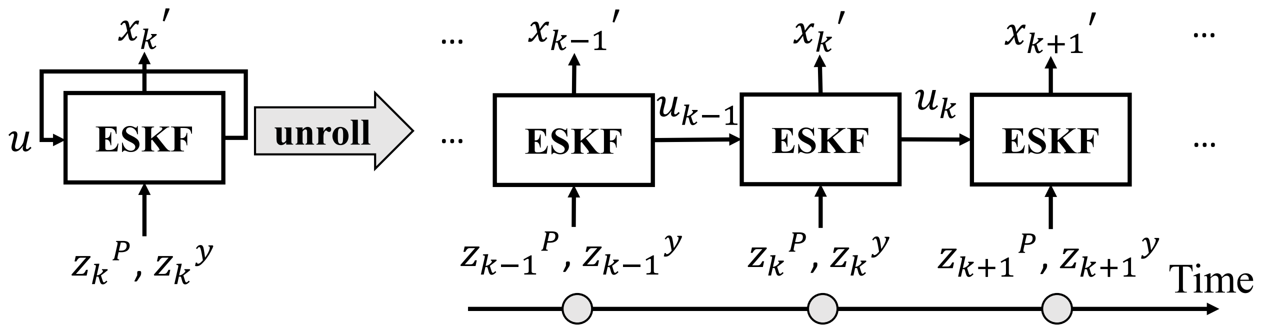

**Figure 4.1:** Training strategy of ESKF.

With the above strategy, we train a shadow ESKF model $\hat{O}$. Alternatively, we can train a shadow LSTM model to extract $\hat{O}$. However, we found this approach did not work well, because some operations like pose transformation are not modeled well under LSTM, and it is hard to make the training converge with unbalanced data (e.g., IMU, LiDAR and GNSS are 50:6.5:1 in data volume).

### 4.1.1 Search-space reduction of $Q$

Though $Q$ is a 15x15 matrix, we found not every value has to be tuned. According to [92], $Q$ can be described with Equation 4.2.

$$\begin{aligned} V_i &= \sigma_{a_n}^2 (\Delta t)^2 I \\ \Theta_i &= \sigma_{\omega_n}^2 (\Delta t)^2 I \\ A_i &= \sigma_{a_\omega}^2 \Delta t I \\ \Omega_i &= \sigma_{\omega_\omega}^2 \Delta t I \\ Q &= \text{diag}(V_i, \Theta_i, A_i, \Omega_i) \end{aligned} \tag{4.2}$$

where $V_i$, $\Theta_i$, $A_i$ and $\Omega_i$ represent velocity, quaternion/pose, accelerometer error state and gyrometer error state. $\Delta t$ is the difference between timestamps. $\sigma_{a_n}$, $\sigma_{\omega_n}$, $\sigma_{a_\omega}$ and $\sigma_{\omega_\omega}$ are the standard deviation of velocity, pose, accelerometer and gyrometer, and they are the variables to be optimized. Each of them is a scalar variable and they are located at the diagonal of the $Q$ matrix. Hence, we limit the optimization process on the 4 variables while avoid touching the others (they can be set to 0), reducing the variables to be optimized from 225 (15x15) to 4.

### 4.1.2 Search-space reduction of $R_G$, $R_L^p$ and $R_L^y$

Similar to $Q$, the search-space of $R_G$, $R_L^p$ and $R_L^y$ can be reduced based on the constraints of the physical world and control theory. As described in Section 2.2, $R_G$, $R_L^p$ and $R_L^y$ are covariance matrices describing deviation of Gaussian noise injected into the related sensor observations, all of them have two properties: 1) it is a diagonal matrix, 2) elements on the diagonal of are non-negative. These properties are based on the related mathematical equations that hold universally [109, 79]. According to previous works on both GNSS and LiDAR sensor development [106], variance of each dimension in the measurements are independent from other elements. Hence, we can fix the values of the elements not on the diagonal and add a check to avoid updating the diagonal elements to negative values. The number of elements to be optimized is reduced from 27 ($R_G$, $R_L^p$ and $R_L^y$ are all $3 \times 3$ matrices) to 9 (the diagonal elements), and the search space of each element is cut to half.

### 4.1.3 Multi-stage optimization

Learning $\hat{O}$ could be based on maximum likelihood estimation (MLE), which seeks a set of parameters that maximizes a likelihood function. However, MLE assumes that the output is solely dependant on the current input. When the output is also dependent on latent variables (i.e., unobserved or hidden variables), MLE does not work well [74]. Such problem exists in ESKF: the input from sensors as well as the ESKF model states decide the prediction. In addition, the data generation frequencies of IMU, LiDAR and GNSS are vastly different (roughly 50:6.5:1 on the KAIST dataset we use [52]), resulting in unstable input dimensions that cannot be easily handled by MLE.

To address the aforementioned issues, *expectation maximization (EM)* [74] can be performed which introduces an extra estimation step. EM has been particularly effective in learning Gaussian Mixture Model (GMM). The dataset used to train GMM consists of points generated from one or more Gaussian processes in different paces. The two steps of EM are:

- **E-Step.** Estimate the expected value for each latent variable.
- **M-Step.** Optimize the parameters using maximum likelihood.

Under EM, the initial estimation by E-step can assign random values to the latent variables. Along the iterations, the optimized model from M-step can estimate the latent parameters for existing and new data points. We adopt this idea and develop a multi-stage optimization technique for ESKF. Specifically, $Q$, $R_G$, $R_L^p$ and $R_L^y$ are partitioned into two groups, *i.e.*, $G_1 = [Q]$ and $G_2 = [R_G, R_L^p, R_L^y]$. Since the two groups have different frequencies and dimensions, we can choose different learning rate and decay rate. For $G_1$, parameters in $R_G, R_L^p, R_L^y$ will be treated as constant and only $Q$ will be optimized. For $G_2$, $Q$ will be constant and other parameters will be optimized.

Notably, EM has been leveraged to tune KF, but it has to be adjusted under TaskMaster because we use the input and output of *another blackbox KF model* for optimization. In Algorithm 1, we summarize the whole training process.

## 4.2 Extracting ESKF with Controllers

Under AS2 and AS3, the adversary has no visibility to the ground-truth output ($x'_k$) of $O$, so the loss $L$ cannot be directly computed for training. On the other hand, $x'_k$ is sent to the controller, who outputs $y_k$ (including steering, throttling and braking) as the control signal,

**Algorithm 1:** Attack workflow under AS1

```
Input  : $N$ measurement $T = [t_1, ..., t_N]$, the output of $O$ refStates, MaxEpoch
Output: Inferred $Q$, $R_G$, $R_L^p$, $R_L^y$
Initialize $x_0$, $P_0$, $Q$, $R_G$, $R_L^p$, $R_L^y$, myStates;
$x_{k-1} \leftarrow x_0$; $P_{k-1} \leftarrow P_0$; cnt $\leftarrow$ 1;
for $i \leftarrow 1$ to MaxEpoch do
    while $cnt \leq N$ do
        $t_k \leftarrow$ get(T,cnt);
        if $t_k$ is from IMU then
            $x_k$, $P_k \leftarrow$ predictIMU($x_{k-1}$, $P_{k-1}$, $t_k$);
        end
        if $t_k$ is from GNSS then
            $x_k$, $P_k \leftarrow$ updateGNSS($x_{k-1}$, $P_{k-1}$, $t_k$);
        end
        if $t_k$ is from LiDAR then
            $x_k$, $P_k \leftarrow$ updateLiDAR($x_{k-1}$, $P_{k-1}$, $t_k$);
        end
        add $x_k$ into myStates;
        $x_{k-1} \leftarrow x_k$; $P_{k-1} \leftarrow P_k$;cnt $\leftarrow$ cnt + 1;
    end
    Loss $\leftarrow$ log(MSE(myStates, refStates)) ;
    optimize($Q$,Loss);
    optimize($R_G$, $R_L^p$, $R_L^y$,Loss);
end
Return $Q$, $R_G$, $R_L^p$, $R_L^y$;
```

which is nonetheless observable. Hence, the attacker may regard the ESKF and the trailing controller as a whole, so she can train the series (ESKF + controller) with the observable ESKF input $(u_{k-1}, z_k^p, z_k^y)$ and the observable controller output$(y_k)$. Then she can readily extract the ESKF $O$ from the trained series. Noticeably, the attacker does not need to know what controllers are used by the victim AVs, and *we do not consider controller as a secret.* In fact, the attacker can implement a trainable controller or even use an out-of-box, open-source implementation.

Below, we first introduce the mechanisms of AD controllers, and then describe how we adjust the workflow of AS1 to fit AS2 and AS3. Finally, we describe of an optional component, anomaly filter.

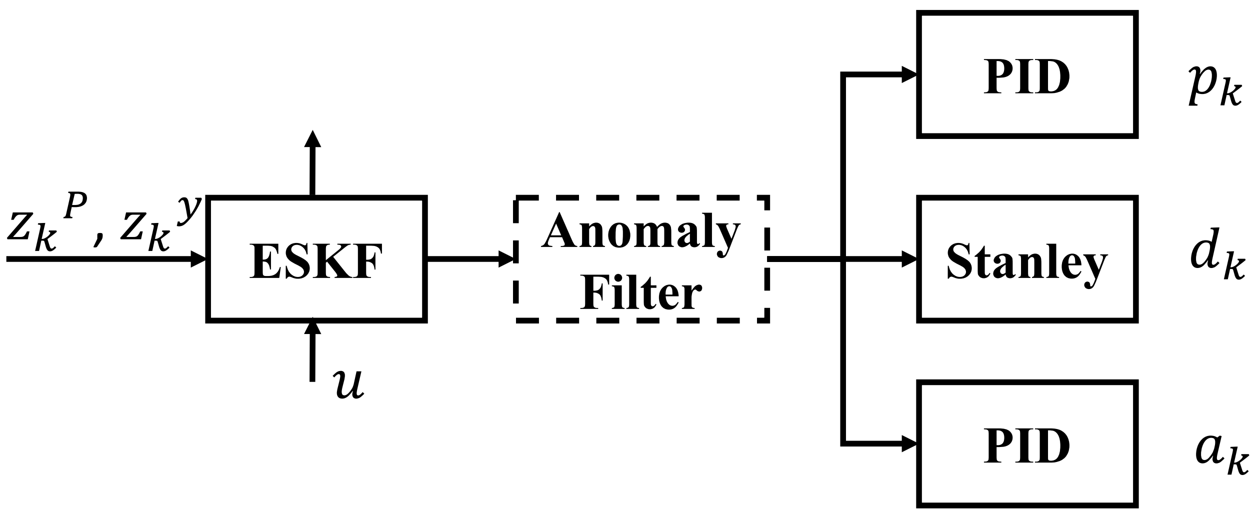

**Figure 4.2:** Workflow of ESKF combined with controllers. Anomaly filter is an optional component included by some AD like Baidu Apollo.

**Mechanisms of the AD controllers.** As described in Section 2.3, Baidu Apollo uses PID controller for longitudinal control and LQR controller for lateral control (steering). When an AV receives a map and a destination point, it generates a planned trajectory consisting of a sequence of reference positions on the map ($tp^*$), and the AD controller generates corresponding control vectors to minimize the error between the current position and the reference positions. PID controller in AD generates the longitudinal control vector, based on the predicted position ($pos_k$) and velocity ($vel_k$) from the ESKF output $x_k$. The PID control vector contains the planned next position ($p_k$) and acceleration ($a_k$), which are used to derive control commands (throttle and brake). They can be computed through the equation below:

$$\begin{aligned} p_k &= K_{pp} \times \text{MinDist}(tp^*, pos_k) + K_{ip} \times \sum_{m=k-M}^{k-1} (p_m) \\ a_k &= K_{pa} \times (max\_speed - vel_k) + K_{ia} \times \sum_{n=k-N}^{k-1} (a_n) \end{aligned} \tag{4.3}$$

Where MinDist computes the minimum distance between the reference positions $tp^*$ and the current position $pos_k$, $max\_speed$ represents the maximum speed allowed on the AV during navigation, $M$ and $N$ are the number of positions and accelerations from the controller output in the past, $p_m$ and $a_n$ are the corresponding positions and accelerations. $K_{pp}$, $K_{ip}$, $K_{pa}$ and $K_{ia}$ are the controller parameters. Notably, the above equation contains "Integral" (modeling the past) and "Proportional" (modeling the present), but does not contain "Derivative" (modeling the future), which is based on Apollo's implementation.

Similar to PID controller, LQR controller generates the lateral control vector (yaw) to derive the control commands (steering), based on the planned trajectory, the past states, and the present state (from ESKF output). However, this process requires solving Discrete-time Arithmetic Riccati Equation (DARE), which cannot be implemented compatible with gradient descending. Specifically, people use iterative methods to get numerical solution of the equation, whose process is not derivable. As such, if we simulate LQR controller after ESKF, we will not be able to derive the gradients to optimize ESKF parameters. To address this issue, we implement *Stanley controller* [40] and use it replace LQR controller. Stanley controller was used for lateral control during the 2005 DARPA Grand Challenge of Autonomous Robotic Ground Vehicles [23] by the Stanford team, who won the first place. It is a perfect match for our goal because its equations related to yaw computation are derivable, as shown in Equation 4.4. Since TaskMaster does not extract the controller parameters, using another controller of similar performance is acceptable.

$$\begin{aligned}
front\_axle\_vec &= (\cos(yaw_k + \frac{\pi}{2}), -\sin(yaw_k + \frac{\pi}{2}))^\top \\
error\_front\_axle &= (x_{min^*}, y_{min^*})^\top \cdot front\_axle\_vec \\
\theta_e &= \text{normalize\_angle}(cyaw_{min^*} - yaw_k) \\
\theta_d &= \arctan 2(k \times error\_front\_axle, vel_k) \\
d_k &= \theta_e + \theta_d
\end{aligned} \tag{4.4}$$

Where $yaw_k$ is the yaw (or heading) derived from $quat_k$ of ESKF's output, $front\_axle\_vec$ is the estimated front axle velocity, $x_{min^*}$ and $y_{min^*}$ are the x and y coordinates of the nearest position on the planned trajectory to the current position, $error\_front\_axle$ represents the error to the reference states on the front axle, $cyaw_{min^*}$ is the yaw associated with the nearest position, normalize_angle normalizes the difference between $cyaw_{min^*}$ and $yaw_k$ into $[-\pi, \pi]$, $\theta_d$ and $\theta_e$ are the cross track error and the heading error, and $d_i$ is the resulted yaw control vector. $k$ is the only tuning parameter and it can be optimized along with ESKF in our method.

**Extracting $\hat{O}$.** In Figure 4.2, we illustrate how ESKF, PID controller and Stanley controller are connected for AS2 and AS3. Compared to AS1, $x_k$ is replaced by $p_k$, $d_k$ and $a_k$ (position, yaw, and acceleration). To accommodate this change, we modify the loss of Equation 4.1 to Equation 4.5.

$$\begin{aligned} L(p,d,a,p',d',a') = {} & \lambda_p \texttt{log}(\frac{1}{N}\sum_{i=1}^{N}\|p_i - p_i'\|^2) \\ & + \lambda_d \texttt{log}(\frac{1}{N}\sum_{i=1}^{N}(d_i - d_i')^2) \\ & + \lambda_a \texttt{log}(\frac{1}{N}\sum_{i=1}^{N}(a_i - a_i')^2) \end{aligned} \tag{4.5}$$

Where $\langle p,d,a\rangle$ and $\langle p',d',a'\rangle$ are the controller outputs linked to $\hat{O}$ and $O$. $\lambda_p$, $\lambda_d$ and $\lambda_a$ are the weights for each controller loss. After empirical analysis, we found the optimization process converges faster when $\lambda_p$ is much higher than $\lambda_a$ and $\lambda_d$, since values of position coordinates ($\sim 3e+5$) are much larger than that of yaw ($\sim 1e+2$) and acceleration ($\sim 1e+1$).

Another difference to AS1 is that $\langle p,d,a\rangle$ will not be fed back to train $\hat{O}$, thereby training becomes non-recurrent. We make such change to avoid amplifying the error to ESKF caused by the inaccurate modeling of the controllers. Training ESKF under AS2 and AS3 follow the same workflow, except the input to ESKF and the output of controllers have noises.

**Anomaly filter** We found some AD systems add another anomaly filter between MSF and controller, when the output of MSF is too too noisy to direct the controller. For instance, Baidu Apollo takes the output of ESKF ($x_k$) and corrects it with other information, before feeding it to the controllers, as shown in Figure 4.2. Since the source code of the anomaly filter is not released in Baidu Apollo, we introduce a Multilayer Perceptron (MLP) model to replace it, which can be trained together with ESKF, under the same loss function. We choose MLP because the input size is small ($x_k$ is 16x1). Our MLP has 5 layers, and each layer has 16 neurons. The activation function is ReLU. The MLP model is initialized by identity matrices and all zero bias.

# Chapter 5

# Evaluation

In this section, we evaluate how TaskMaster recovers ESKF models with the real-world data. To evaluate TaskMaster against different models, we re-implemented one ESKF model based on [92] (termed `Sola-ESKF`), and obtained the blackbox ESKF model of Baidu Apollo v2.5 (termed `Apollo-ESKF`), which is also the major evaluation platform for AD security research (e.g., [17, 54, 94]). For Stanley and PID controllers, we re-implemented them based on [89] and [9]. Our implementation of TaskMaster includes 756 LoC (lines of code) for ESKF and data pre-processing, 213 LoC for controller and 348 LoC for the training process.

We first describe the experiment settings. Then, we elaborate the attack results on `Sola-ESKF` and `Apollo-ESKF` under the three attack settings. Finally, we compare TaskMaster against the baseline system identification and evaluate how TaskMaster can help the spoofing attack proposed in [90]. In Section 5.3, we evaluate the influence of different parameters.

## 5.1 Experiment Settings

### 5.1.1 Evaluation datasets

To evaluate TaskMaster, we use the KAIST Complex Urban sensor traces [52] (termed `KAIST` hereinafter). The authors of [52] collected sensor data, including Image, LiDAR, GPS, IMU and Encoder, from the complex urban areas of four different cities, with a mapping vehicle. In total there are 31 *traces* (each trace corresponds to one trip of the vehicle), and 12 are in highway and 19 are in downtown. Similar to [90], we selected 5 traces among them for evaluation, which are labeled as *local08*, *local31*, *local07*, *highway06* and *highway17* by `KAIST`, because `Sola-ESKF` and `Apollo-ESKF` cannot achieve reliable performance on the rest. According to Section 6.2 of [90], the 5 selected traces have the smallest average MSF state uncertainty in their categories (i.e., local and highway). According to the extended version of [90], these traces all have complete sensor data (e.g., some other traces do not have complete IMU data) and provide a complete motion history.

We use the `KAIST` sensor data as input and feed them to the two ESKF models to obtain two sets of ESKF states output as the ground-truth. We name the dataset consisting of the ESKF states output from `Sola-ESKF` as `Sola-Output`. The second dataset has the ESKF states output from `Apollo-ESKF`, and we name it `Apollo-Output`.

### 5.1.2 Evaluation metrics

We consider two metrics to evaluate TaskMaster, focusing on *fidelity* and *accuracy.*

- **Difference between the parameters (PER).** Since `Sola -ESKF` is implemented by us, we have the ground-truth about the secret parameters. Therefore, we compute the difference between the parameter values learnt from the evaluation datasets and the ground-truth values, which we term *Parameter Error Rate (PER)* and show in the equation below:

$$PER = \frac{\overline{|\hat{\theta} - \theta|}}{\overline{|\theta|}} \tag{5.1}$$

Where $\hat{\theta}$ represents the learnt parameter of the substitutional model and $\theta$ represents the ground truth. The matrix mean is computed on their difference. This metric evaluates the fidelity of TaskMaster. In real-world settings, when the ESKF model parameters are proprietary, we cannot compute this metric. So we compute PER only for `Sola-ESKF`.

- **Distance between the predicted states (SER).** The ultimate goal of the adversary is to build an ESKF model of high prediction accuracy, and the parameters stolen by TaskMaster should serve this purpose. Hence, for each trace, we use the inferred ESKF and the ground-truth ESKF to predict the vehicle states using the sensor inputs, and compute the Root Mean-Squared Error (RMSE) between the states, termed *State Error Rate (SER)*, with the equation below:

$$SER = \sqrt{\frac{1}{N}\sum_{i=1}^{N} ||y_i - y_i'||^2} \tag{5.2}$$

Where $N$ represents the number of positions, $y_i'$ and $y_i$ represent the predicted states (they are vectors) by the ground-truth ESKF $O$ and the inferred ESKF $\hat{O}$. Since the states can be generated by a blackbox ESKF, we evaluate against both `Sola-ESKF` and `Apollo-ESKF`.

### 5.1.3 Experiment parameters

We choose Adam as the optimizer. We set the maximum number of epochs to 200 for the training process. For the learning rate, we adopt step decay [62]. For $Q$, the learning rate is set to $1e-3$ for the first 10 epochs and then changed to $1e-4$, $1e-5$ and finally $1e-6$ for the following 30, 70 and 100 epochs respectively. For $R$, the learning rate is set to $1e-5$ initially for first 10 epochs, and then $1e-6$, $1e-7$ and finally $1e-8$ after 30, 70 and 100 epochs respectively. When the anomaly filter is emulated, its replacement MLP also needs to be trained, and we set the learning rate to $1e-9$ initially and then decreased to $1e-10$ and $1e-11$ after 30 and 80 epochs respectively.

We select a subset of one trace (its positions and corresponding ESKF output) for training, because 1) using the whole trace is time-consuming (it contains hundreds of thousands of samples), and 2) the initialization stage (usually the first 40 seconds) of an MSF module yields unreliable output which should be removed. All the other traces are used for testing (the first 40 seconds' data are removed similarly). We term the number of selected positions for training as $N$, and set it to 1000 after empirical analysis. Other parameters for training, including the three loss weights of Equation 4.5 ($\lambda_p, \lambda_a$ and $\lambda_d$) are set to 100, 1, and 0.1 respectively. The impact of different parameter values in training ($N$, $\lambda_p, \lambda_a$ and $\lambda_d$), initialization and controllers will be evaluated in Section 5.3.

### 5.1.4 Experiment environment

The training and testing process are done on a workstation of one NVIDIA RTX 2080 Ti GPU and one AMD 5950x with 32 GB memory. The code runs on PyTorch 1.11.0 with Nvidia CUDA 11.5 support. All the data used in the experiments are in `float64` format.

| AS | Training | PER | Testing (SER) | | | | |
|---|---|---|---|---|---|---|---|
| | | | *lo08* | *lo07* | *lo31* | *hi06* | *hi17* |
| AS1 | *lo08* | 0.01347 | - | 0.067 | 0.042 | 0.076 | 0.089 |
| | *hi17* | 0.01297 | 0.043 | 0.029 | 0.073 | 0.038 | - |
| AS2 | *lo08* | 0.00978 | - | 0.018 | 0.028 | 0.026 | 0.036 |
| | *hi17* | 0.00206 | 0.024 | 0.013 | 0.018 | 0.036 | - |
| AS3E | *lo08* | 4.5431 | - | 1.76 | 1.24 | 1.33 | 3.15 |
| | *hi17* | 4.6247 | 1.43 | 2.88 | 2.28 | 4.19 | - |
| AS3G | *lo08* | 3.9916 | - | 1.53 | 1.89 | 2.57 | 1.21 |
| | *hi17* | 4.3111 | 1.51 | 0.58 | 1.43 | 5.67 | - |
| AS3R | *lo08* | 5.8869 | - | 4.24 | 2.10 | 1.11 | 1.33 |
| | *hi17* | 4.2485 | 2.12 | 1.87 | 4.11 | 0.88 | - |

**Table 5.1:** Evaluation result on `Sola-ESKF`. The result of each testing trace is represented as SER. PER is the same for each training trace. PER could be larger than 1 if the error between the extracted value and the ground truth is larger than the ground truth itself. AS3E, AS3G and AS3R are AS3 under Exponential, Gamma and Rayleigh noises. "lo" and "hi" are short for "local" and "highway".

## 5.2 Extracting `Sola-ESKF`

In this subsection, we evaluate the effectiveness of TaskMaster when extracting the secret parameters from `Sola-ESKF`. Since we have white-box access to it, we assess both PER (it is averaged for $Q$, $R_G$, $R_L^p$ and $R_L^y$) and SER. To show the impact of the training traces, we selected 2 different traces to train each model. The extracted models are tested on all 5 traces. Table 5.1 summarizes the results.

### 5.2.1 Result on AS1 (Intrusive In-AV Attacker)

The 2 training traces are *local08* and *highway06*, which represents local (roads of downtown areas) and highway navigation respectively. As their environments are vastly different, the sensor noises and the AV kinetics (positions, velocities and heading poses) also have big differences. In general, *local08* has a smaller value variation than *highway06*. For example, both $x$ and $y$ coordinate of *local08* traces vary in 100-meter level (between $3.511e+5$ and $3.514e+5$, $4.022e+5$ and $4.023e+5$, respectively), while $x$ and $y$ coordinate of *highway06* traces vary in 1000-meter level (between $3.49e+5$ and $3.53e+5$, $4.02e+5$ and $4.03e+5$, respectively). For trace length, *local08* has 30,704 recorded data points while *highway06* has 205,375 recorded data points.

For a training trace, we select $N$ points (set to 1000) to construct the training dataset, following this rule: from the first 15000 points, we randomly select a starting point from the points indexed in $[1001, 10000]$, and consecutively collect the following 1000 points for training. We drop the first $N$ points, as ESKF is in the "warm-up" stage in the beginning, and the ESKF output is less stable. The same strategy has been adopted by [90]. For each testing trace, we select the points indexed in $[1001, 15000]$ to construct the dataset. The secret parameters of $\hat{O}$ are all set to random values before training.

As shown in Table 5.1, the extracted $\hat{O}$ is quite similar as the ground-truth $O$: PER is 0.01347 and 0.01483 for the two traces. It indicates TaskMaster is able to learn the secret parameters at very high fidelity. SER ranges from 0.042 to 0.089 (the unit is m). Given that L4-standard AV asks for centimeter-level (0.01) RMSE, this result is satisfactory: if the targeted ESKF has reached centimeter-level RMSE, tuning $\hat{O}$ to the same level is deemed much easier comparing to tuning from the scratch.

We also found the impact of the training traces is fairly small. The average SER resulted from *local08* and *highway17* are 0.0685 and 0.0458 separately. Our result also suggests the cost of attack is quite low: by just collecting 1000 data points on one trace (about *25 seconds of AV driving*), the attacker can extract a high-fidelity and high-accuracy ESKF model. In Section 5.3, we show that increasing $N$ from 1000 to 5000, a small performance gain can be obtained but the training overhead also significantly increase.

### 5.2.2 Result on AS2 (Non-intrusive In-AV Attacker)

Since the attacker cannot get the output of the ESKF model, but only the output of the followed controllers in this setting, the extracted $\hat{O}$ is expected to be less accurate. We follow the same training and testing strategy as AS1 ($N$ is also 1000 points).

Interestingly, the result shows AS2 can achieve even lower PER and SER compared to AS1, e.g., 0.013 vs 0.029 SER when training on *highway17* and testing on *local07.* We speculate controller outputs actually enhance the training performance, as controllers also handle noises.

### 5.2.3 Result on AS3 (AV Follower)

In this setting, we injected noises into the sensor input and controller output.

Though noises following normal distributions are usually used in academic studies about AV security (e.g., [90]), the real-world noises are often more complex. As such, we select Exponential, Gamma and Rayleigh noises based on a survey of noises [14], These noise models are widely used in radiation-based systems, such as X-ray, LiDAR and MRI systems. We define the model of the noise injection as follows.

$$\begin{aligned}\delta(u_{k-1}) &= u_{k-1} + n_1 \\ \delta(z_k^p) &= z_k^p + n_2 \\ \delta(z_k^y) &= z_k^y + n_3 \\ \delta(y_k) &= y_k + n_4\end{aligned} \tag{5.3}$$

where $\delta(u_{k-1})$, $\delta(z_k^p)$, $\delta(z_k^y)$ and $\delta(y_k)$ are measurements from $u_{k-1}$, $z_k^p$, $z_k^y$ and $y_k$ under the influence of noises $n_1$, $n_2$, $n_3$ and $n_4$. We adapt three noise models – Exponential noise, Gamma noise, and Rayleigh noise – for noise injection in AS3 based on a survey [14]. $\lambda$, $\alpha$, $\beta$ and $\sigma$ are model parameters. Three noise models are described as below:

**Exponential noise.** Exponential noise follows a distribution whose probability density function (PDF) is:

$$f(x;\lambda) = \lambda e^{-\lambda x} \tag{5.4}$$

where $\lambda > 0$. Mean of Exponential noise is $\frac{1}{\lambda}$, and standard deviation is also $\frac{1}{\lambda}$.

**Gamma noise.** Gamma noise follows a distribution whose PDF is:

$$f(x;\alpha,\beta) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} \tag{5.5}$$

where $\alpha > 0, \beta > 0$, and $\Gamma(\cdot)$ is gamma function [4]. Mean of Gamma noise is $\frac{\alpha}{\beta}$, and standard deviation is $\frac{\sqrt{\alpha}}{\beta}$.

**Rayleigh noise.** Rayleigh noise follows a distribution whose PDF is:

$$f(x;\sigma) = \frac{x}{\sigma^2} e^{-x^2/2\sigma^2} \tag{5.6}$$

where $\sigma > 0$. Mean of Rayleigh noise is $\sigma\sqrt{\frac{\pi}{2}}$, and standard deviation is $\sigma\sqrt{\frac{4-\pi}{2}}$.

For $n_1$, $n_2$ and $n_3$, we select $\lambda_1 = 10$ for Exponential noises, $\alpha_1 = 0.1, \beta_1 = 2$ for Gamma noises, and $\sigma_1 = 0.05$ for Rayleigh noises. For $n_4$, we select $\lambda_2 = 1$ for Exponential noises, $\alpha_2 = 1, \beta_2 = 2$ for Gamma noises, and $\sigma_2 = 0.5$ for Rayleigh noises. These settings are considered reasonable given that the extracted ESKF can tolerate meter-level error and sensors can tolerate decimeter-level errors. According to Table 5.1, we found PER is significant higher, however, this is expected, as the KF parameters have to be changed to tolerate the noises. SER ranges from 1.26 to 3.07, showing the impact of noises cannot be neglected. Yet, given that the academic implementations of ESKF have meter-level RMSE (see Section 3.1), our RMSE is comparable.

## 5.3 Impact of Parameters

| | 100 | 500 | 1000 | 2000 | 3000 | 4000 | 5000 |
|---|---|---|---|---|---|---|---|
| PER | 0.4377 | 0.1274 | 0.01347 | 0.01562 | 0.01285 | 0.02014 | 0.009974 |
| SER (m) | 0.59 | 0.27 | 0.03 | 0.08 | 0.05 | 0.05 | 0.04 |
| Training Time (s) | 11762.11 | 13270.62 | 13878.90 | 14957.63 | 15724.83 | 16765.74 | 17833.91 |
| Time per Iteration (s) | 168.03 | 189.58 | 198.27 | 213.68 | 224.64 | 239.51 | 254.77 |

**Table 5.2:** The impact of $N$ on AS1. The targeted model is `Sola-ESKF`. *local08* and *highway17* are for training and testing.

| | | $Q$ | | | $R$ | | |
|---|---|---|---|---|---|---|---|
| AS | Original | $[1e-3, 1e-2]$ | $[1e-2, 1e-1]$ | $[1e-1, 1e0]$ | $[1e-6, 1e-5]$ | $[1e-5, 1e-4]$ | $[1e-4, 1e-3]$ |
| AS1 | 0.01347 | 0.01220 | 0.01265 | 0.14031 | 0.03868 | 0.10514 | 0.92062 |
| AS2 | 0.00978 | 0.09540 | 0.12428 | 0.17274 | 0.12237 | 0.16641 | 0.50272 |

**Table 5.3:** Comparison of PER under different parameter initialization ranges in `Sola-ESKF`. The training trace is *local08*. "Original" are the values (matrices) used in previous evaluation.

| | $Q$ | | | $R$ | | |
|---|---|---|---|---|---|---|
| Original | $[1e-3, 1e-2]$ | $[1e-2, 1e-1]$ | $[1e-1, 1e0]$ | $[1e-6, 1e-5]$ | $[1e-5, 1e-4]$ | $[1e-4, 1e-3]$ |
| 0.85 | 0.97 | 1.03 | 1.73 | 0.88 | 2.43 | 5.64 |

**Table 5.4:** Comparison of SER under different parameter initialization ranges in `Apollo-ESKF` in AS2. The training trace is *local08* and the testing trace is *highway17*.

### 5.3.1 The number of points for training ($N$)

We set $N$ to 1000 as the default setting. Here, we evaluate how TaskMaster is influenced by different $N$. Intuitively, a small $N$ will introduce more errors to each state, as ESKF might not be stabilized yet. Though a large $N$ can overcome this issue, the training process will be longer, and more storage will be consumed to store the prior states.

| | $K_{pp}$ | | | $K_{pa}$ | | |
|---|---|---|---|---|---|---|
| Original | $[1e-1, 1e0]$ | $[1e0, 1e+1]$ | $[1e+1, 1e+2]$ | $[1e-2, 1e-1]$ | $[1e-1, 1e0]$ | $[1e0, 1e+1]$ |
| 0.00978 | 0.01027 | 0.05147 | 7.34141 | 0.03107 | 0.36186 | 16.85203 |

**Table 5.5:** Comparison of PER under different controller parameter initialization ranges in `Sola-ESKF` in AS2. The training trace is *local08* and the testing trace is *highway17*.

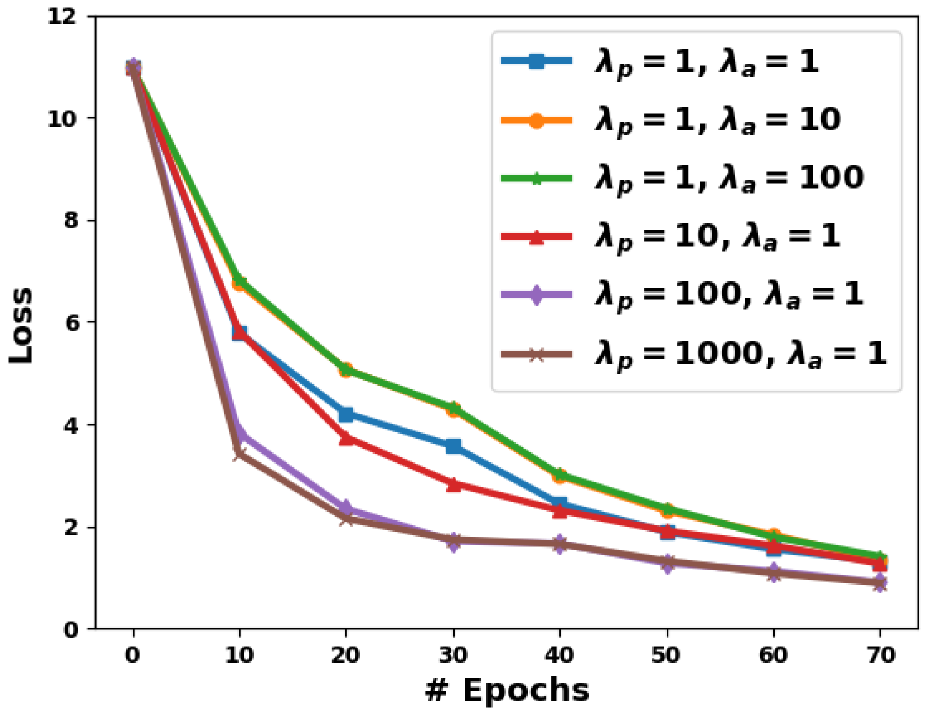

**Figure 5.1:** Loss in every 10 epochs during training with $\lambda_d = 1$. The targeted model is `Sola-ESKF`. *local08* is used for training.

We vary $N$ from 100 to 5000 and evaluate `Sola-ESKF`. It turns out PER and SER are significantly decreased (from 0.4377 to 0.01347 and from 0.59 to 0.03) when $N$ is increased from 100 to 1000, as shown in Table 5.2. When $N > 1000$, PER and SER remain the same level. However, the time overhead increases greatly (from 198.27s to 254.77s per iteration) with increasing $N$ from 1000 to 5000. The benefit brought by a larger $N$ is diminished by the overhead it incurs, and therefore we use $N = 1000$ as default.

### 5.3.2 Weights in loss function ($\lambda_p$, $\lambda_d$ and $\lambda_a$)

As described in Section 4.2 and Equation 4.5, we assign different weights to the loss incurred by different controller output. We assess how the ratio between them impacts the prediction results. We fix $\lambda_d$ to 1, and change $\lambda_p$ from 1 to 1000, and $\lambda_a$ from 1 to 100. Figure 5.1 shows the loss in every 10 epochs given different combination of $\lambda_p$ and $\lambda_a$.

When setting $\lambda_p$ as 1, with the increase of the $\lambda_a$, the optimization process converges similarly comparing to $\lambda_a$ = 1, 10 and 100. In contrast, we can observe a significant improvement on the convergence speed with the increase of $\lambda_p$. The reason for this improvement is that the value scale of position (1e+5) is far larger than the velocity (1e+2) and the heading angle (between 1e0 and 1e+2). Therefore, we set $\lambda_p$ to 100 with the other two weights as 1.

### 5.3.3 Initialization and controller parameters

We initialize the values of $Q$ and $R$ based on the results of existing works [92]. Here we assess the effectiveness of TaskMaster when different initial values are chosen.

We first assess the impact on `Sola-ESKF`, where the ground-truth of $Q$ and $R$ are known. We tested with 3 different ranges: $[1e-3, 1e-2]$, $[1e-2, 1e-1]$ and $[1e-1, 1e0]$ for $Q$, and $[1e-6, 1e-5]$, $[1e-5, 1e-4]$ and $[1e-4, 1e-3]$ for $R$. In each range, we draw 50 random values in uniform distribution, and run the experiment 50 times with the same setting as Section 5.2. Table 5.3 compares the average PER in AS1 and AS2. As we can see, the impact is relatively small for both settings, except when $Q$ falls in $[1e-1, 1e0]$ and $R$ falls in $[1e-4, 1e-3]$. The impact by $R$ is larger, and we speculate this is because the ground-truth $R$ are in smaller range ($1e-5$ level) compared with $Q$ ($1e-2$ level).

We then assess `Apollo-ESKF` in AS2. As the initial values of $Q$ and $R$ are not far away from the values of `Sola-ESKF`, we can learn whether starting from another ESKF model is important to get closer to an industry-grade ESKF. The answer seems to be negative. For $Q$ and $R$, the same ranges are tested as the previous experiment. The result is summarized in Table 5.4.

For AS2 and AS3, PID and Stanley controllers need to be simulated. According to Equation 4.3, some controller parameters need to be initialized ahead. Here we evaluate the impact of these parameters and show the results about two parameters $K_{pp}$ and $K_{pa}$. We chose `Sola-ESKF` as the target model and show its PER under AS2 when different values are used. The results are shown in Table 5.5. It turns out the performance of extracted model degrades drastically, when the controller parameters are in different ranges. To notice, the controller parameters are not secret, and the original values we use come from the online implementations [89]. Therefore, simulating functional controllers are important for attacks under AS2 and AS3, but they do not need to be the same as the targeted AV.

## 5.4 Extracting `Apollo-ESKF`

| AS | Training | Testing (SER) | | | | |
|---|---|---|---|---|---|---|
| | | *l08* | *l07* | *lo31* | *hi06* | *hi17* |
| AS1 | *lo08* | - | 0.37 | 0.42 | 0.91 | 1.18 |
| | *hi17* | 0.95 | 0.72 | 0.68 | 0.89 | - |
| AS2 | *lo08* | - | 1.26 | 1.01 | 1.03 | 0.62 |
| | *hi17* | 1.12 | 0.96 | 0.88 | 0.79 | - |
| AS3E | *lo08* | - | 1.72 | 1.82 | 2.03 | 1.96 |
| | *hi17* | 1.92 | 1.48 | 2.41 | 1.27 | - |
| AS3G | *lo08* | - | 1.45 | 1.63 | 2.11 | 3.07 |
| | *hi17* | 1.26 | 1.43 | 1.55 | 1.90 | - |
| AS3R | *lo08* | - | 1.78 | 2.08 | 2.13 | 1.49 |
| | *hi17* | 1.71 | 1.82 | 1.56 | 1.33 | - |

**Table 5.6:** Evaluation result on `Apollo-ESKF`. The result in each cell is represented as SER, as `Apollo-ESKF` is blackbox.

In this subsection, we report our results of extracting `Apollo-ESKF`, which is blackbox and might have components not modeled by us. In Table 5.6, we show the SER of the 5 testing traces, paired to the two training traces (*local08* and *highway17*). We only measure SER, as we have no knowledge about the ground-truth parameters of `Apollo-ESKF`.

It turns out for AS1 and AS2, the error rate significantly increased. There are 6 cases with SER more than 1 in testing, e.g., training on *local08* and testing on *highway17* (1.18) in AS1 / *local07* (1.26), *local31* (1.01) and *highway06* (1.03) in AS2. All the other SERs are in decimeter level. We speculate the rise of SER is caused by the additional procedures of `Apollo-ESKF` not implemented by us. In addition to the SER on a whole trace, we also take a closer look at the error distribution on different trace locations. In Figure 5.2, we show the distribution of RMSE on 2D locations of *local08* under AS1, and it turns out 60% locations have less than 0.95 SER, and only a small fraction (around 5%) of locations have high SER (more than 2).

With noises injected in AS3, errors of attacker's observation rise to meter-level: the average SER training on *local08* under Exponential, Gamma and Rayleigh noises rise to 1.88, 2.67 and 1.87, respectively. Interestingly, the SERs between `Sola-ESKF` and `Apollo-ESKF` are much closer comparing to AS1 and AS2.

| Testing (SER) | | | | |
|---|---|---|---|---|
| *local08* | *local07* | *local31* | *highway06* | *highway17* |
| 12.60 | 8.15 | 9.18 | 7.90 | 5.31 |

**Table 5.7:** Comparison between `Sola-ESKF` and `Apollo-ESKF` on the same `KAIST` dataset.

Still, we want to point out that the inferred ESKF is useful. In particular, we run `Sola-ESKF` and `Apollo-ESKF` on `KAIST` and derive SER on their output (`Sola-Output` and `Apollo-Output`), under AS2. Table 5.7 shows the SER of all 5 traces. The best case has 5.31 SER while the worst case has as high as 12.60 SER, and the average is 8.63. In the meantime, our extracted $\hat{O}$ trained on *local08* has 0.98 SER in average. Hence, starting from the extracted model, parameter tuning is expected to be much easier for an adversary.

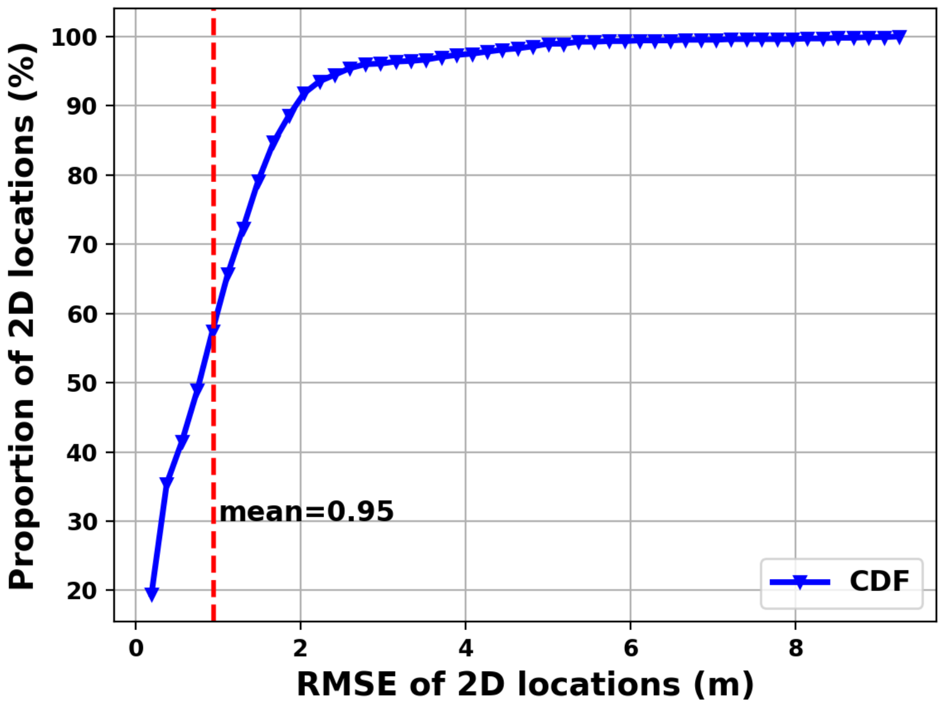

**Figure 5.2:** Distribution of RMSE on 2D locations in AS1 on `Apollo-ESKF`. The training trace is *highway17* and the testing trace is *local08*.

## 5.5 Modeling ESKF with RNN

Given that the output of ESKF is also leveraged as input to generate the next state, we can emulate ESKF with an RNN, and leverage the existing libraries and optimizers from ML frameworks like PyTorch to train it. Here we show the structure of Simple RNN [27] in Equation 5.7.

$$\begin{aligned} h_t &= \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \\ y_t &= \sigma_y(W_y h_t + b_y) \end{aligned} \tag{5.7}$$

Where $x_t$ is the input vector, $h_t$ is the hidden layer, $x_t$ is the output vector, $W_h$, $U_h$, $W_y$, $b_h$ and $b_y$ are parameter matrices and vectors, $\sigma_h$ and $\sigma_y$ are activation functions.

If aligning the equations of ESKF (Equation 2.2, 2.3, 2.5, 2.6) to RNN (Equation 5.7) to draw their connections, $u_{k-1}$ and $z_k$ can be mapped to $x_t$, $x_k$ can be mapped to $y_t$, and $P_k$ can be mapped to $h_t$. The matrices to be tuned, including $Q$, $R_G$, $R_{Lp}$ and $R_{Ly}$, can be mapped to the parameter matrices. The other matrices and vectors (e.g., $G$ in Equation 2.4) can be considered as constant, i.e., not to be tuned. Since the ESKF used in AD system separates the processing pipelines of IMU, GNSS and LiDAR, as described in Section 2.2, we build three RNNs to emulate the three pipelines, as illustrated in Figure 5.3.

The simple RNN uses only one hidden layer. Through the initial exploration, we found the optimization result is unsatisfactory, in part of the higher complexity of ESKF model. For instance, $F$ in Equation 2.2 of KF is changed to $A$ in Baidu's implementation, which also depends on the previous state estimation. Hence, we choose to stack more layers to each RNN to capture such complex input-to-output relations. Stacking layers is a widely used technique for domains like NLP. For instance, stacked LSTM [35, 80] allows the hidden state to operate at different timescale.

In the end, we choose to stack 5, 2, 4 hidden layers for the pipelines of IMU, GNSS and LiDAR respectively. We double the number of layers for LiDAR comparing to GNSS because it takes two sub-phases to update LiDAR, for position state and pose state. When training $\hat{O}$, the output of the target ESKF, termed $x_{k,ref}$, will be compared to the output of $\hat{O}$, or $x_k$, under a loss function, termed $L$, to guide optimization. We use logarithmic of Mean Squared Error (MSE) as $L$, which can be written as Equation 5.8, where $N$ means the total number of output values, $x$ ($x_k \in x$) and $x_{ref}$ ($x_{k,ref} \in x_{ref}$) mean all output values.

$$L(x, x_{ref}) = \mathtt{log}(\frac{1}{N}\sum_{k=1}^{N}(x_k - x_{k,ref})^2) \tag{5.8}$$

Yet, we found emulating ESKF with RNN does not yield satisfactory result. The main reason is that, ESKF directly connects the prior state $x_{k-1}$ and current state $x_k$, but a RNN model does not have such direct connection. In Section **??**, we evaluate this RNN-based

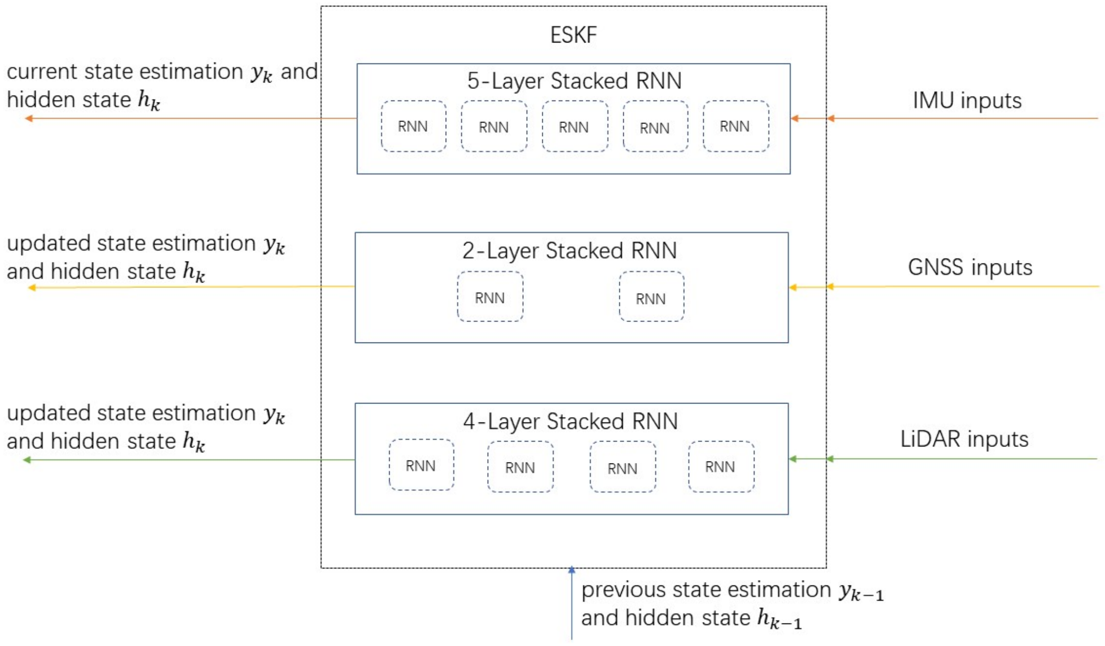
ESKF
current state estimation $y_k$ and hidden state $h_k$
5-Layer Stacked RNN
RNN
RNN
RNN
RNN
RNN
IMU inputs
updated state estimation $y_k$ and hidden state $h_k$
2-Layer Stacked RNN
RNN
RNN
GNSS inputs
updated state estimation $y_k$ and hidden state $h_k$
4-Layer Stacked RNN
RNN
RNN
RNN
RNN
LiDAR inputs
previous state estimation $y_{k-1}$ and hidden state $h_{k-1}$

**Figure 5.3:** The structure of ESKF emulated with RNN.

implementation and compare to the structure adopted by TaskMaster.

**Attack result.** However, it performs not well. The first reason for this poor performance is that **current RNN cell could not get direct access to the previous state (i.e. $y_{k-1}$) to perform the current prediction.** The formation of the current transition matrix $A$ directly depends on $q_{k-1}$, i.e. quaternion in the previous state estimation. However, calculation of the current state $y_k$ could only depends on the previous hidden state $h_{k-1}$ to get the previous state estimation. Though $h_{k-1}$ is the only external input to generate previous state estimation $y_{k-1}$, parameters $W_y$ and $b_y$ will affect the final estimation. Also, the estimation will be processed by an activation function, which will make the whole estimation process non-linear. However, the real process of generating the transition matrix $A$, as well as the whole prediction part of ESKF, does not involve in any non-linear process. This may lead to more parameters or more layers of RNN cells to make the emulate more accurate. However, since each state estimation of our ESKF module is very large (16 elements for each state estimation), an additional RNN cell stacked in our module will mean 3 more

$15 \times 15$ matrices ($W_h$, $U_h$ and $W_y$) and 2 more $15 \times 1$ vectors ($b_h$ and $b_y$) to be trained in our RNN-based module. More variables to train means more states to generate and larger time consumption during training period. Also, difficulty on making the module converged will be greatly increased with more variables.

What makes it worse is that generation of the current hidden state also involves an activation function. Together with the one in generation of the current state estimation, there are in total 3 non-linear processes involved for the current state estimation to get the previous state estimation ($y_{k-1} \rightarrow h_{k-1}$, $h_{k-1} \rightarrow h_k$ and $h_k \rightarrow y_k$). In conclusion, 1) formation of transition matrix $A$, 2) getting direct access to the previous state estimation from $h_{k-1}$ make training the RNN-based module for the prediction part very difficult.

**Modules for GNSS and LiDAR update part perform poorly as well.** Difficulty to get previous state estimation is also one reason for poor training performance of RNN-based GNSS and LiDAR update part. The other problem is that **lack of enough data** for RNN-based module training. As analyzed in Section 3.3, proportion for IMU, LiDAR and GNSS data is 50:6.5:1. Data is enough for prediction part training since large proportion data is for prediction part. For GNSS and LiDAR part, data only takes a small proportion, which means a module with too many parameters to train will be very difficult to converge. Theoretically, increase length of the trace for training (i.e. $N$ in Formula 5.8) may mitigate this problem. However, time consumption for training will be greatly raised with $N$ getting larger. Also, GPU memory will get overflowed if $N$ is set too large (>25000 in our experiment setting) during back propagation period. Lack of data also limits us to stack more RNN cells to make better emulate performance.

In conclusion, **inability of getting indirect access to previous state estimation**, **too many parameters for training** and **lack of GNSS and LiDAR** are 3 key problems for our failure in emulaing ESKF using RNN-based modules. In order to solve these 3 problems, we build our own ESKF module and emulating Baidu Apollo ESKF directly.

| Trace Name | PER of $Q$, SI | PER of $R_G$, SI | PER of $R_L^p$ and $R_L^y$, SI | PER of TaskMaster |
|---|---|---|---|---|
| *local08* | 0.01181 | 2.0984 | 5.1024 | 0.01347 |
| *highway17* | 0.01431 | 4.3518 | 5.3141 | 0.01297 |

**Table 5.8:** Comparison between System Identification (SI) and `Sola-ESKF` on *local08* and *highway17* in AS1.

## 5.6 Comparison to Other System Identification Methods

TaskMaster leveraged a novel optimization framework to find the best $Q$ and $R$. Here we compare the performance TaskMaster and the baseline system identification (SI) methods given the same training traces (*local08* and *highway17*). To save space, we only show the result (i.e., PER) when `Sola-ESKF` is used under AS1.

Specifically, we use the SI toolbox from Matlab [46], define the ESKF structure as a grey-box parametric model [45], and estimate $Q$ and $R$. Though we found there are other open-source and close-source tools to tune specific models, e.g., LTV Models [7] and Matlab Time-Varying MPC [68], none can be directly used to tune ESKF, because 1) ESKF are time varying, and 2) ESKF contains some nonlinear operations in both prediction and update modules. Building a customized SI baseline for ESKF tuning would take considerable human efforts, so we choose the generic Matlab's SI toolbox.

From Table 5.8, it can be seen that SI toolbox achieves similar results in estimating $Q$ compared to TaskMaster, but it performs much worse in $R$ (PERs of $R$ are all over 1 for SI toolbox while TaskMaster has less than 0.02 PER). We speculate the reason is that $Q$ has a simple relation to the input, as shown in Equation 2.2, but $R$ is used in the Update phase, which introduces many non-linear operations like pose transformation among quaternion, Euler angles and rotation matrices (see Appendix **??**). Previous works [25, 84] also reveal that classical SI methods could not achieve good results on non-linear models, and Section 8 elaborates the comparison.

## 5.7 Spoofing Attacks

For the attacks against ML models, learning the model parameters can facilitate the perturbation attacks that manipulate the model's classification result, as shown in [44]. Here, we are interested in whether the breach of ESKF confidentiality facilitates the attack against its integrity, similar to the above ML setting. To this end, we evaluate Fusionripper [90], a recently proposed spoofing attack against localization, with extracted models from TaskMaster. Fusionripper is an opportunistic GNSS spoofing method targeting ESKF-based MSF localization. Despite being effective at attacking MSF from a single sensor source (i.e., GNSS), Fusionripper requires a profiling stage to find the effective attack parameters, during which the attacker needs to *tailgate* victim's AV (or same model) on public roads, and perform trials of attack with different combinations of attack parameters, i.e., constant spoofing distance $d$ and scaling factor $f$. The profiling effort can be prominent (half a day as mentioned in [90]), and likely expose the attacker. Ideally, with TaskMaster, we can construct a *shadow* ESKF model, search for the best attack parameters on attacker's computer, and directly attack the on-road victim. As such, the costly profiling stage can be skipped.

Since the shadow ESKF model is an approximation of the targeted model, we are interested in whether Fusionripper on the shadow ESKF model can maintain the similar level of success rate as which on the targeted model. We have obtained the code and a trace (termed `ba-local`) from the authors of Fusionripper, and tested the cases when `Sola-ESKF` and `Apollo-ESKF` are the targeted model. We first apply the Fusionripper attack on the shadow ESKF by iterating over the same ranges of the attack parameters as used in [90]. We then calculate the attack success rates of all attack parameter combinations and obtain the attack parameter ($d$ and $f$) that can achieve the highest success rate. Next, we apply this attack parameter combination onto the target model.

In the end, we found the success rate is 100% on `Sola-ESKF` but only 7% on `Apollo-ESKF`.

The main reason why the success rate is low on `Apollo-ESKF` might be that the stolen model has not yet reached close enough performance to the ground-truth model. We believe by simulating other components surrounding `Apollo-ESKF`, we can significantly improve the result of this task.

# Chapter 6

# Discussion

## 6.1 Generalization to other KF models

AD may use variants of KF models in localization. For example, unscented Kalman filter and Extended Kalman filter (EKF) are good candidates as they work better with non-linear systems [32].

We believe TaskMaster can also be used to extract these variants, when the structure is approximately known by the attacker. As for the reasons, 1) these variants have similar structure with ESKF (for instance, EKF differs from ESKF only in the equations deriving $x_k$, $P_k$ and $K'$ [67]), and 2) these models are derivable. With the above reasons, the attacker can adopt a similar methodology of TaskMaster.

## 6.2 Limitations

1) We did not deploy the extracted ESKF models on real AV and evaluate them on the real roads, given we have no access to the AV testing and manufacturing process. Yet, we use the real-world traces (`KAIST`) and target ESKF models (`Apollo-ESKF`), and the result shows TaskMaster is effective. We are discussing with the AV vendors to obtain their feedback to our study, as an indicator of the attack practicality. 2) When the AD of the attacker uses a set of sensors with *very different* settings as the targeted AD, the stolen parameters cannot be directly used. However, as described in Section 3.1, industry-grade models like `Apollo-ESKF` is able to achieve good performance on different AVs even without re-tuning. Hence, we believe stealing such models should be useful to attackers in most cases. 3) TaskMaster is less effective against `Apollo-ESKF` comparing to `Sola-ESKF`. We believe the main reason is that `Apollo-ESKF` is more complex than `Sola-ESKF`, and we are unable to emulate all components surrounding `Apollo-ESKF`. 4) Except Baidu Apollo, we have not found another AD system to verify if obfuscation is applied on the localization module. Though Autoware [5] is another popular open-source AD system, we found it only uses LiDAR for localization by default. Though extracting binaries from an AV can help us get another ground-truth MSF, it is impossible without vulnerability exploitation. In the meantime, we will keep inquiring the AD community. 5) We evaluated TaskMaster against Baidu Apollo v2.5 while the latest version is v6.0. The ESKF version is upgraded from v1.0.3 to v1.0.4. We plan to test the latest version, but we expect the changes to be small.

## 6.3 Defense

Though there are a number of extraction attacks against ML models, recent works show defenses are possible. An example is PRADA [58], which analyses the distribution of client's queries and detect the ones that deviate the normal ones. However, these works assume MLaaS (Machine-learning as a Service) settings, where the queries can be audited. This is very difficult in our setting, as attacker's observation cannot be blocked under AS3, and there is no need to actively query ESKF under AS1 and AS2.

A partial solution, which is practical under AS1 and AS2, could be mediating the access to ECUs. Specifically, AV vendors may add "self-destruction" modules to ECUs. Once an attacker tries to break the ECU to get the output of ESKF, the ECUs can wipe out the parameters of ESKF. Additionally, AV vendors can encrypt all the messages over CAN bus.

## 6.4 Future works

Moving beyond localization models, one interesting yet unsolved research problem is, *is control-theory models that are derivable generally vulnerable to model extraction attacks?* Given there are many such models operated in the safety-critical settings (e.g., drones), gaining insights into this problem is important. One research path is to examine whether the existing model extraction attacks against ML models can be *transferred* to this new setting.

The outcome of model extraction attacks against ML models can help the attacks who aim to break the *integrity* of ML models [78] (e.g., test-time evasion attacks). Such analogy could be drawn on the localization models. Section 5.7 makes a preliminary attempt against MSF and we will futher explore this direction.

# Chapter 7

# Conclusion

In this thesis, we systematically studied the confidentiality issues underlying the AD control models, in particular ESKF model used for localization, which has been considered as an intellectual property by AD companies. We designed TaskMaster, a novel optimization-based framework to infer the secret parameters by observing the input and output of an AD. Under 3 practical adversarial settings, we found TaskMaster can achieve very high accuracy for `Sola-ESKF` and comparable accuracy for a complex, industry-grade model `Apollo-ESKF`. We also demonstrated that the classical SI-based methods cannot achieve the similar performance, and TaskMaster can facilitate the attack against the integrity of the localization model. As the first study on the AD model confidentiality, we hope our findings can attract attention from AD industries and security community in addressing this new threat.

# Chapter 8

# Related Work

## 8.1 Security of AD

The research into the security of modern vehicles has been started a decade ago [20, 61, 81, 21], and the attack surface on wireless protocols, CAN bus, etc. was explored. The security of AD has attracted attention from the research community only recently. One direction is to study the sensors leveraged by AD, including LiDAR, IMU, perception, etc. [17, 94, 102, 13, 29, 65, 55, 112, 16, 56, 107], and defense based on physical invariants was proposed [83]. Attacks against the traditional computing architecture, like cache side-channel attacks [66] and malware attacks [53], were examined and found feasible against the software stack of AD. Regarding the security of MSF, only Shen et al. [90] demonstrated its integrity can be tampered, while TaskMaster looks into the confidentiality issues of MSF models.

## 8.2 Model Extraction

Similar to KF models, the parameters of machine-learning models, including the classic models like logistic regression, and DNN, can be considered as secret. Given that a lot of deployed models offer cloud-based API access, an adversary can issue queries and use the detailed responses (e.g., confidence score) to guess the model parameters. As the first step, Tramer et al. proposed equation-solving attack and path-finding attack [101]. Wang et al. studied how the model hyper-parameters used to balance between the loss function and the regularization terms can be stolen [108]. Juuti et al. improved on the existing attacks by generating synthetic queries and proposed defenses [58]. The attack precision is further improved with semi-supervised learning [50] and active learning [19]. While prior works focused on CNN when attacking DNN, recently, attack against RNN was demonstrated feasible [96]. But as described in Section 3.3, successful model extraction against KF against has to overcome a few new challenges, which are addressed by the new design of TaskMaster. Our work also broadens the scope of model extraction attacks from machine-learning models to control-theory models.

## 8.3 System Identification

System identification (SI) [64] builds mathematical model for a dynamic system with statistical analysis. The related methods can be divided into 4 categories [73], but we found directly using SI to steal ESKF parameters does yield satisfactory results. 1) The Bayesian Method treats parameter update as a Bayesian update [1], but a Bayesian optimal estimation might be unrealizable [39]. 2) Maximum Likelihood carries out non-linear, gradient-based optimization to minimize the difference between model prediction and measurement [12, 1, 111], and Expectation Maximization [91, 10] attempts to avoid the non-linear optimization. However, the optimization process is time-consuming. 3) Covariance Matching runs Monte Carlo simulation and checks if the sampled statistics are internally consistent [75, 33, 72, 10]. Though faster, Covariance Matching leads to sub-optimal result. 4) Correlation Techniques assume the sequence of prediction error is zero-mean white Gaussian noise when the model is optimal, and tune the control parameters towards this criteria [69, 70, 77]. However, this assumption does not always hold. Some works have applied SI on simple KF models [57, 11, 26], but none of them are applicable to the MSF models adopted by AVs, especially ESKF. In fact, ESKF is an LTV (linear time-variant) system while KF is an LTI (linear time-invariant) system, and many properties from LTI systems do not hold in LTV systems. Recently, deep-learning based models have been used for SI [71, 2] but again none of them work on ESKF.

## 8.4 Kalman Filter Tuning

There are two kinds of methods in Kalman Filter tuning: 1) tuning by brutal force [6], 2) tune target parameters iteratively with the mathematical properties of Kalman Filter. Efficiency for traditional brutal force tuning is quite low, and results are not satisfactory (RMSE=1.5), compared with the results of TaskMaster. The only advantage of brutal force tuning compared with TaskMaster is that it does not require any preliminary knowledge of the given model. The tuning process is completely based on random selection in parameter space, respectively. This is also the reason why it could achieve precise and stable tuning results.

The mathematics-based Kalman Filter tuning improves its tuning accuracy and stability to some extent (RMSE<1), compared with brutal force tuning. It takes half as much time as what brutal force tuning takes after optimization on algorithm and compilation levels. However, its accuracy and time efficiency is still lower than TaskMaster proposed in this work. What is more, mathematical properties will vary when MSF modules change, which reduces its generality. Therefore, attackers need to re-evaluate mathematical properties of the MSF module under a specific circumstance, e.g., ESKF in Baidu Apollo AV system, and re-implement the whole tuning procedure. In conclusion, mathematics-based tuning approaches are limited in real-world tuning.

# Bibliography

[1] D. Alspach. A parallel filtering algorithm for linear systems with unknown time varying noise statistics. *IEEE Transactions on Automatic Control*, 19(5):552–556, 1974.

[2] B. P. Amiruddin, E. Iskandar, A. Fatoni, and A. Santoso. Deep learning based system identification of quadcopter unmanned aerial vehicle. In *2020 3rd International Conference on Information and Communications Technology (ICOIACT)*, pages 165–169. IEEE, 2020.

[3] ars technica. Google's waymo invests in lidar technology, cuts costs by 90 percent. `https://arstechnica.com/cars/2017/01/googles-waymo-invests-in-lidar-technology-cuts-costs-by-90-percent/`, 2017.

[4] E. Artin. *The gamma function*. Courier Dover Publications, 2015.

[5] Autoware. Autoware-ai/autoware.ai: Open-source software for self-driving vehicles. `https://github.com/Autoware-AI/autoware.ai`, 2020.

[6] awerries. kalman-localization. `https://github.com/awerries/kalman-localization`, 2022.

[7] F. Bagge Carlson. Machine learning and system identification for estimation in physical systems, 12 2018.

[8] Baidu. Apolloauto/apollo: An open autonomous driving platform. `https://github.com/ApolloAuto/apollo`, 2020.

[9] Baidu. Baidu apollo controller module. `https://github.com/ApolloAuto/apollo/tree/r2.5.0/modules/control`, 2020.

[10] V. A. Bavdekar, A. P. Deshpande, and S. C. Patwardhan. Identification of process and measurement noise covariance for state and parameter estimation using extended kalman filter. *Journal of Process control*, 21(4):585–601, 2011.

[11] M. C. Best, A. P. Newton, and S. Tuplin. The identifying extended kalman filter: parametric system identification of a vehicle handling model. *Proceedings of the Institution of Mechanical Engineers, Part K: Journal of Multi-body Dynamics*, 221(1):87–98, 2007.

[12] T. Bohlin. Four cases of identification of changing systems. In *Mathematics in Science and Engineering*, volume 126, pages 441–518. Elsevier, 1976.

[13] A. Boloor, K. Garimella, X. He, C. Gill, Y. Vorobeychik, and X. Zhang. Attacking vision-based perception in end-to-end autonomous driving models. *Journal of Systems Architecture*, page 101766, 2020.

[14] A. K. Boyat and B. K. Joshi. A review paper: noise models in digital image processing. *arXiv preprint arXiv:1505.03489*, 2015.

[15] Business Line. Velodyne lidar awarded perception system contract from mercedes-benz. `https://www.thehindubusinessline.com/business-wire/velodyne-lidar-awarded-perception-system-contract-from-mercedesbenz/article9856916.ece`, 2018.

[16] Y. Cao, N. Wang, C. Xiao, D. Yang, J. Fang, R. Yang, Q. A. Chen, M. Liu, and B. Li. Invisible for both camera and lidar: Security of multi-sensor fusion based perception in autonomous driving under physical-world attacks. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 176–194. IEEE, 2021.

[17] Y. Cao, C. Xiao, B. Cyr, Y. Zhou, W. Park, S. Rampazzi, Q. A. Chen, K. Fu, and Z. M. Mao. Adversarial sensor attack on lidar-based perception in autonomous driving. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2267–2281, 2019.

[18] CBINSIGHTS. Android of the auto industry? how baidu may race ahead of google, tesla, and others in autonomous vehicles. `https://www.cbinsights.com/research/baidu-china-autonomous-vehicles/`, 2018.

[19] V. Chandrasekaran, K. Chaudhuri, I. Giacomelli, S. Jha, and S. Yan. Exploring connections between active learning and model extraction. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pages 1309–1326, 2020.

[20] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*, volume 4, pages 447–462. San Francisco, 2011.

[21] K.-T. Cho and K. G. Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 911–927, 2016.

[22] CISOMAG. Tesla offers us$1 million and a car as bug bounty reward. `https://cisomag.eccouncil.org/tesla-offers-us1-million-and-a-car-as-bug-bounty-reward/`, 2020.

[23] DARPA. The darpa grand challenge: Ten years later. `https://www.darpa.mil/news-events/2014-03-13`, 2014.

[24] M. DeBord. Waymo has launched its commercial self-driving service in phoenix - and it's called 'waymo one'. `https://www:businessinsider:com/waymo-one-driverless-car-servicelaunches-in-phoenix-arizona-2018-12`, 2018.

[25] D. Di Ruscio. Subspace System Identification of the Kalman Filter. *Modeling, Identification and Control*, 24(3):125–157, 2003.

[26] D. Di Ruscio. Subspace system identification of the kalman filter. 2003.

[27] J. L. Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[28] ETH Zürich. Ethzasl msf framework. `https://github.com/ethz-asl/ethzasl_msf`, 2018.

[29] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1625–1634, 2018.

[30] Frida. Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers. `https://frida.re/`, 2022.

[31] B. Friedland. *Control system design: an introduction to state-space methods*. Courier Corporation, 2012.

[32] Q. B. Ge, W. B. Li, R. Y. Sun, and Z. Xu. Centralized fusion algorithms based on ekf for multisensor non-linear systems. *Zidonghua Xuebao/Acta Automatica Sinica*, 39(6):816–825, 2013.

[33] R. Gemson. Estimation of aircraft aerodynamic derivatives accounting for measurement and process noise by ekf through adaptive filter tuning. *Bangalore, India: Department of Aerospace Engineering, Indian Institute of Science*, 1991.

[34] C. Goodin, D. Carruth, M. Doude, and C. Hudson. Predicting the influence of rain on lidar in adas. *Electronics*, 8(1):89, 2019.

[35] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[36] GreenValley International. Libackpack - mobile handheld lidar - 3d mapping system. `https://greenvalleyintl.com/hardware/libackpack/`, 2020.

[37] Hex Rays. Ida pro. `https://hex-rays.com/ida-pro/`, 2022.

[38] HEXAGON. Easymile and velodyne lidar announce three-year agreement. `https://insideunmannedsystems.com/easymile-and-velodyne-lidar-announce-three-year-agreement/`, 2020.

[39] C. G. Hilborn and D. G. Lainiotis. Optimal estimation in the presence of unknown parameters. *IEEE Transactions on Systems Science and Cybernetics*, 5(1):38–43, 1969.

[40] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun. Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing. In *2007 American control conference*, pages 2296–2301. IEEE, 2007.

[41] B. Hofmann-Wellenhof, H. Lichtenegger, and E. Wasle. *GNSS–global navigation satellite systems: GPS, GLONASS, Galileo, and more*. Springer Science & Business Media, 2007.

[42] T. N. N. Hossein, S. Mita, and H. Long. Multi-sensor data fusion for autonomous vehicle navigation through adaptive particle filter. In *2010 IEEE Intelligent Vehicles Symposium*, pages 752–759. IEEE, 2010.

[43] S. C. HPL. Introduction to the controller area network (can). *Application Report SLOA101*, pages 1–17, 2002.

[44] X. Hu, L. Liang, S. Li, L. Deng, P. Zuo, Y. Ji, X. Xie, Y. Ding, C. Liu, T. Sherwood, et al. Deepsniffer: A dnn model extraction framework based on learning architectural hints. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 385–399, 2020.

[45] T. M. Inc. *Nonlinear grey-box model - MATLAB idnlgrey.* Natick, Massachusetts, United State, 2021.

[46] T. M. Inc. *System Identification Toolbox User's Guide.* Natick, Massachusetts, United State, 2021.

[47] Informed Infrastructure. Velodyne's lidar division announces agreement with caterpillar for laser imaging technology. `https://informedinfrastructure.com/25630/velodynes-lidar-division-announces-agreement-with-caterpillar-for-laser-imaging-technology-2/`, 2012.

[48] Inside Unmanned Systems. Easymile and velodyne lidar announce three-year agreement. `https://insideunmannedsystems.com/easymile-and-velodyne-lidar-announce-three-year-agreement/`, 2020.

[49] Intel. Pin - a dynamic binary instrumentation tool. `https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-dynamic-binary-instrumentation-tool.html`, 2022.

[50] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot. High accuracy and high fidelity extraction of neural networks. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020.

[51] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot. High Accuracy and High Fidelity Extraction of Neural Networks . In *Proceedings of the 29th USENIX Security Symposium (USENIX Security '20)*, Boston, MA, August 2020.

[52] J. Jeong, Y. Cho, Y.-S. Shin, H. Roh, and A. Kim. Complex urban dataset with multi-level sensors from highly diverse urban environments. *The International Journal of Robotics Research*, page 0278364919843996, 2019.

[53] S. Jha, S. Cui, S. S. Banerjee, J. Cyriac, T. Tsai, Z. Kalbarczyk, and R. K. Iyer. Ml-driven malware that targets AV safety. In *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2020, Valencia, Spain, June 29 - July 2, 2020*, pages 113–124. IEEE, 2020.

[54] S. Jha, S. Cui, S. S. Banerjee, T. Tsai, Z. Kalbarczyk, and R. Iyer. Ml-driven malware that targets av safety. *arXiv preprint arXiv:2004.13004*, 2020.

[55] Y. Jia, Y. Lu, J. Shen, Q. A. Chen, H. Chen, Z. Zhong, and T. Wei. Fooling detection alone is not enough: Adversarial attack against multiple object tracking. In *International Conference on Learning Representations*, 2019.

[56] P. Jing, Q. Tang, Y. Du, L. Xue, X. Luo, T. Wang, S. Nie, and S. Wu. Too good to be safe: Tricking lane detection in autonomous driving with crafted perturbations. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3237–3254, 2021.

[57] J.-N. Juang, M. Phan, L. G. Horta, and R. W. Longman. Identification of observer/kalman filter markov parameters-theory and experiments. *Journal of Guidance, Control, and Dynamics*, 16(2):320–329, 1993.

[58] M. Juuti, S. Szyller, S. Marchal, and N. Asokan. Prada: protecting against dnn model stealing attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 512–527. IEEE, 2019.

[59] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi. Autoware On Board: Enabling Autonomous Vehicles with Embedded Systems. In *ICCPS'18*, pages 287–296. IEEE Press, 2018.

[60] E. Khartchenko. Vectorization: A key tool to improve performance on modern cpus. `https://software.intel.com/content/www/us/en/develop/articles/vectorization-a-key-tool-to-improve-performance-on-modern-cpus.html`, 2018.

[61] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, et al. Experimental security analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy*, pages 447–462. IEEE, 2010.

[62] S. Lau. Learning rate schedules and adaptive learning rate methods for deep learning. `https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1`, 2017.

[63] S. L. Lauritzen. Time series analysis in 1880: A discussion of contributions made by tn thiele. *International Statistical Review/Revue Internationale de Statistique*, pages 319–331, 1981.

[64] L. Ljung. System identification. *Wiley encyclopedia of electrical and electronics engineering*, pages 1–19, 1999.

[65] J. Lu, H. Sibai, and E. Fabry. Adversarial examples that fool detectors. *arXiv preprint arXiv:1712.02494*, 2017.

[66] M. Luo, A. C. Myers, and G. E. Suh. Stealthy tracking of autonomous vehicles with cache side channels. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pages 859–876, 2020.

[67] S. Lynen, M. W. Achtelik, S. Weiss, M. Chli, and R. Siegwart. A robust and modular multi-sensor fusion approach applied to mav navigation. In *2013 IEEE/RSJ international conference on intelligent robots and systems*, pages 3923–3929. IEEE, 2013.

[68] Mathworks. Time-varying mpc. `https://www.mathworks.com/help/mpc/ug/time-varying-mpc.html`, 2022.

[69] R. Mehra. On the identification of variances and adaptive kalman filtering. *IEEE Transactions on automatic control*, 15(2):175–184, 1970.

[70] R. Mehra. Approaches to adaptive filtering. *IEEE Transactions on automatic control*, 17(5):693–698, 1972.

[71] S. S. Miriyala and K. Mitra. Deep learning based system identification of industrial integrated grinding circuits. *Powder Technology*, 360:921–936, 2020.

[72] A. Mohamed and K. Schwarz. Adaptive kalman filtering for ins/gps. *Journal of geodesy*, 73(4):193–203, 1999.

[73] S. Mohan M, N. Naik, R. Gemson, and M. Ananthasayanam. Introduction to the kalman filter and tuning its statistics for near optimal estimates and cramer rao bound. *arXiv*, pages arXiv–1503, 2015.

[74] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[75] K. Myers and B. Tapley. Adaptive sequential estimation with unknown noise statistics. *IEEE Transactions on Automatic Control*, 21(4):520–523, 1976.

[76] S. Nie, L. Liu, and Y. Du. Free-fall: Hacking tesla from wireless to can bus. *Briefing, Black Hat USA*, 25:1–16, 2017.

[77] B. J. Odelson, A. Lutz, and J. B. Rawlings. The autocovariance least-squares method for estimating covariances: application to model-based control of chemical reactors. *IEEE transactions on control systems technology*, 14(3):532–540, 2006.

[78] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman. Sok: Security and privacy in machine learning. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 399–414. IEEE, 2018.

[79] K. I. Park. *Fundamentals of Probability and Stochastic Processes with Applications to Communications*. Springer Publishing Company, Incorporated, 1st edition, 2017.

[80] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.

[81] J. Petit and S. E. Shladover. Potential cyberattacks on automated vehicles. *IEEE Transactions on Intelligent transportation systems*, 16(2):546–556, 2014.

[82] S. C. M. Post. Chinese internet giant baidu offers free trial robotaxi rides through search and map apps in changsha. `https://www.scmp.com/tech/apps-social/article/3080712/chinese-internet-giant-baidu-offers-free-trial-robotaxi-rides`, 2020.

[83] R. Quinonez, J. Giraldo, L. Salazar, E. Bauman, A. Cardenas, and Z. Lin. {SAVIOR}: Securing autonomous vehicles with robust physical invariants. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pages 895–912, 2020.

[84] M. J. u. Rehman, S. C. Dass, and V. S. Asirvadam. Nonlinear dynamical system identification using unscented kalman filter. In *AIP Conference Proceedings*, volume 1787, page 020003. AIP Publishing LLC, 2016.

[85] T. G. Reid, S. E. Houts, R. Cammarata, G. Mills, S. Agarwal, A. Vora, and G. Pandey. Localization requirements for autonomous vehicles. *arXiv preprint arXiv:1906.01061*, 2019.

[86] Renovo.auto Blog. Renovo selects velodyne as reference lidar provider for advanced automotive development projects. `https://medium.com/renovo-auto-blog/renovo-selects-velodyne-as-reference-lidar-provider-for-aware-automated-mobility-operating-system-f8001b91c6c`, 2017.

[87] Reuters. Ford dissolves its 7.6% stake in velodyne lidar. `https://www.reuters.com/article/us-velodyne-lidar-stake/ford-dissolves-its-7-6-stake-in-velodyne-lidar-idUSKBN2AF1B7?il=0`, 2021.

[88] D. E. Rivera, M. Morari, and S. Skogestad. Internal model control: Pid controller design. *Industrial & engineering chemistry process design and development*, 25(1):252–265, 1986.

[89] A. Sakai, D. Ingram, J. Dinius, K. Chawla, A. Raffin, and A. Paques. Pythonrobotics: a python code collection of robotics algorithms. *CoRR*, abs/1808.10703, 2018.

[90] J. Shen, J. Y. Won, Z. Chen, and Q. A. Chen. Drift with Devil: Security of Multi-Sensor Fusion based Localization in High-Level Autonomous Driving under GPS Spoofing. In *Proceedings of the 29th USENIX Security Symposium (USENIX Security '20)*, Boston, MA, August 2020.

[91] R. H. Shumway and D. S. Stoffer. *Time series analysis and its applications: with R examples*. Springer, 2017.

[92] J. Sola. Quaternion kinematics for the error-state kalman filter. *arXiv preprint arXiv:1711.02508*, 2017.

[93] J. K. Suhr, J. Jang, D. Min, and H. G. Jung. Sensor fusion-based low-cost vehicle localization system for complex urban environments. *IEEE Transactions on Intelligent Transportation Systems*, 18(5):1078–1086, 2016.

[94] J. Sun, Y. Cao, Q. A. Chen, and Z. M. Mao. Towards robust lidar-based perception in autonomous driving: General black-box adversarial sensor attack and countermeasures. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pages 877–894, 2020.

[95] System Plus Consulting. Bosch's 6-axis imu in the apple iphone x. `https://www.systemplus.fr/reverse-costing-reports/boschs-6-axis-imu-in-the-apple-iphone-x/`, 2018.

[96] T. Takemura, N. Yanai, and T. Fujiwara. Model extraction attacks against recurrent neural networks. *arXiv preprint arXiv:2002.00123*, 2020.

[97] Tencent Keen Security Lab. Exploiting wi-fi stack on tesla model s. `https://keenlab.tencent.com/en/2020/01/02/exploiting-wifi-stack-on-tesla-model-s/`, 2020.

[98] Texas Instruments. 3.3-v can transceivers. `https://www.ti.com/cn/lit/ds/slos346h/slos346h.pdf`, 2006.

[99] N. O. Tippenhauer, C. Pöpper, K. B. Rasmussen, and S. Capkun. On the requirements for successful gps spoofing attacks. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 75–86, 2011.

[100] Trail of Bits. Mcsema. `https://github.com/lifting-bits/mcsema`, 2022.

[101] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction apis. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 601–618, 2016.

[102] Y. Tu, Z. Lin, I. Lee, and X. Hei. Injected and delivered: Fabricating implicit control over actuation systems by spoofing inertial sensors. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1545–1562, 2018.

[103] United States Securities and Exchange Commission. Schedule 13g under the securities exchange act of 1934 (amendment no. 1) – velodyne lidar, inc. `https://www.reuters.com/article/us-velodyne-lidar-stake/ford-dissolves-its-7-6-stake-in-velodyne-lidar-idUSKBN2AF1B7?il=0`, 2020.

[104] Unmanned Systems Technology. Velodyne lidar partners with nikon for autonomous vision. `https://www.unmannedsystemstechnology.com/2018/12/velodyne-lidar-partners-with-nikon-for-autonomous-vision/`, 2018.

[105] Velodyne. Velodyne lidar: Envision the future. `https://velodynelidar.com/`, 2022.

[106] G. Wan, X. Yang, R. Cai, H. Li, Y. Zhou, H. Wang, and S. Song. Robust and precise vehicle localization based on multi-sensor fusion in diverse city scenes. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4670–4677. IEEE, 2018.

[107] Z. Wan, J. Shen, J. Chuang, X. Xia, J. Garcia, J. Ma, and Q. A. Chen. Too Afraid to Drive: Systematic Discovery of Semantic DoS Vulnerability in Autonomous Driving Planning under Physical-World Attacks. In *Network and Distributed System Security (NDSS) Symposium, 2022*, April 2022.

[108] B. Wang and N. Z. Gong. Stealing hyperparameters in machine learning. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 36–52. IEEE, 2018.

[109] G. Welch, G. Bishop, et al. An introduction to the kalman filter. 1995.

[110] yegord. Snowman. `https://github.com/yegord/snowman`, 2022.

[111] M. A. Zagrobelny and J. B. Rawlings. Identification of disturbance covariances using maximum likelihood estimation. *Technical report, No. 2014–02*, 2014.

[112] Y. Zhao, H. Zhu, R. Liang, Q. Shen, S. Zhang, and K. Chen. Seeing isn't believing: Towards more robust adversarial attack against real world object detectors. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1989–2004, 2019.