THE TWO FACES OF MOBILE SENSING

BY

TUO YU

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 2020

Urbana, Illinois

Doctoral Committee:

Professor Klara Nahrstedt, Chair
Professor Tarek F. Abdelzaher
Professor Romit Roy Choudhury
Professor Andrew T. Campbell, Dartmouth College
ABSTRACT

The recent popularization of mobile devices equipped with high-performance sensors has given rise to the fast development of mobile sensing technology. Mobile sensing applications analyze the signals generated by human activities and environment changes, and thus get a better understanding of the environment and human behaviors. Nowadays, researchers have developed diverse mobile sensing applications, which benefit people’s living, such as gesture recognition, vital sign monitoring, localization, and identification.

Mobile sensing has two faces. While benefiting people’s lives, its growing capability would also spawn new threats to security and privacy. Exploring the dual character of mobile sensing is challenging. On one hand, while the commercialization of new mobile devices enlarges the design space, it is challenging to design effective mobile sensing systems, which use less or cheaper sensors and achieve better performance or more functionalities. On the other hand, attackers can utilize the sensing strategies to track victims’ activities and cause privacy leakages. It is challenging to find the potential leakages, because mobile sensing attacks usually use side channels and target the information hidden in non-textual data.

To target the above challenges, I present the Mobile Sensing Application-Attack (MSAA) framework, a general model showing the structures of mobile sensing applications and attacks, and how the two faces are connected. MSAA reflects our principle of designing effective mobile sensing systems, i.e., we reduce the cost and improve the performance of current systems by exploring different sensors, various requirements for user/environment contexts, and different sensing algorithms. MSAA also shows our principle of exploring information leakages, i.e., we break a sensing system into basic components, and for each component we consider what user information could be extracted if data are leaked. I take handwriting input and indoor walking path tracking as examples, and show how we design effective mobile sensing techniques and also investigate their potential threats following MSAA. I design an audio-based handwriting input method for tiny mobile devices, which allows users to input words by writing on tables with fingers. Then, I explore the attacker’s capability of recognizing a victim’s handwriting content based on the handwriting sound. I also present an in-shoe force sensor-based indoor walking path tracking system, which enables smart shoes to locate users. Meanwhile, I show how likely a victim can be located if the foot force data are leaked to attackers. Our experiment results show that our applications can achieve satisfactory performance, and also confirm the threats of privacy leakage if they are maliciously used, which reveals the two faces of mobile sensing.
To my parents and my wife, for their love and support.
ACKNOWLEDGMENTS

First of all, I would like to express my sincere thanks to my advisor, Professor Klara Nahrstedt, for her continuous and invaluable guidance during my Ph.D. study. I am truly honored to join her research group and be supervised by her. Her immense knowledge, tremendously valuable advice and insightful comments have inspired my research, and helped me overcome numerous obstacles in my research projects. It is impossible to finish this thesis without her patient support.

I am also grateful to the other committee members, Professor Tarek F. Abdelzaher, Professor Romit Roy Choudhury, and Professor Andrew T. Campbell, for their patience and constructive suggestions. Their invaluable inputs helped me greatly improve my thesis. It is my great honor to have them in my committee.

I would like to thank the professors, researchers, and labmates who offered help during my study. In particular, I would like to thank Professor Carl A. Gunter, Associate Professor Philip B. Godfrey, Assistant Professor Haitham Hassaniah, Assistant Professor Haiming Jin, Associate Professor Lu Su, Wai-Tian Tan, Hongyang Li, Shannon Chen, Phuong V. Nguyen, He Huang, Zhenhuan Gao, Siting Chang, Wenyu Ren, Tarek Elgamal, Tianyuan Liu, Bo Chen, Zhe Yang, Hongpeng Guo, Eric Lee, Joumsup Park, and Mingyuan Wu for their inspiring advice and collaboration in my research projects.

I also want to thank all of my friends who helped and supported my Ph.D. study. Since it is impossible to mention all their names, I may only be able to list some of them: Yiran Zhao, Shuochao Yao, Huajie Shao, Sheng Shen, Rui Yang, Qingxi Li, Mo Dong, Tong Meng, Shen Li, and Dongxin Liu.

I would like to thank the National Science Foundation (under award number CNS-1330491 and OAC-1827126), the Ralph M. and Catherine V. Fisher Professorship, the Yumni and Maxine Pao Memorial Fellowship, and the Chirag Foundation Graduate Fellowship, for their financial support and assistance.

Last but most importantly, I would like to express my most sincere gratitude and appreciation to my wife Yuhang Wang, my father Xiren Yu, and my mother Yun Shi for supporting me spiritually throughout my life in the university. Their love is my most precious treasure.
TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION .............................................. 1
  1.1 Motivation and Challenges .................................. 1
  1.2 Thesis Statement ............................................. 3
  1.3 Thesis Overview ............................................. 3
  1.4 Thesis Organization ......................................... 7

CHAPTER 2 MOBILE SENSING APPLICATION-ATTACK FRAMEWORK .... 8
  2.1 Framework Structure ........................................ 8
  2.2 MSAA Framework as a Research Guideline ................. 11

CHAPTER 3 TABLEWRITE: AUDIO BASED HANDWRITING INPUT FOR
  TINY MOBILE DEVICES ............................................ 13
  3.1 Introduction ............................................... 13
  3.2 Related Work ............................................... 14
  3.3 TableWrite Overview ....................................... 15
  3.4 Word Detector .............................................. 17
  3.5 Hand Motion Tracker ...................................... 19
  3.6 Word Recognizer ............................................ 21
  3.7 Measurement and Evaluation .............................. 24
  3.8 Discussion ................................................ 26
  3.9 Conclusion ................................................ 27

CHAPTER 4 WRITINGHACKER: AUDIO BASED EAVESDROPPING OF HAND-
  WRITING VIA MOBILE DEVICES ................................. 28
  4.1 Introduction ............................................... 28
  4.2 Related Work ............................................... 30
  4.3 WritingHacker Overview ................................ 32
  4.4 Input Detection ............................................ 34
  4.5 Hand Motion Tracking .................................... 37
  4.6 Dictionary Filtering ..................................... 38
  4.7 Word Recognition ......................................... 39
  4.8 Implementation Details ................................... 44
  4.9 Measurement and Evaluation ............................ 45
  4.10 Discussion ............................................... 54
  4.11 Comparison between TableWrite and WritingHacker ....... 56
  4.12 Conclusion ............................................... 57
CHAPTER 1: INTRODUCTION

In recent years, the development of mobile devices, sensors, signal processing and machine learning has triggered the rapid evolution of the mobile sensing technology. The fundamental idea of mobile sensing is to get a better understanding of the environment and human behaviors by executing various sensing strategies in mobile systems. Specifically, different types of sensors embedded in various mobile devices, such as accelerometers in smartwatches, microphones in smartphones, and force sensors in smart shoes, are used to receive the signals generated by human activities or environment changes. By analyzing the signals, a mobile device can derive valuable information, such as a person’s gestures [1, 2], a user’s vital signs [3, 4, 5], and the location of the device [6, 7].

Meanwhile, just as the development of computer networks spawned the network attacks that threaten people’s privacy and finances, new mobile sensing techniques would also lead to unexpected threats. For example, while the motion sensors in a smartwatch can help recognize the user’s gestures [8, 9], if the motion data are leaked, an attacker could recognize the password that the user has input to a turntable password lock [10]. It is also shown that by using the data from a cellphone’s permissionless sensors, such as accelerometer and gyroscope, the attacker can reconstruct the victim’s secret PIN (personal identification number) for unlocking the cellphone [11, 12]. Therefore, besides the benefits brought by mobile sensing, people should also be aware of the danger caused by the abuse of the sensing techniques.

In a word, the fast evolving of mobile sensing has been benefiting people’s lives, but would also raise new threats to security and privacy. Researchers should investigate both the two sides of mobile sensing, which will be discussed in this thesis.

1.1 MOTIVATION AND CHALLENGES

The explosive growth of new mobile sensing techniques is associated with the commercialization of new mobile devices. One typical example is smartwatch. While the history of smartwatches can be traced back to 1980s, the recent large-scale commercialization of smartwatch starts around 2013 [13]. Companies such as Apple and Sony started to equip smartwatches with different kinds of sensors, such as gyroscopes and microphones. Although the sensors embedded in smartwatches are quite similar to those in smartphones, the new wearing position has given rise to the emergence of many new sensing techniques, such as finger, hand and arm gesture recognition and tracking [1, 8, 14], heart rate monitoring [4],
and biometric gait recognition [15]. Besides smartwatches, many other mobile devices have also been rising in recent years, such as smart glasses [16, 17, 18], smart rings [19, 20], and smart shoes [21, 22, 23]. Therefore, there are expansive spaces for researchers to develop new mobile sensing techniques on the new platforms. Compared with previous systems, new mobile sensing designs should achieve better effectiveness, i.e., achieving better performance or more functionalities while using less or cheaper sensors.

Exploring the capability of new mobile devices raises novel mobile sensing techniques that benefit people, but meanwhile also spawns the danger of privacy leakage. Attackers could target on a huge range of privacy information, such as a victim’s name, identity number, and current location. Some threats are not intuitive and hard to be aware of. For example, it has been shown that if the motion data measured by smartwatches were leaked, an attacker could recognize the content that victims input via keyboards [24]. Ignoring the potential security holes can lead to unexpected privacy exposure and economic losses. Thus, besides proposing new mobile sensing designs, we should also reveal the possible danger that they could cause, which help defend users more effectively.

Therefore, the challenge of mobile sensing system design includes the following two aspects.

- **Recognizing user behaviors with effective strategies:** Although there are plenty of sensors in various mobile devices that enlarge the design space, it is still challenging to design an effective mobile sensing system. The fundamental challenges come from two attitudes, i.e., using less or cheaper sensors to achieve better performance, and using the same sensor to achieve more functionalities. It is also challenging to find proper user/environment contexts that can be utilized to achieve these design goals. Since many mobile devices are still in their early stage, researchers need to keep exploring new ways to recognize user behaviors.

- **Revealing potential information leakages:** The danger of the measurement data leakage is usually non-intuitive. Different from the cyber attacks which often focus on literal data, the mobile sensing attacks usually use side channels and target the information hidden in raw sensor data. Thus, it is challenging to find the potential information leakage through the sensor data.

In this thesis, I present a general framework that describes the relationship between the two faces of mobile sensing, and use four system designs as examples to show the ways to address these challenges.
1.2 THESIS STATEMENT

I claim that the following thesis statement is true.

*Every mobile sensing design should be examined with respect to two faces: finding effective designs to recognize user behaviors, and finding potential threats of information leakages.*

1.3 THESIS OVERVIEW

In this thesis, I first present a framework that describes the relationship between the two faces of mobile sensing. Then, as shown in Figure 1.1, I take tiny mobile devices (such as smartwatch) and smart shoes as examples, and show how we can design effective sensing techniques that benefit people, and how we can reveal the potential danger that people should be aware of. This section provides an overview of each work.

![Figure 1.1: Our four approaching showing the two faces of mobile sensing.](image)

1.3.1 Mobile Sensing Application-Attack Framework

Currently, more and more mobile devices are being commercialized, which enlarges the design space and also the attack surface. It is harder to exam how a mobile sensing technique could lead to unexpected information leakages. Therefore, a guideline is necessary to clear the relationship between mobile sensing applications and attacks, and under what conditions the sensing technique can lead to privacy leakages. Moreover, we also need a guideline that shows how we can continuously improve the effectiveness of mobile sensing designs.

In this thesis, I present the Mobile Sensing Application-Attack (MSAA) framework, a general model that shows the structures of mobile sensing applications and attacks, and
how they are connected by signal and data leakages. MSAA shows that a mobile sensing application can be improved by finding the defects of the current design, and then trying various sensors and algorithms to overcome the defects. It is also necessary to investigate what user or environment contextual information is available, and how it can be used to enhance the application. The design focus should be using less or cheaper sensors to achieve better performance or more functionalities. On the other hand, we show that the information leakage could happen in each stage of a mobile sensing application. Specifically, we should consider the physical signal leakage, the raw data leakage, the indirect privacy leakage, and the direct privacy leakage. These leakages enable attackers to extract the user information by using the techniques that are similar to those used by normal mobile sensing applications.

The MSAA framework provides a guideline that helps improve the effectiveness of a mobile sensing design, and also helps explore the possible threats brought by a mobile sensing technique. Following MSAA, we reveal that there are still gaps in the mobile sensing research. While there are approaches that recognize users’ handwriting via audio signals, it is still not clear if the leakage of the handwriting sound alone can expose the handwriting content. While smart shoes are capable of tracking users, it has not been explored if the data leakage from smart shoes exposes users’ physical locations. Based on MSAA, I first design two mobile sensing applications for the audio-based handwriting input and the smart shoe-based user tracking to illustrate how we design effective systems, and then fill the gaps by presenting two corresponding attack methods that evaluate the aforementioned information leakages.

1.3.2 Audio-based Handwriting Input for Tiny Mobile Devices

The popularization of tiny mobile devices has raised the problem that it is hard to efficiently input messages via tiny keyboards or touch screens. The widely used voice input schemes [25, 26] are sometimes inconvenient in quiet-kept scenarios such as libraries and hospitals. It has been shown that smartwatches can track the motion of users’ hands, and thus enable users to write on surfaces with fingers [1]. However, this scheme is not convenient for everyone because usually people do not wear watches on their dominant hands. Soundwrite [27] leverages the embedded microphones in smartphones to record the sound of handwriting on tables and recognizes the input text. However, a user needs to retrain Soundwrite before each use. There are also handwriting input methods based on image-based recognition techniques [28, 29]. However, these methods require large touch screens or computational-expensive cameras, which raise the cost of tiny mobile devices.

To make the input method easier to use, in this work [30], I present TableWrite, an audio-based handwriting input scheme which allows users to input words to mobile devices by writ-
ing on tables with their fingers. The proposed scheme uses widely-embedded microphones and speakers, and does not require any retraining phase once it has been trained. Thus, the sensor cost is reduced, and the extra contextual information required by TableWrite is only the pre-knowledge about the user’s handwriting style, which is acceptable. The prototype system’s experimental results show that the average accuracy of word recognition is around 90%-95% in lab environments, which validates the effectiveness of TableWrite.

1.3.3 Audio-based Handwriting Eavesdropping

After introducing TableWrite, I investigate the case that the audio based input technique is maliciously used. When filling out privacy-related forms in public places such as hospitals or clinics, people usually are not aware that the sound of their handwriting leaks personal information. An attacker can record the sound of handwriting with a microphone, and it is possible that the content of handwriting can be recovered. To expose this danger, we explore the possibility of eavesdropping on handwriting based on the audio signals. Although handwriting on desks has been considered as a new text entry method, the previous approaches are not applicable for the eavesdropping attack. The methods in [31] and [1] enable users to write on surfaces by using smartphones as pens or by wearing smartwatches on wrists. However, these methods assume that users have direct contact with some mobile devices, and thus cannot be used for attack because attackers usually have no access to victims’ devices. While Soundwrite [27] can recognize handwritten words based on audio signals, once the system has been trained, its performance highly depends on the location of handwriting, which makes eavesdropping attack impossible.

To reveal the capacity of the audio-based side-channel attack on handwriting, in these works [32, 33], I present WritingHacker, an audio-based eavesdropping proof-of-concept system which eavesdrops on handwriting via mobile devices. I show that, by keeping a mobile device touching the desk which is used by the victim, the attacker can record the sound of the victim’s handwriting. Then the system can provide a word-level estimate for the content of the handwriting by recognizing the unique patterns of each letter. WritingHacker requires no direct contact with the victim, and is insensitive to the writing position. The experiment results reveal the danger of privacy leakage through the sound of handwriting.

1.3.4 Indoor Walking Tracking Using Smart Shoes

Indoor walking tracking and localization have been popular research areas. Many of the previous approaches are based on wireless anchors [34, 35, 36, 37, 38, 39, 40, 41, 42, 43]. For
instance, Wi-Fi access points generate specific signal strength distributions in indoor space, and mobile devices such as cellphones measure the signal strength to locate themselves [44, 45, 46, 47, 48, 49]. However, the anchors need to be pre-installed, and site surveys are usually necessary, which leads to extra cost. To solve this problem, some previous works proposed inertial sensor-based methods [50, 6, 51]. The accelerometers, gyroscopes and compasses in cellphones or Inertial Measurement Units (IMUs) are used to measure the motion of users and thus estimate the walking paths. However, the electronic compasses can be interfered by magnetic field changes [52, 53, 54], especially for the scenarios where the magnetic field changes significantly [55, 56, 57].

In recent years, the rapid evolution of mobile sensing technology has triggered the rise of smart shoes [22, 21, 58], the devices that can measure the motion of users’ feet and analyze their activities. One typical design of smart shoes is based on force or piezoelectric sensors [22, 59]. In contrast with the previous approaches, in this work [60], I explore the capability of smart shoes, and prove that it is possible to track the walking paths of users in indoor spaces based on the force changes in shoes. I present ShoesLoc, an indoor walking path tracking method based on in-shoe force sensors. ShoesLoc eliminates the cost of the anchor installation and site survey. Moreover, the in-shoe force sensors are hardly interfered by magnetic field changes. Compared with traditional indoor tracking technologies, ShoesLoc is more effective because it does not require the installation of wireless anchors, and has good robustness to environment changes such as the magnetic interference. The extra contexts needed are the user’s training data and the floor map of the walking region, which are available to the application with low cost.

1.3.5 Victim Localization through Force Sensors in Smart Shoes

After introducing ShoesLoc, I consider the case that the force data from smart shoes are leaked to attackers. While the past few years have witnessed the rise of smart shoes, people are still not aware of the possible privacy leakage from smart shoes. It is necessary to explore the possibility of locating an indoor victim based on the force signals leaked from smart shoes. Some previous works have shown the capability of in-shoe IMUs for indoor walking path tracking [61, 62, 63]. However, they implicitly require the location and the structure of the building where the user is walking in, which makes these approaches inapplicable to the attack scenario, because the attacker usually does not know where the building is.

My work [64] is based on the assumption that the attacker has got the force data by hacking the user’s device or cloud servers. I present ShoesHacker, an attack scheme that
reconstructs the corridor map of the building that the victim walks in based on force data only. The corridor map enables the attacker to recognize the building, and thus locate the victim on a global map. ShoesHacker requires no training data from the victim, and needs no knowledge about the building structure. This work reveals the danger of the location privacy leakage through the foot force data.

1.4 THESIS ORGANIZATION

In the following chapters, I will introduce the MSAA framework and the technique details of each of the four aforementioned works. To be more specific,

- In Chapter 2, to provide a guideline that help investigate the two faces of mobile sensing, I present the MSAA framework, which shows the relationship between mobile sensing applications and attacks, and how we can find the gaps in the current mobile sensing research.

- In Chapter 3, to address the “fat finger” problem of tiny mobile devices, I propose an audio-based handwriting input scheme named TableWrite, which allows users to input words to mobile devices by writing on tables with their fingers.

- In Chapter 4, I investigate the case that the audio-based handwriting input scheme is maliciously used by attackers. To show the danger of the privacy leakage through the sound of handwriting, I propose an eavesdropping attack method called WritingHacker, which recognizes the content of the victim’s handwriting based on its audio signal.

- In Chapter 5, to investigate the capability of smart shoes for indoor user tracking, I design ShoesLoc, an indoor walking path tracking method which is based on in-shoe force sensors.

- In Chapter 6, to expose the danger of the location leakage through foot force data, I develop an attack method based on in-shoe force sensors, named ShoesHacker. ShoesHacker can reconstruct the corridor map of the building that the victim walks in, and even locate the building and the victim on a global map.

Finally, in Chapter 7, I conclude the thesis and provide some future research directions.
In this chapter, we present the MSAA framework, a general model that shows the connection between the two faces of mobile sensing.

2.1 FRAMEWORK STRUCTURE

The structure of MSAA is illustrated in Figure 2.1. It contains the part of mobile sensing application and the part of mobile sensing attack.

2.1.1 Structure of Mobile Sensing Application

As shown in Figure 2.1, a typical mobile sensing application contains three components, i.e., sensors, sensing algorithm, and result presentation. The environment changes or user activities generate or modulate certain signals, which contain the valuable information. The signals are received and measured by the sensors embedded in mobile devices. The sensors output raw data, and the devices process the data locally or upload them to servers, depending on the application scenario. The raw data could also be partially processed locally before being uploaded. The sensing algorithm, running on mobile devices or servers, extracts the information of interest. Finally, the information is presented to the user or the service provider.
Besides the signal source, the user or environment context is another important information source. The contextual information can be the prior knowledge about the user’s habits or the settings of the application environment, which are critical for the sensing algorithm design. One typical type of contextual information is the training data provided by the user. The cost of collecting training data is usually low, but the data enable the system to apply machine learning algorithms, which could significantly improve the performance. Another typical type of contextual information is the data collected by the site survey in the application environment. For example, many wireless signal fingerprint-based indoor localization methods need offline site surveys to determine the fingerprint at each location. However, since site surveys require a lot of labor, they should be avoided if possible. Therefore, it is important to consider the trade-off between the contribution of the contextual information and the cost of collecting it. Requiring too many contexts weakens the practicability of the application.

The MSAA framework also includes the workflow of the effective system design. As shown by the dashed lines in Figure 2.1, we first reveal and investigate the defects of the current system. The defects can come from many aspects, such as low recognition accuracy, bad robustness in certain situations, and high setup cost. We then explore different sensors and algorithms to overcome the defects. Meanwhile, depending on the selected sensor, we should also consider what kind of contextual information is needed. If the cost of getting the context is low but the performance gain it brings is huge, the context should be assigned as an extra input to the system. Otherwise, the context should not be required. With this constraint on the contextual information, the design goal should be using less or cheaper sensors to achieve better performance or more functionalities. For example, many handwriting input methods require large touch screens or cameras, which are expensive for tiny mobile devices. In TableWrite, we use microphones as sensors instead to implement an input method, which significantly lowers the sensor cost. In ShoesLoc, we design new sensing algorithms to give force sensors the new functionality of indoor walking path tracking.

2.1.2 Structure of Mobile Sensing Attack

Unfortunately, the nature of the mobile sensing strategies provides a large attack surface to attackers. If not well protected, the user information could leak from the signal source, the raw data, and the processed (or partially-processed) data. The structure of a typical mobile sensing attack is similar to that of an application, but the attack can start from any data leakage point.

If the attacker has physical access to the signal source, the attacker’s mobile sensors can
directly record the signal. For example, if a microphone is deployed near to the victim, the attacker can record the voice of the victim, and launch the voice impersonation attack [65, 66, 67].

In more common cases, the attacker gets the permission of the sensor usage in the victim’s device via malware, and receives the raw sensor data. If the raw data are stored in cloud servers, cyber attacks can also cause date leakages. Then, the attacker runs the sensing algorithm, which extracts the victim’s private information. Many attack strategies have been proposed based on the raw data leakage, such as recognizing the input to a nearby keyboard based on the accelerometer output of a cellphone [68], and detecting the victim’s age group based on the motion sensor readings when the victim is holding and touching a smartphone [69].

It is also possible that the attacker gets the access to the processed data stored in the device or the cloud. Even if the processed data do not directly contain the user information, it is still possible that the attacker can extract other private information via side channels, which is illustrated by the indirect privacy leakage in Figure 2.1. What leak through the indirect privacy leakage are the user behavior patterns instead of the signal data. Therefore, compared with the physical signal leakage and the raw data leakage, the indirect privacy leakage exposes user information at a higher level. One typical example is inferring the driving trajectory of a vehicle by monitoring the audio on/off status of a cellphone, which reflects the driving instructions played by navigation applications [70]. The driving instructions are the processed data of the navigation applications, but they still contain the victim’s location information that is targeted by attackers. If the leaked processed data already contain the information of interest, the attack directly succeeds through the direct privacy leakage in Figure 2.1. This is usually based on malware and cyber attacks, which are outside the scope of mobile sensing.

Similar to the mobile sensing applications, attackers can also make use of the contextual information. However, the availability of contexts is tightly constrained. For example, it is extremely hard for attackers to directly get labeled training data from victims. To keep the feasibility, attack designs should require as little contextual information as possible.

Besides the attack workflows shown in Figure 2.1, there are some other attack methods such as spoofing, i.e., creating fake data or signals and injecting them into the mobile sensing applications. Since these attacks focus more on violating system or network security instead of using mobile sensing techniques, they are out of our scope.
2.2 MSAA FRAMEWORK AS A RESEARCH GUIDELINE

The MSAA framework shows the two sides of mobile sensing. While mobile sensing applications can benefit people, their data and algorithms could also be utilized by attackers. For each mobile sensing technique, researchers should investigate both the two sides. Figure 2.2 illustrates the current research on mobile sensing with respect to different mobile devices. It is clearly shown that, for each sensing application, there are corresponding attack strategies that target on the same signal or user information type. Note that providing a complete approach survey is out of our scope, and Figure 2.2 only gives some examples for illustration.

As illustrated in Figure 2.2, since the newly-commercialized sensors and devices keep bringing new possibilities to mobile sensing techniques, there are always new research approaches that improve the previous designs. MSAA guides us to improve the effectiveness of our designs by utilizing these new resources. For example, the previous motion sensor-based walking path tracking systems can be interfered by magnetic field changes. To overcome this defect and improve the robustness, in ShoesLoc we use the force sensors in the newly-commercialized smart shoes, which have good resistance to the magnetic field interference. Moreover, our algorithm design eliminates the site survey, which reduces the setup cost. To achieve better tracking accuracy, we utilize the contexts of training data and floor maps. The performance gain outweighs the context cost, because the training data can be directly collected from users and the floor maps are usually available online. Therefore, ShoesLoc can achieve better effectiveness compared with the previous approaches. MSAA reflects our principle of designing effective mobile sensing systems, i.e., we reduce the cost and improve the performance of current systems by exploring different sensors, various requirements for user/environment contexts, and different sensing algorithms.

Figure 2.2 also shows the gaps left in the current mobile sensing research. For smartwatches
and some other small mobile devices, researchers have proposed some handwriting input methods that do not use any touch screen. However, to the best of our knowledge, currently there is no work investigating the possibility of inferring the handwriting content. Therefore, a gap exists in the research on the handwriting input, because the attack side shown in the MSAA framework is not explored. Similarly, since smart shoes are newly commercialized products, although there are smart shoe-based approaches such as indoor positioning and floor plan reconstruction, the research on smart shoe-based attacks is still missing. The threats that can be caused by the leakage of shoe sensor data have not been investigated.

The MSAA framework provides a research guideline that shows the connection between the two sides of mobile sensing, and helps researchers find and close the gaps. For example, for the audio signal of handwriting, while we have proposed TableWrite as a new input method based on handwriting sound, we should also consider the case that the audio signal is also received by the attacker, or the data recorded or processed by the user’s device are leaked to the attacker. In this thesis, we focus on the case of the physical signal leakage, and present WritingHacker to reveal the possible danger. Another example is for smart shoes. By presenting ShoesLoc, we first show that the in-shoe force sensors are capable of locating users. Then, following MSAA, we consider the case that the raw force data are leaked to the attacker, and investigate if the attacker can locate the victim. Our approach, named ShoesHacker, closes the gap of the attacks on smart shoes, and can be considered as both a location inferring and a floor plan inferring approach. The discovery of these gaps also reflects our principle of exploring information leakages, i.e., we break a sensing system into basic components, and for each component we consider what user information could be extracted if data are leaked.

There are still empty spaces left in the research field. For instance, while there are works considering the gait spoofing based on cameras and on-hip accelerometers [71, 72], it is still not clear that whether the data collected by smart shoes are sufficient for gait spoofing. Filling these empty spaces is left for our future work.
CHAPTER 3: TABLEWRITE: AUDIO BASED HANDWRITING INPUT FOR TINY MOBILE DEVICES

3.1 INTRODUCTION

The explosive development of tiny mobile devices (e.g., smart watches, smart rings) has raised the “fat finger” problem: because of the shrinking device interfaces, it is hard to efficiently input messages into these devices by keystroking. Although the voice input scheme has been applied in many devices, it is sometimes inconvenient in quiet-kept scenarios such as libraries and hospitals, and would also cause privacy leakage. Therefore, people need a more convenient input method for tiny mobile devices.

To solve this problem, some previous works have proposed handwriting on tables as a new text entry method for mobile devices. The approach in [1] uses smartwatches to track the motion of users’ hands, and thus enables users to write on surfaces with fingers. However, this scheme is not convenient for everyone because usually people do not wear watches on their dominant hands. In [27], the authors leverage the embedded microphones in smartphones to record the sound of handwriting on tables and recognize the input text. However, a user needs to train the system before each use, and once the system has been trained, its performance highly depends on the location of handwriting. A comfortable and efficient handwriting input scheme should not require users to carry devices with their dominant hands, and should not require a training phase before each use. Also, the recognition accuracy should not highly depend on the location of handwriting.

In this chapter, we present TableWrite, an audio-based handwriting input scheme which allows users to input words to mobile devices by writing on tables with their fingers. Once trained, TableWrite does not require any retraining phase. Our method is based on the basic assumption that the handwriting of users is print-style, instead of joined-up writing. We consider this assumption reasonable because it is easy for users to write letters separately.

The main idea of our method is that, when a user wants to write on a surface with a finger, he/she can keep a mobile device such as a smartwatch or a smartphone touching the surface, so that the device can record the sound of the handwriting clearly. Based on pre-collected training data, the device can recognize the words by extracting features from the sound sequence and recognizing the unique patterns of each letter.

The main challenge is that, the training data should be collected only once. The existing audio-based handwriting recognition methods require retraining before each use, because the diversity of audio signal’s multipath propagation is highly impacted by the material of surfaces and the location of handwriting. To solve this problem, we leverage the common
patterns of people’s print-style writing. For instance, a user writes letter ‘F’ with three strokes, and letter ‘O’ with one stroke, which is not impacted by the surface material or the handwriting location. In TableWrite, based on the user’s writing habit, we classify letters into clusters according to stroke numbers. For each cluster, we implement multiclass Support Vector Machines (SVMs) to distinguish letters. To train the SVMs, the user needs to provide the training data only once, and no retraining is needed before each use. Based on the stroke number information, we further apply a dictionary filter to narrow down the word search range and also provide the linguistic word-level correction. Moreover, inspired by the gesture tracking technique, we apply an audio signal reflection based method to track the motion of the user’s hand, which provides valuable features if the relative angle between the device and the handwriting is roughly consistent.

The main contributions of our work are:

- We design a new audio-based handwriting input method that does not require any retraining phase.

- To solve the lack of retraining data, we use the stroke number as a common handwriting pattern, and apply the word filtering and the hand motion tracking technique to further improve the recognition accuracy.

- We implement a prototype system and conduct evaluations. Experiments confirm that TableWrite achieves good accuracy while providing better convenience.

In the rest of this chapter, we first introduce the related work in Section 3.2. Then, we present an overview of the system architecture in Section 3.3. Then we successively introduce the methods used by TableWrite in Section 3.4, 3.5, and 3.6. We evaluate the performance of TableWrite in Section 3.7, have further discussion in Section 3.8, and conclude this chapter in Section 3.9.

3.2 RELATED WORK

*Handwriting recognition:* The automatic handwriting recognition techniques can be classified as offline and online recognition [73]. The former is usually based on the images of handwriting, and the latter also utilizes sensor data such as stylus positions and temporal information during the writing [73, 74].

In this work, we do not apply the image-based recognition techniques such as [28, 29], because not all the mobile devices are equipped with large touch screens or computational-expensive cameras. In contrast, microphones and speakers have been widely embedded
in mobile devices. Therefore, we utilize microphones and speakers to realize handwriting recognition, so that our technique can be easily applied to commercial mobile devices.

**Handwriting on tables as an input method:** There have been some works focusing on enabling handwriting on tables to be a new text entry method for mobile devices. The system named GyroPen in [31] enables users to use smartphones as pens to write on tables, and uses gyroscopes and accelerometers to track the trajectories. This method is not suitable for other mobile devices such as smartwatches because it requires users to hold their devices in hands. The system in [1] enables a user to write on surfaces with fingers. A smartwatch worn on the user’s wrist records the accelerometer signals of user’s hand, which can be utilized to recognize the user’s handwriting with the algorithm of gesture recognition. Some other approaches also recognize handwriting gestures with smartwatches [75, 76]. However, these approaches require users to wear watches on their dominant hands. Using the system from [27], after executing a training phase, a user can write with a finger on a table, and the microphone in a smartphone records the sound of handwriting. However, once the location of the smartphone is changed, the training phase must be repeated, which is inconvenient for users.

In this work, we focus on a more practical case, where users do not need to hold or wear the mobile device by their dominant hands. The training data should be collected only once, and the method’s performance should not be highly impacted by the surface material or the writing location.

### 3.3 TABLEWRITE OVERVIEW

In this section, we first describe the basic assumptions of our design, and then give an overview of the system architecture and the workflow of our method.

#### 3.3.1 Basic Assumptions

Our system is designed based on the basic assumption that the user’s handwriting is print-style, instead of joined-up writing. We consider this assumption reasonable, because most handwriting input methods encourage users to write in print style for higher accuracy. We recognize handwriting in units of words and focus on the word-level recognition. With our input method, for each written word, the device can display a list of candidates for the user to select. We notice that there are two different ways to write with fingers: using fingernail or finger pulp. In this work we require users to write with their fingernails to generate stronger audio signals. During the writing, the relative angle between the device and the
user’s writing hand should be roughly similar to that during the training data collection. This requirement does not lead to extra user efforts, because people’s writing poses are usually constant due to their writing habits.

Except the user’s writing hand, the distance from the device to the other moving objects should be at least longer than 40 cm. Nearby moving objects could interfere parts of our system, but TableWrite can still achieve good performance, as will be discussed in Section 3.8.1. The noise level of the environment cannot be too high, otherwise the handwriting signal is masked by the noise. We focus on the typical writing scenarios such as meeting rooms, where the average noise level is usually lower than 50 dB [77].

3.3.2 System Architecture

As shown in Figure 3.1, our system mainly consists of the following components.

*Speaker and recorder*: The speaker plays continuous wave signals with high frequencies (17-19.45 kHz), which is nearly inaudible to humans. The recorder receives the signal reflected by the user’s hand and the sound of handwriting with the embedded microphone, and sends the sound stream to the word detector and the hand motion tracker.

*Word detector*: It receives the sound stream and extracts the sound fragment and the stroke number for each letter in a word. The results are sent to the word recognizer. The hand motion tracker also receives the start and end time point for each letter from this component.

*Hand motion tracker*: By analyzing the reflected audio signal, it tracks the tiny distance change between the user’s finger and the device. The curve of the distance change for each letter is stored as a motion fragment, which is sent to the word recognizer.

*Word recognizer*: It first finds all the candidate words from a dictionary based on the stroke information. Then, it calculates the similarity between the recorded fragments and each candidate word. The best matches are output.
Before the first time the system can be used, the user should provide labeled training data. The training data include the sound and motion fragments and the number of strokes for each letter, which are used to train the SVMs. Since the training phase is conducted only once, it does not impact user experience. To maintain the performance of the hand motion tracker, the relative angle between the device and the hand during use should be similar to that during the training phase. We consider it reasonable because it is easy for the user to keep such angle roughly consistent, e.g., always put the cellphone vertically to the line of handwriting.

In this chapter, we take the capital letters ‘A’-‘Z’ as examples to show how TableWrite works. However, our technique can be easily applied to other characters. Next we introduce the main components in detail, respectively.

3.4 WORD DETECTOR

The design of the word detector is based on signal processing in time domain, which consists of two steps, namely stroke detection and letter detection. First, we introduce the definition of a stroke and the concept of letter clusters.

3.4.1 Definition of Stroke and Letter Cluster

Stroke: In our work, we define a stroke as a mark made by a writing instrument with a single touch to a surface. This means that we consider the letter ‘B’ as a two-stroke letter, even if it has multiple turning points.

Letter Clusters: Based on our basic assumption about print-style writing, we divide the capital letters into three clusters according to the stroke number:

\[ C_1 = \{ 'C', 'G', 'L', 'M', 'O', 'S', 'U', 'V', 'W', 'Z' \}, \]
\[ C_2 = \{ 'B', 'D', 'J', 'K', 'P', 'Q', 'R', 'T', 'X' \}, \]
\[ C_3 = \{ 'A', 'E', 'F', 'H', 'I', 'N', 'Y' \}. \]

Thus each letter in cluster \( C_u \) has \( u \) strokes (\( u = 1, 2, 3 \)). Note that the clusters can be adjusted based on the writing habits of different users.
3.4.2 Stroke Detection

Firstly, to filter out the signal generated by the speaker and reflected by the hand from the sound stream, we use a FIR (Finite Impulse Response) low pass filter with the passband frequency of 4 kHz and the stopband frequency of 6 kHz. Then, we detect each stroke in the continuous sound stream. A naive method is setting a constant threshold and once the sound signal magnitude or instantaneous power exceeds the threshold, a stroke is detected. However, since noise levels are usually different in different environments, and the average noise power varies over time, it is hard to assign such a threshold. We use a method similar to the Constant False Alarm Rate (CFAR) algorithm [78] to solve this problem.

Let us assume that the noise power follows Gaussian distribution. Its average power and standard deviation at time $t$ are denoted by $\mu(t)$ and $\sigma(t)$. We denote the amplitude of the received audio signal by a discrete time series $x(t)$, and use a sliding window of size $W$ to calculate the average noise power at time $t$ [79]:

$$\mu(t) = \frac{1}{W}A(t) + (1 - \frac{1}{W})\mu(t-1), \quad (3.1)$$

where

$$A(t) = \frac{1}{W} \sum_{k=t-W+1}^{t} |x(k)|^2. \quad (3.2)$$

The standard deviation at time $t$ is calculated in a similar way:

$$\sigma(t) = \frac{1}{W}B(t) + (1 - \frac{1}{W})\sigma(t-1), \quad (3.3)$$

where

$$B(t) = \sqrt{\frac{1}{W} \sum_{k=t-W+1}^{t} (|x(k)|^2 - A(t))^2}. \quad (3.4)$$

A potential start point of a stroke is detected if

$$|x(t)|^2 > \mu(t) + \alpha_1\sigma(t), \quad (3.5)$$

and a potential end point of a stroke is detected if

$$|x(t)|^2 < \alpha_2\bar{\mu}, \quad (3.6)$$

where $\alpha_1$ and $\alpha_2$ are two constant parameters. $\bar{\mu}$ is the average noise power when there is no input. It can be updated by measuring the average noise power between each two adjacent
detected letters. The reason why we do not use $|x(t)|^2 > \alpha_2 \mu$ for start point detection is that it is impossible to measure $\mu$ if the start point has not been detected.

A stroke is recognized if the interval between the potential start and end point is longer than a constant threshold $\beta_1$, as shown in Figure 3.2(a). In this way we avoid the impact of burst noise. If the interval between two detected strokes is shorter than another constant threshold $\beta_2$, we combine these two strokes as a single stroke.

![Stroke and Letter Detection](image)

(a) Stroke detection.  
(b) Letter detection.

Figure 3.2: Example for letter detection.

### 3.4.3 Letter Detection

The second step is to extract the sound fragment for each letter. A stroke's start point is also a letter's start point if it is the first start point after the end point of the previous letter. If there is no stroke detected in constant duration $\beta_3$ after the end point of a stroke, the end point is also the end point of the letter. One example is shown in Figure 3.2(b).

Then the measured sound signal between the start and end point of the letter is stored as a sound fragment. Assume that the word written by the user has $N$ letters. We use $s_k$ to denote the stroke number of the $k$th letter in the word. For example, word “SEA” has $N = 3$, and $s_1 = 1$ for letter ‘S’ $\in C_1$. Then $\{s_k|k \in [1, N]\}$ and the sound fragments are passed to the word recognizer. The start and end time point for each letter is also sent to the hand motion tracker to help crop the motion fragment for each letter.

### 3.5 HAND MOTION TRACKER

The recent development in device-free gesture tracking technique [80, 81, 82] shows that it is possible to recognize the content of handwriting via the motion trajectory of the user’s hand. In [83], a device-free gesture tracking scheme named LLAP (Low-Latency Acoustic
Phase) utilizes a speaker to generate an audio signal, and tracks the motion of a user’s hand by calculating the phase shift of the signal reflected by the hand. It is possible to use this scheme to track a user’s hand motion and thus recognize the handwriting. However, at least two microphones are needed to track the handwriting on a 2-D surface. Many devices such as smartwatches are equipped with only one microphone, and thus do not have such redundancy. Therefore, to ensure the wide applicability of TableWrite, only one microphone can be used. We apply LLAP only to track the 1-D distance change between the device and the hand, and will show that this information still helps to enhance the performance of TableWrite.

A speaker on the device is used to generate continuous wave signal with a certain frequency. When the wave is reflected by the hand, the motion of the hand causes a continuous phase shift. Thus, the accumulated phase shift of the reflected signal is related to the distance that the hand has moved by. Note that the sound of handwriting can be filtered out with CIC (Cascaded Integrator Comb) filters. We use the microphone to receive the reflected signal, and measure this distance change. During the implementation, multiple frequencies are synchronously used to mitigate the multi-path effect. We use 17-19.45 kHz with the interval 350 Hz to avoid audible noises. The sampling rate is 48 kHz. Please refer to [83] for details.

Since the start and end points of each letter have been determined by the word detector, we can easily crop the curve of distance change for each letter from the continuous measured result. Figure 3.3 shows the example of hand tracking results for ‘B’, ‘C’ and ‘N’. The three letters are distinguishable based on the cropped curves. The cropped curve for each letter is output as a motion fragment, which is further processed by the word recognizer.

![Graphs](image-url)

Figure 3.3: Examples for motion fragments.
3.6 WORD RECOGNIZER

This component finds candidate words based on the information of stroke number. Then it extracts the features of the sound and motion fragments for each letter, and recognizes the whole word according to the features, the trained SVMs, and the candidate word list.

3.6.1 Dictionary Filter

One of the key design points of TableWrite is reducing the size of candidate word set for the word classifier. We use a dictionary database to store the commonly used words and a filter to extract the validated words whose stroke numbers match with the input set of stroke numbers \{s_k\}.

Specifically, the dictionary filter receives a set of stroke numbers \{s_k\} from the word detector. Then it finds all the words that match with \{s_k\} from the dictionary. The length of a matched word should be N, and the stroke number of the kth letter should be s_k. For instance, if \{s_1 = 1, s_2 = 3, s_3 = 3\}, the words extracted from the dictionary should be \{“SHE”, “SEE”, ...\}, which contain the most possible word written by the user. All the extracted words consist the candidate word list.

The dictionary used by the dictionary filter can be scenario-dependent. For instance, for a doctor working in a hospital, the dictionary can contain more professional medicine-related words. We will evaluate the performance of TableWrite with different dictionaries in Section 3.7.

3.6.2 Feature Extraction

Feature Extraction for Sound Fragments

For each sound fragment, we extract features mainly from frequency domain. Even if the features in frequency domain are usually impacted by the locations of handwriting, our experiments will prove that they still benefit the letter recognition. For each sound fragment, we calculate the FFT coefficients F(\omega), \omega \in [0, \omega_0], and extract the following features, which are often used in audio classification [84]:

- **Brightness**: the centroid of frequency spectrum, denoted as

\[
\omega_C = \int_0^{\omega_0} \omega |F(\omega)|^2 d\omega / \int_0^{\omega_0} |F(\omega)|^2 d\omega. \tag{3.7}
\]
• **Bandwidth:**

\[
\omega_B = \sqrt{\int_0^{\omega_0} (\omega - \omega_C)^2 |F(\omega)|^2 d\omega / \int_0^{\omega_0} |F(\omega)|^2 d\omega.}
\]

(3.8)

• **Pitch Frequency:** \(\omega_P = \arg \max_{\omega \in [0,\omega_0]} F(\omega)\).

• **Number of Large Frequency Peaks:** the number of the frequency peaks whose values are larger than \(\gamma_3 \max_\omega F(\omega)\), denoted by \(n_P\). \(\gamma_3\) is a constant threshold and \(0 < \gamma_3 < 1\). We set \(\gamma_3 = 0.65\).

• **Mel Frequency Cepstrum Coefficients (MFCCs):** we firstly divide each sound fragment into frames with length \(W_f\). Two adjacent frames have overlap \(W_o\). The number of cepstrum coefficients for each frame is denoted by \(N_c\). Thus, for each frame \(i = 1, 2, ..., N_f\), we get a vector of MFCCs: \([m_{i1}, ..., m_{iN_c}]\). Then we calculate

\[
m_j = \sum_{i=1}^{N_f} m_{ij}, \ j = 1, ..., N_c.
\]

(3.9)

In our prototype, the sampling rate of audio signal is 48 kHz. We set \(W_f = 128\), \(W_o = 48\) and \(N_c = 12\).

We denote the features for each sound fragment by:

\[
SF = [\omega_C, \omega_B, \omega_P, n_P, m_1, ..., m_{N_c}].
\]

(3.10)

The sound fragment based feature vector for the \(k\)th letter in the input word is denoted by \(SF_k\).

Feature Extraction for Motion Fragments

Different from sound fragments, motion fragments do not contain periodic signal, and thus spectrum analysis is not applicable. Moreover, the absolute distance change in the motion fragment cannot be used as a feature because it varies with the handwriting’s size. Therefore, we extract the following features mainly based on the relative changes among the peaks/valleys in the curves. Note that the curves have been smoothed with the moving average window with size 30 ms, and the start point’s altitude of the curve is set to be 0.

• **Number of peaks/valleys in the curves.** The minimum distance between two peaks/valleys is 30 ms, so that smaller peaks/valleys are ignored.
• The altitude of the lowest valley divided by that of the highest peak.

• The altitude of the second highest/lowest peak/valley divided by that of the highest/lowest peak/valley. If there is only one peak/valley, the output is 0.

• Relative time point of the highest/lowest peak/valley. The time length of the curve is resized to 1. The output falls in [0,1].

• Relative time point of the second highest/lowest peak/valley. If there is only one peak/valley, the output is the same with the previous feature.

• Index of the highest/lowest peak/valley. The peaks/valleys are arranged in chronological order.

• Index of the second highest/lowest peak/valley. If there is only one peak/valley, the output is the same with the previous feature.

For the \( k \)th letter in the input word, we denote the features extracted from its motion fragment by \( MF_k \). Therefore, for a word with \( N \) letters, the \( k \)th letter is labeled by \( \{s_k, SF_k, MF_k\} \). In our work, we set the stroke number \( s_k \in \{1,2,3\} \). Recall that \( s_k \) has been determined by the input detector.

3.6.3 Word Classifier

The word classification first evaluates the possibility for each input letter to be a given letter, base on which it selects the final outputs from the candidate word list.

The algorithm is based on multi-class SVMs. Recall that we have divided all letters into clusters \( C_u \) \((u = 1,2,3)\). For each cluster, we use a set of SVMs to distinguish letter \( i \) and \( j \), denoted by SVM\(^{i,j}_u \), where \( i,j \in C_u \) and \( i < j \) in alphabetical order. In the training phase, we collect a set of training sound and motion fragments for each letter in \{'A',..., 'Z'\}, and extract their features. Then, we train each SVM\(^{i,j}_u \) with the features of letter \( i \) and \( j \).

In the using phase, for each input letter with stroke number \( s_k \), we input \( \{SF_k, MF_k\} \) to each SVM\(^{i,j}_{s_k} \) where \( i,j \in C_{s_k} \) and \( i < j \). We use \( p_k(l) \) to indicate the probability for the \( k \)th letter in the input word to be a given letter \( l \) in \{'A',..., 'Z'\}. According to the classification result of each SVM\(^{i,j}_{s_k} \), \( p_k(i) \) or \( p_k(j) \) is increased by 1.

For instance, if an input letter has 3 strokes, we pick each pair of letters in \( C_3 \) and run the corresponding SVM. E.g., we first pick up ‘A’ and ‘E’, and run SVM\(^3_{AE} \). If the letter is classified as ‘A’, we increase \( p_k(A) \) by 1, otherwise \( p_k(E) \) is increase by 1. After we have picked up all the pairs of letters in \( C_3 \), \( p_k(A),...,p_k(Z) \) indicates the number of times that
letter ‘A’,...,‘Z’ has won the classifications. Note that in this example, for all the letters \( l \) in \( C_1 \) and \( C_2 \), \( p_k(l) = 0 \).

To eliminate the impact of different cluster sizes, after all the classifications for each letter, we calculate \( \hat{p}_k(l) = p_k(l)/\text{size}(C_{s_k}) - 1 \). Then \( \hat{p}_k(l) \) indicates the degree to which the \( k \)th letter in the input word matches letter \( l \).

Based on \( \hat{p}_k(l) \), we calculate the weight of each candidate word and output the best matches as the recognition results. Assume that there are \( M \) words in the candidate word list. For each candidate word, we calculate

\[
P_m = \sum_{k=1}^{N} \hat{p}_k(l^m_k), m \in [1, M],
\]

where \( l^m_k \) denotes the \( k \)th letter in the \( m \)th candidate word. Therefore, \( P_m \) denotes the probability for the \( m \)th candidate word to be the input word. Then, we select the top-\( K \) candidate words with the highest \( P_m \) as the final outputs. \( K \) is a configurable value, and the output words can be displayed on the device’s screen for users to choose.

### 3.7 MEASUREMENT AND EVALUATION

In this section, we evaluate the performance of our prototype system. We implement the speaker, the recorder and the word detector on a commercial Android smartphone, and implement the other components on a laptop. Note that our scheme can be easily applied to tiny mobile devices equipped with speakers and microphones. The sampling rate of microphone is 48 kHz, and the format of audio data is PCM 16 bit per sample. During the tests we use three separate dictionaries. For daily life, we use the top-2000 and top-5000 commonly used words [85]. For clinic scenario, we extract the top-9000 words in the MIMIC II Clinical Database [86]. We set \( W = 200, \alpha_1 = 16, \alpha_2 = 1, \beta_1 = 45 \text{ ms}, \beta_2 = 68 \text{ ms}, \text{ and } \beta_3 = 227 \text{ ms.} \)

We set our test environment in a laboratory with low near-field noise. Nine volunteers are invited to take the tests. Each of them writes words on a wooden desk with one finger (using fingernail). The range of characters is ‘A’-'Z’. We place the smartphone on the same desk. The distance between the smartphone and the handwriting is around 20-30 cm, and the relative angle between them is roughly fixed during the test. To collect training data, we require each volunteer to write each letter in print style for 10 times, the signals of which are used to train the word classifier for each volunteer.
3.7.1 Performance of Word Recognition

In this subsection, we evaluate the accuracy of the final output of TableWrite. We conduct the test for each dictionary. In each test, each of our volunteers writes 1000 words randomly selected from the dictionary. We select test words with two different strategies. With *frequency-based selection*, we assign each word in the dictionary a weight, which is the word’s frequency of appearance. The probability for a word to be selected is equal to its weight divided by the sum of all the weights in the dictionary. With *uniform selection*, we select test words following uniform distribution. Following each strategy, we collect 9000 samples during each test. The results are shown in Figure 3.4. The curves show how the probability of correct recognition changes with the number of returned candidate words (horizontal axis), i.e., the rank-k accuracy represents the probability that the correct recognition result is included in the top-k best matches returned by TableWrite.

![Figure 3.4: Accuracy of word recognition.](image)

The results of experiments show that, under the condition of print-style writing and low near-field noise, the word recognizer of our system achieves an accuracy of 90.7%-93.6% for the frequency-based word selection, and 94.5%-95.9% for the uniform word selection. It is also shown that, for all the test cases, it is highly possible (> 99%) that the correct recognition results are included in the top-5 returned candidate words. This confirms the practicability of TableWrite as an input method. Moreover, the performance of TableWrite is consistent for the dictionary of medicine-related words, which shows that TableWrite also works well in specific scenarios, e.g., for doctors in hospitals.
3.7.2 Contribution of Different Components

To better understand the contribution of the dictionary filter, the word classifier and the hand motion tracker, we analyze the change of the recognition accuracy when different components are disabled. In 

*Case 1*, we use dictionary filter only and the recognition result is selected from the candidate word list uniform-randomly, which rules out the contribution of hand motion tracker and the word classifier. In 

*Case 2*, the word classifier (multi-class SVMs) is also applied, but only the features extracted from sound fragments are used. In 

*Case 3*, the features extracted from motion fragments are further used. The same data from the previous test are used, and we consider both the frequency-based word selection and the uniform word selection. The results are shown in Figure 3.5(a) and 3.5(b).

The results of Case 1 show that, since the dictionary filter narrows the space of candidate words, over 37% of written words can be correctly recognized even without any machine learning method. The results of Case 2 and 3 show the improvement made by the SVMs and the hand motion tracker, successively. Therefore, the contribution made by each of these components is confirmed.

![Recognition accuracy with different components (frequency-based selection).](image)

![Recognition accuracy with different components (uniform selection).](image)

(a) Recognition accuracy with different components (frequency-based selection).

(b) Recognition accuracy with different components (uniform selection).

Figure 3.5: Recognition accuracy with different components.

3.8 DISCUSSION

3.8.1 Limitations of TableWrite

While we have designed various methods to improve the performance of TableWrite, there are still some limitations. When the user is writing with TableWrite, the hand motion...
tracker can be interfered by nearby moving objects, especially when the distance to the moving object is less than 40 cm, based on our test results. The current solution is that, when nearby moving objects (such as walking people) exist, the user can disable the hand motion tracker, which does not stop TableWrite from working. The experiment results in Section 3.7.2 show that the recognition accuracy still achieves 80% in this case.

On the other hand, if the background audio noise is too strong, it is hard to distinguish the handwriting signal and the noise. While it is possible to segregate the handwriting sound by using multiple microphones if the number of noise sources is limited [87], defending against noise is still a challenging problem for all the audio-based approaches. We leave this problem for our future work.

3.8.2 Possible Ways to Improve Accuracy

There are still possible ways to further improve the performance of TableWrite. For instance, we can apply the semantic check when the user writes a sentence instead of a single word. Since TableWrite returns an ordered candidate list for each input word, it is easy to apply the syntactic correction and select the best candidate words that compose a meaningful sentence.

Another promising way is applying the deep learning method. With enough training data, we can train a deep learning model that works for everyone, and thus eliminate the training phase and probably achieve a higher accuracy. However, running the model on tiny mobile devices is expensive, and would also hurt the realtime performance.

3.9 CONCLUSION

In this chapter we present TableWrite, an audio-based handwriting input scheme which allows users to input words to mobile devices by writing on tables. We designed the components of word detector, hand motion tracker and word recognizer, which enable mobile devices to recognize users’ handwriting. The experimental results show that, the accuracy of word recognition is higher than 90%, which shows the effectiveness of TableWrite.
CHAPTER 4: WRITINGHACKER: AUDIO BASED EAVESDROPPING OF HANDWRITING VIA MOBILE DEVICES

4.1 INTRODUCTION

It has been proven that the sound of typing keyboards leaks information, which implies the possibility of acoustic side-channel attacks [88, 89, 90, 91]. However, it has not yet been recognized that the sound of handwriting also leaks information. People’s writing habits follow some common patterns (e.g., stroke number), which make audio-based handwriting recognition possible. The rapid evolution of mobile terminals compounds the danger of audio-based eavesdropping of handwriting in public environments. For instance, when a person is filling out privacy-related forms on a desk in medical environments such as hospitals or clinics, an attacker can record the sound of handwriting with a smartphone, and recover the content of handwriting. To expose this danger, in this chapter we investigate the possibility of eavesdropping on handwriting via mobile devices based on audio signal processing and machine learning.

Although handwriting on desks has been considered as a new text entry method for small mobile devices, this technique is not applicable for the eavesdropping attack. The methods in [31] and [1] enable users to write on surfaces by using smartphones as pens or by wearing smartwatches on wrists. However, these methods assume that users have direct contact with some mobile devices, and thus cannot be used for attack because attackers usually have no access to victims’ devices. In [27], the authors leverage the embedded microphone in commercial smartphones to record the sound of handwriting on surfaces and recognize the input text. However, once the system has been trained, its performance highly depends on the location of handwriting, which makes eavesdropping attack impossible. In this chapter, we investigate the capacity of audio-based side-channel attack under the condition that the attacker’s mobile device has no direct contact with the victim, and the performance of the attack does not highly depend on the handwriting location.

In this chapter, we present WritingHacker, an audio-based eavesdropping proof-of-concept system which eavesdrops on handwriting via mobile devices. Our method is based on the basic assumption that the handwriting of victims is print-style, which means that we currently ignore the impact of joined-up writing. We consider this assumption reasonable because most of privacy-related forms require print-style writing [92].

The main idea of our attack method is that, by keeping a mobile device such as a smartphone touching the desk which is used by a victim, an attacker can record the sound of the victim’s handwriting. After regaining the device, the attacker can reconstruct the words
that the victim has written by extracting features from the sound sequence and recognizing
the unique patterns of each letter.

The main challenge is that, in the scenarios of attack, labeled training data from victims
are always unavailable. Moreover, the performance of existing audio-based handwriting
recognition methods highly depends on the location of handwriting, which can hardly be
controlled by the attacker. We show that, using WritingHacker, an attacker can leverage the
common patterns of people’s print-style handwriting, which are not affected by the afore-
mentioned factors. For instance, different letters sometimes have different stroke numbers.
Letter ‘A’ usually has three strokes, while letter ‘C’ has only one stroke, which makes it
easy to distinguish between these two letters even if victims’ training data are unavailable.
Meanwhile, the stroke number for each letter is not impacted by handwriting locations. In
WritingHacker, we classify all the letters into different clusters according to their stroke
numbers, and implement a Support Vector Machine (SVM) for each cluster to distinguish
the in-cluster letters. To train the SVMs, the attacker can collect training data from other
people instead of the victims. To provide the linguistic word-level correction and narrow
down the word search range, we further apply a dictionary filter, which is also based on
letter clustering. Moreover, we design the algorithm of letter time length based offsetting
(LTLO) to utilize the time length diversity among the letters written by the victim. Fur-
thermore, inspired by the gesture tracking technique, we also apply an audio signal reflection
based method to track the motion of the victim’s hand, which provides valuable features if
the relative position between the device and the handwriting is known by the attacker.

The main contributions of our work are:

- To the best of our knowledge, we are the first to investigate the possibility of eaves-
dropping on handwriting based on audio signals.

- To solve the lack of training data from victims and reduce the impacts of diverse writing
  locations, we propose to use the stroke number as a common handwriting pattern, and
  apply the method of letter clustering.

- We design the methods of dictionary filtering and LTLO, and apply the hand motion
  tracking technique, which significantly improve the recognition accuracy.

- We implement a prototype system on Android-laptop platform and conduct evalu-
  ations. Experiments confirm that privacy leakage through eavesdropping on handwrit-
  ing is highly probable under certain conditions such as print-style writing and low
  near-field noise.
WritingHacker is still a proof-of-concept system based on our basic assumption. However, it sheds light on a new threat to people’s privacy caused by mobile devices.

We organize the rest of this chapter as follows. After introducing related work in Section 4.2, we present an overview of the system architecture and the attack method in Section 4.3. Then we successively introduce the methods and algorithms used by the system components in Section 4.4-4.7. We include the implementation details and performance evaluation of WritingHacker in Section 4.8 and 4.9. We discuss the experiment results in Section 4.10, compare WritingHacker and TableWrite in Section 4.11, and then conclude this chapter in Section 4.12.

4.2 RELATED WORK

Leakage of privacy via text entry methods: The topic about the leakage of privacy via text entry methods has become popular in recent years. Many existing eavesdropping attacks of text entries focus on keyboards, which are the most common input tools for electronic equipments. Besides the approaches based on video of typing sessions [93] or timing information of key presses [94], the other methods are usually based on audio or acceleration signal processing.

By extending the technique of human voice recognition, attackers can recognize the keys that a victim has pressed by eavesdropping on keystrokes. Asonov et al. firstly showed that keyboards are vulnerable to attacks based on differentiating the sound emanated by different keys [88]. In [89], Zhuang et al. presented an audio-based attack method which combines standard machine learning and speech recognition techniques. Berger et al. in [90] presented a dictionary attack which can reconstruct a single typed word from audio signals with a dictionary of words. For the acceleration-based eavesdropping, Marquardt et al. revealed that an application with access to accelerometer readings on a smartphone can recover text entered on a nearby keyboard [68]. However, the audio/acceleration signals of handwriting are much more diverse compared with those of keystrokes. For instance, the keyboard audio signal for each keystroke usually contains a press region and a release region [91]. Thus it is easier to select appropriate features for machine learning algorithms to compare different keystrokes. For the case of handwriting, the acoustic signals of different strokes vary with people. Thus the previous approaches cannot be applied to our scenario.

The universalization of smart mobile devices is also increasing the risk of privacy leakage. For example, Wang et al. has shown the danger that the motion sensors in smartwatches can leak information about what the user is typing [24]. However, this approach requires the direct contact between victims and mobile devices, which is not practical in the scenario.
of eavesdropping attack.

In our work, WritingHacker eavesdrops on handwriting based on audio signal processing, because compared with acceleration signals, audio signals are less influenced by the propagation distance in solid media such as desks.

**Handwriting Recognition:** The techniques for automatic handwriting recognition (HWR) can be classified as online and offline case [73]. Offline recognition usually processes images of handwriting only, whereas online recognition can further utilize rich sensor data such as stylus positions and temporal information during the writing process [73, 74].

In this work we focus on audio signal based attack, and thus do not apply the image-based offline recognition techniques such as Optical Character Recognition (OCR) [95]. For the online case, multiple sensors can be used to improve recognition accuracy. In [96], Nakai et al. used pen pressure as a feature based on the observation that the pressure represents pen ups and downs as well as the temporal pattern of handwriting in a continuous manner. In [97, 98, 99], motion sensors embedded in pens or Micro Electro Mechanical Systems were also utilized to sense the motion produced by written characters. Different from these approaches, we mainly utilize the widely installed microphones in mobile devices to recognize handwriting.

**Handwriting on desks as an input method:** Since handwriting is a convenient input method other than keystroking, there have been some works focusing on enabling handwriting on desks to be a new text entry method for small mobile devices. The system named GyroPen in [31] enables users to use their smartphones as pens to write on desks by using embedded gyroscopes and accelerometers. The system in [1] enables a user to write on surfaces with fingers. A smartwatch worn on the user’s wrist records the accelerometer signals of user’s hand, which can be utilized to recognize the user’s handwriting with the algorithm of gesture recognition. However, these approaches are not suitable for attack because they require users to have direct contact with mobile devices.

Using the system from [27], after executing a training phase, a user can write with a finger on surfaces, and the microphone in a smartphone records the sound of handwriting. However, once the location of the smartphone is changed, the training phase must be repeated. Since the training data from victims are usually unavailable in practice, this system cannot be used for eavesdropping attack.

In [83], a device-free gesture tracking scheme named LLAP (Low-Latency Acoustic Phase) utilizes a speaker to generate an audio signal, and tracks the motion of a user’s hand by calculating the phase shift of the signal reflected by the hand. It is possible to use this scheme to track a victim’s hand motion and thus recognize the handwriting. However, this scheme only works in a limited distance (< 35 cm) and thus cannot be applied to all the
attack scenarios. We apply LLAP only to enhance the performance of WritingHacker under certain conditions.

In this chapter, we focus on a more practical case where the attacker’s mobile device has no direct contact with the victim, and the attacker has no control on the distance or relative angle between the device and the victim’s handwriting.

4.3 WRITINGHACKER OVERVIEW

In this section, we first describe our basic assumptions, and then have an overview of the system architecture and the workflow of our attack method.

4.3.1 Basic Assumptions

As mentioned before, our attack method is based on the basic assumption that the handwriting of the victim is print-style. We consider this assumption reasonable because print-style writing is commonly required by important privacy-related forms. Based on the observation that people often write separate words instead of sentences when filling out forms, currently we mainly focus on the word-level recognition.

During the attack, we do not assume that the attacker has any knowledge about the pen type, the desk material, or the approximate relative angle between the eavesdropping device and the handwriting. However, as will be shown during the evaluation, better performance can be achieved if these details are known. It is easy for the attacker to get this information if the victim is writing in a public place. The moving objects within 40 cm from the eavesdropping device, except the victim’s writing hand, could interfere the hand tracking of WritingHacker. In this case, as will be discussed in Section 4.5, the attacker can close the hand tracking function to avoid the interference, and our method can still achieve good accuracy. Moreover, we currently focus on the attack scenarios where the average background noise is not too strong (< 50 dB), which is common in the waiting rooms and meeting rooms.

4.3.2 System Architecture

As shown in Figure 4.1, our system mainly consists of the following components.

*Speaker, recorder and accelerometer:* The speaker continuously plays a continuous wave signal with high frequencies, which is nearly inaudible to humans. The recorder records the signal reflected by the victim’s hand and the sound of handwriting with the embedded
microphone, and sends the recorded signal to the input detector and the hand motion tracker in stream. The output of the accelerometer is used to combat near-field noise.

*Input detector*: This component receives the sound stream and extracts the *sound fragment* for each letter. The subcomponent of stroke detection detects each stroke of handwriting, and the letter detection subcomponent detects and extracts sound fragments, each of which contains the audio signal for a single letter. The sound fragments together with the number of strokes for each letter are sent to the word recognizer, while the number of strokes is also passed to the dictionary filter to narrow word search range. The hand motion tracker also receives the start and end time point for each letter from this component.

*Hand motion tracker*: It is an optional component. By analyzing the reflected audio signal, it tracks the tiny distance change between the victim’s hand/pen and the device. The curve of the distance change for each letter is stored as a *motion fragment*, which is sent to the word recognizer.

*Dictionary filter*: The input to this component includes the number of strokes for each letter in each word. The filter reads the dictionary database and provides a list of candidate words.

*Word recognizer*: This component calculates the similarity between the recorded sound fragments and each word in the candidate list. The best match is the recognition result. The feature extraction subcomponent extracts the features for each letter, and the subcomponent of LTLO provides an offset for each candidate word. Finally, the word classifier recognizes each word using machine learning algorithms.

### 4.3.3 Attack Method

Our attack method consists of the following phases.

**Phase 1 (Eavesdropping)**: To commit the handwriting eavesdropping without being noticed, an attacker can leave the mobile device on the desk where a victim is writing on a piece of paper, which does not violate non-direct-contact. The speaker in the device generates a nearly-inaudible audio signal. The recorder records the reflected signal and the sound of
the victim’s handwriting via the microphone. Then the attacker gets the device back. Note that the mobile device is not necessarily a cellphone. Some tiny devices, or even wireless controlled microphones and speakers can also be used for the attack.

**Phase 2 (Training Data Collection):** The attacker needs to hire some other people instead of the victim to collect labeled training data. The training data should include the sound stream and the number of strokes for each character (e.g., ‘A’-‘Z’) from different people’s handwriting. We assume that it is easy for the attacker to collect enough training data. The data are used to train the word classifier. Note that Phase 2 can be executed ahead of Phase 1. However, to achieve better performance, the attacker can execute Phase 1 first, and during Phase 2 the attacker can use the same type of pen and material of desk as those used by the victim in Phase 1. If the attacker knows the approximate relative angle between the device and the handwriting, the same angle should be followed in this phase.

**Phase 3 (Recognition):** The stored sound stream is processed by the rest of WritingHacker’s components, and then the attacker gets the estimate for the victim’s handwriting. If the approximate relative angle between the device and the handwriting is unknown, the distance between them is larger than 35 cm, or there are other moving objects around the victim, the component of hand motion tracker should be disabled. Note that phase 2 and phase 3 are offline phases.

In this work, we take the capital letters ‘A’-‘Z’ as examples to show how WritingHacker works. However, our technique can be easily applied to other characters.

Next, we introduce the algorithms of the four main components (input detector, hand motion tracker, dictionary filter and word recognizer) in detail respectively.

### 4.4 INPUT DETECTION

The method used by the word detector is based on signal processing in time domain. In the previous chapter, we proposed an input detection method for finger-based writing. In this chapter, we apply a similar design for pen-based writing, which consists of stroke detection and letter detection. The definition of stroke in Section 3.4.1 is still applied here.

#### 4.4.1 Letter Clusters

In this chapter, we keep using the three letter clusters defined in Section 3.4.1. Each letter in cluster $C_u$ has $u$ strokes ($u = 1, 2, 3$). The concept of clusters helps narrow the word search range in the dictionary filter and reduce the number of classes in the word classifier.
We admit that due to people’s different writing habits, for some people it is possible that some letters should be classified into different clusters. For instance, some people write letter ‘J’ with one stroke instead of two strokes. Sometimes the stroke order of the same letter would also vary with people. However, based on our basic assumption that victims’ handwriting is print-style, we observe that people’s print-style writing has limited variety, and the stroke order of the same letter written by a victim usually does not change. For instance, while there are two ways to write letter ‘J’, there is usually only one way to write letter ‘X’. Thus the attacker can adjust the clusters and their corresponding training data by traversing different print writing styles to find the most reasonable guess for the attack scenario. Therefore we leave it to our future work to rule out the impact of the stroke number’s diversity.

4.4.2 Stroke Detection

Similar to TableWrite, we first use a FIR low pass filter to eliminate the signal generated by the speaker and the signal reflected by the hand. The passband is 4 kHz and the stopband is 6 kHz. Then, we apply the modified version of CFAR algorithm [78] to combat the background noise and detect each stroke in the continuous sound stream.

Assume that the noise power follows Gaussian distribution. We denote its average power and standard deviation at time \( t \) by \( \mu(t) \) and \( \sigma(t) \), and denote the audio signal by a time series \( x(t) \). Based on a sliding window of size \( W \), the average noise power at time \( t \) can be calculated by [79]:

\[
\mu(t) = \frac{1}{W^2} \sum_{k=t-W+1}^{t} |x(k)|^2 + (1 - \frac{1}{W})\mu(t-1). 
\]

(4.1)

The standard deviation \( \sigma(t) \) can be calculated similarly.

A potential start point of a stroke is detected if

\[
|x(t)|^2 > \mu(t) + \gamma_1 \sigma(t),
\]

(4.2)

and a potential end point of a stroke is detected if

\[
|x(t)|^2 < \gamma_2 \bar{\mu},
\]

(4.3)

where \( \gamma_1 \) and \( \gamma_2 \) are two constant parameters that are independent of noise levels. \( \bar{\mu} \) is the average noise power when there is no input. The initial value of \( \bar{\mu} \) is measured during the
very beginning of recording before the first stroke appears. Then we update it by measuring the average noise power between each two adjacent detected letters.

A stroke is detected if the time interval between the potential start and end point is longer than a constant threshold $\Gamma_1$, as shown in Figure 4.2(a). In this way the impact of burst noise can be avoided. If the time interval between two detected strokes is shorter than $\Gamma_1'$, we combine these two strokes and consider them as a single stroke.

![Figure 4.2: Example for input detection.](image1)

*Impact of multiple turning points:* Sometimes a stroke with multiple turning points would show similar features as a multi-stroke letter. One example is shown in Figure 4.3(a). According to our definition of a stroke, the letter ‘M’ contains one stroke. Due to the turning points included in the stroke, it looks like that the single fragment contains three strokes. However, by comparing its signal with that of the real three-stroke letter ‘A’ in Figure 4.3(b), it is easy to find that the time interval between each two adjacent “strokes” in ‘M’ is much shorter than that in ‘A’. Thus the threshold $\Gamma_1'$ can help to rule out the impact of false strokes. In this case, since both the two time intervals in ‘M’ are shorter than $\Gamma_1'$, the three “strokes” are combined as a single stroke.

![Figure 4.3: Comparison between a multi-turning-point stroke and a multi-stroke letter.](image2)
4.4.3 Letter Detection

After each stroke has been detected, this subcomponent extracts the sound fragment for each letter. The start point of a letter is declared if it is the first start point of a stroke after the end point of the previous letter. The end point of a letter is declared if there is no stroke detected in constant duration $\Gamma_2$ after the end point of a stroke. Then the end point of this stroke is the end point of the letter. One example is shown in Figure 4.2(b).

Then the measured sound signal between the start and end point of the letter is output and stored as a single fragment. The fragments detected will be further processed by the feature extraction subcomponent. We use $s_k$ to denote the number of strokes for the $k$th letter in a word with $N_{\text{letter}}$ letters. For example, word “VITAMIN” has $N_{\text{letter}} = 7$, and $s_1 = 1$ for letter ‘V’ $\in C_1$. Then $\{s_k | k \in [1, N_{\text{letter}}]\}$ is passed to the dictionary filter and the word classifier, which will help to select proper word clusters and narrow word search range. The start and end time point for each letter is also sent to the hand motion tracker to help crop the measurement result for each letter.

4.4.4 Usage of Accelerometer

To combat the impact of near-field noise, we use a technique similar to [79]. When reading the sound sequence from the recorder, the input detector also reads the measured acceleration magnitude from the embedded accelerometer. If a possible letter is detected but the average acceleration in a window $W_a$ around the start point of the letter is lower than a threshold $\Gamma_a \bar{a}$, the letter is ignored. $\Gamma_a$ is a constant value and $\bar{a}$ is the average acceleration when there is no letter input. The intuition is that, although accelerometers can be impacted by voices [100, 101, 102], the acceleration signal generated by the shaking table surface during the writing is stronger than that caused by the noise. Thus, by setting $\Gamma_a$ appropriately, we can distinguish the two signals, and the input detector accordingly only records the sound generated by handwriting. In this way, the impact of near-field burst noise is partly avoided. However, frequent burst noise would still lead to the missing of letters and how to rule out its impact is still an open question. We leave this problem for our future work.

4.5 HAND MOTION TRACKING

In Section 3.5, we have shown that the device-free gesture tracking technique can help distinguish the written letters. It is also possible that the attacker can recognize the content of handwriting via the motion trajectory of the victim’s hand. The challenge is that, the
nature of device-free tracking makes it hard for the device to determine which body part it
is tracking, especially for the attack scenario. Thus, it suffers low accuracy when tracking
the whole trajectory of the victim’s hand. However, during our experiments we find that,
in the short time slot of writing a single letter in a word, the other body parts of the victim
usually do not move a lot. Therefore, the short-term tracking result for a single letter can
still correctly reflect the tiny motion of the victim’s hand and pen, and thus helps to improve
the performance of handwriting recognition.

The LLAP-based hand motion tracking method in Section 3.5 is also applied in WritingHacker, and we can get a motion fragment for each input letter. However, due to the
weaker reflected signals, LLAP does not work well if the distance between the device and
the handwriting is larger than 35 cm. Also, if there are nearby moving objects around the
victim, the hand motion tracker should be disabled. However, WritingHacker still achieves a
good performance in this case, which will be shown in Section 4.10.1. Moreover, even if the
precise relative angle between the device and the handwriting is unknown, a good accuracy
is still maintained if the angle error is within $\pm 45^\circ$, which will be shown in Section 4.9.7.

4.6 DICTIONARY FILTERING

In WritingHacker, we apply the same dictionary filter as in Section 3.6.1. However, it is
more important for the dictionary database to be scenario-specialized, which is worth further
discussion.

4.6.1 Scenario-Specialized Dictionary Database

It has been shown that the frequency of commonly used words follows Zipf’s law [103],
which implies that we can narrow our word search range to the most commonly used words.
We can further specialize our dictionary database according to the attack scenarios. For
illustration, we obtain the top 5000 most common words in Intensive Care Units (ICUs)
by analyzing the statistical data from the MIMIC II Clinical Database [86], which contains
notes and reports from approximately 32000 ICU patient admissions. As shown in Figure
4.4, the frequency of words still follows Zipf’s law even under such a specialized scenario.
We observe that the ranks of words vary with scenarios. For instance, the frequency of the
word “vitamin” used in ICUs is much higher than that in daily life, which confirms the need
of scenario-specialized databases for the attack purpose.

In WritingHacker, once the attack scenario (e.g., in hospital) is determined, we use the
most commonly used words in that scenario as the dictionary database. Note that because
of the nature of our system design, the system performance does not highly depend on the attack scenario. Instead, the accuracy of word recognizer is related to whether the dictionary database is constructed properly. If the database is not specialized according to the attack scenario or its size is too small, it is more likely that the words, written by the victim, are not covered by the database. However, if the size of the database is too large, the rarely used words would extend the word search range and thus impact recognition accuracy. We will evaluate the impact of the database size during our experiments.

4.6.2 Generating Candidate Word List

The component of dictionary filter has access to the scenario-specialized dictionary database and extracts candidate words according to the number of strokes for each letter in the word. The strategy has been shown in Section 3.6.1. For example, for the received \( \{s_1 = 3, s_2 = 2, s_3 = 2, s_4 = 1, s_5 = 3\} \), the candidate words extracted from the top-5000 daily words [85] are \{“APPLE”, “APPLY”\}, one of which is the most possible word that the victim has written. The candidate word list is then passed to the word recognizer.

4.7 WORD RECOGNITION

The component of word recognizer extracts the features of the sound and motion fragments for each letter, calculates the offset for each candidate word, and then recognizes the whole word according to the features, the trained SVMs, the candidate word list, and the offsets.
4.7.1 Feature Extraction

We use the same features introduced in Section 3.6.2. The features extracted from the sound fragment of the \( k \)th letter in the input word is denoted by \( F_k \), while those extracted from the motion fragment is denoted by \( F'_k \). Thus, for each input word, the \( k \)th letter \( g_k \) is labeled by \( \{ s_k, F_k, F'_k \} \). Recall that \( s_k \in \{1, 2, 3\} \) and it has been determined by the input detector. If the hand motion tracker is disabled, \( F'_k \) is removed, which does not impact the following word recognition.

4.7.2 Letter Time Length based Offsetting (LTLO)

![Figure 4.5: Average time length for each letter based on the data collected from volunteers.](image)

Besides the extracted features, the various time length of different sound fragments can also help to distinguish different letters. Figure 4.5 illustrates the average time length for each letter based on the data from 10 volunteers. It is shown that the letters with more strokes tend to have longer time length. Moreover, even if two letters have the same stroke number, they would still have distinctly different time length. For instance, while both ‘E’ and ‘H’ have three strokes, ‘E’ has a longer time length because it contains one more turning point in the strokes. This motivates us to consider the time length difference among letters.

However, because of people’s various writing speeds, it is unreliable to compare the letter time length from victims and that from training data providers. Thus, the letter time length cannot be used as an feature directly. Instead, we compare the time length differences among the letters written by the victim only. In spite of people’s different writing speeds, these differences are constant if the victim writes the word at a constant speed. We use these differences as features, and compare them with those extracted from the training data.
ALGORITHM 4.1: Letter Time Length based Offsetting

Input: $d, k \in [1, N_{letter}]$; $D(l), l \in L; \Gamma_d$; $\tilde{W}_m = \{ \tilde{g}_k^m \in L | k \in [1, N_{letter}] \}, m \in [1, M]$;
Output: $\tilde{P}_m, m \in [1, M]$;

// Define a function
1 Function $f_d(x, y)$ is
2  
if $x - y < -\Gamma_d$ then
3      return $-1$;
4  else if $x - y > \Gamma_d$ then
5      return $1$;
6  else
7      return $0$;

// Initialization
8 $\tilde{P}_m = 0, m \in [1, M]$; $T_{i,j}^{test} = 0, i, j \in [1, N_{letter}]$; $T_{i,j}^m = 0, m \in [1, M], i, j \in [1, N_{letter}]$;

// Calculate time length difference tables for candidate words
9 foreach $m \in [1, M]$ do
10    foreach $i, j \in [1, N_{letter}], i < j$ do
11      $T_{i,j}^m = f_d(D(\tilde{g}_i^m), D(\tilde{g}_j^m))$;

// Calculate time length difference table for test word
12 foreach $i, j \in [1, N_{letter}], i < j$ do
13    $T_{i,j}^{test} = f_d(d_i, d_j)$;

// Calculate offset for each candidate word
14 foreach $m \in [1, M]$ do
15    $w = 0$;
16    foreach $i, j \in [1, N_{letter}], i < j$ do
17      $w += |T_{i,j}^m - T_{i,j}^{test}|$;
18    $\tilde{P}_m = 1 - w / [N_{letter} (N_{letter} - 1)]$;
19 return $\tilde{P}_m, m \in [1, M]$;

Candidate words: Time length difference tables: Offset:

“CAN”

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>A</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

“SEA”

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>I</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

Input word:

?: 396 ms
?: 724 ms
?: 618 ms

Figure 4.6: An example for LTLO.
The algorithm of LTLO is shown in Algorithm 4.1. We denote the time length of each input letter by \( d_k \), and the average time length of letter \( l \) in the training data by \( D(l) \), which has been shown in Figure 4.5. The candidate word list is denoted by \( \{ \tilde{W}_m | m \in [1, M] \} \), where \( M \) is the number of the candidate words. \( \Gamma_d \) is a constant threshold. During the implementation we set \( \Gamma_d = 63 \) ms. We use the example in Figure 4.6 to illustrate the algorithm. Assume that the input letters have strokes \( \{ s_1 = 1, s_2 = 3, s_3 = 3 \} \), and the candidate words are \{“CAN”, “SEA”\}. For each candidate word, we calculate a time length difference table (line 9-11), which denote the supposed time length differences among letters if the candidate word were the correct answer. One more time length difference table is also calculated based on the measured time length of each input letter (line 12-13). Then, we compare the table of the input word with those of the candidate words, i.e., “CAN” and “SEA”, and calculate the offset \( \tilde{P}_m \) for each candidate word (line 14-18). As shown in Figure 4.6, since ‘A’ and ‘N’ have similar time length, and ‘E’ has a much longer time duration compared with ‘A’, the tables of “CAN” and “SEA” are different. The result of comparison shows that the table of “SEA” is more similar to that of the input word, and thus “SEA” gets a larger offset.

4.7.3 Word Classifier

The function of word classifier consists of letter scoring and word selection. The former step evaluates the possibility for each detected letter to be a given letter. The latter step selects the most possible word according to the candidate word list, the offsets, and the results of letter scoring.

Letter Scoring

The algorithm we use to score the detected letters is based on the multi-class SVM. Since all letters have been divided into three clusters, we design specialized SVMs for each cluster. In detail, for the cluster \( C_u \) (\( u = 1, 2, 3 \)), we use a set of SVMs \( V_{ij}^u \) to distinguish the difference between letter \( i \) and \( j \), where \( i, j \in C_u \) and \( i < j \) in alphabetical order.

For each letter \( l \in L = \{ ‘A’, ..., ‘Z’ \} \), we collect a set of training sound fragments. Each set should contain the data collected from multiple people’s handwriting, as well as the actual written letters. For each sound fragment, we calculate its audio features as denoted by Equation (3.10). Thus, we get the feature sets \( T_i, l \in L \). Then we train each \( V_{ij}^u \) (\( u = 1, 2, 3 \)) with the features \( T_i \) and \( T_j \). Note that in this case \( i, j \in C_u \cap L \).

The steps of letter scoring are shown in Algorithm 4.2 (line 2-11). According to the \( s_k \) of
each input letter, the feature set \( \{ F_k, F'_k \} \) is input to each \( V_{ij}^{sk} \), where \( i, j \in C_{sk} \) and \( i < j \) (line 4). We use \( p_k(l) \) to indicate the probability for \( g_k \) to be a given letter \( l \) in \( L \). According to the classification result of \( V_{ij}^{sk} \), \( p_k(i) \) or \( p_k(j) \) is increased (line 5-8). We eliminate the impact of different cluster sizes in line 9-10. Note that if \( u \neq s_k \), \( \tilde{p}_k(l) = 0 \) for all \( l \in C_u \). In line 11, we apply the stroke duration based correction to further improve accuracy.

**Algorithm 4.2: Word Classification**

**Input:** \( \{ s_k, F_k, F'_k \}, k \in [1, N_{\text{letter}}]; P_m, m \in [1, M]; V_{ij}^u, i, j \in C_u, i < j, u = 1, 2, 3; C_u, \)

\( u = 1, 2, 3; \tilde{W}_m = \{ \tilde{g}_k^m \in L | k \in [1, N_{\text{letter}}] \}, m \in [1, M] \);

**Output:** \( \hat{W} \);

// Initialization
1 \( p_k(l) = 0, l \in L, k \in [1, N_{\text{letter}}]; \hat{W} = \emptyset; \)

// Letter Scoring
2 foreach \( k \in [1, N_{\text{letter}}] \) do
3     foreach \( i, j \in C_{sk}, i < j \) do
4         Test \( \{ F_k, F'_k \} \) with \( V_{ij}^{sk} \);
5         if \{ \( F_k, F'_k \) \} is classified as \( i \) then
6             \( p_k(i)++ \);
7             else
8                 \( p_k(j)++ \);
9
10 foreach \( k \in [1, N_{\text{letter}}], l \in L \) do
11 \( \tilde{p}_k(l) = p_k(l)/(\text{size}(C_{sk}) - 1); \)

12 Execute stroke duration based correction;

// Word Selection
12 foreach \( m \in [1, M] \) do
13     \( P_m = \sum_{k=1}^{N_{\text{letter}}} \tilde{p}_k(\tilde{g}_k^m); \)
14 \( \hat{W} = \hat{W}_m, \) where \( m = \arg \max_{m \in [1, M]} \{ P_m + \gamma_d N_{\text{letter}} \tilde{P}_m \}; \)
15 If ties exist, break ties;
16 return \( \hat{W} \);

*Stroke Duration based Correction:* Besides the patterns in letters’ various time lengths, we find that some common patterns are also followed in the stroke level. For instance, the time durations for some specific strokes are differentiable: the second stroke of ‘Q’ is much shorter than the other strokes in the two-stroke letters. It is possible to recognize this pattern without any machine learning algorithm. In our implementation we set a constant threshold \( r_Q \). If the ratio between the time lengths of the first and the second stroke in a two-stroke letter \( g_k \) is larger than \( r_Q \), we increase the \( \tilde{p}_k(‘Q’) \) by \( p_\Delta \). During the implementation we set \( r_Q = 3.3 \) and \( p_\Delta = 0.5 \).
Word Selection

Because \( \tilde{p}_k(l) \) indicates the degree to which the input \( g_k \) matches letter \( l \), it helps to select the best estimate from the candidate word list.

The steps of word selection are also shown in Algorithm 4.2 (line 12-16). Recall that we denote the candidate word list by \( \{ \tilde{W}_m \mid m \in [1, M] \} \), and the length of each candidate word is equal to that of the input word. Then for each \( \tilde{W}_m \), we evaluate the probability for it to be the input word by calculating \( P_m \) (line 12-13). Recall that we have calculated \( \tilde{P}_m \) as the offset based on letter time lengths. Then, we select the one with the highest \( P_m + \gamma_d N_{\text{letter}} \tilde{P}_m \) as our final output (line 14), where \( \gamma_d \) controls the weight of the offset. During the implementation we set \( \gamma_d = 100 \).

To break possible ties (line 15), for each candidate word in a tie, we calculate the number of letters that are the best match with the input letters, i.e.,

\[
N_m = \sum_{k=1}^{N_{\text{letter}}} 1[\tilde{g}_k^m = \arg \max_{l \in L} \tilde{p}_k(l)].
\]  

(4.4)

Then we output the word with the largest \( N_m \) in the tie. If the tie still exists, we output the candidate word with the highest frequency of use according to the statistical data in the database.

4.8 IMPLEMENTATION DETAILS

In this section, we describe the implementation details of WritingHacker. Except for the recorder, all components can be implemented either on smartphones or on other mobile computers controlled by attackers. In our prototype, we implement the recorder and the input detector on a smartphone, and implement the other components on a laptop.

4.8.1 Implementation of Recorder and Input Detector

We implement the recorder and the input detector on a commercial Android smartphone. Its model is Galaxy S5 with Android 6.0.1 as its operating system. The smartphone is equipped with 2.5 GHz CPU and 2 GB RAM.

Even though sometimes there are two built-in microphones on a smartphone, we use only one of them to ensure that WritingHacker is implementable on most mobile devices such as smartwatches. The sampling rate of microphone is 48 kHz, and the format of audio data is PCM 16 bit per sample.
The working mode of the accelerometer is \textit{SENSOR\_DELAY\_NORMAL}. It has three outputs for three directions: \(a_x(t)\), \(a_y(t)\) and \(a_z(t)\). We calculate
\[
a(t) = \sqrt{a_x(t)^2 + a_y(t)^2 + a_z(t)^2}
\]
as the amplitude of acceleration, and set \(W_a = 1000\), \(\Gamma_a = 2.4\).

The input detector reads the output of recorder in real time and buffers the sound fragments. The original sound stream is also stored to be further processed by the hand motion tracker. We set \(W = 200\), \(\gamma_1 = 16\), \(\gamma_2 = 1\), \(\Gamma_1 = 45\) ms, \(\Gamma_1' = 68\) ms, and \(\Gamma_2 = 227\) ms, which are fine-tuned based on the performance of the input detector on a validation dataset.

4.8.2 Implementation of Hand Motion Tracker, Dictionary Filter and Word Recognizer

We have implemented the hand motion tracker, the dictionary filter and the word recognizer on a laptop (Thinkpad T430) using Java and Matlab script. The recorded sound stream and the output of input detector are transmitted to the laptop once the attacker has regained the smartphone.

For the dictionary filter, we implement three separate dictionary databases according to different attack scenarios. For the scenario of daily life, we use the top-2000 and top-5000 commonly used word lists available in [85]. For clinic scenario, we extract all the words in the MIMIC II Clinical Database [86], and then use the top-9000 words as our database. For the word recognizer, we use the API of Spider [104] on Matlab to realize SVM. The features of sound and motion fragments are calculated by Java.

4.9 MEASUREMENT AND EVALUATION

In this section, we evaluate the performance of our prototype system by conducting experiments on our testbed in a controlled environment. We set our test environment in one laboratory, where the near-field noise is low and there is no nearby moving object. The range of characters is ‘A’-‘Z’. Twenty two volunteers #1-22 are recruited in a university department. Eleven of them (#1,3,5,7,...,21) are male and eleven of them (#2,4,6,8,...,22) are female. For the experiments in Section 4.9.2-4.9.4, we collect the training data set for the word classifier from two volunteers #1,2. Each volunteer writes each character in print style for 10 times. Thus, for each character the size of training data is 20. Except in Section 4.9.6, we use the same type of ballpoint pens (0.5 mm) during the tests. We will evaluate the impacts of different pen sizes in Section 4.9.6. The handwriting is on a paper with standard
A4 size (210 mm × 297 mm) on a wooden desk. We place the smartphone on the same desk. Except in Section 4.9.6, the distance between the smartphone and the handwriting is around 20-30 cm. The relative angle between them is roughly fixed during the tests. We will measure the impacts of relative angle changes in Section 4.9.7. We train SVMs in the character spaces of $C_1 - C_3$, respectively. The volunteers (#1-22) in the following tests are required to follow the same letter clusters ($C_1 - C_3$). However, before the test they were not aware that their data are used for testing an eavesdropping system.

4.9.1 Performance of Input Detector

In this subsection we evaluate the performance of input detector. Ten volunteers #3-12 write characters ‘A’-'Z' repeatedly on the paper. Each volunteer repeats writing each letter by 40 times. The other settings are the same as those in the training phase. To compare the intercepted sound fragments with the original sound sequences, the recorder is configured to store the original sound sequence in the smartphone’s storage. The total number of the written characters is 10400. According to our results of the comparison between the intercepted sound fragments and the actual sound sequences, the accuracy of letter detection is 95.73%. The false positive rate is 2.80%, while the false negative rate is 1.47%.

The main cause of false positive is the near-field burst noise during the experiment. Even though we have used the accelerometer to reduce its impact, the false positive still exists if the near-field burst noise happens frequently. We find that if the user is knocking on the desk while the near-field noise happens at the same time, the input detector would consider it as the start point of a letter because the output of the accelerometer exceeds the threshold $\Gamma_a \bar{a}$ and the duration of noise exceeds $\Gamma_1$.

The cause of false negative is that sometimes the sound of the volunteers’ handwriting is so weak that the input detector does not detect the start point. One simple solution is making $\gamma_1$ smaller. However, this would lead to a higher false positive rate. The parameter $\gamma_1$ controls the trade-off between the false positive rate and the false negative rate.

4.9.2 Performance of Normal SVM

Before we evaluate the accuracy of word recognizer, we first prove that applying the normal SVM technique to our attack scenario directly can only achieve low accuracy. In this subsection we evaluate the case where only normal SVM is applied to our system. To rule out the benefits of letter clustering and dictionary filtering, all the letters are considered as
in the same cluster, and we evaluate the accuracy of letter recognition only. The component of hand motion tracking and LTLO are disabled, and thus the hand motion based features are not used. Besides the features in Equation (3.10), the feature extraction also takes the number of strokes as a normal feature for a single letter.

We firstly consider the common case where the attacker has no access to the labeled training data from the victim. After the system has been trained with the training data set from the two volunteers #1,2, we use the smartphone to record the handwriting of ten other volunteers #13-22. During the test the three volunteers use a different wooden desk from that used during the training stage. The range of characters is ‘A’-’Z’. The volunteers write each character in print style 20 times. The other settings are the same as those in the training stage.

To show the impacts of the unavailability of victims’ training data and the changing writing locations, we also consider the case where the labeled training data from the victim are available. In the training stage, we collect the training data from the ten volunteers #13-22 individually. Each letter in ‘A’-’Z’ is repeated 20 times. During the training, the locations of smartphone and handwriting are fixed. In the test stage we design two test cases. In both cases, each volunteer writes letter ‘A’-’Z’ 20 times. However, in Case 1, the handwriting locations are randomly distributed within 30 cm from the smartphone on a different desk. In Case 2, the handwriting is at exactly the same location on the same desk as that in the training stage.

![Figure 4.7: Accuracy of recognition for letters ‘A’-‘Z’ using SVM only.](image)

We calculate the average accuracy of letter recognition for all cases. The results are shown in Figure 4.7. When the training data from the victims are unavailable, the average accuracy is 21.58%. The average accuracies for Case 1 and Case 2 are 39.58% and 67.33%, respectively.

In Case 2, since the writing locations are fixed on the same desk, the multi-path effect in
handwriting sounds’ propagation is eliminated, which leads to the relatively high accuracy. In Case 1, even if the training data from the victims are still available, the change of writing surfaces and locations causes the mismatch between the training data and the test data. Although we have avoided using the amplitude-related features during feature extraction, the multi-path effect impacts the other features such as bandwidth and pitch frequency. Since attackers can hardly detect the accurate writing locations of victims, it is hard to rule out this negative effect.

It is obvious that when the labeled data from victims are not available, the average accuracy drops significantly. Since the training and test data are collected from different people, victims’ various writing habits and changing writing locations lead to this drop. Even if the word-level semantic analysis can be applied to exclude some obviously-wrong letter recognition results, it is still hard to hit the real written word because of this low letter recognition accuracy. Thus we prove that using normal SVM technique only cannot achieve our design goals, and it is necessary to apply our newly designed methods.

4.9.3 Performance of Word Recognizer

In this subsection we evaluate the accuracy of the final output of WritingHacker. We continue to use the training data collected from the two volunteers #1,2. For the dictionary database, we use three different sets of words: top-2000 commonly used daily words, top-5000 commonly used daily words, and top-9000 commonly used medicine-related words from the MIMIC II Clinic database.

We repeat the test for each individual database. In each test, each of our 20 volunteers #3-22 writes 1000 words selected from the database. For each word in the database, we assign its frequency of appearance as its weight. To select a word for test, we randomly select a word from the weighted database, i.e, the probability for a word to be selected is equal to its appearance frequency divided by the sum of all the frequency values in the database. Following this rule, during each test we collect 20000 samples. The results are shown in Figure 4.8. The error bars show the second highest and the second lowest accuracy among the 20 test results of the 20 volunteers. We also repeat the tests for the case that words are selected from the databases following uniform distribution. Furthermore, we collect 20 training data for each letter from each volunteer in #3-22, and evaluated the accuracy when the training data from victims are available. The selection of test words in this case also follows uniform distribution. The results are also included in Figure 4.8.

The results of experiments show that, under the condition of print-style writing, low near-field noise and no nearby fast-moving objects, the word recognizer of our system achieves an
accuracy of 55%-71% for the frequency-based word selection, and 69%-87% for the uniform word selection. Moreover, the results for the medicine-related words confirm that Writing-Hacker can work well in specific attack scenarios such as clinics or hospitals. It can also be seen that the accuracy achieved when we apply frequency-based word selection is lower than that when words are uniformly selected. The reason is that the words used at a higher frequency are usually shorter, which enlarges the average size of candidate word lists in the case of frequency-based word selection and thus reduces the average accuracy. Usually, a victim only writes a few words in a privacy-related form, and the short function words such as ‘the’ are often omitted. Thus in the real case a higher accuracy should be achieved, which is confirmed by the test result that the recognition accuracy for the MIMIC II Clinic Database is higher than that for the daily words.

The system achieves high accuracy when the training data from the victim are available, which implies the possibility for a new input method. We discuss this in the next section.

4.9.4 Contribution of Different Components

To better understand the contribution of the dictionary filter, the word classifier, LTLO and the hand motion tracker, we analyze the change of the recognition accuracy when different components or subcomponents are disabled. In Case 1, we use dictionary filter only and the final recognition result is selected from the candidate word list uniform-randomly, which rules out the contribution of hand motion tracker and the word recognizer. In Case 2, the multi-class SVMs are also applied, and only the features extracted from sound fragments
are used. In Case 3, LTLO is also applied. In Case 4, the features extracted from motion fragments are further used. The same data from the previous test are used, and we consider both the frequency-based word selection and the uniform word selection. The results are shown in Figure 4.9(a) and 4.9(b).

![Graph showing accuracy of recognition](image)

Figure 4.9: Accuracy of recognition when different components and subcomponents are used.

The results of Case 1 show that, since the dictionary filter narrows the space of candidate words, over 38% of written words can be correctly recognized even without any machine learning method. The results of Case 2, 3 and 4 show the improvement made by the multi-class SVMs, LTLO and the hand motion tracker, successively. Therefore, the contribution made by each of our methods is confirmed.

During the previous tests we assume that the words for test are included in the dictionary database. In the real case, it is possible that the written word does not hit the database. Although Zipf’s law ensures that the hit rate would be high enough if the size of database is sufficiently large, we still design a test to evaluate the impact of the dictionary database’s size in the following subsection.

4.9.5 Impacts of Dictionary Database Size

The size of the dictionary database controls the trade-off between the hit rate and the recognition accuracy under the condition that the word has hit the database. A larger database size implies that the word written by the victim is covered by the database with higher probability. However, this increases the space of candidate words for word recognizer and thus would reduce the accuracy even if the word has hit the database. On the other
hand, if the database is too small, it is highly probable that the word written by the victim does not exist in the database, which leads to the failure of recognition directly. Fortunately, since the frequency distribution of commonly used words follows Zipf’s law, it is possible for attackers to maintain a relatively small database while achieving an acceptable accuracy.

To evaluate the impact of dictionary database size, we use the top-9000 commonly used medicine-related words as a baseline database, and change the size of the dictionary database from top 1000 to 9000 with step 500. Similar to the previous tests, we assign the frequency of appearance as a word’s weight in the baseline database. The words for test are randomly selected from the weighted baseline database, where the probability for a word to be selected is proportional to its appearance frequency. If the selected word is not covered by the dictionary database, the recognition for this word fails. Otherwise, it is recognized with the dictionary database. We collect training data from volunteers #13,14 following the same method used in the previous tests, and collect 10000 test samples from volunteers #3-12 for each size of the dictionary database.

Figure 4.10 shows the average accuracy when the size of dictionary database varies. We also draw the theoretical hit rate as well as the recognition accuracy when only the dictionary filter is used, i.e., the output is uniform-randomly selected from the candidate word list. When the size of the dictionary database is much smaller than that of the baseline database, the enlargement of database size significantly increases the hit rate, and thus improves recognition accuracy. However, when the database’s size is large enough, because of the Zipf’s distribution of word usage, the growth rate of the hit rate is much lower. Meanwhile, the increased database size enlarges the candidate word lists, which in turn reduces the recognition accuracy. This effect can be clearly observed in the case where only the dictionary filter is used: the accuracy drops when the database size is larger than 7500 - 8000. The usage of SVM, LTLO and hand motion tracker slows this drop. However, the slow-increasing hit rate makes the average recognition accuracy no longer increase significantly.

These results imply that it is unnecessary for the attacker to collect all the possible words for the specialized attack scenario. When the size of the dictionary database has been large enough, collecting more rarely used words does not help a lot. These results also confirm that even though our previous experiment uses subsets of English words only, the experiment still reflects the performance of WritingHacker in the real-word case because the top commonly used words have been covered by the dictionary databases.
Figure 4.10: Accuracy of recognition when size of dictionary database changes.

Figure 4.11: Average accuracy of recognition when distance, desk material and pen size change.

4.9.6 Impacts of Distance, Surface Material and Pen Size

One common concern about our attack method is the impact of the distance between the attacker’s device and the victim’s handwriting. To avoid being noticed by the victim, it is possible that the attacker can only deploy the device such as the smartphone far away from the victim. The hand motion tracker does not work well in this case due to weak reflected signals. In this subsection we prove that WritingHacker still maintains a good performance even for a much longer distance between the device and the handwriting. Moreover, we also evaluate the performance of WritingHacker when a different pen size (1.0 mm) is used.

During the test we change the distance between the smartphone and the volunteer’s handwriting from 40 cm to 200 cm with step 40 cm. Since the hand motion tracker does not work well when the distance is larger than 35 cm, the hand motion tracker is disabled. For each test ten volunteers #13-22 write 1000 words which are randomly and uniformly selected from the top-2000 commonly used daily words. The training data come from volunteers #3,4, which are collected following the same method used in the previous tests. We repeat the test on two desks with different materials: wood and plastics. To evaluate the impacts of different pen sizes, we also require that the volunteers use 1.0 mm ballpoint pens and repeat the training and tests.

The experimental results in Figure 4.11 show that, although the accuracy drops slightly due to the disabled hand motion tracker, WritingHacker still keeps a recognition accuracy higher than 61% in all the cases even from 200 cm away. Moreover, within the range of 40-200 cm, the accuracy only decreases slightly. This is caused by the truth that acoustic waves propagating along surfaces follow a lower attenuation rate, and thus the handwriting signals received by the smartphone through the desks are still identifiable. The comparable
results for the wooden/plastic desks and the 0.5 mm/1.0 mm pens imply that it is possible for WritingHacker to maintain performance on different types of surface materials and different pen sizes.

4.9.7 Impacts of Relative Angle Changes

In the previous experiments, the relative angle between the device and the handwriting is roughly fixed during training and tests. In this subsection, we relax this requirement, and evaluate WritingHacker’s performance when the relative angle during tests is different from that during training.

During the tests, we change the relative angle from $-90^\circ$ to $90^\circ$ with the interval of 22.5°, where 0° means that the relative angle is the same as that during training. The distance between the device and the handwriting is still around 20-30 cm. For each angle, each of the volunteers #3-12 writes 100 words that are uniformly selected from the top-2000 commonly used daily words. The training data are the same with those used in Section 4.9.5.

We calculate the average word recognition accuracy for each pair of symmetric angles, and the results are shown in Figure 4.12. When the angle difference is within $\pm45^\circ$, the performance of WritingHacker is barely impacted. When the angle difference is larger than $\pm67.5^\circ$, the angle error cancels the contribution of the hand motion tracker, which should be disabled in this case. Therefore, even if the precise relative angle is unknown to the attacker, as long as the difference between the true angle and the attacker’s estimated angle is within $\pm45^\circ$, the accuracy is not obviously impacted.
4.9.8 Performance for Lowercase Letters

The previous experiments are conducted with capital letters. In this subsection, we evaluate the performance of WritingHacker with lowercase letters, which are divided into two clusters:

\[ C_1 = \{ 'a', 'b', 'c', 'e', 'g', 'h', 'l', 'm', 'n', 'o', 'q', 'r', 's', 'u', 'v', 'w', 'y', 'z' \}, \]

\[ C_2 = \{ 'd', 'f', 'i', 'j', 'k', 'p', 't', 'x' \}. \]

We collect new training data from volunteers #5,6 and test data from volunteers #13-22, following the same method in Section 4.9.3. The results are shown in Figure 4.13.

The results show that, the accuracy is 50%-68% for the frequency-based word selection, and 72%-77% for the uniform word selection. Therefore, it is shown that WritingHacker still maintains a good performance for lowercase letters. The slight decrease in accuracy is caused by the truth that, the stroke number diversity of lowercase letters is lower than that of capital letters, which increases the candidate word search range.

4.10 DISCUSSION

4.10.1 Impacts of Nearby Moving Objects and Unknown Relative Position

To use the hand motion tracker, the approximate relative angle between the device and the handwriting should be known to the attacker, and the distance between them should be shorter than 35 cm. Otherwise, the hand motion tracker does not benefit the system performance. If there are other moving objects around the victim during the attack, the motion tracking accuracy would also be impacted. In these case, the hand motion tracker should be disabled. However, the results of Case 3 in Figure 4.9(a) and 4.9(b) show that, even if we do not use the hand motion tracker, WritingHacker can still achieve the accuracy around 60% and 70% for different word selection methods. Therefore, WritingHacker can still maintain a good accuracy even if there are nearby moving objects and the relative position between the device and the handwriting is unknown.

4.10.2 Possible Ways to Improve Accuracy

In our work we propose a general attack method for unspecified writing scenarios. There are still several possible ways to further improve the performance of our method by taking
the writing constraints into account. For instance, WritingHacker can provide a list of candidates (such as second/third-most-probable words) for each recognized word to help human observation. If a word from the handwriting on the form “Medical History” is recognized as “ANYWAY”, and the second-most-probable word is “ANEMIA”, it is easy for the attacker to correct the result by using the second candidate word.

Currently, we mainly focus on the word-level handwriting recognition, because people often write separate words instead of sentences when filling out forms. However, the sentence-level recognition is still applicable in the scenarios where victims write whole sentences. Attackers can further correct the recognized results based on sentence-level semantic laws and human observation. The candidate words can also help in this case.

4.10.3 Defending Against Attack

To defend against the eavesdropping attack proposed in this chapter, a potential victim should check the desk for suspicious devices such as smartphones or detectaphones. Writing in a noisy environment can significantly increase the difficulty of eavesdropping. However, it is still possible to segregate the sound of handwriting by using multiple microphones [87]. If possible, the potential victim should also avoid print-style writing and use joined-up writing, which invalidates the stroke detector.

Contrary to intuition, writing on a stack of paper is not an effective way to resist the eavesdropping attack. With the same experiment setting as in Section 4.9.3, we require the volunteers to write on a paper stack of 3 cm in thickness, and the words for test are uniform-randomly selected from the top-2000 commonly used daily words. The audio signals received by the microphone are still strong enough, and the recognition accuracy achieves 73%. This is because the attenuation rate of audio signals in solid media is low, and even the 3 cm paper stack can hardly block the signals.

4.10.4 Limitations of WritingHacker

As shown by the experiment settings, currently WritingHacker only works under controlled environments. We focus on the circumstance where victims follow certain print-style writing, and the near-field noise is low. In our evaluation, we collect data from 22 volunteers, which seems to be a relatively small subject population, because a larger subject population would lead to a greater diversity of writing habits. Thus, WritingHacker is still a proof-of-concept system. However, WritingHacker has revealed the danger of a new audio-based attack on handwriting, which is the main purpose of our work. It is our future work to design a fully
functional system and conduct experiments with a larger group of subjects.

4.11 COMPARISON BETWEEN TABLEWRITE AND WRITINGHACKER

![Diagram showing the relationship between the available handwriting information and the handwriting recognition accuracy.]

Figure 4.14: Relationship between the available handwriting information and the handwriting recognition accuracy.

While TableWrite and WritingHacker are designed for different scenarios, they focus on similar signal sources and thus share some common designs. Regardless of the difference between the pen and the finger writing, TableWrite and WritingHacker together show how the handwriting recognition rate shifts with the various information known to the system. As illustrated in Figure 4.14, a higher recognition accuracy can be achieved when more information about the handwriting is available. WritingHacker focuses on the case where no training data are available from the victim. By recognizing the stroke number and utilizing the dictionary knowledge, we can achieve an accuracy around 50%. When more information such as other people’s training data and hand tracking results are available, the recognition rate keeps increasing and can reach 70%. TableWrite considers the case where the training data can be directly collected from the user. The accuracy can be higher than 90% if the hand motion tracker is also trained by the user. Thus, Figure 4.14 illustrates the amount of information needed to restore the handwriting content.

On the other hand, the different application and attack scenarios make the designs of TableWrite and WritingHacker differ. Working as an input method, TableWrite can be fully configured by users. Users can control the writing settings such as the dictionary used, and the relative position between the device and the writing hand. Compared with TableWrite,
WritingHacker needs to handle more complex situations because the attacker has no control on the victim. For example, depending on different writing scenarios, we need to consider the impacts of different dictionary types and sizes. Since the user’s writing position is previously unknown, we also consider in which situation the hand motion tracker should be disabled. Due to the lack of the victim’s training data, the word recognition rate drops, and we design LTLO to improve the accuracy. These designs are not effective if applied to TableWrite, where users have already controlled the unknown factors.

4.12 CONCLUSION

In this chapter, we present WritingHacker, a prototype system which explores the possibility of audio-based eavesdropping on handwriting via mobile devices. Based on the letter clustering, we design the components of input detector, hand motion tracker, dictionary filter and word recognizer, which enable mobile devices to record and recognize victims’ handwriting. WritingHacker makes it possible for an attacker to violate a victim’s privacy by keeping a mobile device (e.g., a smartphone) touching the desk being used by the victim. The experimental results show that, under certain conditions, the accuracy of word recognition reaches around 70% - 80%, which reveals the danger of privacy leakage through the sound of handwriting.
CHAPTER 5: SHOESLOC: IN-SHOE FORCE SENSOR-BASED INDOOR WALKING PATH TRACKING

5.1 INTRODUCTION

In recent years, the rapid evolution of mobile sensing technology has triggered the rise of smart shoes [22, 21, 58], the devices that can measure the motion of users’ feet and analyze their activities. One typical design of smart shoes is based on force or piezoelectric sensors. For instance, Nike+ sensors [59] are used to count the steps of runners, and Stridalyzer [22] is a typical force mapping system that helps analyze the gaits of users. In this chapter, we further explore the capability of in-shoe force sensors, and prove that it is possible to track the walking paths of users in indoor spaces based on force changes in shoes.

Indoor tracking and localization have been popular research areas. Most of the previous approaches are based on wireless anchors [34, 35, 36, 37, 38, 39, 40, 41, 42, 43]. For instance, Wi-Fi access points generate specific signal strength distributions in indoor space, and mobile devices such as cellphones measure the signal strength to locate themselves [44, 45, 46, 47, 48, 49]. However, the anchors need to be pre-installed, and site surveys are usually necessary, which lead to extra cost. In some large-scale indoor spaces such as shopping malls and airports, the number of Wi-Fi access points that cover each location is often limited, because they are deployed to provide network access instead of localization service [105]. Moreover, in many cases such as power outage, the anchors do not work. To solve these problems, some previous works proposed inertial sensor-based methods [50, 6, 51]. The accelerometers, gyroscopes and compasses in cellphones or Inertial Measurement Units (IMUs) are used to measure the motion of users and thus estimate the walking paths. However, the electronic compasses can be interfered by magnetic field changes [52, 53, 54], especially for the scenarios where the magnetic field changes significantly [55, 56], such as in manufacturing plants [57]. In contrast with the previous approaches, if we use force sensors alone to locate users, the cost of the anchor installation and site survey can be eliminated. Moreover, the in-shoe force sensors are hardly interfered by magnetic field changes.

In this chapter, we present ShoesLoc, an indoor walking path tracking method which is based on in-shoe force sensors. The main idea is that, based on the signals from the force sensors deployed in insoles, we estimate the walking direction change and the stride length of each step made by the user. We then combine this information with the constraint of barriers on floor maps to determine the walking path and the current position of the user.

The main challenge is that, without the compass, the walking direction of the user is unknown, which makes it hard to track the walking trajectory. To solve this problem,
ShoesLoc estimates the relative angle change of the walking direction instead. We extract features from the force signals, and train a Support Vector Machine (SVM) regression model to estimate the direction change made by each step of the user. Then, by accumulating the direction changes of the steps, we can track the change of the user’s heading direction. The stride length of each step can also be estimated in a similar way. To further determine the absolute walking path of the user, we apply a particle filter to utilize the constraint of the barriers on floor maps. The wrong walking trajectories that go across barriers such as walls are eliminated in this way.

Moreover, based on the observation that a long straight walking trajectory must be parallel to one of the straight hallways on the map, we propose the direction correction algorithm to improve the performance of the particle filter, which reduces the cumulative direction error and the computation time. Furthermore, to handle the common case that carrying handbags or backpacks changes the force distribution in shoes, we propose the weight normalization method to normalize the force data, and thus reduce the impact of carrying bags.

The main contributions of our work are:

- To the best of our knowledge, we are the first to explore the possibility of using in-shoe force sensors to track the walking paths of users, which has low deployment cost and is resistant to magnetic field changes.

- To address the major challenge of estimating walking trajectories without inertial sensors, we design the direction change and stride length estimation methods based on force signals, and apply the particle filter to determine walking paths.

- We propose the direction correction algorithm to further improve the performance of the particle filter. We also propose the weight normalization method to handle the impact of handbags and backpacks.

- We implement a prototype system and conduct extensive evaluations. Experimental results show that ShoesLoc can achieve the average location error of 0.9-1.3 m.

Besides being used as a new user tracking technique for smart shoes, ShoesLoc is also specifically effective for the scenarios where the anchors for localization are not available or the magnetic interference is non-negligible, such as in power outages or in the radiology departments of hospitals.

We organize the rest of this chapter as follows. After introducing the related work in Section 5.2, we present an overview of ShoesLoc’s framework in Section 5.3. Then, we successively introduce the methodologies and algorithms used by the system components...
in Section 5.4-5.7. In Section 5.8 and 5.9, we evaluate the performance of ShoesLoc with comprehensive experiments. We have further discussion in Section 5.10, and conclude this chapter in Section 5.11.

5.2 RELATED WORK

*Indoor localization and tracking:* The research on indoor user tracking is tightly related to the indoor localization technology, and they have similar technical foundations. Multiple technologies have been applied in the field of indoor localization. For instance, there are systems based on radio frequency identification [106, 107], ultra-wideband transmission [34, 35], Bluetooth [36, 37], ultrasonic [38, 39] and video signals [108]. However, these methods require the installation of anchors in indoor spaces to communicate with users’ devices or generate specific signal distributions. Therefore, they do not work in the regions where no anchors are deployed. The Wi-Fi received signal strength (RSS)-based localization technology solves this problem by using the widely installed Wi-Fi access points as the anchors [44, 45, 46, 109]. However, there are cases where the access points are out of power or do not work, such as in power outages. Moreover, a site survey is usually needed to learn the RSS distribution on the whole floor, which leads to extra cost.

Compared with the indoor localization methods, the indoor tracking methods [40, 41, 42, 43] usually achieve a higher location accuracy because they can make better use of historical localization results by further applying techniques such as Kalman filters [47, 48, 49]. However, with the same fundamentals as the indoor localization technology, the previous indoor tracking approaches suffer the same aforementioned problems. On the other hand, the surveillance camera-based indoor tracking methods [110, 111] require the installation of cameras, which are usually even more expensive than wireless anchors. Some other approaches [112, 113] use cellphone cameras instead to provide indoor navigation services. However, the use of cellphone cameras is energy-expensive and would also cause privacy issues.

In our work, ShoesLoc can work in a larger range of application scenarios because it uses the force sensors in users’ shoes and does not require extra anchors or cameras. Moreover, except the floor map, ShoesLoc does not require any site survey or wireless signal measurements, which reduces the cost of the system setting up.

*Motion sensor-based walking tracking:* To eliminate the need of anchors, some previous works proposed to use the inertial and magnetic sensors to measure the motion of users, and thus track the walking trajectories. For instance, in [6], Zee used the motion sensors in cellphones to estimate the walking direction and count the steps of users, which can be used to recognize the walking paths if the floor map is available. In [51], based on the inertial
sensors in cellphones, the algorithms for step detection, heading direction detection and step length estimation are also proposed to construct the walking paths of users.

However, the accuracy of these approaches is usually impacted by the position or the orientation of the phone carried by the user, which makes them impractical for real-life use [7]. Some other previous works instead utilize the foot-mounted IMUs. For instance, in [63], IMUs are used to measure the motion of users’ feet, based on which the system can provide absolute positioning. In [61, 62], sensor tags are embedded into shoes, which can perceive the user’s moving trace. However, the IMU-based approaches are usually vulnerable to magnetic interference, because the electronic compasses can be impacted by magnetic changes [52, 53, 54]. For example, in [6] it is shown that, the presence of magnetic materials such as metal in buildings can disturb a compass’s perception of North, and the direction error of compasses in an office building can be as high as 30°. Thus, these approaches are not applicable for the regions where the magnetic field changes significantly [57, 55, 56]. In contrast, in ShoesLoc we use force sensors, which are resistant to most environment changes. Therefore, compared with the IMU-based approaches, ShoesLoc can work in more diverse scenarios.

To improve the localization accuracy, many previous motion sensor-based works applied the particle filters [6, 63, 7], which help eliminate the invalid trajectories that go across barriers in maps. However, the particle filters were applied based on the condition that the absolute walking direction can be measured by the sensors. Since the force sensors cannot provide the accurate walking direction, the classic particle filters do not work well because the lower accuracy of the estimated heading direction makes valid particles dropped easily, which causes huge computation burden. To solve this problem, we improve the particle filter by designing the direction correction algorithm, which can correct the walking direction based on the map information and further reduce the computation time.

**Force sensor-based walking tracking:** Some previous works also considered the usage of in-shoe force sensors. However, most of them only use force sensors to count steps or analyze the gaits of users [114, 115]. In [116], the force sensors are used to count the steps made by the user, and thus help the inertial and magnetic sensors to track the gait states. Some commercial products such as the ReTiSense Stridalyzer [22] use force sensors to generate the force maps in shoes, which help doctors analyze the walking patterns of users. To the best of our knowledge, our work is the first to use in-shoes force sensors to track the walking paths of users.
5.3 SHOESLOC OVERVIEW

In this section, we first show our basic assumptions, and then give an overview of the sensor deployment and the system architecture of our tracking method.

5.3.1 Basic Assumptions

The design of ShoesLoc is based on the assumption that the floor plan of the walking region is available to the user’s device. We consider this assumption reasonable, because there are multiple possible sources of floor plans, such building management offices and Google Maps. We also assume that the structure of the floor plan is not completely centrosymmetric. Otherwise, there are always at least two possible positions for the user. We could utilize the landmarks on the floor plan to weaken this assumption, which will be discussed in Section 5.10.1.

We assume that the smart shoes are equipped with a sufficient number of force sensors, as will be shown in Section 5.3.2. In our design, we focus on the walking speed of 1 - 3 m/s. A lower walking speed could blur the difference between straight and turning steps, and a higher walking speed usually means that the user is running, which is uncommon in indoor spaces and is also out of our scope.

5.3.2 Sensor Deployment

Ideally, to monitor the force change on the whole insole, we should deploy a mass of force sensors to cover each point on the insole. However, because of the cost restriction, most of the commercial smart shoes and foot force mapping systems deploy only a limited number of force sensors for each insole. One typical deployment is shown in Figure 5.1. For each insole, five force sensing resistors are deployed at the positions of inside metatarsals, outside metatarsals, outside midfoot, arch, and heel.

In our work, we use the ReTiSense Stridalyzer insoles [22] to measure the force changes at the aforementioned five positions. We assign indices \(i = 1, \ldots, 5\) to the five sensors on each insole. When the user is walking, each sensor keeps measuring the force with the sampling rate of 15 Hz. Therefore, the inputs for our system are 10 discrete time series, denoted by \(\{u_l^i(t), u_r^i(t) | i = 1, \ldots, 5, \ t \geq 0\}\), where \(u_l^i(t)\) is the data sequence received from sensor \(i\) in the left insole, and \(u_r^i(t)\) is from sensor \(i\) in the right insole. Although \(u_l^i(t)\) and \(u_r^i(t)\) are discrete time series, we write them as continuous time series for convenience, which does not impact our following analysis. We will show that it is possible to further relax the sensor
number requirement during our experiments in Section 5.8.1 and 5.8.2.

Figure 5.1: Force sensor deployment for each insole.

5.3.3 System Architecture

As shown in Figure 5.2, our system mainly consists of the following components.

Data preprocessor: This component receives the raw data \( \{u_i^l(t), u_i^r(t)|i = 1, ..., 5, t \geq 0\} \) from the force sensors, and then prepares the data for analysis. It first segments the data for each step made by the left or the right foot, and then adjusts the amplitude of the data to reduce the impact of the changing weight of the user. The outputs are the step segments from the left and the right insoles. Each step segment contains the force sequence from each sensor in each shoe for each step made by the user.

Direction change estimator: This component estimates the direction change made by each left or right step. For the step segments of each step, it first extracts a set of features, and then calculates the angle change based on SVM. Note that the output is the relative angle change made by each step instead of the user’s absolute walking direction.

Stride length estimator: To handle the impact of the changing walking speed, this component tracks the change of the stride length based on the step segments. Similar to the direction change estimator, it roughly estimates the stride length of each step based on SVM.

Walking path recognizer: This component determines the walking path with an improved particle filter. The inputs are the angle change sequence and the stride length sequence, as well as the map of the walking region. To handle the issue of high cumulative errors, we further design the direction correction algorithm to better utilize the map information.
Before the system can be used, a user needs to provide training data to train the components during the offline stage. The training format will be shown in Section 5.5.3 and 5.6.3. During the online stage, our method can track the position of the user after a convergence phase.

Next, we introduce the algorithms of these components in detail, respectively.

5.4 DATA PREPROCESSOR

The key design point of this component is to crop the force data sequences for each step, and further process the data to remove the effect of the changing body weight caused by carrying backpacks or handbags.

5.4.1 Step Segmentation

We process \( \{ u^L_i(t) | i = 1, ..., 5 \} \) and \( \{ u^R_i(t) | i = 1, ..., 5 \} \), respectively. To accurately detect the start and end point of each step, we focus on the total foot force on the insoles, i.e., \( u^L(t) = \sum_{i=1}^{5} u^L_i(t) \) and \( u^R(t) = \sum_{i=1}^{5} u^R_i(t) \). One example is shown by the last subgraph in Figure 5.3. A naive method is finding the peaks in the signal sequence and each peak denotes one step. However, due to the various walking state, sometimes two peaks appear for each step (e.g., peak 1 and 2 in Figure 5.3), which is especially the case when the user is walking at a low speed.

Inspired by the fact that the force made by a foot tends to be 0 when the foot has left the ground, we segment steps by detecting the low values of force data. Take the data from the left shoe as an example, a potential start point of a step is detected if \( u^L(t) \geq \gamma_1 \), and a potential end point of the step is detected if \( u^L(t) < \gamma_1 \), where \( \gamma_1 \) is a constant parameter and should be much smaller than the body weight of the user. In our implementation, we set \( \gamma_1 = 10 \) kg. To avoid the impact of the small noise peaks that are higher than \( \gamma_1 \), a step is detected only if the max force value between its potential start and end point is higher than a constant threshold \( \gamma_2 \). Clearly, \( \gamma_2 \) controls the robustness to the noise peaks, and we set \( \gamma_2 = 20 \) kg. If the max force value is lower than \( \gamma_2 \), we discard this potential step. One discarded potential step is shown in Figure 5.3. For each step, we crop the \( u^L_i(t) \) (or \( u^R_i(t) \)) between its start and end point, and denote the step segments by \( s^L_{i,k}(\tau) \) (or \( s^R_{i,k}(\tau) \)), where \( i = 1, ..., 5 \), and \( k = 1, 2, ... \) is the index for each left (or right) step. \( \tau \) denotes the time since the start point of the step, i.e., \( \tau = 0 \) at the beginning of the step segment. One example of the step segments for a single left step is labeled by five red boxes in Figure 5.3.
Figure 5.3: Data from the sensors while the user is walking. The data of the right foot between the two dashed straight lines show the force changes during a right turn.

5.4.2 Weight Normalization

It is common that people would carry handbags or backpacks when they are walking, which changes body barycenters and thus changes the amplitude and distribution of the force on the sensors. To address this issue, we design a method to normalize the force data from each sensor during the online stage.

Note that it is invalid to normalize the force segments by simply dividing the signals by their maximum or average values, because the force distribution is not proportional to the user’s weight. Since we mainly focus on the amplitude changes of the force peaks in the step segments, we normalize the peak amplitudes only.

During the offline stage, the user is required to provide training data to train the direction change estimator and the stride length estimator, which will be introduced in Section 5.5.3 and 5.6.3. We further show that the training data can also provide information for normalization. We denote the cropped step segments in the whole training data by $\tilde{s}_{i,k}(\tau)$ and $\tilde{s}_{i,k}^r(\tau)$, where $i = 1, ..., 5$, $k = 1, ..., K$ and $\tau$ is in the range of the segment’s time length. Then, we calculate the maximum and minimum value of the peak amplitudes for each sensor by

$$
\tilde{s}_{i,\text{max}}^l = \max_{k=1,...,K} \max_\tau \tilde{s}_{i,k}^l(\tau), \quad i = 1, ..., 5,
$$

(5.1)
\[
\tilde{s}_{i,k}^{l,(\tau)} = \min_{k=1,...,K} \max_{\tau} \tilde{s}_{i,k}^{l}(\tau), \quad i = 1,...,5. \tag{5.2}
\]

In case the training data contains outliers, we can delete the smallest and the largest 2% of \( \max_{t} \tilde{s}_{i,k}(\tau) \) during the calculation. \( \tilde{s}_{i}^{l,\max} \) and \( \tilde{s}_{i}^{l,\min} \) can be calculated in the same way.

During the online stage, while the user is taking the \( k \)th left or right step, we calculate \( s_{i}^{l,\max}, s_{i}^{l,\min} \) or \( s_{i}^{r,\max}, s_{i}^{r,\min} \) among the previous \( G \) steps (i.e., step \( k - G \) to \( k - 1 \)). The parameter \( G \) must be large enough to guarantee that the user makes multiple turns in the \( G \) steps. We set \( G = 100 \) during our implementation. The calculation is similar to Equation (5.1) and (5.2), and the same outlier elimination method is applied. For the \( k \)th left step, the normalized force segments are

\[
\tilde{s}_{i,k}^{l}(\tau) = \left( \tilde{s}_{i,k}^{l}(\tau) - s_{i}^{l,\min} \right) \frac{s_{i}^{l,\max} - s_{i}^{l,\min}}{s_{i}^{l,\max} - s_{i}^{l,\min}} + s_{i}^{l,\min}, \quad i = 1,...,5. \tag{5.3}
\]

For the \( k \)th right step, \( \tilde{s}_{i,k}^{r}(\tau), \quad i = 1,...,5 \) are calculated in a similar way. The intuition is that, the range of the force peak amplitudes measured in the online stage is normalized to be the same with that of the peak amplitudes in the training data. Note that after the normalization, the force value around the start and end point would be smaller than 0. However, it does not impact the feature extraction in the following components.

One common scenario is that, the user is carrying a bag with a single hand or a single shoulder, which causes the unbalance of the force changes on different sensors. This unbalance can be complex and vary with different walking habits. For instance, when the user is carrying a heavy bag with the left hand, the body can either tilt to the left or right, which leads to different distributions of the signals’ peak amplitudes on the ten sensors. As shown by Equation (5.3), our normalization method handles this problem by normalizing the force data from different sensors separately. After weight normalization, the range of the force peak amplitudes on each sensor is the same with that in the training data. Therefore, the unbalance of the force changes is eliminated.

For the \( k \)th step of the left/right foot, the data preprocessor outputs the normalized force sequence for each sensor \( i \) in the shoe as \( \tilde{s}_{i,k}^{l}(\tau) \) or \( \tilde{s}_{i,k}^{r}(\tau) \), which is sent to the direction change estimator and the stride length estimator.

### 5.5 DIRECTION CHANGE ESTIMATOR

When a user makes a turn during walking, the force distribution in the shoes changes. In this section, we show that it is possible to estimate the walking direction changes based on the force changes on the insoles. We first show our observations for the turning patterns
of human, and then introduce how the direction change estimator extracts features and estimates the direction changes.

5.5.1 Turning Pattern Observation

It is intuitive that when a person makes a turn, the force made by the feet on insoles changes. Figure 5.3 shows an example of the force changes during turning. The person makes a right turn around the 3,000th millisecond. We can observe that the amplitude of the total force also changes. This is caused by the fact that when the person makes the right turn using the right foot, the maximum force on the right inside metatarsals drops, as shown by the first subgraph of Figure 5.3. Therefore, it is possible to recognize the turning actions according to the force changes on insoles. We further explore the possibility of estimating the angle changes of the walking direction based on the force data.

In ShoesLoc, we keep measuring the force data from both shoes. This is based on the observation that, the data from one shoe alone are not sufficient enough for turning detection. As illustrated by Figure 5.4(a) and 5.4(b), there are at least two possible ways for a user to make a right turn. In Figure 5.4(a), during the turning the user turns the body by putting extra force on the left foot. We define a key step as the step on which the user puts extra force to change the body direction. In Figure 5.4(a) the left foot makes the key step, while in Figure 5.4(b), the right foot makes the key step. Apparently, the force data of key steps contain more information for direction change measurement. However, the non-key steps do not have distinct force change and thus would not help the turning detection. For instance, in the case shown by Figure 5.3, the person makes the right turn following the turning type shown in Figure 5.4(b). The right foot makes the key step, and the data from the left shoe do not show obvious change. Therefore, it is necessary to measure the force change in both shoes, otherwise some turning actions would not be detected.

Moreover, simply classifying the walking actions as “going straight” and “making a left/right turn” is not sufficient for walking path tracking. One example is shown in Figure 5.4(c). When the user is making a 90° right turn at a low speed, it is possible that there are multiple key steps, and each key step makes a direction change smaller than 90°. However, by accumulating the angle changes of the key steps, we can still get the correct direction change. On the other hand, the paths in maps are not necessarily latticed. For instance, in Figure 5.4(d) there are two possible walking paths after the user makes a right turn. Therefore, it is necessary to estimate the direction change made by each step, instead of simply classifying the step as “straight” or “turning”.

In the direction change estimator, we extract features from the step segments of each
left/right step, and then estimate the direction change made by the step based on a trained SVM.

5.5.2 Feature Extraction

For the step segments of each step, we extract a set of features. The pattern in the features should not be highly impacted by the user’s walking speed, and thus the features we use are based on the amplitude of the force signals. Specifically, for the normalized step segment $\bar{s}_{l,i,k}(\tau)$ or $\bar{s}_{r,i,k}(\tau)$, the features are

$$F_{l,i,k} = \max_{\tau} \bar{s}_{l,i,k}(\tau), \quad i = 1, 2, ..., 5,$$

$$F_{r,i,k} = \max_{\tau} \bar{s}_{r,i,k}(\tau), \quad i = 1, 2, ..., 5.$$  

Therefore, for the $k$th step made by the left or the right foot, the feature set $\{F_{l,i,k}|i = 1, 2, ..., 5\}$ or $\{F_{r,i,k}|i = 1, 2, ..., 5\}$ are sent to the subcomponent of angle regression.

5.5.3 Angle Regression

Training format: We first show how the training data for the angle regression are collected. The range of the direction change is $[-90^\circ, 90^\circ]$, where $0^\circ$ denotes that the user’s walking direction does not change. The positive and negative values respectively denote the angles of the left and right turns made by the user. During the offline training, the user needs to make turns with 9 different angles, i.e., $-90^\circ$ to $90^\circ$ with the interval of $22.5^\circ$, as shown by Figure 5.5. Each time the user makes a turn, only one key step should be made. The training data for each angle include both the cases where the left or the right foot makes the key step. The training data are labeled with the corresponding angles, and are used to
train a SVM regression model. Based on our observations during the experiments, we only consider the direction change within $[-90^\circ, 90^\circ]$ for each footstep. This is because people usually do not make a turn larger than $90^\circ$ within a single step. Instead, they make multiple shorter steps and the direction change of each step is smaller than $90^\circ$. Even if a person makes a $180^\circ$ turn within a single step, it is natural that the velocity direction needs to be reversed, which leads to a stop or a pause in stepping. This stop or pause can be easily detected and ShoesLoc will restart the tracking process with the current location.

Ideally, the force changes made by the left and right foot should be symmetric, and the size of the training data could be reduced by half. However, in practice people’s walking patterns would be asymmetric [117], which means the feature patterns of left and right foot are asymmetric. Moreover, the diversity of the hardware used for measurement makes the models trained for left and right foot unexchangeable. Therefore, it is necessary to collect training data from both insoles.

**SVM regression:** In the offline stage, we extract the features from the training data following the method in Section 5.5.2, and then train two linear kernel-based SVM regression models for the left and right foot, respectively.

In the online stage, for the $k$th left or right step made by the user, the angle regression subcomponent receives the extracted features $\{F_{i,k}^l|i=1,2,...,5\}$ or $\{F_{i,k}^r|i=1,2,...,5\}$. Then, it predicts the angle change caused by the step with the corresponding trained model. The results are bounded in the range of $[-90^\circ, 90^\circ]$. While the user keeps walking, we get two sequences of angle changes caused by the left and right steps. We then merge the two sequences, and index them according to their time stamps. Thus, the output of the angle regression is a single angle change sequence, denoted by $\{\Delta\theta_n|n=1,2,\ldots\}$. The left subgraph of Figure 5.6 shows one example of $\{\Delta\theta_n\}$. The performance of the direction change estimator will be evaluated in Section 5.8.1.

![Figure 5.6: Example of the angle change sequence and the stride length sequence.](image)
5.6 STRIDE LENGTH ESTIMATOR

To track the walking path of the user, counting steps and estimating walking direction are insufficient, because the user’s stride length would change. Most of the previous approaches estimate the stride length by using inertial sensors [118]. With ShoesLoc, we prove that it is possible to estimate the stride length based on the force changes on insoles. To give an estimate for the stride length of each step, the stride length estimator extracts another set of features and also applies an SVM regression algorithm. We first show our observation for the relationship between the stride length and the foot force, and then introduce our detailed design.

![Figure 5.7: Data from the left shoe when a user walks with different average stride lengths.](image)

5.6.1 Stride Length Observation

Intuitively, the foot force changes with different stride lengths. One extreme example is that, when a person is running, the stride length is longer compared with that of the normal walking, and the impact force on insoles is obviously stronger. Moreover, the changes of foot force also reflect the stride frequency, which is also correlated with the walking speed and the stride length.
In Figure 5.7, we show the changes of the foot force when a user walks with different stride lengths. Since the stride length is positively correlated to the walking speed during the uncontrolled walking [119], we require the user to walk along the same corridor at different speeds during the test, which makes the average stride length differ. It is shown that, when the user is walking with the longer stride length, the peak values of the total force on the insole increase. This is because the total impact force on the insoles is stronger when the user is walking at a higher speed, which usually means a longer stride length during the uncontrolled walking. Meanwhile, when the stride length is longer, we also notice that the force peaks on the outside midfoot drop obviously, and those on the outside metatarsals also decrease slightly. This reflects the user’s walking habit that the foot force moves toward the body center when he is walking faster with a longer stride length. These clearly show that the foot force on insoles can be used as the features to estimate the stride length.

On the other hand, as shown by the signals between the two dashed lines in Figure 5.7, the time length of a step tends to be shorter when the stride length is longer. The relative time points of the force peaks at some sensor positions also shift. These show that the step time length as well as the force peak positions are also associated with the stride length.

The stride length estimator extracts a set of features based on our observation.

5.6.2 Feature Extraction

We use the normalized step segments $\bar{s}_{l,i,k}(\tau), i = 1, \ldots, 5$ from the left foot as examples to show how the features are extracted. The same process also applies to the step segments from the right foot.

Since the change of stride length is often related to the change of walking speed, the stride frequency is an important feature for stride length estimation. We use the time length of each step segment $\bar{s}_{l,i,k}(\tau)$ as a metric of the stride frequency. Note that for the same step $k$, the time lengths of $\bar{s}_{l,i,k}(\tau)$ for different sensor $i$ are the same. The time length for the $k$th step is denoted as $L_{l,k}$.

Moreover, we calculate each step’s loading rate and unloading rate, which denote the speeds at which the foot force reaches its peak and falls back to near-zero values, respectively:

$$\hat{L}_{i,k} = \frac{\max_{\tau} s_{l,i,k}(\tau)}{\arg \max_{\tau} s_{l,i,k}(\tau)}, \quad i = 1, 2, \ldots, 5,$$  

(5.6)
\[
\tilde{L}_{l,k}^i = \frac{\max_{\tau} s_{i,k}^l(\tau)}{L_k^l - \arg\max_{\tau} s_{i,k}^l(\tau)}, \quad i = 1, 2, \ldots, 5. \tag{5.7}
\]

Note that for each \( s_{i,k}^l(\tau) \), \( \tau \) is in the range of \([0, L_k^l]\). Since \( \arg\max_{\tau} s_{i,k}^l(\tau) \geq L_k^l \) never happens, \( \tilde{L}_{l,k}^i \) is always positive. We also calculate the loading and unloading rate of the total force, i.e., we repeat the calculation in Equation (5.6) and (5.7) for the signal \( \sum_{i=1}^{5} s_{i,k}^l(\tau) \). The results are denoted as \( \hat{L}_{l,k}^i \) and \( \tilde{L}_{l,k}^i \). We also reuse the features extracted in Section 5.5.2.

Therefore, the features extracted from the \( k \)th left step for the stride length regression are

\[
[L_k^l, \hat{L}_k^l, \tilde{L}_k^l, \hat{L}_1^l, \tilde{L}_1^l, \ldots, \hat{L}_5^l, \tilde{L}_5^l, F_{1,k}^l, \ldots, F_{5,k}^l]. \tag{5.8}
\]

The features of the right steps can be calculated in a similar way.

5.6.3 Stride Length Regression

Training format: During the offline stage, the user is required to walk the same distance at different speeds, which lead to different stride lengths. The number of steps are counted and thus we get the average stride length by dividing the distance by the number of steps. For the test with each speed, we label the step segments with the average stride length in this test. The reason why we do not measure the actual stride length of each individual footstep is that, the cost of training data collection would be too high if the user needs to measure each stride length. Therefore, considering the practicality of ShoesLoc, we use the average stride length instead, which is easy to calculate.

SVM regression: Similar to the idea in Section 5.5.3, we train two linear kernel-based SVM regression models for the left and right foot to estimate the stride length. In the online stage, for the \( k \)th left/right step made by the user, the stride length regression subcomponent receives the features as in Formula (5.8), based on which it estimates the stride length of the step. We merge the two sequences of stride length from the two feet, and denote it by \( \{d_n'|n = 1, 2, \ldots\} \). We further smooth the sequence of \( \{d_n\} \) with the exponential moving average (EMA), i.e., \( d_1 = d_1^l \) and \( d_n = \alpha \cdot d_n + (1 - \alpha) \cdot d_{n-1} \) when \( n > 1 \). The coefficient \( \alpha \) is a constant smoothing factor \((0 < \alpha < 1)\), which controls the weights of the current and the previous observations, e.g., a higher \( \alpha \) discounts older stride length values faster. We set \( \alpha = 0.6 \), which is fine-tuned based on the performance of the stride length estimator on a validation dataset. Then the stride length sequence \( \{d_n'|n = 1, 2, \ldots\} \) is the output. One example is shown in the right subgraph of Figure 5.6. Note that \( d_{n-1} \) and \( d_n \) denote the estimated values for different feet. We will evaluate the performance of the stride length
estimator in Section 5.8.2.

5.7 WALKING PATH RECOGNIZER

The component of walking path recognizer receives the angle change sequence \( \{ \Delta \theta_n \} \), the stride length sequence \( \{ d_n \} \), as well as the map of the localization region, and runs a map-based particle filter to locate the user.

5.7.1 Key Idea of Walking Path Recognizer

The key idea is shrinking the possibilities for the user’s location by constraining the estimated walking path with the barriers (e.g., walls) on the map. Initially, the user’s location is unknown to ShoesLoc. However, based on the angle change sequence and the stride length sequence, we can estimate the possible walking trajectories of the user. Only the trajectories on the correct walking path can fit into the map, while the others are eliminated because they imply that the user has run into walls or other barriers. For each eliminated trajectory, we recreate a new valid trajectory based on the surviving ones. Therefore, once all the walking trajectories converge to a unique path, ShoesLoc can locate the user’s location.

Some previous approaches have similar ideas, and apply augmented particle filters to solve the localization problem [6, 7]. Most of them use the accelerometer, gyroscope and compass to measure the user’s walking direction, which helps determine the walking trajectory. However, in our work we are facing a more complex case because we only know the relative angle changes instead of the user’s absolute walking direction. Therefore, we can only have a roughly estimated walking trajectory which has cumulative direction errors. If the cumulative errors are not eliminated, even the correct trajectory could be dropped because of the direction errors. Therefore, the particle filters proposed in the previous works cannot be directly applied, because they are based on absolute direction measurement and cannot handle large cumulative errors, which lead to a low location accuracy and a long computation time. To address this challenge, we improve the particle filter by proposing the direction correction algorithm, which makes better use of the map information than the previous approaches.

**Direction Correction Algorithm:** It is based on the natural observation that when the user is walking straight, the walking direction must be the same with that of some straight hallway on the map. Therefore, when we detect that the user is walking straight, we can correct the current estimated walking direction to the most similar hallway direction. This will efficiently reduce the cumulative errors of the walking direction. Moreover, since most of the hallways are horizontal or vertical on the map, there are usually only four possible
directions of straight paths, which significantly reduces the computation time of the particle filter and makes realtime tracking and localization possible. We discuss the case of irregular-shape hallways in Section 5.10.2. The details of the algorithm are included in the workflow of the particle filter (Step 2 and 4) in the following subsection.

5.7.2 Improved Particle Filter

We first give the definition of particles used in the particle filter.

*Particle:* In our algorithm, a particle is defined as a possible walking state of the user. For the $n$th step made by the user, we denote a particle by the vector

$$p^m_n = [x^m_n, y^m_n, \theta^m_n], \quad m = 1, \ldots, M, \quad n = 0, 1, \ldots, \quad (5.9)$$

where $(x^m_n, y^m_n)$ denotes the 2D location of the user on the map, $\theta^m_n$ is the absolute walking direction, $m$ is the index of particles, and $M$ denotes the total number of particles, which is a constant for a given map. $n = 0$ means that the user has not started to move and the particles are at their initial positions. Both $(x^m_n, y^m_n)$ and $\theta^m_n$ are defined in an orthogonal coordinate system, where $\theta^m_n = 0$ is the positive direction of the x-axis and $\theta^m_n = 90^\circ$ is the positive direction of the y-axis.

Since the user’s location is initially unknown, we distribute the particles randomly on the map. Based on the estimated direction change and stride length, we keep updating the particles for the user’s each step. Then, the location changes of the $M$ particles show $M$ possible walking trajectories of the user. For the invalid trajectories blocked by the barriers on the map, we drop the corresponding particles and regenerate new ones on the valid trajectories. All the $M$ trajectories will converge to a single path, and thus we can determine the correct location of the user.

The algorithm of the improved particle filter is shown in Algorithm 5.1. It has the following steps. Note that Step 1-5 in the algorithm do *not* refer to the physical footsteps made by the user.

(Step 1) *Initialization:* The algorithm takes the floor map $MAP$ as an input, which includes the locations of hallways as well as barriers such as walls. As we do not know the user’s initial location, we uniformly select $U$ locations in the regions where the user can stay from the map with a constant density (line 1). For each location, we select $V$ initial walking directions, i.e., $0, 360^\circ/V, \ldots, 360^\circ(V - 1)/V$, and for each direction, we generate $W$ particles (line 2-7). Thus, the total number of particles is $M = UVW$. Larger $U$, $V$ and $W$ would provide higher localization accuracy, but also increase the computation time. Typically,
Figure 5.8: Example showing how the particle filter works based on the data in Figure 5.6. (a) shows the real walking path of the user. (b)-(e) show how the particles converge. (e)-(f) show how the location of the user is tracked after the convergence. (g) shows the traceback of the walking path.

$V$ should be no less than 4. During our implementation, we set the density of the initial locations as $25/m^2$, and $V=12$, $W=2$. One example of the initial particles are shown in Figure 5.8(b).

Moreover, based on $MAP$, we can get the direction set $D$ of the hallways on the map (line 9). Typically, if all the hallways are latticed, the direction set is $D = \{0^\circ, 360^\circ, 90^\circ, 180^\circ, 270^\circ\}$. Note that $D$ can be modified according to the map. For example, if $MAP$ contains the hallways in Figure 5.4(d), $45^\circ$ and $225^\circ$ are also added to $D$.

After the initialization, the execution of Step 2-5 is triggered by each step made by the user.

**Step 2** Straight trajectory detection: For the $n$th step made by the user ($n \geq 1$), the filter receives $\Delta \theta_n$ and $d_n$. Our observation shows that when the user makes a turn, it is unlikely that the degree change made by the key step is smaller than $40^\circ$. Therefore, the estimated result $|\Delta \theta_n| < 20^\circ$ almost means that the user is walking along a straight path. We set $\Delta \theta_n = 0^\circ$ in this case (line 11). Moreover, for the $n$th step, if $\Delta \theta_{n-3}, ..., \Delta \theta_n$ are all $0^\circ$, we label this step as “straight” (line 12), which implies that the user is walking along a straight hallway.

**Step 3** Particles update: We update all the $M$ particles for each new footstep. For the $m$th particle, we first set $m' = m$ and update particle $p_{n-1}^{m'}$ to $p_n^m$ following line 14-20, where $\delta_n$ and $\lambda_n$ are Gaussian random variables with zero means. In our prototype system, the standard deviation of $\delta_n$ is 0.1 m, while that of $\lambda_n$ is $10^\circ$. Note that the cumulative error
ALGORITHM 5.1: Procedure of Walking Path Recognizer

```
Input: Floor map MAP; Initial location number U; Initial direction number V; Particle redundancy factor W; Angle change sequence \{Δθ_n\mid n = 1, 2, \ldots\}; Stride length sequence \{d_n\mid n = 1, 2, \ldots\};
Output: Location sequence \{(x_n, y_n)\mid n = 1, 2, \ldots\} and “converged”/“unconverged” labels;
// (Step 1) Initialization
1 Uniformly select U initial locations in the walking region of MAP;
2 m ← 0;
3 foreach selected initial location (x, y) do
4  foreach v = 1, ..., V do
5     m ← m + 1;
6     p_n^m \triangleq [x_n^m, y_n^m, θ_n^m] ← [x, y, 360^\circ (v - 1)/V];
7     8 M ← UVW;
9 Generate direction set D based on MAP;
// (Step 2-5) Run the following loop once for each footstep made by the user
10 foreach footstep n = 1, 2, ..., do
11     // (Step 2) Straight trajectory detection
12     if |Δθ_n| < 20^\circ then Δθ_n ← 0^\circ ;
13     if \sum_{j=0}^{n-1} |Δθ_{n-j}| = 0^\circ then Label step n as “straight” ;
// (Step 3) Particles update
14     foreach m = 1, ..., M do
15         m' ← m;
16         while true do
17             // Update particle p_n^{m'} to p_n^m
18             Generate random variables δ_n ∼ N(0, 0.1^2), λ_n ∼ N(0, (10^\circ)^2);
19             x_n^m ← x_n^{m'} + (d_n + δ_n) \cdot \cos(θ_n^{m'} + Δθ_n + λ_n);
20             y_n^m ← y_n^{m'} + (d_n + δ_n) \cdot \sin(θ_n^{m'} + Δθ_n + λ_n);
21             θ_n^m ← (θ_n^{m'} + Δθ_n) \mod 360^\circ ;
22             p_n^m \triangleq [x_n^m, y_n^m, θ_n^m];
23             if line segment between (x_n^{m'}, y_n^{m'}) and (x_n^m, y_n^m) intersects with barriers in MAP
24                 then
25                     Randomly select m' \in \{1, ..., M\};
26                 else Break ;
// (Step 4) Direction correction
27     if step n is labeled as “straight” then θ_n^m ← \arg \min_{ω \in D} \{180^\circ - \mid |θ_n^m - ω| - 180^\circ \};
// (Step 5) Convergence check
28     \hat{x}_n ← \frac{1}{M} \sum_{m=1}^{M} x_n^m;
29     \hat{y}_n ← \frac{1}{M} \sum_{m=1}^{M} y_n^m;
30     σ ← \sqrt{\frac{1}{M} \sum_{m=1}^{M} ((x_n^m - \hat{x}_n)^2 + (y_n^m - \hat{y}_n)^2)};
31     if σ < σ_T then Label (\hat{x}_n, \hat{y}_n) as “converged” ;
32     else Label (\hat{x}_n, \hat{y}_n) as “unconverged” ;
33 return \{(x_n, y_n)\mid n = 1, 2, \ldots\} and “converged”/“unconverged” labels;
```

caused by the calculation in line 19 will be canceled by the direction correction. Then, we check whether the trajectory from \(p_n^{m'}\) to \(p_n^m\) intersects any barriers (line 21). If so, \(p_n^m\) is
dropped. A new \( m' \) is randomly selected from \( \{1, \ldots, M\} \) (line 22), based on which we pick a new particle \( p^n_{m'-1} \) and update it to \( p^n_m \) following line 16-20. This process (line 15-23) is repeated until a valid \( p^n_m \) is generated.

**Step 4** Direction correction: Recall that we have gotten the direction set \( D \) in Step 1. If the \( n \)th step is labeled as “straight”, we change the value of \( \theta^n_m \) of each particle to the closest value in \( D \) (line 24). Take \( D = \{0^{\circ}(360^{\circ}), 90^{\circ}, 180^{\circ}, 270^{\circ}\} \) as an example. If the 10th step is “straight” and \( \theta^{100}_{10} = 169^{\circ} \), we set \( \theta^{100}_{10} = 180^{\circ} \); if \( \theta^{101}_{10} = 350^{\circ} \), we set \( \theta^{101}_{10} = 0^{\circ} \). Note that both Step 2 and 4 form the direction correction algorithm.

**Step 5** Convergence check: When Step 2-4 are repeated, eventually the particles converge to have similar locations. In line 25-26, we calculate the location center \( (\tilde{x}_n, \tilde{y}_n) \) of the particles, which is the estimated user location. Moreover, we also calculate the standard deviation of the location differences to the center (line 27). If the standard deviation is lower than the accepted threshold \( \sigma_T \), the particles have converged and we label the estimated location as “converged”, otherwise the location is labeled as “unconverged” and would have a low accuracy (line 28-29). In our implementation, we set \( \sigma_T = 2 \) m. For the \( n \)th footstep, \( (\tilde{x}_n, \tilde{y}_n) \) and its label are the final outputs of our system. Figure 5.8(b)-(e) show how the particles keep converging while the user is walking. Note that in Figure 5.8(c), the particles in the horizontal routes cannot be eliminated because the absolute walking direction is unknown yet.

After the particles have converged, the system can still track the location change of the user. Figure 5.8(e)-(f) show how ShoesLoc continues locating the user after convergence. If the walking path of the user before convergence is needed, we can trace back the trajectories of the particles, as shown in Figure 5.8(g).

If the particle filters used in the previous works are directly applied, a large amount of particles on valid paths will be dropped in Step 3 due to the cumulative direction error, which leads to long computation time. The direction correction in Step 4 significantly reduces the cumulative direction error and prevents the valid particles from being dropped, and thus reduces the computation time.

5.8 EVALUATION CONSIDERING CONTINUOUS WALKING

In this section, we first evaluate the accuracy of the direction change estimator and the stride length estimator. Then, we evaluate the performance of ShoesLoc in the case of continuous walking by conducting experiments in two different indoor spaces. The case of frequent walking state changes such as making frequent turns and pauses will be considered in Section 5.9. The training and test data are collected from 5 male and 2 female volunteers.
with different ages (23-52), heights (1.58-1.83 m) and weights (53-98 kg).

For the sensors in the shoes, we use the product of ReTiSense Stridalyzer [22] to collect the force data. Each insole contains five force sensors, and continuously transmits the data to our host device via a Bluetooth connection. The other components are implemented on a laptop host (MacBook Pro with 2.7 GHz Intel Core i7 and 16 GB RAM) using Matlab script. However, as Section 5.9 will show, the computation can be done on mobile devices and it does not impact the practicality of ShoesLoc.

5.8.1 Performance of Direction Change Estimator

In this subsection, we evaluate the performance of the direction change estimator. We collect training data from each of the 7 volunteers following the format in Section 5.5.3. For each turning direction in Figure 5.5, each volunteer repeats walking 12 times. For 6 times the left foot makes the key steps, while the right foot makes the key steps for the other 6 times. We manually label the key steps, which are used to train the direction change estimator.

During the test, each of the volunteers is asked to walk in free style and make 40 turns. We record the walking of the volunteers with cameras, and measure the direction change made by each key step with a protractor. Moreover, the steps are manually divided into two classes, i.e., “straight” and “turn”. The probability distributions of the errors are shown in Figure 5.9. Note that the sign of the errors follows Figure 5.5, i.e., a positive error means that the estimated direction change shifts leftward compared with the ground truth, while a negative error means the estimated value shifts rightward. The average signed errors for the two classes are $-2.30^\circ$ and $-1.95^\circ$, while the average absolute errors for the two classes are 16.79$^\circ$ and 16.04$^\circ$, respectively. Moreover, the results also show that, when the user keeps moving, the cumulative error of the walking direction is not negligible, which confirms the necessity of applying the direction correction to cancel the cumulative error by making full use of the map information.

We further analyze the performance when the sensor number changes. We restrict the sensor number in each insole to be 1-5 respectively, and select the sensors that can achieve the best performance. The metric is the average error of both the “straight” and “turn” steps, and the two classes have the equal size. The results are shown in Figure 5.10. For instance, if only 3 sensors can be deployed in each shoe, they should be at the inside metatarsals, the outside metatarsals and the outside midfoot to achieve the best performance. The average absolute error is 21.37$^\circ$. It is shown that, by deploying only 4 sensors we can achieve a similar performance to the case of 5 sensors. Thus, for the purpose of walking direction change estimation, it is possible to relax the requirement of 5 sensors if not all of them
can be deployed. Moreover, Figure 5.10 implies that the sensor at the inside metatarsals makes the biggest contribution, which accords with our intuition that there are distinct force changes at the inside metatarsals when people are turning.

5.8.2 Performance of Stride Length Estimator

To collect the training data for the stride length estimator, the volunteers are required to walk along a straight path of 40 m. It is hard to control the stride length during walking. Thus, we require each volunteer to repeat this process 5 times, and each time walk at a different average speed, which ranges in 1-3 m/s at intervals of 0.5 m/s. We get the average stride length by dividing the distance by the number of steps, and the labeled results are used as the training data.
During the test, each volunteer walks along the same path 5 times. Each time the speed is not tightly controlled, and the average speed varies around 1-3 m/s. We get the ground truth of each footstep’s stride length by cameras and tape measures. The probability distribution of the errors are shown in Figure 5.11. The average signed error is $-1.22$ cm, and the average absolute error is $6.25$ cm. Since the average stride length is $72.60$ cm, the error is $8.61\%$, which is sufficient for the purpose of walking path tracking because the cumulative error can be reduced by the particle filter.

We also evaluate the performance of the stride length estimator when the number of available sensors are changed. For a given number of sensors, we select the sensors that can achieve the lowest error. The results are given in Figure 5.12. It is shown that, when only 3 sensors can be deployed, the average absolute error is still no larger than $6.41$ cm. Moreover, it can be seen that the sensor at the heel is the most important one for the stride length estimation.

5.8.3 Impact of Carrying Bags

As mentioned before, carrying handbags or backpacks changes the force distribution on insoles. In ShoesLoc we design the weight normalization method to reduce this impact. To evaluate the robustness of ShoesLoc in the face of changing body barycenter, we require each volunteer to carry a bag of 2 kg or 5 kg. Four of the volunteers carry the bag with two shoulders and the others carry the bag by hands. The experiments in Section 5.8.1 and 5.8.2 are repeated, and the cumulative distribution functions (CDFs) of the absolute errors are shown in Figure 5.13 and 5.14.

Figure 5.13 and 5.14 show that, by applying weight normalization, the error of the direction change and stride length estimation can be reduced. The average angle errors without the normalization are $28.04^\circ$ and $29.55^\circ$, while after the normalization the errors are $17.39^\circ$ and
19.07°, respectively. The average errors of stride length estimation are also improved from 7.12 cm and 9.38 cm to 6.37 cm and 6.32 cm, respectively. Therefore, with the weight normalization method, we can significantly reduce the impact of carrying bags.

5.8.4 Performance of ShoesLoc

To evaluate the overall performance of ShoesLoc, we require the volunteers to walk in two indoor spaces, as shown in Figure 5.15. During the test, each of the 7 volunteers walks randomly in each test region for 30 minutes. The possible walking paths are shown in Figure 5.15. During the walking, the volunteers are asked to carry their backpacks and handbags (1-5 kg). The way that they carry the bags is not controlled, e.g., the backpack can be carried by either one or two shoulders. During each test, the volunteers are allowed to take breaks in place. The break time is not included in the 30 minutes. The stoppages can be easily detected since the force signals stop showing periodic changes. The tracking process is resumed when a break ends. We use cameras to record the real walking trajectories, and use time stamps to match them with the localization results. The trained models in Section 5.8.1 and 5.8.2 are used in this experiment.

Figure 5.15: The floor maps of the test regions. The dashed lines show the path used in Section 5.8.5.

Before ShoesLoc can track the location of the user, the particles must converge (Step 5 of Algorithm 5.1). It is meaningless to evaluate the tracking accuracy before the convergence, because the user’s location has not yet been uniquely determined. Moreover, it is hard to control the time length of the convergence phase because it depends on the complexity of the walking path and the floor structure, as well as the walking speed of the user. Therefore, we divide the working process before and after the convergence into two phases, i.e., the convergence phase and the tracking phase, and mainly focus on the tracking phase. During each walk, once the particles have converged and entered the tracking phase, we continue
recording 500 steps made by the volunteer. After that we restart ShoesLoc and the convergence phase begins again. Thus, each walk of 30 minutes is divided into several tests, and each test has two separate phases. During the experiment in Figure 5.15(a), 40 tests are generated, i.e., each volunteer achieves 5.7 convergences on average. For the test in Figure 5.15(b), 38 tests are generated.

![Location Error Graph](image)

Figure 5.16: Location error during two walks.

As examples, the location errors during two walks are shown in Figure 5.16. We also evaluate the performance of the particle filter used in [6] where the direction correction is not applied. Figure 5.16(a) shows that, ShoesLoc keeps converging while the person is walking, and the convergence phase ends around the 110th step. After that, the system enters the tracking phase and still maintains high accuracy. If tracing back is applied, the accuracy during the convergence phase can be significantly improved. On the other hand, if direction correction is not applied, the particles once tend to converge before the 95th step. However, after that they keep diffusing during the whole walk, and thus the user cannot be localized. The reason is that, without the direction correction algorithm, it is easy for the particles on valid routes to go cross the walls because of the cumulative direction errors. Therefore, the direction correction algorithm significantly improves the performance of the particle filter, which is also confirmed by the other example shown in Figure 5.16(b).

The average step number that a volunteer takes before convergence is 69 in Figure 5.15(a) and 85 in Figure 5.15(b). The main reason why more steps are needed to converge in Figure 5.15(b) is that, its floor map contains more symmetric corridors, which makes the convergence a little harder.

We also evaluate the location error during the tracking phase of all the tests. The results are shown in Figure 5.17. The average errors for the two floor maps are 0.88 m and 1.30 m, respectively. The case without direction correction is excluded because the particles cannot converge. Moreover, if tracing back is applied, the average errors for the whole walking paths
Figure 5.17: CDF of location error in two floor maps.

Figure 5.18: CDF of location error when the walking speed changes.

are 0.86 m and 1.05 m, respectively. Thus, the average location error of ShoesLoc is lower than that of the Wi-Fi RSS-based methods (2-3 m) [120], and is comparable with that of the inertial sensor-enhancing approaches (1.5-2 m) [50]. Meanwhile, ShoesLoc requires no site survey and has better robustness to magnetic interference.

To evaluate the realtime performance of ShoesLoc, we also record the computation time. When the particle number is 60,000, the average computation time for each step is 0.35s. If direction correction is not applied, the average computation time is 0.73s. This is because the high dropping rate of the particles on valid paths increases the time cost of generating new valid particles. In some extreme cases, more than 90% of the particles could be dropped at a step, which leads to a long computation time. Given that the average time length of people’s each step is around 0.70-0.81s [121], ShoesLoc is able to provide realtime localization and tracking. A shorter computation time can be achieved, if the particle number is further reduced. In Section 5.9, we will show that ShoesLoc is also efficient when working on a mobile device.

5.8.5 Impact of Walking Speed

In Section 5.8.4, we have no control on the walking speed of users. In this subsection, we evaluate the impact of the walking speed on the performance of ShoesLoc. Since it is hard to measure and control the actual instantaneous speed of users, we use the average speed instead. The average speed used for our tests ranges from 1 m/s to 3 m/s at intervals of 0.5 m/s. As shown in Figure 5.15(a), each volunteer is required to go along a given path 5 times. For each time, a different average speed is followed. During each walking, an experimenter walks after the volunteer and uses a timer to help the volunteer reach the destination at the given average speed. We then calculate the location error in the tracking phase. The
results are shown in Figure 5.18. When the average speed changes from 1 m/s to 3 m/s, the average errors of all the volunteers’ walks are 1.09 m, 0.83 m, 0.81 m, 0.98 m and 0.85 m, respectively. Therefore, the change of the walking speed does not seriously impact the performance of ShoesLoc. This is because the features used for direction change estimation are not highly impacted by the walking speed, and the design of the stride length estimator naturally handles the speed changes well.

5.9 EVALUATION CONSIDERING FREQUENT TURNS, PAUSES AND SHOE TYPE CHANGES

In Section 5.8, some walking paths selected by the volunteers go through long straight hallways. In our daily life, it is also likely that users’ walking paths include more short straight lines with more turns, e.g., walking from a desk to a printer or to a meeting room. Moreover, users would also make short pauses while walking, e.g., looking at a poster, or greeting someone. On the other hand, it is possible that the type of the shoes that the user wears is different during the training and the daily usage. In this section, we evaluate the impacts of frequent turns, pauses, and shoe type changes.

In the indoor space shown in Figure 5.19, we conduct a similar experiment as in Section 5.8.4. We recruit another group of volunteers (6 males and 5 females) with different ages (21-53), heights (1.62-1.85 m) and weights (57-104 kg). The same training format is followed, and all the volunteers wear hard bottom shoes during the training. To test ShoesLoc’s performance on mobile devices, we implement our components on an iPad (2017) with iOS 11.2.2 operating system. To adapt to the computing power of the device, we adjust the density of initial particle locations, and the particle number is 45,000.

We design four test cases for each volunteer. In Case 1, the volunteer walks randomly for 10 minutes. The only rule is that a left or right turn must be made at each intersection. To make the comparison among different test cases meaningful, we record the walking path, and require the volunteer to walk along the same path in the following test cases. In Case 2, the volunteer is required to make a 2-second pause every 10 seconds (controlled by an interval timer), and continue to walk along the original direction after the pause. In both Case 1 and Case 2, all the volunteers wear hard bottom shoes. In Case 3, the volunteer follows the same path, but wears sneakers with air cushions instead and makes no stop. In Case 4, the volunteer wears sneakers and repeats Case 2.

The test results without traceback are shown in Figure 5.20. The average errors of the four test cases are 1.03 m, 0.99 m, 1.15 m and 1.27 m, respectively. The average step numbers before convergence are 90, 95, 92 and 97, respectively.
The comparison between Case 1 and the experiment in Section 5.8.4 shows that, frequently turning does not impact the average error. On the other hand, while the floor plan in Figure 5.19 has a similar scale to those in Figure 5.15, the step number before convergence increases. This is because the floor plan contains multiple corridors in similar shapes, and more short straight lines make it harder to uniquely determine the user’s location. However, ShoesLoc can still converge and achieve a good accuracy.

The comparison between Case 1 and Case 2 as well as Case 3 and Case 4 shows that there is no obvious impact caused by frequent pauses. Making a stop and restarting walking would slightly increase the direction and stride length estimation errors for the footsteps before and after the pause. However, the particle filter adds random variables ($\delta_n$ and $\lambda_n$) to the estimated values, which reduces the impact of the estimation errors for these footsteps. The EMA smoothing in the stride length estimator further mitigates this impact.

The results of Case 3 and Case 4 show a slight decrease in accuracy. This is because air cushions attenuate the maximum impact force on insoles when the shoes are touching the ground [122], which also changes the force distribution on the sensors. However, the weight normalization method normalizes the measured data and keep the distribution of the force peak amplitudes consistent with that in the training data. Therefore, ShoesLoc still maintains good performance even if the shoe type in the tests is different from that in the training.

During the tests, the average computation time for each step is 0.43s, which is still efficient for realtime tracking. We also analyze the device’s energy log with the Instruments application of Xcode, and use the energy usage level as the metric [123]. The level ranges from 0 to 20, which indicates how much energy an application is using at the given time. The sampling rate is 1 Hz, and a lower average level means that the application consumes
fewer energy. The average energy usage level of ShoesLoc during the tests is 10/20. As a benchmark, our test shows that the average level of Apple Maps is 8/20. Therefore, the energy consumption of ShoesLoc is acceptable since users typically do not keep using the tracking service all the time.

5.10 DISCUSSION

5.10.1 Accelerating Convergence

Before ShoesLoc can locate the user, the user needs to keep walking until the particles converge. The speed of the convergence depends on the user’s walking speed, the complexity of the walking path, as well as the structure of the barriers on the map. To speed up the convergence, similar to the idea in [124], we can further make use of the special facilities on the map to reduce the possible region of the user’s initial location. For instance, the user must go through entrances to enter buildings. Also, by using the force sensors in the shoes, it is easy to detect the actions of going up/down stairs or using elevators. By recognizing the start point of the user’s walking path, the convergence process can be significantly sped up. In our work, to guarantee the practicality of ShoesLoc, we only focus on the general case that the initial location is unknown. The walking path tracking with known initial locations is a special case of the challenge solved by ShoesLoc.

5.10.2 Hallways in Irregular Shapes

Although most hallways are straight, some of them would still have complex shapes. For instance, in some shopping malls the hallways are curving or in the shape of arcs. However, usually the radius of the arcs are large enough that they can be considered as straight hallways by ShoesLoc. Specifically, if the direction difference between the start and the end of a curving hallway is smaller than 30°, we treat it as a straight hallway. If the direction changes a lot, we can crop the hallway into shorter hallways and treat each of them as a straight hallway. For more complex irregular hallways, we can simply consider them as non-straight hallways, which can still be handled by ShoesLoc.

5.10.3 Security Issue

As what we have mentioned, there have been commercial force sensors or force mapping systems for shoes [22, 21, 58, 59]. Some of these products allow users to upload their data
to the cloud for analysis or statistics. The design of ShoesLoc implies that, if the data are leaked, it is possible for unauthorized third-parties to recognize the daily walking paths of the users. Although the training data would be unavailable, it is still possible to learn the walking pattern of the users by recognizing some common actions. For example, it is easy to detect the action of going up/down stairs, during which a user needs to make several left or right turns at each floor. Then the attacker can crop the training data for the direction change estimation. We will continue to explore the privacy leakage through the force data from smart shoes in Chapter 6.

5.10.4 General Model for Walking Path Tracking

Currently, because of the diversity of walking habits, ShoesLoc requires each user to provide training data before use. Moreover, our weight normalization method assumes that the weight of the user is consistent during the training data collection. If we train a general model by simply mixing the training data from different people, the signal measured during the user’s walking will be incorrectly normalized to fit the data patterns of the heaviest or lightest training data providers. Some other factors such as the measurement differences caused by the device diversity would also lead to a low accuracy of the general model. However, there are still potential solutions. For instance, based on a large amount of data, it is possible to train the general model with deep neural networks. We leave it to our future work to design a system that works without requiring training data from each user.

5.10.5 Foot Force as A Biometric Identifier

The diversity of foot force patterns implies the possibility of using the foot force information as a biometric identifier. It has been shown that, due to the various walking habits, gaits can be used for person identification [15, 71, 72]. The patterns of foot force are clearly associated with the user’s gait. It is even possible that the information hidden in foot force properly includes the gait information, and can be used to restore the user’s walking habit. Thus, investigating the possibility of using force sensors for biometrics is promising.

5.10.6 Obtaining and Preprocessing Floor Maps

In our work, we assume that the indoor floor maps are available to users’ devices. In practice, there are multiple possible sources of floor maps. For instance, Google Maps [125] provides the floor plans of a large number of buildings, which can be fetched by users’ devices.
Moreover, since the rooms and hallways on the maps are labeled with different colors, the walking regions and barriers can be programmatically recognized with image processing techniques. For the buildings where digital floor plans are not available yet, there have been approaches that reconstruct indoor maps based on crowdsourcing techniques [126, 127]. In our implementation, after we have obtained the floor plan image, we preprocess the floor map by dividing it into a matrix of small grids with the resolution of 0.2 m × 0.2 m. The grids occupied by walls and barriers are programmatically recognized and labeled in the matrix according to their colors in the image. In this way, we can locate possible walking regions and detect if a walking trajectory goes through walls or barriers.

5.11 CONCLUSION

In this chapter, we present ShoesLoc, a novel indoor walking path tracking method that is based on in-shoe force sensors. We design algorithms to estimate the walking direction change and the stride length based on the force signals, and apply the particle filter to determine the walking path. We further propose the direction correction algorithm and the weight normalization method to improve the particle filter’s performance and handle the impact of handbags and backpacks. The experimental results show that ShoesLoc achieves the accuracy of 0.9-1.3 m, which is sufficient for walking path tracking.
CHAPTER 6: SHOESHACKER: INDOOR CORRIDOR MAP AND USER LOCATION LEAKAGE THROUGH FORCE SENSORS IN SMART SHOES

6.1 INTRODUCTION

The recent popularization of wearable devices has been impacting people’s lives. For instance, many smartwatches can measure heart rate [4], and smart shoes can count footsteps or detect users’ fatigue levels [22, 21, 58, 23]. However, besides the benefits brought by the new devices, people should also be aware of the danger caused by measurement data leaks. It has been shown that if the motion data measured by smartwatches were leaked, an attacker could recognize the content that victims input via keyboards [24], or the passwords of turntable password locks [10]. However, it has not yet been recognized that the data from smart shoes also leak information. Many smart shoes are equipped with force or piezoelectric sensors. For instance, ReTiSense [22] and Zhor-Tech [23] provide foot force mapping systems that help analyze users’ gaits. It has not been explored if the force mapping systems are vulnerable to side-channel attacks. Moreover, many products allow users to upload their data to cloud servers for statistical analysis, which increases the attack surface. In this chapter, we show that if an attacker has access to the foot force data, it is possible to reconstruct the corridor map of the building that the victim walks in, and even locate the building and the victim on a global map.

Some previous works have shown the capability of smart shoes for indoor walking path tracking [63, 61, 62], among which ShoesLoc [60] is the first system that utilizes in-shoe force sensors to track users. However, ShoesLoc requires floor plans and users’ training data as additional inputs, and thus cannot be used for attack, because the attacker usually does not know the building and cannot directly collect the training data from the victim. In our work, we assume that the attacker has got the force data by hacking the user’s device or cloud servers. However, no training data or prior knowledge about the floor plan is needed.

In this chapter, we present ShoesHacker, an attack scheme that reconstructs corridor maps and locates victims based on the smart shoes equipped with force sensors. Our method is based on the basic assumption that the corridors are latticed, which means that all the turning angles in corridors are 90°. We consider this assumption reasonable because the latticed corridors are common in office buildings, which are the main targets of attackers.

The main idea of our attack method is that, during the victim’s daily use of smart shoes, we extract training data to train support vector machine (SVM) models, based on which we can estimate the victim’s walking trajectories. By merging the trajectories, we can reconstruct the corridor map of the floor, and also locate the victim on the floor. By comparing the
corridor map with the floor plans provided by some global maps such as Google Maps [125], the attacker could recognize the building, and thus locate the victim on the global maps.

The main challenge is that, due to people's diverse walking habits, we need to train an individual machine learning model for each victim to recognize the turning steps during walking. However, in attack scenarios, labeled training data are unavailable. We show that it is possible to extract the training data by recognizing the action of going up or down stairs. For instance, in a typical office building, when a victim walks from the first floor to the third floor, the walking path goes through three stair landings, during which the victim makes three U-turns. We propose the stair landing detection algorithm to detect the action of walking on stairs, and recognize the turning steps in the U-turns, which can be used for training. Thus, the training data can be collected during the victim's daily life, which makes the supervised learning applicable.

Another challenge is that, the walking paths of the victim are scattered, i.e., the victim usually does not traverse the whole floor at once. Thus, it is hard to estimate the whole corridor map during a single walk. We propose that we can estimate the trajectory of each walk, and keep merging the trajectories. As the victim will eventually visit all the corridors during the daily use, ShoesHacker can reconstruct the whole corridor map. We design methods to detect the continuous walking in corridors, and estimate the trajectory of each walk. We propose the path merging algorithm to merge the trajectories, even if the estimated trajectories contain errors and their absolute directions are unknown.

The main contributions of our work are:

- To the best of our knowledge, we are the first to explore the possibility of reconstructing corridor maps based on in-shoe force sensors, and reveal the danger of location privacy leakage through smart shoes.

- To handle the lack of the training data from victims, we propose to extract the training data during the victim’s daily usage. We design the stair landing detection algorithm, and recognize the turning steps made on stair landings, which can be used to train our machine learning models.

- To reconstruct corridor maps, we propose the path merging algorithm, which can merge the estimated walking trajectories even if the absolute walking directions are unknown.

- We implement a prototype system, and design a metric to evaluate its performance. Experiments confirm that the privacy leakage through the force data from smart shoes is highly probable under certain conditions.
ShoesHacker is still a proof-of-concept system based on our basic assumption. However, it sheds light on a new threat to people’s location privacy caused by smart shoes.

We first introduce related works in Section 6.2, and present a system overview in Section 6.3. Then we successively introduce the methods used by ShoesHacker in Section 6.4 - 6.9. In Section 6.10, we evaluate the performance of ShoesHacker with experiments. We have further discussion in Section 6.11, compare ShoesLoc and ShoesHacker in Section 6.12, and conclude this chapter in Section 6.13.

6.2 RELATED WORK

Mobile device-based side-channel attacks: While smart mobile devices are evolving rapidly, exploring the side-channel privacy leakage in mobile devices has been a hot research topic for years. For instance, the mobile devices can be fingerprinted based on the hardware imperfections of accelerometers [128], speakers [129], or cameras [130]. By using the inertial sensors in a smartwatch, an attacker can track the victim’s hand motion, and recognize the content that the victim inputs via keyboards [24] and keypads [131], or guess the password of a turntable password lock [10]. By using the data from a cellphone’s permissionless sensors such as accelerometer and gyroscope, the attacker can reconstruct the victim’s secret PIN for unlocking the cellphone [11, 132].

However, while researchers have been exploring the privacy leakage via smartphones and smartwatches, the possible information leakage via smart shoes has not been discovered. We are the first to reveal the threat of localization attacks on smart shoes.

User localization attacks: With the fast popularization of location-based services, people have been aware of the harmfulness of location leakages. The exposed location traces can be used to infer the victim’s activities, habits, or even identity [133, 134]. While researchers and the industry have designed and adopted many location protection approaches to protect users’ geographical locations [135, 136], users’ location traces can still be inferred via multiple channels such as social networks [137, 138], call records [139], and people-nearby services [140].

It has also been shown that cellphones are vulnerable to side-channel localization attacks. For instance, Powerspy [141] shows that it is possible to locate a user by reading the phone’s power consumption, because the power usage reflects the received signal strength (RSS) from cellular base stations, which helps infer the user location. ACComplice [142] reveals that the signal measured by the accelerometer of a cellphone can be used to infer the driving trajectory of a vehicle. In [70], it is shown that the driving instructions played by navigation applications change the audio on/off status of the phone, which provides a side channel for
victim tracking.

However, since commercial smart shoes are still at the primary stage, the possible location privacy leakage from them has not been well-investigated. In our work, we explore the possibility of locating people based on the force data leaked from smart shoes.

**Indoor localization:** A large number of works have been done in the field of indoor localization [36, 107, 34, 38]. While most methods are designed for legitimate use cases, potentially their techniques can also be used for localization attacks. For instance, the Wi-Fi RSS can help locate users [44, 45, 46], but can also be used to locate victims once the data are leaked [143]. ShoesLoc [60] demonstrates that, if the floor plan is available, the user can be located based on in-shoe force sensors. However, ShoesLoc can hardly be applied for attacks, because the floor plan and training data are usually unavailable to the attacker. In our work, we require no pre-known floor plan of the target building, and design algorithms to extract training data when the victim is walking in stairwells. Some other works show that the inertial measurement units (IMUs) embedded in shoes can also locate users [63, 61, 62]. However, these works estimate the path of a single walk, which can hardly determine the user location when the building is unknown. In our work, we design an algorithm to merge multiple walking paths and reconstruct the corridor map, based on which the building can be recognized. If the smart shoes are equipped with IMUs instead of force sensors, the attack can still be conducted by combining the previous IMU-based tracking methods and our work. Thus, ShoesHacker can serve as a framework that works for both force sensors and IMUs.

**Floor plan reconstruction:** While most of the floor plan reconstruction methods are based on videos or images [126, 127], there are also works that utilize acoustic signals [144], laser [145], or inertial sensors plus landmarks [146, 147, 148]. For instance, CrowdInside [146] utilizes the cellphone motion sensors to estimate people’s walking traces, and thus reconstruct the floor plan through crowd-sourcing. However, this approach requires the existence of points of interest, such as elevators, to reset the errors in the estimated walking traces. By contrast, except of the walking on stairs, ShoesHacker does not require any point of interest on walking paths, and can handle the walking trajectories with larger errors. In the work of MapGENIE [149], the authors show that the data collected from foot-mounted IMUs can also be used to reconstruct the floor plan. As mentioned before, the path merging algorithm of ShoesHacker also works for the IMU based approaches, and our algorithm can handle an even more general case that no digital compass is used and the absolute walking direction is unknown.
6.3 SHOESHACKER OVERVIEW

In this section, we first introduce the threat model and the basic assumptions, and then give an overview of the sensor deployment and the system architecture of our attack method.

6.3.1 Threat Model and Basic Assumptions

The purpose of the attack is to reconstruct the corridor map of the building that the victim daily walks in, which makes it possible to further locate the building and the victim. The corridors are latticed, i.e., the turning angles of intersections are $90^\circ$. We assume that the attacker knows the rough region that the building locates in, e.g., a university campus or an industrial park. The floor maps of the buildings in the region are available to the attacker, which is practical since many global maps such as Google Maps [125] provide indoor floor maps. We also assume that the floor structures of these buildings are diverse, because we can hardly distinguish two buildings with the same floor structure.

We assume that the smart shoes are equipped with a sufficient number of force sensors, which will be shown in Section 6.3.2. The force data have been leaked from the mobile device that is paired with the shoes, from the cloud, or during the data transmission. If the data are leaked in real time, we can further locate the victim in real time.

The attack is based on the daily use of smart shoes. Before the corridor map can be estimated, the victim should have traversed the floor of the target building several times, and have used staircases (not necessarily in the target building). The attack period can be days, weeks or months, which is acceptable since the typical attack scenario is locating the office building that the victim works in.

6.3.2 Sensor Deployment

Currently, most of the smart shoe-based force mapping systems deploy plenty of force sensors in insoles. For instance, the sensor deployment of the ReTiSense Stridalyzer insoles [22] is shown in Figure 6.1. For each insole, eight force sensing resistors are deployed at the positions of hallux, fourth toe, inner metatarsals, outer metatarsals, outer midfoot, arch, inner heel, and outer heel.

In this chapter, we use the off-the-shelf ReTiSense Stridalyzer insoles to show how the location information would leak from the force data. For each insole, we assign indices $i = 1, ..., 8$ to the eight sensors. While the victim is walking, each sensor keeps outputting a sequence of force values, and the sampling rate is 15 Hz. As mentioned in Section 6.1, we
assume that the attacker has access to these raw data via the data leakage. We treat the 16 force data sequences as 16 discrete time series, denoted by \{u^l_i(t), u^r_i(t) | i = 1, \ldots, 8, \ t \geq 0\}, where \(u^l_i(t)\) is the data sequence received from sensor \(i\) in the left insole, and \(u^r_i(t)\) is from sensor \(i\) in the right insole. Although \(u^l_i(t)\) and \(u^r_i(t)\) are discrete time series, we write them as continuous time series for convenience, which does not impact our following analysis. In Section 6.10.2, we show that it is possible to conduct the attack with less sensors.

6.3.3 System Architecture

As shown in Figure 6.2, our system mainly consists of the following components.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{system_architecture.png}
\caption{System architecture of ShoesHacker.}
\end{figure}

\textit{Step detector:} It receives the raw signals \{\(u^l_i(t), u^r_i(t)\}\} from the 16 force sensors, and extracts the data for each step made by the left or right foot. The outputs are the step segments from the left and right insoles. Each step segment contains the force data from each sensor for each step made by the victim.

\textit{Training data extractor:} It detects the action of walking on stairs, and recognizes the turning steps made on stair landings. These turning steps are extracted as training data, which are used to train the walking path estimator.

\textit{Continuous walking detector:} This component recognizes the victim’s continuous walking in corridors based on the frequency domain analysis. Since we aim to recover the corridor map of the floor, we focus on the continuous walking in corridors instead of the intermittent walking in rooms.

\textit{Walking path estimator:} This component reconstructs each of the victim’s walking paths in corridors. For each footstep, it extracts features from the step segments, and estimates the
heading direction change based on SVM. Then, it reconstructs the path of each continuous walk. Note that the output walking paths are at the same scale, but do not necessarily follow the same rotation. The output paths are stored in the dataset of historical paths.

**Corridor map estimator:** Based on the historical paths, this component keeps estimating and updating the map of the corridors in the floor plan. If the victim is currently walking in corridors and the data are available in real time, it is also possible to locate the victim.

**Building recognizer:** Assume that the corridor maps of candidate buildings have been stored in a pool. This component searches in the pool and finds the corridor map that is most similar to the output of the corridor map estimator. The corresponding building of the best match is the recognition result, and thus the building is located.

Before the system can estimate the floor plan, the victim must have gone up or down stairs several times, so that enough training data can be collected to train the machine learning models. Note that the attacker can keep updating the training data during the victim’s daily walking.

Next, we introduce the algorithms of these components in detail, respectively.

### 6.4 STEP DETECTOR

This component aims to crop the force data sequences for each footstep. We treat \( \{u^l_i(t) | i = 1, ..., 8\} \) and \( \{u^r_i(t) | i = 1, ..., 8\} \) respectively. Similar to the method used in [60], we first calculate the total foot force on the insoles, i.e., \( u^l(t) = \sum_{i=1}^{8} u^l_i(t) \) and \( u^r(t) = \sum_{i=1}^{8} u^r_i(t) \), as shown by the last subgraph in Figure 6.3. Since the foot force always falls back to near-zero values when the shoe leaves the ground, we can segment steps by detecting the low values of \( u^l(t) \) or \( u^r(t) \). The start point of a left step is detected if \( u^l(t) \geq \gamma_1 \), and its end point is detected if \( u^l(t) < \gamma_1 \), where \( \gamma_1 \) is a constant threshold and \( \gamma_1 = 10 \) kg in our implementation. The right step detection follows the same method. An example of a detected left step is labeled in the last subgraph of Figure 6.3.

For each detected step, we crop the \( u^l_i(t) \) (or \( u^r_i(t) \)) between its start and end points. The cropped step segments are denoted by \( s^l_{i,k}(\tau) \) (or \( s^r_{i,k}(\tau) \)), where \( i = 1, ..., 8 \), and \( k = 1, 2, \ldots \) is the index for each left (or right) step. Note that the left and right steps are indexed independently, and \( s^l_{i,k}(\tau) \) is contiguous to \( s^r_{i,k}(\tau) \). Moreover, \( \tau \) denotes the time length since the start point of the step, and \( \tau = 0 \) at the beginning of the step segment. An example of the eight step segments for one left step is labeled by the red boxes in Figure 6.3.

As a result, for the \( k \)th left or right step, the step detector outputs the force data as \( s^l_{i,k}(\tau) \) or \( s^r_{i,k}(\tau) \), \( i = 1, ..., 8 \), which are sent to the training data extractor and the continuous
walking detector. For the purpose of periodic signal detection, $u^l(t)$ and $u^r(t)$ are also sent to the continuous walking detector.

### 6.5 TRAINING DATA EXTRACTOR

In this section, we first show the necessity of training an individual machine learning model for each victim, and then introduce how the training data can be collected during the activity of going up or down stairs.

#### 6.5.1 Necessity of Training Data for Walking Path Estimator

To estimate the walking paths of the victim based on force data only, it is necessary to train a machine learning model to recognize the victim’s actions, such as making turns. Since commercial smart shoes are still in infancy, the measurement differences caused by the diversity of embedded force sensors cannot be ignored. Thus, it is hard to train a general model that works for any smart shoes. While the learning technique of deep neural networks could be applied, a large amount of data is needed, which is less practical for attackers. Therefore, compared with training a single model that works for everyone, it is
more valuable to investigate if it is possible to train individual models for different victims.

Since we aim to reconstruct the topological corridor map only, the scale correctness is less important than the structure correctness of the output map. Thus, we do not estimate the stride length, the accuracy of which impacts the map scale only. We will show how we handle the stride length in Section 6.7.4.

Intuitively, to recognize the corners in the floor plan, it is necessary to detect the action of turning. It has been shown in [60] that SVM regression models are capable of estimating the direction change made by each step based on force data. Thus, we need to collect the labeled training data to train the SVM models for the turning angle estimation. We propose to extract the training data during the action of going up or down stairs. As shown in Figure 6.4, when a victim goes from Level 1 to Level 3, it is natural that he/she makes three U-turns at the three stair landings. Thus, it is possible to extract the data to train the turning angle estimation model, if we can detect the action of walking on stairs and thus determine the turning steps on the stair landings. Note that we do not assume the existence of stairs in the target building. Besides the target building, the training data can also be collected from any other buildings that have stairs.

6.5.2 Stair Landing Detection

There have been approaches that detect the action of going on stairs using inertial sensors [150, 124]. However, how to recognize this action based on the force data from shoes is still an open question. A naive method is monitoring the peak value changes of the entire force in shoes. Intuitively, because of the need of acceleration and deceleration, the impact force on insoles increases when the person is going on stairs. Thus, the increase of foot force implies the action of walking on stairs. However, in the real case, the entire foot force cannot be directly measured, because the force sensors are discretely and non-uniformly embedded in insoles. The sum of all the sensors’ force values does not reflect the real total impact force made by the foot. Moreover, this method suffers high false positive rate, because many other actions such as running and jumping can also lead to the changes of the maximum impact force.

To address these challenges, we propose the stair landing detection algorithm for in-shoe force sensors. The key idea is that, we detect the periodic change of the peak amplitudes of the force signal from each embedded sensor. As illustrated by Figure 6.5, while the sum of all the sensors’ outputs does not show any obvious change, the peak amplitudes of some sensors change obviously. This is because, depending on the person’s walking habit, some foot parts do not touch the stairs. As labeled by the red dashed lines in Figure 6.5(a), when the person
is going up stairs, the sensors at the inner and outer heel almost have no output since the heel does not touch the stairs. The output appears again when footsteps are made on stair landings. Thus, periodic changes exist in the signals. Even if the person walks following the habit shown in Figure 6.5(b), the periodic peak amplitude changes at the fourth toe are still visible, which is because walking through multiple floor levels is fundamentally a periodic action. The similar phenomenon also occurs for the case of going down stairs, as shown in Figure 6.5(c). Thus, by detecting such periodic changes, we can recognize the footsteps made on stairs. This method also eliminates the impacts of instantaneous impact force changes caused by running and jumping, because these actions do not show periodic force peak changes.

The observation illustrated by Figure 6.5 also shows that it is unreliable to use only one sensor to detect the action of going on stairs. When the person is walking on stairs, sometimes a sensor has visible periodic output but sometimes not, depending on various walking habits. In Figure 6.5(c), the outputs of the sensors at the inner and outer heel show obvious amplitude changes when the steps are made on stairs. However, in Figure 6.5(b), there is no obvious change at the heel. Instead, the output from the fourth toe contains periodic amplitude changes. Therefore, using the output from a single sensor is unreliable,
and we should make use of all the sensors to improve the detection accuracy.

The stair landing detection algorithm consists of five steps. Algorithm 6.1 shows Steps 1-4, which process the data from a single sensor in the left or right insole. Steps 1-4 are repeated for each of the 16 sensors, and in Step 5 we aggregate all the results. For brevity, in Steps 1-4, we make \(s_k(\tau), k = 1, 2, ..., K\) denote the \(K\) continuous step segments from one sensor, i.e., \(s^{l}_{i,k}(\tau)\) or \(s^{r}_{i,k}(\tau)\), \(k = 1, 2, ..., K\). Note that Steps 1-5 in the algorithm do not refer to the physical footsteps made by the victim.

(Step 1) *Signal conversion:* Figure 6.6(a) shows an example of the raw signal from a sensor (the partitions of step segments are omitted). Since we focus on the amplitude of the signal, in line 1, we first extract the peak value of each step segment, denoted by \(p[k]\). The peak value array extracted from Figure 6.6(a) is shown in Figure 6.6(b). In the following steps we process \(p[k]\) instead of \(s_k(\tau)\).

Figure 6.6: Example of signal conversion and rough stair detection. The data come from the sensor at the heel of the right foot.

(Step 2) *Rough stair detection:* We roughly detect the existence of stairs using a sliding window of size \(W = 30\). \(W\) is tuned based on the observation that there are usually less than 8 left/right steps in each flight of stairs [151].

In line 4-6, we first remove the steady component from the signal, and then conduct the Fast Fourier Transform (FFT) with a Hanning window. Two examples in frequency spectrum are illustrated in Figure 6.7. It is shown that strong periodic signal exists when the person is walking on stairs and stair landings. In line 7 we consider that the periodic signal exists if a peak in the amplitude-frequency diagram is higher than a constant threshold \(A_T = 60\) kg, and store the index of the peak’s frequency bin in \(r[m]\).

In line 8-9, we determine the ranges of the continuous steps made on stairs and stair landings, and crop them by *stair frames*. Two examples of stair frames are shown in Figure 6.6(b). Each stair frame contains a continuous walk from one floor to another, e.g., stair frame \((20, 69)\) denotes that the person is walking on a staircase from the 20th to the 69th right step. The range accuracy will be further refined in Step 3. We store the stair frames in set \(\Pi\). Note that the boundary condition checking for \(r[k_1 - 1]\) and \(r[k_2 - W + 2]\) in line 9 is
ALGORITHM 6.1: Turning step detection based on a single sensor

Input: Step segments $s_k(\tau)$, $k = 1, ..., K$;
Parameter: Window size $W$; Amplitude threshold $A_T$; Minimum stair frame size $\Gamma$; Minimum period $\psi_T$; Minimum standard deviation $\sigma_{T1}$; Maximum standard deviation $\sigma_{T2}$; Maximum peak width $\psi_d$; Maximum turning step difference rate $r_p$;
Output: Stair frame set $\Pi$: Turning step sets $U_g, g = 1, ..., |\Pi|$; Stair periods $\tilde{\psi}_g, g = 1, ..., |\Pi|$;

// (Step 1) Signal conversion
1 $p[k] \leftarrow \max_{\tau} s_k(\tau), k = 1, 2, ..., K$;

// (Step 2) Rough stair detection
2 $r[m] \leftarrow 0, k = 1, 2, ..., K$;
3 foreach $m = 1, 2, ..., K-W+1$ do
4 \[
\hat{p}[k] \leftarrow p[k] - \frac{1}{W} \sum_{k=m}^{m+W-1} p[k], k = m, ..., m + W - 1;
\]
5 Get Hanning window $h = \text{hanning}(W)$;
6 \{\{A[n]|n = 1, ..., W\} \leftarrow \frac{1}{W} |\text{FFT}(\{\{\hat{p}[k] \cdot h[k-m+1]|k = m, ..., m + W - 1\})|\};
7 if peaks exist in \{\{A[n]\} with altitude larger than $A_T$ then $r[m]$ \leftarrow \text{the bin index of the first peak with altitude larger than $A_T$};
8 $\Pi \leftarrow \emptyset$;
9 foreach $(k_1, k_2) \text{ that satisfies } k_2 - k_1 > \Gamma, r[k] > 0, k = k_1, ..., k_2-W+1 \text{ and } r[k_1-1] = 0,$
   $r[k_2-W+2] = 0$ do $\Pi \leftarrow \Pi \cup \{(k_1, k_2)\}$;

// (Step 3-4) Run the following loop for each stair frame in $\Pi$
10 foreach $(k_1, k_2) \in \Pi, g = 1, ..., |\Pi|$ do
11 // (Step 3) Period Estimation
12 $\Psi \leftarrow \emptyset$;
13 $k_1' \leftarrow \infty; k_2' \leftarrow -\infty;
14$ foreach $m = k_1^g, ..., k_2^g - W + 1$ do
15 \[
\psi \leftarrow \frac{W}{(r[m] - 1) - 0.5};
\]
16 $\sigma \leftarrow \frac{1}{W} \sum_{k=m}^{m+W-1}(\hat{p}[k] - \frac{1}{W} \sum_{k=m}^{m+W-1} p[k])^2$;
17 if $\psi \geq \psi_T$ and $\sigma \geq \sigma_{T1}$ then \[\begin{align*}
N_\psi & \leftarrow \lfloor W/\psi \rfloor; \\
& \text{Break } \{p[k]|k = m, ..., m + \psi N_\psi - 1\} \text{ into vectors } v_j, j = 1, ..., N_\psi, \text{ each with length } \psi;
\]
18 $\sigma' \leftarrow \frac{1}{\psi N_\psi} \sum_{j=1}^{N_\psi} \|v_j - \frac{1}{N_\psi} \sum_{j=1}^{N_\psi} v_j\|^2$;
19 if $\sigma' \leq \sigma_{T2}$ then \[
\Psi \leftarrow \Psi \cup \{\psi\};
\]
20 if $m < k_1'$ then $k_1' \leftarrow m$;
21 if $m + W - 1 > k_2'$ then $k_2' \leftarrow m + W - 1$;
22 \[
\hat{p}[k] \leftarrow \hat{p}[k_1^g]; k_2^g \leftarrow k_2';
\]
23 $\tilde{\psi}_g \leftarrow \text{mod}(\Psi)$;

// (Step 4) Turning Step Extraction
24 Find all peaks in $p[k]$ (or $-\hat{p}[k]$), $k = k_1^g, ..., k_2^g$ that meet the following conditions:
25 (1) The peaks are separated by more than the minimum peak distance $\tilde{\psi}_g - \psi_d$;
26 (2) The peak height is larger than $\hat{p}$ (or $-\hat{p}$), where $\hat{p} = \frac{1}{k_2^g - k_1^g + 1} \sum_{k=k_1^g}^{k_2^g} p[k]$;
27 Denote the average peak height by $\bar{p}$ (or $\bar{\hat{p}}$). Denote the indexes of the peaks by set $U_1$ (or $U_2$);
28 if $|\bar{p} - \bar{\hat{p}}| > |\bar{p} - \bar{\hat{p}}|$ then $U \leftarrow U_1$; else $U \leftarrow U_2$;
29 $\tilde{U}_g \leftarrow \emptyset$;
30 foreach $k \in U$ do
31 foreach $k' = k - 4, ..., k + 4$ do
32 \[
\text{if } |p[k'] - p[k]| < p[k] r_p \text{ then } \tilde{U}_g \leftarrow \tilde{U}_g \cup \{k'\};
\]
33 return $\Pi, \tilde{U}_g \text{ and } \tilde{\psi}_g, g = 1, ..., |\Pi|$;
omitted for brevity, and the minimum stair frame size is $\Gamma = 35$ during our implementation.

The following Steps 3, 4 are executed for each stair frame $(k^g_1, k^g_2)$, $g = 1, ..., |\Pi|$.

(Step 3) Period Estimation: One peculiarity of signal $p[k]$ is that, since the stair number between adjacent stair landings is usually constant, the period of $p[k]$ in each stair frame should also be roughly constant. Although the steps in each period also include a various number of steps made on a stair landing, based on our observation during our experiment, the step number difference is usually limited during a single walk. Moreover, since the index $k$ is the count of steps instead of time, this period is independent of the person’s walking speed on stairs. We can further tune the ranges of stair frames and get more accurate period length.

In line 13-23, for the signal within each stair frame $(k^g_1, k^g_2)$, we apply a new sliding window of size $W$. In line 14-15, we calculate the signal period $\psi$ based on the FFT result in Step 2, as well as the standard deviation $\sigma$ of the signal in the window. The results are discarded if the $\psi$ is shorter than $\psi_T$, or $\sigma$ is smaller than $\sigma_{T1}$ (line 16). $\psi_T = 5$ and $\sigma_{T1} = 2$ kg are two constant thresholds. This lowers the false positive rate by reducing the impacts of burst signals and weak periodic noises.

In line 18, we break the signal in the window into segments based on the period length $\psi$, and treat them as vectors. The trailing signal is discarded if the window length is not the integral multiple of $\psi$. One example is shown in Figure 6.8. Then, we calculate the standard deviation $\sigma'$ of the vectors, and check if it is smaller than a constant threshold $\sigma_{T2} = 16$ kg (line 19-20). Note that the impact of various vector length has been removed during the calculation in line 19. If $\sigma' \leq \sigma_{T2}$, the signals in different segments are similar enough to each other, and we record the period $\psi$ in set $\Psi$. After the sliding window has gone through the stair frame, we use the most common element in $\Psi$ as the signal period $\bar{\psi}_g$ in the stair frame (line 25). Moreover, we shrink the stair frame range by finding the first and the last window position that satisfies $\sigma' \leq \sigma_{T2}$ (line 22-24). Two examples are given in Figure 6.6(b). It is shown that this method fine-tunes the start and end points of the on-stair walk.

(Step 4) Turning Step Extraction: While the start and end points of the stair frames have been estimated, we still need to determine the steps made on stair landings. Naturally, the steps made on stairs are more than those made on stair landings. Thus, if the signal $p[k]$ flips between two states, i.e., on-stair state and on-landing state, the state with less steps should be the on-landing state.

Based on our observation on 15 volunteers, there are two pattern types of the signal $p[k]$ in a stair frame, as shown in Figure 6.9. The two data examples are collected from two volunteers who are going up stairs, but at the same sensor position. It is shown that the
signal is lower when the first volunteer is walking on stairs (Figure 6.9(a)). However, this is inverse for the second volunteer (Figure 6.9(b)). To handle this, in line 26-29 we find the positive/negative peaks which are higher/lower than the average signal, and then calculate the average positive and negative peak heights. As shown in Figure 6.9(a), if the average negative peak height is nearer to the average signal, the positive peaks should denote the steps made on stair landings (line 30). Otherwise, the negative peaks denote the steps made on stair landings.

We need further extend the on-landing steps since currently only one step is detected for each stair landing. In line 31-34, for each detected on-landing step, we search in its adjacent eight steps, and regard the steps with similar signal values as additional on-landing steps. In our implementation we set the similarity threshold as $r_p = 0.3$. The step indexes of all the on-landing steps are included in set $\tilde{U}_g$.

Therefore, for each stair frame $(k_1^g, k_2^g)$, the output of Steps 3, 4 is the turning step set $\tilde{U}_g$. The total stair frame number is $|\Pi|$.

(Step 5) Aggregation: Following Algorithm 6.1 (Steps 1-4), we can recognize the turning steps based on the output of a single sensor. The same steps are applied to all the 16 sensors, and the results are denoted by $\tilde{U}_{i,g}^l, g = 1, ..., |\Pi_i^l|, i = 1, ..., 8$ and $\tilde{U}_{i,g}^r, g = 1, ..., |\Pi_i^r|, i = ...$.
1, ..., 8. We aggregate all the turning steps by

\[
\hat{U}_l = \bigcup_{i=1}^{8} \bigcup_{g=1}^{\|\Pi_l\|} \hat{U}_{i,g}, \quad \hat{U}_r = \bigcup_{i=1}^{8} \bigcup_{g=1}^{\|\Pi_r\|} \hat{U}_{i,g}.
\] (6.1)

\(\hat{U}_l\) and \(\hat{U}_r\) include the steps that can be used to train the machine learning model in the walking path estimator. Note that we have not distinguished the left-turn and right-turn steps, which will be discussed in Section 6.5.3.

In daily life, there are some other activities that would lead to the periodic foot force changes, such as doing exercises. However, our design handles this problem well. As shown in Step 3, we utilize the peculiarity that the stair number between every two stair landings is usually fixed and not too small. Thus, the period of signal \(p[k]\) on staircases is usually stable, which is unique compared with those of other periodic activities. Therefore, our algorithm achieves a low false positive rate when detecting the action of going on stairs, which will also be confirmed by the experiment in Section 6.10.1.

Since Algorithm 6.1 also outputs the stair frame set and the signal periods, we can easily recognize the level that the person has gone to. For instance, in a typical office building, if a stair frame is \((121, 162)\) and \(\bar{\psi}_g = 10\), the person should have gone up or down by 2 levels because \((162 - 121 + 1)/10 \approx 4\) stair sections appear in this frame. The attacker can use this feature to locate the building level that the victim is walking in.

6.5.3 Training Data Labeling

Absolute direction change labeling: For each step in \(\hat{U}_l\) and \(\hat{U}_r\), we need to label the heading direction change that it makes, which is hard if there is no observation about the victim’s walking. However, since a U-turn is made on each stair landing, we can first label the absolute direction change made by each turning step.

If we merge the steps in \(\hat{U}_l\) and \(\hat{U}_r\) and sort them in time order, usually the adjacent continuous left and right steps will fall in a group of 2 - 7 steps. The steps in each group are made on the same stair landing. We name each group as a stair landing group. Three examples are show in Figure 6.10. Note that the first step on the stair landing is excluded from the group, because it is actually a step made on a stair, even if it also causes the walking direction change. Therefore, if we denote the size of a stair landing group by \(L\), the average absolute direction change made by each step in the group is \(180^\circ/(L + 1)\). We label the steps in the group with this angle (in \((0^\circ, 90^\circ]\)).

Turning direction labeling: While we have labeled the training steps with absolute direction
changes, the turning direction (i.e., left or right) of the steps is still unknown. Since there is no common rule about whether the staircases should be left-handed or right-handed, currently we cannot determine whether a step is turning left or right. However, the turning directions are usually inverse in two adjacent walks on staircases. For instance, a person always makes left/right U-turns when going up stairs, and then makes right/left U-turns when going down stairs. This makes it possible to distinguish different turning directions. While we have gotten the stair frames in Algorithm 6.1, it is easy to distinguish different walks in stairwells. By default, we simply label the turning steps in the first walk with “left”, and label those in the second walk as “right”, and so on. If we flip the direction labels, the output of the corridor map estimator will simply be symmetrically flipped, which does not impact the effectiveness of our attack method, because the attacker can still easily recognize the real floor plan on maps using our output.

Therefore, for each turning step in $\hat{U}_l$ and $\hat{U}_r$, we label it with the absolute direction change in $(0°,90°]$ and the turning direction (“left” and “right”). For the convenience of expression, we reformat the two labels following the format in Figure 6.11. Thus, the label attached to a turning steps is a value within $[-90°, 0°) \cup (0°, 90°]$. We still need to extract the training steps that make no direction change (i.e., steps labeled with $0°$).

*Straight step labeling:* In Section 6.6, we will show how we can detect the continuous walking in corridors. Assume that we have known which steps belong to the continuous walking based on the output of the continuous walking detector. Based on the observation that the number of straight steps is much larger than that of turning steps, we randomly select the steps during the continuous walking, and regard them as straight steps (labeled with $0°$). Indeed, these “straight” steps would contain turning steps. However, as long as the training data size is sufficiently large and the balance among the training steps with different degrees is carefully kept, the impacts of the wrongly-labeled steps is limited, which will be confirmed by the experiment in Section 6.10.2.
Thus, the training data extractor detects the steps that can be used for training, and assigns them with the degrees within $[-90^\circ, 90^\circ]$. The signals of the training steps and their labels are sent to the walking path estimator.

6.6 CONTINUOUS WALKING DETECTOR

Typically, the indoor walking can be classified into two types: in-room walking and in-corridor walking. The former contains complex turning and intermittence, while the latter is usually continuous. As mentioned in Section 6.1, ShoesHacker aims to recover the corridor map of the floor plan, which is sufficient for attackers to locate the building on global maps. Thus, we only focus on the in-corridor walking.

In practice, it is hard to recover the whole corridor map at once, because the victim is unlikely to traverse the whole floor within one walk. Instead, we need to extract multiple in-corridor walks during the victim’s daily walking, based on which we estimate multiple walking paths and reconstruct the corridor map. Thus, we need to detect the continuous walks in corridors and separate them from in-room walks.

The method we use is based on frequency-domain analysis, which is similar to Step 2 of Algorithm 6.1. However, we conduct FFT on the consecutive force data directly, instead of on the peak value sequences. Since the two feet have the same in-corridor/room walking state, we only use the data from the left shoe to detect the in-corridor walking. We apply a sliding window with size $W' = 60$ on the total force $u_l(t)$, the sampling rate of which is still 15 Hz. Assume that the window currently starts at time $t'$ in $u_l(t)$, we remove the steady component by

$$\tilde{u}_l(t) = u_l(t) - \frac{1}{W'} \sum_{t=t'}^{t'+W'-1} u_l(t'), \quad t = t', ..., t' + W' - 1.$$  \hspace{1cm} (6.2)

Then we conduct FFT on $\tilde{u}_l(t)$ by

$$\{A_l[n]|n = 1, ..., W'\} = \frac{1}{W'} \|FFT(\{\tilde{u}_l(t) \cdot h'[t - t' + 1]|t = t', ..., t' + W' - 1\})\|,$$  \hspace{1cm} (6.3)

where $h'[t]$ denotes a Hanning window with length $W'$. If there exists a peak in $\{A_l[n]\}$ with altitude larger than 5 kg, we consider that the person is walking during $t'$ to $t' + W' - 1$. If we detect that the person keeps walking during $t_1$ to $t_2$, and $t_2 - t_1 >= 10$ s, we regard the signals in $t = t_1, ..., t_2$ contain an in-corridor walking path. The corresponding step segments $s_{i,k}^l(\tau)$ and $s_{i,k}^r(\tau)$ within this time range are sent to the walking path estimator to restore the trajectory of this walk.
Note that the continuous walking detector would find multiple in-corridor walks in the signal $u(t)$. Assume that $N_w$ in-corridor walks are detected. We output the step segments of each walk separately, and the walking path estimator will process them respectively.

6.7 WALKING PATH ESTIMATOR

When a person is walking on a floor, the force distribution changes in shoes imply the changes of walking directions. With the basic assumption that all the corners in the building are $90^\circ$, we show that it is possible to roughly estimate the person’s walking path based on the force changes on the insoles.

We first show our design consideration, and then introduce how we extract features, estimate direction changes, and generate the estimated walking path for each of the $N_w$ walks detected by the continuous walking detector.

6.7.1 Design Considerations

When a person is making a turn, the foot force on insoles is different from that during a straight walking. As shown in Figure 6.3, when the person makes a right turn using the right foot around the 4th second, the force values at the hallux and the inner metatarsals are obviously different from those during the straight walking. Thus, it is possible to detect the turning actions based on the force changes. Moreover, as shown in [60], when the training data are available, the heading direction change in degrees can be further estimated.

Although we assume that the paths in maps are latticed, we cannot simply classify the footsteps as “going straight” and “making a left/right turn”. As shown in Figure 6.12, typically a person makes multiple steps to make a $90^\circ$ turn, and each step makes a direction change smaller than $90^\circ$. Thus, we need to estimate the heading direction change made by each step, and accumulate the changes to detect if the turning action really happens.

In the walking path estimator, we extract features from step segments, and then estimate the direction changes made by the steps using trained SVMs, which is similar to the method in [60]. Moreover, based on the sequence of the direction changes, we further estimate the person’s walking path.

6.7.2 Feature Extraction

For the step segments of each step, we extract a set of features. The pattern in the features should not be highly impacted by the victim’s walking speed, and thus the features we use
are mainly based on the amplitude of the force signals. Specifically, for the step segment $s^{l}_{i,k}(\tau)$ or $s^{r}_{i,k}(\tau)$, the features are

$$F^{l}_{i,k} = \max_{\tau} s^{l}_{i,k}(\tau), \quad i = 1, 2, ..., 8,$$

or

$$F^{r}_{i,k} = \max_{\tau} s^{r}_{i,k}(\tau), \quad i = 1, 2, ..., 8. \quad (6.4)$$

Therefore, for the $k$th step made by the left or the right foot, the feature set $\{F^{l}_{i,k} | i = 1, 2, ..., 8\}$ or $\{F^{r}_{i,k} | i = 1, 2, ..., 8\}$ are sent to the subcomponent of angle regression.

### 6.7.3 Angle Regression

In Section 6.5.3, we have gotten the labeled training data. In the offline stage, the data from left and right foot are used to train two SVM regression models with linear kernels, respectively. Since people’s walking patterns of left and right feet are proved to be asymmetric [117], it is necessary to train a separate SVM model for each foot.

In the online stage, the walking path estimator would receive multiple step segment sets belonging to different in-corridor walks. We process each walk respectively. In each walk, for the victim’s $k$th left or right step, the corresponding SVM receives the extracted features $\{F^{l}_{i,k} | i = 1, 2, ..., 8\}$ or $\{F^{r}_{i,k} | i = 1, 2, ..., 8\}$. Then, the direction change caused by the step is predicted. The result follows the format shown in Figure 6.11. For each walk, we get two angle change sequences of the left and right steps. The two sequences are merged and indexed by their time stamps, and the result is a single angle change sequence. We denote this sequence by $\{\Delta \theta_{n} | n = 1, 2, \ldots\}$, and an example is shown in Figure 6.13(a). Note that $\{\Delta \theta_{n}\}$ is the output for a single in-corridor walk. We repeat the same process for each walk.

### 6.7.4 Path Generation

For each of the $N_{w}$ walks, based on its angle change sequence $\{\Delta \theta_{n}\}$, we can roughly estimate the person’s walking path. Intuitively, in order to draw the walking path, besides
the walking direction changes, the stride length should also be necessary. However, due to the lack of the observation on the victim, it is hard to estimate the stride length of each step. Previous works show that the stride length and the cadence are linearly correlated [152]. However, the detailed parameters of the linear relation vary among different people [153]. Since we only aim to roughly estimate the shape of the corridor structure, we assume that the stride lengths of all the steps follow the same parameter $l_{\text{stride}}$. Obviously, $l_{\text{stride}}$ only impacts the scale of the estimated map output by ShoesHacker. Without loss of generality, we set $l_{\text{stride}} = 0.77\, \text{m}$, and follow this scale in the following parameter setting.

In sequence $\{\Delta \theta_n\}$, if the cumulative direction change of three continuous steps is larger than $60^\circ$ or lower than $-60^\circ$, the three steps are labeled as potential left or right turning steps. Assume that the indexes of the continuous potential turning steps are $a, ..., b$, we label the step with the index $\lfloor (a + b)/2 \rfloor$ as a left or right turning step. The other steps are recognized as straight steps. All the steps have the same stride length $l_{\text{stride}} = 0.77\, \text{m}$.

We discard the steps before the first turning point or after the last turning point. Without loss of generality, we set the coordinates of the starting point (the first turning point) of the path as $(0, 0)$, and the direction of the first step is parallel with the vector $(0, 1)$. Thus, we can plot the path based on $\{\Delta \theta_n\}$. One example is shown in Figure 6.13(b). For the $w$th in-corridor walk, the path can be stored as a tuple sequence $\{(x^w_i, y^w_i) | i = 1, ..., N^w_c\}$, which denotes the coordinates of the corners (including the start and end points) on the walking path. $N^w_c$ is the total number of turns during the $w$th walk, and $w = 1, ..., N_w$.

Note that the tuples in $\{(x^w_i, y^w_i) | i = 1, ..., N^w_c\}$ are ordered, and the coordinates only show the relative locations of the turning points. Then absolute location and direction of the path are unknown.

### 6.8 CORRIDOR MAP ESTIMATOR

The corridor map estimator merges the paths $\{(x^w_i, y^w_i) | i = 1, ..., N^w_c\}, w = 1, ..., N_w$, and reconstructs the corridor map in the floor plan.

### 6.8.1 Design Considerations

As illustrated by Figure 6.14, the basic operation of the corridor map estimator is merging a pair of paths. However, the absolute locations and directions of the estimated paths are unknown. Thus, we cannot merge two paths by simply overlapping them. Some previous works in the field of image processing have proposed image blending methods to merge the images with common regions [154, 155]. While we already have the paths stored as
sequences, the problem can be simplified. A naive solution is treating each walking path as an action sequence, such as “go straight for 5 m, turn left, go straight for 10 m, turn right, ...”, and find if there are similar segments between each pair of paths. The paths can be merged if continuous segments match. However, two paths would have the same shape but with different corridor orders. One example is shown in Figure 6.15(a). Path 1 and 2 have the same shape, but their action sequences are different and thus cannot be easily merged.

To handle this problem, the key idea of the corridor map estimator is that, when merging two paths, we treat the first path as an undirected graph, and check if any part of the second path can fit into the graph. If the matching part is long enough, the two paths can be merged, and the non-matching part of the second path is added to the graph to form a new undirected graph, which is the base of the next path merging. We name this undirected graph as base graph. As shown in Figure 6.15(b), with this method, Path 2 can completely fit into Path 1.

During the path merging, we also need to handle the case that some nearby intersections (or corridors) are actually the same one, or two orthogonal corridors intersect each other at an undetected intersection. Thus, our algorithm should merge nearby intersections, or even create new intersections. Three examples are shown in Figure 6.16. Note that in Figure 6.16(c), even if the person does not make turns when walking from point a to b and c to d, we still need to create a new intersection between the two edges.

In the following subsections, we describe the details of our algorithm. The workflow is illustrated in Figure 6.17.
6.8.2 Path Preprocessing

For each in-corridor walking path, the walking path estimator outputs \( \{(x_i^w, y_i^w)|i = 1, ..., N_c^w\} \). For the convenience of graph-based processing, we first convert this tuple sequence to the format of vertex set \( V_w \), edge set \( E_w \) and traverse order \( \vec{\omega}_w \), as shown in Figure 6.17. We treat each path corner as a vertex in \( V_w \), denoted by \( v \). We make \( v.x \) and \( v.y \) denote the coordinates of the vertex. The elements in \( E_w \) follow the format \( e(v_1, v_2) \), which denotes the edge (path) between two vertices \( v_1, v_2 \). All the edges are undirected, i.e., \( e(v_1, v_2) = e(v_2, v_1) \). \( \vec{\omega}_w \) is a sequence of vertices, which denotes the order by which the person went through the vertices. \( \vec{\omega}_w[i] \) denotes the \( i \)th vertex in \( \vec{\omega}_w \). During the conversion, we merge the nearby vertices or edges, and create new vertices or edges if new intersections are generated.

The algorithm is shown in Algorithm 6.2. In line 2 we start to traverse each tuple in \( \{(x_i^w, y_i^w)|i = 1, ..., N_c^w\} \). In line 3-4, if the current corner is near to a previously-visited vertex, we consider that the person went through the previous vertex, and merge the two vertices. In our implementation, the distance bound of vertex merging is \( d_{T1} = 4 \) m. A larger \( d_{T1} \) could increase the risk of merging two real intersections by mistake. The case in Figure 6.16(a) is handled by this method.

In line 6-13, we check if the current corner is near to any previous visited edge by drawing a vertical line to the edge. If the foot point is on the edge and its distance to the current corner is shorter than \( d_{T1} \), we break the edge to two edges and create a new vertex at the foot point. Then we consider that the current corner is at the new vertex. The case that the corner is near to the edge but the foot point is off the edge is equivalent to the case that the corner is near to one of the edge’s endpoints, which has been handled in line 3-4. This method handles the case in Figure 6.16(b). If no nearby edge is detected, a new vertex is
ALGORITHM 6.2: Path preprocessing

Input: Path tuple sequence \( \{(x_i^w, y_i^w)\}_{i = 1, \ldots, N_w^w}\); parameter: Distance threshold \(d_{T_1}\); output: Vertex set \(V_w\); Edge set \(E_w\); Traverse order \(\bar{\phi}_w\);

\( V \leftarrow \emptyset; E \leftarrow \emptyset; \bar{\phi} \leftarrow \square; \)

\( \text{foreach } i = 1, \ldots, N_w^w \text{ do} \)

\( \quad \text{if } \sqrt{(x_i^w - v.x)^2 + (y_i^w - v.y)^2} < d_{T_1}, \exists v \in V \text{ then} \)

\( \quad \quad \text{// Merge nearby intersections} \)

\( \quad \quad \text{Add } v \text{ to the end of } \bar{\phi}; \)

\( \quad \text{else} \quad \quad \text{foreach } e(v_1, v_2) \in E \text{ do} \)

\( \quad \quad \text{Calculate the vertical line from vertex } (x_i^w, y_i^w) \text{ to edge } e(v_1, v_2); \)

\( \quad \quad \text{if the foot point } (x, y) \text{ is on the edge, and } \sqrt{(x_i^w - x)^2 + (y_i^w - y)^2} < d_{T_1} \text{ then} \)

\( \quad \quad \quad \text{// Merge nearby corridors} \)

\( \quad \quad \quad \text{Create a new vertex } v, \text{ where } v.x = x, v.y = y; \)

\( \quad \quad \quad V \leftarrow V \cup \{v\}; \)

\( \quad \quad \quad E \leftarrow E \setminus \{e(v_1, v_2)\}; E \leftarrow E \cup \{e(v_1, v), e(v, v_2)\}; \)

\( \quad \quad \quad \text{Insert } v \text{ between any adjacent } v_1 \text{ and } v_2 \text{ in } \bar{\phi}, \text{ and add } v \text{ to the end of } \bar{\phi}; \)

\( \quad \quad \text{break; } \)

\( \quad \text{if no new vertex is added in line 6-13 then} \)

\( \quad \quad \text{Create a new vertex } v, \text{ where } v.x = x_i^w, v.y = y_i^w; \)

\( \quad \quad V \leftarrow V \cup \{v\}; \)

\( \quad \quad \text{Add } v \text{ to the end of } \bar{\phi}; \)

\( \quad \text{end if} \)

\( \text{if } \|\bar{\phi}\| < 2 \text{ then continue; } \)

\( \text{// Add new vertex if two edges intersects} \)

\( \text{Denote the current last two elements in } \bar{\phi} \text{ by } v_1 \text{ and } v_2; \)

\( \text{if the segment from } v_1 \text{ to } v_2 \text{ successively intersects edges } e(v_1^j, v_2^j) \in E, j = 1, \ldots, J \text{ then} \)

\( \text{foreach } j = 1, \ldots, J \text{ do} \)

\( \quad \text{Denote the intersection by } (x, y). \text{ Create a new vertex } v, \text{ where } v.x = x, v.y = y; \)

\( \quad V \leftarrow V \cup \{v\}; \)

\( \quad E \leftarrow E \setminus \{e(v_1^j, v_2^j)\}; E \leftarrow E \cup \{e(v_1^j, v), e(v, v_2^j), e(v_1, v), e(v, v_2)\}; \)

\( \quad \text{Insert } v \text{ between any adjacent } v_1 \text{ and } v_2 \text{ in } \bar{\phi}; \)

\( \quad v_1 \leftarrow v; \)

\( \text{end if} \)

\( \text{if no new vertex is added in line 20-26 then} \quad E \leftarrow E \cup \{e(v_1, v_2)\}; \quad \)

\( \text{Create a new path tuple sequence } \{(x_i^w, y_i^w)\} \text{ based on } \bar{\phi} \text{ and } V, \text{ and then reverse the tuple order ;} \)

\( \text{Repeat line 1-27 with the new path tuple sequence as the input. Get new } V, \text{ } E \text{ and } \bar{\phi}, \text{ and relabel} \)

\( \text{them by } V_w, \text{ } E_w \text{ and } \bar{\phi}_w, \text{ respectively; } \)

\( \text{return } V_w, \text{ } E_w \text{ and } \bar{\phi}_w; \)

created for this corner (line 14-17).

In line 19-27, we create the new edge from the last visited vertex to the current vertex, and also check if the new edge intersects with any previous edges. If so, we break the two intersected edges to four edges, and add a new vertex at the intersection point. The case shown Figure 6.16(c) is handled by this method.

In line 28-29, we traverse the path from the other end, and repeat the previous process. The reason is that some merging would be left out if the path is traversed only once. For
example, in Figure 6.16(b), if we traverse the path from a to b, the merging does not happen. We need to traverse again from b to a to detect this need of merging.

For each path, we get $V_w$, $E_w$ and $\vec{o}_w$, based on which we conduct the following path merging method.

6.8.3 Path Merging

The path merging algorithm is shown in Algorithms 6.3-6.5. Initially, among the paths to merge, we randomly select one path as the base graph. Its vertex set and edge set are denoted by $V$ and $E$. We first need to extend the base graph following Algorithm 6.3, and then merge each path into the base graph with following Algorithm 6.4. After each merging, we re-conduct Algorithm 6.3 on the base graph before we try to merge the next path.

**ALGORITHM 6.3:** Base graph extension

<table>
<thead>
<tr>
<th>Input:</th>
<th>Edge set $E$;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>Extended edge set $\hat{E}$;</td>
</tr>
<tr>
<td>$1$</td>
<td>$E' \leftarrow \emptyset;$</td>
</tr>
<tr>
<td>$2$</td>
<td>foreach $e(v_1, v_2) \in E$ and $e(v_1, v_2) \notin E'$ do</td>
</tr>
<tr>
<td></td>
<td>$V' \leftarrow {v_1, v_2};$</td>
</tr>
<tr>
<td>$4$</td>
<td>while the size of $V'$ has changed do</td>
</tr>
<tr>
<td></td>
<td>foreach $v \in V'$ do</td>
</tr>
<tr>
<td></td>
<td>foreach $e(v'_1, v'_2) \in E$ do</td>
</tr>
<tr>
<td></td>
<td>if $v'_1 = v$ or $v'_2 = v$, and $e(v'_1, v'_2)$ is collinear with $e(v_1, v_2)$ then</td>
</tr>
<tr>
<td></td>
<td>$V' \leftarrow V' \cup {v'_1, v'_2}$;</td>
</tr>
<tr>
<td>$7$</td>
<td>foreach combination of two vertices $v_a, v_b$ in $V'$ do</td>
</tr>
<tr>
<td></td>
<td>$E' \leftarrow E' \cup {e(v_a, v_b)}$;</td>
</tr>
<tr>
<td>$9$</td>
<td>$\hat{E} \leftarrow E \cup E'$;</td>
</tr>
<tr>
<td>$10$</td>
<td>return $\hat{E}$;</td>
</tr>
</tbody>
</table>

**Base Graph Extension:** The reason why we need to extend the base graph is that, while we have merged some nearby edges during the path preprocessing or the previous merging, the edges between some collinear vertices would still be missing. As illustrated in Figure 6.18(a), after we have merged vertex c and d to edge $e(a, b)$, edge $e(b, d)$ does not exist in $E$. Then, even though the new path actually goes through $e(b, d)$, it cannot be directly fitted into the base graph because $e(b, d)$ is broken into $e(b, c)$ and $e(c, d)$. In Figure 6.18(b), after we have merged vertex b and b', edge $e(a, c)$ is still not in $E$, and thus the new path cannot directly fit into $e(a, c)$.

For the convenience of future merging, we first extend the base graph following Algorithm 6.3. The idea is that, for each edge $e(v_1, v_2)$ in the base graph, we try to find all the other edges that are both collinear with and connected to $e(v_1, v_2)$ (line 6-7). If a new edge $e(v'_1, v'_2)$ satisfies this condition, we further search for the other edges that are both collinear with...
Figure 6.18: The necessity of extending the edge set in the base graph.

and connected to $e(v'_1, v'_2)$. This process is repeated until no new edge is founded. Then we add new edges to fully connect all the vertices on these edges (line 8). The output is the extended edge set $\hat{E}$. For instance, in Figure 6.18(a) all the 6 edges among a, b, c, d are included in $\hat{E}$, and in Figure 6.18(b), $e(a, b), e(a, c), e(b, c)$ are included in $\hat{E}$.

**Merging A New Path to Base Graph:** After we have extended the base graph, in Algorithm 6.4, we start to measure how well the new path can fit into the base graph. We define the overlapping part of two paths as a *matched segment*, as illustrated in Figure 6.14. Since any vertex could be the start point of the matched segment, and the initial walking direction is unknown, we try all the vertices and initial directions. For each vertex $v_1$ in the base graph, we try each edge $e(v_1, v_2)$ as the start edge of the path (line 2-3). Given the start edge, we run Algorithm 6.5 to get the matching level of the path (line 4).

In Algorithm 6.5, since any vertex on the new path could be the start point of the matched segment, we try each vertex $\vec{o}_w[i]$ as the start point (line 2). In line 3-7, we check the length difference between the start edge in the base graph and the edge $e_w(\vec{o}_w[i], \vec{o}_w[i + 1])$ in the new path. If the difference is larger than a constant threshold $d_{T2} = 10$ m, we consider that the two edges do not match and continue to try the next start point. If the two edges match, we traverse the following edges of the new path and find if the similar edges exist in the base graph (line 8-22).

Assume that the current vertex on the new path is $\vec{o}_w[j]$, and the previous trajectory $[\vec{o}_w[i], ..., \vec{o}_w[j - 1]]$ matches with the path $[..., v_1, v_2]$ in the base graph. We check if there is an edge $e(v_2, v_3)$ in the base graph that makes the corner from $e(v_1, v_2)$ to $e(v_2, v_3)$ and the corner from $e_w(\vec{o}_w[j - 2], \vec{o}_w[j - 1])$ to $e_w(\vec{o}_w[j - 1], \vec{o}_w[j])$ have the same turning direction (line 10-11). Among all the found edges, we select the one that has the minimum edge length difference $d$ from $e_w(\vec{o}_w[j - 1], \vec{o}_w[j])$ (line 12-14). If $d \leq d_{T2}$, the two edges match, and we update the matched segment (line 17-20) and continue finding the next matched edge in the base graph.

During the matching process, we record the matched segment and the maximum number of matched edges. If two matched segments have the same matched edge number, we select the one with the minimum cumulative edge length difference (line 21-22). The matched segment is returned to Algorithm 6.4.

Back to line 4 in Algorithm 6.4, we try each start edge, and record the best matched
ALGORITHM 6.4: Merge the base graph and a new path

Input: Base graph’s vertex set \( V \) and extended edge set \( \hat{E} \); New path’s vertex set \( V_w \), edge set \( E_w \), and traverse order \( \delta_w \);

Parameter: Minimum matching edge number \( N_e \); Distance threshold \( d_{T1} \);

Output: Merged vertex set \( \hat{V} \); Merged edge set \( \hat{E} \); Edge counter set \( C_E \);

1. \( \hat{V} \leftarrow V; \hat{E} \leftarrow \hat{E}; N_M' \leftarrow 0; d_M' \leftarrow \infty; \delta_B \leftarrow []; \delta_p \leftarrow []; \)
2. foreach \( v_1 \in \hat{V} \) do
   3. foreach \( e(v_1, v_2) \in \hat{E} \) do
      4. Run Algorithm 6.5 on \( V, \hat{E}, e(v_1, v_2), V_w, E_w \) and \( \delta_w \).
      5. if \( N_M > N_M' \), or \( N_M = N_M' \) and \( d_M < d_M' \) then
         6. \( N_M' \leftarrow N_M; d_M' \leftarrow d_M; \delta_B' \leftarrow \delta_B; \delta_p' \leftarrow \delta_p; \)
    7. if \( N_M' < N_e \) then return \( \hat{V}, \hat{E} \) and \( C_E \);
8. foreach adjacent elements \( v_1, v_2 \) in \( \delta_B \) do
    9. \( C_E(e(v_1, v_2)) \leftarrow C_E(e(v_1, v_2)) + 1; \)
    // \( C_E(e(v_1, v_2)) \) denotes the counter for \( e(v_1, v_2) \). If it does not exist, create a new one with initial value 1
10. Rotate and rescale the new path to make the coordinates of the first and last vertices in \( \delta_p \) equal to those of the first and last vertices in \( \delta_B \). Update the coordinates of vertices in \( V_w \) accordingly;
11. Compare \( \delta_w \) and \( \delta_p \), and get the prefix path \( \delta_{pre} \) and the suffix path \( \delta_{suf} \). Reverse \( \delta_{pre} \);
12. \( v_{last} \leftarrow \) the first element in \( \delta_B \);
13. foreach \( i = 1, \ldots, ||\delta_{pre}|| \) do
14. if for vertex \( \delta_{pre}[i] \in V_w, \exists v \in V, \sqrt{(\delta_{pre}[i].x - v.x)^2 + (\delta_{pre}[i].y - v.y)^2} < d_{T1} \) then
15. \( v_{current} \leftarrow v; \)
16. else
17. Create a new vertex \( v_{current} \), where \( v_{current}.x = \delta_{pre}[i].x, v_{current}.y = \delta_{pre}[i].y; \)
18. \( \hat{V} \leftarrow \hat{V} \cup \{v_{current}\}; \)
19. \( \hat{E} \leftarrow \hat{E} \cup \{e(v_{last}, v_{current})\}; \)
20. \( v_{last} \leftarrow v_{current}; \)
21. return \( \hat{V}, \hat{E}, C_E \);

segment (line 5-6). If the largest matching edge number is larger than a constant threshold \( N_e = 3 \), we consider that the path can fit into the base graph (line 7).

To merge the path, we first rotate and rescale the path to make the first and last vertices of its matched segment overlap those of the matched segment in the base graph. The coordinates of all the vertices in \( V_w \) are updated accordingly (line 9). Then we crop the prefix and suffix paths that lie before and after the matched segment in the merged path. We reverse the prefix path, and then treat \( \delta_{pre} \) and \( \delta_{suf} \) as two separated paths that need to be added into the base graph (line 10). The merging process in line 11-20 follows a similar vertex merging method as in Algorithm 6.2. Finally, the algorithm outputs the new base graph \( \hat{V}, \hat{E} \). We conduct Algorithm 6.3 on \( \hat{E} \) to connect the collinear vertices, and then start to process the next path with the new base graph, as shown in Figure 6.17.

If a path fails to be merged to the base graph, we skip it and continue to merge the next path. Once all the paths have been tried, we loop back and try to merge the un-merged
The algorithm details the process of finding the longest matched segment between a base graph and a new path. It involves iterating over potential matches, considering edge length differences, and updating counters for each match. The algorithm aims to find the best matched segment in the new path, which can be used to infer the turning direction from the base graph.

Eliminating Inaccurate Edges: This involves identifying and removing edges that may not accurately represent the current state of the victim. This is crucial for maintaining the integrity of the merged graph and ensuring that the path selected is the most recent location of the victim. The algorithm reduces the impact of inaccurate edges by discarding those with low counter values, thereby improving the accuracy of the merged path.

This process is iterated until no more paths can be merged, ensuring the most recent and accurate representation of the victim’s location.
different paths as the base graph, and check the consistency of the results.

6.9 BUILDING RECOGNIZER AND EVALUATION METRIC DESIGN

We assume that the attacker knows the region that the target building locates in, and the actual corridor maps of all the candidate buildings in that region have been stored in the corridor map pool. After the corridor map has been estimated, a comparison metric is needed to find the best match in the pool, so that the target building can be recognized. Moreover, for the purpose of our evaluation, we also need a metric to quantify the similarity between the estimated graph and the ground truth.

![Figure 6.19: Comparison based on Hamming distance.](image)

Intuitively, we can treat the graph as topological structures, and apply the corresponding mathematical algorithms to evaluate their difference [156]. However, the vertices have the location information, which is lost if we apply the topology-based comparison metric. Moreover, the vertices have relative locations instead of absolute locations, and the directions and scales of different graphs would also be different. Thus, the geographical location-based comparison cannot be directly applied. Our metric should resolve these problems.

A naive metric is based on calculating the Hamming distance between two binary images. One example is shown in Figure 6.19(a), where we aim to compare two triangles. We first divide the images into cells, and a cell is occupied if a line of the triangle goes through it, as illustrated by Figure 6.19(b). Assume that the larger blue triangle is the ground-truth graph and the smaller red triangle is the estimated graph. For each cell occupied by the estimated graph, we calculate the distance to the nearest cell occupied by the ground-truth graph, as in Figure 6.19(c). The cumulative distance is the Hamming distance from the estimated graph to the ground-truth graph. Note that this distance changes if we use the red triangle as the ground-truth graph, i.e., the comparison is asymmetric.

The comparison in ShoesHacker is more complex because the graphs follow different coordinate systems, i.e., they have different directions and coordinate origins. Moreover, since we have no stride length information, the scale of the estimated graph also changes with different \( l_{\text{stride}} \). To solve these problems, we apply a brute-force method for the comparison. We make \( A \) denote the ground-truth graph and \( B \) denote the estimated graph. As shown
in Figure 6.20, we zoom, rotate and move the estimated graph over the ground-truth graph, and calculate the Hamming distance from $B$ to $A$ (denoted by $\text{diff}_A(B)$) in each case. The time cost is acceptable because the graphs has been rasterized. Moreover, to eliminate the impact of the graph size, we divide $\text{diff}_A(B)$ by the number of the cells occupied by $B$. The result is denoted by $\text{diff}'_A(B)$. We record the smallest $\text{diff}'_A(B)$ during this process, and denote it by $\text{minDiff}_A(B)$.

However, the metric $\text{minDiff}_A(B)$ is defective because of two issues. As illustrated by Figure 6.21(a), we can always achieve the lowest Hamming distance by rescaling the estimated graph to a extremely small size, and put the estimated graph on any edge of the ground-truth graph. In Figure 6.21(b), graph $B$ and $B'$ have the same distance to $A$, but obviously $B'$ is more similar to $A$. To handle these issues, during the graph zooming, rotating and moving in the previous comparison, we also exchange $A$ and $B$, and compute $\text{diff}'_B(A)$. In the case shown in Figure 6.21(a), $\text{diff}'_B(A)$ will be huge, and in Figure 6.21(b), $\text{diff}'_B(A) > \text{diff}'_{B'}(A)$. Thus, we use $\text{diff}(A, B) = (\text{diff}'_A(B) + \text{diff}'_B(A))/2$ instead of $\text{diff}'_A(B)$ during the comparison. $\text{diff}(A, B)$ keeps a good balance when we rescale the estimated graph. When the estimated graph is either too small or too large, $\text{diff}(A, B)$ increases. The smallest $\text{diff}(A, B)$ during this process is denoted by $\text{minDiff}(A, B)$.

In ShoesHacker, we use $\text{minDiff}(A, B)$ as the metric to evaluate the similarity between two corridor maps. Note that $A$ and $B$ are non-exchangeable, because only the ground-truth graph $A$ contains the scale information while the estimated graph $B$ does not, and only $B$ is rescaled during the calculation. If we set the cell size as $1 \times 1 \text{m}^2$, the physical significance
of \( \text{minDiff}(A, B) \) is the average distance (in meters) from a location in the estimated graph to the nearest location in the ground-truth graph, when the two graphs have been best overlapped.

To recognize the target building, we compare the estimated corridor map with each map in the corridor map pool using \( \text{minDiff}(A, B) \). The map with the minimum \( \text{minDiff}(A, B) \) is the best match, and the corresponding building is the recognition result. Therefore, ShoesHacker has located the building that the victim walks in.

### 6.10 EVALUATION

In this section, we first evaluate the performance of the stair landing detection, and then measure the accuracy of the angle regression based on the extracted training data. We also show the accuracy of the walking path estimator for individual paths. The overall performance of the corridor map estimator is evaluated in Section 6.10.4. Moreover, in Section 6.10.5 we reveal the danger of the global user location leakage based on the corridor maps generated by ShoesHacker. The data are collected from 6 male and 4 female volunteers with different heights (1.60 - 1.83 m, mean: 1.71 m, median: 1.73 m) and weights (55 - 103 kg, mean: 75 kg, median: 68 kg).

We use the product of ReTiSense Stridalyzer [22] to collect the force data in shoes. Each insole contains eight force sensors, and continuously transmits the data to a cellphone via a Bluetooth connection. Then, the data are uploaded to our laptop host (MacBook Pro with 2.7 GHz Intel Core i7 and 16 GB RAM), which analyzes the data using Matlab scripts.

#### 6.10.1 Performance of Stair Landing Detection

In this subsection, we evaluate the performance of the training data extractor. More specifically, we measure how well the steps made on stair landings are distinguished from normal steps. The quality of the extracted training data will be reflected during the experiment in Section 6.10.2.

During the evaluation, we consider the impacts of shoe type, stair length, and walking speed. We divide the 10 volunteers into two groups. Under the constraint that each group has 3 males and 2 females, group members are randomly assigned. The volunteers in Group 1 wear hard bottom shoes, while those in Group 2 wear sneakers with air cushions. In the first building, each volunteer first keeps doing random exercises, such as jumping, running, or marking time, for 5 minutes. After that the volunteer keeps walking for 2 minutes. Then the volunteer goes from Level 1 to Level 4 using stairs and then goes back. The walking on stairs
is repeated 3 times, each time at different speeds. The three speeds can be roughly classified as slow, medium, and fast, but they are not tightly controlled. Each time the volunteer reaches Level 1 or Level 4, a 30-second walk on flat floor is taken before the volunteer can go back to the stairs. The same test is repeated in the second building. The step number in a flight of stairs is 14 in the first building and 9 in the second one. During the tests, the insoles measure the force changes, and we record the walking with cameras. The steps on stair landings are manually labeled based on the video.

On average, 1149 steps are recorded for each person in each building, among which 121 steps are made on stair landings. Since the number of on-stair landing steps (positive examples) is much less than that of other steps (negative examples), the data are imbalanced. Moreover, for training data collection, we only focus on on-stair landing steps. Thus, we use precision and recall as the metrics to evaluate the performance, which are illustrated in Figure 6.22. The error bars show the maximum and minimum metric values in each test case. The precision ranges in 89.3% - 93.4%, while the recall ranges in 76.9% - 88.6%. Therefore, around 90% of the extracted training data are correct, and in Section 6.10.2 we will show that this is sufficient for the walking direction change estimation. Although the recall is lower, it is still acceptable because there is a trade-off between precision and recall, and we put more weight on precision to keep the purity of the extracted training data. Figure 6.22 also shows that different shoe types do not obviously impact the performance of the training data extractor. The recall in the second building is lower, but the precision is consistent and around 80% of the on-stair landing steps can still be detected.

![Figure 6.22: Precision and recall of stair landing detection.](image1)

![Figure 6.23: Performance of stair landing detection when sensor number changes.](image2)

We further analyze the performance when the sensor number changes. In each insole, we restrict the sensor number to be 8, 6, 4, 2, respectively, and select the sensors that can achieve
the highest average recall over all the test cases. The results are shown in Figure 6.23. For example, if we could deploy only 2 sensors in each shoe, deploying the sensors at the fourth toe and the inner heel can achieve the best recall (39.4%), while the precision is 97.3%. The results imply that it is possible to loose the sensor number requirement without impacting the precision. However, it takes the attacker longer time to collect enough training data, because the recall obviously drops.

6.10.2 Performance of Angle Regression

In this subsection, we evaluate the performance of the angle regression in the walking path estimator. We first examine the distribution of the training data collected in Section 6.10.1. For each volunteer, we merge the data collected during all the test cases and build a personal training dataset.

Figure 6.24 shows the probability distribution of the step number in each stair landing group. In most cases a person takes 3 - 4 steps on a stair landing. Note that the first step on the stair landing has been excluded, as mentioned in Section 6.5.3. The probability distribution of the corresponding labels (in absolute values) is shown in Figure 6.25. Since the labels are calculated by $\frac{180^\circ}{L+1}$, and the step number $L$ is an integer usually ranging in $[1, 7]$, the absolute degree can only be $22.5^\circ$, $25.7^\circ$, $30^\circ$, $36^\circ$, $45^\circ$, $60^\circ$, or $90^\circ$. Apparently, the training data between $60^\circ$ and $90^\circ$ is vacant. However, we will show that the performance of the angle regression is still acceptable in this situation.

![Figure 6.24: Probability distribution of step number in stair landing groups.](image)

![Figure 6.25: Probability distribution of labeled training data. Note that $0^\circ$-steps are not shown since it is not collected during stair landing detection.](image)
Figure 6.26: Probability distribution of angle estimation error.

Figure 6.27: Average absolute angle estimation error when the sensor number changes.

Figure 6.25 also raises the problem of the training data imbalance. While the data between $\pm 25.7^\circ$ and $\pm 60^\circ$ are sufficient, we have only limited data for $\pm 22.5^\circ$ and $\pm 90^\circ$. Thus, we do oversampling on the data of $\pm 22.5^\circ$ and $\pm 90^\circ$ by randomly replicating existing examples, and make sure that each degree has 10 examples of turning steps. As mentioned in Section 6.5.3, the training steps of $0^\circ$ are directly sampled from the continuous walking, and thus their number is always sufficient. In practice, the attacker can wait for a longer time to collect enough training data for each degree. The sampled training data are used to train the angle regression model.

To evaluate the performance of the angle regression, each of the 10 volunteers is asked to walk in free style and make 20 turns. Each turn is $\pm 90^\circ$ and can be made by multiple turning steps. The walking is recorded with cameras, and the direction change made by each turning step is manually measured. Moreover, we also collect 20 straight steps from each volunteer. Note that 5 of the volunteers still wear hard bottom shoes, while the others wear sneakers with air cushions. Totally 676 steps are collected, among which 476 steps are turning steps.

The probability distribution of estimation errors is shown in Figure 6.26. The sign of the errors follows Figure 6.11. The average signed error is $-2.6^\circ$, and the average absolute error is $20.3^\circ$. ShoesLoc [60] also estimates the direction change made by each step, and achieves a lower average absolute error (around $16^\circ$). However, ShoesLoc directly collects the training data from its user, while ShoesHacker extracts the training data during the victim’s daily walking. The inaccuracy in our training data leads to the lower estimation accuracy. However, under the basic assumption that the corridors are latticed, this accuracy is still acceptable, which will be shown in Section 6.10.3.

Furthermore, we analyze the performance when the sensor number changes. We follow
the same sensor selection as shown in Figure 6.23. The results are shown in Figure 6.27. It is shown that, when the sensor number is reduced to 6, i.e., the sensors at the inner metatarsals and the outer heel are removed, the average absolute error is 24.0°, which drops only slightly. Thus, it is possible to further reduce the sensor number. However, as shown in Figure 6.23, the current 8-sensor setting achieves a higher recall rates, which speeds up the training data collection.

6.10.3 Performance of Walking Path Estimator

In this subsection, we show the performance of the walking path estimator when estimating a single walking path. The same volunteer groups with the two shoe types participate in the test, and we directly use the angle regression models trained in Section 6.10.2.

![Figure 6.28: The floor plans of the test regions.](image)

During the test, each volunteer walks along 10 randomly selected paths in each of the three buildings shown in Figure 6.28(a)-(c). The average length of the paths is 78.2 m. At the beginning and the end points of each path, the volunteer is required to intermittently hover in place for 1 minute, as if he/she is moving in rooms. When walking on the given path, the volunteer keeps moving continuously. The walking speed is not controlled, and the average speed varies around 1 - 3 m/s.

Based on the force data, the continuous walking detector detects the start and end of each continuous walk, and the walking path estimator outputs the estimated paths. We compare each estimated path with the ground truth using the metric minDiff(A, B), and Figure 6.29 illustrates some examples. The probability distribution of the results is shown
Figure 6.29: Examples of path estimation. The estimated path has been rotated and rescaled to fit the real path during the calculation of \( \text{minDiff}(A, B) \).

in Figure 6.30. The average \( \text{minDiff}(A, B) \) is 1.5 m, i.e., when the estimated and the real paths are best overlapped, the average distance from a location on the estimated path to the nearest location on the real path is around 1.5 m. In Figure 6.30, the errors around 4 - 11 m are caused by the errors of the turning action detection. \( \text{minDiff}(A, B) \) increases significantly once a turn is missed or wrongly recognized. However, as shown by the figure, more than 84.3% of the paths can be estimated with \( \text{minDiff}(A, B) < 4 \) m. Moreover, the wrong estimated path can hardly fit into the base graph, and will be eventually removed based on the low \( C_E \) counter value. Thus, the walking path estimator provides a good basis for the corridor map estimator, as will be shown in Section 6.10.4.

6.10.4 Performance of Corridor Map Estimator

To evaluate the overall performance of the corridor map estimation, each of the 10 volunteers is required to walk in two of the five indoor spaces, as shown in Figure 6.28. As in the previous experiments, 5 volunteers wear hard bottom shoes, and the others wear sneakers with air cushions. In each floor, the volunteer walks for 30 minutes. During the first 15 minutes, the volunteer randomly traverse in the corridors multiple times. The speed is roughly stable and not controlled. We use cameras to record the walking path. After that, we check if all the corridors have been gone through. If not, the volunteer is required to go along a path that goes through the missing corridors. In the following 15 minutes, the same process is repeated, but the volunteer consciously walks at the non-uniform speed, i.e., the volunteer keeps changing the walking speed during the test. During the tests, the volunteers are allowed to take breaks, and the break time is excluded from the test time. The stoppages can be easily detected by the continuous walking detector. We continue using the trained models in Section 6.10.2.
For each 15-minute walk, we uniformly divided the walk into 5 parts in the time domain, and treat each part as a single continuous walk. Thus, 40 test cases are collected from the five indoor spaces, and each test case contains 5 walks. For the data of each test case, we run ShoesHacker to estimate the corridor map. Five examples are shown in Figure 6.31. Some adjacent intersections are not correctly merged because the distance between them exceeds the threshold \( d_{T1} \). However, the rough shapes of the corridor maps are still correct.

Since our comparison metric \( \text{minDiff}(A, B) \) is based on the shapes of maps instead of their graph structure, the comparison between the estimated maps and the ground truth is not highly impacted, and the attacker still has a good chance to recognize the building, as will be shown in Section 6.10.5.

We compare each estimated map with the ground truth using \( \text{minDiff}(A, B) \), and the results are shown in Figure 6.32. The average \( \text{minDiff}(A, B) \) is 6.1 m. The large errors of some estimation results are caused by the wrong turning direction estimation. If the estimated path contains wrong turns, it can hardly be merged to the base graph. As shown in Figure 6.31(b), some corridors are missing in the estimated map because the walking paths going through them failed to be merged. The wrong turning detection would also add corridors that do not exist. One example is shown in Figure 6.31(c). The corridors marked by the blue dashed line are wrongly generated, because the left turn made by the volunteer at point A is not correctly detected. In the real case, it is possible to further avoid these errors based on the counter \( C_E \). The attacker can keep monitoring the daily data from the victim, and remove the corridors that are rarely visited.

We further evaluate the impacts of shoe types and walking speed changes. In Figure
Figure 6.32: Probability distribution of \( \min \text{DiFF}(A,B) \) between estimated maps and real maps.

Figure 6.33: Impacts of different shoe types and changing speed.

6.33, the test results are classified, and each error bar presents the second largest/smallest \( \min \text{DiFF}(A,B) \) in each test class. It is shown that the impact of different shoe types is not obvious. However, the changing walking speed increases the estimation error. This is because the changing speed causes the stride length changes, which can hardly be estimated by the attacker, and we assume that the stride length is constant during walking.

As mentioned before, if the force data are leaked to the attacker in real time, the attacker can locate the victim in the building. After the tests, we overlap each estimated map with the real floor map based on the match result that we got when computing \( \min \text{Diff}(A,B) \). On each of the estimated paths that compose the estimated map, we select all the turning points as the sampling points to estimate the localization error. Based on the video record, we find the real locations of the volunteer when he/she is walking on this path. Then we calculate the distance between the sampled turning points and the real locations. The cumulative distribution function (CDF) of the localization errors is given in Figure 6.34. The average error is 5.4 m, which shows that if the attacker has determined the building (as will be shown in Section 6.10.5), the victim could be further localized. Moreover, as long as the victim’s current walking path has been merged to the base graph, the average time cost of updating the victim’s location for each future footstep is 98.6 ms. Thus, if the force data for each footstep are leaked to the attacker in real time, the victim can be located with low time latency.
6.10.5 Performance of Building Recognizer

In this subsection, we aim to show how likely the attacker can recognize the building based on the output of the corridor map estimator. We randomly select 35 floor plans from a map dataset of a university campus [157]. The corridors in each floor plan are manually labeled. Together with the five floors shown in Figure 6.28, we build a corridor map pool that contains 40 corridor maps for the recognition test.

For each of the 40 estimated maps from the five buildings in Section 6.10.4, we compare it with each of the 40 corridor maps in the pool using minDiff\((A, B)\), and sort the results. The corridor map with the smallest minDiff\((A, B)\) is the best match. In Figure 6.35, we plot the Cumulative Matching Characteristics (CMC) curve, which shows how the probability of correct recognition (y-axis) changes with the number of the candidates output by ShoesHacker (x-axis), i.e., the rank-k accuracy represents the probability that the correct recognition result is included in the top-k best matches [158].

As shown in Figure 6.35, the rank-1 accuracy is 77.5\% for the five buildings. One typical recognition error is shown in Figure 6.36. The result implies that, if the victim is known
to be on the campus of 40 buildings, with the probability around 78%, the attacker can correctly guess the building that the victim lives or works in. The accuracy can be higher if more candidates are allowed, e.g., the rank-10 accuracy reaches 95.0%. Moreover, in Section 6.10.4 we have shown that once the building has been determined, the attacker can even locate the victim with the average error lower than 6 m, if the force data are leaked in real time.

Therefore, ShoesHacker reveals the danger of the location leakage through the force data from smart shoes.

6.11 DISCUSSION

6.11.1 Limitation of ShoesHacker

Atypical staircases: Currently ShoesHacker only handles the typical staircases as shown in Figure 6.4, which are common in office buildings. In other public places such as shopping malls, there would also be atypical staircases such as spiral stairs. However, as long as the stair landings exist and the stair number between adjacent floors is consistent, ShoesHacker can still detect the steps on stair landings. The limitation is that, if the stair landings do not follow the U-turn type, the direction labels of the extracted training data would be incorrect. We leave it to our future work to design a method that also works for atypical staircases.

Symmetry in floor plans: It is possible that the structure of a floor is completely symmetric, which makes it hard to correctly merge the paths. One possible solution is making use of landmarks to pin the paths, which is similar to the idea in [146, 124, 148]. For example, victims must go through entrances to enter buildings, and reach new floors using elevators or stairs (already handled by ShoesHacker). By recognizing the landmarks on walking paths, the symmetric floor structure could be handled. In our work, we only focus on the general case that no landmark is found, and leave it to our future work to further utilize the landmarks.

Irregular corridor structures: As mentioned previously, the basic assumption of our attack method is that the corridors are latticed. Although most intersections have only 90° turns and most corridors are straight, some of them could have more complex shapes. Due to the estimation error in the angle regression, the error of the walking path estimator will increase if our basic assumption does not hold. One possible solution is that, we first use ShoesHacker to roughly find a group of candidate buildings that have small minDiff(A, B) values. Then, for each candidate, we assume that its floor map is the ground truth, and apply the particle filter-based localization approaches such as ShoesLoc [60] on it. If the candidate floor is the
correct guess, the particles should be able to converge stably, otherwise they keep diffusing. We will continue to investigate how to handle the irregular corridor structures.

Multiple target buildings: Currently, for the attack scenario of office buildings, we assume that the victim works and walks in a single building, which is true for many occupations. However, it is also possible that the victim daily walks in multiple buildings. With the stair landing detection, we can detect the action of walking on stairs, and thus roughly estimate the moment when the victim enters or leaves a building if elevators are not used. However, it is still difficult to distinguish different buildings. One possible solution is counting the number of floor levels that the victim has gone through in each building. If the floor levels of two buildings differ, we can still distinguish them. In the future research, we can investigate other possible ways to handle the case of multiple target buildings.

6.11.2 Extending ShoesHacker

Improving localization accuracy: If the attacker has correctly determined the building, based on our test the average error of the victim localization is 5.4 m. It is possible to further improve the localization accuracy in this case. Since the floor map has been known, we can determine the length of the corridors, and thus measure the average stride length. This information can be used to train a stride length estimation model, based on which ShoesLoc [60] can be applied. The accuracy of ShoesLoc is around 1 m, which makes the room-level localization possible. Moreover, while we use some constant thresholds such as $d_{T1}$, they can be further fine-tuned based on the determined floor map. For example, $d_{T1}$ can be reduced if multiple intersections are found merged by mistake when compared with the floor map. This helps increase the accuracy of the future attacks on the victim.

Crowdsensing: Waiting for one victim to traverse the whole floor could take a long time. If the attacker has accesses to multiple victims’ data and these victims are known to work in the same building, our method can merge the estimated paths from different victims. This reduces the time until the whole corridor map is reconstructed.

Guessing floor plans: As an intermediate product, the estimated corridor map can already leak important information. For example, if the workplace of the victim is already known, but the indoor structure of the building is unknown, the attacker can guess the floor plan based on the corridor map and the building outline, which is especially critical for confidential departments.

Exploring other information leakages: Besides the victim’s position, other privacy information could also leak from the force data. For example, around the noontime, if the force data from a victim show no footstep, it is likely that the victim is having lunch. If the non-
walking time of a victim perfectly fits into a class schedule, the victim could be a student. Therefore, it is possible to infer victims’ living habits and occupations based on foot force data. We leave it to our future work to further explore other information leakages from smart shoes.

6.11.3 Defending Against ShoesHacker

One intuitive way to defend against the attack is protecting the force data from exposure. To narrow the attack surface, an effective solution is processing force data locally and avoiding uploading raw data to the cloud. Typically, smart shoes are connected to a user’s cellphone via Bluetooth. The cellphone can preprocess the force data before uploading them to the cloud. This prevents the walking path estimation, as long as the Bluetooth connection and the user’s device are well protected.

6.12 COMPARISON BETWEEN SHOESLOC AND SHOESHACKER

The performance of ShoesLoc and ShoesHacker cannot be directly compared, because they are designed based on different assumptions. As shown in Figure 6.37, ShoesLoc assumes that the floor plan is known, the training data are available, and it has no strong requirement on the corridor structure. On the other hand, ShoesHacker assumes that the corridors are latticed, but does not need the floor plan of the target building (the floor plans of candidate

Figure 6.37: Assumptions of ShoesLoc and ShoesHacker.
buildings are still needed), and requires no training data from the victim. Thus, as illustrated in Figure 6.37, the basics of ShoesLoc and ShoesHacker are opposite.

There are some similar subcomponents in ShoesLoc and ShoesHacker. For example, both of them use SVM regression models to estimate the relative direction change made by each footstep. However, since ShoesHacker extracts training data by itself, the training data contain errors. The average angle estimation error increases from 16.04$^{\circ}$ to 20.3$^{\circ}$, even if more sensors have been used by ShoesHacker. Similarly, while ShoesLoc estimates the stride length of each footstep, ShoesHacker does not get this information due to the lack of training data. Because of these differences, most of the components of ShoesLoc and ShoesHacker are uniquely designed and cannot be applied to each other.

6.13 CONCLUSION

We present ShoesHacker, an attack scheme that explores the possibility of reconstructing corridor maps and locating victims based on the force sensors in smart shoes. We propose the stair landing detection algorithm to extract training data during the victim’s daily walking, and design the path merging algorithm to merge the estimated walking trajectories. Moreover, we design a metric that can evaluate the similarity between two corridor maps. ShoesHacker shows that it is possible for an attacker to locate the victim if the force data leakage happens. The experimental results show that, in a dataset that contains 40 buildings, the attacker can recognize the correct building with an accuracy around 78%, and can even further locate the victim with an accuracy better than 6 m. Thus, ShoesHacker reveals the danger of the location privacy leakage through the foot force data.
CHAPTER 7: CONCLUSIONS AND FUTURE WORK

To conclude the thesis, I first summarize the research findings, and then discuss some future research directions.

7.1 SUMMARY

In this thesis, we propose the MSAA framework, which describes the connection between the two faces of mobile sensing. MSAA shows how we design an effective mobile sensing system by exploring different sensors, various requirements for user/environment contexts, and different sensing algorithms. It also shows the way to explore potential user privacy leakages, i.e., breaking a system into basic components, and for each component considering the possible privacy exposure if data are leaked. Moreover, our framework can be a research guideline that helps researchers find and close the gaps in the mobile sensing research.

Then, following the MSAA framework, I use four mobile sensing designs as examples to show how we can propose effective sensing techniques, and how we can explore the corresponding threats that can be caused by these techniques. On one hand, TableWrite and ShoesLoc are new designs that benefit people’s lives. TableWrite provides convenient handwriting input method for tiny mobile devices, and ShoesLoc provides a new indoor tracking method for smart shoes. On the other hand, WritingHacker and ShoesHacker reveal the threat of privacy leakage if the sensing techniques are maliciously used. WritingHacker shows that it is possible to recognize the handwriting content if the victim’s handwriting sound is recorded by the attacker. ShoesHacker reveals the possibility of locating the victim if the force data from smart shoes are leaked. These confirm our statement, i.e., every mobile sensing design should be considered with respect to its benefits and its potential threats, which is reflected in the MSAA framework. The rapid evolution of mobile devices amplifies the need for new sensing techniques, while ignoring the information leakage will expose users to unpredictable danger.

7.1.1 Mobile Sensing Application-Attack Framework

In Chapter 2, I present the MSAA framework, a general model showing the structures of mobile sensing applications and attacks. MSAA provides a guideline on how to design effective mobile sensing systems, i.e., exploring the defects of the previous approaches, and overcoming the defects by trying different sensors, context requirements and algorithms. In
my thesis, TableWrite shows how we reduce the sensor cost of handwriting input by using microphones as sensors and utilizing the contextual information of users’ handwriting styles. ShoesLoc shows how we improve the accuracy and robustness of indoor walking path tracking by using new force sensors and utilizing the floor map context. On the other hand, MSAA also describes the way mobile sensing applications and attacks are connected. If not well protected, the user information can leak through the physical signal leakage, the raw data leakage, the indirect privacy leakage, and the direct privacy leakage. Based on this structure, MSAA provides a guideline that helps explore the possible information leakages that can be brought by a mobile sensing technique. In my thesis, WritingHacker considers the case where the handwriting signal is leaked through the physical signal leakage, and ShoesHacker shows the danger if the foot force signal is leaked through the raw data leakage. WritingHacker and ShoesHacker reveal the threats caused by the possible leakages in TableWrite and ShoesLoc, which matches the structure of the MSAA framework.

7.1.2 TableWrite: Audio based Handwriting Input for Tiny Mobile Devices

To address the problem that it is hard to efficiently input messages on tiny mobile devices, in Chapter 3, I present TableWrite, an audio-based handwriting input scheme. TableWrite allows users to input words to mobile devices by writing on tables with fingers. The design is mainly based on the contextual information of the user’s writing habit, i.e., the stroke number of each letter. The key feature is that, once trained by a user, TableWrite does not require any retraining phase before each use. To reduce the impacts of audio signal’s multipath propagation, we design multiple features that maintain consistency even when writing positions keep changing. We apply machine learning and gesture tracking techniques to further improve the accuracy of handwriting recognition. The word recognition accuracy of 90%-95% is achieved, which validates the effectiveness of the new sensing technique.

7.1.3 WritingHacker: Audio based Eavesdropping of Handwriting via Mobile Devices

In Chapter 4, we investigate the possibility of the handwriting content leakage if the technique of TableWrite is maliciously used. By presenting a proof-of-concept system, WritingHacker, I show how the attackers could eavesdrop on and recognize handwriting content via nearby mobile devices. To reduce the impacts of various writing habits and writing locations, the system utilizes the methods of letter clustering, dictionary filtering and letter time length based offsetting. Moreover, if the relative position between the device and the handwriting is known, the hand motion tracking method can further enhance the system’s
performance. The experimental results show that the accuracy of word recognition can reach 70%-80%, which helps raise the public’s concern about privacy leakage through the handwriting sound.

7.1.4 ShoesLoc: In-Shoe Force Sensor-Based Indoor Walking Path Tracking

To show that smart shoes are capable of tracking users’ walking path, in Chapter 5, I present ShoesLoc, an indoor walking path tracking method based on in-shoe force sensors. Based on the force signals from a user’s shoes, it is possible to estimate the walking direction change and the stride length of each step with machine learning techniques. I apply a particle filter to combine this information with the environment context of the barriers labeled on floor maps, and thus can determine the walking path and the current position of the user. To solve the problem of the low accuracy caused by cumulative walking direction errors, I improve the particle filter by designing the direction correction algorithm. I also propose the weight normalization method to handle the impact of handbags and backpacks. The experimental results show that, after a convergence phase, ShoesLoc achieves the average location error of 0.9-1.3 m. Moreover, it requires no wireless anchor or extra site survey, and has good robustness to interferences. Thus, ShoesLoc achieves better effectiveness compared with the previous approaches.

7.1.5 ShoesHacker: Indoor Corridor Map and User Location Leakage through Force Sensors in Smart Shoes

In Chapter 6, I further investigate the attacker’s capability of locating the victim if the force data from the victim’s smart shoes are leaked. I present ShoesHacker, an attack scheme that reconstructs the corridor map of the building that the victim walks in based on force data only. The corridor map enables the attacker to recognize the building, and thus locate the victim on a global map. To handle the lack of training data, we design the stair landing detection algorithm, based on which we extract training data when victims are walking in stairwells. ShoesHacker estimates the trajectory of each walk, and applies the path merging algorithm to merge the trajectories. Moreover, I design a metric to quantify the similarity between corridor maps, which makes building recognition possible. The building recognition accuracy reaches 77.5% in a 40-building dataset, and the victim can be located with an average error lower than 6 m, which reveals the danger of privacy leakage when the force data from smart shoes are not protected.
7.2 FUTURE RESEARCH DIRECTIONS

While technologies keep evolving, the sensors and the computing power of mobile devices also keep growing, which will bring researchers great opportunities and also challenges. For the approaches introduced in this thesis, there are still aspects waiting for further exploration.

**Generalizing and extending MSAA framework:** The current MSAA framework only focuses on the information leakages in mobile sensing applications. However, as a guideline, MSAA can also be generalized for other fields. For example, inspired by MSAA, we can find multiple possible information leakages in the security system of a company. The lighting on/off status directly shows whether an office building is empty (physical signal leakage). The running speed of an electricity meter could reflect if any high-power appliances are running (raw data leakage). The water bill of the office building could indirectly show the number of employees working in the building (indirect privacy leakage). Therefore, MSAA can be generalized as a guideline that helps examine the possible sources of information leakages. Moreover, while MSAA mainly considers the information leakages, we can further extend it to handle the injection attacks. For instance, if the software of TableWrite is hacked by the attacker, the attacker can inject fake handwriting sounds to the word detector, or inject fake features to the word recognizer, and thus disturbs the input method or generates malicious input content. These injection points can be added to the MSAA framework in the future, so that it can cover more attack threats.

**Training general models with deep learning:** Currently, deep learning has been widely used in many research fields, such as image-based object recognition [159, 160, 161], social network analysis [162, 163], and sensor-based human activity recognition [164, 165]. It is also promising to apply deep learning to our current approaches for better performance. For instance, in TableWrite and ShoesLoc, we can train a general handwriting/walking model that works for every user, so that we can eliminate the training phase. However, there are several challenges to address. First, due to the limited computing power and energy of tiny mobile devices, it is expensive to run neural network classifier locally. One possible solution is uploading the data to the cloud and running the classifier on cloud servers. However, this requires network connection, which is inconvenient and less practical, especially for TableWrite. Second, as mentioned in Section 5.10.4, we also need to address the issue caused by the device diversity. For instance, currently there is no industrial standard for the force sensors in smart shoes, and different smart shoes from the same manufacturer can even have different outputs for the same use case. Due to the low dimension number of force data, the impacts of the device diversity could seriously impact the performance of the general model. It is valuable to further address these challenges.
Defending against attacks: While we have revealed the possible threats caused by the malicious use of the new techniques, it is still an open question to defend against the threats. In Section 4.10.3 and 6.11.3, we have shown some defense methods against WritingHacker and ShoesHacker. However, as more and more sensors are embedded in new mobile devices, it is harder to prevent side-channel attacks. For instance, if smart shoes are equipped with accelerometers and compasses, it is possible to locate the victim based on motion data instead, and the algorithm of ShoesHacker is still applicable to the attack. Therefore, investigating the capability of the sensors in different mobile devices is always valuable, and is worth researchers’ tireless exploration.

Exploring sensor fusion-based applications and attacks: It is possible to further improve the effectiveness of our applications if the constraint on sensor cost is loose. While ShoesLoc requires the floor map as an input, we can eliminate this requirement by applying sensor fusion. For example, foot motion sensors can be used to reconstruct the floor map [149]. In this case, the output of the motion sensor can be seen as a new contextual input to ShoesLoc. For the attack scenario, although WritingHacker has used accelerometers to handle the near-field burst noise, in this thesis we mainly focus on the attacks that use a single sensor type. The assumption that the device is equipped with multiple types of sensors can shrink the attack scope, and requiring multiple data leakage sources reduces the attack success rate. However, if the device limitation is loose, sensor fusion can help handle more complex attack scenarios. For example, while ShoesHacker uses force sensors for victim tracking, the motion sensors can also help track the victim [61] if the magnetic field interference is negligible. It is possible that the fusion of force and motion sensors can further improve the tracking accuracy. Exploring the benefits that sensor fusion can bring to our applications and attacks is left for our future work.
REFERENCES


[146] M. Alzantot and M. Youssef, “Crowdinside: automatic construction of indoor floor-


[156] P. Van Mieghem, “Can the topology of two graphs be compared by one number?” *Delft University of Technology*, p. 72, 2013.


