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

TUO YU and KLARA NAHRSTEDT, University of Illinois at Urbana-Champaign

The past few years have witnessed the rise of smart shoes, the wearable devices that measure foot force or track foot motion. However, people are not aware of the possible privacy leakage from in-shoe force sensors. In this paper, we explore the possibility of locating an indoor victim based on the force signals leaked from smart shoes. We 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. We estimate the trajectory of each walk, and propose the path merging algorithm to merge the trajectories. Moreover, we design a metric to quantify the similarity between corridor maps, which makes building recognition possible. Our experimental results show that, 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 through smart shoes.

CCS Concepts: • **Information systems** → **Mobile information processing systems**; *Location based services*; • **Human-centered computing** → **Mobile devices**; *Ubiquitous and mobile computing systems and tools*; • **Security and privacy** → Domain-specific security and privacy architectures .

Additional Key Words and Phrases: Corridor map reconstruction, force sensors, smart shoes

ACM Reference Format:

Tuo Yu and Klara Nahrstedt. 2019. ShoesHacker: Indoor Corridor Map and User Location Leakage through Force Sensors in Smart Shoes. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 3, 3, Article 120 (September 2019), 29 pages. <https://doi.org/10.1145/3351278>

1 INTRODUCTION

The recent popularization of wearable devices has been impacting people's lives. For instance, many smartwatches can measure heart rate [36], and smart shoes can count footsteps or detect users' fatigue levels [4, 24, 40, 54]. However, besides the benefits brought by the new devices, people should also be aware of the danger caused by measurement data leakages. 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 [45], or the passwords of turntable password locks [30]. 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 [40] and Zhor-Tech [54] 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 paper, 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.

Authors' address: Tuo Yu, tuoyu2@illinois.edu; Klara Nahrstedt, klara@illinois.edu, University of Illinois at Urbana-Champaign, Urbana, Illinois, 61801.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2019 Association for Computing Machinery.

2474-9567/2019/9-ART120 \$15.00

<https://doi.org/10.1145/3351278>

Proc. ACM Interact. Mob. Wearable Ubiquitous Technol., Vol. 3, No. 3, Article 120. Publication date: September 2019.

Some previous works have shown the capability of smart shoes for indoor walking path tracking [27, 28, 48], among which ShoesLoc [51] 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 paper, 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 [16], 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 paper 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 2, and present a system overview in Section 3. Then we successively introduce the methods used by ShoesHacker in Section 4 - 9. In Section 10, we evaluate the performance of ShoesHacker with experiments. We have further discussion in Section 11, and conclude our paper in Section 12.

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 [13], speakers [11], or cameras [29]. 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 [45] and keypads [31], or guess the password of a turntable password lock [30]. 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 [9].

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 [12, 14]. While researchers and the industry have designed and adopted many location protection approaches to protect users' geographical locations [19, 22], users' location traces can still be inferred via multiple channels such as social networks [26, 41], call records [52], and people-nearby services [47].

It has also been shown that cellphones are vulnerable to side-channel localization attacks. For instance, Powerspy [32] 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 [20] reveals that the signal measured by the accelerometer of a cellphone can be used to infer the driving trajectory of a vehicle. In [55], 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 paper, 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 [3, 42, 49, 53]. 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 [2, 6, 21], but can also be used to locate victims once the data are leaked [25]. ShoesLoc [51] 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 paper, 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 [27, 28, 48]. 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 [10, 15], there are also works that utilize acoustic signals [38], laser [43], or inertial sensors plus landmarks [1, 5, 23]. For instance, CrowdInside [5] 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 [37], 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.

3 SYSTEM OVERVIEW

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

3.1 Threat Model

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° . 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 [16] provide indoor floor maps.

We assume that the smart shoes are equipped with a sufficient number of force sensors, which will be shown in Section 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.

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 [40] is shown in Fig. 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.

Fig. 1. Force sensor deployment for each insole.

In this paper, 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 $i = 1, \dots, 8$ 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 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 $\{u_i^l(t), u_i^r(t) | i = 1, \dots, 8, t \geq 0\}$, where $u_i^l(t)$ is the data sequence received from sensor i in the left insole, and $u_i^r(t)$ is from sensor i in the right insole. Although $u_i^l(t)$ and $u_i^r(t)$ are discrete time series, we write them as continuous time series for convenience, which does not impact our following analysis. In Section 10.2, we show that it is possible to conduct the attack with less sensors.

3.3 System Architecture

As shown in Fig. 2, our system mainly consists of the following components.

Fig. 2. System architecture of ShoesHacker.

Step detector: It receives the raw signals $\{u_i^l(t), u_i^r(t)\}$ 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.

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.

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.

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.

4 STEP DETECTOR

This component aims to crop the force data sequences for each footstep. We treat $\{u_i^l(t) | i = 1, \dots, 8\}$ and $\{u_i^r(t) | i = 1, \dots, 8\}$ respectively. Similar to the method used in [51], we first calculate the total foot force on the insoles, i.e., $u^l(t) = \sum_{i=1}^8 u_i^l(t)$ and $u^r(t) = \sum_{i=1}^8 u_i^r(t)$, as shown by the last subgraph in Fig. 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 γ_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 Fig. 3.

For each detected step, we crop the $u_i^l(t)$ (or $u_i^r(t)$) between its start and end points. The cropped step segments are denoted by $s_{i,k}^l(\tau)$ (or $s_{i,k}^r(\tau)$), where $i = 1, \dots, 8$, and $k = 1, 2, \dots$ is the index for each left (or right) step. Note that the left and right steps are indexed independently, and $s_{i,k}^l(\tau)$ is contiguous to $s_{i,k}^r(\tau)$. Moreover, τ 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 Fig. 3.

As a result, for the k th left or right step, the step detector outputs the force data as $s_{i,k}^l(\tau)$ or $s_{i,k}^r(\tau)$, $i = 1, \dots, 8$, 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.

Fig. 3. Data from force sensors while a person is walking.

Fig. 4. A walk from Level 1 to Level 3.

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.

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 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 [51] 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 Fig. 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.

5.2 Stair Landing Detection

There have been approaches that detect the action of going on stairs using inertial sensors [7, 46]. 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.

Fig. 5. Examples showing the signals when the victim is going up or down stairs. The signals come from the right foot.

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 Fig. 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 Fig. 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 Fig. 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 Fig. 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 Fig. 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 Fig. 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 Fig. 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 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 $s_k(\tau)$, $k = 1, 2, \dots, K$ denote the K continuous step segments from one sensor, i.e., $s_{i,k}^l(\tau)$ or $s_{i,k}^r(\tau)$, $k = 1, 2, \dots, K$. Note that Steps 1-5 in the algorithm do not refer to the physical footsteps made by the victim.

(Step 1) Signal conversion: Fig. 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 Fig. 6(a) is shown in Fig. 6(b). In the following steps we process $p[k]$ instead of $s_k(\tau)$.

Fig. 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 [17].

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 Fig. 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 Fig. 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 Π . Note that the boundary condition checking for $r[k_1 - 1]$ and $r[k_2 - W + 2]$ in line 9 is 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_1^g, k_2^g) , $g = 1, \dots, |\Pi|$.

(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

ALGORITHM 1: Turning step detection based on a single sensor

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

// (Step 1) Signal conversion
1 $p[k] \leftarrow \max_{\tau} s_k(\tau)$, $k = 1, 2, \dots, K$;
// (Step 2) Rough stair detection
2 $r[k] \leftarrow 0$, $k = 1, 2, \dots, K$;
3 **foreach** $m = 1, 2, \dots, K - W + 1$ **do**
4 $\tilde{p}[k] \leftarrow p[k] - \frac{1}{W} \sum_{k=m}^{m+W-1} p[k]$, $k = m, \dots, m + W - 1$;
5 Get Hanning window $h = \text{hanning}(W)$;
6 $\{A[n] | n = 1, \dots, W\} \leftarrow \frac{1}{W} \|FFT(\{\tilde{p}[k] \cdot h[k - m + 1] | k = m, \dots, m + W - 1\})\|$;
7 **if** peaks exist in $\{A[n]\}$ with altitude larger than A_T **then** $r[m] \leftarrow$ the bin index of the first peak with altitude larger than A_T ;
8 $\Pi \leftarrow \emptyset$;
9 **foreach** (k_1, k_2) that satisfies $k_2 - k_1 > \Gamma$, $r[k] > 0$, $k = k_1, \dots, k_2 - W + 1$ 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 Π
10 **foreach** $(k_1^g, k_2^g) \in \Pi$, $g = 1, \dots, |\Pi|$ **do**
// (Step 3) Period Estimation
11 $\Psi \leftarrow \emptyset$;
12 $k'_1 \leftarrow \infty$; $k'_2 \leftarrow -\infty$;
13 **foreach** $m = k_1^g, \dots, k_2^g - W + 1$ **do**
14 $\psi \leftarrow \lfloor W / (r[m] - 1) - 0.5 \rfloor$;
15 $\sigma \leftarrow \sqrt{\frac{1}{W} \sum_{k=m}^{m+W-1} (p[k] - \frac{1}{W} \sum_{k=m}^{m+W-1} p[k])^2}$;
16 **if** $\psi \geq \psi_T$ and $\sigma \geq \sigma_{T1}$ **then**
17 $N_\psi \leftarrow \lfloor W / \psi \rfloor$;
18 Break $\{p[k] | k = m, \dots, m + \psi N_\psi - 1\}$ into vectors v_j , $j = 1, \dots, N_\psi$, each with length ψ ;
19 $\sigma' \leftarrow \sqrt{\frac{1}{\psi N_\psi} \sum_{j=1}^{N_\psi} \|v_j - \frac{1}{N_\psi} \sum_{j=1}^{N_\psi} v_j\|^2}$;
20 **if** $\sigma' \leq \sigma_{T2}$ **then**
21 $\Psi \leftarrow \Psi \cup \{\psi\}$;
22 **if** $m < k'_1$ **then** $k'_1 \leftarrow m$;
23 **if** $m + W - 1 > k'_2$ **then** $k'_2 \leftarrow m + W - 1$;
24 $k_1^g \leftarrow k'_1$; $k_2^g \leftarrow k'_2$;
25 $\tilde{\psi}_g \leftarrow \text{mode}(\Psi)$;
// (Step 4) Turning Step Extraction
26 Find all peaks in $p[k]$ (or $-p[k]$), $k = k_1^g, \dots, k_2^g$ that meet the following conditions:
27 (1) The peaks are separated by more than the minimum peak distance $\tilde{\psi}_g - \psi_d$;
28 (2) The peak height is larger than $\bar{p}$ (or $-\bar{p}$), where $\bar{p} = \frac{1}{k_2^g - k_1^g + 1} \sum_{k=k_1^g}^{k_2^g} p[k]$;
29 Denote the average peak height by $\hat{p}$ (or $\check{p}$). Denote the indexes of the peaks by set U_1 (or U_2);
30 **if** $|\hat{p} - \bar{p}| > |-\check{p} - \bar{p}|$ **then** $U \leftarrow U_1$; **else** $U \leftarrow U_2$;
31 $\tilde{U}_g \leftarrow \emptyset$;
32 **foreach** $k \in U$ **do**
33 **foreach** $k' = k - 4, \dots, k + 4$ **do**
34 **if** $|p[k'] - p[k]| < p[k] r_p$ **then** $\tilde{U}_g \leftarrow \tilde{U}_g \cup \{k'\}$;
35 **return** Π , $\tilde{U}_g$ and $\tilde{\psi}_g$, $g = 1, \dots, |\Pi|$;

Fig. 7. Example of FFT results.

Fig. 8. Example of breaking the signal in window into vectors.

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_1^g, k_2^g) , we apply a new sliding window of size W . In line 14-15, we calculate the signal period ψ based on the FFT result in Step 2, as well as the standard deviation σ of the signal in the window. The results are discarded if the ψ is shorter than ψ_T , or σ is smaller than σ_{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 ψ , and treat them as vectors. The trailing signal is discarded if the window length is not the integral multiple of ψ . One example is shown in Fig. 8. Then, we calculate the standard deviation σ' 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 ψ in set Ψ . After the sliding window has gone through the stair frame, we use the most common element in Ψ 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 Fig. 6(b). It is shown that this method fine-tunes the start and end points of the on-stair walk.

Fig. 9. Two types of peak amplitude signals ($p[k]$) during a walk in a stairwell.

(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 Fig. 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 (Fig. 9(a)). However, this is inverse for the second volunteer (Fig. 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 Fig. 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.

Fig. 10. Examples of stair landing groups.

Fig. 11. Direction format of angle regression. Examples of left and right 45° turns made by a left foot are given.

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 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 $\tilde{U}_{i,g}^l, g = 1, \dots, |\Pi_i^l|, i = 1, \dots, 8$ and $\tilde{U}_{i,g}^r, g = 1, \dots, |\Pi_i^r|, i = 1, \dots, 8$. We aggregate all the turning steps by

$$\hat{U}^l = \bigcup_{i=1}^8 \bigcup_{g=1}^{|\Pi_i^l|} \tilde{U}_{i,g}^l, \quad \hat{U}^r = \bigcup_{i=1}^8 \bigcup_{g=1}^{|\Pi_i^r|} \tilde{U}_{i,g}^r. \quad (1)$$

$\hat{U}^l$ 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 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 10.1.

Since Algorithm 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 $\tilde{\psi}_g = 10$, 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.

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 Fig. 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 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 $(0^\circ, 90^\circ]$ and the turning direction (“left” and “right”). For the convenience of expression, we reformat the two labels following the format in Fig. 11. Thus, the label attached to a turning steps is a value within $[-90^\circ, 0^\circ) \cup (0^\circ, 90^\circ]$. 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, 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 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 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 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 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

$$\tilde{u}^l(t) = u^l(t) - \frac{1}{W'} \sum_{t=t'}^{t'+W'-1} u^l(t), \quad t = t', \dots, t' + W' - 1. \quad (2)$$

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

$$\{A^l[n] | n = 1, \dots, W'\} = \frac{1}{W'} \left\| \text{FFT}(\{\tilde{u}^l(t) \cdot h'[t - t' + 1] | t = t', \dots, t' + W' - 1\}) \right\|, \quad (3)$$

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 $t_2 - t_1 \geq 10$ s, we regard the signals in $t = t_1, \dots, t_2$ 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 $u^l(t)$. 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.

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° , 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.

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 Fig. 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 [51], 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 Fig. 12, typically a person makes multiple steps to make a 90° turn, and each step makes a direction change smaller than 90° . 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 [51]. Moreover, based on the sequence of the direction changes, we further estimate the person's walking path.

Fig. 12. Example of making a right turn with multiple steps.

Fig. 13. Example of the angle change sequence and its corresponding path.

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 $s_{i,k}^l(\tau)$ or $s_{i,k}^r(\tau)$, the features are

$$F_{i,k}^l = \max_{\tau} s_{i,k}^l(\tau), \quad i = 1, 2, \dots, 8, \quad \text{or} \quad F_{i,k}^r = \max_{\tau} s_{i,k}^r(\tau), \quad i = 1, 2, \dots, 8. \quad (4)$$

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

7.3 Angle Regression

In Section 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 [50], 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 $\{F_{i,k}^l | i = 1, 2, \dots, 8\}$ or $\{F_{i,k}^r | i = 1, 2, \dots, 8\}$. Then, the direction change caused by the step is predicted. The result follows the format shown in Fig. 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 $\{\Delta\theta_n | n = 1, 2, \dots\}$, and an example is shown in Fig. 13(a). Note that $\{\Delta\theta_n\}$ is the output for a single in-corridor walk. We repeat the same process for each walk.

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 [33]. However, the detailed parameters of the linear relation vary among different people [8]. 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_{stride} . Obviously, l_{stride} only impacts the scale of the estimated map output by ShoesHacker. Without loss of generality, we set $l_{\text{stride}} = 0.77$ 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° or lower than -60° , the three steps are labeled as potential left or right turning steps. Assume that the indexes of the continuous potential turning steps are $a, \dots, b$, 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 $l_{\text{stride}} = 0.77$ 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 Fig. 13(b). For the w th in-corridor walk, the path can be stored as a tuple sequence $\{(x_i^w, y_i^w) | i = 1, \dots, N_c^w\}$, which denotes the coordinates of the corners (including the start and end points) on the walking path. N_c^w is the total number of turns during the w th walk, and $w = 1, \dots, N_w$. Note that the tuples in $\{(x_i^w, y_i^w) | i = 1, \dots, N_c^w\}$ are ordered, and the coordinates only show the relative locations of the turning points. Then absolute location and direction of the path are unknown.

8 CORRIDOR MAP ESTIMATOR

The corridor map estimator merges the paths $\{(x_i^w, y_i^w) | i = 1, \dots, N_c^w\}$, $w = 1, \dots, N_w$, and reconstructs the corridor map in the floor plan.

8.1 Design Considerations

As illustrated by Fig. 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 [18, 39]. 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

Fig. 14. Example of path merging.

Fig. 15. Two paths with same shape but with different corridor orders.

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 Fig. 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 Fig. 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 Fig. 16. Note that in Fig. 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 Fig. 17.

Fig. 16. Examples of merging nearby intersections and corridors, and creating new intersections.

Fig. 17. Workflow of corridor map estimator. In this example, $\{(x_i^1, y_i^1) | i = 1, \dots, N_c^1\}$ is selected to generate the initial base graph.

8.2 Path Preprocessing

For each in-corridor walking path, the walking path estimator outputs $\{(x_i^w, y_i^w) | i = 1, \dots, N_c^w\}$. 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 $\vec{o}_w$, as shown in Fig. 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., $e(v_1, v_2) = e(v_2, v_1)$. $\vec{o}_w$ is a sequence of vertices, which denotes the order by which the person went through the vertices. $\vec{o}_w[i]$ denotes the i th vertex in $\vec{o}_w$. 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 2. In line 2 we start to traverse each tuple in $\{(x_i^w, y_i^w)\}$. In line 3-4, if the current corner is near to a previously-visited vertex, we consider that the person went through the previous

ALGORITHM 2: Path preprocessing

Input: Path tuple sequence $\{(x_i^w, y_i^w) | i = 1, \dots, N_c^w\}$;
Parameter: Distance threshold d_{T1} ;
Output: Vertex set V_w ; Edge set E_w ; Traverse order $\vec{o}_w$;

```

1  $V \leftarrow \emptyset$ ;  $E \leftarrow \emptyset$ ;  $\vec{o} \leftarrow []$ ;
2 foreach  $i = 1, \dots, N_c^w$  do
3   if  $\sqrt{(x_i^w - v.x)^2 + (y_i^w - v.y)^2} < d_{T1}, \exists v \in V$  then
4     // Merge nearby intersections
5     Add  $v$  to the end of  $\vec{o}$ ;
6   else
7     foreach  $e(v_1, v_2) \in E$  do
8       Calculate the vertical line from vertex  $(x_i^w, y_i^w)$  to edge  $e(v_1, v_2)$ ;
9       if the foot point  $(x, y)$  is on the edge, and  $\sqrt{(x_i^w - x)^2 + (y_i^w - y)^2} < d_{T1}$  then
10        // Merge nearby corridors
11        Create a new vertex  $v$ , where  $v.x = x, v.y = y$ ;
12         $V \leftarrow V \cup \{v\}$ ;
13         $E \leftarrow E \setminus \{e(v_1, v_2)\}; E \leftarrow E \cup \{e(v_1, v), e(v, v_2)\}$ ;
14        Insert  $v$  between any adjacent  $v_1$  and  $v_2$  in  $\vec{o}$ , and add  $v$  to the end of  $\vec{o}$ ;
15        break;
16   if no new vertex is added in line 6-13 then
17     Create a new vertex  $v$ , where  $v.x = x_i^w, v.y = y_i^w$ ;
18      $V \leftarrow V \cup \{v\}$ ;
19     Add  $v$  to the end of  $\vec{o}$ ;
20   if  $\|\vec{o}\| < 2$  then continue;
21   // Add new vertex if two edges intersects
22   Denote the current last two elements in  $\vec{o}$  by  $v_1$  and  $v_2$ ;
23   if the segment from  $v_1$  to  $v_2$  successively intersects edges  $e(v_1^j, v_2^j) \in E, j = 1, \dots, J$  then
24     foreach  $j = 1, \dots, J$  do
25       Denote the intersection by  $(x, y)$ . Create a new vertex  $v$ , where  $v.x = x, v.y = y$ ;
26        $V \leftarrow V \cup \{v\}$ ;
27        $E \leftarrow E \setminus \{e(v_1^j, v_2^j)\}; E \leftarrow E \cup \{e(v_1^j, v), e(v, v_2^j), e(v_1, v), e(v, v_2)\}$ ;
28       Insert  $v$  between any adjacent  $v_1$  and  $v_2$  in  $\vec{o}$ ;
29        $v_1 \leftarrow v$ ;
30   if no new vertex is added in line 20-26 then  $E \leftarrow E \cup \{e(v_1, v_2)\}$ ;
31 Create a new path tuple sequence  $\{(x_i^w, y_i^w)\}$  based on  $\vec{o}$  and  $V$ , and then reverse the tuple order ;
32 Repeat line 1-27 with the new path tuple sequence as the input. Get new  $V, E$  and  $\vec{o}$ , and relabel them by  $V_w, E_w$  and  $\vec{o}_w$ , respectively;
33 return  $V_w, E_w$  and  $\vec{o}_w$ ;

```

vertex, and merge the two vertices. In our implementation, the distance bound of vertex merging is $d_{T1} = 4$ m. The case in Fig. 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 Fig. 16(b). If no nearby edge is detected, a new vertex is 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 Fig. 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 Fig. 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.

8.3 Path Merging

The path merging algorithm is shown in Algorithms 3-5. 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 3, and then merge each path into the base graph with following Algorithm 4. After each merging, we re-conduct Algorithm 3 on the base graph before we try to merge the next path.

ALGORITHM 3: Base graph extension

Input: Edge set E ;
Output: Extended edge set $\hat{E}$;

```

1  $E' \leftarrow \emptyset$ ;
2 foreach  $e(v_1, v_2) \in E$  and  $e(v_1, v_2) \notin E'$  do
3    $V' \leftarrow \{v_1, v_2\}$ ;
4   while the size of  $V'$  has changed do
5     foreach  $v \in V'$  do
6       foreach  $e(v'_1, v'_2) \in E$  do
7         if  $v'_1 = v$  or  $v'_2 = v$ , and  $e(v'_1, v'_2)$  is collinear with  $e(v_1, v_2)$  then  $V' \leftarrow V' \cup \{v'_1, v'_2\}$ ;
8   foreach combination of two vertices  $v_a, v_b$  in  $V'$  do  $E' \leftarrow E' \cup \{e(v_a, v_b)\}$ ;
9  $\hat{E} \leftarrow E \cup E'$ ;
10 return  $\hat{E}$ ;
```

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 Fig. 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 Fig. 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 $e(a, c)$.

Fig. 18. The necessity of extending the edge set in the base graph.

For the convenience of future merging, we first extend the base graph following Algorithm 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 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 Fig. 18(a) all the 6 edges among a, b, c, d are included in $\hat{E}$, and in Fig. 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 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 Fig. 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 5 to get the matching level of the path (line 4).

ALGORITHM 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 $\vec{o}_w$;
Parameter: Minimum matching edge number N_e ; Distance threshold d_{T1} ;
Output: Merged vertex set $\tilde{V}$; Merged edge set $\tilde{E}$; Edge counter set C_E ;

```

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

In Algorithm 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 $[\vec{o}_w[i], \dots, \vec{o}_w[j-1]]$ matches with the path $[\dots, v_1, v_2]$ 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 $e_w(\vec{o}_w[j-1], \vec{o}_w[j])$

ALGORITHM 5: Given start edge in base graph, find the longest matched segment between base graph and new path

Input: Base graph's vertex set V and extended edge set $\hat{E}$; Starting edge $e(v_1, v_2)$; New path's vertex set V_w , edge set E_w and traverse order $\vec{o}_w$;

Parameter: Edge length difference threshold d_{T2} ;

Output: Maximum matched edge number N_M ; Edge length difference d_M ; Best matched segment in base graph $\vec{o}_B$; Best matched segment in new path $\vec{o}_P$;

```

1  $N_M \leftarrow 0$ ;  $d_M \leftarrow \text{Inf}$ ;  $\vec{o}_B \leftarrow []$ ;  $\vec{o}_P \leftarrow []$ ;
2 foreach  $i = 1, \dots, \|\vec{o}_w\| - 1$  do
    //  $e_w(\cdot, \cdot)$  denotes an edge in  $E_w$ ;  $e(\cdot, \cdot)$  denotes an edge in  $\hat{E}$ ;  $\|e(\cdot, \cdot)\|$  is the length of  $e(\cdot, \cdot)$ ;
3      $d_m \leftarrow \|e_w(\vec{o}_w[i], \vec{o}_w[i+1])\| - \|e(v_1, v_2)\|$ ;
4     if  $d_m > d_{T2}$  then continue;
5      $\vec{o}_b \leftarrow [v_1, v_2]$ ;
6      $\vec{o}_p \leftarrow [\vec{o}_w[i], \vec{o}_w[i+1]]$ ;
7      $N_m \leftarrow 1$ ;
8     foreach  $j = i + 2, \dots, \|\vec{o}_w\|$  do
9          $d \leftarrow \text{Inf}$ ;  $v_b \leftarrow 0$ ;  $v_p \leftarrow 0$ ;
10        foreach  $e(v_2, v_3) \in \hat{E}$  and  $v_3 \neq v_1$  do
11            if turning direction from  $e(v_1, v_2)$  to  $e(v_2, v_3)$  is same with that from  $e_w(\vec{o}_w[j-2], \vec{o}_w[j-1])$  to
                 $e_w(\vec{o}_w[j-1], \vec{o}_w[j])$  then
12                 $d' = \|e_w(\vec{o}_w[j-1], \vec{o}_w[j])\| - \|e(v_2, v_3)\|$ ;
13                if  $d' < d$  then
14                     $d \leftarrow d'$ ;  $v_b \leftarrow v_3$ ;  $v_p \leftarrow \vec{o}_w[j]$ ;
15        if  $d > d_{T2}$  then break;
16        else
17             $v_1 \leftarrow v_2$ ;  $v_2 \leftarrow v_3$ ;
18            Add  $v_b$  to the end of  $\vec{o}_b$ ; Add  $v_p$  to the end of  $\vec{o}_p$ ;
19             $N_m \leftarrow N_m + 1$ ;
20             $d_m \leftarrow d_m + d$ ;
21            if  $N_m > N_M$ , or  $N_m = N_M$  and  $d_m < d_M$  then
22                 $N_M \leftarrow N_m$ ;  $d_M \leftarrow d_m$ ;  $\vec{o}_B \leftarrow \vec{o}_b$ ;  $\vec{o}_P \leftarrow \vec{o}_p$ ;
23 return  $N_M, d_M, \vec{o}_B$  and  $\vec{o}_P$ ;

```

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 4.

Back to line 4 in Algorithm 4, we try each start edge, and record the best matched 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 $\vec{o}_{\text{pre}}$ and $\vec{o}_{\text{suf}}$ 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 2. Finally, the algorithm outputs the new base graph $\tilde{V}, \tilde{E}$. We conduct Algorithm 3 on $\tilde{E}$ to connect the collinear vertices, and then start to process the next path with the new base graph, as shown in Fig. 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 paths. This process is repeated until no more path can be merged.

If the current merged path is estimated based on the latest data leaked from the victim, the end point of the suffix path $\vec{o}_{\text{suf}}$ in the merged graph is the latest location of the victim.

Eliminating Inaccurate Edges: It is possible that some wrong edges are merged to the base graph, which cannot be directly detected. To handle this, we keep a counter for each edge, which records the number of times the edge appears in the matched segments. In Algorithm 4, we keep a counter set C_E that denotes these counters. When enough data have been collected, the attacker can discard the edges with low counter values. Thus, the impacts of inaccurate edges can be reduced. It is also possible that the path selected as the base graph is incorrect, which leads to wrong merging results. This can be avoided by trying different paths as the base graph, and check the consistency of the results.

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.

Fig. 19. Comparison based on Hamming distance.

Fig. 20. Brute-force comparison.

Fig. 21. Issues in the naive metric.

Intuitively, we can treat the graph as topological structures, and apply the corresponding mathematical algorithms to evaluate their difference [44]. 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 Fig. 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 Fig. 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 Fig. 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_{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 Fig. 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 Fig. 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 Fig. 21(b), graph B and B' have the same distance to A , but obviously B' is more similar to A . To hand 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 Fig. 21(a), $\text{diff}'_B(A)$ will be huge, and in Fig. 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.

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 10.4. Moreover, in Section 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 [40] 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.

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 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 Fig. 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 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. Fig. 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.

Fig. 22. Precision and recall of stair landing detection.

Fig. 23. Performance of stair landing detection when sensor number changes.

Fig. 24. Probability distribution of step number in stair landing groups.

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 Fig. 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.

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 10.1. For each volunteer, we merge the data collected during all the test cases and build a personal training dataset.

Fig. 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

Fig. 25. Probability distribution of labeled training data. Note that 0° -steps are not shown since it is not collected during stair landing detection.

Fig. 26. Probability distribution of angle estimation error.

Fig. 27. Average absolute angle estimation error when the sensor number changes.

Section 5.3. The probability distribution of the corresponding labels (in absolute values) is shown in Fig. 25. Since the labels are calculated by $180^\circ / (L + 1)$, and the step number L is an integer usually ranging in $[1, 7]$, the absolute degree can only be 22.5° , 25.7° , 30° , 36° , 45° , 60° , or 90° . Apparently, the training data between 60° and 90° is vacant. However, we will show that the performance of the angle regression is still acceptable in this situation.

Fig. 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 5.3, the training steps of 0° 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 Fig. 26. The sign of the errors follows Fig. 11. The average signed error is -2.6° , and the average absolute error is 20.3° . ShoesLoc [51] also estimates the direction change made by each step, and achieves a lower average absolute error (around 16°). 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 10.3.

Furthermore, we analyze the performance when the sensor number changes. We follow the same sensor selection as shown in Fig. 23. The results are shown in Fig. 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 Fig. 23, the current 8-sensor setting achieves a higher recall rates, which speeds up the training data collection.

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 10.2.

Fig. 28. The floor plans of the test regions.

Fig. 29. Examples of path estimation. The estimated path has been rotated and rescaled to fit the real path during the calculation of $\text{minDiff}(A, B)$. Fig. 30. Probability distribution of $\text{minDiff}(A, B)$ between estimated paths and real paths.

During the test, each volunteer walks along 10 randomly selected paths in each of the three buildings shown in Fig. 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 $\text{minDiff}(A, B)$, and Fig. 29 illustrates some examples. The probability distribution of the results is shown in Fig. 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 Fig. 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 10.4.

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 Fig. 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 10.2.

Fig. 31. Examples of estimated corridor maps. The oblique lines are caused by the intersection merging.

Fig. 32. Probability distribution of $\min\text{Diff}(A, B)$ between estimated maps and real maps.

Fig. 33. Impacts of different shoe types and changing speed.

Fig. 34. CDF of localization error.

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 Fig. 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 $\min\text{Diff}(A, B)$ 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 10.5.

We compare each estimated map with the ground truth using $\min\text{Diff}(A, B)$, and the results are shown in Fig. 32. The average $\min\text{Diff}(A, B)$ 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 Fig. 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 Fig. 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 Fig. 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,

Fig. 35. CMC curve of building recognition.

Fig. 36. Example of an error during building recognition. The red lines show the estimated map for the floor in Fig. 28(e). The extra turn at A causes the wrong matching.

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 Fig. 34. The average error is 5.4 m, which shows that if the attacker has determined the building (as will be shown in Section 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.

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 [34]. The corridors in each floor plan are manually labeled. Together with the five floors shown in Fig. 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 10.4, we compare it with each of the 40 corridor maps in the pool using $\text{minDiff}(A, B)$, and sort the results. The corridor map with the smallest $\text{minDiff}(A, B)$ is the best match. In Fig. 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 [35].

As shown in Fig. 35, the rank-1 accuracy is 77.5% for the five buildings. One typical recognition error is shown in Fig. 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 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.

11 DISCUSSION

11.1 Limitation of ShoesHacker

Atypical staircases: Currently ShoesHacker only handles the typical staircases as shown in Fig. 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 [1, 5, 46]. 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 paper, 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 $\text{minDiff}(A, B)$ 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 [51] 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.

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. 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 [51] can be applied. The accuracy of ShoesLoc is around 1 m, which makes the room-level localization possible.

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.

12 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.

ACKNOWLEDGMENTS

This research was funded by the National Science Foundation under award numbers CNS-1330491 and OAC-1827126, and the Ralph M. and Catherine V. Fisher Professorship. The views and conclusions contained in this

document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsors.

REFERENCES

- [1] Heba Abdelnasser, Reham Mohamed, et al. 2016. SemanticSLAM: Using environment landmarks for unsupervised indoor localization. *IEEE Transactions on Mobile Computing* 15, 7 (2016), 1770–1782.
- [2] Nasim Alikhani, Shima Amiranloo, Vahideh Moghtadaiee, and Seyed Ali Ghorashi. 2017. Fast fingerprinting based indoor localization by Wi-Fi signals. In *2017 7th International Conference on Computer and Knowledge Engineering (ICCCKE)*. IEEE, 241–246.
- [3] Marco Altini, Davide Brunelli, Elisabetta Farella, and Luca Benini. 2010. Bluetooth indoor localization with multiple neural networks. In *IEEE 5th International Symposium on Wireless Pervasive Computing 2010*. IEEE, 295–300.
- [4] Altra. 2018. Altra IQ powered by iFit. Retrieved March 22, 2018 from <https://www.altrarunning.com/iq>
- [5] Moustafa Alzantot and Moustafa Youssef. 2012. Crowdinside: automatic construction of indoor floorplans. In *SIGSPATIAL*. ACM, 99–108.
- [6] Paramvir Bahl, Venkata N Padmanabhan, et al. 2000. RADAR: An in-building RF-based user location and tracking system. In *IEEE infocom*, Vol. 2. INSTITUTE OF ELECTRICAL ENGINEERS INC (IEEE), 775–784.
- [7] Ling Bao and Stephen S Intille. 2004. Activity recognition from user-annotated acceleration data. In *International conference on pervasive computing*. Springer, 1–17.
- [8] Chiraz BenAbdelkader, Ross Cutler, and Larry Davis. 2002. Stride and cadence as a biometric in automatic person identification and verification. In *Proceedings of Fifth IEEE International Conference on Automatic Face Gesture Recognition*. IEEE, 372–377.
- [9] David Berend, Bernhard Jungk, and Shivam Bhasin. 2018. Guessing your PIN right: Unlocking smartphones with publicly available sensor data. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 363–368.
- [10] Si Chen, Muyuan Li, Kui Ren, and Chunming Qiao. 2015. Crowd map: Accurate reconstruction of indoor floor plans from crowdsourced sensor-rich videos. In *2015 IEEE 35th International conference on distributed computing systems*. IEEE, 1–10.
- [11] Anupam Das and Nikita Borisov. 2014. Poster: Fingerprinting smartphones through speaker. In *Poster at the IEEE Security and Privacy Symposium*. Citeseer.
- [12] Yves-Alexandre De Montjoye, César A Hidalgo, Michel Verleysen, and Vincent D Blondel. 2013. Unique in the crowd: The privacy bounds of human mobility. *Scientific reports* 3 (2013), 1376.
- [13] Sanorita Dey, Nirupam Roy, Wenyan Xu, Romit Roy Choudhury, and Srihari Nelakuditi. 2014. AccelPrint: Imperfections of Accelerometers Make Smartphones Trackable.. In *NDSS*.
- [14] Julien Freudiger, Reza Shokri, and Jean-Pierre Hubaux. 2011. Evaluating the privacy risk of location-based services. In *International conference on financial cryptography and data security*. Springer, 31–46.
- [15] Ruipeng Gao, Mingmin Zhao, Tao Ye, Fan Ye, Yizhou Wang, Kaigui Bian, Tao Wang, and Xiaoming Li. 2014. Jigsaw: Indoor floor plan reconstruction via mobile crowdsensing. In *Proceedings of MobiCom '14*. ACM, 249–260.
- [16] Google. 2018. Google Maps. Retrieved September 1, 2018 from <https://www.google.com/maps/about/partners/indoormap/>
- [17] HM Government. 2013. Approved Document K: protection from falling, collision and impact. Retrieved July 7, 2019 from <https://www.gov.uk/government/publications/protection-from-falling-collision-and-impact-approved-document-k>
- [18] Nuno Gracias, Mohammad Mahoor, Shahriar Negahdaripour, and Arthur Gleason. 2009. Fast image blending using watersheds and graph cuts. *Image and Vision Computing* 27, 5 (2009), 597–607.
- [19] Marco Gruteser and Dirk Grunwald. 2003. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proceedings of the 1st international conference on Mobile systems, applications and services*. ACM, 31–42.
- [20] Jun Han, Emmanuel Owusu, Le T Nguyen, Adrian Perrig, and Joy Zhang. 2012. Accomplice: Location inference using accelerometers on smartphones. In *2012 Fourth International Conference on Communication Systems and Networks (COMSNETS 2012)*. IEEE, 1–9.
- [21] Suining He and S-H Gary Chan. 2016. Wi-Fi fingerprint-based indoor positioning: Recent advances and comparisons. *IEEE Communications Surveys & Tutorials* 18, 1 (2016), 466–490.
- [22] Baik Hoh, Marco Gruteser, Hui Xiong, and Ansaif Alrabady. 2007. Preserving privacy in gps traces via uncertainty-aware path cloaking. In *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 161–171.
- [23] Yifei Jiang, Yun Xiang, Xin Pan, Kun Li, Qin Lv, Robert P Dick, Li Shang, and Michael Hannigan. 2013. Hallway based automatic indoor floorplan construction using room fingerprints. In *Proceedings of UbiComp '13*. ACM, 315–324.
- [24] Kickstarter. 2018. Digitsole Smartshoe The World's First Intelligent Sneaker. Retrieved March 22, 2018 from <https://www.kickstarter.com/projects/141658446/digitsole-smartshoe-the-worlds-first-intelligent-s>
- [25] John Krumm and Eric Horvitz. 2004. LOCADIO: Inferring Motion and Location from Wi-Fi Signal Strengths.. In *mobiQuitous*. 4–13.
- [26] Muyuan Li, Haojin Zhu, Zhaoyu Gao, Si Chen, Le Yu, Shangqian Hu, and Kui Ren. 2014. All your location are belong to us: Breaking mobile social networks for automated user location tracking. In *Proceedings of MobiHoc '14*. ACM, 43–52.
- [27] Chang Liu, Lei Xie, Chuyu Wang, Jie Wu, and Sanglu Lu. 2015. FootStep-tracker: an anchor-free indoor localization system via sensing foot steps. In *Adjunct Proceedings of UbiComp/ISWC'15*. ACM, 285–288.

- [28] Chang Liu, Lei Xie, Chuyu Wang, Jie Wu, and Sanglu Lu. 2016. Track Your Foot Step: Anchor-free Indoor Localization based on Sensing Users' Foot Steps. In *2016 IEEE 13th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 201–209.
- [29] Jan Lukáš, Jessica Fridrich, and Miroslav Goljan. 2006. Digital camera identification from sensor pattern noise. *IEEE Transactions on Information Forensics and Security* 1, 2 (2006), 205–214.
- [30] Anindya Maiti, Ryan Heard, Mohd Sabra, and Murtuza Jadliwala. 2018. Towards Inferring Mechanical Lock Combinations using Wrist-Wearables as a Side-Channel. In *Proceedings of WiSec '18*. ACM, 111–122.
- [31] Anindya Maiti, Murtuza Jadliwala, Jibo He, and Igor Bilogrevic. 2018. Side-channel inference attacks on mobile keypads using smartwatches. *IEEE Transactions on Mobile Computing* 17, 9 (2018), 2180–2194.
- [32] Yan Michalevsky, Aaron Schulman, Gunaa Arumugam Veerapandian, Dan Boneh, and Gabi Nakibly. 2015. Powerspy: Location tracking using mobile device power analysis. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*. 785–800.
- [33] Meg Morris, Robert Iansek, Thomas Matyas, and Jeffery Summers. 1998. Abnormalities in the stride length-cadence relation in parkinsonian gait. *Movement disorders: official journal of the Movement Disorder Society* 13, 1 (1998), 61–69.
- [34] University Polic Department of University of Illinois. 2019. Building Emergency Action Plans. Retrieved April 18, 2019 from <https://police.illinois.edu/emergency-preparedness/building-emergency-action-plans/>
- [35] Sakrapee Paisitkriangkrai, Chunhua Shen, and Anton Van Den Hengel. 2015. Learning to rank in person re-identification with metric ensembles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1846–1855.
- [36] Dung Phan, Lee Yee Siong, Pubudu N Pathirana, and Aruna Seneviratne. 2015. Smartwatch: Performance evaluation for long-term heart rate monitoring. In *2015 International Symposium on Bioelectronics and Bioinformatics (ISBB)*. IEEE, 144–147.
- [37] Damian Philipp, Patrick Baier, Christoph Dibak, Frank Dürr, Kurt Rothermel, Susanne Becker, Michael Peter, and Dieter Fritsch. 2014. Mapgenie: Grammar-enhanced indoor map construction from crowd-sourced data. In *IEEE PerCom*. IEEE, 139–147.
- [38] Swadhin Pradhan, Ghufan Baig, Wenguang Mao, Lili Qiu, Guohai Chen, and Bo Yang. 2018. Smartphone-based Acoustic Indoor Space Mapping. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 2 (2018), 75.
- [39] Vlado Rankov, Rosalind J Locke, et al. 2005. An algorithm for image stitching and blending. In *Three-Dimensional and Multidimensional Microscopy: Image Acquisition and Processing XII*, Vol. 5701. International Society for Optics and Photonics, 190–200.
- [40] ReTiSense. 2018. The new Stridalyzer INSIGHT Clinical Diagnostic Solution. Retrieved March 8, 2018 from <http://www.retisense.com/>
- [41] Mudhakar Srivatsa and Mike Hicks. 2012. Deanonymizing mobility traces: Using social network as a side-channel. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 628–637.
- [42] Enno-Edzard Steen, Marco Eichelberg, Wolfgang Nebel, and Andreas Hein. 2012. A novel indoor localization approach using dynamic changes in ultrasonic echoes. In *Ambient Assisted Living*. Springer, 123–133.
- [43] Hartmut Surmann, Andreas Nüchter, and Joachim Hertzberg. 2003. An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments. *Robotics and Autonomous Systems* 45, 3–4 (2003), 181–198.
- [44] P Van Mieghem. 2013. Can the topology of two graphs be compared by one number? *Delft University of Technology* (2013), 72.
- [45] He Wang, Ted Tsung-Te Lai, and Romit Roy Choudhury. 2015. Mole: Motion leaks through smartwatch sensors. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM, 155–166.
- [46] He Wang, Souvik Sen, Ahmed Elgohary, Moustafa Farid, Moustafa Youssef, and Romit Roy Choudhury. 2012. No need to war-drive: Unsupervised indoor localization. In *Proceedings of MobiSys '12*. ACM, 197–210.
- [47] Jialin Wang, Hanni Cheng, Minhui Xue, and Xiaojun He. 2017. Revisiting Localization Attacks in Mobile App People-Nearby Services. In *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*. Springer, 17–30.
- [48] Oliver Woodman and Robert Harle. 2008. Pedestrian localisation for indoor environments. In *Proceedings of UbiComp '08*. ACM, 114–123.
- [49] Henk Wymeersch, Stefano Marano, Wesley M Gifford, and Moe Z Win. 2012. A machine learning approach to ranging error mitigation for UWB localization. *IEEE transactions on communications* 60, 6 (2012), 1719–1728.
- [50] Yujiang Xiang, Jasbir S Arora, and Karim Abdel-Malek. 2011. Optimization-based prediction of asymmetric human gait. *Journal of Biomechanics* 44, 4 (2011), 683–693.
- [51] Tuo Yu, Haiming Jin, and Klara Nahrstedt. 2019. ShoesLoc: In-Shoe Force Sensor-Based Indoor Walking Path Tracking. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 3, 1 (2019), 31.
- [52] Hui Zang and Jean Bolot. 2011. Anonymization of location data does not work: A large-scale measurement study. In *Proceedings of the 17th annual international conference on Mobile computing and networking*. ACM, 145–156.
- [53] Yang Zhao, Kaihua Liu, Yongtao Ma, Zheng Gao, Yangguang Zang, and Jianfu Teng. 2017. Similarity analysis-based indoor localization algorithm with backscatter information of passive UHF RFID tags. *IEEE Sensors Journal* 17, 1 (2017), 185–193.
- [54] Zhor-Tech. 2019. Pressure Sensors. Retrieved April 1, 2019 from <https://zhortechnology.com/foot-pressure-sensor/>
- [55] Xiaoyong Zhou, Soteris Demetriou, Dongjing He, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, Carl A Gunter, and Klara Nahrstedt. 2013. Identity, location, disease and more: Inferring your secrets from android public resources. In *CCS '13*. ACM, 1017–1028.