Digital Communications and Networks 8 (2022) 791–803

Contents lists available at ScienceDirect

Digital Communications and Networks

journal homepage: www.keaipublishing.com/dcan

DeepPOSE: Detecting GPS spoofing attack via deep recurrent neural network

Peng Jiang, Hongyi Wu, Chunsheng Xin [*]

ECE Dept. and School of Cybersecurity, Old Dominion University, Norfolk, VA, 23529, USA

A R T I C L E I N F O

Keywords:
GPS spoofing attack
Position estimation
Recurrent neural network

A B S T R A C T

The Global Positioning System (GPS) has become a foundation for most location-based services and navigation systems, such as autonomous vehicles, drones, ships, and wearable devices. However, it is a challenge to verify if the reported geographic locations are valid due to various GPS spoofing tools. Pervasive tools, such as Fake GPS, Lockito, and software-defined radio, enable ordinary users to hijack and report fake GPS coordinates and cheat the monitoring server without being detected. Furthermore, it is also a challenge to get accurate sensor readings on mobile devices because of the high noise level introduced by commercial motion sensors. To this end, we propose DeepPOSE, a deep learning model, to address the noise introduced in sensor readings and detect GPS spoofing attacks on mobile platforms. Our design uses a convolutional and recurrent neural network to reduce the noise, to recover a vehicle's real-time trajectory from multiple sensor inputs. We further propose a novel scheme to map the constructed trajectory from sensor readings onto the Google map, to smartly eliminate the accumulation of errors on the trajectory estimation. The reconstructed trajectory from sensors is then used to detect the GPS spoofing attack. Compared with the existing method, the proposed approach demonstrates a significantly higher degree of accuracy for detecting GPS spoofing attacks.

1. Introduction

The use of GPS services has surged in recent years. The GPS tracking device market is currently worth 1.57 billion USD and expects to reach 3.38 billion by 2025 [1]. The ability to acquire the real-time location of a moving object is crucial to safety in many applications. For example, the navigation system in an autonomous vehicle acquires GPS signals to calculate the instant longitude, latitude, speed, and course to help a car to reach its destination. However, the vigorous development of GPS-enabled devices and the low-cost spoofing devices has stimulated malicious users or attackers to initiate GPS attacks. Due to the open nature of the civilian GPS signal structure and ubiquitousness of unencrypted GPS signals [2], it is easy for an attacker to launch a GPS spoofing attack from a portable programmable off-the-shelf radio device such as HackRF, or USRP in a distance where the radio transmitter can interfere with the legitimate GPS signals [3–6]. Once the attacker takes over the GPS signals in a certain area, the navigation system running on the target vehicle will be fooled into following a wrong route crafted by the attacker. Researchers have demonstrated that it is possible to change the course of a Tesla or simply force it to drive off-road by using a HackRF while the vehicle is driving in an autopilot mode [7]. Similarly, attackers can also control a drone or a vehicle to an unsafe area [8–11] or rob an unwitting Pokémon Go player while following the navigation to nearby Pokestops at isolated locations.

Fig. 1 demonstrates a life-threatening GPS spoofing attack against an autonomous vehicle. The target vehicle (marked in black) is following a planned route to approach the destination on the right. Before the vehicle reaches the intersection, a spoofer sends the crafted GPS signals by using a directional antenna. As the signal strength of spoofed signals is stronger than the strength of the legitimate signal received from the satellites, the vehicle will recalculate its current location based on the crafted signals from the spoofer and is thus fooled to place itself to the spoofed position (the left side of the road). Next, the navigation system will recalculate a new route instantly to reach the original destination. On the new route, the vehicle goes straight instead of turning right at an intersection. Thus, the vehicle will deviate from the road.

In addition to navigation, GPS data has been widely used by many other applications and services to improve services and user experiences. For example, ride-hailing services such as Uber and Lyft use GPS to track both drivers and passengers to calculate correct fares and ensure the safety of both. The GPS spoofing attack raises a critical concern for such services.

* Corresponding author.
E-mail addresses: pjiang@odu.edu (P. Jiang), h1wu@odu.edu (H. Wu), cxin@odu.edu (C. Xin).

https://doi.org/10.1016/j.dcan.2021.09.006
Received 16 September 2020; Received in revised form 1 September 2021; Accepted 13 September 2021
Available online 17 September 2021

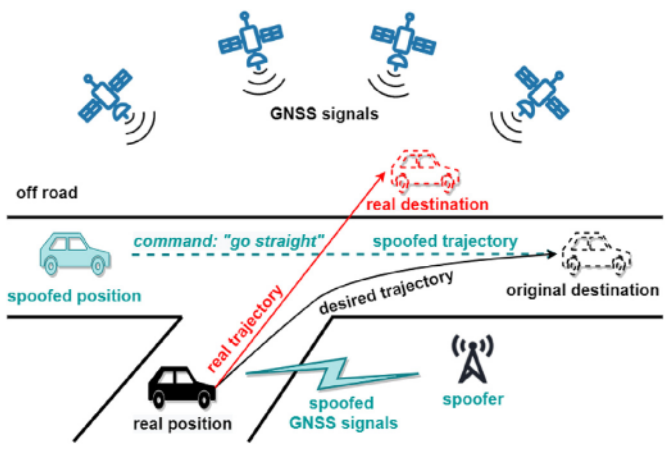


Fig. 1. A demonstration of the GPS spoofing attack against the autonomous vehicle.

In this research, we propose DeepPOSE, a deep learning model that utilizes motion sensors on mobile platforms to estimate vehicle positions and further detect GPS spoofing attacks. Note that in addition to GPS data, the motion sensor data is also often collected by services and applications. For example, Uber and Lyft use accelerometer and gyroscope data from users to monitor irregular activities, such as an unexpected long stop or a car crash. DeepPOSE is composed of two components, a *vehicle position estimator* and a *GPS spoofing detector*. In order to estimate the vehicle position from the noisy motion sensors readings, DeepPOSE combines convolutional neural network and sequence-to-sequence neural networks, which is a variant of the recurrent neural network that converts sequences from one domain into another [12,13]. In our design, we take the multidimensional sensor measurements as the source sequence, and the vehicle states, including the speed and direction, as the target sequence.

To detect GPS spoofing, we propose lightweight and efficient algorithms that detect the GPS spoofing attack by recovering the real-time trajectory from sensor data and comparing it with the trajectory reconstructed by the GPS signals. It aligns the sensor data with a street map to significantly reduce the error accumulation of trajectory estimation from the sensor data. This results in a higher performance of spoofing detection.

In summary, this paper makes the following contributions.

- We design an effective scheme for vehicle position estimation based on noisy motion sensor data using the convolutional neural network and sequence-to-sequence neural networks.
- We devise a novel algorithm to reconstruct vehicle trajectory from noisy motion sensor data with a map alignment scheme. This reduces the error accumulation in trajectory estimation and achieves effective GPS spoofing detection.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 explains the threat models. Section 4 describes the overall architecture of DeepPOSE. We present design details of the two components of DeepPOSE, the vehicle position estimator and spoofing detector, in Sections 5 and 6. The performance of DeepPOSE is evaluated in Section 7. Finally, we conclude the paper in Section 8.

2. Related work

For GPS spoofing attacks, the attacker's motivation can be manifold. An attacker may intend to acquire the valuable goods on board or hijack the important person sitting in an autonomous vehicle or cause chaos by jamming the GPS signal in a particular area and resulting in massive GPS failures. An evil Uber or Lyft driver can also report a *plausible* route to the application server to escape the tracking of the current vehicle from the monitoring center.

2.1. GPS spoofing attack

The open-source GPS signal generator *gps-sdr-sim* [14] makes it possible to craft the GPS signals for a set of fake trajectories or navigation routes against various targets by using a device that costs less than $300 [9]. Based on the requirements for a successful GPS spoofing attack, an attacker can successfully spoof the navigation system [4,15,16], unmanned aircraft [5,17], or ships [6], and control the target entities to follow a different route and reach as far away as possible from the intended locations. Most of the above spoofing attacks focus on delivering the target on the pre-determined route.

2.2. GPS spoofing detection

A considerable number of studies on countermeasures against GPS spoofing attacks exist in the literature. The authors of [18] utilized the Doppler shift of the GPS signals and the local clock to detect the existence of GPS spoofing. Similar to the encryption mechanism applied to the military GPS signal reception, cryptographic techniques were proposed in Refs. [19,20] to defend against the GPS spoofing attack in the civilian satellite system. However, adopting an encryption mechanism requires significant changes to the current GPS architecture; hence it is not practical to be implemented. Even if this is possible, it is also extremely hard to upgrade all the GPS transponders. To address this issue, the authors in Ref. [21] proposed a detection method called Crowd-GPS-Sec, which deploys a global scale message monitoring system to localize the source of the GPS spoofing attack against the moving airborne targets by analyzing the *Time Difference of Arrival* (TDoA) of position advertisements observed on different sensors. However, this detection mechanism relies on the support of the huge number of sensors deployed across the world and cannot be effectively deployed on commercial mobile platforms. There was one study on GPS spoofing detection and mitigation for UAVs [22]. This solution, nevertheless, requires the existence and assistance of nearby unspoofed UAVs.

The inertial sensors navigation system shows its robustness and resilience to defend against wireless signal spoofing and jamming attacks. It is more flexible to be implemented on the mobile platform to defend against the GPS spoofing attack [23–25]. However, the accumulated error degrades the navigation capability of the mobile devices significantly and makes the inertial sensor unable to reliably estimate the vehicle position for a long time.

DeepPOSE utilizes deep learning techniques to reconstruct the vehicle trajectory effectively on the basis of noisy motion sensor measurements. In recent years, deep learning has been widely used in processing tasks with noisy multi-dimensional input due to its superior capability in modeling spatial and temporal relationships with a flexible combination of structures. ConvLSTM [26] is the first model proposed to extend the convolution structure inside the recurrent neural network to capture the spatiotemporal correlations for precipitation based on the sequenced input data. DeepSense [27] and DeepMove [28] are inspiring works that apply deep learning to the mobile sensing applications for both regression and classification problems, such as vehicle position tracking and human activity recognition. However, both approaches overlook the long-term dependency in both input and output sequences, which is critical for spatial-temporal modeling.

3. Threat model

As discussed in the introduction, we consider the following threat model with the presence of a malicious user (Eve) who can either manipulate the GPS signal on his own device or broadcast the crafted GPS signal using the open-source software and an external antenna to fool a nearby device [14].

Case 1. Eve is an evil driver of ride-hailing services such as Uber and Lyft, capable of manipulating the nearby devices' GPS receptions before reporting their real-time location to the ride-hailing monitoring center. The objective of Eve is to take the passenger, Alice, to an unreported route without raising the alarm.

As the ride-hailing applications have access to both Alice's and Eve's mobile devices to track their real-time locations, Eve needs to hijack the GPS signal received by Alice in order to bypass the detection from the monitoring center. Otherwise, their real trace can be easily revealed in Alice's GPS report. Meanwhile, the ride-hailing App on both Eve's and Alice's devices can report their motion sensor readings along with GPS data to monitor irregular activities, such as an unexpected long stop. As a result, Alice's GPS single can be hijacked by Eve. However, because Eve cannot compromise Alice's device to alter the motion sensor measurements, the monitoring center can still use motion sensor measurements from Alice's device to verify if Alice's GPS signal is spoofed or not.

Case 2. In this scenario, Eve is an evil attacker who is capable of broadcasting spoofed GPS signals to the nearby autonomous vehicle, Alice, using the open-source GPS software [14]. As Eve intends to mislead Alice into following a spoofed route instead of the original one, Eve has to "teleport" Alice to a spoofed position, then emulates the movement from the new position by sending crafted GPS signals. But Alice's motion sensor measurements remain untouched in this attack because Eve cannot compromise Alice's device to alter the motion sensor measurements remotely.Under this circumstance, Alice can recover the real trajectory using motion sensor measurements to determine if she follows a wrong route or not.

In summary, the defender can use motion sensor measurements from Alice's device to reconstruct the real trajectory for detecting the GPS spoofing attacks in both cases.

4. DeepPOSE framework

This section introduces the proposed framework, DeepPOSE, for GPS spoofing detection using motion sensor data. As shown in Fig. 2, DeepPOSE has two components, a vehicle position estimator and a GPS spoofing detector.

4.1. Vehicle position estimator

This component contains a deep neural network for estimating the vehicle speed and direction in a sequence, which is used to infer the vehicle position. This model takes a sequence of sensor measurements as input and predicts a sequence of vehicle speed and direction accordingly. When training the model, we use the GPS coordinates from the training datasets to compute the real speed and direction, which are compared with the predicted vehicle speed and direction using the sensor measurements in the training datasets as the model input. Here we assume that the training datasets are from a trustable source, and hence the GPS coordinates are trusted. After the model is trained, GPS coordinates are no longer needed. It only needs the motion sensor measurements as input and predicts the vehicle's speed and direction. Nevertheless, before feeding the raw sensor measurements to the deep neural network, we need to perform pre-processing, including the sensor reorientation and denoising.

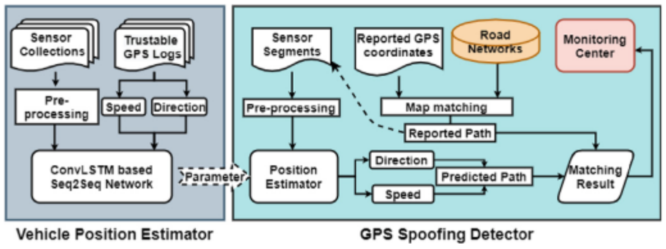


Fig. 2. Architecture of DeepPOSE.

4.2. GPS spoofing detector

This component obtains the GPS logs reported by the user and uses the sensor measurements and a street map to determine whether the reported GPS logs are spoofed. In this paper, we use the OpenStreetMap (OSM) [29], which contains all indexed edge and vertex information in an area. The detector matches the reported GPS logs with the street map to obtain the path from the source vertex to the destination vertex. It then aligns the sensor measurements with each segment on the path and reconstructs a trajectory using the position estimator with the aligned sensor measurements. The spoofing detector compares the trajectory constructed from the GPS logs and the trajectory constructed from the sensor measurements and then determines whether there is a spoofing attack.

4.3. Data pre-processing

4.3.1. Mobile sensor reorientation

For the mobile sensor platform, the device's position affects the measurement significantly. There are two relevant coordinate systems that exist simultaneously for a mobile device in a moving vehicle. The first one is the desired "canonical" coordinate, as illustrated by the red axes in Fig. 3, which is used to characterize the vehicle motion status. The second one is the three-axis accelerometer configuration (black axes) in some arbitrary orientation in terms of the placement of the mobile device on the vehicle's dashboard.

Let $\mathbf{a} = (\mathbf{a}_x, \mathbf{a}_y, \mathbf{a}_z)$ be the vector of the three acceleration measurements taken at a given point in the sampling interval, and $\mathbf{d} = (\mathbf{d}_s, \mathbf{d}_v, \mathbf{d}_f)$ denotes vehicle's coordinate system. $\mathbf{d}_f$ is the moving direction of the vehicle. In order to eliminate the differences between two coordinate systems and derive the useful vehicle speed estimation, we use a rotation matrix R' [30] to align the value measured from the mobile sensor to the "canonical" coordinate after each measurement. The new rotating sensor measurement can be expressed as $\mathbf{d} = \mathbf{a} \cdot R'$

4.3.2. Vehicle speed and direction extraction

In real-world applications, most mobile devices cannot access the vehicle states such as the speed and direction (steeling wheel angle)

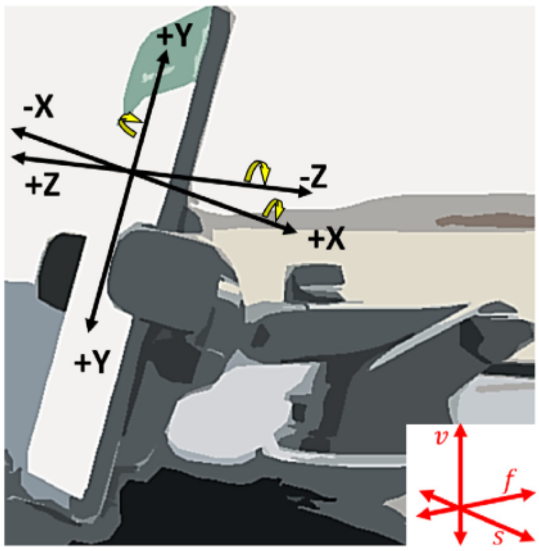


Fig. 3. Coordinate systems. The red axes denote the "canonical" coordinate, and the purple axes denote the accelerometer coordinate related to the position of the phone.

directly from the OBD-II port on a moving vehicle. Usually, the vehicle speed and direction are derived from a series of GPS coordinates. The current vehicle speed can be calculated by dividing the distance between the current position i and the previous position $i-1$ ($i>1$) by the elapsed time. However, we need to know the position of the next location $i+1$ for obtaining the current heading direction at position i. Both the distance ($\mathcal{D}_{\langle i,j\rangle}$) and bearing ($\mathcal{B}_{\langle i,j\rangle}$) between position i and j can be calculated from the following equation, given two adjacent GPS coordinates $\mathcal{C}_i$ and $\mathcal{C}_j$, and each of them contains a pair of latitude and longitude information

$$\begin{aligned} a &= \sin^2(\Delta\varphi) + \cos\varphi_i \cdot \cos\varphi_j \cdot \sin^2(\Delta\lambda/2) \\ c &= 2 \cdot \arctan 2(\sqrt{a}, \sqrt{1-a}) \\ \mathcal{D}_{<i,j>} &= R \cdot c \\ \mathcal{B}_{<i,j>} &= \arctan 2\sin\Delta\lambda \cdot \cos\varphi_{i+1}, \cos\varphi_i \cdot \sin\varphi_j - \\ &\quad \sin\varphi_i \cdot \cos\varphi_j \cdot \cos\Delta\lambda \end{aligned} \tag{1}$$

where φ_i is the latitude at $\mathcal{C}_i$, $\Delta\lambda$ represents the longitude difference between $\mathcal{C}_i$ and $\mathcal{C}_j$, and R is the earth's radius.

Once we can get the distance and bearing between two sets of GPS coordinates, we can obtain the vehicle speed v_i and relative direction $\theta_{\langle i,i+1\rangle}$ at position i as follows.

$$\begin{aligned} v_i &= \frac{\mathcal{D}_{\langle i-1,i\rangle}}{t_i - t_{i-1}} \\ \theta_{\langle i,i+1\rangle} &= \mathcal{B}_{\langle i+1,i+2\rangle} - \mathcal{B}_{\langle i,i+1\rangle} \end{aligned} \tag{2}$$

4.3.3. *Driving patterns analysis*

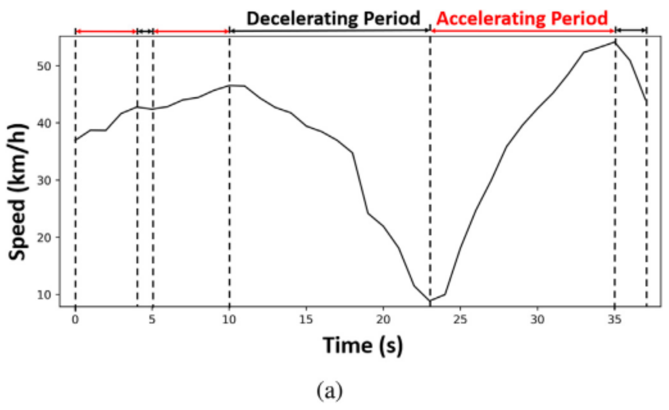
Accelerating, decelerating, and turning are common driving actions in real-world driving scenarios. These actions are performed by human drivers and are restricted by natural physical boundaries such as the vehicle's powertrain and road speed limits. Therefore, they show a strong periodicity. For example, a driver can take a relatively long time to speed up to the road limit from a stationary position, but it only takes a few seconds to stop completely in front of a stop sign. Analyzing such physical boundaries is important for our study to understand the composition of the noise in sensor data and physical laws that affect a moving vehicle, as well as to provide constructive guidelines for customizing parameters in our DeepPOSE framework.

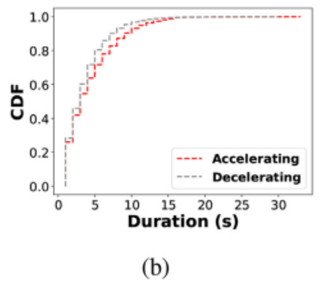
Fig. 4 provides a detailed driving behavior analysis based on the trip data from the BDD-100K [31] dataset, which is one of the largest driving video datasets in the literature, with 100K driving videos and 10 tasks aligned with the detailed GPS/IMU records to reveal the driving patterns and vehicle trajectories. The datasets reflect the diversity of geography, environment and weather, covering various driving conditions. Fig. 4(a) is a speed diagram of one trip in the dataset. From the figure, we can tell the selected trip has three accelerating and decelerating periods in one trip segment. Fig. 4(b) plots the CDF of the average time spent in acceleration and deceleration of vehicles in all trips in the dataset. From this figure, we observe that 90% of the acceleration and deceleration actions are completed within 10 s, which matches the fact that the average 0–60 MPH time for a vehicle is about 7–8 s. However, many other maneuvers, such as lane changes and overtaking, may require less time to complete than acceleration. As shown in Fig. 4(c), most subtle speed changes only take less than 5 s.

These features give us insights into changes in speed and direction in real-world driving scenarios and how to collect the most crucial data for vehicle position estimation. For instance, vehicle speed estimation may depend on the vehicle's state in the past 10 seconds or longer, whereas the estimation of the current direction may require a shorter observation time. In the next section, we will discuss how DeepPOSE utilizes these physical boundaries for the sequence-to-sequence approach and reconstructs a more accurate vehicle trajectory for GPS spoofing detection.

5. Vehicle position estimation

In this section, we introduce the detailed design of Vehicle Position Estimator, a core component in the DeepPOSE framework using deep learning techniques to estimate the vehicle speed and direction by taking a sequence of sensor measurements as input.





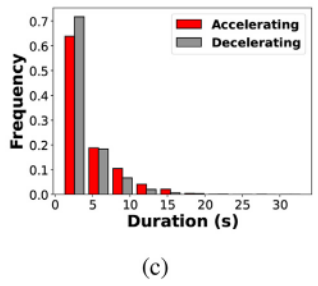


Fig. 4. Statistical analysis of speed changes of all trips in BDD-100K dataset: a) one sample trip from the dataset, (b) CDF of the average time taken for accelerating and decelerating, c) histogram of the average time taken for accelerating and decelerating.

5.1. *Model inputs and outputs*

We assume there are K different sensors with measurements denoted as $\mathcal{I} = \{\mathcal{I}_1, \ldots, \mathcal{I}_K\}, k \in \{1, \ldots, K\}$. Sensor k has the sampling frequency f_k. Let d_k represent the number of axles for each sensor, e.g., measurements along x, y, and z axes. The GPS coordinates $\mathcal{C}$ are sampled at the frequency f_c. We align the raw sensor measurements in Fig. 5 to explain how to turn the raw sensor measurement into the format we need for the deep learning model.

For ease of description, we define a time unit as the interval between two adjacent GPS coordinates, i.e., $1/f_c$ second. In our experiments, it is equal to 1s because the GPS sampling rate is 1 Hz. Take sensor k as an example. We use m_k to represent the total number of sensor measurements from sensor k collected in each trip. Let n denote the number of GPS coordinates. Let l_k be the number of sensor data samples of sensor k in one-time unit, i.e., $l_k = f_k/f_c$. If sensors are not sampled at the same sampling rate, we need to downsample the high-frequency sensor to ensure the same number of measurements per time unit.

Unlike other deep learning models that require each data input to be paired with a target or label, the sequence-to-sequence model used in our

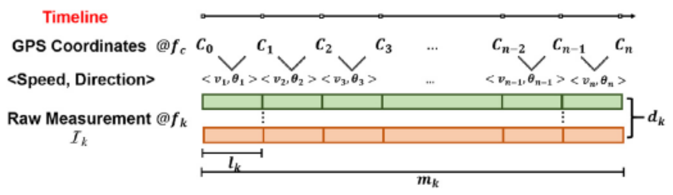


Fig. 5. Alignment between the motion sensor measurements and GPS coordinates.

design requires a series of inputs to be paired with a series of target speeds and directions in the time domain. When training the model, the real vehicle speed and direction can be derived from adjacent GPS coordinates in the training datasets by using (2), which are assumed trustable. After all sensor measurements are aligned with the correct targets, we split the raw measurements into small sequences with the same number of the time unit, n_τ. We use a 3-dimensional matrix, $\mathcal{X}^k$, to represent an input sequence generated by the sensor k. The depth of the input is n_τ, which is also the number of encoder/decoder in the sequence-to-sequence model. The height of the input is d_k, which depends on the number of axles of sensor k. The width of the input is $\omega \cdot l_k$, where ω is the size of the sliding window to control the number of sensor measurements aligned to one single target. When multiple sensors are considered, we stack the inputs into a $n_\tau \times \sum_k d_k \times \omega \cdot l_k$ matrix $\mathcal{X}$, as shown in Fig. 6. For example, suppose we measure the data from a mobile platform with two three-axis sensors ($\sum_k d_k = 6$), the sequence length (n_τ) is set to 10, and the sampling rate for both sensors is set to 50 Hz. The GPS sampling rate is 1 Hz. The real speed or direction can then be computed every second. There is a target every second. A straightforward approach is to use sensor measurements within 1 s to predict one speed or direction data point. However, generally, the performance will increase if we use a sliding window of sensor measurements, e.g., within the last *omega* seconds, to predict the speed or direction of the current second. Suppose we let *omega* $= 5$, then the input to our model has the shape $n_\tau \times \sum_k d_k \times \omega \cdot l_k$ or $10 \times 6 \times (5 \times 50)$.

As we discussed in Section 4.1.3, the change of vehicle speed and direction is limited by different physical laws and boundaries. Therefore, we use different parameters to train models for the estimation of vehicle speed and direction separately. A unified symbol, $\mathcal{Y}$, denotes the output vector corresponding to each input vector. When the model is used to estimate the vehicle speed, $\mathcal{Y} = \boldsymbol{v}$, where $\boldsymbol{v} = \{v_t\}$, $t \in \{1, \ldots, n_\tau\}$. Similarity, $\mathcal{Y} = \boldsymbol{\theta}$ when the target is the vehicle direction. For the rest of this paper, all vectors are denoted by bold $\mathcal{Y}$, and an instant target value at time t is denoted by $\mathcal{Y}_t$.

Since hardware specifications limit the choice of d_k and l_k, the most significant factor that could affect the selection of the value of n_τ is the time duration of the common driving maneuvers. Based on the analysis of all trips from the BDD-100K dataset, we have discovered the driving maneuvers related to the speed and direction change, such as accelerating and decelerating, are usually completed within 10 s. More than 90% of common maneuvers such as lane changes and overtaking require less than 5 s. With the sampling rate as 1 Hz, n_τ should range from 5 to 10 s. We will discuss how the selection of n_τ affects the system performance in Section 7.

5.2. Sequence-to-sequence modeling

Sequence-to-sequence learning (seq2seq) as a variant of RNN has achieved great success in machine translation, speech recognition, chatbots, text summarization and other series of data learning. Unlike a single or stacked RNN, which operates on a sequence and feeds its own outputs for subsequent cells, most seq2seq models are encoder-decoder models composed of a set of two RNNs. The first RNN *encoder* trains the input data and then passes the last state of its recurrent layer as an initial state to the first recurrent layer of the *decoder*. Meanwhile, the decoder obtains the state of the encoder's last recurrent layer and uses it as an initial state to its first recurrent layer, the input of the decoder is the target sequences such as speed or direction that we want to estimate.

From the following equation, we know that the goal of the encoder LSTM is to estimate the conditional probability $p(y_1, \ldots, y_{n_\tau}|x_1, \ldots, x_{n_\tau})$, where $(x_1, \ldots, x_{n_\tau})$ is the input sequences of sensor measurements, and $(y_1, \ldots, y_{n_\tau})$ is the vehicle target sequences. h is the accumulated hidden representation of input sequences $(x_1, \ldots, x_{n_\tau})$ based on the last hidden state produced by the LSTM encoder.

$$p(y_1, \ldots, y_{n_\tau}|x_1, \ldots, x_{n_\tau}) = \prod_{t=1}^{n_\tau} p(y_t|h, y_1, \ldots, y_{t-1}) \tag{3}$$

The overall scheme is outlined in Fig. 7. Based on the sequence-to-sequence model, the encoder LSTMs process the input sequence $\mathcal{X}$ of n_τ elements and pass the internal state (hidden state) representation to the next encoder until it reaches the last encoder. Then, the *first* decoder (on the right-hand side) gets the hidden state generated by the last encoder and adds one initial input $\mathcal{Y}_0$ to generate its target output $\mathcal{Y}_1$, and updates the hidden states. The second decoder takes the new hidden state and $\mathcal{Y}_1$ as inputs to generate a new output $\mathcal{Y}_2$, and so forth. Eventually, the target sequence $\mathcal{Y}$ is obtained, which is $(y_1, \ldots, y_{n_\tau})$. As the decoding process continues in a recursive manner, we just need to give the initial state to the decoding trip. For example, if the vehicle is moving from a stationary position, then $\mathcal{Y}_0$ should be set to zero. In other cases, it should be the vehicle speed or direction before starting the new decoding process.

Algorithm 1 explains the detailed steps in the decoding process. The length of the decoding sequence is controlled by n_τ. We first encode the sensor inputs in the encoding model and learn the hidden states $\boldsymbol{h}$ from the input sequences as the key feature. Once we know the starting value of the output sequence, then we start to decode each sequence recursively until we reach the end of the sequence.

Algorithm 1 Sequence decoding algorithm

```
 1: INPUT: Sensor input $\boldsymbol{\mathcal{X}}$, initial target $\mathcal{Y}_0$, encoder
    model $\boldsymbol{Enc}(\cdot)$, and decoder model $\boldsymbol{Dec}(\cdot)$
 2: OUTPUT: $\boldsymbol{\mathcal{Y}}$
 3: Get encoding state value $\boldsymbol{h} = \boldsymbol{Enc}(\boldsymbol{\mathcal{X}})$
 4: Set target sequence $\boldsymbol{\mathcal{Y}} = []$;
 5: Set current target $\boldsymbol{\mathcal{Y}}_c = \mathcal{Y}_0$;
 6: for $t = 1 \to n_\tau$ do
 7:     $\boldsymbol{\mathcal{Y}}', \boldsymbol{h}' = \boldsymbol{Dec}(\boldsymbol{\mathcal{Y}}_c, \boldsymbol{h})$;
 8:     $\boldsymbol{\mathcal{Y}}.append(\boldsymbol{\mathcal{Y}}')$;
 9:     $\boldsymbol{h}' \to \boldsymbol{h}$;
10:     $\boldsymbol{\mathcal{Y}}' \to \boldsymbol{\mathcal{Y}}_c$;
11: end for
12: return Prediction Target $\boldsymbol{\mathcal{Y}}$
```

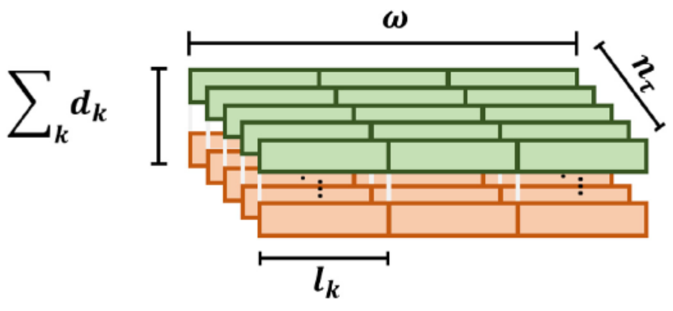


Fig. 6. A general representation of model input.

5.3. ConvLSTM network

The standard sequence-to-sequence network is composed of a set of vanilla LSTM encoder and decoder units to interpret the sequential data. The vanilla LSTM network suffers from vanishing and exploding gradient problems. Hence it is easy to overfit. To address this issue, the ConvLSTM unit was proposed in Ref. [26] to make it more robust by introducing the convolution operations in exploring spatiotemporal features of the time series data. The ConvLSTM is a good tool for vehicle speed and direction estimation as the LSTM part can capture the temporal transition of a moving vehicle, and the convolution layer can capture the local spatial data. As the core function of LSTM, the memory cell c_t plays a vital role to

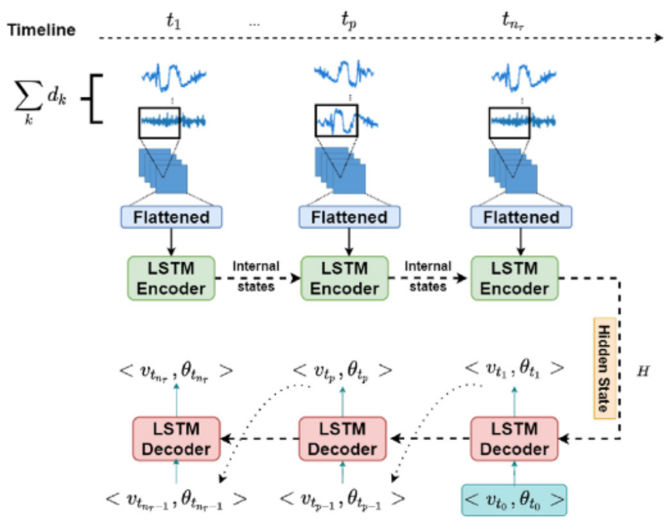

Timeline
t_1
t_p
t_{n_τ}
$\sum_k d_k$
Flattened
LSTM Encoder
Internal states
Hidden State
H
$< v_{t_{n_\tau}}, \theta_{t_{n_\tau}} >$
$< v_{t_p}, \theta_{t_p} >$
$< v_{t_1}, \theta_{t_1} >$
LSTM Decoder
$< v_{t_{n_\tau -1}}, \theta_{t_{n_\tau -1}} >$
$< v_{t_{p-1}}, \theta_{t_{p-1}} >$
$< v_{t_0}, \theta_{t_0} >$

Fig. 7. Architecture of the proposed ConvLSTM-based Sequence-to-Sequence Network. The hidden layer representation of accelerometer and gyroscope data is encoded from left to right with LSTM cells. The final hidden state of the sequence is forwarded to the decoder as part of the input. The initial speed or direction needs to be provided to start the decoding process.

accumulate the previous hidden states and apply the input x_{t-1} to the next step of the sequence. However, in addition to the standard LSTM that only uses features in one dimension, ConvLSTM is capable of taking multi-dimensional data as inputs (multi-dimensional sensor data in our case). The detailed state transition of convLSTM can be described by the following formula.

$$\begin{aligned}
&\mathbf{i}_t = \sigma(\mathbf{W}_{xi}{}^*\mathbf{x}_t + \mathbf{W}_{hi}{}^*\mathbf{h}_{t-1} + \mathbf{b}_i),\\
&\mathbf{f}_t = \sigma\big(\mathbf{W}_{xf}{}^*\mathbf{x}_f + \mathbf{W}_{hf}{}^*\mathbf{h}_{t-1} + \mathbf{b}_f\big),\\
&\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_{xc}{}^*\mathbf{x}_t + \mathbf{W}_{hc}{}^*\mathbf{h}_{t-1} + \mathbf{b}_c),\\
&\mathbf{o}_t = \sigma(\mathbf{W}_{xo}{}^*\mathbf{x}_t + \mathbf{W}_{ho}{}^*\mathbf{h}_{t-1} + \mathbf{b}_o),\\
&\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t),
\end{aligned} \tag{4}$$

where σ is the logistic sigmoid function, $\mathbf{i}_t$, $\mathbf{o}_t$, $\mathbf{c}_t$ and $\mathbf{h}_t$ are vectors to represent values of the input gate, forget gate, output gate, cell activation, and cell output at time t, respectively.

DeepPOSE employs three convolution layers for each encoder, with 64, 128, and 256 filters, receptively. Each convolution layer uses a kernel with the size (1,5). After each convolution layer, a batch normalization layer and a max-pooling layer are followed to reduce the internal covariate shift [32] and the spatial size of the feature.

5.4. Loss function

In the vehicle position estimation, two components are required: vehicle speed and direction. Due to the change of vehicle speed and direction following different patterns, as mentioned in Section 4.1.3, we need to train two models with different parameters to obtain the best performance. So, for each individual model, the task is to infer the vehicle speed, or direction, given a sequence of sensor inputs over time. This is a standard regression problem because the speed and direction are floating values. Thus, we select the mean square error as the cost function.

In the training process, we use $D(\cdot)$ to denote the decoder output of our model and $\langle \mathcal{X}, \mathcal{Y} \rangle$ to denote the training samples and labels. Then the loss function for training the model is given as follows.

$$\mathcal{L} = \sum_t^{n_\tau} \ell(\boldsymbol{Dec}(\mathcal{X}_t), \mathcal{Y}_t) + \sum_j \lambda_j P_j \tag{5}$$

The second term in (5) is the regularization function, and λ_j controls the importance of penalty or regularization terms.

6. GPS spoofing detection

The underlying idea of spoofing detections in our design is to compare whether the path constructed by the reported GPS coordinates and the path constructed by the position estimator follow the same driving pattern. If the GPS signals are spoofed, the constructed path will be quite different from the path constructed from the motion sensor data. However, it is a great challenge to reconstruct a reasonably accurate trajectory path from the noisy motion sensor data. This is because the estimation error from the noisy motion sensor data is relatively large, and such a large estimation error quickly accumulates to be out of control, making the reconstructed path useless — for example, Fig. 8 shows the accumulated error of the distance estimation from the starting point while driving in an urban area for 3 min. We can see even after only 3 min, the displacement estimation error has accumulated more than 800 m for a distance of only 4000 m, with the best noise-canceling method we have tested.

To address this issue, we propose a smart map alignment scheme to fit the *reconstructed path* from motion sensor data to the street map, which is of great benefit to reducing the error accumulation since the errors are often reset to zero on a new road segment. Next, we describe this scheme.

6.1. Map-aided alignment

Let $\mathcal{G}$ denote a street map, which is a directed graph in an area, where vertices and edges represent intersections and road segments between intersections, respectively. If a path $\mathcal{P}$ exists between two intersections/vertices v_i and v_j, it means that we can reach v_j by following a set of road segments or edges from v_i, i.e., $\mathcal{P} : e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_n$, where $e_1.start = v_i$, and $e_n.end = v_j$.

Directly mapping the motion sensor data to a street map is a great challenge due to the high noise of the former. To address this challenge, we first map the reported GPS data on the street map to obtain the path traveled by the driver, $\mathcal{P} : \{e_1 \rightarrow \ldots \rightarrow e_k\}$. Then, we align the sensor measurements $\mathcal{X}$ with the road segments or edges according to the timestamp of the corresponding GPS coordinates $\mathcal{C}$. Specifically, we first find the GPS coordinates $\mathcal{C}_i$ on edge e_i. Then, based on the timestamps of GPS coordinates $\mathcal{C}_i$, we find the corresponding sensor measurements $\mathcal{X}_i$. Thus, we align the sensor measurements with the edges, assuming that there is no GPS spoofing. In the case that there is GPS spoofing, we still use this scheme to align sensor measurements. Our detection algorithm can actually utilize this incorrect alignment to detect GPS spoofing.

Unfortunately, even the GPS map alignment is not trivial in our problem because the real driving trajectory and the trajectory represented in the map can be different, which is caused by various factors,

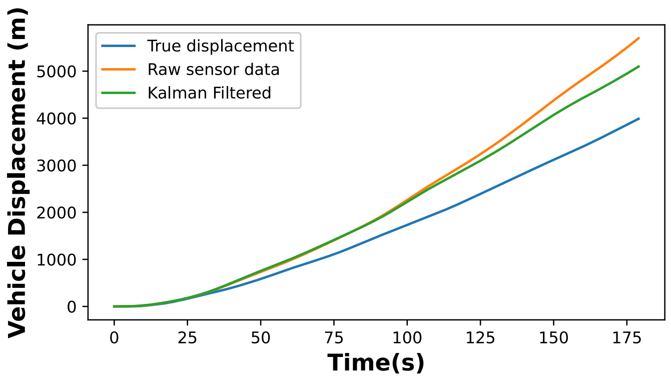


Fig. 8. The displacement estimation of a vehicle in a 3-min real-world driving based on the derived vehicle speed from the accelerometer readings.

including GPS measurement noises, wide road conditions, etc. For example, Fig. 9 shows a real trajectory of the vehicle (red nodes) and the matched nodes on the street map (green nodes) when the driver makes a left turn at a wide intersection. We notice the turn takes about 7 s (GPS sampling rate is 1 Hz), and the real driving trajectory drifts from the projected path on the map, $\{e_1, e_2, e_3, e_4\}$. It is reasonable to drift from edge e_1 because the vehicle needs to change lanes before making a left turn. But, it is not likely to follow e_2 and e_3 strictly to make a sharp turn in real-world traffic. Therefore, we have to precisely align the starting and ending points of sensor measurements in order to correctly reconstruct a turn from the motion sensors when fitting the reconstructed path on the real path.

Inspired by the work in Refs. [33,34], we use two normal distributions to simulate the spatial and direction probabilities of vehicles, which defines the possibility of mapping the GPS point $\mathcal{C}_i$ vertex v_j on the street map based on the spatial and angle differences, respectively.

$$P(v_j, \mathcal{C}_i) = \frac{1}{\sigma_s\sqrt{2\pi}} e^{-d(v_j,\mathcal{C}_i)^2 / 2\sigma_s^2} \tag{6}$$

$$Q(e_j, \mathcal{C}_i) = \frac{1}{\sigma_d\sqrt{2\pi}} e^{-\Delta\mathcal{B}(e_j,\mathcal{C}_i)^2 / 2\sigma_d^2} \tag{7}$$

where $d(v_j, \mathcal{C}_i)$ is the distance between the report GPS coordinate $\mathcal{C}_i$ and the GPS coordinate of vertex v_j on the street map, and $\Delta\mathcal{B}(e_j, \mathcal{C}_i)$ is the bearing difference between edge e_j and $\mathcal{C}_i$, which is expressed as $\mathcal{B}_{\langle v_{j+1}, v_j\rangle} - \mathcal{B}_{\langle i+1, i\rangle}$, where v_{j+1}, v_j are the vertices on e_j, and $\mathcal{B}_{\langle i+1,i\rangle}$ is the bearing between $\mathcal{C}_i$ and $\mathcal{C}_{i+1}$. The bearing calculation is in (1). σ_s and σ_d are the standard deviations of the position measurement error and directional measurement error obtained from the empirical experiments. Combining (6) and (7), we obtain:

$$P_j^i = P(v_j, \mathcal{C}_i) \times Q(e_j, \mathcal{C}_i) \tag{8}$$

which represents the probability of GPS coordinate $\mathcal{C}_i$ being mapped to vertex v_j. We apply the above model to align the GPS coordinates with each vertex on the route.

6.2. Detection mechanism

We assume a user continuously reports its current GPS coordinates $\mathcal{C}$ to the application server, together with the motion sensor measurements. In order to detect the GPS spoofing attack, the application server utilizes all the GPS coordinates and sensor measurements in an interval starting from a past time point to the current time. This makes it possible to detect GPS spoofing attacks dynamically and report the attack in time to avoid possible severe damage. Certainly, it can also be used to detect if there is a GPS spoofing attack for a completed trip to catch fraud.

6.2.1. Detection for threat model case 1

Let $\mathcal{C}$ and $\mathcal{X}$ denote the GPS coordinates and sensor data, respectively. Algorithm 2 describes the GPS spoofing detector. The algorithm first uses a map-matching algorithm in Ref. [33] to construct a path $\mathcal{P}$ based on the GPS coordinates $\mathcal{C}$ and the street map $\mathcal{G}$. In this paper, we use the OpenStreetMap [29]. After the GPS-based path $\mathcal{P}$ is constructed, then for each road segment or edge $e_i \in \mathcal{P}$, we find the timestamps of the GPS coordinates corresponding to e_i, and accordingly identify the list of motion sensor data that is in the same time frame. Let $\mathcal{X}_i$ denote the motion sensor data corresponding to e_i. This process is explained from line 9 to line 14. To be more specific, we need to search the exact starting and ending GPS points ($\mathcal{C}_{start}$ and $\mathcal{C}_{end}$) for each edge on the path and then pack the sensor measurements between the timestamps at these two GPS positions into $\mathcal{X}_i$. The core function we use to match a GPS point and a vertex, *MatchGPS* in lines 9 and 13 is described in Algorithm 3.

Algorithm 2 Case 1 Spoofing Detection Algorithm

```
1:  Input: Map: $\mathcal{G} = (V, E)$, reported GPS coordinates: $\mathcal{C}$, sensor measurements: $\mathcal{X}$, sequence decoder: $\mathcal{D}$
2:  $\mathcal{P}: \{e_1 \to \ldots \to e_k\}$ = ST-Matching($\mathcal{G}, \mathcal{C}$) // return all connected edges on the route
3:  Initialize an empty queue $\mathcal{M}$ = []
4:  $s_i = e_i.start$ // starting vertex of edge i
5:  $v_i = e_i.end$ // ending vertex of edge i
6:  $i = 1$
7:  while $i \le k$ do
8:      $j = i$
9:      $\mathcal{C}_{start}$ = MatchGPS($s_i, \mathcal{C}$)
10:     while $distance(s_i, v_j) < \delta$ and $j \le k$ do
11:         $j = j + 1$
12:     end while
13:     $\mathcal{C}_{end}$ = MatchGPS($v_j, \mathcal{C}$)
14:     $t_{start}, t_{end} \leftarrow$ timestamps of $\mathcal{C}_{start}$ and $\mathcal{C}_{end}$.
15:     $\mathcal{X}_i \leftarrow$ sensor data from $t_{start}$ to $t_{end}$
16:     $Speed, Direction \leftarrow \mathcal{D}(\mathcal{X}_i)$
17:     $T_{\mathcal{D}(\mathcal{X}_i)} \leftarrow$ trajectory from $Speed$ and $Direction$
18:     $T_{\mathcal{C}_{start}\to\mathcal{C}_{end}} \leftarrow$ trajectory from $\mathcal{C}_{start}$ to $\mathcal{C}_{end}$
19:     Add $diff_{(T_{\mathcal{D}(\mathcal{X}_i)}, T_{\mathcal{C}_{start}\to\mathcal{C}_{end}})}$ to queue $\mathcal{M}$
20:     $i = i + 1$
21: end while
22: if $\frac{\sum_i \mathcal{M}[i]}{|\mathcal{M}|} \ge \alpha$ then
23:     Spoofing Detected!
24: end if
```

Algorithm 3 Match GPS Algorithm

```
1: Input: Target vertex: $v_j$, GPS coordinates $\boldsymbol{C}$
2: Number of GPS coordinates: $n \leftarrow len(\boldsymbol{C})$
3: Initialize the best match id: $k \leftarrow 1$
4: for $i = 1 \to n$ do
5:     calculate $P_j^i$ from (8
6:     if $P_j^i > P_j^k$ then $k \leftarrow i$
7:     end if
8: end for
9: Return $\mathcal{C}_k$
```

Once we have obtained the GPS starting point, we need to search along the road to the ending point. In line 10, we introduce a constant δ to control the number of edges for consideration before concluding this road segment. After analyzing the road structures in the city of Norfolk, Virginia, we have found that the average and medium distances between two vertices on the map are 122 and 91 m. According to this finding, if two or more vertices are in close distance, it is likely to be an intersection and requires more attention to the reconstruction. For example, edges e_2 and e_3 in Fig. 9 are in this case. In this paper, we set δ to 90 m, which means multiple short edges will be concatenated as a longer road segment. This allows us to capture enough data that matches the driving pattern. If the edge distance is too small, it may separate a whole driving

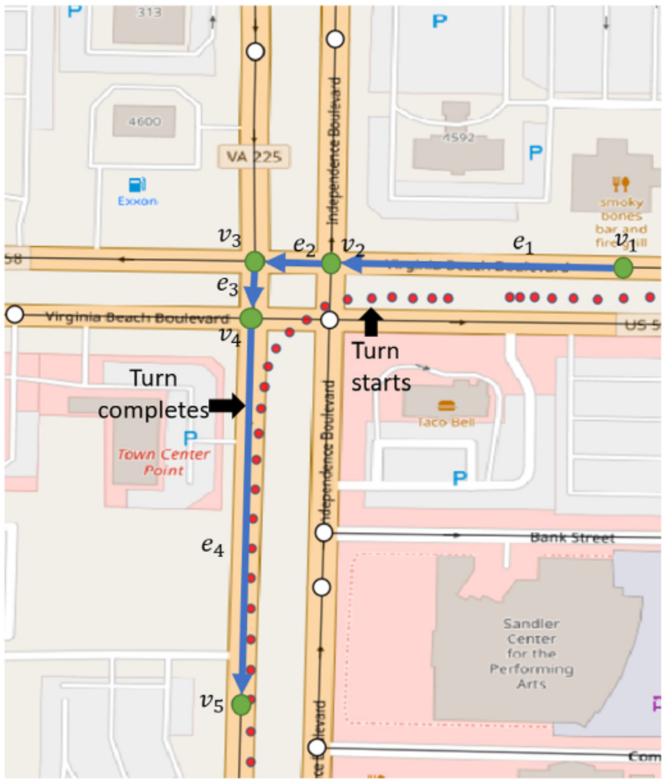


Fig. 9. An illustration of mapping the starting and ending GPS point of a left turn at an intersection. Red dots represent the GPS coordinates, and green nodes represent the matched vertices on the map, which are connected by the blue edges. The black arrow shows the direction of all other edges.

pattern into pieces. It will increase the difficulty of interpreting the patterns behind the wheel.

After we have obtained the ending position and packed the sensor collections between these two points into $\mathcal{X}_i$ (line 15), we can reconstruct the real trajectory by inferring the vehicle speed and direction transition with the sequence decoder $\mathcal{D}(\cdot)$. Moreover, we also reconstruct the trajectory from the GPS coordinates from the starting point to the endpoint on edge e_i (or combine edges as discussed above). For each pair of trajectories, we use the well-known *Dynamic Time Wrapping* (DTW) algorithm [35] to measure the similarity between them. Let $dtw(T_A, T_B)$ represents the difference between two trajectories T_A and T_B. However, as the length of road segments in a street map may vary in a broad range, we normalize $dtw(T_A, T_B)$ by dividing it by the length of both trajectories as in (9). It is denoted as $diff_{(T_A,T_B)}$. The normalized trajectory difference is used as the evaluation criterion in line 19 to score the overall similarity between $T_{\mathcal{D}(\mathcal{X}_i)}$ and $T_{\mathcal{C}_{start}\to\mathcal{C}_{end}}$.

$$diff_{(T_A,T_B)} = \frac{dtw(T_A,T_B)\times 2}{len(T_A)+len(T_B)} \tag{9}$$

The result on each selected edge is stored in a queue $\mathcal{M}$. After the average trajectory difference from all edges is obtained, we compare it to a threshold α to distinguish whether this user is under the GPS spoofing or not. The selection of the threshold values α is based on the characteristics of trips, e.g., the average trip length, and is empirically set based on the characteristics of the trip in an area. We will have a further discussion in Section 7.3.2 regarding the selection of α.

6.2.2. Detection for threat model case 2

In Threat Model Case 2, Eve can start a spoofing attack when Alice is in the middle of a road. As illustrated in Fig. 1, an external attacker Eve misleads the autonomous vehicle to go off-road at an intersection. This is rather challenging as the spoofing detector needs to identify the attack very quickly. We propose Algorithm 4 for this type of threat, which can achieve live detection even if the vehicle is currently driving in the middle of a road.

Algorithm 4 Case 2 Spoofing Detection Algorithm

```
1: Input: Vehicle last vertex: $v_l$, GPS coordinates $\boldsymbol{C}$,
   sequence decoder: $\mathcal{D}$
2: Use Algorithm 2 to detect spoofing until $v_l$
3: if no spoofing detected until $v_l$ then
4:     $t_{start} \leftarrow$ timestamps of $\mathcal{C}_{v_l}$
5:     $\mathcal{X}_i \leftarrow$ sensor data from $t_{start}$ to the current time
6:     $T_{v_l\to v_j} \leftarrow$ trajectory from $v_l$ to the estimated
       position computed by the travel time on the road
       segment since $v_l$
7:     $T_{\mathcal{D}(\mathcal{X}_i)} \leftarrow$ trajectory from $\mathcal{D}(\mathcal{X}_i)$
8:     if $diff_{(T_{\mathcal{D}(\mathcal{X}_i)},T_{v_l\to v_j})} > \beta$ then
9:         Spoofing Detected!
10:    end if
11: end if
```

The basic idea behind Algorithm 4 is to determine whether the vehicle travels along the planned path to the vertex on the map when the vehicle is driving on the current edge towards the next edge e_n. To be specific, Algorithm 4 uses the last vertex v_l passed by the vehicle, the next edge e_n in the planned path, and GPS coordinates $\mathcal{C}$ and sensor measurements $\mathcal{X}$ as the input. First, we check whether the vehicle is spoofed until the last vertex v_l using the detection algorithm in the preceding section (Algorithm 2). If spoofing is detected, there is no need to continue as we cannot trust the current position at all. If there is no spoofing detected so far, then we can extract the sensor measurements from the timestamp of $\mathcal{C}_{v_j}$ to the current time. Similar to Algorithm 2, we need to use the sequence decoder to extract the instant speed and direction to reconstruct the trajectory. This process will go on until the end of the current edge, v_j is reached. In order to analyze the similarity between the current path and the planned path, the same evaluation metric (9) is adopted. The $diff_{(T_{\mathcal{D}(\mathcal{X}_i)},T_{v_l\to v_j})}$ represents the difference between the current path and the planned path. Another threshold β is used to determine whether the current vehicle follows the planned route or not. Similar to α, the selection of the threshold values β is also based on the characteristics of trips, which will be discussed in Section 7.3.3.

7. Performance evaluation

In this section, we evaluate the performance of our proposed framework, DeepPOSE, in detecting real-world GPS spoofing attacks.

7.1. Data source

DeepPOSE is trained and evaluated on two different datasets collected from real-world driving scenarios, BDD-100K, and another custom dataset collected by the authors in the city of Norfolk, Virginia.

7.1.1. BDD-100K [31]

It is one of the largest driving video datasets with 100K driving videos aligned with the detailed GPS and measurements from multiple motion sensors to reveal the driving patterns and vehicle trajectories. The dataset preserves the diversity of geography, environment and weather to cover various driving conditions. We select 40k valid trips as our training data, and each trip contains 40 seconds of driving data. Fig. 10(a) shows the distribution of average travel distance in each trip from the selected 40K training set.

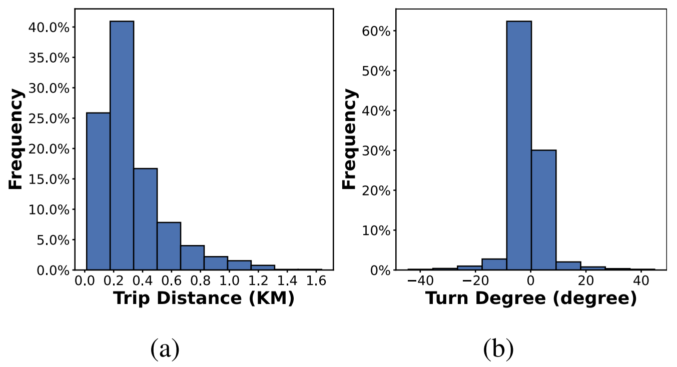


Fig. 10. a) Histogram of driving distance in BDD-100K; b) histogram of turn degree in BDD-100K.

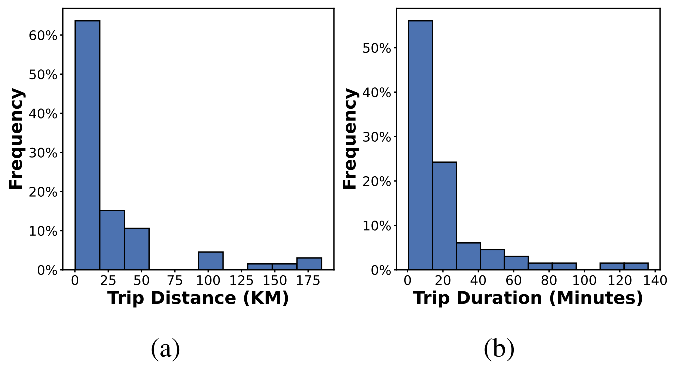


Fig. 11. a) Histogram of driving duration in Custom Dataset; b) histogram of driving distance in Custom Dataset.

7.1.2. *Custom dataset*

As trips in BDD-100K are fragmentized, we cannot fit the trip data in BDD-100K directly to implement the experiments that require a longer travel period. To this end, we further collect a customized dataset for the purpose of validating the performance of our model for real world GPS spoofing attacks. Data was collected from a Samsung Galaxy S7 mounted on the car dashboard at the same position. The sampling rate of the accelerometer and gyroscope is 50 Hz, and both measurements are aligned with the vehicle's GPS signal, which is sampled at 1 Hz. Our custom dataset contains trips driving in urban and rural areas for over 24 h, and most of the data is collected in good weather to eliminate poor GPS signal reception. During the collection, we mainly focus on the commuting distance in the urban area (around 20 min). We also include a few long-distance road trips, as shown in Fig. 11. Compared with BDD-100K, our custom dataset has a wider distribution in terms of driving distance and time duration for long term driving analysis.

7.2. *Vehicle position estimation*

Having an accurate estimation of the current vehicle position is crucial for the GPS spoofing detection algorithm. Thus, in this section, we evaluate the performance of vehicle position estimation, where the speed and direction are estimated based on the raw sensor measurements.

7.2.1. *Speed and direction estimation accuracy*

We evaluate the speed and direction estimation based on different combinations of source input sensors measurements, denoising techniques, and important hyperparameters. We have implemented the DeepPOSE framework on both BDD-100K and our custom dataset. The traditional sensor fusion technique is used as the baseline measurement. Sensor fusion is a method of double-integrating on the acceleration sensor readings to obtain the vehicle's displacement. The raw sensor measurements are denoised by one of the most widely used filters, Kalman filter [36], in our experiments. Table 1 presents error rates in speed and direction estimation compared with the baseline model (sensor fusion).. In this table, we compare the sensor fusion method (SensorFusion) with DeepPOSE in three variants: 1) “POSE-BDD-RAW”, the DeepPOSE model trained with the raw sensor measurements in the BDD-100K dataset, 2) “POSE-BDD-KM”, the DeepPOSE model trained with the sensor measurements from BDD-100K dataset, after applying the Kalman filter, and 3)“POSE-CUS-KM”, the DeepPOSE model trained with the sensor measurements from our customized dataset after applying the Kalman filter. The error rate is obtained by using the following formula: $\varepsilon_s = \sum_{t=1}^{n_\tau} \left| S_t - \tilde{S}_t \right| / \tilde{S}_t$, where $\tilde{S}_t$ is the actual vehicle speed or direction obtained from the filtered GPS signals.

From Table 1, we observe that DeepPOSE reduces the estimation error significantly compared with the sensor fusion approach in all scenarios. For example, the speed error is as low as 5% when we only use accelerometer sensor measurements as input, whereas the error rate of the sensor fusion approach is about 15%.

We next evaluate the performance of DeepPOSE by feeding different types of sensor data. We find the accelerometer itself has a good estimation of the vehicle speed and direction. Adding gyroscope data can improve the overall performance because it measures the rotation speed of an object. But the gyroscope measurement alone is not enough to estimate the vehicle speed. Moreover, it is not surprising to find that applying the Kalman filter can improve performance by eliminating additional noise introduced by the vehicle, such as engine vibration and road feedbacks. The performance of our custom dataset has decreased due to a relatively small amount of the driving data compared with the BDD-100K dataset. But it still reaches an acceptable error rate.

7.2.2. *Accuracy of position estimation*

Once we have the vehicle speed and direction information, we can reconstruct the position for a moving vehicle. Now we examine the overall performance of the vehicle position estimation.

Displacement error of each trip. Table 2 shows the displacement error measured by the difference between the ground truth displacement of each trip and the values integrated from the vehicle speed obtained via DeepPOSE and Sensor Fusion. We notice $n_\tau = 10$ has the best performance for vehicle speed reconstruction in Table 1.Hence we fix n_τ to 10 in the rest of the experiments. Results in Table 2 reveal that DeepPOSE can achieve an average error of 26.8 m with a standard deviation of 5.2 m in the selected 10k validation trips from the BDD-100K dataset. Sensor

Table 1
Speed and direction estimation error when the size of the sliding window (ω) is set to 3.

Selected Features	Accel Only						Gyro Only						Accel + Gyro					
n_τ	5		10		20		5		10		20		5		10		20	
$\mathcal{T}$	v/%	θ/%	v/%	θ/%	v/%	θ/%	v/%	θ/%	v/%	θ/%	v/%	θ/%	v/%	θ/%	v/%	θ/%	v/%	θ/%
SensorFusion	11	/	15	/	17	/	/	/	/	/	/	/	/	/	/	/	/	/
POSE-BDD-RAW	9	9	7	11	10	13	15	9	15	12	19	13	9	10	6	11	9	13
POSE-BDD-KM	6	4	5	6	8	8	10	9	12	11	15	10	4	4	3	6	5	8
POSE-CUS-KM	8	6	7	7	9	12	13	13	14	15	15	12	9	8	7	10	12	13

Table 2
Average displacement error per trip with standard deviation.

	Mean Absolute Error (meter)	
	BDD-100K	Custom Dataset(Urban Area)
DeepPOSE-KM	26.8 ± 5.2	36 ± 6.1
DeepPOSE-RAW	32.2 ± 6.7	40 ± 7.6
DeepPOSE-KM-noConv	48.4 ± 10.2	52 ± 13.9
DeepPOSE-RAW-noConv	53.6 ± 12.5	58 ± 15.7
SensorFusion	902 ± 63	1014 ± 70

Fusion results in a mean error of 902 m. This result beats the benchmark model [27], which can achieve a mean error of 40.43 m with a standard deviation of 5.24 m. The mean displacement error of our custom dataset is slightly higher than that of BDD-100K, which is about 36 m.

Once we combine the displacement and direction values, we can reconstruct the real vehicle trajectory as illustrated in Fig. 12.

7.2.3. Impact of sequence control variables

In the estimation of vehicle speed and direction by using the sequence-to-sequence model, two control variables affect the performance significantly, i.e., the length of the sequence, and the length of sliding windows. These two factors reflect two different degrees of cyclicality. The following discussion explains whether our sequence-to-sequence-based model is capable of capturing the periodic patterns in driving behaviors from the empirical results revealed in Section 4.1.3.

7.2.3.1. ***Estimation error versus length of sequence.*** Table 1 reflects how the choice of the length of the sequence, n_τ, affects the estimation accuracy. The shorter sequence length is helpful for direction estimation. This finding matches our discussion in Section 4.1.3 regarding the different cyclical levels of driving patterns, i.e., the state transitions time for speed change actions such as accelerating and decelerating. If n_τ for the speed estimation is too small, DeepPOSE may not learn the complete state transition of a vehicle in the training process, and vice versa. Too many repeated features may eventually degrade the performance.

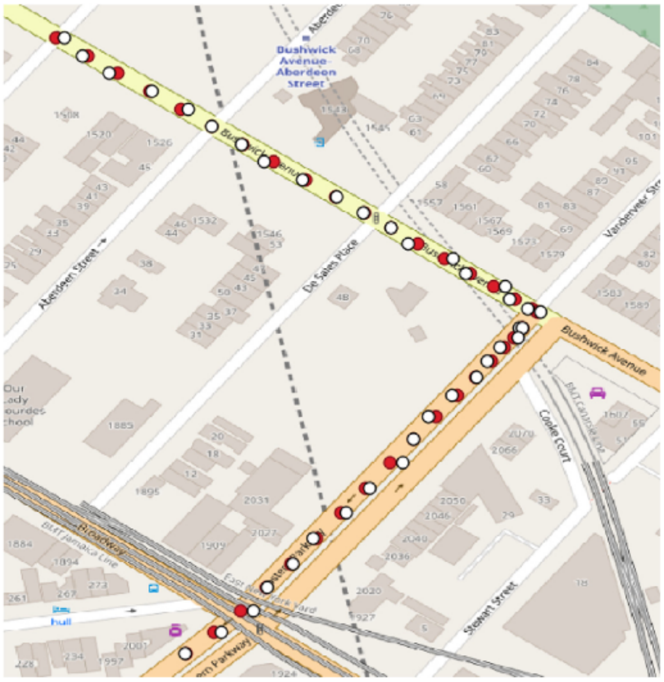


Fig. 12. A reconstruction sample from BDD-100K dataset. This trip segment lasts 35s and starts from the bottom of this figure. Red dots reveal the vehicle's trajectory computed from the GPS coordinates of the trip, while the white dots represent the estimated trajectory from the motion sensor of the trip.

However, different from the speed change, subtle maneuvers such as lane turning, passing, and changing use less time. Thus, as reflected in the results, the optimal n_τ for the direction estimation is smaller than the value for the speed estimation.

7.2.3.2. ***Displacement error versus sliding window (ω).*** We consider how the reconstruction accuracy is affected by the sliding window ω, a control variable that determines how many useful sensor measurements should be considered in one input data, which shows another type of periodic pattern that reflects the subtle change of vehicle in each time sequence. Based on the distribution of the drivers' average time for each individual action (Fig. 4(c)), we choose 3s, 5s, and 7s, which account for 60%, 85%, and 90% of the evaluation distribution. When the width of the sliding window increase more sensor measurements will be used to compose one single input $\mathcal{X}$. Table 3 shows the mean error of each trip in the BDD-100K dataset when we increase the length of the sliding windows, with n_τ set to 10 s. We observe that when we increase the width of the window width from 1 s to 5 s, the performance increases. This is because the more sensor measurements we consider in one input, the more detailed operational measurements will be included. However, the performance stops increasing when the width of the sliding window is over 7 s because this model is overwhelmed with too much repeated data, including noises. On the other hand, increasing the width of the sliding windows also increases the model training time. Considering the training efficiency and performance, a sliding window of 3 s is the best choice in our scenario.

7.3. GPS spoofing detection

7.3.1. Effect of map alignment on trajectory estimation

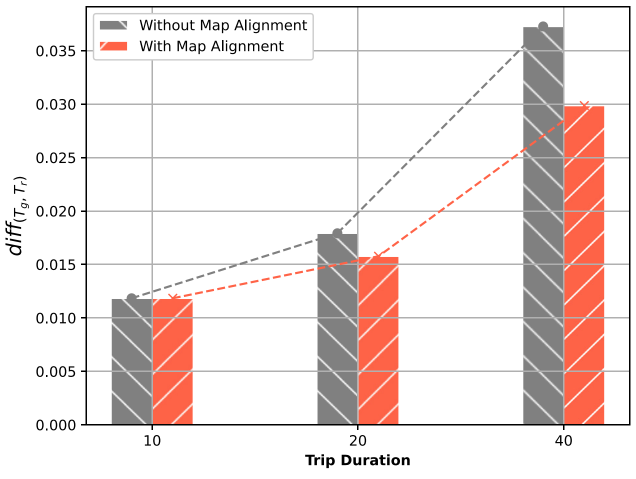
As illustrated in Fig. 12, the proposed vehicle position estimator can reconstruct the vehicle trajectory from the motion sensor measurements. Next, we examine the estimation error range of the trajectory computed from motion sensor data, with and without the map alignment. We select 10,000 trips from the BDD-100K dataset, as well as all the trips from our custom dataset, which have longer trip duration. For each trip, we first get the ground truth trajectory T_g from the GPS coordinates, and the trajectory T_r computed based on the sensor measurements, with and without using the map alignment. We use (9) to get the normalized trajectory differences between T_g and T_r in each set. Fig. 13(a) shows the difference between the trips selected from BDD-100K, and Fig. 13(b) shows the same results from our custom dataset when there is no spoofing attack. It is clear that the trajectory estimation error from the motion sensor data increases when the trip duration is longer due to the error accumulation in trajectory estimation. However, after we have applied the map alignment on the motion sensor data of the trips and then computed the trajectory similarity, the error accumulation of the trajectory reduces significantly, especially for long-duration trips.

7.3.2. Detection for threat model case 1

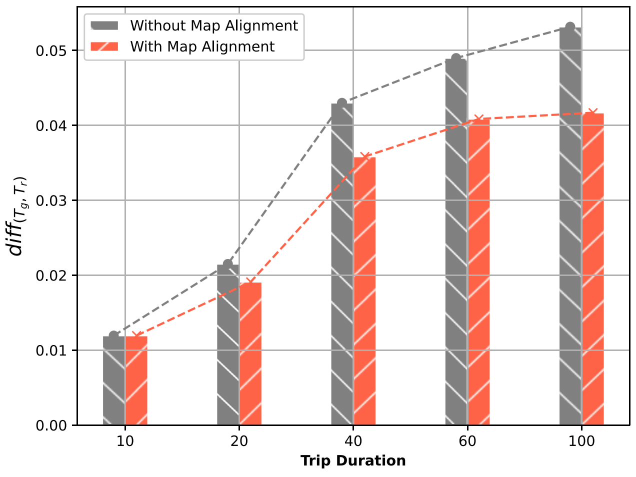
To simulate the GPS attack, we randomly select 5000 trips of various lengths from the BDD-100K dataset and keep the GPS coordinates and the

Table 3
Displacement error for different sliding window size ω in BDD-100K dataset.

Windows Size	Input Shape	Kernel Size	Mean Absolute Error (m)
1	10 × 6 × 50	[1,5]	29.7
		[1,15]	27.6
3	10 × 6 × 150	[1,5]	26.4
		[1,15]	24.9
5	10 × 6 × 250	[1,5]	25.5
		[1,15]	24.9
7	10 × 6 × 350	[1,5]	26.1
		[1,15]	25.8



(a)



(b)

Fig. 13. The average difference between the reconstructed trajectory and the ground truth trajectory on (a) BDD-100K and (b) custom dataset, with and without the assistant of the map.

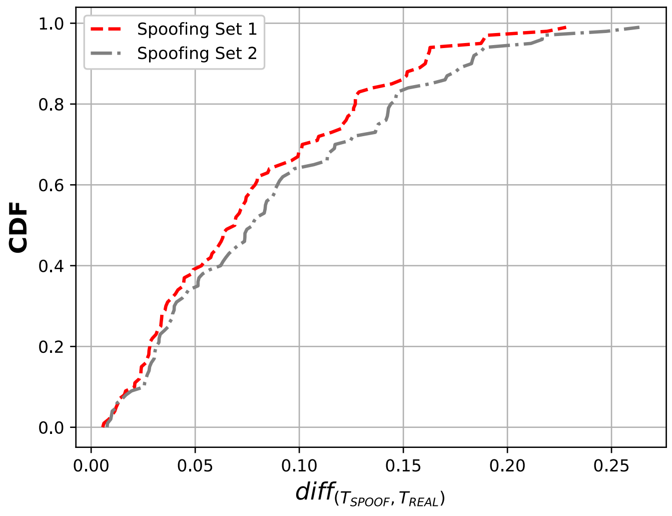


Fig. 14. The CDF of the difference between the “Spoofed” trajectory and the “Real” trajectory in the spoofing set.

sensor measurements of each trip as the input to the spoofing detection algorithm. That is, those trips are unspoofed trips. The objective of using the unspoofed set is to obtain the false alarm rate. We then select another 10000 trips and create two spoofing sets as follows. Each time, we pick two trips from the trip pool with the same length, and this action repeats 5000 times. Those 5000 pairs constitute the spoofed set in the experiment. For each pair of trips, we use T_{REAL} to denote the true trip and the other trip as the spoofed route, which is denoted as T_{SPOOF}. We keep the sensor measurements of the true route T_{REAL} and the GPS coordinates of the spoofed route T_{SPOOF}, and use them as the input to the spoofing detection algorithm. The second spoofing set is created in a similar manner. The only difference between these two sets is the driving pattern. The trip pairs in the first spoofing set have smaller turning angles, while the second spoofing set pairs contain wider or larger turning angles. Fig. 14 plots the CDF of the trajectory difference between the spoofing and the real routes, T_{REAL} and T_{SPOOF}, in both sets. We notice that trips in set 1 have a smaller trajectory difference. For the GPS spoofing detection, we set the parameter n_τ of the vehicle position estimator to 10 or 5 for speed and direction estimation.

We apply the proposed spoofing detector to two spoofing sets (marked in grey and orange) and the unspoofed set (marked in blue) in Fig. 15. Besides the detection accuracy, the false alarm rate is also an important factor affecting the system performance. The false alarm rate is the percentage of unspoofed trips being misclassified as spoofed trips by the detection algorithm. Certainly, a good threshold should achieve a high detection accuracy while suppressing false alarms.

In Fig. 13, we can find the average reconstruction error for the trip of different lengths in the unspoofed sets. That is, the reconstruction error in the figure is between the trajectory of sensor measurements and the trajectory of the unspoofed GPS coordinates of the same trip. The reconstructed errors can be expressed as a normal distribution, $E_{bdd} \sim \mathcal{N}(0.026, 0.002)$, as shown in Fig. 13. Hence, in order to suppress the false alarm caused by the reconstruction error, the threshold α should be set to 0.03. To see this, in the experiments, we change the threshold value, α, from 0.02 to 0.04. From the results in Fig. 15, setting α to 0.03 achieves a good balance where we can have an acceptable detection rate (88%) as well as a low false alarm rate (4%). Compared to with performance of the two spoofing sets, set 2 has a better detection rate because the larger turning angles of the trips give more useful information for the detection algorithm resulting in a higher trajectory difference.

In the real world scenario, the application server usually does not need to wait until the trip completes to validate the trip authenticity. Furthermore, the server may not even need the entire data from the origination point of the trip to detect spoofing. In other words, the application server can carry out dynamic spoofing detection based on the

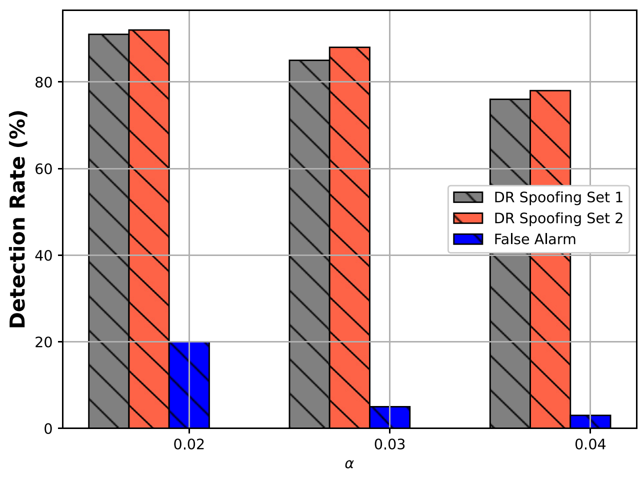


Fig. 15. Detection rates and false alarm rates for BDD-100K dataset.

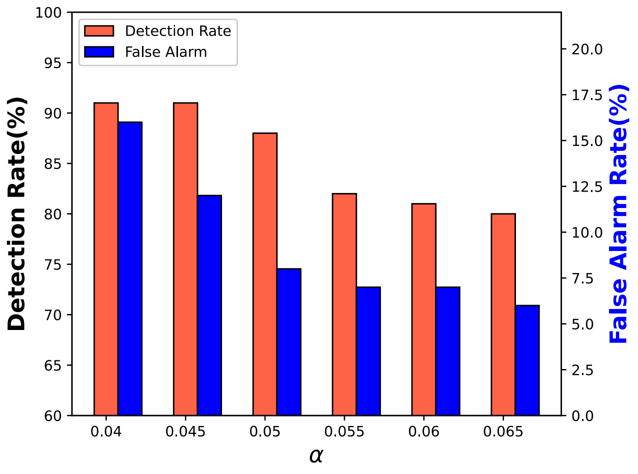


Fig. 16. Detection and false alarm rate for the custom dataset.

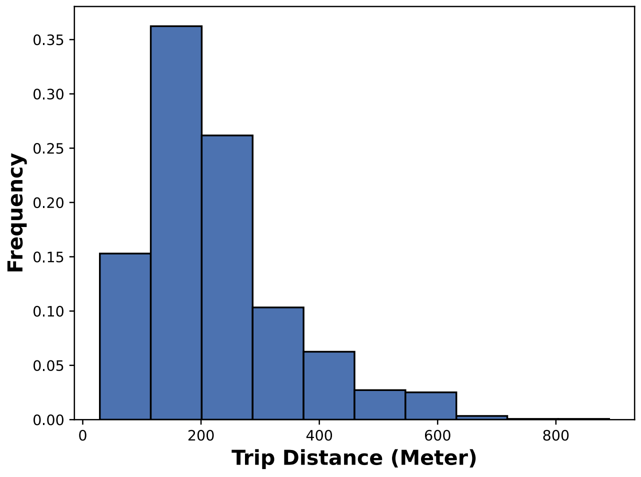


Fig. 18. The histogram of the average trip distance in the dataset for threat model case 2.

current GPS and motion sensor data within an interval from a recent time point to the current time, as discussed in Section 6.2. To evaluate spoofing detection in such scenarios, we divide 24-h driving data in the customized set into intervals of the following lengths: 40s, 60s, 120s, 180s, and 300s. Thus we have obtained about 2100, 1400, 700, and 200 trip segments, respectively, in each subset for the corresponding interval duration above (40–300s). The reconstructed errors of the unspoofed set can be expressed as a normal distribution, $E_{cus} \sim \mathcal{N}(0.037, 0.004)$, as shown in Fig. 13. Hence, to suppress the false alarm, the threshold α should be set to 0.045.

To test GPS spoofing, we use 80% of trip segments in each subset to create a spoofing trip set in a manner similar to the previous section. The remaining 20% of trip segments serve as an unspoofed trip set to obtain the false alarm.Fig. 16 shows the results for different values of α for all trip segments created previously. The black plot represents the detection accuracy, and the orange bar plot shows the false alarm rate when the threshold increases. The detection rate and false alarm vary with different α. Overall, $\alpha = 0.045$ or 0.05 results in a good balance between the detection rate and false alarm.

The BDD-100K dataset only includes short trip segments. Next, we evaluate the performance of the spoofing detector for the trip segments with longer durations in the custom dataset. The results are shown in Fig. 17. We observe that the detection rate increases with the increase of the travel distance.This is not surprising because a longer trip contains more information, which makes it possible to better detect the spoofing attack.

7.3.3. Detection for threat model case 2

To simulate the live spoofing attack as illustrated in Fig. 1, the attacker may initial the attack from the corner of an intersection or in the middle of a street. We decompose the subset of the custom dataset into edge level road segments, which are the traveling records between two connected edges on the road map. It is used to represent a special connection, i.e., the intersection. Fig. 18 shows the trip distribution of the dataset that includes 10,000 trips. We use 80% of them to create the spoofing set, and the remaining 20% remains as the unspoofed set. Finally, we create the spoofing set with 5,000 spoofing trips by randomly pairing one trip's GPS with the motion sensor measurements of other trips.

We apply Algorithm 4 to the datasets. The reconstruction errors of the unspoofed dataset can be expressed as a normal distribution $E_{inst} \sim \mathcal{N}(0.014, 0.004)$. Hence, we set the parameter β to 0.02. The trip progress indicates the percentage of distance the vehicle has traveled on the last road segment of the pre-planned path in a trip. The GPS spoofing is assumed to start at the beginning of this last road segment. From Fig. 19,

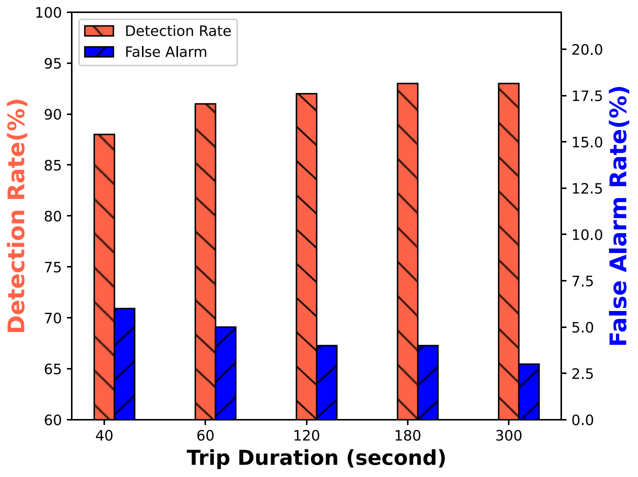


Fig. 17. Performance for trips with different durations in the custom dataset, $\alpha = 0.045$.

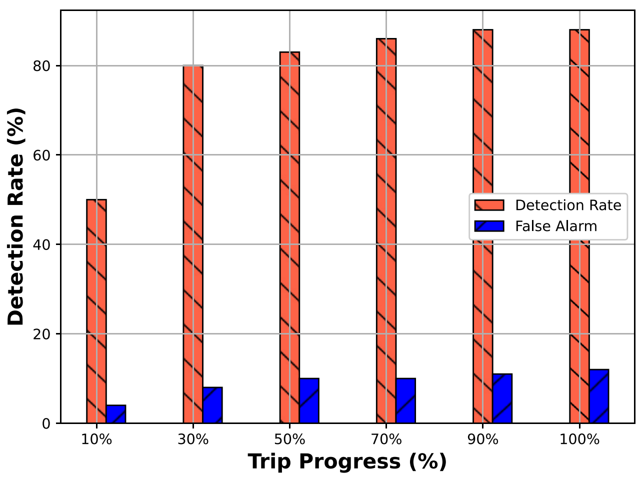


Fig. 19. The detection rate along with the progress of the trip when $\beta = 0.02$.

we observe that when the vehicle reaches 30% of a road segment, the detection accuracy is about 80% while the false alarm rate is less than 8%. The detection rate is further improved when the vehicle runs further on the road. Overall, our algorithm can quickly detect a spoofing attack within a short time after the launch of the GPS spoofing attack.

8. Conclusion

In this paper, we introduce a novel DeepPOSE framework for detecting GPS spoofing attacks. DeepPOSE includes two components: a vehicle position estimator and a spoofing detector. The vehicle position estimator integrates convolutional and sequence-to-sequence recurrent neural networks to capture the vehicle driving speed and direction from the motion sensor data. The vehicle speed and direction are then used to calculate the trajectory of the vehicle. The spoofing detector compares the trajectory with the one reconstructed from the GPS coordinates reported by the user to detect if there is a GPS spoofing attack. We have used two datasets to evaluate DeepPOSE. The experiment results indicate that DeepPOSE can effectively detect spoofing attacks in both cases of the threat model.

Conflict of interest and authorship conformation form

The authors whose names are listed immediately below report the following details of affiliation or involvement in an organization or entity with a financial or non-financial interest in the subject matter or materials discussed in this manuscript. Please specify the nature of the conflict on a separate sheet of paper if the space below is inadequate.

Acknowledgement

This work was supported in part by NSF under Grants CNS-1950704, CNS-1828593, and OAC-1829771, ONR under Grant N00014-20-1-2065, NSA under Grant H98230-21-1-0278, and the Commonwealth Cyber Initiative.

References

[1] Global GPS tracking devices market data survey report 2013-2025, URL https://brandessenceresearch.biz/Heavy-Industry/Global-GPS-Tracking-Devices-Market/Summary, (Accessed 10 May 2019)..
[2] B. Hofmann-Wellenhof, H. Lichtenegger, J. Collins, Global Positioning System: Theory and Practice, Springer Science & Business Media, 2012.
[3] K. Wang, S. Chen, A. Pan, Time and Position Spoofing with Open Source Projects, 2015.
[4] M.L. Psiaki, T.E. Humphreys, B. Stauffer, Attackers can spoof navigation signals without our knowledge. Here's how to fight back GPS lies, IEEE Spectrum 53 (8) (2016) 26–53.
[5] A.J. Kerns, D.P. Shepard, J.A. Bhatti, T.E. Humphreys, Unmanned aircraft capture and control via GPS spoofing, J. Field Robot. 31 (4) (2014) 617–636.
[6] J. Bhatti, T.E. Humphreys, Hostile control of ships via false GPS signals: demonstration and detection, NAVIGATION, J. Inst. Navig. 64 (1) (2017) 51–66.
[7] Tesla model 3 spoofed off the highway - regulus navigation system hack causes car to turn on its own, URL https://www.regulus.com/blog/tesla-model-3-spoofed-off-the-highway-regulus-researches-hack-navigation-system-causing-car-to-steer-off-road/, 2020 (Accessed 10 February 2020).
[8] M. L. Psiaki, T. E. Humphreys, Protecting GPS from spoofers is critical to the future of navigation, IEEE spectrum 10.
[9] Hacking a phone's GPS may have just got easier, URL https://www.forbes.com/sites/parmyolson/2015/08/07/gps-spoofing-hackers-defcon/, 2015 (Accessed 10 May 2019).
[10] T. Humphreys, Statement on the Vulnerability of Civil Unmanned Aerial Vehicles and Other Systems to Civil GPS Spoofing, University of Texas at Austin, 2012, pp. 1–16.
[11] K.C. Zeng, Y. Shu, S. Liu, Y. Dou, Y. Yang, A practical GPS location spoofing attack in road navigation scenario, in: Proceedings of International Workshop on Mobile Computing Systems and Applications, 2017, pp. 85–90.
[12] I. Sutskever, O. Vinyals, Q.V. Le, Sequence to sequence learning with neural networks, in: Proceedings of Advances in Neural Information Processing Systems (NeurIPS), 2014, pp. 3104–3112.
[13] J. Gehring, M. Auli, D. Grangier, D. Yarats, Y.N. Dauphin, Convolutional sequence to sequence learning, in: Proceedings of International Conference on Machine Learning (ICML), 2017, pp. 1243–1252.
[14] Opensource Software-Defined GPS Signal Simulator, https://github.com/osqzss/gps-sdr-sim. (Accessed 10 May 2019).
[15] M.Ö. Demir, G.K. Kurt, A.E. Pusane, On the limitations of GPS time-spoofing attacks, in: Proceedings of IEEE International Conference on Telecommunications and Signal Processing (TSP), 2020, pp. 313–316.
[16] S. Narain, A. Ranganathan, G. Noubir, Security of GPS/INS based on-road location tracking systems, in: Proceedings of IEEE Symposium on Security and Privacy (SP), 2019, pp. 587–601.
[17] D.P. Shepard, J.A. Bhatti, T.E. Humphreys, A.A. Fansler, Evaluation of smart grid and civilian uav vulnerability to gps spoofing attacks, in: Proceedings of Radionavigation Laboratory Conference, 2012.
[18] P. Papadimitratos, A. Jovanovic, GNSS-based positioning: attacks and countermeasures, in: Proceedings of IEEE Military Communications Conference (MILCOM), 2008, pp. 1–7.
[19] K. Wesson, M. Rothlisberger, T. Humphreys, Practical cryptographic civil GPS signal authentication, NAVIGATION, J. Inst. Navig. 59 (3) (2012) 177–193.
[20] M.G. Kuhn, An asymmetric security mechanism for navigation signals, in: Proceedings of International Workshop on Information Hiding, Springer, 2004, pp. 239–252.
[21] K. Jansen, M. Schäfer, D. Moser, V. Lenders, C. Pöpper, J. Schmitt, Crowd-GPS-sec: leveraging crowdsourcing to detect and localize GPS spoofing attacks, in: Proceedings of IEEE Symposium on Security and Privacy (SP), 2018, pp. 1018–1031.
[22] A. Eldosouky, A. Ferdowsi, W. Saad, Drones in distress: a game-theoretic countermeasure for protecting uavs against GPS spoofing, IEEE Internet of Things Journal 7 (4) (2019) 2840–2854.
[23] J.-H. Lee, K.-C. Kwon, D.-S. An, D.-S. Shim, GPS spoofing detection using accelerometers and performance analysis with probability of detection, Int. J. Contr. Autom. Syst. 13 (4) (2015) 951–959.
[24] S. Khanafseh, N. Roshan, S. Langel, F.-C. Chan, M. Joerger, B. Pervan, GPS spoofing detection using RAIM with INS coupling, in: Proceedings of IEEE Location and Navigation Symposium-PLANS, 2014, pp. 1232–1239.
[25] R.E. Ebner, R.A. Brown, Integrated GPS/inertial navigation apparatus providing improved heading estimates, uS Patent 5 (Aug. 12 1997) 657, 025.
[26] S. Xingjian, Z. Chen, H. Wang, D. Yeung, W. Wong, W. Woo, Convolutional LSTM network: a machine learning approach for precipitation nowcasting, in: Proceedings of Advances in Neural Information Processing Systems (NeurIPS), 2015, pp. 802–810.
[27] S. Yao, S. Hu, Y. Zhao, A. Zhang, T. Abdelzaher, Deepsense: a unified deep learning framework for time-series mobile sensing data processing, in: Proceedings of World Wide Web Conference, 2017, pp. 351–360.
[28] J. Feng, Y. Li, C. Zhang, F. Sun, F. Meng, A. Guo, D. Jin, Deepmove: predicting human mobility with attentional recurrent networks, in: Proceedings of the World Wide Web Conference, 2018, pp. 1459–1468.
[29] OpenStreetMap contributors, Planet dump, retrieved from, https://planet.osm.org, 2017, https://www.openstreetmap.org (Accessed 10 May 2019).
[30] Y. Wang, J. Yang, H. Liu, Y. Chen, M. Gruteser, R.P. Martin, Sensing vehicle dynamics for determining driver phone use, in: Proceedings of International Conference on Mobile Systems, Applications, and Services, 2013, pp. 41–54.
[31] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, T. Darrell, Bdd100k: a diverse driving dataset for heterogeneous multitask learning, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2020 (Accessed 10 May 2020).
[32] S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, in: Proceedings of International Conference on Machine Learning (ICML), 2015, pp. 448–456.
[33] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, Y. Huang, Map-matching for low-sampling-rate GPS trajectories, in: Proceedings of ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 2009, pp. 352–361.
[34] C. Chen, Y. Ding, X. Xie, S. Zhang, Z. Wang, L. Feng, Trajcompressor: an online map-matching-based trajectory compression framework leveraging vehicle heading direction and change, IEEE Trans. Intell. Transport. Syst. 21 (5) (2019) 2012–2028.
[35] D.J. Berndt, J. Clifford, Using dynamic time warping to find patterns in time series, in: Proceedings of KDD Workshop, 1994 (Accessed 10 May 2019).
[36] H. Musoff, P. Zarchan, Fundamentals of Kalman Filtering: a Practical Approach, American Institute of Aeronautics and Astronautics, 2009.