Smart Applications with Enriched Sensors on Mobile Devices

Xinye Lin

Doctor of Philosophy

School of Computer Science
McGill University, Montreal
2018-04

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Doctor of Philosophy.

Dedication

This thesis is dedicated to my best friend Yiling, my parents and my sister for their precious love and support.

Acknowledgements

I would like to present my most sincere gratitude to my supervisors, Professor Xiao-Wen Chang and Professor Xue Liu, who have guided me through this long journey, provided me with precious advice both in the academic area and everyday life. Their persistence and self-discipline have inspired me for countless times when I was lost in the kingdom of research.

I would also like to express my deepest thanks to Professor Xingming Zhou and Professor Xiaodong Wang. Professor Zhou is a great computer scientist who dedicates his whole life to this field, and his deep understanding and broad vision about this subject has been my beacon over the years. Professor Wang is a respectful and warm-hearted elder brother, giving a lot of insightful advice about my career and life, and has provided me with endless support during my stay in China.

I am also extremely lucky to have all the volunteer participants who contributed in the experiments of my research, sacrificing their personal time to provide first-handed data in this research area.

I thank my girlfriend Yiling, my parents and sister, my great friend Lorenz Luthe for their firmly support all over the years.

I am much grateful for this opportunity of studying in McGill and will never forget these marvelous years in Montreal.

Grandescunt Aucta Labore.

Abstract

Empowered by embedded sensors with increasing variety and precision, mobile devices like smart phones and smart watches are gifted with unprecedented capabilities. With the proliferation of these devices, new applications emerge spanning from health assistance to activity profiling. Novel solutions for existing mobile computing problems also become possible.

Inspired by the versatile sensors, this thesis seeks to take full use of them to address two problems: indoor localization and text-entry on the smart watch, and proposes *LocMe* and *SHOW*, respectively.

LocMe provides a new perspective of solving the indoor localization problem without the requirement of any infrastructure. By sensing the user's locomotion states and synergizing the new locomotion-constraint with the classic wall-constraint in a particle filter, it achieves better convergence and accuracy than existing wall-constraint only methods, with negligible extra complexity. Our field tests show that *LocMe* can achieve a median localization error of 1.1 m, which is over 68% lower than the localization algorithm with only the wall-constraint under the same test conditions.

SHOW senses the user's wrist movement with a smart watch to recognize his/her handwritings, hence enables a novel text-entry method. By implementing rotation injection, *SHOW* manages to use real hand-writing traces as seeds to automatically generate quantities of traces which are both effective and robust for training purposes. Our experiments show that *SHOW* can effectively generate 60 traces from a single real handwriting trace and achieve high accuracy at 99.9% when recognizing the 62 different characters written by 10 volunteers. Furthermore, having more screen space after removing the virtual key-

board, *SHOW* can display 4x candidate words for autocompletion. Aided by the tolerance of character ambiguity and accurate character recognition, *SHOW* achieves over 70% lower mis-recognition-rate, 43% lower no-response-rate in both daily and general purposed text-entry scenarios, and 33.3% higher word suggestion coverage than the tap-on-screen method using a virtual QWERTY keyboard.

Abrégé

Dotés de capteurs intégrés de plus en plus variés et précis, les appareils mobiles tels que les téléphones intelligents et les montres intelligentes sont dotés de capacités sans précédent. Avec la prolifération de ces dispositifs, de nouvelles applications vont de lassistance sanitaire au profilage dactivité. De nouvelles solutions pour les problèmes informatiques existants deviennent également possibles.

Inspirée par les capteurs polyvalents, cette thèse vise à les utiliser pleinement pour résoudre deux problèmes: la localisation en intérieur et la saisie de texte sur smart watch, et proposer respectivement LocMe et SHOW.

LocMe fournit une nouvelle perspective pour résoudre le problème de localisation en intérieur sans lexigence de toute infrastructure. En détectant les mouvements de lutilisateur et en synergisant la nouvelle contrainte de locomotion avec une paroi de contrainte classique dans un filtre à particules, il obtient une meilleure convergence et précision que les méthodes existantes de contraintes murales, avec une complexité supplémentaire négligeable. Nos tests sur le terrain montrent que LocMe peut atteindre une erreur de localisation médiane de 1,1 m, soit plus de 68% inférieure à lalgorithme de localisation avec seulement la contrainte de paroi dans la même condition de test.

SHOW détecte le mouvement du poignet de lutilisateur avec une montre intelligente pour reconnaître son écriture manuscrite, permettant ainsi une nouvelle méthode de saisie de texte. En implémentant linjection de rotation, SHOW parvient à utiliser des traces réelles décriture manuscrite comme des semences pour générer automatiquement des quantités de traces à la fois efficaces et robustes à des fins de formation. Nos expériences montrent que SHOW peut effectivement générer 60 traces à partir dune seule trace décriture réelle et

atteindre une grande précision à 99,9% en reconnaissant les 62 caractères différents écrits par 10 volontaires. De plus, avec plus despace sur lécran après avoir supprimé le clavier virtuel, SHOW peut afficher 4x mots candidats pour lauto-complétion. Grâce à la tolérance de lambiguïté des caractères et à la reconnaissance précise des caractères, SHOW atteint un taux de non-reconnaissance inférieur de plus de 70%, un taux de non-reconnaissance de 43% dans les scénarios quotidiens et généraux et un taux de 33,3% de plus dans la couverture de mots suggérés que la méthode tap-on-screen en utilisant un clavier virtuel QWERTY.

Contents

List of Figures

List of Tables

List of Acronyms

AP Access Point.

BDT Bagging of decision trees.

CDF Cumulative Distribution Function.

DEFF Dominant escalator featured frequency.

HCI Human-computer interaction.

IMU Inertial measurement unit.

KNN K nearest neighbors.

LD Locomotion detector.

LED Light-emitting diode.

LocMe Human locomotion and map exploitation based indoor localization.

LocMe-WO LocMe with wall-constraint only.

LSTM Long short term memory.

MEMS Micro-electro-mechanical system.

MIMO Multiple-input and multiple-output.

MLR Multinomial logistic regression.

NAC Normalized auto-correlation.

NB Native bayes.

NN Neural network.

POI Place of interest.

RF Random forests.

SHOW Smart handwriting on watches.

SVM Support vector machine.

UI User interface.

Chapter 1

Introduction

The past few years have witnessed the explosive growth of mobile devices such as smart phones, tablets, smart watches and wristbands. In 2017 alone, a total of 1.47 billion units of smartphones [2] and 113.2 million of wearable devices (including smart watches and wristbands) [3] are shipped worldwide. On one hand, mobile devices have brought fairly powerful computing resources on the go to everyone around the world, weaving a ubiquitous platform that covers the most people in history; on the other hand, the unprecedented portability of these mobile devices has become a de facto life companion of human beings, accompanying their users anywhere and anytime. In this new era of mobile computing, new possibilities for all kinds of applications arise in two aspects: 1. New device forms like smart watches and wristbands bring inspiring application contexts and interaction needs; 2. Enriched sensors provide enhanced or even unprecedented capabilities of learning about the users' behaviours, activites and living/working environment. With these opportunities, we revisit both existing and emerging problems in mobile applications and seek for novel solutions. In particular, we focus on infrastructure-free indoor localization and text-entry on smart watches, and have developed *LocMe* [4] and *SHOW* [5] respectively.

LocMe employs the sensors such as accelerometer, magnetometer and barometer on the smart phone to detect the user's locomotion state. It analyzes the association between these locomotion states and special places of interest (POIs). For example, in an elevator, a user always moves vertically up or down; on an escalator, the user always moves monotonically up and down and experiences special vibrations due to the moving escalator steps. Based on

these associations, *LocMe* creates locomotion-constraints to filter out unrelated POIs once the user's locomotion states are detected. Furthermore, *LocMe* combines the locomotion-constraint with the wall-constraint, which assumes that no steps can cross the wall, and apply a particle filter to finally locate the user on the indoor map.

SHOW aims at providing a novel text-entry method on a smart watch by allowing the user to handwrite on horizontal surfaces. It collects the sensory traces of the user's wrist as he/she handwrites, and applies machine learning algorithms to learn and recognize these traces. *SHOW* invents a rotation injection technique, which can generate huge amount of traces from a single real trace, hence much alleviates the difficulty of data collection. Furthermore, *SHOW* designs intuitive gestures for character separation and correction, and achieves autocompletion with comparable or even better efficiency than the existing tap-on-screen input method.

This thesis is organized as follows.

- The first chapter briefly introduces the motivation, challenge and contribution of our work.

- The second chapter explains the background of our research and serves three major purposes: 1. Elaborate on the popularity of mobile devices and explain how they penetrate into every single aspect of our daily lives. 2. Introduce and present an overall perspective about the various sensors on modern mobile devices, explaining how they work and what kind of data they may provide; 3. Enumerate major categories of applications based on mobile devices employing their sensors and show a general picture about the cutting-edge research status in this area.

- The third and fourth chapter elaborate on how *LocMe* and *SHOW* are designed, implemented and tested, and introduce related work, respectively.

- The last chapter concludes this thesis, with discussion on the limitations of our current work and the future research directions.

1.1 Motivation

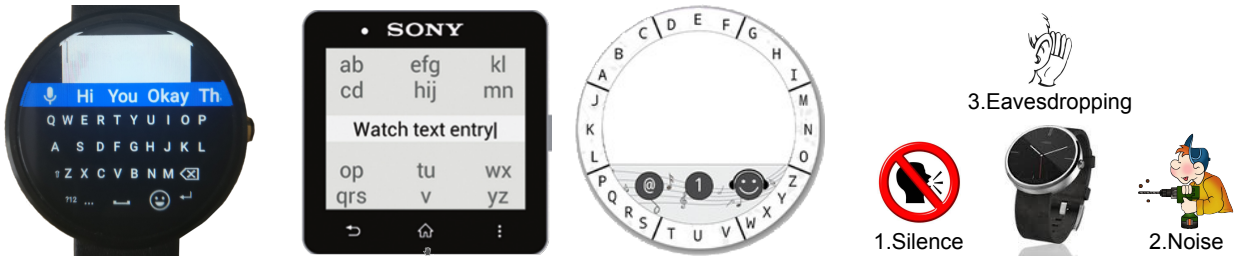
The motivation of the research work presented in this thesis is threefold, as explained in the following subsections.

1.1.1 Practical needs

Both *LocMe* and *SHOW* are motivated by practical needs from the user.

For *LocMe*, indoor localization is one of the most popular services on mobile devices. The cutting-edge indoor localization techniques can reach decimeter-level accuracy. However, these techniques require widely deployed infrastructure to work. They either rely on wireless access points (APs) with MIMO (multiple-input and multiple-output) capability [6–9], or need densely deployed LED lights which are not widely available [10,11]. These infrastructure may have existed in large cities, but are still absent in rural areas or less developed countries. As a result, an indoor localization without any needs for infrastructure is much in demand. At the meantime, due to its low cost, mobile devices like smart phones are prevalently owned even for people in poor areas. Therefore it makes a great solution if we can use mobile device as the only equipment in need for indoor localization. This is the main motivation for *LocMe*.

As for *SHOW*, text-entry on smart watches is becoming a leading application. Unfortunately, due to the small screen size of the watch, the tap-on-screen input method which is dominant on current smart phones can not be applied to the smart watch nicely with the densely packed virtual keyboard (Fig. 1.1a). Existing systems aiming at improving the



(a) (b) (c) (d)

Figure 1.1: Limitations of current input methods on the smart watch. (a) Densely packed virtual keyboard with the default qwerty layout. (b,c) Redesigned virtual keyboards [12,13], which are inconsistent with the one on smart phones. (d) Voice input.

tap-on-screen method end up creating new virtual keyboard layout (Fig. 1.1b and 1.1c). Its inconsistency with the keyboard on other devices imposes an unfriendly learning curve and causes difficulties for users who frequently switch between the phone and the watch [12]. Alternatively, the de facto first choice text entry method on the watch, i.e. the voice input, faces the following limitations as listed in Fig. 1.1d: 1. The scenario might not be suitable for speaking. 2. The environment might be noisy. 3. The personal information is prone to eavesdropping. Consequently, a novel text-entry method for smart watches is highly desired, and it is the primary motivation for *SHOW*.

1.1.2 Hardware and software are ready

Our essential idea for *LocMe* is based on the observation that there are many indoor POIs, in which the user express special locomotion patterns. If we can monitor the user's locomotion state, it is possible to infer which POI the user is passing through. After that, given an indoor map, with the POIs we just inferred, it is feasible to find the user's path and pinpoint his/her location.

The core idea for *SHOW* is based on the observation that when the user handwrites, the

smart watch on the wrists will move accordingly. Therefore it is possible to use some sensors to record the watch's moving trace and infer what the user has written.

It is straightforward to see that both *LocMe* and *SHOW* rely on the prerequisite of successfully recording and analyzing a certain kind of locomotion trace. Luckily, such prerequisite is already satisfied for a modern mobile device. Hardware wise, the sensors in need such as the accelerometer, gyroscope, magnetometer, barometer etc., are equipped by default in most off-the-shelf mobile devices. Software wise, main stream mobile operating systems such as iOS and Android provide the developers with rich APIs and libraries that the sensors of interest can be easily configured and programmed.

1.1.3 Sensing opportunities

Last but not least, mobile devices are carried by their users almost all the time, and this characteristic gives both *LocMe* and *SHOW* perfect opportunities for collecting sensory data. For example, *LocMe* requires the host device to be carried by the user while he/she is walking, taking elevators/escalators, and making turns, etc. It is hard to find any digital device other than mobile devices such as smart phones and watches that can easily meet such a requirement. The same happens to *SHOW*, for which no other devices than the smart watch can record the user's handwriting trace without bringing in inconvenience to their normal activities.

1.2 Challenges

On one hand, there is a fundamental challenge of sensory noise removal, which *LocMe* and *SHOW* both face. On the other hand, they each have application-specific challenges. We explain them in the following subsections.

1.2.1 Noise removal.

There are three kinds of noises in the collected sensory data: The instrumental noises, the environmental interference, and the arbitrary user movements.

1. **The instrumental noises** come with the sensors, it can be caused by the sensor malfunction, lack of calibration, or simply instrumental bias when the sensor is manufactured.

2. **The environmental interference** is caused by noises coming from the surrounding environment. For example, the magnetometer's readings are affected by electronic devices around it; the barometer's readings are affected by the air flow around it and the weather changes.

3. **The arbitrary user movements** are those unintentional movements by the user which affect the sensor readings. For example, the user's breath, body shaking and speaking affect the readings of the accelerometer and gyroscope.

It is critical for both *LocMe* and *SHOW* to remove these noises, so that the collected sensory data accurately describe the user's locomotion state.

1.2.2 *LocMe*: Locomotion state recognition.

Another tough challenge *LocMe* faces is distinguishing and detecting all kinds of locomotion states. Existing research mostly rely on machine learning techniques to classify the user activities [14–22]. They have two limitations: 1. Machine learning models for classification mostly work like a black box, where the high dimensional feature space is hard to be interpreted by human; 2. These models require large amount of user data for accurate classification, which is hard to acquire because the human activities of interest are both time

and energy consuming. Furthermore, existing research on activity profiling does not give much attention about how human locomotion states are related to different POIs, thus can hardly be used for localization. In Chapter 3, we will show that *LocMe* not only gives insights about how to distinguish different locomotion states, but also explores their association with different POIs to benefit the localization process. Also, we explain how these locomotion states are generated by carrying out extensive field tests, so that large quantities of user data for training is not mandatory.

1.2.3 *SHOW*: Large scale data collection.

SHOW applies machine learning models to learn about the handwriting traces. These models need to be trained with large amount of data before they can achieve high accuracy of recognition. Unfortunately, collecting data is a difficult task in *SHOW* because the test participants have to repeatedly handwrite the same characters, which is time-consuming, labor intensive and tedious. How to efficiently acquire large amount of handwriting traces is a prominent challenge *SHOW* faces. In Chapter 4, we will show that our proposed rotation injection technique is a nice answer to this challenge.

1.3 Contributions

As aforementioned, this thesis consists of two smart applications using sensors on mobile devices, and our contribution in the two applications are as the following.

For *LocMe*, our contribution is two-fold:

- First, we propose *LocMe*, a service that applies a synergy of the locomotion-constraint and the wall-constraint to achieve fast and accurate indoor localization. It guarantees faster convergence, and reduces the median of the localization error in our field tests

by 68% than the wall-constraint only methods.

- Second, by analyzing the user's locomotion states in special connectors such as stairs and escalators, *LocMe* can distinguish different floors and achieve dynamic map reloading.

For *SHOW*, our contribution is three-fold.

- First, we propose and implement *SHOW*, an accurate handwriting recognition system using smart watch. Our experiments show that it can reach an average of 99.9% character-wise recognition accuracy of the 62 different handwriting characters (A-Z, a-z and 0-9) from 10 volunteers writing on horizontal surfaces.

- Second, to our best knowledge, *SHOW* represents the first work that employs the potential inconsistency of watch positions to automatically generate sensor traces. It substantially alleviates the difficulties of collecting large amount of training samples, and boosts the robustness and generalizability of the dataset.

- Third, *SHOW* is the first work that achieves word autocompletion for handwriting with smart watches, and show potential for both daily and general purposed text-entry tasks with comparable efficiency, over 70% lower mis-recognition-rate, 43% lower no-response-rate, and 33.3% higher word suggestion coverage than the tap-on-screen method using a virtual QWERTY keyboard.

Chapter 2

Background

2.1 The Popularity of Mobile Devices

Modern mobile devices have penetrated into our everyday life and are becoming capable of almost any daily task. In this section, we present a full picture of the mobile devices' popularity from three aspects: the users, the apps and the usage paradigms.

Users. The most representative type of mobile devices is the smart phone. Fig. 2.1 displays the top 15 countries with the largest number of smart phone users as of April, 2017 [23] and the respective penetration in terms of percentage of the population [24]. It shows that, in many of the developed countries, the smart phone user penetration can be as high as 70%. In fact, 27 out of the top 50 countries with the most smart phone users have a high penetration over 50% [23]. Early in 2015, the smart phone penetration already surpassed 50% of the global population [25]. If we consider the hundreds of millions of other devices like tablets, smart watches and wrist bands that are manufactured and sold annually [3], it is reasonable to infer that mobile device is becoming the most owned digital equipment in human history.

Apps. Fig. 2.2 shows the changes of number of available apps in Google Play Store and Apple App Store from July, 2008 to December, 2017 [26, 27]. Within 10 years, the number of available apps has increased by over 200 times.

The current Google Play Store list 60 categories of apps, and Apple App Store lists 25

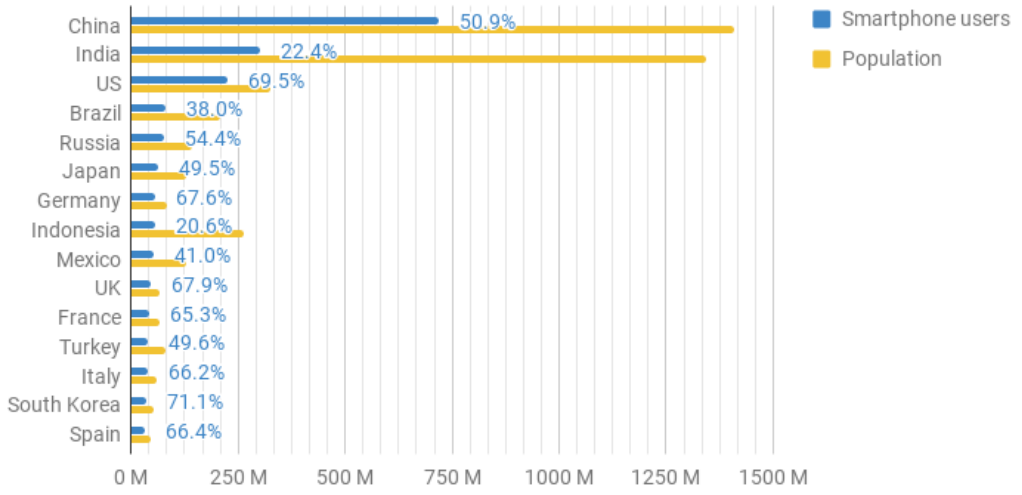


Figure 2.1: Top 15 countries with the most smartphone users as of April, 2017.

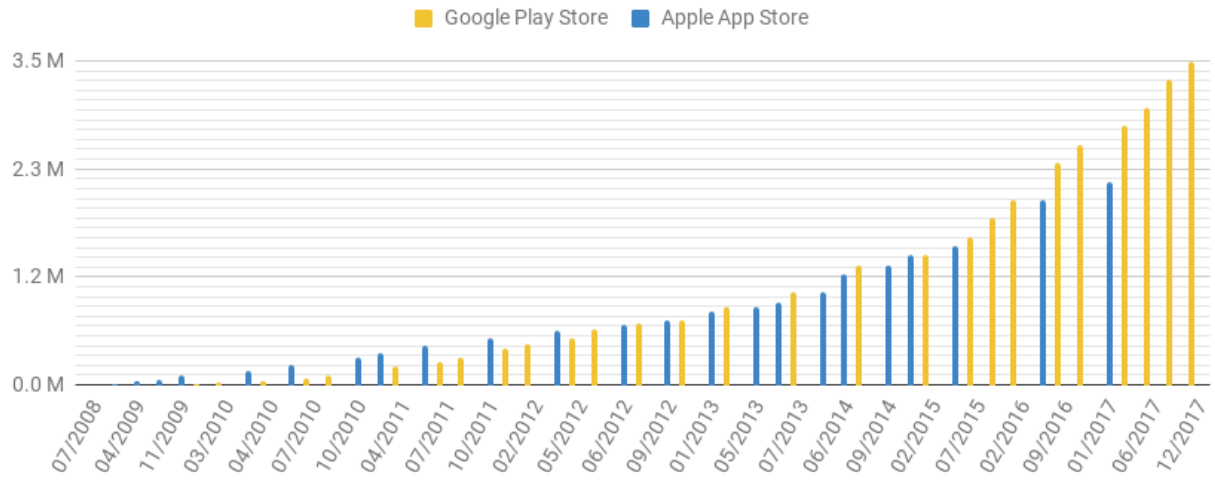


Figure 2.2: The number of available apps in Google Play Store and Apple App Store from July, 2008 to December, 2017.

categories and 61 subcategories. As a conclusion, the mobile apps are abundant both in terms of type and number, and have covered almost every aspect of our lives, bringing the user great convenience and efficiency.

Usage paradigms. From the user's perspective, the time and energy spent on mobile devices are also increasing rapidly. According to the investigation report by ComScore, the user in US spent 171 minutes on average on mobile phones every day [28], which is twice the time of 2011 [29–31]. Furthermore, new wearable devices such as smart watches and wristbands have much longer stand-by time (up to a month) and special features like water-resistance, which enable them to be accompanying the user almost all the time. As a result, mobile device is the irrefutable top choice for learning about user behaviors and sensing different contexts.

2.2 Sensors on the Mobile Devices

Table. 2.1 lists the sensors equipped on most of the off-the-shelf mobile devices. Among them, the accelerator, gyroscope and magnetometer are usually referred to as Inertial Measurement Units (IMU). Together they give the essential information about the user's locomotion state. IMU sensors are the building stones that support a wide range of applications, from activity profiling, fitness tracking to localization. Lying in the core of these applications, the idea of utilizing IMU sensory data is a two-step process called dead reckoning, which first integrates instantaneous IMU measurements, such as acceleration and angular speed into spatial quantities like angles and displacements, then based on them to derive higher level information such as locations and human activities. Unfortunately, the inevitable instrumental noises, arbitrary human movements and the sensitivity of integral operation greatly limit the precision of dead reckoning over time. In comparison with IMU sensors, other sensors may directly measure, or derive without integral operations, the spatial quantities. For example, the proximity sensor can directly measure the distance of an obstacle, and from the atmospheric pressure measured by the barometer, we can directly get the altitude. Besides, extra information from non-IMU sensors can help to accomplish missions impossible

Name	Measurement	Unit	Dimension
Accelerometer	acceleration	m/s^2	3D
Gyroscope	angular speed	rad/s	3D
Magnetometer	ambient magnetic field	μT	3D
Barometer	atmospheric pressure	hPa	1D
Light	ambient light level	lux	1D
Proximity	distance of obstacles	m	1D
Heart rate sensor	heart rate	bpm	1D
Microphone	sound frequency	Hz	1D
Camera	pixel matrix		

Table 2.1: Sensors equipped on most of the off-the-shelf mobile devices and their measurement.

for IMU sensors. For example, using light sensors we can directly infer whether a phone is in the pocket or not. All these sensors combined enable mobile devices to accomplish a wide range of everyday tasks, and they are one of the most important reasons that makes modern mobile phones "smart". In comparison, traditional non-smart phones (usually referred to as *feature phones*) usually only come with the microphone.

In the following subsections, we briefly introduce the physical mechanisms behind the first seven sensors in Table. 2.1, which are relatively new to mobile devices, and help the reader to get a general idea about how these sensors work and what kind of data they may provide.

2.2.1 Accelerometer

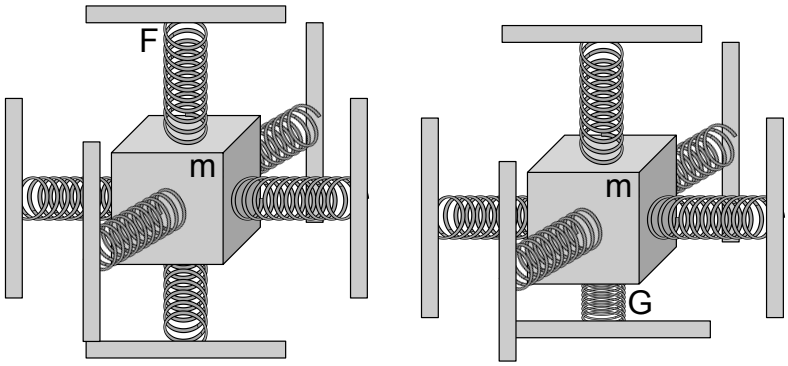
An accelerometer measures the acceleration based on Newton's Second Law, which is more commonly known as the law of inertia, and it is described as,

$$F = ma$$

where, m is the mass of the object, F the force imposed on the object, and a the result acceleration. In practice, an accelerometer has a proof mass in the center, and first measures the force imposed on it, then deduce the acceleration with the above equation. Fig. 2.3 shows the most primitive form of an accelerometer initialized with free fall state and gravity, respectively. The displacement of the center mass causes transformation of the springs, who then reports the force over different axes, and the acceleration is computed thereafter. In modern devices, the springs are replaced with electronic units such as capacitors and piezoelectric sensors [32], and can be manufactured into extremely small pieces. These sensors that use micro-level electronic parts instead of traditional mechanical parts are usually referred to as the *MEMS (micro-electro-mechanical systems)*.

2.2.2 Gyroscope

A gyroscope measures the device's angular velocity. A primitive gyroscope with mechanical parts is shown in Fig. 2.4a. The inner most disc (the rotor) rotates rapidly around the central spin axis, and due to the *inertia of momentum*, the pointing direction of the rotation axis will stay unchanged if no external torque is applied. This characteristic gives the gyroscope a reliable reference of direction. The outermost frame is usually mounted to a fixed



(a) (b)

Figure 2.3: Primitive accelerometers. (a) Free fall state; (b) With gravity.

pivot base, and can only rotate around the base's axis, hence has only one degree of freedom of rotation. The middle gimbal can rotate both around its own axis and the base's axis, thus has two degrees of freedom of rotation. Similarly, the innermost gimbal has three degrees of freedom of rotation. This structure then enables the central disc to rotate freely around three axes in the space. The angular velocity along all three axes can then be measured.

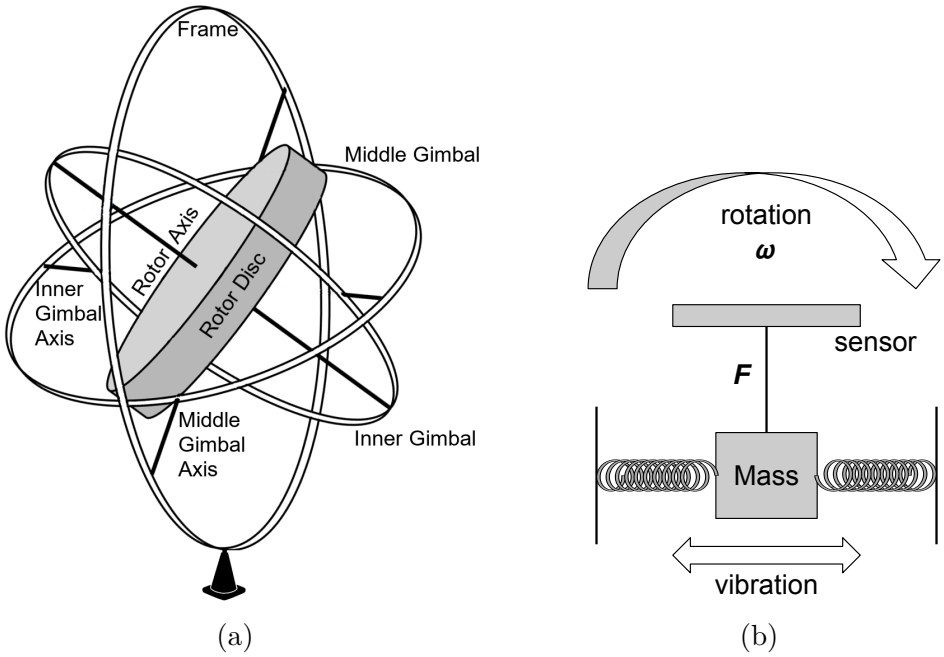


Figure 2.4: Gyroscope. (a) Primitive version with mechanical parts; (b) MEMS version taking use of the Coriolis effect.

In modern mobile devices, the MEMS version of the gyroscope usually has a vibrating structure and employs the *Coriolis effect*, as illustrated in Fig. 2.4b. The central mass vibrates constantly and when the host device rotates, the mass tends to keep on vibrating in the old plane, and a Coriolis force can be detected by the adjacent sensor. Then the angular velocity can be deduced from the following equation.

$$\boldsymbol{F} = -2m\boldsymbol{\omega} \times \boldsymbol{v}$$

where F is the measured Coriolis force, m the mass of the vibrator, $\boldsymbol{\omega}$ is the angular velocity and $\boldsymbol{v}$ the velocity of the vibrator with respect to the host device.

2.2.3 Magnetometer

A magnetometer senses the magnetic field around the device, and provides information about its strength, direction, and changes. Most magnetometers first detect the *Lorentz force*, then deduce the magnetic field from it. The *Lorentz force* is explained in the following

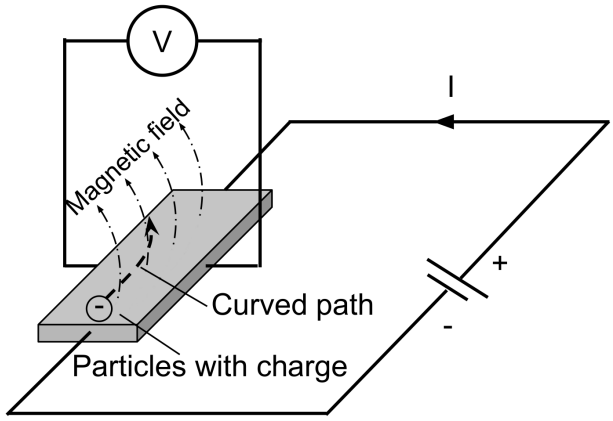


Figure 2.5: A magnetometer using the Hall Effect.

equation. When a particle of charge q moves with velocity $\boldsymbol{v}$ in an electric field $\boldsymbol{E}$ and magnetic field $\boldsymbol{M}$, the particle experiences a Lorentz force $\boldsymbol{F}$, where

$$\boldsymbol{F} = q(\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{M}).$$

Although Lorentz force is experienced by the particles of charge at the micro scale, it leads to a few phenomena that can be easily observed and measured. For example, when there is no magnetic field at presence, the particles with charge will move along a straight path; when a magnetic field is present, these particles will move along a curved path due to the Lorentz

force and many of them will collide and accumulate on one side of the path, other particles with opposite charges will accumulate on the opposite side of the path. This will establish a steady electric potential between the two sides of the path which can be measured. This phenomenon is named the *Hall Effect* and is widely used to detect and measure the magnetic field, as illustrated in Fig. 2.5.

2.2.4 Barometer

A barometer measures the atmospheric pressure. Traditional barometers involve two types, the liquid barometer and the aneroid barometer. Liquid barometers have a vertical tube filled with liquid (usually water, mercury or oil), which is directly connected to a reservoir whose surface is exposed to the open air. The changes of atmospheric pressure are indicated by the height of the liquid in the cube. Aneroid barometers ship with specifically designed alloy capsules, which will expand or contract when the pressure on it changes. Such expansion/contraction is amplified by a lever and then presented on a reader. MEMS version of barometers takes use of the piezoelectric sensors, which will gain an electric potential under pressure.

2.2.5 Light Sensor

A light sensor measures the ambient light level. It usually consists of a photodiode, which will absorb the incoming photons and generate electric current. After measuring the current, the light level can be computed.

2.2.6 Proximity Sensor

A proximity sensor detects if any object is near the sensor. Most mobile devices have optical proximity sensors (Fig. 2.6). Such proximity sensor has a transmitter which keeps

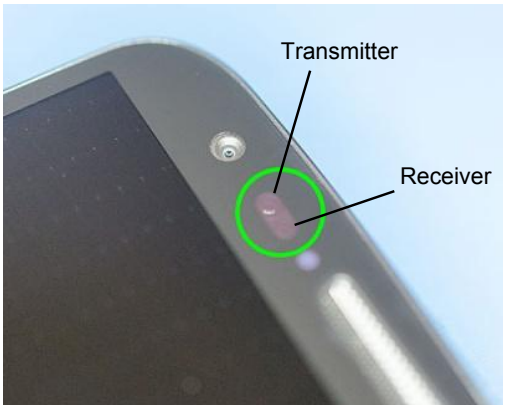


Figure 2.6: An optical proximity sensor on a smart phone.

emitting signals (usually infrared), and a receiver aligned beside the transmitter waiting for incoming signals. When there is no object in close proximity, the emitted signals will not reflect and the receiver detects nothing; when an object comes into the vicinity of the sensor, the emitted signal will get blocked by the object and reflected to the receiver, and the sensor then detects the presence of the object.

It is noteworthy that on some devices, the light sensor is used as the proximity sensor. When an object comes into the vicinity of the sensor, it also blocks the ambient light and causes an observable drop in the detected light level, which can be used as the indicator of the target object's presence.

2.2.7 Heart Rate Sensor

Heart rate sensors are mostly equipped on wrist-worn devices such as smart watches and wristbands. They consist of an LED that emits light onto the skin and a receiver that detects the reflection. Due to the pulse of the veins, the reflected light shows a periodic pattern which can be used to determine the heart rate. To avoid the interference from ambient light, it is usually suggested that the heart rate sensor be closely attached to the skin. Fig. 2.7 shows

a heart rate sensor in working condition on a smart watch, the green light is emitted from the aforementioned LED in the sensor.

Figure 2.7: A heart rate sensor in working condition on a smart watch.

2.3 Applications Employing the Sensors

The evolution of sensors on mobile devices has given rise to a wide range of applications. These applications can be roughly divided into three major categories: *Healthcare*, *User Input* and *Human Behaviors and Activities*. In this section, we summarize and introduce the most recent research work in these three categories.

2.3.1 Healthcare

Sensors on mobile devices can help users and researchers in the health related area from the following two aspects.

Provide alternative solutions for in-home health status assessing. Many of the health assessing tasks can hardly be accomplished at home because the professional medical

equipment involved are either not cost effective for personal use or lack of portability. Sensors on mobile devices, though not specifically designed for these assessing tasks, can provide health related information from multiple aspects. When various types of information is fused together, the user's health status in concern can be successfully inferred.

For example, Sugarmate is an application on smart phones that can assess the user's blood glucose level [33]. By collecting the accelerometer data, Sugarmate infers the user's physical activities; by fusing the data from accelerometer, light sensor and microphone, it infers the user's sleep quality; at last, by combining the activity and sleep data with the user recorded food and drug intake, it can infer the user's blood glucose level. MindfulWatch takes use of the accelerometer and gyroscope on a smart watch to monitor the user's breathing during a meditation, and based on that assesses the user's respiration condition [34]. HeartSense takes use of the gyroscope to achieve accurate heart-rate estimation on smart phones [35]. By combining the sensing data and usage pattern of the mobile phones, MoodExplorer manages to detect the compound emotion of the user [36]. Ben-Zeev et al. derives information from sensors on the smartphone and help to achieve psychiatric assessment.

Provide new possibilities for easing the lives of people with special health conditions. People with special health conditions suffer from various kinds of inconvenience in everyday life. For example, the elderly people face high risks of falling [37, 38]. Mobile sensors enable automatic fall detection and make possible of fast medical rescue [39–41]. People with schizophrenia need continuous monitoring of their psychiatric symptoms so that in-time intervention and treatment adjustment can be provided, and Wang et al. manage to accomplish such tasks with the help of sensors on mobile phones [42]. By combing data from the accelerometer and gyroscope, the difficulty of interacting with mobile phones for people with tremor can be effectively alleviated [43–46]. With the help of the microphone on mobile devices, researchers also developed applications that help people with hearing-loss to

gain better awareness of the surroundings [47,48]. By fusing sensory data on a smart watch, Mario A. Gutierrez manages to predict the blood alcohol level [49], thus helps people with drinking problems to improve their life styles.

2.3.2 User Input

With the fast development of various types of mobile apps, the need of user-device interaction is increasing rapidly, and many new challenges arise. For example, some smart phones are having large touch screens, which causes difficult for one-hand input. Among all the user input scenarios, text-entry is one of the most frequently encountered. However, due to the limited screen size, text-entry on mobile devices suffers from high mis-tap rate [5]. Furthermore, unlike a physical keyboard, the virtual keyboard used on mobile devices cannot provide effective feedback, which much increases the difficulty for users with visual impairment [50, 51]. Therefore, improving the input on mobile devices with the help of sensors on board is becoming a research direction that draws a lot of attention.

Blindtype explores eyes-free typing on a touchpad using one thumb [52]. It allows the user to tap on an imaginary QWERTY keyboard while receiving text feedback on a separate screen. By sensing high-resolution pressure, shear, and pinch deformations on a soft surface, DeformWear enables expressive and precise input for emerging mobile devices even without touch screens [53]. Shimon et al. explore non-touch screen gestures for smart watches to bypass the limitation of small screens [54]. By recording the user's input with the gyroscope, Change et al. propose novel interaction methods for one-hand input on smart phones with large screens [55]. WatchOut explores leverages the capability of sensors on smart watches to design a new family of input gestures like tap and swipe, which can be extended to the watch's case, bezel and band [56].

2.3.3 Human Behaviors and Activities

The most important use of sensors on mobile devices is to learn about the user's behaviors and activities. It is sometimes referred to as *activity recognition*, which is usually in close relation with *context sensing* and *human-computer interaction (HCI)*.

Mago senses the Hall Effect patterns caused by different transportation tools, and fuse it with the accelerometer to recognize the user's mode of transportation [57]. Guan et al. take use of the IMU sensory data from wearable devices and feed them into ensembles of deep LSTM (long short term memory) learners for activity recognition [58]. Bae et al. collect sensory data from mobile phones to identify drinking episodes of young adults [59].

Driving as a special human behavior which has heavy impact on safety also attracts lots of researchers' attention. SafeDrive uses smart watch sensors to detect abnormal user activities of the driver and help to reduce distracted driving [60]. Karatas et al. and SafeWatch apply similar idea to detect if the driver's hand is away from the steering wheel [61,62], and monitor the steering angle. V-Sense employs the phone's sensors to monitor the steering status of the vehicle and notify the driver when dangerous maneuvers are detected [63].

Benefiting from the fact that mobile devices are accompanying the users for almost all the time, **sleep tracking** becomes another typical sensing application on mobile devices. SleepMonitor takes use of the accelerometer on a smart watch to monitor the respiration rate and sleep positions [64]. Similarly, the Toss 'N' Turn system can detect sleep and wake states and analyze the user's sleep quality [65]. Aiming at providing the user with an unobtrusive way of sleep tracking, Chen et al. develop a system based on the best effort sleep (BES) model which can use the smart phone's sensors to track the user's sleep in an unobtrusive way [66].

Fitness and sports is another hot area where researchers leverage mobile devices to sense human activities. TrailSense collects gait-based sensory data about the hiker's walking

with a smartphone to help analyze the risks in mountain climbing and hiking [67]. Bajpai et al. leverage wearable devices to monitor the user's fitness level and calorie consumption [68]. RecoFit proposes to use wrist worn sensors to recognize and monitor strength-training activities, which can be well applied on wristbands and smart watches [69].

As the deep learning model gains success in many recognition related areas, More and more researchers start to feed sensory data to machine learning models to achieve robust activity recognition. Bhattacharya et al. employs the Restricted Boltzmann Machine to recognize human activities with smart watches [70]. Bajpai et al. take use of neural network to train and learn about the user's physical activities [68]. By using SVM (support vector machine), Kubo et al. combine the accelerometer data of both smart phones and watches to achieve context recognition, and manage to present dynamic UI (user interface) on the device accordingly [71]. Weiss et al. compare the recognition performance using machine learning algorithms with smart phones and smart watches, respectively, and conclude that smart watch based activity recognition are better suited for biomedical and health applications [72].

Chapter 3

LocMe: Human Locomotion and Map Exploitation based Indoor Localization

3.1 Introduction

Indoor localization is one of the most popular services on mobile devices. The cutting-edge indoor localization techniques can reach decimeter-level accuracy. However, these techniques require widely deployed infrastructure to work. They either rely on wireless access points (APs) with MIMO capability [6–9], or need densely deployed LED lights which are not widely available [10, 11]. These infrastructures may have existed in large cities, but are still absent in rural areas or less developed countries.

To realize infrastructure-free indoor localization, one can apply the Pedestrian Dead Reckoning (PDR) method, which monitors the inertial sensor readings and estimates the user's step length and heading directions, then updates the user's locations on a step basis. However, such method drifts drastically over time because of the accumulated error. A common approach to removing this drift is to apply a filter with special constraints, so that inaccurate estimations of the locations can be eliminated automatically. The most intuitive constraint is the wall-constraint, which rules out any step that goes through a wall [73–76]. Some other work cluster the user's special indoor activities as "landmarks" to help reduce the localization ambiguity between steps [77]. For this kind of indoor localization systems, two challenges remain open: 1. The converging speed for locating the user is not fast. Simply

imposing the wall-constraint or activity landmarks is effective but not efficient for eliminating invalid locations. 2. The system can not automatically load new maps upon floor changes. Existing solutions for floor detection either require crowd-sourcing and a central server for collecting and distributing information [78, 79], which is not viable in an infrastructure free scenario, or need extremely detailed information of the building (e.g. the moving time of the elevator between floors, the number of steps of the staircases [80]), which is beyond the scope of an indoor map and cannot be retrieved without massive manpower.

Aiming at the two challenges, we propose *LocMe*, an indoor localization service based on locomotion state detection and map exploitation. To our best knowledge, *LocMe* is the first effort toward synergizing the wall-constraint and locomotion-constraint to speed up the convergence of the infrastructure-free indoor localization and achieve better accuracy.

LocMe is motivated by several important observations. First, indoor maps are widely available today with rich annotations for places of interest (POIs), such as staircase, elevators, escalators, etc. Second, in the indoor environment, a user could express a set of locomotion states (e.g. walking, turning, taking escalators or elevators). Each state can only take place within a limited set of POIs whose coordinates are already given by the map. For example, if the user is detected to be moving vertically, his/her possible locations must be in an elevator. Third, the indoor POIs covers all possible paths leading to other floors, which means locating users at these places gives us information about floor changes. Based on that, we can infer which floor the user has travelled to, hence dynamically load the new floor map.

Bearing these observations into the design, *LocMe* utilizes the accelerometer, gyroscope, magnetometer and barometer, which are embedded in many off-the-shelf mobile devices, to distinguish different locomotion states, and employs them together with the wall-constraint to shrink the location search space and achieve automatic map reloading for new floors.

The rest of this chapter is organized as follows. Section 3.2 to 3.6 elaborates each module of *LocMe*. Section 3.7 evaluates *LocMe* in terms of convergence speed, effectiveness and

accuracy. Section 3.8 presents the background and related work of *LocMe*. Section 3.9 concludes this chapter.

3.2 System Overview

LocMe incorporates two essential ideas: 1. Use the embedded sensors on the mobile devices to distinguish the user's locomotion states. 2. Extract POI and wall information from the indoor map, then formalize the locomotion-constraint and wall-constraint to search for the user's location.

The architecture of *LocMe* is shown in Fig. 3.1. The underlying *SensorAgent* directly talks to the Android OS and collects the sensory data in need. *SensorAgent* preprocesses these data and pass them to *LocomotionDetector (LD)*, which is a collection of detectors that report the user's current locomotion state. The detected state is then fed into *LocalizationAgent*, which leverages the locomotion state and the POI information from *MapAgent* and applies a particle filter to pinpoint the current location of the user. *MapAgent* is responsible for extracting POI info from the vanilla map and loading in new maps upon the notification of the floor change from *LocalizationAgent*.

3.3 Sensor Agent

SensorAgent preprocesses the raw sensory readings by completing the following three tasks.

3.3.1 Coordinate transformation.

Accelerometer, gyroscope and magnetometer report readings with regards to the phone's local coordinate system (Fig. 3.2a). However, such coordinate system depends on the ori-

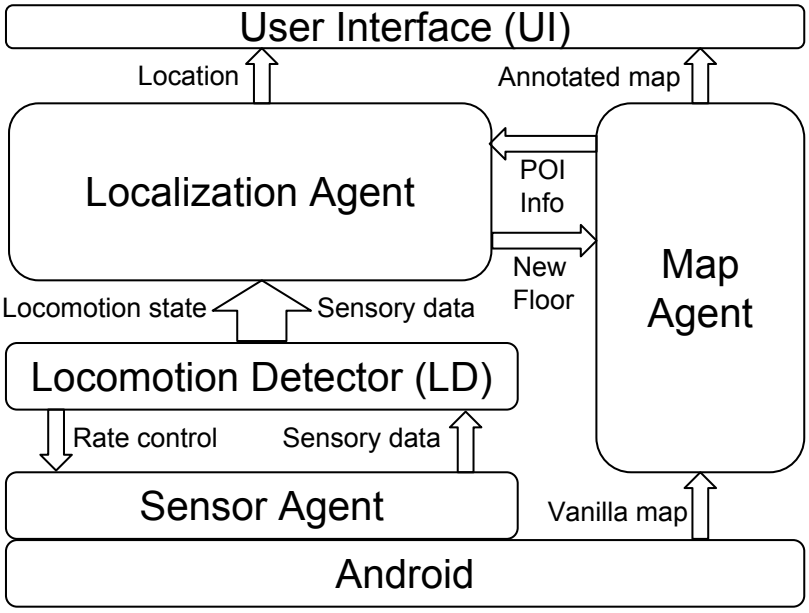


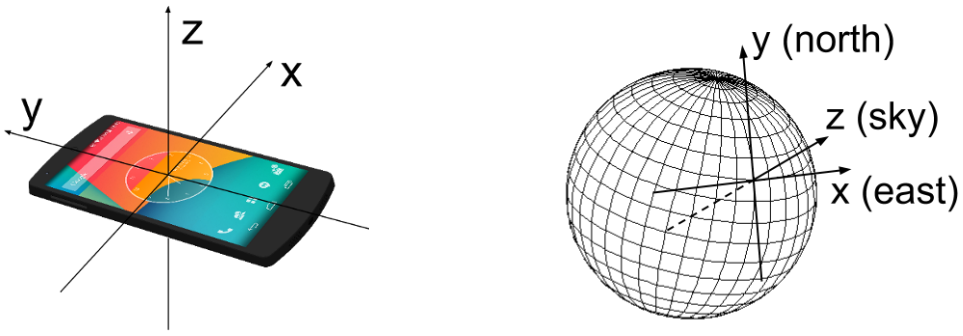
Figure 3.1: *LocMe* architecture.

entation of the phone and is subject to changes if the user carries the mobile device in different positions while walking around. To locate the user, we are actually interested in the world coordinate system (Fig. 3.2b). Thus, we always transform the raw sensory data from the phone's local coordinate system to the world coordinate system. This is achieved by employing the rotation vector reported by the Android system.

A rotation vector $\boldsymbol{r} = (x\sin(\frac{\theta}{2}), y\sin(\frac{\theta}{2}), z\sin(\frac{\theta}{2}))^\top$ means that, we can rotate the local coordinate system around $\boldsymbol{r}$ by θ to get the world coordinate system. If we denote the elements of $\boldsymbol{r}$ as r_x, r_y and r_z respectively, its corresponding rotation matrix R is computed as,

$$R = \begin{bmatrix} 1-2r_y^2-2r_z^2 & 2r_xr_y-2r_z\cos\frac{\theta}{2} & 2r_xr_z+2r_y\cos\frac{\theta}{2} \\ 2r_xr_y+2r_z\cos\frac{\theta}{2} & 1-2r_x^2-2r_z^2 & 2r_yr_z-2r_x\cos\frac{\theta}{2} \\ 2r_xr_z-2r_y\cos\frac{\theta}{2} & 2r_yr_z+2r_x\cos\frac{\theta}{2} & 1-2r_x^2-2r_y^2 \end{bmatrix}.$$

Given any sensory data in the local coordinate system, e.g. a 3D acceleration reading $\mathbf{a}$, its transformation $\mathbf{a_w}$ in the world coordinate system will be $\mathbf{a_w} = R\mathbf{a}$.



(a) Local. (b) World.

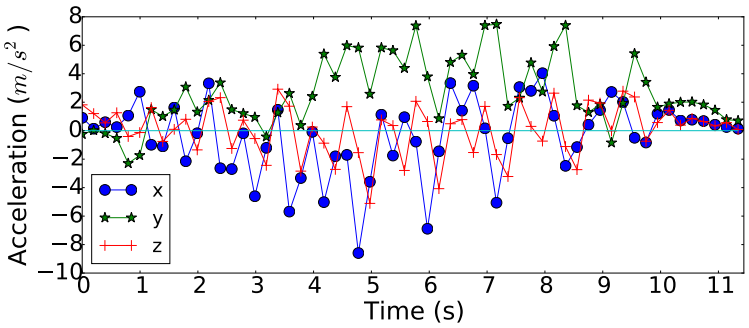
Figure 3.2: The two coordinate systems involved in *LocMe*.

3.3.2 Noise removal.

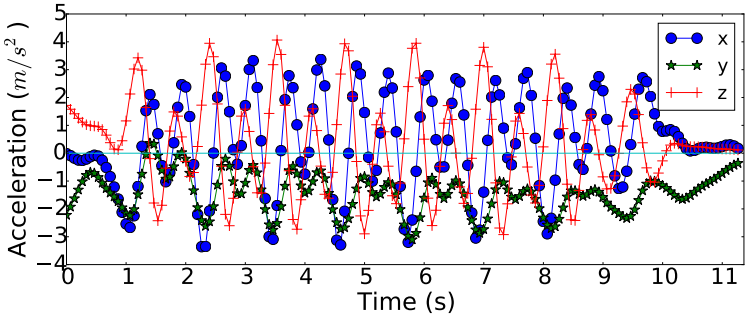
LocMe applies a 4th order low-pass Butterworth filter to remove noises, because our tests show that all the human locomotion states of *LocMe*'s interest are signals with relative low frequency. For example, the typical human walking frequency is around 2 Hz, and the vibration frequency of an escalator is mostly around 4 Hz.

3.3.3 Sensing Rate Control.

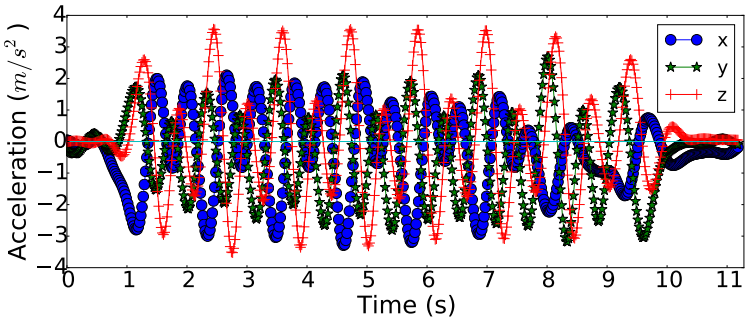
Off-the-shelf smartphones have different pre-defined options of the sensing rate. For Android, the standard sensing rates include 5Hz (Normal), 16.7 Hz (UI), 50 Hz (Game), and the fastest mode (the fastest sampling rate provided by the actual sensor hardware). Different levels trade-off monitoring granularity with energy consumption and processing time. As shown in Fig. 3.3, generally a higher sampling rate can capture more details, and after the noise removal (detailed in following paragraphs), it shows smoother curves and



(a) Low (Android-Normal): 5Hz.



(b) Medium (Android-UI): 16.7Hz.



(c) High (Android-Game): 50Hz.

Figure 3.3: Accelerometer readings with different sample rates (after low-pass filtering). The periodical pattern is well preserved with the medium sampling rate (b), while in contrast, at a low sampling rate (a) the periodicity becomes indistinguishable.

clearer patterns. *LocMe* dynamically adjusts the sampling rate to effectively preserve the device's battery. According to our tests, *LocMe* can operate under a sampling frequency of 16.7Hz, which is lower than that being used in previous studies (usually $\geq$ 50Hz) [73,81–86]. This avoids *LocMe* from draining the mobile device's battery in a short while.

3.4 Locomotion Detector (LD)

LD detects the indoor human behaviors in a two-level hierarchy. At the top level, human behaviors are divided into four *locomotion categories*: walking, taking elevators, taking escalators and static, as listed in the second header row of Table 3.1. Then, each category is separated into different *locomotion states* (the 3rd header row in Table 3.1).

LD walks through a decision tree illustrated in Fig. 3.4. It takes in the preprocessed sensory data from *SensorAgent*, and based on a few key characteristics to first decide the *locomotion category*. Upon a category decision, a corresponding finer detector will further determine the *locomotion state*. For example, a walking detector will further decide whether the user is walking on a level plane or going upstairs/downstairs.

POIs	**Locomotion**									
	Category	Static	Walking				Taking Elevators		Taking Escalators	
	State	static	level	downstairs	upstairs	turn	up	down	up	down
Stairs		✓	✓	✓	✓	✓	×	×	×	×
Elevators		✓	✓	×	×	✓	✓	✓	×	×
Escalators		×	×	✓	✓	×	×	×	✓	✓
Intersections		✓	✓	×	×	✓	×	×	×	×
Other		✓	✓	×	×	×	×	×	×	×

Table 3.1: Relations between the locomotion states and POIs. "✓" means the state is allowed in the POI, "×" otherwise.

Here we first present the following observations:

1. Upon each state change, either the *static* state or *level walking* state is always involved.

   This observation implies that *level walking* and *static* are bridging states that interme-

diate between other states. In another word, without the presence of a *level walking* or *static* state, no state can transit to other states.[1]

2. The only state that leads to the POI change is the *level walking* state.
   This observation is straightforward, and based on it, we can conclude that two states joined by a *static* state must occur in the same POI. LD defines a special event called *significant motion*, which is a short period of acceleration readings with high magnitude.

3. States in the walking category are always registered as significant motion, and states in non-walking categories are never registered as significant motion.

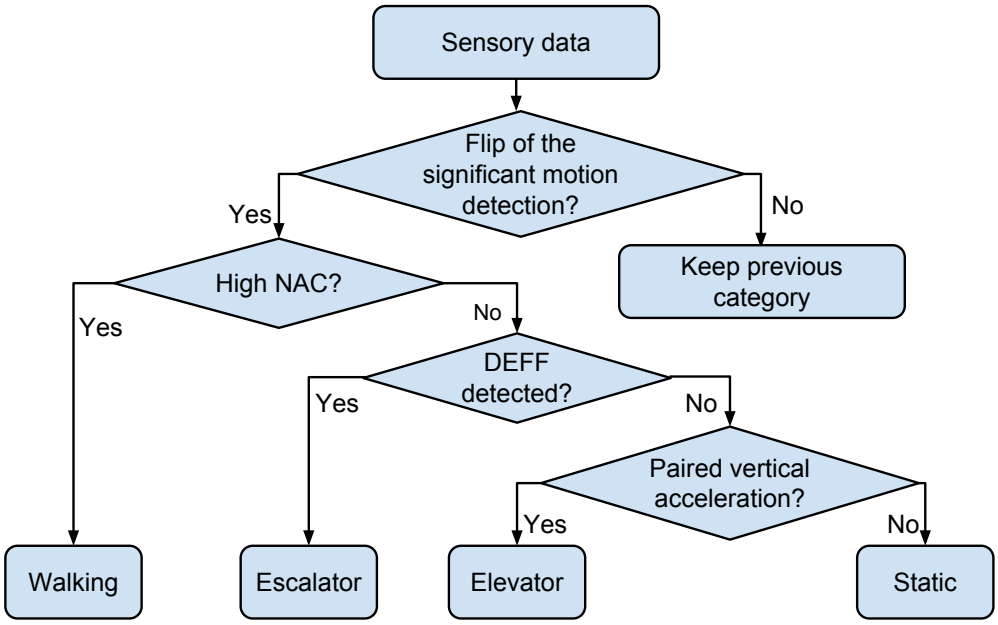


Figure 3.4: The decision tree used by *LocMe* to distinguish different locomotion categories. DEFF: Dominant Escalator Featured Frequency, a special frequency pattern observed only in acceleration readings of escalators.

[1]We ignore unusual cases such as turning around and reversing directions when walking on a flight of stairs.

Based on these observations, at any moment, if the significant motion state is not flipped, LD thinks that the user's locomotion category is not changed, as shown in the top right branch of Fig. 3.4. Such policy has two advantages:

1. Avoid non-necessary reports of state changes. For example, a user is taking an elevator upward to the 5th floor, and the elevator stops at the 3rd floor while other people coming inside. The user's state is now changed to static, but such state change is meaningless to *LocMe* because the user's location is not changed and he/she will definitely keep on taking this elevator, hence remain in the previous locomotion state, until the destination floor. Such state changes will not flip the significant motion status, thus will be ignored.

2. Avoid arbitrary user movements. The significant motion will only be detected when the accelerator readings reach a certain amplitude over a time period, this can rule out most arbitrary user movements.

In the following sections, we explain the detector for each locomotion category and the rest part of the decision tree.

3.4.1 Walking Detection.

The most important characteristic of human walking is the periodicity, which is absent in other locomotion categories. *LocMe* takes advantage of such periodicity to detect the walking state. To be more specific, *LocMe* maintains a history of the 3D-accelerations and calculates the normalized auto-correlation (NAC) over each of the three dimensions. NAC describes the similarity between a signal and a lagged version of itself. If a signal is periodical, given a lag which is close to the signal's period, the NAC will be close to 1. NAC is widely adopted for walking analysis [73, 87, 88]. The NAC of an acceleration signal $a(n)$ at k, given a lag λ,

is defined as,

$$\mathrm{NAC}(k,\lambda) = \frac{\sum_{i=0}^{\lambda-1}[a(k+i)-\mu(k,\lambda)][(a(k+i+\lambda)-\mu(k+\lambda,\lambda)]}{\lambda\sigma(k,\lambda)\sigma(k+\lambda,\lambda)}.$$

Here, $\sigma(k,\lambda)$ and $\mu(k,\lambda)$ represent the standard deviation and mean of data series $[a(k), a(k+1), \ldots, a(k+\lambda)]$, respectively. According to our experiment and previous studies [73, 89], a typical two-step walking period is within 0.4∼2 s, given the current sampling rate f of *LocMe*, we can constrain the range of lag λ in $[0.4f, 2f]$, and by searching for the largest NAC in this range, the two-step walking period T can be found by

$$T(k) = \frac{1}{\underset{\lambda\in[0.4f,2f]}{\mathrm{argmax}}\ \mathrm{NAC}(k,\lambda)}. \tag{3.1}$$

LD stores the acceleration readings since the last time the existence of significant motion has flipped, as $a(n)$, and dynamically computes the largest NAC to decide whether the user is walking. Furthermore, using the λ corresponds to the largest NAC, *LocMe* also detects the steps.

Theoretically, the NAC over all three dimensions should be at a high level. But in reality, we observe that only the vertical dimension always expresses a high NAC (>0.8), and the NAC of the other two dimensions on the horizontal plane are usually not as high as expected (average around 0.5). The reason is that the horizontal factors of acceleration are much less significant than the vertical one. When the user is a soft walker, or is having the phone in a loose pocket, the horizontal factors of acceleration may be even less distinguishable, hence more sensible to interference. Thus, *LocMe* only requires the NAC of vertical acceleration to be at a high level (>0.8 as in the current setting), and the average of the other two dimensions to be larger than a loose threshold (0.5 in current settings).

After confirming the user is walking, *LocMe* further breaks it down into four possible states: *level walking*, *walking upstairs*, *walking downstairs* and *turn*. It is worth noting that while the first three walking states are mutually exclusive, the turn state cannot exist by itself and must be accompanied by one of the other three walking states.

Level/Upstairs/Downstairs. *LocMe* distinguishes these three states by analyzing the barometer readings. A barometer measures the atmospheric pressure which corresponds to the altitude. The barometer readings cannot be used directly because they change constantly even at the same place due to the weather and air flow. Luckily, while detecting different walking states, *LocMe* is only interested in the altitude changes on a step basis, which is at a small scale both in time and space. At such a small scale, the interference from the weather change can be ignored. To cancel out the interference caused by the air flow, *LocMe* uses the difference of readings from two consecutive steps at the same stage of walking (i.e., heel-strikes).

Ideally, if the user is walking upstairs/downstairs, each step should monotonically lead to a lower/higher barometer reading. However, this monotonicity is affected by different phone positions. The reason is that during a gait cycle (two consecutive steps), the two legs enter the stance phase (supporting the body) and the swing phase (rotating around the pelvis) alternatively. When one leg is in the swing phase, it registers much greater change of sensor reading than the other leg which is in the stance phase. We test three kinds of phone positions: in pant pocket, in shirt pocket, and held in hand, and find that when the phone is in the pant pocket, it will record such difference between two consecutive steps (Fig.3.5). In contrast, the upper body is relative stable, and if the device is put in the upper pocket or held by the user in hand, not much step-wise difference will be recorded. Being aware of this, we monitor the barometer readings of every two steps (i.e., consecutive steps of the same leg). And if the corresponding curve expresses monotonicity, *LocMe* concludes the user

is walking upstairs/downstairs. After that, whether it is up or down is derived from the sign of changes of barometer readings.

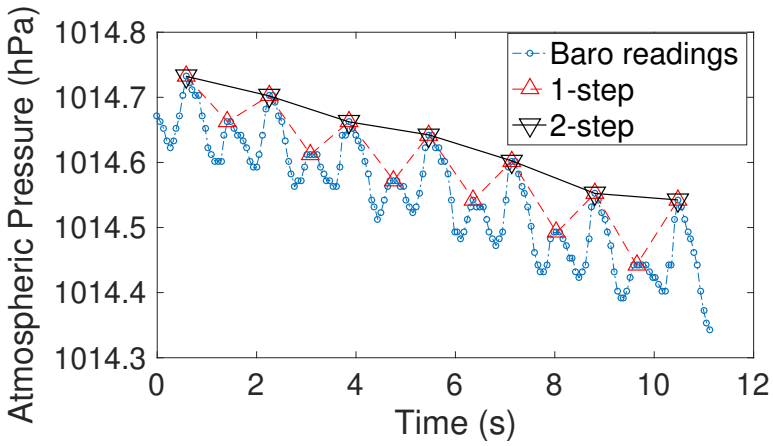


Figure 3.5: Barometer readings for walking upstairs. When the device is put in the pant pocket, the curve on a 1-step basis does not express monotonicity. Instead, the curve sampled on a 2-step basis (solid line) shows clear monotonicity.

Turn Detection. To detect the turn state, *LocMe* first denotes each intersection as a set of vertices and edges, as shown in Fig. 3.6. Upon the detection of a step, if it crosses the edge of an intersection (the dashed line in Fig. 3.6), *LocMe* regards it as the *entering step* and begins to monitor the changes of the user's heading direction; Following that, when a new step crossing another edge of the same intersection, which is called the *leaving step*, is detected, *LocMe* will go through the following procedure to decide whether a turn has been made.

First, *LocMe* checks if the two edges that the user has stepped across are jointed. If not, the user is just walking straightly across the intersection. Second, *LocMe* resorts to the magnetometer to monitor the changes of the user's heading direction. Although the magnetometer is known to suffer from interference from various electronic equipment widely existed in an indoor environment [90], it is intuitive to mitigate such interference by only

calculating the direction change between consecutive readings sampled in a short period. A problem here is that the device's heading may vary at different stages of the gait cycle, and not necessarily corresponds to the user's heading direction. However, it is reasonable to assume that the heading of the device at the same stage of two consecutive gait cycles stay the same, hence their difference is an accurate estimation of the change of the user's heading directions at that two moments. Therefore, *LocMe* only calculates the difference of magnetometer readings at two continuous heel-strikes of the same leg after the entering step. Once the user walks out of the intersection, all the recorded direction changes are summed into one angle (denoted as θ_u). Then *LocMe* compares it with the intersection's turning angle θ (Fig. 3.6) obtained by the *MapAgent* beforehand. θ is formed by the wall which the user is turning toward and the extending line of the wall which the user is turning away from. If $\theta_u \geq \theta$, *LocMe* concludes that the user has made a turn.

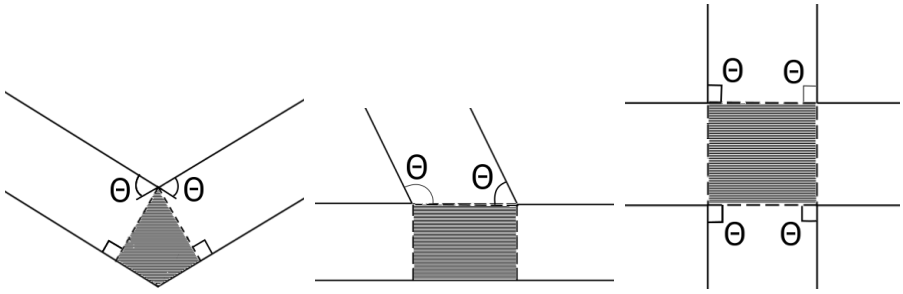


Figure 3.6: Three types of intersections. The shadowed parts are considered as the intersection area, and θ are possible turning angles formed by one passage wall and the extending line of the jointed wall of the other passage.

In the field tests, we observe that, at the leaving step, many users have not fully turned across the central line of the next passage yet. Therefore *LocMe* applies two relaxation schemes: 1. θ_u is computed at the subsequent step of the leaving step; 2. *LocMe* only requires $\theta_u \geq 0.9\theta$. Also, the detection of an entering step could be a false positive, in which case *LocMe* will never find a corresponding leaving step, unless it mistakes a new entering

step as the leaving step. To avoid such mistakes, *LocMe* will ignore an entering step if no leaving step is detected within the subsequent n steps, where n is dependent on the size of the intersection. These measures effectively increase the precision of the turn detection by 32%.

Step Detection. LD also detects the steps after confirming the user is walking. It maintains a cache queue C with length λ, which corresponds to the largest NAC in Eq. 3.1, and stores the sensory readings of the same step in C. Any time λ is updated, C is extended or shrunken correspondingly. Upon each new acceleration reading, LD deals with the following cases:

- If the walking state persists, enqueue the new readings at the end of C, remove the oldest ones in C if it's full.
  - If a peak in C is found, the peak is regarded as a heel-strike. The peak's timestamp, the current locomotion state and all the sensory readings in the cache, are reported to the *Localization Agent*. Then, LD empties C.
  - if no peak is found, C is kept unchanged.
- If the walking state does not continue, C is emptied.

3.4.2 Escalator Detection.

We investigate a total number of 75 escalators spread in metro stations, shopping malls and other public buildings in our city, and find that most of them express a sole dominant vibration frequency within the range 3.5∼4.5 Hz, as shown in Fig. 3.8a. This vibration is caused by the meshing between the step-chain and the sprocket which drives the chain (Fig. 3.7). Interestingly, this frequency is not observed in any other indoor behaviors. We

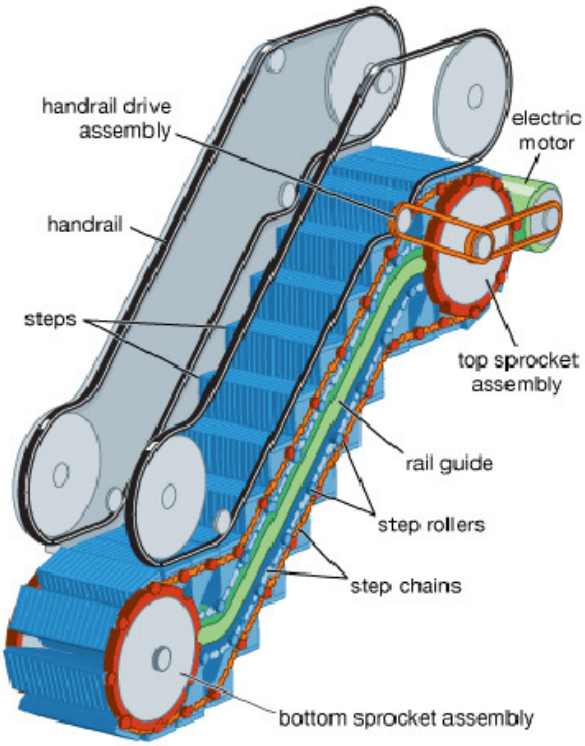
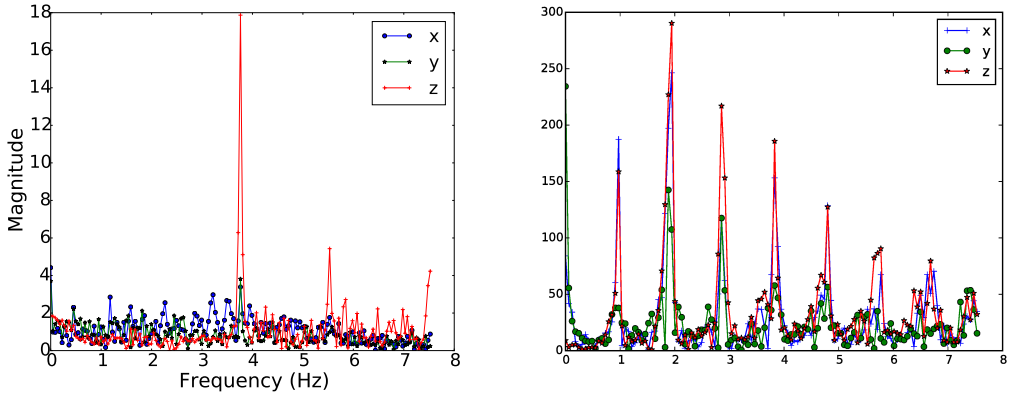


Figure 3.7: A diagram by Encyclopaedia Britannica, Inc. showing the mechanical structure of an escalator [91]. The step chain meshes with the sprocket when the escalator is moving, and causes constantly vibration to users on it.

name such special dominant frequency as Dominant Escalator Featured Frequency (DEFF). *LocMe* detects whether a user is taking escalator by examining the existence of DEFF. In contrast, Fig. 3.8b shows the frequency domain of acceleration for walking. Though it also has a peak within 3.5∼4.5 Hz, it is not dominant in comparison with other harmonics.

The barometer readings are used to determine whether the escalator is going up or down. Because the vibration of the escalator is at a low amplitude, its interference to the barometer is negligible. In our tests, the barometer readings express clear monotonicity while the user is taking an escalator. To make the detection result more robust, *LocMe* only uses the difference of barometer readings at the start and the end of the escalator.



(a) (b)

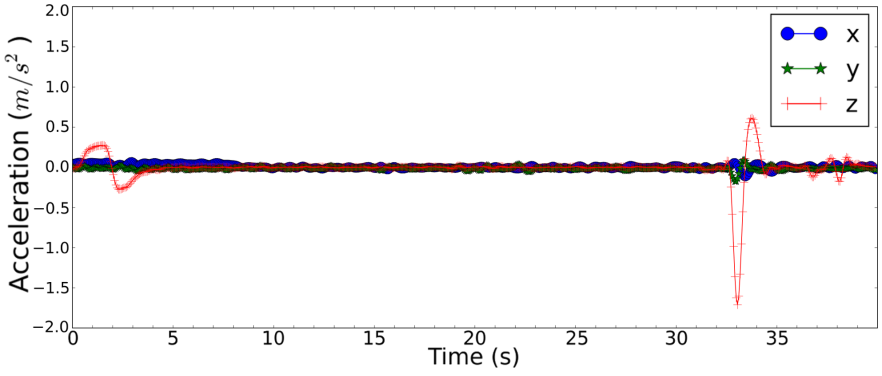
Figure 3.8: Frequency domain of the acceleration signals. (a) Taking an escalator. DEFF locating within 3.5∼4.5 Hz is clearly observed. (b) Periodic patterns for walking, with a lot of harmonics.

3.4.3 Elevator Detection.

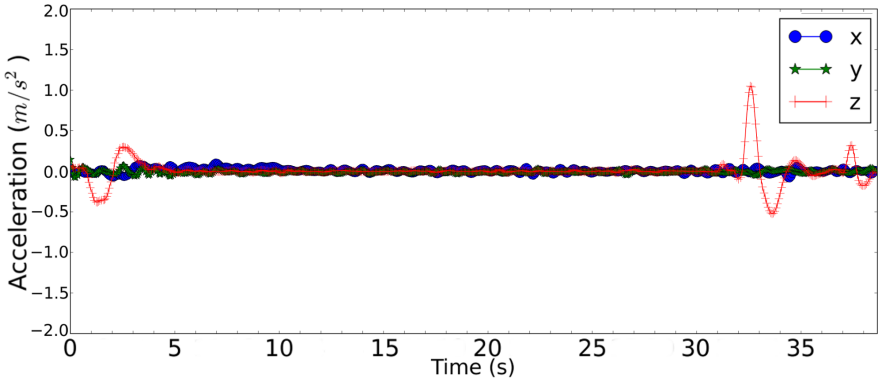
The dominant motion experienced by the user in an elevator is along the vertical axis, with three distinctive phases: activation, equilibrium and deactivation. Through the three phases, the accelerator readings have the three following features:

- The acceleration factors on the horizontal plane average at (or very close to) zero.
- The vertical acceleration averages at (or very close to) zero during the equilibrium phase.
- The vertical acceleration experiences a peak-valley pair in both the activation and deactivation phases, and the peak-valley order is reversed in the two phases.

The three phases are illustrated in Fig. 3.9. Combine the three features, even though the elevator does not register high amplitude of acceleration, it can be well separated from those noise readings caused by arbitrary human movements. Also, by utilizing these features,



(a) Up



(b) Down

Figure 3.9: Accelerometer readings for elevators.

LocMe can recognize the cases when the elevator stops for other passengers getting in and out. A user will be regarded as out of the elevator only when a *level walking* state is detected afterwards.

3.4.4 Static State Detection.

If all the previous detectors report negative result, and both the mean accelerometer readings and the first-order derivative of barometer readings stay close to zero over time, *LocMe* will report a *static* state.

Each time a new step or state change is detected, LD reports to the *LocalizationAgent*,

hence triggers a location update. The next section explains how *LocalizationAgent* works.

3.5 LocalizationAgent

LocalizationAgent fulfills the two major tasks of *LocMe*: location updating and floor detection.

3.5.1 Location Updating.

LocalizationAgent applies the particle filter algorithm discussed in [76] to update the user's location. Similar algorithms have also been adopted and proven to be effective by previous works [73–75, 80]. Particularly in *LocMe*, we incorporate the locomotion constraint to further speed up convergence of the algorithm.

A particle filter maintains a set (denoted as *LocationSet*) of N weighted points to sample the user's location state space. In this set, each potential location is represented as a particle $P_i = (x_i, y_i, s_i, d_i, state_i, w_i)$, where (x_i, y_i) is the 2D coordinates, s_i is the user's stride length, d_i is the walking direction, $state_i$ is the locomotion state reported by LD and w_i is the particle's weight.

Initialization. At the beginning, *LocMe* generates N particles with the same weight, where $w_i^{(0)} = 1/N,\ i = 1, 2, \ldots, N$. Depending on the available initial information, there are different cases that *LocMe* may boot up from. First, a complete set of initial information is given as $P^{(0)} = (x^{(0)}, y^{(0)}, s^{(0)}, d^{(0)}, state^{(0)}, w^{(0)})$, then the N particles are just N copies of $P^{(0)}$. Second, only a part of the initial information is known. If the starting position is unknown, then the positions for the N initial particles will be uniformly picked from the allowed positions on the current floor. If the initial step length $s^{(0)}$ or heading direction $d^{(0)}$

is unknown, they will be uniformly picked from a default range. If the initial state of the user is unknown, it's assumed as *static*.

Updating. The particles are updated upon the detection of a step or a state change.

Upon the kth update, if a new state $state^{(k)}$ is reported, *LocMe* simply assigns it to each particle, so that,

$$P^{(k)} = (x^{(k-1)}, y^{(k-1)}, s^{(k-1)}, d^{(k-1)}, state^{(k)}, w^{(k-1)}).$$

If a step is detected, then the probability of getting the new state $P^{(k)}$ is computed as,

$$\begin{aligned} p\left(P^{(k)} \middle| P^{(k-1)}\right) &= p\left(x^{(k)}, y^{(k)}, s^{(k)}, d^{(k)} \middle| x^{(k-1)}, y^{(k-1)}, s^{(k-1)}, d^{(k-1)}\right) \\ &= p\left(s^{(k)}, d^{(k)} \middle| x^{(k-1)}, y^{(k-1)}, s^{(k-1)}, d^{(k-1)}\right) \\ &\cdot p\left(x^{(k)}, y^{(k)} \middle| s^{(k)}, d^{(k)}, x^{(k-1)}, y^{(k-1)}, s^{(k-1)}, d^{(k-1)}\right) \\ &= t_2\left(\begin{bmatrix} s^{(k)} \\ d^{(k)} \end{bmatrix} \middle| \begin{bmatrix} s_i^{(k-1)} \\ d_i^{(k-1)} + \delta_i^{(k-1)} \end{bmatrix}, \begin{bmatrix} \sigma_s^2 & \\ & \sigma_d^2 \end{bmatrix}\right) \\ &\cdot t_2\left(\begin{bmatrix} x_i^{(k)} \\ y_i^{(k)} \end{bmatrix} \middle| \begin{bmatrix} x_i^{(k-1)} \\ y_i^{(k-1)} \end{bmatrix} + s_i^{(k)} \begin{bmatrix} \cos(d_i^{(k)}) \\ \sin(d_i^{(k)}) \end{bmatrix}, \begin{bmatrix} \sigma_x^2 & \\ & \sigma_y^2 \end{bmatrix}\right), \end{aligned} \tag{3.2}$$

where t_2 is the density function of Student's t distribution with 2 degrees of freedom; δ_i is the changes of heading direction detected by the phone using the method proposed in [73]; the standard deviations σ_s, σ_d, σ_x and σ_y are configurable system parameters.

Therefore, when the kth update is a step detection, *LocMe* uses the following distribution

to sample and update the ith particle's position, step length and heading direction.

$$\begin{bmatrix} s_i^{(k)} \\ d_i^{(k)} \end{bmatrix} \sim T_2 \left(\begin{bmatrix} s_i^{(k-1)} \\ d_i^{(k-1)} + \delta_i^{(k-1)} \end{bmatrix}, \begin{bmatrix} \sigma_s^2 & \\ & \sigma_d^2 \end{bmatrix} \right) \tag{3.3}$$

$$\begin{bmatrix} x_i^{(k)} \\ y_i^{(k)} \end{bmatrix} \sim T_2 \left(\begin{bmatrix} x_i^{(k-1)} \\ y_i^{(k-1)} \end{bmatrix} + s_i^{(k)} \begin{bmatrix} \cos(d_i^{(k)}) \\ \sin(d_i^{(k)}) \end{bmatrix}, \begin{bmatrix} \sigma_x^2 & \\ & \sigma_y^2 \end{bmatrix} \right),$$

where T_2 is the Student's t distribution with 2 degrees of freedom. *LocMe* uses the Student's t distribution because its heavy tail can help the particle cloud to quickly increase the coverage of the search space, so that the system faces lower pressure of resampling (explained later).

After each update, *LocalizationAgent* checks the following two constraints.

- **Wall-constraint**. Any user can not walk through a wall. Thus if any two consecutive location points form a line through a wall, the latter added location is highly unlikely to be valid. *LocMe* uses subscript W to denote variables related to the wall-constraint and Algorithm 1 to validate it.

- **Locomotion-constraint**. The current locomotion state and location have to follow the relation listed in Table 3.1. Because both the number and area of the POIs are usually small, this constraint can effectively reduce the search space and speed up the convergence of the particle filter. *LocMe* uses subscript L to denote variables related to the locomotion-constraint and Algorithm 2 to validate it.

In the two constraints, we have ϵ_W and $\epsilon_{L,state_i^{(k)}}$ both satisfying $0 < \epsilon \ll 1$. Here ϵ is an extremely small value, which ensures that particles breaking the constraints get eliminated quickly. Meanwhile, ϵ is not zero, giving the constraint-breaking particles a chance to survive in case the constraint violation is a false positive. Particularly, $\epsilon_{L,state_i^{(k)}}$ is related to the performance of state detectors in LD. The more precise a state detector is, the smaller

Algorithm 1: Wall-constraint validation.

Input: $(P_i^{(k)}, P_i^{(k-1)})$, $WallSet$
Output: $w_{W,i}^{(k)}$
for $wall \in WallSet$ **do**
  if $(P_i^{(k)}, P_i^{(k-1)}) \cap wall! = \varnothing$ **then**
    $w_{W,i}^{(k)} := \frac{\epsilon_W}{1-\epsilon_W} w_i^{(k-1)}$
    break
  end
  $w_{W,i}^{(k)} := w_i^{(k-1)}$
end

$\epsilon_{L,state_i^{(k)}}$ is. If the kth update is a step update, *LocMe* will validate both the wall-constraint and the locomotion-constraint; if it is only a state update, then the wall-constraint will not get violated and only the locomotion-constraint will be validated. *LocMe* validates the constraints and updates each new particle's weight using Algorithms 1 and 2 accordingly.

Finally, the location updating algorithm is given in Algorithm. 3. As the user walks around, invalid location points will get excluded by the two constraints and $LocationSet^{(k)}$ eventually converges to points concentrating on a small area, then the weighted center of the set will be output as the localization result.

Algorithm 2: Locomotion-constraint validation.

Input: $P_i^{(k)}$
Output: $w_{L,i}^{(k)}$
poi :=getPOI($P_i^{(k)}$)
if $P_i^{(k)}.state \notin getAllowedState(poi)$ **then**
  $w_{L,i}^{(k)} := \frac{\epsilon_{L,state_i^{(k)}}}{1-\epsilon_{L,state_i^{(k)}}} w_i^{(k-1)}$
  break
else
  $w_{L,i}^{(k)} := w_i^{(k-1)}$
end

Algorithm 3: Location updating algorithm. α in line 14 balances the wall-constraint and locomotion-constraint.

Input: $LocationSet^{(k-1)}$, $WallSet$, $POISet$
Output: $LocationSet^{(k)}$, $\bar{\boldsymbol{\mu}}^{(k)}$

`/* Update the particles. */`

1 **for** $P_i^{(k-1)} \in LocationSet^{(k-1)}$ **do**
2 **if** *LD reports a new state* $state^{(k)}$ **then**
3 $state_i^{(k)} := state^{(k)}$
4 **else**
5 $state_i^{(k)} := state_i^{(k-1)}$
6 **end**
7 **if** *LD reports a step* **then**
8 Generate $(x_i^{(k)}, y_i^{(k)}, s_i^{(k)}, d_i^{(k)})$ using Eq. (3.3)
9 Use Algorithm 1 to get $w_{W,i}^{(k)}$
10 **else**
11 $(x_i^{(k)}, y_i^{(k)}, s_i^{(k)}, d_i^{(k)}) := (x_i^{(k-1)}, y_i^{(k-1)}, s_i^{(k-1)}, d_i^{(k-1)})$
12 **end**
13 Use Algorithm 2 to get $w_{L,i}^{(k)}$
14 $w_i^{(k)} := \alpha w_{W,i}^{(k)} + (1-\alpha) w_{L,i}^{(k)}$
15 $P_i^{(k)} := (x_i^{(k)}, y_i^{(k)}, s_i^{(k)}, d_i^{(k)}, state_i^{(k)}, w_i^{(k)})$
16 **end**

`/* Normalize the weight. */`

17 $w_{sum}^{(k)} := \sum_{j=1}^{N} w_j^{(k)}$
18 **for** $i = 1$ *to* N **do**
19 $w_i^{(k)} = \dfrac{w_i^{(k)}}{w_{sum}^{(k)}}$
20 **end**

`/* Output the current estimated location. */`

21 $\bar{\boldsymbol{\mu}}^{(k)} := \sum_{i=1}^{N} w_i^{(k)} \begin{bmatrix} x_i^{(k)} \\ y_i^{(k)} \end{bmatrix}$

Resampling. As the user walks around, many particles eventually become low-weighted and should be eliminated, and those with high weights should be replicated to impose more

influence on the localization result. This process is called *resampling*. *LocMe* uses the multinomial resampling method [92].

Upon resampling, *LocMe* computes the sum of all N particles' weights and denotes it as W. Then the range $[0, W]$ is divided into N segments, where the ith segment's length equals to the weight of the ith particle. Then N random numbers in the range $[0, W]$ are uniformly generated. For each random number, if it falls into the ith segment, the ith particle will be drawn once and be placed into the new particle set.

Resampling may lead to impoverished particle set, in which only a few heavy-weighted particles survived. In this case, the survived particles may be too concentrated to cover potential interested locations. To avoid such impoverishment, *LocMe* monitors the effective number of particle (ENP) [76]

$$N_{\text{enp}}^{(k)} = 1/\sum_{i=1}^{N} \left(w_i^{(k)}\right)^2,$$

where $w_i^{(k)}$ is the weight of the ith particle after the kth update. *LocMe* only resamples the particle set when $N_{\text{enp}}^{(k)}$ is smaller than a given threshold.

Complexity Analysis. In comparison to the existing particle-filter based localization algorithms, ours brings in additional complexity by checking the locomotion constraint. To understand its impact, we first clarify the complexity of checking the wall-constraint. Intrinsically, it can be formalized as a geometric intersection problem, where given n line-segments (walls and the target step), we want to verify if there are intersections between them. A classic solution is the sweep-line algorithm [93], whose computational complexity is $O(n \log n)$. Moreover, our problem is actually a simpler case, since we only care whether one line-segment (the target step) intersects with the others (walls). Thus the complexity can be reduced to $O(n)$.

For checking the locomotion-constraint, it consists of two parts. The first is the function $getPOI(P_i^{(k)})$, which checks which POI the potential location $P_i^{(k)}$ is in. It is a point-in-polygon problem, and can be solved by the simple even-odd-rule, i.e., form a ray from the current location point and check how many times this ray intersects with the POI polygon. If the number of intersections is odd, the location point is inside the POI, otherwise not. This part costs $O(1)$, since the number of edges of each POI polygon is fixed. The total cost for checking all m POIs is $O(m)$. The second part is verifying if the state of $P_i^{(k)}$ is allowed in the POI. Since the number of each POI's allowed states is already fixed, this part can also be done in $O(1)$ time. In total, the proposed algorithm brings in extra computational cost of $O(m)$. Because the number of POIs m is guaranteed to be smaller than the number of walls n, it is safe to conclude that the extra complexity of $O(m)$ in our algorithm is negligible.

3.5.2 Floor Detection.

The second essential task of *LocalizationAgent* is to detect whether a floor change has taken place, and if yes, which floor the user has come to. It is obvious that only through three places the user can travel to another floor: stairs, escalators or elevators, and these three POIs are often referred to as *connectors*. To achieve floor detection, *LocMe* must find answers to the following three questions.

1. When does the user enter and leave the connectors?

2. Is the user ended moving up or down?

3. How many floors the user has actually traveled?

LocMe cares for the first question, because the user may go up and down in an elevator or stairwell before leaving it. Then question 2 and 3 help *LocMe* to determine the final

floor number. In the following paragraphs, we elaborate how *LocMe* gets answers to these questions for different connectors.

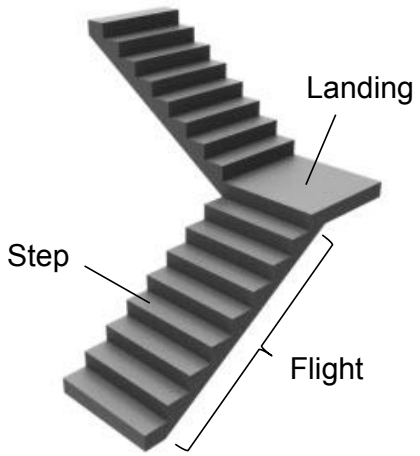


Figure 3.10: A diagram by Oakpointe showing the structure of a staircase [1].

Stairs. A staircase connecting two adjacent floors usually consists of a few *flights* of steps; each two *flights* are connected by a short horizontal platform called a *landing*, as illustrated in Fig. 3.10. We observe that, within the same stairwell, each floor has the same number D of *landings*. Although D is not directly available from the indoor map, it usually satisfies $D \leq 4$. In *LocMe*, each *landing* is detected as a continuous short series of *level walking* states. If such series is longer than a given threshold, *LocMe* will think the user has walked out of the stairwell. Suppose *LocMe* detects l *landings* and a total number of n stair steps, then the number of the floor change is determined as f, such that:

$$C_{\min} \leq \frac{nS}{f} \leq C_{\max}, \quad f \in \{l/D | D = 1, 2, 3, 4\} \cap \mathbb{N}.$$

Here $\mathbb{N}$ is the set of natural numbers; $C_{\min}$ and $C_{\max}$ are the minimal and maximal allowed ceiling height of a floor, and S the average riser height of the stair step. According to the

international building code [94], the height of the indoor stair riser is regulated to be within 102∼178 mm, thus we take the average and set the default S to 140 mm. The minimum ceiling height of each floor is defaulted at 2.1 m, and the default maximum is an empirical value (3.5 m). We believe it is reasonable that *LocMe* has knowledge about the user's current location in a large scale, for example, at the city level. Thus it is feasible for *LocMe* to ship a table of floor height regulations for different cities, and preload the local ones to replace these parameters here.

Escalators. An escalator always connects the same two floors. *LocMe*'s *MapAgent* will try to derive which floors are connected by a specific escalator from the indoor map. If this knowledge cannot be extracted, *LocMe* assumes that the escalator connects adjacent floors.

Elevators. *LocMe* uses the barometer readings of the user entering and exiting elevators to decide the floor change in an elevator. The user's entrance is determined as the latest timestamp of any walking state, and the exiting moment is determined as that of the first detected walking state after one or a continuous series of *taking elevator* state. Assume the barometer reading is P_1 upon the user's entrance of the elevator, and P_2 upon the user's exit, the elevation change h can be computed as

$$h = (-RT/g)\ln(P_2/P_1),$$

where T is the temperature, R is the universal gas constant, g is the gravitational acceleration. With h, the floor difference f is acquired by $f = \lfloor (h/H) \rceil$, where H states the average height of a floor in the current building.

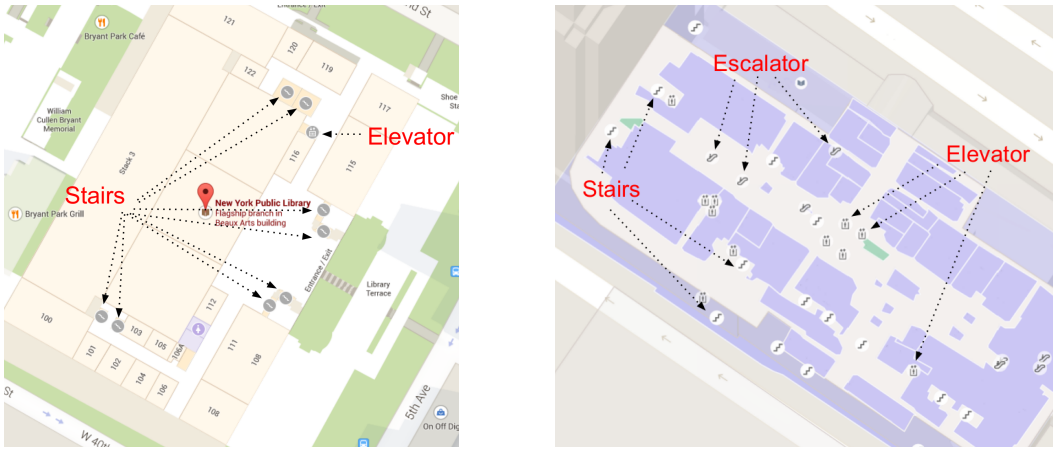
By default, *LocMe* uses empirical value for T (22 C°) and H (2.4 m). Such default settings do not perform well, because T varies over different time of the year and heating

conditions, and H is dependent on building type. However, we discussed before that *LocMe* can check a floor height regulation table to find a more reasonable local H. For T, once the user has used the stairs or escalator, *LocMe* can employ atmospheric data and floor detection results from them to derive a more accurate T. By updating H and T, the 75th percentile of floor detection error of elevators can be reduced to 0 (Fig. 3.12a).

3.6 Map Agent

Indoor maps are becoming widely available. On one hand, leading map service providers are constantly collecting and releasing indoor maps for major public buildings such as shopping malls and airports. On the other hand, detailed floor plans exist for the purpose of evacuation during emergencies, and these floor plans are usually open to public. As an example, Fig. 3.11 shows indoor maps of public buildings we obtained from Google Maps and Bing Maps. These maps contain rich POI information which can benefit *LocMe*.

Given a raw indoor map, *MapAgent* loads it in and fulfills the following two tasks.



(a) Google Maps. (b) Bing Maps.

Figure 3.11: Indoor maps hold rich information about POIs.

Map Alignment among Different Floors. When the user travels to a new floor, we want to map the existing particles directly onto the new floor map so the localization algorithm does not need to reboot. To achieve this, *MapAgent* uses the following three types of anchor points to align the maps of different floors: contour vertices of the building, the elevator and the stairwell. Because there must be at least one staircase connecting two adjacent floors, it is guaranteed that, besides the building's contour vertices, *LocMe* can always find at least 4 extra points to conduct the alignment.

Map Annotation.

- **Walls**: Upon getting an indoor map, *LocMe* stores each wall as a line segment $(start, end)$, where *start* and *end* are the coordinates of the two endpoints. For all walls connected along the same line, they are joined and stored as one long segment.

- **POIs**: For each POI, *LocMe* assigns it with a *contour* component which stores coordinates of all the vertices of the POI. Beside of that, some complementary information is also stored. For example, an elevator or stairwell is stored as a tuple $(contour, floors)$, where $floors$ is a list of floors this elevator/stairwell connects; an escalator is stored as $(contour, direction, startFloor, endFloor)$, where *direction* describes whether the escalator is going upwards or downwards, and $startFloor$ and $endFloor$ denotes the floor numbers that the escalator start and end with respectively. An intersection is stored as $(contour, angles)$, where *contour* is a list of edges which enclose the intersection area (dashed line in Fig. 3.6), and *angles* is a 2D array which stores the turn angles (θs in Fig. 3.6).

3.7 Evaluation

3.7.1 Locomotion Detector Performance

We recruited 17 test subjects to walk around different indoor POIs to test the performance of LD. For each subject, the ground truth of his/her locomotion state is manually recorded by another observer. These tests cover 42 staircases, 75 escalators and 31 elevators spread all over shopping malls, metro stations and campus buildings in Montreal. The test results are presented in Table 3.2.

3.7.2 Floor Detection Performance

The cumulative distribution function (CDF) of the floor detection error is shown in Fig. 3.12a. Escalator always gives the correct floor detection hence is not illustrated here. When using the default parameters to detect floor changes by elevator, only 20% detection results are correct. However, by using average floor height derived from local building regulation code and temperature derived from previous escalator/stairs data, this rate is significantly improved to 75%.

3.7.3 Field Test

We carry out a field test in a 70m×100m office building with 7 volunteers. Each volunteer is asked to travel through a path over three floors. The test building's indoor map is illustrated in Fig. 3.13, where the path is marked with black lines, and arrows indicate the volunteer's walking direction. The path length is about 500 steps, taking an average of 310 seconds. The tests are recorded on video, which provides us the ground truth.

Throughout the test, the number of particles N is set to 500, the threshold for N_{enp} is

Table 3.2: Performance of the Locomotion Detector.

(a) By locomotion category.

		Ground Truth			
		walk	escalator	elevator	static
Detection	walk	**93.5%**†	0%	0%	0%
	escalator	5.7%	**90.3%**	7.9%	0%
	elevator	0.8%	9.7%	**92.1%**	0%
	static	0%	0%	0%	**100%**

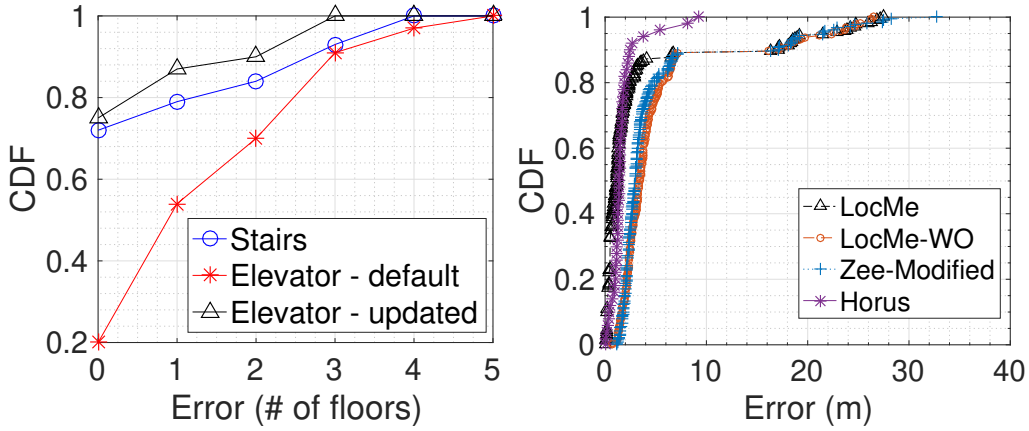
† All results here are computed as $\frac{\#\text{ of detection}}{\#\text{ of groundtruth}}$. Diagonal (bold) elements are the *recall*. Same in Tab.(b).

(b) By locomotion state.

			Ground Truth								
			walk				escalator		elevator		static
			level	upstairs	downstairs	turn	up	down	up	down	static
Detection	walk	level	**85.9%**	2.7%	3.5%	N/A	0%	0%	0%	0%	0%
		upstairs	3.2%	**91.7%**	0%	N/A	0%	0%	0%	0%	0%
		downstairs	3.6%	0%	**89.9%**	N/A	0%	0%	0%	0%	0%
		turn*	N/A	N/A	N/A	**91.4%**	N/A	N/A	N/A	N/A	N/A
	escalator	up	2.6%	5.6%	0%	N/A	**90.0%**	0%	8.5%	0%	0%
		down	3.4%	0%	5.5%	N/A	0%	**90.7%**	0%	7.3%	0%
	elevator	up	1.3%	0%	0%	N/A	10.0%	0%	**91.5%**	0%	0%
		down	0%	0%	1.1%	N/A	0%	9.3%	0%	**92.7%**	0%
	static	static	0%	0%	0%	N/A	0%	0%	0%	0%	**100%**

* Not like other locomotion states, the turn state is non-exclusive with other states in the same category. Thus here we only list the true-positive rate of its detection.

set to 50. The initial user position is set to unknown. The initial step length is randomly picked from $[0.7, 1.2]$ m, the initial heading direction is randomly picked from the range of $\pm 20°$ of the true direction. Although different test subjects have different walking habits, *LocMe* manages to detect their locomotion states and locate them along the path, outputting similar error distributions over time. Here we take one subject as an example to illustrate the effectiveness of the constraint synergy.



(a) Floor detection error.

(b) Location error.

Figure 3.12: CDF of the floor detection error and the location error for different algorithms over the test path.

As shown in Fig. 3.14, at first, the error is high because all particles are uniformly chosen and given the same weight. After the user makes the first turn (between 1 and 2 in Fig. 3.13(a)), *LocMe* immediately detects it and leads to a significant drop in the error. Later, when *LocMe* detects the elevator as the user exiting it, another abrupt drop of the error is observed. At this moment, the new floor number is also determined, and the new floor map is loaded with a known start point on it. This ensures that the particle filter can continue to work without reinitialization and also a low error level at the beginning of a new floor. Previous particle-filter based systems will fail at this point because they can

(a) (b) (c)

Figure 3.13: Test path in an office building. The numbers denote the order of checkpoints on the path. For example, the user starts from 1, walks to 2 on floor (a), takes the elevator down to floor (b), then walks through 3 to 4, etc. At the end, the user returns to 1. The dashed lines represent sub-paths which the users are free to choose.

not detect and load in the new floor map. As the user walks upstairs from 103s to 184s and downstairs from 213s to 228s, *LocMe* detects the corresponding states and confines the potential locations within the stairwell, maintaining a low error level. After the user leaves the stairwell (6 in Fig. 3.13(a)), there is a long period of level walking from 229s to 278s, and the error begins to accumulate and rise. In the end, the last turn (near 2 in Fig. 3.13(a)) is detected and the error is again reduced to a low level around 0.7m.

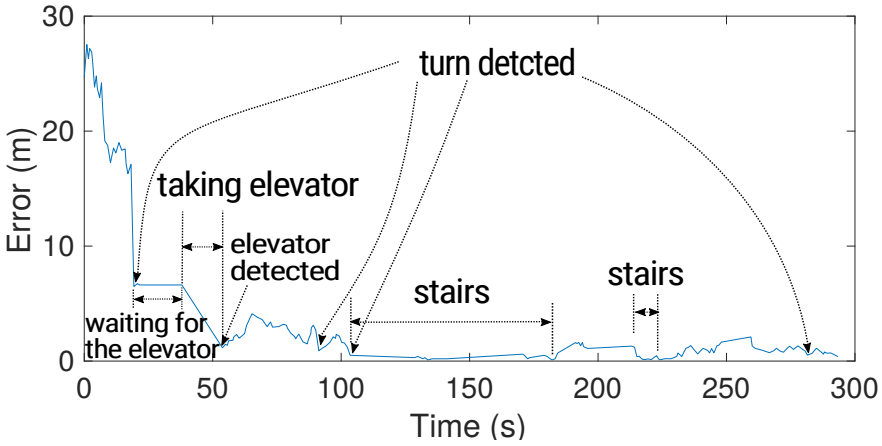


Figure 3.14: The localization errors over the test path.

Effectiveness of the constraint synergy. To illustrate how the synergy of constraints have sped up the particle filter's convergence, we implement a modified version of Zee [73] and wall-constraint-only version of *LocMe* (*LocMe*-WO) for comparison. First, we explain how Zee and *LocMe*-WO work.

Zee. Zee is an indoor localization system which aims at automating the collection of WiFi fingerprints [73]. It applies the wall-constraint to a particle filter to simply eliminate any point that violates the constraint. It also relies on the indoor map to get all the wall information, but cannot detect the floor changes and will fail when the user moves to a new floor. To make Zee function in a multi-floor environment, we directly feed the floor detection

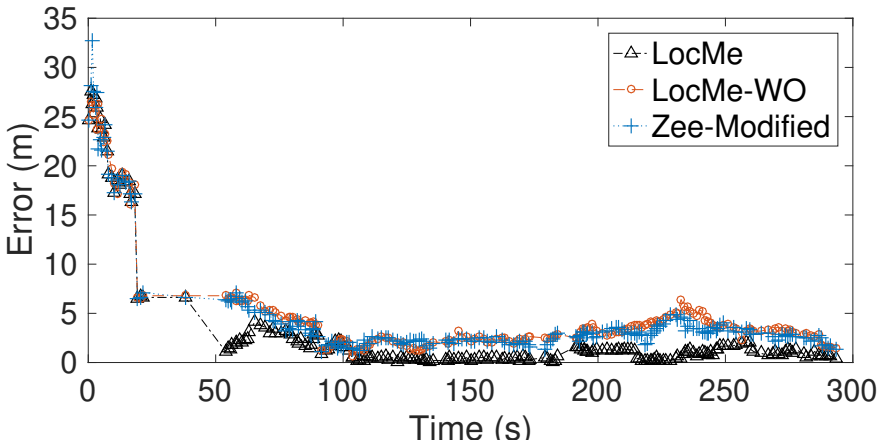


Figure 3.15: Comparison of the convergence of different particle filter based indoor localization methods.

result of *LocMe* to Zee. Upon a floor change, Zee will load the new map, and inherit all the existing particles and use them to initialize on the new map. We denote such a modified Zee system as *Zee-Modified*.

***LocMe*-WO**. It is a special version of *LocMe* that it will not update $w_{L,i}^{(k)}$ and will set $\alpha = 1$ in Algorithm 3, thus completely excludes the locomotion-constraint.

Fig. 3.15 illustrates the localization error over time for different particle filter based methods. It is clearly shown that, with the locomotion-constraint, *LocMe* converges faster than LocMe-WO and Zee-modified, and confines the error at a lower level for most of the time. In general, how much *LocMe* outperforms the wall-constraint-only methods depends on the specific user path, if the user tends to travel through the staircases, elevators and escalators more frequently, the more likely *LocMe* will benefit from the locomotion-constraint and maintain a low level of error. However, even when the user's path covers no POIs such that locomotion-constraint can not be applied, *LocMe* actually becomes LocMe-WO and its performance is still guaranteed to be no worse than wall-constraint only methods.

Table 3.3: Average error over all the test subjects of different localization methods in the field test.

	Error (m)		
	Mean	Median	85% Percentile
LocMe	**3.4**	**1.1**	3.1
LocMe-WO	5.4	3.4	6.3
Zee-Modified	5.5	3.6	6.5
Horus	2.7	1.3	**1.7**

Localization Accuracy. To understand the localization performance of *LocMe*, we further implement the Horus system [95] as a representative for popular localization systems built upon the WiFi infrastructure. Horus first measures the Received Signal Strength (RSS) of different WiFi Access Points (AP), and manually mark the locations. Then it builds up the probability distribution of the RSS readings over different locations. After that, each time a new measurement of different APs at location x is acquired, the system infers x by using maximal likelihood estimation.

Fig. 3.12b illustrates the CDF of *LocMe*, *LocMe*-WO, Zee-Modified and the Horus system. It can be observed that *LocMe* achieves similar or even higher accuracy than the Horus system at most of the time. The few points with high errors are at the beginning of the test, when the search space is initialized as the whole floor and not get reduced by any constraint yet. Such kind of error can be eliminated if we apply back-propagation after the particle filter converges to a concentrated group of points [73, 76]. In contrast, *LocMe*-WO and Zee-modified have more points with higher errors, because the search space is not reduced as efficiently without the locomotion-constraint. Table. 3.3 illustrates the errors of different localization methods. The listed results are the average over all the test subjects. From these results, we can see that *LocMe* reduces the mean localization error by over 37% and the median error by over 68% than the wall-constraint only methods.

3.8 Related Work

Indoor Maps. Indoor maps are widely available and the related service have been there for years [96–99]. Currently, information from the indoor map providers may be not accurate enough and is difficult to be retrieved automatically. However, for two reasons, we are confident that extracting accurate POI data automatically from indoor maps will become a reality soon: First, each building has a detailed floor plan serves purposes like renovation, emergency evacuation and fire hazard inspection, and such indoor maps are highly accurate. Map providers like Google are already collecting them by allowing the users to upload. Second, these floor plans usually come as simple line graphs, or even as digital script from designing software such as AutoCAD, and POI data within them can be extracted easily with some knowledge of the designing software or computer vision techniques.

Indoor Human Activities. There are three major interests relating to this topic.

The first one is about the human walking behavior, which is important in the medical care systems [82, 84, 89, 100].

The second interest is about estimating the human walking direction, which is essential for indoor localization [73, 75, 90].

The third is about activity profiling, which is highly beneficial for developing personal digital assistant and recommendation. Many efforts have been made in this direction. However, some of them only use statistical features of sensor readings for classification [85, 101–103], which cannot capture the short-term characteristics of different locomotion states like *LocMe*. UnLoc [77] clusters the activities of different users to tag landmarks, this crowd-sourcing manner is not suitable for the infrastructure-free scenario of *LocMe*; also, each landmark is directly used to reset the user's location, thus the uniqueness of the landmark is critical, which is validated by WiFi fingerprints, and again is not available for *LocMe*. In contrast,

LocMe does not need the detected POI to be unique, false ones will eventually get eliminated by the particle filter.

Floor Detection. Traditional floor detection approaches mainly rely on RF fingerprinting. For example, SkyLoc takes use of the GSM signals [104], and WiFi based fingerprinting is also well discussed [105–107]. The performances of these approaches are dependent on the training process which is hard to scale up to multiple buildings. Besides, they will not work in a infrastructure free scenario. Ftrack [78] and B-Loc [79] achieve the floor detection in a crowd-sourcing manner. They detect the encounters between users and share it among the users. This kind of systems need a central server who collects and distributes the user traces, which will not work in an environment without network access. Vanini and Giordano also employed the barometer and succeeded in detecting the floor transition, but were not capable of detecting the number of floor changes [108]. In contrast, *LocMe* works independently from any infrastructure and does not need the input from other users, and manages to give accurate floor detection which could be used directly to make map reloading decisions.

3.9 Conclusion

We propose *LocMe*, an infrastructure free indoor localization service which exploits the rich information from indoor maps and applying both the locomotion-constraint and wall-constraint, it achieves faster and more accurate localization than existing algorithms with a single constraint, but keeps the same level of complexity. Our field tests show that *LocMe* reaches a median localization error of 1.1 m and mean error of 3.4 m, which is over 37% and 68% lower than those of the wall-constraint only methods. In addition, *LocMe* can automatically detect the floor changes.

The actual improvement of localization accuracy and converging speed of synergizing

the constraints is dependent on the user's choice of indoor path. Nevertheless, *LocMe* can guarantee its accuracy and converging speed to be no worse than the wall-constraint only methods in any cases. Furthermore, particle-filter based localization methods can achieve even higher accuracy by using the estimated results to do backward smoothing. We leave it as our future work.

Chapter 4

SHOW: Smart Handwriting on Watches

4.1 Introduction

As the smart watch gains stronger computational capabilities and more versatile sensors, it has become a new digital hub on which the users install tons of applications and fulfill all kinds of daily tasks such as fitness tracking, message sending/receiving and note keeping, etc. After Google announced the next generation of Android watch to be completely phone independent [109], smart watch is transforming from a companion gadget to a full-fledged device and will be capable of performing a much wider range of tasks. Consequently, text entry on the smart watch is becoming a fundamental use case. Unfortunately, the watch's small screen drastically limits the performance of the tap-on-screen input method, which is the dominant one for touch-screen devices. Existing research aiming at overcoming the limitation of small screen leads to specially designed virtual keyboard, which is extremely different from the traditional QWERTY layout, hence has a steep learning curve. They may cause further inconvenience as the user switches between different devices from time to time. At the meantime, voice input methods are highly dependent on the environment. It cannot be used at places either require silence or has a lot of noises. Besides, it suffers from high risk of privacy leak when used in public.

Considering that the smart watch is always wrapped around the user's wrist, it is intuitive to relate it to the most traditional input method, i.e. handwriting. We can use the IMU (inertial measurement units) sensors such as the accelerometer and gyroscope to monitor

the hand/wrist/arm's movements and use the data to learn and recognize handwritings. In fact, using wearable sensors to track and recognize the user's gestures is not new, and many efforts have been made to recognize the finger, hand or arm gestures [110–113]. However, these attempts are not specifically made to achieve writing recognition and are subject to various limitations. For example, ArmTrak [110] requires movements of the entire arm, which will easily exhaust the user if used as a text input method, and it can not be applied in space limited places such as a vehicle; The work in [111] employs classification algorithms which heavily rely on the actual data samples but suffers from the difficulties of data collection at a large scale. To address these limitations, we propose *SHOW*, a system designed for recognizing handwriting with smart watches on horizontal surfaces. By asking the user to write with the elbow as the support point, it finds a balance between comfort and efficiency, and allows the users to input in the same way as they handwrite. By implementing rotation injection, *SHOW* manages to seed from a small set of collected traces and automatically generate a large quantity of traces, which are proven to be effective and robust, for the use of machine learning. Furthermore, by defining simple yet distinct gestures for indicating word separation and character correction, *SHOW* realizes character-level to word-level text entry, which we prove to be efficient for both daily and general-purposed text-entry tasks and express a few nice advantages over the tap-on-screen method.

The rest of this chapter is organized as follows: In Section 4.2, we first give a breakdown analysis of handwriting, and reveal the fundamental observations that support *SHOW*. In Section 4.3, we introduce the overall system design of *SHOW*. Then we explain the character recognition and word autocompletion in Section 4.4 and Section 4.5, respectively. We evaluate *SHOW* in Section 4.6. In Section 4.7, we introduce related work. In the end, we conclude the work and discuss the future work in Section 4.8.

4.2 Comprehend Handwriting with A Smartwatch

In this section, we look into the handwriting from three aspects: the watch-wearing hand, the support point and the watch rotation.

4.2.1 Watch hand v.s. Writing hand

The first challenge we face is that, *SHOW* requires the watch to be worn on the writing hand, but in reality, most people do not. This observation is also confirmed by our 10 test participants, where only 2 of them wear the watch on the writing hand. This convention is a legacy from the non-smart watch era where the only functionality of a watch is for displaying time, and people wear it on non-dominant hand to minimize the damage to the watch from daily activities. Such legacy is vanishing rapidly nowadays for the following three reasons: 1. Smart watches have far richer functionalities than a traditional one, and will be reached to by the user far more frequently. 2. Other than *SHOW*, many apps on the smart watch require the watch to be worn on the dominant hand for better functioning. For example, fitness tracking apps for basketball/tennis/badminton need to monitor the movements of the dominant hands. 3. Modern smart watches are designed to be used during sport activities, equipped with water-proof material and gorilla glass, and are much more resistant to daily damages than traditional ones. We have also surveyed 40 smart watch users (including our 10 test participants), and all of them express that wearing a watch on the non-dominant hand is only a habitual thing, and find it no compulsion to switch to the other hand if the apps can provide improved performance.

4.2.2 Support Point: the Controller of Speed, Comfort, and Amplitude

When the user handwrites, there are different potential support points: the palm bottom / wrist, the elbow or the shoulder, and the choice will affect our experience of writing speed, comfort and size. To understand how different support points affect the handwriting, we recruit 10 volunteers and ask them to write with a pen and the index finger, respectively. We observe that, when writing with a pen, the participants tend to use the bottom palm or the wrist as the support point to the moving fingers; when writing with the index finger, the participants tend to move the hand and the forearm and use the elbow as the support point. Furthermore, we observe that only when writing in the air the participants use the shoulder joint as the support point. These support points determine the length of levers we use to produce the writing traces, and consequently the amplitude of body movements, the comfort (or intensity of labor), the speed of writing, and the size of writing. Based on our observations and the participants' feedback, we summarize the effects of taking different support points for handwriting in Table 4.1. As the support point changes from palm bottom / wrist to the elbow and then to the shoulder joint, the amplitude of movements, the intensity of labor, and the writing size increase, but the writing speed decreases.

Fig. 4.1 illustrates sensory data from the same user writing letter "A" with the wrist, elbow and shoulder as the support point, respectively. Evidently, as the support point moves up, the amplitude of movements increases. In *SHOW*, we consider the case of handwriting with elbow as the support point, for the following reasons:

1. When taking the palm bottom / wrist as the support point, the amplitude of movement is relatively small because the watch is located near the support point in this case. As a result, the movements of the hand and fingers will only register a stabilized version at the watch. Therefore, it is extremely difficult, if not impossible, to recognize what the user writes

Table 4.1: Effects of taking different support points of writing. "↑" stands for increase, "↓" otherwise.

Support point	Amplitude of movements	Intensity of labor	Writing speed	Writing size
Shoulder	↑	↑	↓	↑
Elbow				
Palm/wrist				

without additional finger-wrapped sensors.

2. When taking the shoulder as the support point, the amplitude of movement is large since the entire arm is moving. This amplitude requires larger space thus greatly limits the potential application scenarios. For example, the user can hardly write with the entire arm when he is in a vehicle. Also, the user gets tired more easily in this case with the whole arm in the air and moving constantly.

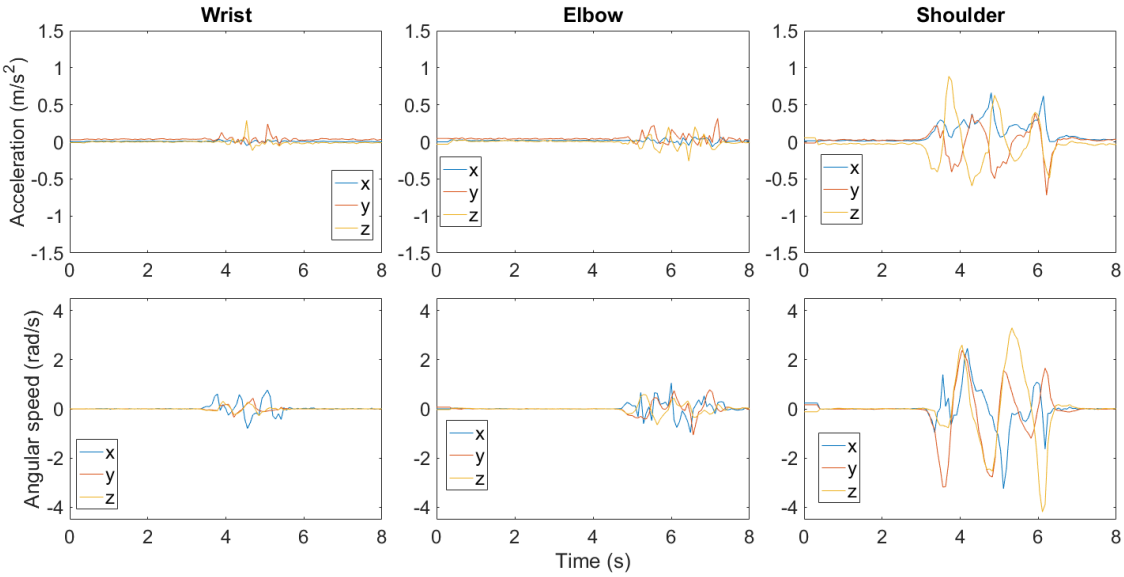


Figure 4.1: Sensory data of writing with different joints as the support point.

In conclusion, taking elbow as the support point is a choice that achieves the best balance

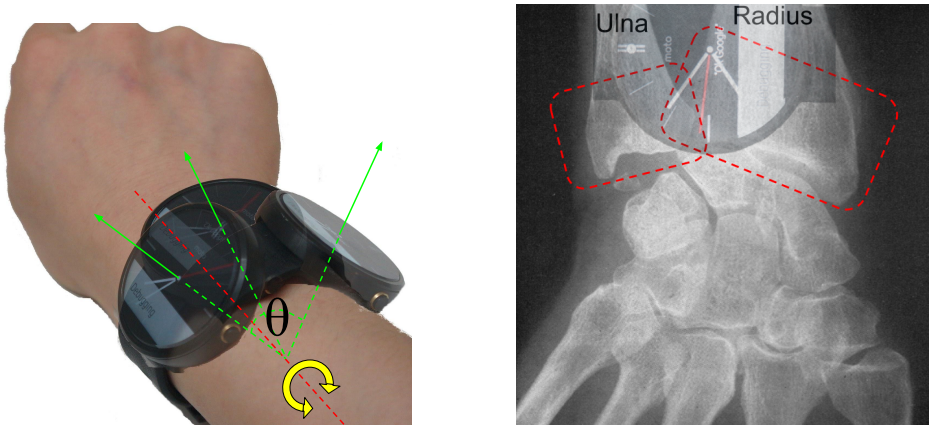
between amplitude of movements (space required), intensity of labor (comfort), writing speed and writing size. Fig. 4.3a gives an illustration of the user writing on a horizontal surface with the elbow as the support point.

Requiring the elbow as the support point filters movements from the upper arm and the whole body, largely constrains the possible movements of the wrist in space. This highly benefits *SHOW* and helps it to achieve a high character recognition accuracy. In the meantime, this requirement restricts potential application scenarios. For example, when writing on a vertical surface, it is usually hard to find something to support the elbow and will lead to degraded handwriting recognition performance. For the same reason, *SHOW* cannot work well while the user is standing or walking. However, as the need of typing on smart watch while walking or standing is rare anyway, and when sitting, the user can always find a horizontal surface for use (using the thigh in the worst case), we consider the tradeoff between recognition accuracy and number of application scenarios constrained by the elbow support requirement as acceptable.

4.2.3 Watch Rotation: Challenge and Opportunity

We observe that people tend to wear the watches loosely for more comfort. Therefore, each time the user puts on the watch, its facing direction could be different. We hereby refer to such inconsistency between watch facing directions as the *static rotation*. After the watch has been put on, it may also swing and rotate during the handwriting process, and we refer to such rotation as *dynamic rotation*.

The existence of watch rotation is both a challenge and an opportunity. It is a challenge because it requires massive human effort to collect writing samples with respect to different potential rotations. For models based on machine learning algorithms, the final performance of writing recognition is greatly constrained by the rotations. On the contrary, it is an



(a) (b)

Figure 4.2: Watch rotation around the arm axis. (a) An illustration of the rotation. (b) An X-ray image showing the anatomy of the wrist joint [114] that constrains the rotation.

opportunity because, once we understand the effects of the rotation, we can use it to generate data from the same user writing with different watch face directions. With this approach, the effort of collecting more handwriting data can be substantially reduced. Moreover, the robustness of *SHOW*'s character recognition can be enhanced by taking the rotation into account.

It follows that the comprehension of the watch rotation is the key to convert the challenge into our chance.

First, the rotation is confined between the bulged ends of ulna and radius bones of the wrist joint, as shown in Fig. 4.2b. If we take the watch facing direction when the hand is at rest as the initial direction, the rotation of the watch is confined within a range about $\pm 30°$ (θ in Fig. 4.2a).

Second, we observe that even though the watch is loosely wrapped around the wrist, the friction between the skin and the watch prevents it from swinging, which means the possible *dynamic rotation* within a writing trace is small and negligible.

Based on the two observations, we implement the *rotation injection*, which is detailed in

Section 4.3. It helps *SHOW* to increase its robustness and considerably reduce the effort of collecting handwriting data. Our experiments show that the average accuracy of *SHOW* can be increased by 16.5% after the rotation injection.

4.3 System Overview

SHOW is designed for use on horizontal surfaces, and requires the user to use the elbow as the support point (Fig. 4.3a). It allows the user to write in a semi-cursive way: For each single character, the user can write cursively with his/her personal style (no need to be the printed version), but for any two consecutive characters, there should be a full stop in between (which we denote as the *soft separation* and is introduced in Section. 4.5).

(a) Elbow as the support point. (b) Use the other hand to select a word.

Figure 4.3: Illustration of writing with *SHOW* on a horizontal surface.

As the user writes, each character is recognized by *SHOW*, and a suggestion list of words with the current input as prefix is updated and displayed dynamically. When the intended word appears in the suggestion list, the user needs to keep the writing hand at still, then uses the other hand to tap on it (Fig. 4.3b). By doing so, the tapped word will be selected

and *SHOW* will be notified the beginning of the next word. If the intended word is the first candidate in the suggestion list, the user can also make a *hard separation* gesture (introduced in Section. 4.5) to indicate the end of the current word, and the first candidate word will be selected automatically.

The architecture of *SHOW* is shown in Fig. 4.4. It collects the accelerometer and gyroscope data from the smart watch. After the noises have been removed, the data processing goes through two stages: Character recognition and word recognition.

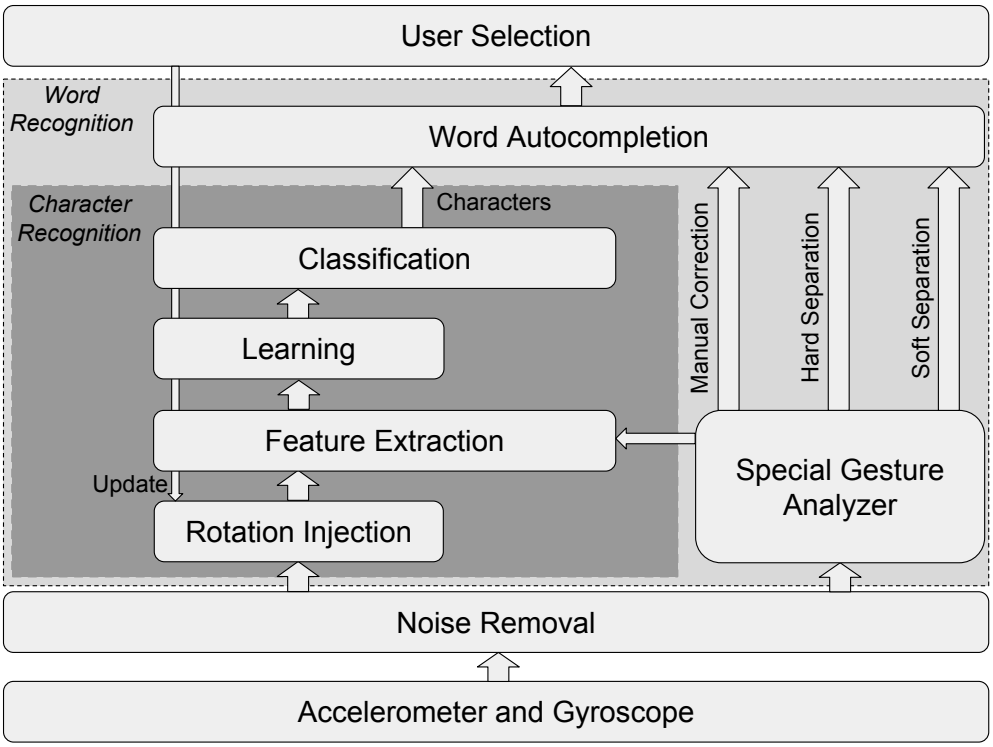
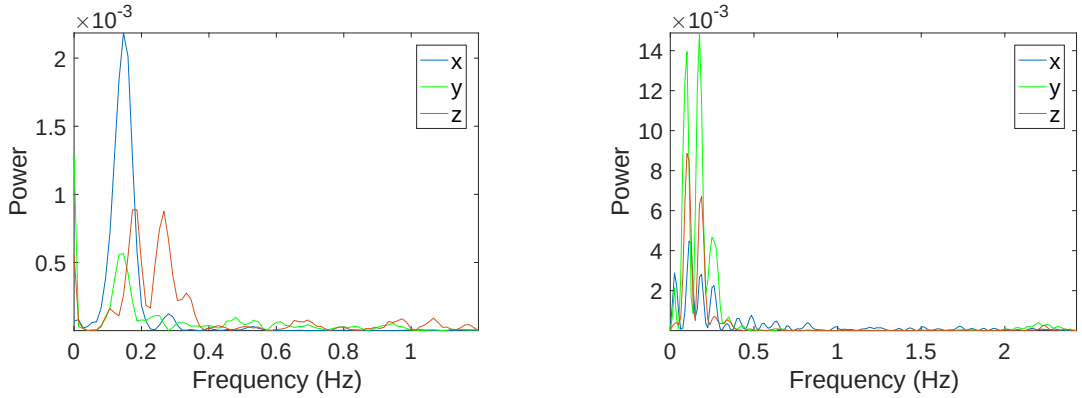


Figure 4.4: Architecture of *SHOW*.

4.3.1 Noise Removal

Human handwriting is in general a low-frequency activity. Fig. 4.5 shows the power spectrum of one of the volunteers writing letter "A". The energy of the writing trace is concentrated on the lower frequency range. Hence, for all the traces *SHOW* collects on the watch, a 10th order butterworth 2 Hz low-pass filter is applied to remove potential high-frequency noises.



(a) Accelerometer. (b) Gyroscope.

Figure 4.5: The power spectrum of a handwriting trace in the frequency domain.

4.3.2 Data Flow

At the first *Character Recognition* stage, for all the labeled handwriting traces that *SHOW* collected for the training purpose, they are taken as seeds for generating extra traces, using rotation injection. After that, each trace's features are extracted and they are fed into a machine learning model to train the classifier.

For the user's input traces when he/she is actually using *SHOW*, they are first fed into the *Special Gesture Analyzer*, which will detect the special gestures that *SHOW* defines for better

word autocompletion. Only when the traces are not recognized as any special gestures, their features will get extracted and be fed into the pre-trained classifier.

At the second *Word Recognition* stage, all characters recognized by the classifier and special gestures detected by the *Special Gesture Analyzer* are sequentially processed to produce a list of word suggestions.

After the two stages of data processing are completed, the user will have a suggestion list of words from which he/she can choose the intended one. His/her choice will also label the corresponding traces, which will later be employed to update the classifier.

In the following sections, we will elaborate how each data processing stage is designed and implemented.

4.4 Character Recognition

4.4.1 Rotation Injection

As we discussed in Section 4.2, people tend to wear the watch in a relatively loose fashion which results in the *dynamic rotation* and *static rotation*. The *dynamic rotation* is negligible and we mainly consider the *static rotation* here. For each recorded trace $\boldsymbol{x}$, we generate 60 rotated versions of it by doing the following *rotation injection*,

$$\forall i, \boldsymbol{x}'(T_i) = \boldsymbol{R}(\beta)\boldsymbol{x}(T_i), \quad \beta = -30^\circ, -29^\circ, \ldots, -1^\circ, 1^\circ, \ldots, 29^\circ, 30^\circ.$$

Here, $\boldsymbol{R}(\beta_i) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\beta_i & -\sin\beta_i \\ 0 & \sin\beta_i & \cos\beta_i \end{bmatrix}$ is the rotation matrix of β_i over the arm axis (x axis); T_i denotes the ith timestamp.

Rotation injection mimics the slightly different directions the user may wear the watch. The ±30° rotation range is determined by the wrist anatomy as explained in Section 4.2. After the injection, each handwriting trace we collect from the user will produce 60 traces, thus the number of training samples are greatly increased. Theoretically we can increase the rotation granularity to produce more traces, but our tests show that with the 1° rotation granularity, we can already acquire sufficient traces for the later model training stage. *Rotation injection* also brings in extra variety to the dataset and increases the robustness and generalizability of the model. We will show this in the evaluation later.

4.4.2 Feature Extraction

In [115], Emmanuel experiments on human activity recognition with machine learning algorithms, and analyses the effectiveness of 41 types of features extracted from the data collected by body-attached sensors. Among these features, Xu et al. show that three sets are particularly effective for classifying arm and hand related gestures [111]. In *SHOW*, we also take these features for learning about handwritings. Table 4.2 lists the accelerometer related features and how they are computed respectively. Features with prefix "DC" refer to the DC components of the fourier transform, and those with "AC" correspond to the AC components. For *ACEnergy*, *ACLowEnergy*, *DCMean*, *DCArea*, *CPostureDist*, *ACAbsMean*, and *ACAbsArea*, they are axis-dependent and will each be converted to three features. Also, for each feature in Table 4.2, there is a corresponding version for gyroscope. They sum up to 46 features in total.

4.4.3 Learning

We test seven machine learning algorithms for *SHOW*. The following paragraphs briefly explain them and Table 4.3 presents details about the corresponding test settings.

Table 4.2: Accelerometer features used in *SHOW*. Each feature listed here also has a corresponding gyroscope feature.

Feature	***Computing formula***	***Description***
ACEnergy	$\sum_{i=1}^{n_f/2} \mathrm{m}_i^2$	The total energy of a signal without the DC component.
ACLowEnergy	$\sum_{f_i \in [0,0.7]} m_i^2$	The energy of the signal within [0,0.7] Hz.
DCArea	$\sum_{i=1}^{n_s} a_i$	The area under a signal.
DCMean	N/A	The mean of a signal, which is a measure of the static component of acceleration that changes with the body.
DCTotalMean	N/A	The mean of a signal over all axes, captures the overall posture information contained in the DC component.
DCPostureDist	N/A	The mean distances between the X-Y, X-Z, and Y-Z acceleration axis. It captures the body posture information.
ACAbsArea	$\sum \lvert a_{axis_i} \rvert$	The area under the absolute values of a signal.
ACAbsMean	$\mathrm{mean}_i \lvert a_i \rvert$	The mean over absolute values of a signal.
ACTotalAbsArea	$\sum_{i=1}^{n_{axis}} \mathrm{ACAbsArea}_{axis_i}$	The area under the absolute value of a signal, summed over all the axes.

a: acceleration reading.
f_i: The ith frequency component.
m_i: The magnitude of the ith FFT coefficient.
n_{axis}: number of axes for a sensor.
n_f: Number of frequency components after FFT.
n_s: Number of sensory readings.

Bagging of Decision Trees (BDT) When the classification boundary is complicated, the decision tree method tends to form deep trees and overfits. Bagging is an ensemble technique which aggregates the results of many decision trees to alleviate such an overfitting problem. It takes the decision tree as the base learner, and trains it with different bootstrap training sets B_i. B_i is a replica of the original data set S, and is generated by uniformly drawing samples (with replacement) from S. When classifying a new sample, a poll is made among the results of all the trained trees and the most voted one is taken as the final

result [116].

Random Forests (RF) Random forests is another technique that works on an ensemble of decision trees. Unlike the BDT algorithm, which randomly draws the data samples, RF randomly picks a subset of features to train different decision trees [117]. The final classification result is also based on the poll of all the trees.

Table 4.3: Test settings for the classification algorithms.

Algorithm	**Parameters**	**Tested Values**
BDT	# of trees	{30, 40, 50}
	Max allowed splits	{100, 200, 300}
RF	# of trees	{30, 40, 50}
	Max allowed splits	{100, 200, 300}
SVM	Breakdown strategy	{OneVSOne, OneVSAll}
	Kernel function	{linear, quadratic, cubic, gaussian}
MLR	N/A	N/A
NB	N/A	N/A
NN	# of hidden layers	{1, 2, 3}
	# of nodes / layer	{20, 30, 40}
KNN	k	{20, 40, 60, 80}
	distance metric	{euclidean, correlation, cosine}

Support Vector Machine (SVM) It finds a hyperplane which leads to the largest margin between data points from two classes. It is usually used in binary classification problems. To apply it to multi-class problem, we need to first break it down to multiple binary classification problems and build many binary SVMs first. There are two strategies to achieve such a breakdown: OneVsOne and OneVsAll [118, 119]. In the OneVsOne strategy, a binary SVM is trained for each pair of classes. When classifying a new instance, it is tested against each binary SVM, and the final result is based on the poll over the result of each binary SVM. In the OneVsAll strategy, for each class, a binary yes or no classification problem is formed and a corresponding binary SVM is trained. When classifying a new

instance, it is also tested against each binary SVM, and the result with the highest score taken as the final decision.

Multinomial Logistic Regression (MLR) Logistic regression outputs the possibilities of an instance being each of the classes, and takes the one with the highest possibility as the final result. However, it is only applied to binary classification problems. In contrast, MLR extends it to a multi-class problem, where the classes are nominal. In *SHOW*, the set of different writing characters are also nominal, hence we can apply MLR.

Naive Bayes (NB) Given an instance, NB estimates the posterior probability of the instance belonging to each class and takes the class with the highest probability as the classification result. NB is based on the assumption that, given a class, different features are conditionally independent from each other.

Neural Network (NN) Generally speaking, a neural network consists of the input, hidden, and output layers, each of which is composed of neurons. Using a few hidden layers, neural networks are capable of capturing the subtle, non-linear relationships between the input and output. In a classification task, whose input is continuous, it is sensible to use sigmoid neurons in hidden layers and softmax function in the output layer. The whole network is a feed-forward network that is trained by applying the back-propagation algorithm.

K Nearest Neighbors (KNN) When classifying a new instance, KNN searches the instance's k nearest neighbors in the feature space, and takes the majority class among these neighbors as the result.

4.5 From Character to Word

After achieving character-level recognition, the next task for *SHOW* is to present word suggestions based on the current user input, so that the word can be autocompleted and the

text-entry speed be improved. According to Zipf's Law [120], in a large corpus, a given word's frequency is approximately inversely proportional to its rank of frequency. This indicates that most intended words can be autocompleted with a short prefix input. Traditionally, such a word autocompletion task is fulfilled by a frequency-based statistical model, which has been proven to be effective and is widely adopted for every day text-entry tasks both on the personal computers and mobile phones [121–123]. The idea of such model consists of two steps: 1. Build up an n-gram dictionary using the word frequency from a specific linguistic database and/or the user's own vocabulary, such as an address book or a contact list. 2. Present a suggestion list where potential words are sorted with probability. However, there are two challenges that prevent *SHOW* from adopting this model directly.

First, the autocompletion model assumes that the character can be detected one at a time, so that the corresponding word suggestions will refresh upon each new incoming character. An example is shown in Fig.4.6a, when "g" and "r" is detected, we have the illustrated suggestions, and if the next detected character is "o", then "great" and "grew" can be removed from the list and new suggestions like "ground" and "grow" can be added. In previous character recognition tests of *SHOW*, the participants are asked to write the characters one at a time, so each recorded trace only contains one character, but in reality, the user tends to write fast and the real-time sensory trace may include a few characters sequentially. In this case, *SHOW* has to separate the characters first.

Second, the autocompletion model also assumes the character to be always correctly detected. This is not true in practice. For example, in the tap-on-screen context, because the user's finger actually taps an area instead of a single point, the input character will sometimes be mistaken as the ones adjacent to them on the keyboard. This problem is drastically exacerbated on the watch, where more keys are packed into each unit area due to the small screen size. In our evaluation, we show that this "mis-tap" rate is as high as 16.2% on average. In *SHOW*, similar problem exists where an input character can be

misclassified as another one in our recognition stage. To deal with this challenge, *SHOW* has to be tolerant with the character ambiguity and be able to present suggestions accordingly. Take the same example in Fig. 4.6a, if a new character is recognized as "u" after "gro", the system should consider the probability of it being a mistake. For the tap-on-screen method, the potential confusion for "u" may be "i", "y", "j", "h", so words like "grief" or "grip" should also be considered. For *SHOW*, possible confusion of "u" can be "a", thus words like "grant" or "graph" should be considered.

In the following sections, we will elaborate upon how *SHOW* addresses the two challenges. After that, a few use cases are presented to explain *SHOW*'s advantages over the tap-on-screen input method.

(a) Tap-on-screen. (b) *SHOW*

Figure 4.6: An example of word autocompletion based on characters input. Character "g" and "r" are the current input.

4.5.1 Character Separation

When the user writes the characters sequentially, *SHOW* considers two types of character separation. The first is the *soft separation*, where the traces of two characters should be separated for recognition, but they belong to the same word and should be considered

together for current word autocompletion. The second is the *hard separation*, where the two characters belong to different words, which could be imagined as the space character. *SHOW* assigns special hand gestures to the two types of separations so that the user can notify the system about them. It's straightforward to see that the *soft separation* appears much more frequently than the *hard separation*. Therefore, *SHOW* assigns a gesture that needs as little time and effort as possible to indicate the *soft separation*.

- Soft separation gesture. The user can rest the hand at still for a short moment to indicate a soft separation, and this gesture can be detected as a zero pulse (amplitude smaller than a given threshold in implementation) on all 3 dimensions for both the acceleration and the angular speed (Fig. 4.7b). Our tests show that such a moment can be as short as 0.2 seconds to be successfully detected, which requires little cost in terms of both time and human effort.

- Hard separation gesture. We observe that the wrist flipping gesture around the arm axis never occurs during the handwriting process, and can be used as an indicator of the hard separation. To input a hard separation, the user quickly rotates the wrist around the arm axis (x-axis of the watch) towards the outside direction and then rotates back. This gesture can be detected as a deep valley followed by a high peak on the x-axis of the gyroscope, while the other two dimensions stay much lower comparatively (bottom subfigure of Fig. 4.7a).

4.5.2 Character Ambiguity

Because the characters cannot be guaranteed to be correctly recognized, *SHOW* has to take potential confusions into consideration. In the classifier training stage, *SHOW* can output a confusion matrix which contains the information about the possibility of confusing

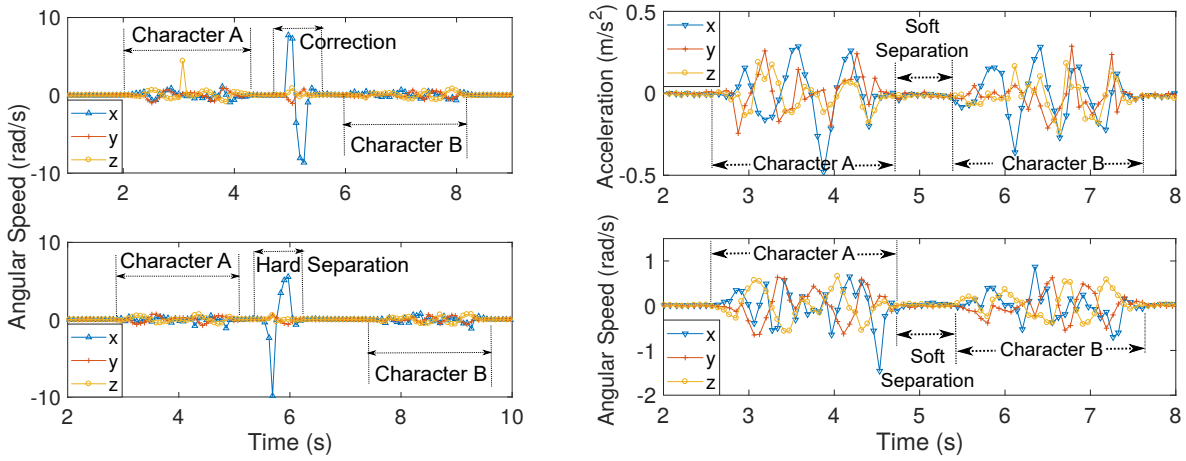


Figure 4.7: Examples of special gestures. (a) Top: Manual correction; Bottom: Hard separation; (b) Soft separation.

one character with another. Then given n input characters $C_1, C_2, \ldots, C_n$ since the last hard separation, where C_n is the latest one, the probability of suggesting the word W is computed as:

$$P(W) = \sum_{C_1^{(R)}C_2^{(R)}\ldots C_n^{(R)} \in \mathcal{C}^n} P\left(W|C_1^{(R)}C_2^{(R)}\ldots C_n^{(R)}\right) \prod_{i=1}^{n} P(C_i^{(R)}|C_i), \tag{4.1}$$

where $C_i^{(R)}$ represents the possible recognized result of the actual input C_i, $\mathcal{C}^n$ is the space containing all possible character series with length n, $P\left(W|C_1^{(R)}C_2^{(R)}\ldots C_n^{(R)}\right)$ is the probability of having word W given the prefix $C_1^{(R)}C_2^{(R)}\ldots C_n^{(R)}$, which can be computed by looking up the relative frequency of all words with the same prefix in the pre-built language database [124], and $P(C_i^{(R)}|C_i)$ is the probability of recognizing C_i as $C_i^{(R)}$. One thing we need to clarify here is that, when the user inputs the series $C_1C_2\ldots C_n$, *SHOW* does not know what it actually is. For each input C_i, *SHOW* actually takes the recognized result with highest

probability to represent it. In another word, Eq. 4.1 is implemented as:

$$P(W) = \sum_{C_1^{(R)}C_2^{(R)}...C_n^{(R)} \in \mathcal{C}^n} P\left(W|C_1^{(R)}C_2^{(R)}\dots C_n^{(R)}\right) \prod_{i=1}^{n} P\left(C_i^{(R)}|\operatorname*{argmax}_{C^{(R)}} P(C^{(R)}|C_i)\right). \quad (4.2)$$

Here $\operatorname{argmax}_{C^{(R)}} P(C^{(R)}|C_i)$ is the output of the classifier we trained beforehand.

This computation has an implicit assumption that, even if the user sees that his/her input character is mis-recognized, he/she will not correct it, but instead keeps on inputting. If we remove such assumption, Eq. 4.2 can be further simplified as:

$$P(W) = \sum_{C_n^{(R)} \in \mathcal{C}} P\left(W|C_1^{(R)}C_2^{(R)}\dots C_n^{(R)}\right) P\left(C_n^{(R)}|\operatorname*{argmax}_{C^{(R)}} P(C^{(R)}|C_n)\right). \quad (4.3)$$

In reality, whether to use Eq. 4.2 or Eq. 4.3 depends on the user's input style. If the user tends to correct the mis-recognized input, he/she has to make extra efforts for the correcting operation, but benefits *SHOW* by reducing the search space of candidate words. If the user does not care to correct the current mis-recognized characters, *SHOW* has to do more computations to deal with the ambiguity. In the implementation of *SHOW*, it starts with Eq. 4.2, and employs the detection of the manual correction gesture (explained in the next subsection) as a heuristic to decide whether it should switch to Eq. 4.3.

4.5.3 Manual Correction

In some cases, the user may accidentally input a wrong character, or the input character has been mistakenly recognized as another, and he/she wants to manually correct it. *SHOW* supports such a correction by allowing the user to make a *correction gesture*. This gesture is defined similarly as the hard separation gesture, only with an opposite wrist flipping direction. An example of its sensory readings is shown in the top subfigure of Fig. 4.7a. This

gesture is recognized by a high peak followed by an deep valley on the x-axis of gyroscope, with the readings of the other two dimensions being at a much lower level.

4.5.4 Recognition Feedback

When the user chooses a suggestion from *SHOW*'s autocompletion list, he/she is also confirming the current input. *SHOW* can use this feedback information to improve the recognition. This is achieved by the following steps:

1. The collected traces are separated into small pieces using the soft separation, with each piece corresponding to one character.

2. After the user chooses a suggested word, the current input prefix of that word is used to label its corresponding sensory trace.

3. When the labeled traces reach a certain amount, they are merged back to the original training dataset, and the character recognition model will be trained again.

4.5.5 Special Use Cases

Upper-case vs lower-case. For the tap-on-screen input method, the input letters are of lower cases by default, and if the user wants to input an upper-case letter specifically, he/she will have to tap the "shift" key first, tap the intended letter and then tap the "shift" key again, which means three taps are needed for one upper-case letter. In contrast, in *SHOW* the upper-case and lower-case letters are distinguished directly because of their intrinsic differences in the handwriting traces. Only in a few special cases where the upper-case and lower-case letters are simply different in terms of size, such as "c", "o", and "s". For these special letters, *SHOW* allows the user to prefix them with an "∧" character and a soft separation to explicitly notify the system that the upper-case is intended.

Digits. When writing the digits, the tap-on-screen method also requires 3 taps, because the default virtual keyboard layout only presents alphabetical letters, and the user has to tap a special key (the one marked as "?12" at the bottom left of Fig. 4.6a) to enable/disable the digit input mode before/after inputting the digits. On the contrary, *SHOW* is trained directly with the handwriting traces of digits and can distinguish them from the alphabetical letters, hence no extra operations from the users are needed.

Length of the suggestion list. As illustrated in Fig. 4.6a, the virtual keyboard occupies the most part of the watch screen, and leaves very limited space for displaying the suggestion list. In contrast, *SHOW* is free to use the whole screen to display the suggestion list (Fig. 4.6b). Thus, it can display much more (4 times on average) word suggestions at one time. Furthermore, with the spare screen space, *SHOW* has the luxury to use different sizes of fonts to prioritize the most likely suggestion for the user's ease of selection.

4.6 Evaluation

4.6.1 Experiment settings

We recruited 10 volunteers to collect the handwriting samples. The detailed information about these volunteers are listed in Fig. 4.8. They all wore the watch around the wrist on the writing hand and adjusted the tightness till comfortable.

SHOW is implemented on the Android platform. We tested on two different models of smart watches: Moto 360 and Samsung Galaxy Gear. Data from the accelerometer and the gyroscope are collected at the system's default UI sampling rate (about 16.7 Hz).

The google-10000-english list [124] is used to build up the n-gram word model in *SHOW*. It contains the 10,000 most common English words in order of frequency, extracted from

Google's Trillion Word Corpus.

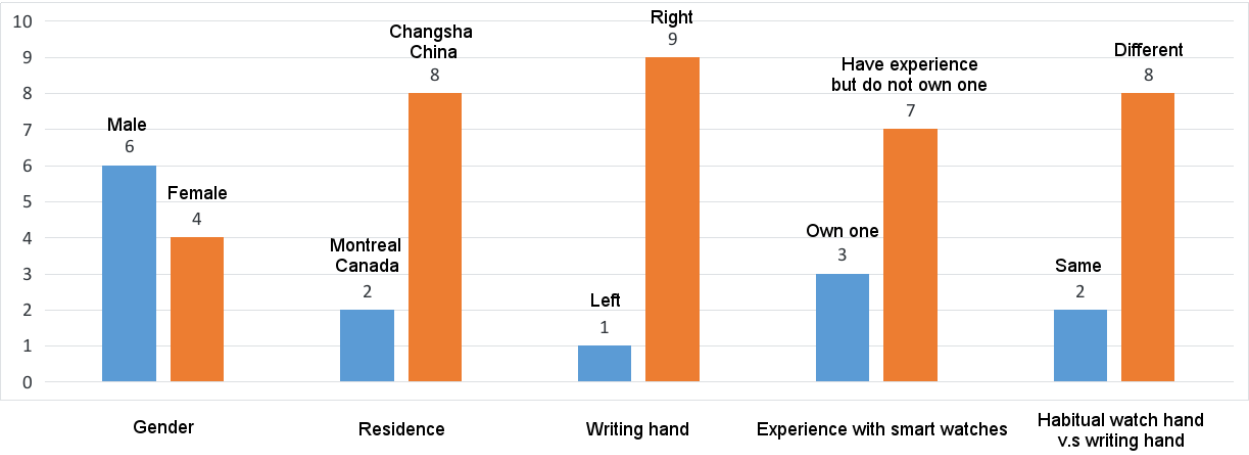


Figure 4.8: Information about the 10 volunteers.

4.6.2 Character Recognition Tests

In these tests, each volunteer is asked to write the character A-Z, a-z and 0-9 on the paper, one at a time and each character for 30 times. When writing the characters, the only requirement is to use their elbow as the support point. The size of the character is recommended to be 1/4 of a US letter paper, but we do not draw a box or set any constraints alike to enforce such recommendation. Therefore, even for the same character and the same volunteer, the sizes of writings may vary.

We carry out the following three kinds of training and recognition tests, all with 10-fold validation.

1. **Single person test**. In this test, for each user we train a personal classifier, using his/her data only.

2. **Cross person test**. In this test, all the users' data are used to train a general classifier, and the classifier is evaluated with all the user's data.

3. **Rotation Injection test**. For each of the above test, we conduct two versions of experiments, one uses only the original data without rotation injection, and the other one uses data with rotation injection.

The experiment results are shown in Tab. 4.4, we give detailed analysis in the following subsections.

Single-person recognition. This test evaluates the effectiveness of the features. From Table 4.4, we can tell that for the single-person test without RI, the highest average accuracy can achieve 91.0% (NN), this implies that the feature set we use are informative enough for high accuracy handwriting recognition.

Cross-person recognition. The cross-person recognition test evaluates the model's generalizability. From Tab. 4.4, the highest average accuracy without RI is achieved by NN at 89.4%. However, almost for all the classification algorithm, we notice an accuracy drop in the cross-person test compared to that in the single-person test. This implies that the model is not well generalized. That is where RI shows its charm.

Effectiveness of rotation injection. For both single-person and cross-person tests, and for all the classification algorithms, we observe a great boost to the recognition accuracy after adding in the rotation injection. The average accuracy boost is 16.5%, with the highest gaining a 24.5% accuracy improvement (SVM). Also, if we compare the results of cross-person with RI to those of single-person without RI, we can also observe a great accuracy improvement. This indicates that the rotation injection operation also increases the model's generalizability and robustness.

Table 4.4: Recognition accuracy of the tests.

Algorithm	Average accuracy in single-person test		Average accuracy in cross-person test	
	without RI	**with RI**	**without RI**	**with RI**
BDT	79.2%	96.5%	77.2%	98%
RF	76.9%	98.0%	76.2%	97.8%
SVM	75.4%	**99.9**%	76.9%	**99.8**%
MLR	72.7%	88.0%	69.4%	87.2%
NB	76.3%	93.8%	74.3%	91.4%
NN	**91.0**%	95.6%	**89.4**%	93.3%
KNN	68.8%	83.4%	69.5%	81%

The recognition performance. Table 4.4 shows that, when trained with RI, some of the classification algorithms can give great recognition accuracy, at an average over **95%** (BDT, RF, SVM, NN). And SVM in particular reaches an average accuracy of **99.9%**. This brings great confidence for further word autocompletion.

4.6.3 Input Efficiency

To understand the input efficiency of *SHOW*, we compare it with the tap-on-screen method from three aspects: error rate, word coverage and time cost.

Error rate. We define the following error rates:

- Mis-recognition-rate-TAP. This rate reveals how often the user's tap on the screen is mistakenly interpreted. It is computed as:

$$\text{mis-recognition-rate-TAP} = \frac{\text{\# of mis-taps}}{\text{total \# of taps}},$$

  where the total number of taps takes into account all the user intended input, including

the characters, the backspace symbol, the "shift" key for case switching, and the space symbol, etc.

- No-response-rate-TAP. This rate reveals how often the user's tap is not detected at all. It is computed as:

$$\text{no-response-rate-TAP} = \frac{\text{\# of not responsed taps}}{\text{total \# of taps}},$$

- Mis-recognition-rate-SHOW. This rate reveals how often a user input is incorrectly recognized by *SHOW*. It is computed as:

$$\text{mis-recognition-rate-}SHOW = \frac{\text{\# of incorrectly recognized characters}}{\text{total \# of written characters}},$$

  where the total number of written characters takes into account both the intended characters, hard/soft separation and the correction gestures.

- No-response-rate-*SHOW*. This rate reveals the frequency of the written characters not being detected at all. It is computed as:

$$\text{no-response-rate-}SHOW = \frac{\text{\# of not detected written characters}}{\text{total \# of written characters}}$$

To test these error rates, we consider two cases.

- In the first case, we randomly pick 20 short sentences from the MacKenzie PhraseSet [125], which is widely used in the HCI community for evaluating text input methods. The statistical information of the chosen phrase set is shown in Fig. 4.10. The phrase set's correlation to the English language is computed with the tool provided by [125], and the high value indicates that the phrase set is representative of the language. By

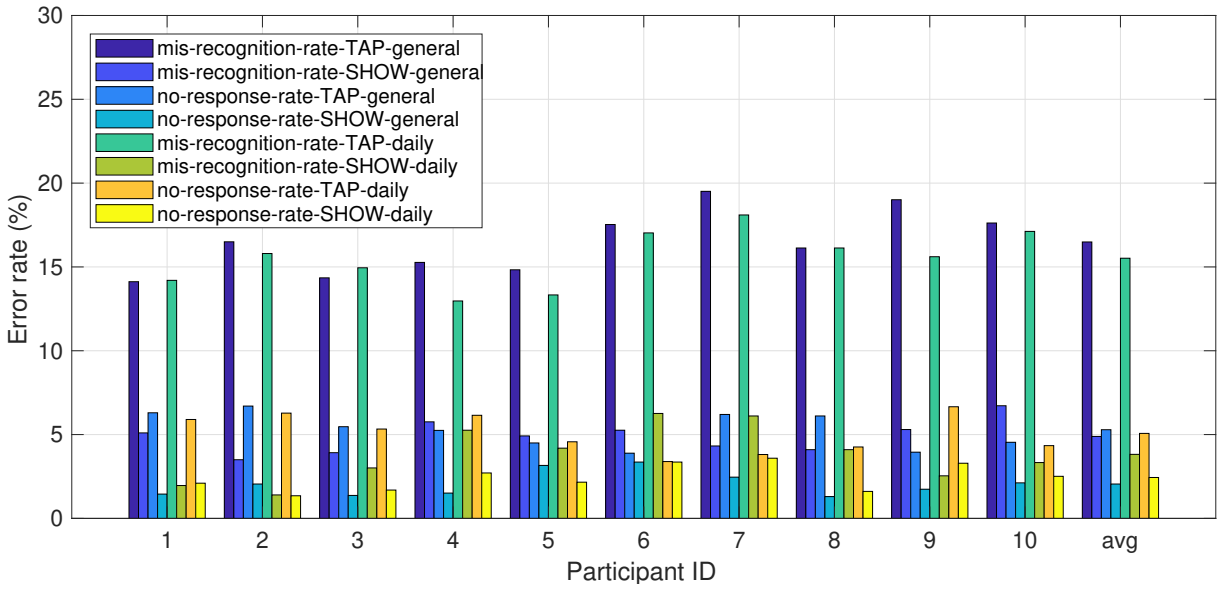


Figure 4.9: Error rate of the tap-on-screen method and *SHOW*.

asking the user to input this phrase set, we evaluate the performance of *SHOW* as a general purposed text-entry method (tagged with "general" in Fig. 4.9).

- In the second case, we choose 20 frequently used sentences from the participants' daily smart watch activities, such as "In a meeting, call you back", "what do you like for dinner", etc. This evaluates *SHOW*'s performance in daily text-entry tasks (tagged with "daily" in Fig. 4.9).

In both cases, we first allow the 10 participants to get familiar with the target phrase set, then ask them to input the text on the watch with the tap-on-screen method and *SHOW* respectively, with sufficient rest time in between (>30 mins). These measures prevent the performance of the two input methods from being affected by the participants' familiarity or weariness of the tests. The result is shown in Fig. 4.9. Not surprisingly, due to the densely packed virtual keyboards, all 10 participants experience high mis-recognition-rate with the tap-on-screen method, average at 16.5% in the general-purposed case and 15.5% in the daily

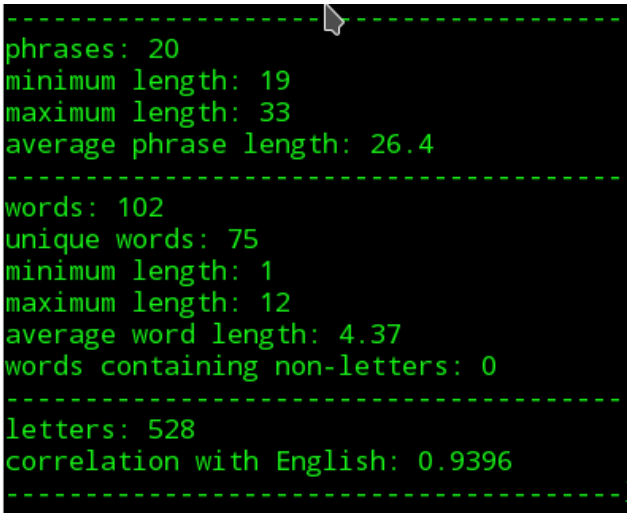


Figure 4.10: Statistical information of the chosen phraseset for general purposed text entry test.

case, and also a high no-response-rate, average at 5.3% in the general-purposed case and 5.1% in the daily case. In comparison, ***SHOW*'s average mis-recognition-rate (4.9% for general-purposed case and 3.8% for the daily case) are 70% and 75% lower respectively, and the average no-response-rate (2.3% for the general-purposed case and 2.4% for the daily case) are 46% and 43% lower, respectively**. From the participants' feedback, the frequent mis-recognition and loss of response not only slow down the input speed, but also produce an annoying experience which prevents the user from future use of that input method.

Furthermore, results in Fig. 4.9 demonstrate that there is no significant difference in the performances of *SHOW* between general purposed and daily text-entry tasks. Thus, we conclude *SHOW* can be applied nicely in both use cases.

Word Coverage. We adopt the same linguistic database for the tap-on-screen method and *SHOW* to give the word suggestions based on the input prefix. Therefore, given the input

characters being detected correctly, the two input methods will output the same suggestions. However, because *SHOW* has more spare space to display the word list, it can present 4x more suggestions than the tap-on-screen method on average. We ask each participant to randomly choose 20 phrases from the MacKenzie PhraseSet [125] to test with the two input methods, and find that **on average 33.3% of the intended words are presented by *SHOW* but not the tap-on-screen method**.

Time cost. The total time cost for inputting is affected by many factors, including the error rates and word coverage we discussed, as well as the time cost for input each character. If we assume that the tap-on-screen method and *SHOW* have the same word autocompletion coverage, then when a word is successfully suggested by *SHOW* but not by the tap-on-screen method, we can assume the tap-on-screen method already has it in the list but only has no space to display it. In the best case, the user can make one extra tap to scroll the suggestion list and find the desired word. Also, in the best case, one mis-tap/mis-recognition event will lead to two more input (1 for the correction operation and 1 for re-input the intended character); one no-response event will lead to 1 more input. Thus, the total input time of n characters can be estimated as:

$$t = (1 + 1 * r_{NS} + 2 * r_{Mis} + 1 * r_{NR})n\bar{t}.$$

where r_{NS} is the rate of words being suggested but not shown on the current screen; r_{Mis} is the mis-recognition rate; r_{NR} is the no-response rate; $\bar{t}$ is the average time cost of inputting a single character. By timing the participants' actual input, we find that $\bar{t}$ for *SHOW* is about 1.9 seconds, and for the tap-on-screen method it is 1.1 seconds, this large difference is caused by the soft separation of *SHOW*, where the user needs to wait a small period ($\geq$0.2s) to indicate the end of a character. Despite of *SHOW*'s higher unit time cost for each single

character, its lower mis-recognition-rate and no-response-rate will compensate as the input length increases. Using the error rates we get in previous tests, we find that *SHOW* only needs 16.8% more time to input the same number of characters.

However, this result is acquired at the best case for the tap-on-screen method with simplified assumptions:

1. We have assumed that all the words that are not suggested can be found by scrolling the candidate list by only one tap. In reality, considering that *SHOW* can present up to 4x candidate words than the tap-on-screen method, the user may need up to 3 extra taps with the tap-on-screen method to finally reach the intended word.

2. We have assumed that the correction operation will not be mis-recognized. In reality, this is not true, and the tap-on-screen method may need more extra taps since it has a higher mis-recognition rate.

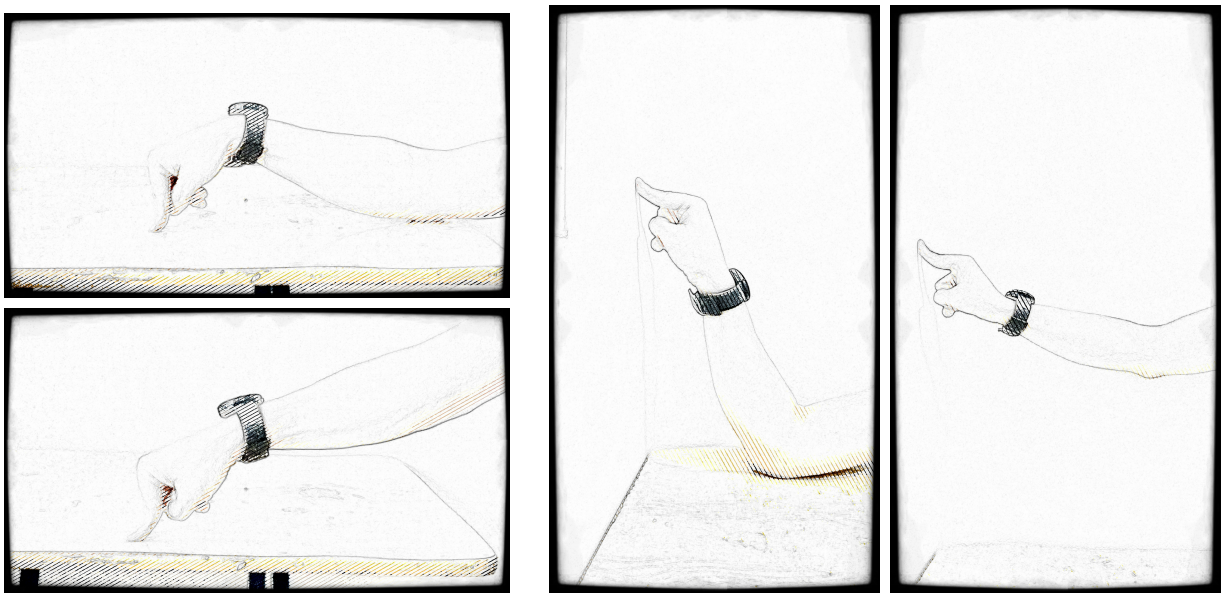
3. We have assumed that it takes the same time to input alphabetical letters, digits and punctuation symbols for the tap-on-screen method. But in reality, for the tap-on-screen method, extra taps are needed to switch the input mode for digits and punctuation symbols. In contrast, *SHOW* can recognize them directly without extra cost.

Therefore, we conclude that *SHOW*'s input efficiency in terms of time cost is comparable with the tap-on-screen method.

4.6.4 Performance on different surfaces

In this section, we evaluate *SHOW*'s character recognition performance on both horizontal and vertical surfaces. Furthermore, for each surface, we consider two scenarios: the elbow is supported by a surface or hanging in the air without support, as shown in Fig. 4.11. In Section. 4.6.2 we have shown that SVM can achieve best performance with the current

feature set and proven the effectiveness of rotation injection, thus in this test we use SVM with rotation injection by default. The test result is listed in Table. 4.5, from which we can



(a) Horizontal Surface. (b) Vertical surface

Figure 4.11: Writing with *SHOW* on different surfaces.

Table 4.5: *SHOW*'s character recognition performance on different surfaces.

Recognition Accuracy	Horizontal		Vertical	
	with support	w/o support	with support	w/o support
Single-person	99.9%	80.5%	87.3%	62.1%
Cross-person	99.8%	67.5%	71.3%	54.1%

find that, *SHOW* performs much better on a horizontal surface than on a vertical surface. Furthermore, if the elbow is hanging in the air without any support, the recognition accuracy will drop drastically. The reason is that the swinging elbow adds in interference and more freedom to the wrist's movement in the space, which significantly increases the difficulty of

capturing its patterns. Therefore, *SHOW* is best suited for use on a horizontal surface with the elbow as the support point.

4.7 Related Work

4.7.1 Writing Recognition

The methodologies for writing recognition can be divided into three categories: IMU sensor based, computer vision based and image recognition based methodologies. The first two capture the dynamic trace of the writing and recognize the writings afterwards. On the contrary, the third one takes in the writing result as a static image and analyzes accordingly.

IMU sensor based writing recognition. Shen et al. design ArmTrak which employs the IMU sensors on a smart watch to track the 3D posture of the entire arm. ArmTrak can recognize some hand gestures based on the arm posture changes. This implies that the relatively small movements in handwriting, which mostly involves only fingers and wrists, can not be detected easily by ArmTrak. In fact, ArmTrak reports a median error of 7.9 cm and 9.2 cm respectively for wrist and elbow location estimation. It is almost the size of an entire character written by the user. In many cases, the user has to input in a space-limited area (e.g. in vehicle), hence ArmTrak's measurement on the entire arm movement is not applicable. On contrast, *SHOW* assumes the user's input with the elbow as the support point, which only involves the movements of hands and forearms, hence can work nicely in these space-limited areas.

Xu et al. detect index finger writings on a flat surface using the smartwatch [111]. The sensor readings they acquire results from movements of index finger tendon, therefore it requires the watch to be tightly wrapped over the wrist. However, we observe that in real

life people do not wear the watch in a tight fashion. In this condition, we have shown that the recognition results using the feature set provided in [111] are not as good. *SHOW* on the other hand, takes the potential watch rotation into consideration and trains the model based on it. Our evaluation confirms *SHOW*'s effectiveness against such rotation interference.

The Airwriting system [126] also achieves handwriting recognition using wearable IMU sensors. However, it requires the sensor to be attached to the back of the hand, hence records much more distinguishable movements than the smart watch could. Apparently, this attaching position is not applicable for a smart watch. Besides, Airwriting uses special IMU sensors which can sample at 819.2 Hz, and it is far beyond the capability of the commodity smart watches, for whose sensors the highest sampling rate is around 100 Hz.

Computer vision based writing recognition. Another way of recognizing handwriting relies on computer vision techniques [127–131]. These techniques require a special device (e.g. Kinect) to capture the spatial-temporal traces. Unfortunately, even if the smart watch is becoming an independent device, it is not likely to be equipped with a camera in the near future. Thus, these techniques are not suitable for smart watches. However, these researches give valuable insights about the temporal characteristics within the trace. These insights can help IMU based systems because the sensory data they collect are also temporal data. In fact, [110] is actually using a Kinect to generate the ground truth about the gesture.

Image based writing recognition. A traditional way of learning about handwriting is based on the analysis of visible traces saved in an image [132–134]. This kind of work is also known as off-line handwriting recognition in literature [135]. This is a completely different methodology than what we employed in *SHOW*, because in this method, the temporal information of physical movements of the human body is completely hidden, and the image of writing traces are only the "results". However, research in this category has built up

a great knowledge base about the continuous handwriting, which is the most natural and efficient way of writing in daily life. All handwriting recognition systems based on IMU sensors like *SHOW* will have to deal with continuous writing to further push the limit of inputting efficiency.

4.7.2 Word Auto-Completion

Word auto-completion is an important speedup technique for both text-entry and search queries. Many researchers propose to explore the input context for extra information to improve the word suggestions [136,137]. This is viable in the database or web search scenario, where much public information such as the query history or click logs are ready for use, but not viable for text-entry method on a personal device, as the user may have privacy concern. However, if the user agrees to let *SHOW* log his/her input history, many context-exploration autocompletion models can directly be applied.

When considering error-tolerant autocompletion, some researchers proposed to define a general distance between the indexed words and the current input prefix, and all candidate words within this distance are presented as suggestions [138–140]. This approach will substantially extend the search space and increase the complexity. In *SHOW*, we use the classifier's confusion matrix as the prior knowledge to restrict possible ambiguous characters in a small set. In this way, the model is enhanced with a certain level of ambiguity tolerance, and the search space and complexity are kept at a low level. Our approach can be envisioned as a subset of such distance-based error-tolerant autocompletion.

4.8 Conclusion

We propose *SHOW*, a handwriting recognition system which provides a new and efficient way for text entry on the smart watch. By manually injecting rotation into the sensory data, *SHOW* effectively circumvents the problem of inconsistent watch wearing positions and the difficulties of collecting large amount of user writing samples, and greatly improves the recognition robustness and accuracy. *SHOW* also addresses the possible input ambiguity caused by mis-recognition and achieve word autocompletion. It has a much lower error rate than the tap-on-screen method and achieves comparable input efficiency.

Chapter 5

Conclusion and Discussion

In this thesis, we discussed the opportunities brought up by rapidly advanced mobile devices and various sensors aboard, and based on them proposed *LocMe* and *SHOW*, which addressed the infrastructure-free indoor localization and text-entry on smart watches from novel perspectives, respectively. Additionally, we conducted comprehensive experiments and had shown their advantages over existing work. Specifically, *LocMe* guarantees faster convergence than the wall-constrained only methods, and reduces the median of the localization error in our field tests by 68%. *SHOW* manages to seed large quantities of handwriting traces from one single user trace, and achieves over 70% lower mis-recognition-rate, 43% lower no-response-rate, and 33.3% higher word suggestion coverage than the tap-on-screen method using a virtual QWERTY keyboard. During the development of *LocMe* and *SHOW*, we also encounter a few new challenges, which are our future research directions and can be summarized as the following four parts.

5.1 The re-initialization problem for *LocMe*

For all particle filter based localization systems, they face a re-initialization problem. Because there are cases that all survival particles are incorrect so that the filter will never converge to the correct result, and the system has to re-initialize. There are two questions to be answered for re-initialization, i.e. “when” and “how”.

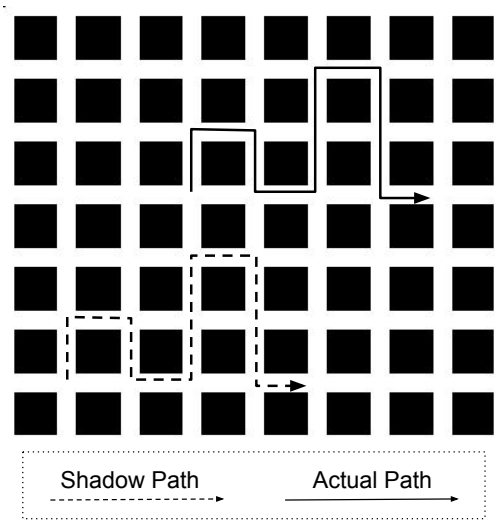


Figure 5.1: An example of shadow path in a building with a grid layout.

When to re-initialize. To find out when to re-initialize, we first need to understand why the particle filter may reach a stage that needs re-initialization.

During the development of *LocMe*, we notice a special kind of path, which we hereby refer to as the *shadow path*. As the name suggests, a *shadow path* is not the actual path the user travels through, but it always follows the user's actual path and has the same shape as illustrated in Fig. 5.1. In a wall-constraint only system, once the particle filter concentrates onto a group of particles on the shadow path, the wall-constraint validation can not give any hint about it, because for any point on the actual path, whether the user violates the wall-constraint or not, its corresponding particle on the shadow path will have exactly the same constraint validation result. Therefore, the filter itself can never realize it is on the shadow path without any external prompt, until the shadow path comes to an end. For *LocMe*, the use of locomotion-constraint brings in different POIs as check points along the path, thus it can break down long shadow paths into smaller pieces, and the filter can leap out of each shadow path earlier. However, there are still rare cases where no POIs are checked along a long shadow path or there are exactly the same POIs at the same positions on the shadow

path as on the actual path. In these rare cases, we need a prompt from the user to detect the shadow path.

In current work, we use the detection of shadow path in the filter as an indicator for re-initialization.

How to re-initialize. When the system decides to re-initialize, it should not just roll back to the state at the very beginning, where the system knows nothing about the user's location or locomotion state. Instead, plenty of information in the localization history can be used. Therefore, the system can re-initialize to a particular state in the history, then quickly converge to the correct location. In current work, we propose the following two re-initialize strategies.

- **Shift to the actual path**. The system will trace all particles in the current filter back to the previous POI, and shift the corresponding paths to other POIs of the same type and size, then validate the path again. If the shifted path survives all the constraint validation, it can be regarded as the actual path, and the newest particle on this path can be put into the re-initialized filter.

- **Seed from the previous POI**. If no actual paths are found, we can re-initialize the filter with the previous POI as a seed. In general, POIs work as special endpoints that break the long shadow path into pieces. Therefore, rolling back to previous POI has a great chance to take the particle filter out of the trap of shadow path. After that, a radius is computed by the number of steps and average step length since the last time the user reached that POI, and all particles within the circle centered at the previous POI will be evenly put back into the filter again.

5.2 Handwriting privacy leak

In the modern era, people interact with all kinds of digital devices and frequently input personal information on them. Most users had encountered with phishing websites, credit card fraud, spamming emails etc., and are vigilant about potential privacy leak when input on digital devices. In contrast, the confidence in the security of handwriting on paper is still high. With the proliferation of smart watches, and our progress with *SHOW*, it indicates that handwriting on paper is prone to information leak and no more safe. The challenge for this work is threefold. From an attacker's perspective,

1. The amount of handwriting data collected from the target user is extremely limited.

2. The handwriting can be cursive and continuous.

3. The handwriting traces are not labeled.

To address the first challenge, we need to find a way to generate extra handwriting traces. In fact, our work in *SHOW* has already demonstrated that, with the help of rotation injection, a large amount of robust handwriting traces from a limited sample set can be generated.

To address the second challenge, we propose a smart word separation algorithm which is based on the gesture of relocating the palm after a word is written.

To address the third challenge, we propose a crowd-sourcing based clustering method combined with word prediction algorithms used in $SHOW^2$, to generate guesses about the written content.

This work is in an early experiment stage, but our proposed solutions for challenge one and two have already shown promising results.

5.3 Cursive and Context-aware: A Smarter *SHOW*

SHOW makes the word prediction solely by using the current input characters as a prefix. But in reality, there is a lot of context information, such as the user's message history, application logs etc., that can be explored if *SHOW* is granted with the user's permission. Also, the words' semantic relations can also be utilized for prediction [141, 142]. Taking full use of these information resources, *SHOW* can be further improved for phrase or even sentence level suggestions.

At the current stage, *SHOW* accepts short pause as the soft separation to distinguish different characters, which is still counter-intuitive considering that people tend to handwrite in a cursive and continuous way, where a natural separation is usually at the word level. Therefore, accepting the handwriting word as a whole trace and extracting each character from it automatically would be the ideal recognition paradigm for *SHOW*, and is our major working direction.

5.4 A Chinese Version of *SHOW*

It would be nice to extend the work of *SHOW* to other languages. The English writing system is based on a limited set of symbols, which makes it possible to learn the user handwriting traces with respect to each letter. In contrast, the Chinese characters are hieroglyphic, and each one is different from the other. There are about 3500 commonly used characters in Chinese, making it impossible to learn the trace of each one. At current stage, we have conceived two possible strategies to deal with this challenge.

1. Tear each character into smaller units, i.e. the strokes. The basic types of strokes are limited, so we can still use the learning method. However, this means the user has to make a soft separation between each stroke, the input efficiency will be drastically reduced.

2. Postpone the recognition to the later word autocompletion stage. We can sacrifice the recognition accuracy at the character level, where *SHOW* only gives a best-effort guess list of the possible input, and relies on the intra-word relation between subsequent characters to reduce the ambiguity. Unfortunately, as the Chinese words are usually short (typically consist of two or three characters), such intra-word relation is also limited.

Our early stage tests demonstrate the effectiveness of the two strategies, but the overall performance is still much lower than *SHOW* for English. We consider adapting *SHOW* to Chinese and achieving a comparable performance with English as our major research direction.

References

[1] Oakpointe. Stair anatomy. `https://www.stairpartsandmore.com/stair-anatomy/`, 2017.

[2] IDC. Worldwide quarterly mobile phone tracker. 2018.

[3] IDC. Worldwide quarterly wearable device tracker. 2018.

[4] Xinye Lin, Xiao-Wen Chang, and Xue Liu. LocMe: Human locomotion and map exploitation based indoor localization. In *2017 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 131–140, March 2017.

[5] Xinye Lin, Yixin Chen, Xiao-Wen Chang, Xue Liu, and Xiaodong Wang. SHOW: Smart Handwriting on Watches. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(4):151:1–151:23, January 2018.

[6] Swarun Kumar, Stephanie Gil, Dina Katabi, and Daniela Rus. Accurate Indoor Localization with Zero Start-up Cost. In *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*, MobiCom '14, pages 483–494, New York, NY, USA, 2014. ACM.

[7] Deepak Vasisht, Swarun Kumar, and Dina Katabi. Decimeter-level localization with a single wifi access point. In *Proceedings of the 13th USENIX Conference on Networked Systems Design and Implementation*, NSDI '16, pages 165–178, 2016.

[8] Manikanta Kotaru, Kiran Joshi, Dinesh Bharadia, and Sachin Katti. SpotFi: Decimeter Level Localization Using WiFi. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 269–282, New York, NY, USA, 2015. ACM.

[9] Deepak Vasisht, Swarun Kumar, and Dina Katabi. Decimeter-Level Localization with a Single WiFi Access Point. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 165–178, 2016.

[10] Liqun Li, Pan Hu, Chunyi Peng, Guobin Shen, and Feng Zhao. Epsilon: a visible light based positioning system. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI '14, pages 331–343. USENIX Association, 2014.

[11] Ye-Sheng Kuo, Pat Pannuto, Ko-Jen Hsiao, and Prabal Dutta. Luxapose: Indoor Positioning with Mobile Phones and Visible Light. In *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*, MobiCom '14, pages 447–458, New York, NY, USA, 2014. ACM.

[12] A. Komninos and M. Dunlop. Text Input on a Smart Watch. *IEEE Pervasive Computing*, 13(4):50–58, October 2014.

[13] Touchone. Touchone keyboard. `http://www.touchone.net/`, 2017.

[14] Muhammad Shoaib, Stephan Bosch, Ozlem Durmaz Incel, Hans Scholten, and Paul J. M. Havinga. A Survey of Online Activity Recognition Using Mobile Phones. *Sensors*, 15(1):2059–2085, January 2015.

[15] A. Moncada-Torres, K. Leuenberger, R. Gonzenbach, A. Luft, and R. Gassert. Activity classification based on inertial and barometric pressure sensors at different anatomical locations. *Physiological Measurement*, 35(7):1245, 2014.

[16] Jorge-L. Reyes-Ortiz, Luca Oneto, Albert Samà, Xavier Parra, and Davide Anguita. Transition-Aware Human Activity Recognition Using Smartphones. *Neurocomputing*, 171:754–767, January 2016.

[17] Akram Bayat, Marc Pomplun, and Duc A. Tran. A Study on Human Activity Recognition Using Accelerometer Data from Smartphones. *Procedia Computer Science*, 34:450–457, January 2014.

[18] Jian Bo Yang, Minh Nhut Nguyen, Phyo Phyo San, Xiao Li Li, and Shonali Krishnaswamy. Deep Convolutional Neural Networks on Multichannel Time Series for Human Activity Recognition. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 3995–4001, Buenos Aires, Argentina, 2015. AAAI Press.

[19] Muhammad Shoaib, Stephan Bosch, Ozlem Durmaz Incel, Hans Scholten, and Paul J. M. Havinga. Fusion of Smartphone Motion Sensors for Physical Activity Recognition. *Sensors*, 14(6):10146–10176, June 2014.

[20] Yonggang Lu, Ye Wei, Li Liu, Jun Zhong, Letian Sun, and Ye Liu. Towards unsupervised physical activity recognition using smartphone accelerometers. *Multimedia Tools and Applications*, 76(8):10701–10719, April 2017.

[21] Yongjin Kwon, Kyuchang Kang, and Changseok Bae. Unsupervised learning for human activity recognition using smartphone sensors. *Expert Systems with Applications*, 41(14):6067–6074, October 2014.

[22] Charissa Ann Ronao and Sung-Bae Cho. Human activity recognition with smartphone sensors using deep learning neural networks. *Expert Systems with Applications*, 59:235–244, 2016.

[23] Newzoo. Global mobile market report, 2017.

[24] WorldoMeters. World population clock. `http://www.worldometers.info/world-population/`, 2017.

[25] Cindy Liu. Worldwide internet and mobile user: Emarketer's updated estimates for 2015, 2015.

[26] AppBrain. Number of available applications in the google play store from december 2009 to december 2017. `http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/`, 2017.

[27] Apple and AppleInsider. Number of available apps in the apple app store from july 2008 to january 2017. `http://www.statista.com/statistics/263795/number-of-available-apps-in-the-apple-app-store/`, 2017.

[28] ComScore. U.s. cross-platform future in focus, 2017.

[29] Charles Newark-French. Mobile apps put the web in their rear-view mirror. `http://flurrymobile.tumblr.com/post/113367503685/mobile-apps-put-the-web-in-their-rear-view-mirror`, 2011.

[30] Xinye Lin, Xiao Xia, Shaohe Lv, and Xiaodong Wang. Reserach on the predictability of mobile app usage. In *The 7th Joint Conference on Harmonious Human Machine Environment*, September 2011.

[31] Xiao Xia, Xinye Lin, Xiaodong Wang, Xingming Zhou, and Deke Guo. Apps at hand: Personalized live homescreen based on mobile app usage prediction. *IEICE Transactions on Information and Systems*, E96.D(12):2860–2864, 2013.

[32] Craig Aszkler. Acceleration, shock and vibration sensors. In *Sensor Technology Handbook*, pages 137–159. Elsevier, 2005.

[33] Weixi Gu, Yuxun Zhou, Zimu Zhou, Xi Liu, Han Zou, Pei Zhang, Costas J. Spanos, and Lin Zhang. SugarMate: Non-intrusive Blood Glucose Monitoring with Smartphones. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(3):54:1–54:27, September 2017.

[34] Tian Hao, Chongguang Bi, Guoliang Xing, Roxane Chan, and Linlin Tu. MindfulWatch: A Smartwatch-Based System For Real-Time Respiration Monitoring During Meditation. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(3):57:1–57:19, September 2017.

[35] Reham Mohamed and Moustafa Youssef. HeartSense: Ubiquitous Accurate Multi-Modal Fusion-based Heart Rate Estimation Using Smartphones. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(3):97:1–97:18, September 2017.

[36] Xiao Zhang, Wenzhong Li, Xu Chen, and Sanglu Lu. MoodExplorer: Towards Compound Emotion Detection via Smartphone Sensing. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(4):176:1–176:30, January 2018.

[37] WHO Ageing and LC Unit. Who global report on falls prevention in older age. *World Health Organization*, 2008.

[38] Jeffrey M Rothschild, David W Bates, and Lucian L Leape. Preventable medical injuries in older patients. *Archives of internal medicine*, 160(18):2717–2728, 2000.

[39] J. K. Lee, S. N. Robinovitch, and E. J. Park. Inertial Sensing-Based Pre-Impact Detection of Falls Involving Near-Fall Scenarios. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 23(2):258–266, March 2015.

[40] L. J. Kau and C. S. Chen. A Smart Phone-Based Pocket Fall Accident Detection, Positioning, and Rescue System. *IEEE Journal of Biomedical and Health Informatics*, 19(1):44–56, January 2015.

[41] B. Aguiar, T. Rocha, J. Silva, and I. Sousa. Accelerometer-based fall detection for smartphones. In *2014 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*, pages 1–6, June 2014.

[42] Rui Wang, Weichen Wang, Min S. H. Aung, Dror Ben-Zeev, Rachel Brian, Andrew T. Campbell, Tanzeem Choudhury, Marta Hauser, John Kane, Emily A. Scherer, and Megan Walsh. Predicting Symptom Trajectories of Schizophrenia Using Mobile Sensing. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(3):110:1–110:24, September 2017.

[43] Katrin Plaumann, Milos Babic, Tobias Drey, Witali Hepting, Daniel Stooss, and Enrico Rukzio. Improving Input Accuracy on Smartphones for Persons Who Are Affected by Tremor Using Motion Sensors. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(4):156:1–156:30, January 2018.

[44] Katrin Plaumann, Milos Babic, Tobias Drey, Witali Hepting, Daniel StooSS, and Enrico Rukzio. Towards Improving Touchscreen Input Speed and Accuracy on Smartphones for Tremor Affected Persons. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, UbiComp '16, pages 357–360, New York, NY, USA, 2016. ACM.

[45] Benoit Carignan, Jean-François Daneault, and Christian Duval. Measuring Tremor with a Smartphone. In *Mobile Health Technologies*, Methods in Molecular Biology, pages 359–374. Humana Press, New York, NY, 2015.

[46] Alan Michael Woods, Mariusz Nowostawski, Elizabeth A. Franz, and Martin Purvis. Parkinsons disease and essential tremor classification on mobile device. *Pervasive and Mobile Computing*, 13:1–12, August 2014.

[47] M. Mielke and R. Brueck. Design and evaluation of a smartphone application for non-speech sound awareness for people with hearing loss. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 5008–5011, August 2015.

[48] Liu Sicong, Zhou Zimu, Du Junzhao, Shangguan Longfei, Jun Han, and Xin Wang. UbiEar: Bringing Location-independent Sound Awareness to the Hard-of-hearing People with Smartphones. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(2):17:1–17:21, June 2017.

[49] Mario A. Gutierrez, Michelle L. Fast, Anne H. Ngu, and Byron J. Gao. Real-Time Prediction of Blood Alcohol Content Using Smartwatch Sensor Data. In *Smart Health*, Lecture Notes in Computer Science, pages 175–186. Springer, Cham, November 2015.

[50] Hanlu Ye, Meethu Malu, Uran Oh, and Leah Findlater. Current and Future Mobile and Wearable Device Use by People with Visual Impairments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 3123–3132, New York, NY, USA, 2014. ACM.

[51] Radu-Daniel Vatavu. Visual Impairments and Mobile Touchscreen Interaction: State-of-the-Art, Causes of Visual Impairment, and Design Guidelines. *International Journal of Human-Computer Interaction*, 33(6):486–509, June 2017.

[52] Yiqin Lu, Chun Yu, Xin Yi, Yuanchun Shi, and Shengdong Zhao. BlindType: Eyes-Free Text Entry on Handheld Touchpad by Leveraging Thumb's Muscle Memory. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(2):18:1–18:24, June 2017.

[53] Martin Weigel and Jürgen Steimle. DeformWear: Deformation Input on Tiny Wearable Devices. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(2):28:1–28:23, June 2017.

[54] Shaikh Shawon Arefin Shimon, Courtney Lutton, Zichun Xu, Sarah Morrison-Smith, Christina Boucher, and Jaime Ruiz. Exploring Non-touchscreen Gestures for Smartwatches. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 3822–3833, New York, NY, USA, 2016. ACM.

[55] Youli Chang, Sehi L'Yi, Kyle Koh, and Jinwook Seo. Understanding Users' Touch Behavior on Large Mobile Touch-Screens and Assisted Targeting by Tilting Gesture. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 1499–1508, New York, NY, USA, 2015. ACM.

[56] Cheng Zhang, Junrui Yang, Caleb Southern, Thad E. Starner, and Gregory D. Abowd. WatchOut: Extending Interactions on a Smartwatch with Inertial Sensing. In *Proceedings of the 2016 ACM International Symposium on Wearable Computers*, ISWC '16, pages 136–143, New York, NY, USA, 2016. ACM.

[57] Ke-Yu Chen, Rahul C. Shah, Jonathan Huang, and Lama Nachman. Mago: Mode of Transport Inference Using the Hall-Effect Magnetic Sensor and Accelerometer. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(2):8:1–8:23, June 2017.

[58] Yu Guan and Thomas Plötz. Ensembles of Deep LSTM Learners for Activity Recognition Using Wearables. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(2):11:1–11:28, June 2017.

[59] Sangwon Bae, Denzil Ferreira, Brian Suffoletto, Juan C. Puyana, Ryan Kurtz, Tammy Chung, and Anind K. Dey. Detecting Drinking Episodes in Young Adults Using Smartphone-based Sensors. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(2):5:1–5:36, June 2017.

[60] Landu Jiang, Xinye Lin, Xue Liu, Chongguang Bi, and Guoliang Xing. SafeDrive: Detecting Distracted Driving Behaviors Using Wrist-Worn Devices. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(4):144:1–144:22, January 2018.

[61] Chongguang Bi, Jun Huang, Guoliang Xing, Landu Jiang, Xue Liu, and Minghua Chen. Safewatch: A wearable hand motion tracking system for improving driving safety. In *Internet-of-Things Design and Implementation (IoTDI), 2017 IEEE/ACM Second International Conference on*, pages 223–232. IEEE, 2017.

[62] C. Karatas, L. Liu, H. Li, J. Liu, Y. Wang, S. Tan, J. Yang, Y. Chen, M. Gruteser, and R. Martin. Leveraging wearables for steering and driver tracking. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, April 2016.

[63] Dongyao Chen, Kyong-Tak Cho, Sihui Han, Zhizhuo Jin, and Kang G. Shin. Invisible Sensing of Vehicle Steering with Smartphones. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '15, pages 1–13, New York, NY, USA, 2015. ACM.

[64] Xiao Sun, Li Qiu, Yibo Wu, Yeming Tang, and Guohong Cao. SleepMonitor: Monitoring Respiratory Rate and Body Position During Sleep Using Smartwatch. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(3):104:1–104:22, September 2017.

[65] Jun-Ki Min, Afsaneh Doryab, Jason Wiese, Shahriyar Amini, John Zimmerman, and Jason I. Hong. Toss 'N' Turn: Smartphone As Sleep and Sleep Quality Detector. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 477–486, New York, NY, USA, 2014. ACM.

[66] Zhenyu Chen, M. Lin, Fanglin Chen, N. D. Lane, G. Cardone, Rui Wang, Tianxing Li, Yiqiang Chen, T. Choudhury, and A. T. Campbell. Unobtrusive sleep monitoring using smartphones. In *2013 7th International Conference on Pervasive Computing Technologies for Healthcare and Workshops*, pages 145–152, May 2013.

[67] Keunseo Kim, Hengameh Zabihi, Heeyoung Kim, and Uichin Lee. TrailSense: A Crowdsensing System for Detecting Risky Mountain Trail Segments with Walking Pattern Analysis. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(3):65:1–65:31, September 2017.

[68] A. Bajpai, V. Jilla, V. N. Tiwari, S. M. Venkatesan, and R. Narayanan. Quantifiable fitness tracking using wearable devices. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 1633–1637, August 2015.

[69] Dan Morris, T. Scott Saponas, Andrew Guillory, and Ilya Kelner. RecoFit: Using a Wearable Sensor to Find, Recognize, and Count Repetitive Exercises. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 3225–3234, New York, NY, USA, 2014.

[70] S. Bhattacharya and N. D. Lane. From smart to deep: Robust activity recognition on smartwatches using deep learning. In *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pages 1–6, March 2016.

[71] Yuki Kubo, Ryosuke Takada, Buntarou Shizuki, and Shin Takahashi. Exploring Context-Aware User Interfaces for Smartphone-Smartwatch Cross-Device Interaction. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(3):69:1–69:21, September 2017.

[72] G. M. Weiss, J. L. Timko, C. M. Gallagher, K. Yoneda, and A. J. Schreiber. Smartwatch-based activity recognition: A machine learning approach. In *2016 IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*, pages 426–429, February 2016.

[73] Anshul Rai, Krishna Kant Chintalapudi, Venkata N. Padmanabhan, and Rijurekha Sen. Zee: Zero-effort crowdsourcing for indoor localization. In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, Mobicom '12, pages 293–304, New York, NY, USA, 2012. ACM.

[74] Oliver Woodman and Robert Harle. Pedestrian Localisation for Indoor Environments. In *Proceedings of the 10th International Conference on Ubiquitous Computing*, UbiComp '08, pages 114–123, New York, NY, USA, 2008. ACM.

[75] Fan Li, Chunshui Zhao, Guanzhong Ding, Jian Gong, Chenxing Liu, and Feng Zhao. A reliable and accurate indoor localization method using phone inertial sensors. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 421–430. ACM, 2012.

[76] H. Nurminen, A. Ristimaki, S. Ali-Loytty, and R. Piche. Particle filter and smoother for indoor localization. In *2013 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–10, October 2013.

[77] He Wang, Souvik Sen, Ahmed Elgohary, Moustafa Farid, Moustafa Youssef, and Romit Roy Choudhury. No Need to War-drive: Unsupervised Indoor Localization. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, pages 197–210, New York, NY, USA, 2012. ACM.

[78] Haibo Ye, Tao Gu, Xiaorui Zhu, Jinwei Xu, Xianping Tao, Jian Lu, and Ning Jin. Ftrack: Infrastructure-free floor localization via mobile phone sensing. In *Pervasive Computing and Communications (PerCom), 2012 IEEE International Conference on*, PerCom '12, pages 2–10. IEEE, March 2012.

[79] Haibo Ye, Tao Gu, Xianping Tao, and Jian Lu. B-loc: Scalable floor localization using barometer on smartphone. In *Mobile Ad Hoc and Sensor Systems (MASS), 2014 IEEE 11th International Conference on*, pages 127–135. IEEE, 2014.

[80] V. Radu and M. K. Marina. HiMLoc: Indoor smartphone localization via activity aware Pedestrian Dead Reckoning with selective crowdsourced WiFi fingerprinting. In *2013 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–10, October 2013.

[81] Melania Susi, Valérie Renaudin, and Gérard Lachapelle. Motion mode recognition and step detection algorithms for mobile phone users. *Sensors*, 13(2):1539–1562, 2013.

[82] Wiebren Zijlstra and At L Hof. Assessment of spatio-temporal gait parameters from trunk accelerations during human walking. *Gait & Posture*, 18(2):1–10, October 2003.

[83] T. von Buren, P.D. Mitcheson, T.C. Green, E.M. Yeatman, A.S. Holmes, and G. Troster. Optimization of inertial micropower generators for human walking motion. *IEEE Sensors Journal*, 6(1):28–38, February 2006.

[84] A.T.M. Willemsen, F. Bloemhof, and H.B.K. Boom. Automatic stance-swing phase detection from accelerometer data for peronealcmerve stimulation. *IEEE Transactions on Biomedical Engineering*, 37(12):1201–1208, December 1990.

[85] Jun Yang. Toward physical activity diary: Motion recognition using simple acceleration features with mobile phones. In *Proceedings of the 1st International Workshop on Interactive Multimedia for Consumer Electronics*, IMCE '09, pages 1–10, New York, NY, USA, 2009. ACM.

[86] Sheng Zhong, Li Wang, A.M. Bernardos, and Mei Song. An accurate and adaptive pedometer integrated in mobile health application. In *IET International Conference on Wireless Sensor Network, 2010. IET-WSN*, pages 78–83, November 2010.

[87] Rolf Moe-Nilssen and Jorunn L. Helbostad. Estimation of gait cycle characteristics by trunk accelerometry. *Journal of Biomechanics*, 37(1):121–126, January 2004.

[88] Chihiro Mizuike, Shohei Ohgi, and Satoru Morita. Analysis of stroke patient walking dynamics using a tri-axial accelerometer. *Gait & Posture*, 30(1):60–64, July 2009.

[89] Justin J. Kavanagh and Hylton B. Menz. Accelerometry: A technique for quantifying movement patterns during walking. *Gait & Posture*, 28(1):1–15, July 2008.

[90] Nirupam Roy, He Wang, and Romit Roy Choudhury. I Am a Smartphone and I Can Tell My User's Walking Direction. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '14, pages 329–342, New York, NY, USA, 2014. ACM.

[91] Encyclopædia Britannica. Diagram of an escalator, 207.

[92] J. D. Hol, T. B. Schon, and F. Gustafsson. On Resampling Algorithms for Particle Filters. In *2006 IEEE Nonlinear Statistical Signal Processing Workshop*, pages 79–82, September 2006.

[93] M. I. Shamos and D. Hoey. Geometric intersection problems. In *, 17th Annual Symposium on Foundations of Computer Science, 1976*, pages 208–215, October 1976.

[94] 2015 international building code. `http://codes.iccsafe.org/app/book/toc/2015/I-Codes/2015%20IBC%20HTML/index.html`.

[95] Moustafa Youssef and Ashok Agrawala. The horus WLAN location determination system. In *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, MobiSys '05, pages 205–218. ACM, 2005.

[96] Google. A new frontier for google maps: mapping the indoors. http://googleblog.blogspot.ca/2011/11/new-frontier-for-google-maps-mapping.html, 2011.

[97] Rahul Desai. Nokia leads the way with indoor mapping. http://360.here.com/2012/07/16/nokia-leads-the-way-with-indoor-mapping/, 2012.

[98] Marek Strassenburg-Kleciak. Openstreetmap to create indoor maps. http://geospatialworld.net/Magazine/MArticleView.aspx?aid=31102, 2014.

[99] Google. Google maps. https://www.google.com/maps/about/partners/indoormaps/, 2015.

[100] David A Winter, Aftab E Patla, and James S Frank. Assessment of balance control in humans. *Med Prog Technol*, 16(1-2):31–51, 1990.

[101] D. Gusenbauer, C. Isert, and J. Krösche. Self-contained indoor positioning on off-the-shelf mobile devices. In *2010 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–9, September 2010.

[102] J. Yin, Q. Yang, and J. J. Pan. Sensor-Based Abnormal Human-Activity Detection. *IEEE Transactions on Knowledge and Data Engineering*, 20(8):1082–1090, August 2008.

[103] Stephen J. Preece, John Y. Goulermas, Laurence P. J. Kenney, Dave Howard, Kenneth Meijer, and Robin Crompton. Activity identification using body-mounted sensorsa review of classification techniques. *Physiological Measurement*, 30(4):R1, April 2009.

[104] A. Varshavsky, Anthony LaMarca, Jeffrey Hightower, and E. de Lara. The SkyLoc Floor Localization System. In *Fifth Annual IEEE International Conference on Pervasive Computing and Communications, 2007. PerCom '07*, pages 125–134, March 2007.

[105] S. Gansemer, S. Hakobyan, S. Puschel, and U. Grosmann. 3d WLAN indoor positioning in multi-storey buildings. In *IEEE International Workshop on Intelligent Data Acquisition*

and Advanced Computing Systems: Technology and Applications, 2009. IDAACS 2009, pages 669–672, September 2009.

[106] F. Alsehly, T. Arslan, and Z. Sevak. Indoor positioning with floor determination in multi story buildings. In *2011 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–7, September 2011.

[107] Hung-Huan Liu and Yu-Non Yang. WiFi-based indoor positioning for multi-floor Environment. In *TENCON 2011 - 2011 IEEE Region 10 Conference*, pages 597–601, November 2011.

[108] S. Vanini and S. Giordano. Adaptive context-agnostic floor transition detection on smart mobile devices. In *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 2–7, March 2013.

[109] Android. Android wear 2.0 developer preview. `https://developer.android.com/wear/preview/index.html`, 2016.

[110] Sheng Shen, He Wang, and Romit Roy Choudhury. I am a Smartwatch and I can Track my User's Arm. In *Proceedings of the 14th annual international conference on Mobile systems, applications, and services*, 2016.

[111] Chao Xu, Parth H. Pathak, and Prasant Mohapatra. Finger-writing with Smartwatch: A Case for Finger and Hand Gesture Recognition Using Smartwatch. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, HotMobile'15, pages 9–14, New York, NY, USA, 2015. ACM.

[112] Lorenzo Porzi, Stefano Messelodi, Carla Mara Modena, and Elisa Ricci. A Smart Watch-based Gesture Recognition System for Assisting People with Visual Impairments. In *Proceedings of the 3rd ACM International Workshop on Interactive Multimedia on Mobile & Portable Devices*, IMMPD '13, pages 19–24, New York, NY, USA, 2013. ACM.

[113] D. Moazen, S. A. Sajjadi, and A. Nahapetian. AirDraw: Leveraging smart watch motion sensors for mobile human computer interactions. In *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 442–446, January 2016.

[114] O J Lewis, R J Hamshere, and T M Bucknill. The anatomy of the wrist joint. *Journal of Anatomy*, 106(Pt 3):539–552, May 1970.

[115] Emmanuel Munguia Tapia. *Using machine learning for real-time activity recognition and estimation of energy expenditure*. Phd thesis, Massachusetts Institute of Technology, 2008.

[116] Thomas G. Dietterich. An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. *Machine Learning*, 40(2):139–157, August 2000.

[117] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1, August 1995.

[118] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, March 2002.

[119] Koby Crammer and Yoram Singer. On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines. *J. Mach. Learn. Res.*, 2:265–292, March 2002.

[120] G. Zipf. *Selective Studies and the Principle of Relative Frequency in Language*. Harvard University Press, Cambridge, MA, 1932.

[121] Joshua Goodman, Gina Venolia, Keith Steury, and Chauncey Parker. Language Modeling for Soft Keyboards. In *Proceedings of the 7th International Conference on Intelligent User Interfaces*, IUI '02, pages 194–195, New York, NY, USA, 2002. ACM.

[122] Afsaneh Fazly and Graeme Hirst. Testing the Efficacy of Part-of-speech Information in Word Completion. In *Proceedings of the 2003 EACL Workshop on Language Modeling for Text*

Entry Methods, TextEntry '03, pages 9–16, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[123] Christopher P. Willmore, Nicholas K. Jong, and Justin S. HOGG. Text prediction using combined word n-gram and unigram language models, December 2015.

[124] first20hours. google-10000-list. `https://github.com/first20hours/google-10000-english`, 2017.

[125] I. Scott MacKenzie and R. William Soukoreff. Phrase Sets for Evaluating Text Entry Techniques. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '03, pages 754–755, New York, NY, USA, 2003. ACM.

[126] Christoph Amma, Marcus Georgi, and Tanja Schultz. Airwriting: a wearable handwriting recognition system. *Personal and Ubiquitous Computing*, 18(1):191–203, February 2013.

[127] Sharad Vikram, Lei Li, and Stuart Russell. Handwriting and Gestures in the Air, Recognizing on the Fly. In *Proceedings of the CHI*, volume 13, 2013.

[128] X. Zhang, Z. Ye, L. Jin, Z. Feng, and S. Xu. A New Writing Experience: Finger Writing in the Air Using a Kinect Sensor. *IEEE MultiMedia*, 20(4):85–93, October 2013.

[129] K. K. Biswas and S. K. Basu. Gesture recognition using Microsoft Kinect #x00ae;. In *The 5th International Conference on Automation, Robotics and Applications*, pages 100–103, December 2011.

[130] Yi Li. Hand gesture recognition using Kinect. In *2012 IEEE International Conference on Computer Science and Automation Engineering*, pages 196–199, June 2012.

[131] Zhou Ren, Jingjing Meng, Junsong Yuan, and Zhengyou Zhang. Robust Hand Gesture Recognition with Kinect Sensor. In *Proceedings of the 19th ACM International Conference on Multimedia*, MM '11, pages 759–760, New York, NY, USA, 2011. ACM.

[132] Louis Vuurpijl and Lambert Schomaker. *Coarse Writing-Style Clustering Based on Simple Stroke-Related Features*. 1996.

[133] H. Bunke, M. Roth, and E. G. Schukat-Talamazzini. Off-line cursive handwriting recognition using hidden markov models. *Pattern Recognition*, 28(9):1399–1413, September 1995.

[134] Alex Graves and Juergen Schmidhuber. Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 545–552. Curran Associates, Inc., 2009.

[135] R. Plamondon and S. N. Srihari. Online and off-line handwriting recognition: a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):63–84, January 2000.

[136] Ziv Bar-Yossef and Naama Kraus. Context-sensitive Query Auto-completion. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 107–116, New York, NY, USA, 2011. ACM.

[137] Liangda Li, Hongbo Deng, Anlei Dong, Yi Chang, Ricardo Baeza-Yates, and Hongyuan Zha. Exploring Query Auto-Completion and Click Logs for Contextual-Aware Web Search and Query Suggestion. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 539–548, Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee.

[138] Surajit Chaudhuri and Raghav Kaushik. Extending Autocompletion to Tolerate Errors. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 707–718, New York, NY, USA, 2009. ACM.

[139] Shengyue Ji, Guoliang Li, Chen Li, and Jianhua Feng. Efficient Interactive Fuzzy Keyword Search. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 371–380, New York, NY, USA, 2009. ACM.

[140] Guoliang Li, Shengyue Ji, Chen Li, and Jianhua Feng. Efficient Fuzzy Full-text Type-ahead Search. *The VLDB Journal*, 20(4):617–640, August 2011.

[141] Kenneth C. Arnold, Krzysztof Z. Gajos, and Adam T. Kalai. On Suggesting Phrases vs. Predicting Words for Mobile Text Composition. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16, pages 603–608, New York, NY, USA, 2016. ACM.

[142] Nestor Garay-Vitoria and Julio Abascal. Text prediction systems: a survey. *Universal Access in the Information Society*, 4(3):188–203, March 2006.