FTrack: Infrastructure-free Floor Localization via Mobile Phone Sensing

Haibo Ye[1], Tao Gu[2], Xiaorui Zhu[1], Jinwei Xu[1], Xianping Tao[1], Jian Lu[1], Ning Jin[2]
[1]State Key Laboratory for Novel Software Technology
Nanjing University, Nanjing 210093, China
{yhb, zxr, xjw}@smail.nju.edu.cn, {txp, lj}@nju.edu.cn
[2]Department of Mathematics and Computer Science
University of Southern Denmark, Odense M, DK-5230, Denmark
{gu,njin}@imada.sdu.dk

***Abstract*—Mobile phone localization plays a key role in the fast-growing Location Based Applications domain. Most of the existing localization schemes rely on infrastructure support such as GSM, WiFi or GPS. In this paper, we present FTrack, a novel floor localization system to identify the floor level in a multi-floor building on which a mobile user is located. FTrack uses the mobile phone's accelerometer only without any infrastructure support. It does not require any prior knowledge of the building such as floor height. By capturing user encounters and analyzing user trails, FTrack finds the mapping from the traveling time (when taking the elevator) or the step counts (when walking on the stairs) between any two floors to the number of floor levels. The mapping can then be used for mobile users to pinpoint their current floor levels. We conduct both simulation and field studies to demonstrate the effectiveness of FTrack. Our field trial in a 10-floor building shows that FTrack achieves an accuracy of over 90% after two hours in our experiment.**

***Keywords*-Mobile Phone Localization, Floor Localization, Accelerometer.**

I. Introduction

With the increasing pervasiveness of mobile phones, we have witnessed the rapid growth of location based applications (LBAs) in recent years. Such applications can be micro-blogging [1], location based advertising [2], local traffic [3], etc. In a multi-floor building environment, knowing the floor level of a mobile user is particularly useful to a variety of LBAs. For example, in location based advertising, advertisements can be delivered to mobile users based on their floor levels in a shopping mall. In emergency situations, locating the floor level of a user fast and accurately can be very critical for emergency services such as medical assistance or life saving. This problem is known as the *floor localization* problem, determining the floor level in a building on which a mobile user is located.

Localization has attracted extensive research efforts in recent years. A variety of indoor localization methods [4], [5], [6], [7] has been proposed. Techniques based on radio signals from WiFi access points or GSM base stations have shown promising. Radar [5] operates on WiFi fingerprints, and it is capable of achieving up to an accuracy of 5m in indoor environments. However, even such an idealized scheme may not be adequate to identify the floor level accurately because it can easily lead to confusion between two neighbor floors. PlaceLab [4] uses WiFi and GSM signals. The idea is to war-drive an area to create a radio map of existing WiFi/GSM access points, and a mobile phone localizes itself by comparing overheard WiFi/GSM access points against the radio map. However, the limitations of these methods have been well studied.

First, WiFi/GSM based localization relies on pre-installed infrastructure under dense deployment. Such infrastructures may not be available in many buildings. In addition, studies have shown that WiFi signal strengths vary significantly throughout the day and these variations are not uniform [8]. An infrastructure-free solution would be highly desirable for floor localization. Second, existing methods typically require calibration which is expensive and not scalable. Radar needs to carefully calibrate WiFi signal strengths at many locations, and the calibration process is time-consuming and may not scale over large areas. War-driving in PlaceLab is also intolerably time-consuming in order to cover a large portion of space. The recurring cost in war-driving can be excessive and undesirable. An ideal solution should be cheap and scalable.

A variety of sensors embedded in smartphones has recently motivated the research on sensor-assisted localization methods [9], [10], [11]. The basic idea is to leverage the mobile phone's accelerometer and electronic compass to measure the walking speed and orientation of the mobile user. The user's location can be easily computed by double-integrating the acceleration readings over time. However, electronic compasses and accelerometers are highly noisy [12], causing the user location to diverge from the actual location. Constandache et al. [9], [10] propose a better approach by identifying acceleration signatures in human walking patterns (i.e., the nature up and down bounce), and then multiplying step count with the user's step size to obtain the location. However, Escort [9] leverages fixed beacons for constantly calibration, and CompAcc [10] makes use of possible walking paths extracted from Google Maps [13].

In this paper, we present FTrack, an infrastructure-free floor localization scheme. FTrack makes use of the mobile phone's accelerometer only, and does not require any infrastructure support (i.e., WiFi/GSM). Using acceleration data, a simple

978-1-4673-0258-6/12/$31.00 ©2012 IEEE

solution may work based on the following two observations. First, when people walk up and down the stairs the same number of floors, the number of step counts is more or less the same. By knowing the height of each floor level, we can easily compute the number of floors a user travels. Similarly, when people take the elevator to go up and down the same number of floors, the traveling time of the elevator is the same (assuming no stop in between). By computing the moving speed of the elevator based on accelerometer, we can obtain the number of floors the user travels, assuming the height of each floor is known. However, the knowledge about floor height has to be made available to the user. This may not be feasible in practice. More importantly, it may not be scalable since floor height may be different from one building to another. We propose a novel solution in FTrack which does not need any prior knowledge of the building.

Our idea works as follows. The floor height may be different from floor to floor (e.g., this is particularly the case of a shopping mall where the ground floor is higher than other floors). When people take elevator, we can easily detect if the user goes up or down and record the traveling time of the elevator based on accelerometer. We want to create a table which maps the traveling time between any two floor levels to the number of floors the user travels. If the initial floor level of a user is known in advance or the current floor level has been discovered by FTrack (e.g., typically the ground floor), by measuring the traveling time of the elevator, we can look up the table to find out the user's floor level. The same principle can be applied to walking up/down stairs. In this case, the mapping is done from the number of step counts to the number of floor levels the user travels. Since the step counts of a user can be computed using accelerometer, we can find out the current floor easily.

FTrack creates the mapping table leveraging user interaction. In a building environment, people often encounter each other on any floor or in the elevators. User trails are logged when they travel. By examining information such as user encounters, going up /down, users joining and leaving in the trails, FTrack generates the mapping table, which can be then used for mobile users to pinpoint their current floor levels.

In summary, this paper makes the following contributions.

1) We propose a novel floor localization system, named FTrack, which identifies the floor level on which a mobile user is located. FTrack makes use of the mobile phone's accelerometer only, and it does not rely on any infrastructure such as WiFi/GSM.
2) By capturing user encounters, we design our algorithms to determine the current floor level of a user, without any prior knowledge of the building (i.e., floor height). We show theoretically that our algorithms are both feasible and scalable.
3) We conduct comprehensive simulations to analyze FTrack. We also conduct a field study in a 10-floor building, and the results show that FTrack achieves an accuracy of over 90% after two hours in our experiment.

The rest of this paper is organized as follows. Section II presents an overview of FTrack, followed by the detailed design in Section III. Section IV presents our theoretical analysis, and Section V describes our evaluation. Section VI discusses the related work, and finally, Section VII concludes the paper.

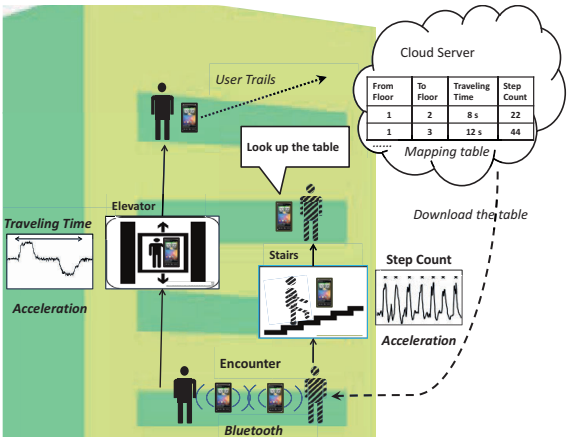


Fig. 1. Overview of FTrack

II. System Overview

We give an overview of our system in this section, as shown in Fig. 1. The system operates in two phases. In the first phase, when users travel in the building, mobile clients collect and transfer their trails to the cloud server. User trails contains information about user encounters, going up/down, the traveling time of the elevator, and the step counts when walking on the stairs. The server runs our algorithms to compute the mapping from the traveling time or the step counts to the number of floors. With enough trails collected, we can obtain the mappings for any two floors. In the second phase, mobile clients can measure the traveling time of the elevator or the step counts and the direction (i.e., up/down), and pinpoint the current floor by looking up the table.

FTrack recognizes user activities of taking elevators or walking on stairs using accelerometer. Based on these activities detected, we define two states of the mobile user—a moving (M) state when the user moves up/down between floors, and a static (S) state when a user stays on a floor, including the situation which the elevator stops temporarily on a floor. The states of a user may change over time. User trail is defined as a sequence of state changes which is $T_i = \langle S_1, M_1, S_2, M_2, S_3, \ldots S_n \rangle$.

Trails from different people are collected and transferred to the cloud server. We design an algorithm to first group people from the same floor where they encounter each other, and then sort all the groups by examining the sequence of state changes in the user trails. Finally, each group can be mapped into a unique floor level, and a series of tuples, i.e., $\langle FromFloor, ToFloor, Time, Steps \rangle$ is stored in the mapping table, as shown in Fig. 1 (i.e., the table in the cloud server). The algorithm runs incrementally with new user trails until the maximum number of tuples is found.

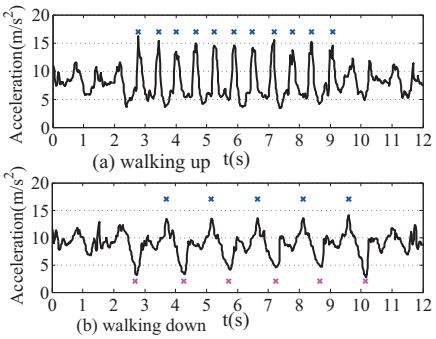


Fig. 2. Acceleration readings when walking on the stairs

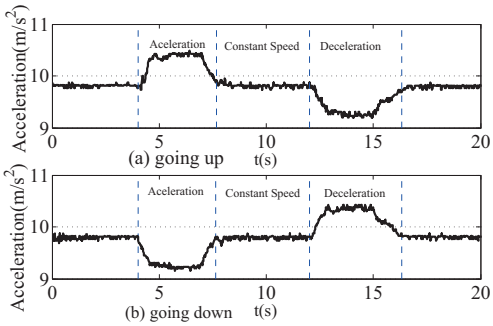


Fig. 3. Acceleration readings when taking the elevator

To access FTrack, mobile users first download the mapping table from the server. When they travel in the building, the traveling time of the elevator can be measured and the step counts can be computed using the mobile phone's accelerometer. By looking up the mapping table, FTrack can track users' floor levels provided that their initial floor levels are known in advance or their current floor levels have been discovered. It worth knowing that while the first phase is done at the cloud server, and localization is done locally at mobile clients. In this way, FTrack provides better user privacy protection as compared to the existing approaches [10], [11] which rely on the server to compute their locations.

III. System Design

We now describe the system design. The first phase of FTrack consists of the following stages—state recognition, user encounter detection, user trails collection, grouping and merging, mapping table generation.

A. User State Recognition

To detect user states (i.e., S or M) and their moving direction (i.e., up or down) at any time, we use the mobile phone's accelerometer. Acceleration readings show clear signatures when walking on the stairs or taking the elevator. Fig. 2(a) shows the acceleration readings when a person walks up one floor, and Fig. 2(b) shows the acceleration readings when the person walks down one floor. As shown in Fig. 2(a), the rhythms between walking on the same floor and walking up the stairs show very different, each spike in the positive direction roughly corresponds to a step which can be used to count the number of steps. Similar observations can be found when walking down the stairs. In this case, as shown in Fig. 2(b), spikes appear in both the positive and negative directions, and each spike corresponds to a step.

When people take elevators up or down, each travel process consists of three stages—acceleration, moving at a constant speed, and deceleration. Fig. 3 shows the acceleration data when a person takes the elevator up and down two floors, respectively. Both figures show the above three stages clearly. The acceleration readings at the second stage maintain about 9.8 due to the gravity. One important observation is that the traveling time of going up and down the same number of floors is the same.

We apply a simple machine learning approach to recognize the activities of walking on the same floor, walking up/down the stairs, and taking the elevator up/down. Recognition can be easily done in real time—collecting acceleration data, extracting features (i.e., mean, variance, energy and correlation), and applying the Naïve Bayes classifier for recognition. This approach is similar to [14], which have showed a high recognition accuracy.

B. User Encounter Detection

In a multi-floor building, people often encounter each other in the elevators, or on any floor level. FTrack detects user encounters for any user who is in the S state, i.e., when people are staying on any floor level. To detect encounters, we take a simple approach by measuring the Received Signal Strength Indicator (RSSI) of the Bluetooth radio on the mobile phone. When the RSSI level grows higher than a threshold, we can conclude that the two users are in close proximity (i.e., they encounter each other). The approach guarantees a high True Positive (TP) rate (i.e., the encounters detected are true encounters). Although there may be some missing detections due to signal variation in indoor environments, it does not affect the operation of FTrack. Other approach [9], detecting encounters by playing and listening to a specific tone at inaudible frequencies, can be used at the price of complexity and computation cost.

C. User Trails

FTrack records user trail which is a sequence of the states of S and M, containing information about user encounters, going up/down, the traveling time of the elevator, and the step counts when walking the stairs. User trails typically begin and stop with the S states (since a user must be present on a floor when entering a building).

The structure of the S state is defined as $\langle UserID, StartTime, EndTime, Encounters \rangle$, where $UserID$ is the ID of the user, $StartTime$ and $EndTime$ is the start and end time of the S state, $Encounters$ is a structure which records user encounters if any. $Encounters$ is

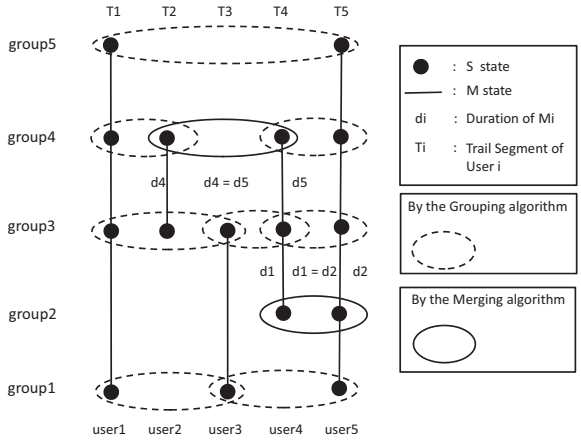


Fig. 4. Grouping and Merging

defined as a set of $\langle ID, Time\rangle$, where ID is the encountered user's ID and $Time$ is the time when the encounter occurs.

The structure of the M state is defined as $\langle UserID, Duration, ActionType, Direction\rangle$, where $UserID$ is the ID of the user, $Duration$ is the duration when the user is in the M state (i.e., the traveling time of the elevator when taking the elevator, the step counts when walking on the stairs), $ActionType$ is the action type which can be E (elevator) or S (stairs), $Direction$ is the direction of $ActionType$ which can be Up (going up) or $Down$ (going down).

The user trails are collected by mobile clients, and transferred to the server periodically. On receiving the trails, the server first convert them into trail segments according to the field values in the $(ActionType, Direction)$ pair. For example, given a user's trail, $T_i = \{S_{i1}, M_{i1}, S_{i2}, M_{i2}, S_{i3}, M_{i3}, \ldots, S_{i6}, M_{i6}, S_{i7}\}$ where $\{M_{i1}, M_{i2}, M_{i3}\}$ has a value of (E, Up) in the $(ActionType, Direction)$ pair, $\{M_{i,4}\}$ has a value of (S, Up), and $\{M_{i5}, M_{i6}\}$ has a value of $(E, Down)$, respectively. Hence, we divide the trail into three segments, $TS_{i1} = \{S_{i1}, M_{i1}, S_{i2}, M_{i2}, S_{i3}, M_{i3}, S_{i4}\}$, $TS_{i2} = \{S_{i4}, M_{i4}, S_{i5}\}$, and $TS_{i3} = \{S_{i5}, M_{i5}, S_{i6}, M_{i6}, S_{i7}\}$. We then convert all the going-down (i.e., $(E, Down)$ and $(S, Down)$) segments to the going-up (i.e., (E, Up) and (S, Up)) segments. For example, TS_{i3} can be converted to $TS_{i3} = \{S_{i7}, M_{i6}, S_{i6}, M_{i5}, S_{i5}\}$ with all the $Direction$ fields changed to Up. Hence, each trail segment contains a connected subsequence of the M states which have the field values of (E, Up) or (S, Up) in the $(ActionType, Direction)$ pair. These trail segments will be used as inputs for the algorithms described in the next section.

D. Grouping and Merging

The historical information contained in user trails is used to infer the number of the floors. We illustrate this process using an example, as shown in Fig. 4. Assume that we have five users in a 5-floor building, at the starting point, users 1, 3 and 5 are on the ground floor, user 2 is on the 3rd level, and user 4 is on the 2nd level. User 1 takes the elevator from the ground floor to the 5th level, while the elevator stops on the 3rd and 4th levels, respectively. User 1 encounters user 3 on the ground floor, users 2 and 3 in the elevator on the 3rd level, user 2 in the elevator on the 4th level, and finally user 5 in the elevator on the 5th level. If any two users who are in the S state encounter each other, it is obvious that they are on the same floor. Based on this rule, we generate many groups, indicated by dotted lines in Fig. 4. By knowing the sequence of member joining and leaving a group, we can infer whether a group is at a higher level than another. We can then sort all the groups from the lowest to highest. Finally, each group can be mapped to a corresponding floor. This grouping process is done by the grouping algorithm, as illustrated in Algorithm 1. The algorithm runs incrementally with new trail segments until the maximum number of floors is found. This algorithm has a complexity of $O(n^2 \ln n)$, where n is the total number of trail segments.

Finding the total number of floors by Algorithm 1 depends on the number of trail segments. The more segments we collect, the faster the result we obtain. In a multi-floor building, if there are users on each level and at least one group on each level (we name the trails collected which meet this requirement the *complete trails*), we can ensure the *completeness* and *correctness* of the result with high probability (we will show our analysis in Section IV-C). However, collecting the *complete trails* takes time which may be longer in a building with less crowds. To accelerate the grouping process, we propose the merging algorithm, which is capable of inferring additional knowledge based on the information contained in the trails. The algorithm is designed according to the following rules.

***Rule** 1:* For people stay on the same floor initially, after they travel towards the same direction (i.e., up or down, maybe at different time) for the same period of time (i.e., the same traveling time of the elevator, or the same step counts on the stairs), they must end up on the same floor.

To formulate the rule, we define the following propositions.

A_1 : $\exists\{S_m, M_m, S_{m+1}\} \subseteq TS_i$
A_2 : $\exists\{S_n, M_n, S_{n+1}\} \subseteq TS_j$
A_3 : $M_m.duration = M_n.duration$
A_4 : $\{S_m, S_n\} \subseteq G$
A_5 : $\{S_{m+1}, S_{n+1}\} \subseteq G'$

where TS_i is a trail segment of user i, S and M represent the S and M states, respectively, G and G' are groups. Rule 1 is then formulated as follows.

R_1 : $A_1 \wedge A_2 \wedge A_3 \wedge A_4 \rightarrow A_5$

For example, users A and B first encounter each other on the ground floor (user A is in state S_m and user B is in state S_n, S_m and S_n are in the same group based on the grouping algorithm). Later, each of them takes the elevator up at different time. If they travel at the same period of time, and

end up in state S_{m+1} for user A and S_{n+1} for user B, we can infer that S_{m+1} and S_{n+1} are in the same group although they do not encounter on the new floor.

Following the same principle but performing a backward derivation, we have the second rule.

***Rule** 2:* Suppose people travel the same period of time and in the same direction (i.e., up or down). If they end up on a floor and encounter each other, they must come from the same floor.

Rule 2 is formulated as follows.

$R_2: \quad A_1 \wedge A_2 \wedge A_3 \wedge A_5 \rightarrow A_4$

By applying both Rules 1 and 2, the groups generated by the grouping algorithm can be merged into larger groups. The merging process is described in our merging algorithm, as shown in Algorithm 2. This algorithm has a complexity of $O(n^3 \ln n)$, where n is the number of trail segments. With larger groups, we can find the *complete trails* more quickly.

After obtaining a number of groups, we sort all the groups by examining the state changes from one group to another. As a result, we obtain one or more directed graphs, in which each vertex represents a group and each directed edge represents the floor level information, pointing from a lower level to a higher level. Since a trail segment has a value of either (E, Up) or (S, Up) in the $(ActionType, Direction)$ pair, the direct graph is basically a directed acyclic graph (DAG). Finally, we select the longest path in the DAG, and the vertices in this path have a one-to-one correspondence to the floor levels. We can then pinpoint the floor level of any user with the S state in this path.

FTrack works provided that the initial floor level of a user is known in advance. A user can choose to supply the initial floor level when entering a building. Otherwise, we can determine her/his current floor level when she/he encounters someone whose floor level is known by FTrack.

E. Mapping Table Generation

We are now ready to generate a table which maps the traveling time to the number of floors a user travels for any two floor levels. For each floor pair (i, j) in the final path we obtain, we first find the corresponding groups G_i and G_j for floor i and j, respectively. We then search for a segment, $\{S_i, M, S_j\}$, from the entire trail segments, where S_i is in G_i and S_j is in G_j, and find the value in the $Duration$ field of M. Finally, we obtain a tuple $\langle FromFloor, ToFloor, Time, Steps\rangle$ for floor pair (i, j), where $Time$ is the traveling time of the elevator, and $Steps$ is the step counts of walking on the stairs. This process runs incrementally with new trail segments until all the floor pairs are found. For a building with f floors, the total number of floor pair (i, j) is $f(f-1)/2$. The floor pair is symmetric, i.e., (i, j) and (j, i) is treated the same, because going up/down for a fix number of floor takes the same traveling time (for the elevator) and the same step count (for the stairs). All the tuples are stored in the mapping table. To utilize FTrack, mobile clients download the mapping table from the cloud server. When FTrack detects a user in the M state, the floor level (i.e., the $ToFloor$ field of the tuple) can be easily obtained by looking up the table by the fields of $FromFloor$ and $Duration$ of the M state.

Algorithm 1 Grouping algorithm

```
for all Ti ∈ TrailSet do
    currentSSet.add(Ti.firstS())
end for
noTimeOverlap = true
while currentSSet.isNotEmpty() do
    oldestS = get_oldest_S(currentSSet)
    for all S_i ∈ currentSSet do
        if time_overlap(oldestS, S_i) then
            noTimeOverlap = false
            if encounter_in_overlap(oldestS, S_i) then
                new_group(oldestS, S_i)
            end if
        end if
    end for
    if noTimeOverlap then
        currentSSet.remove(oldestS)
        if oldestS.hasNextS() then
            currentSSet.add(oldestS.nextS())
        end if
    end if
end while
```

IV. Theoretical Analysis

We provide the theoretical analysis of FTrack in this section.

A. Modeling of Groups

We model the S states using a random graph $\Gamma_{n,N}$ with n labeled vertices and N edges. A group can be viewed as a set of connected vertices (i.e., connected component [15]) in $\Gamma_{n,N}$, where a vertex represents a S state in the group (n is the total number of S states), and an edge may exist between two S states according to our grouping and merging algorithms.

B. Finding p'

We denote p as the probability that there exists an edge between any two vertices in $\Gamma_{n,N}$ which is created by the grouping algorithm only (i.e., user encounters), and we assume an equal p for people encounter each other on any floor. We denote p' as the probability that there exists an edge between any two vertices which is created by both the grouping and merging algorithms (i.e., user encounters, and merging rules).

We now compute the probability of p'_k for any floor, i.e., floor k. p'_k consists of p (i.e., the probability of people encounter on floor k) and $p'_j (j \neq k)$ from any other floor j. For any pair of the S states on floor k, the probability that their next S states are on floor m $(k < m)$ is $1/(f-k)^2$. We know the probability of merging any two S states on floor m is p'_m. Therefore, the probability of merging any pair of S states on floor k by the S states on all the upper floors is $(p'_{k+1} + p'_{k+2} \ldots + p'_f)/(f-k)^2$, by the S states on all the

Algorithm 2 Merging algorithm

while $true$ **do**
  $group = get\_not\_enlarged\_group(all\_groups)$
  if $group.isEmpty()$ **then**
    $break$
  end if
  $group.setEnlarged(true)$
  $enlarge\_group(group)$
end while

$enlarge\_group(group)$
$clustersOfS = same\_next\_M\_duration(group)$
for all $cluster \in clustersOfS$ **do**
  $next = get\_next\_S\_set(cluster)$
  if $next.size > 1 \bigcap \neg in\_same\_group(next)$ **then**
    $newGroup = merge\_group(next)$
    $enlarge\_group(newGroup)$
  end if
end for
$clustersOfS = same\_before\_M\_duration(group)$
for all $cluster \in clustersOfS$ **do**
  $before = get\_before\_S\_set(cluster)$
  if $before.size > 1 \bigcap \neg in\_same\_group(before)$ **then**
    $newGroup = merge\_group(before)$
    $enlarge\_group(newGroup)$
  end if
end for

lower floor levels is $(p'_1 + p'_2 \ldots + p'_{k-1})/(k-1)^2$. We then obtain p' on floor k as follows.

$$p'_k = \begin{cases} p + \dfrac{(p'_{k+1} + p'_{k+2} \ldots + p'_f)}{(f-k)^2} & k = 1 \\ p + \dfrac{(p'_1 + p'_2 \ldots + p'_{k-1})}{(k-1)^2} + \dfrac{(p'_{k+1} + p'_{k+2} \ldots + p'_f)}{(f-k)^2} & k \in (1, f) \\ p + \dfrac{(p'_1 + p'_2 \ldots + p'_{k-1})}{(k-1)^2} & k = f \end{cases} \quad (1)$$

where f is the total number of floors. We plot the ratio of p' vs p in Fig. 5 for different floor levels (i.e., 5 levels to 40 levels), where each p'_k is set to p initially. The figure shows p' converges after iterations, and it also shows a bigger ratio of p' vs p for lower buildings.

C. Proofs of Completeness and Correctness

It is shown in [15] that the random graph mentioned in section IV-A is almost surely be connected if

$$p' > \frac{1}{n}(1+\epsilon)\ln n \quad (2)$$

With a large n, the value of $(1+\epsilon)\ln n/n$ is small, therefore a smaller p' can meets the above equation. It implies that with enough S states, after grouping and merging, there exists a large group on each floor which may contain all the S states on this floor. We denote the largest group on floor i as G_i.

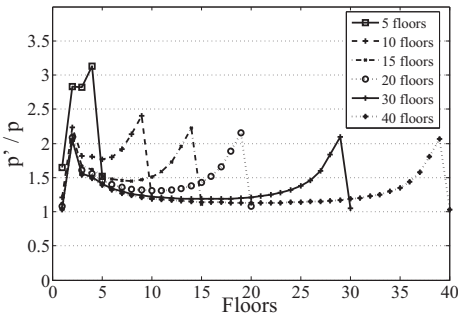


Fig. 5. p' vs p

***Theorem** 4.1:* For any G_i and G_{i+1}, if our algorithm can infer which is on a higher floor, we can then surely find a path which each group in the path has a one-to-one correspondence to the floor level.

Proof: Since our algorithm knows which (G_i or G_{i+1}) is higher, after sorting all the groups, there must exists a path from G_1 to G_f, which has a one-to-one correspondence to the floor level. ■

With Theorem 4.1, we define the *correctness* of our algorithm as when the number of floors are correctly found.

***Theorem** 4.2:* For any G_i and G_j, if our algorithm can infer which is on a higher floor, we can then surely find all the floor pairs (i, j), where $i < j$.

Proof: Since our algorithm knows which (G_i or G_j) is higher, by Theorem 4.1 we can find the floor levels of each group. Furthermore, there exists a segment $\{S_i, M_i, S_j\}$ with S_i in G_i and S_j in G_j. Based on M_i, we can find the floor pair (i, j). Finally, all the floor pairs can be found. ■

With Theorem 4.2, we define the *completeness* of our algorithm as when all the floor pairs are correctly found.

We now analysis the *correctness* and *completeness* of our algorithm. We define $P_{i,j}$ as the probability our algorithm can find which group (G_i or G_j) is higher. We denote n'_i and n_i as the number of the S states in G_i and on floor i, respectively.

We show how to find $P_{i,j}(i < j)$. For any S state, S_k, on floor k, its preceding state in the trail segment is denoted as $S_{k,pre}$. If there exists a S' in G_j, and its preceding state is in G_i, the algorithm can infer that G_j is on a higher floor than G_i. Therefore, $P_{i,j}$ is equal to the probability that at least one S' exists. For any S_j in G_j, we first get the probability that $S_{j,pre}$ is on floor i, which is $1/(j-1)$. Then, if $S_{j,pre}$ is on floor i, we can get the probability that it is not in G_i, $1 - n'_i/n_i$. Therefore, the probability that at least one S' exists is

$$P_{i,j}(i < j) = 1 - (1 - \frac{n'_i}{n_i})\frac{n'_j}{j-1} \quad (3)$$

Finally, our algorithm is correct with a probability of

$$P = \prod_{2}^{f} P_{i-1,i} \tag{4}$$

and our algorithm is complete with a probability of

$$P_{all} = \prod_{1 \leqslant i < j}^{f} P_{i,j} \tag{5}$$

From Equation 2, we have $n'/n \approx 1$, and $P_{i,j}$ is high with a large n. When $n \to \infty$, we then have $P = 1$ and $P_{all} = 1$. With enough S states collected, in a high probability (≈ 1), we are able to find the correct number of floor levels and all the floor pairs.

V. Evaluation

We evaluate FTrack using both simulation and field studies. In this section, we first present our simulation results, and then report the experiments from the field study.

A. Simulation Methodology

We design a simulator to evaluate the efficiency and scalability of FTrack. Since the process of walking on stairs is rather simple, the simulator only models the process of taking the elevator in a multi-floor building. The parameters of the simulator are listed as follows.

- f: maximum number of floors.
- p: probability of user encounters on the same floor.
- d: number of trail segments used by our algorithms.

The simulation process is divided into cycles, and each cycle simulates the process that the elevator goes up from, or down to the ground floor, with people entering and exiting the elevator from or to any levels. We model the process of people entering the elevator from the ground floor as the Poisson distribution. The expected number of the Poisson distribution is set to half of the maximum load of a typical elevator, (i.e., 8 persons). People on the ground floor may go up to any floor with a probability of $1/(f-1)$. From any other floor, a person may enter the elevator with a probability of p_e, and go to other $(f-2)$ floors with an equal probability $1/(f-2+k)$, where k is a constant, except to the ground floor which is k times bigger (i.e., $k/(f-2+k)$). When people enter or exit the elevator from a floor, the number of people on that floor gets updated; by multiplying p_e, we obtain the number of persons wants to enter the elevator. Each cycle starts from the ground floor, we first compute the number of people entering the elevator and which floor they want to go, the elevator goes up from the ground floor, and stops when people exiting or entering, and comes back the ground floor when the cycle ends. We assume 5 minutes for each cycle on average. User trails are collected continuously over the time. FTrack computes the mapping table after each cycle using the user trails collected from all the previous cycles. The table is then used to find the floor levels for people involved in the next cycle. The process runs recursively, and we report our results in the next section.

B. Simulation Results

Figures 6(a) and 6(d) shows the CDF when the correct number of floors is found by the grouping and merging algorithms, and the grouping algorithm, respectively. While the results show that it requires less time with fewer floors for both cases, with the same number of floors, the grouping and merging algorithms compute the correct results much faster. One observation is that, for a tall building (i.e., 20-floor in the figures), the CDF for the grouping and merging algorithms grows steeply, demonstrating the effect of the merging algorithm. Fig. 6(a) also shows that our algorithms find the correct number of floors in less than four hours in a 10-floor building, and less than ten hours in a 20-floor building. We believe that, in reality, we will obtain the result much faster for a higher probability of user encounters (note that we assume a small probability i.e., 0.01 in our simulation setting).

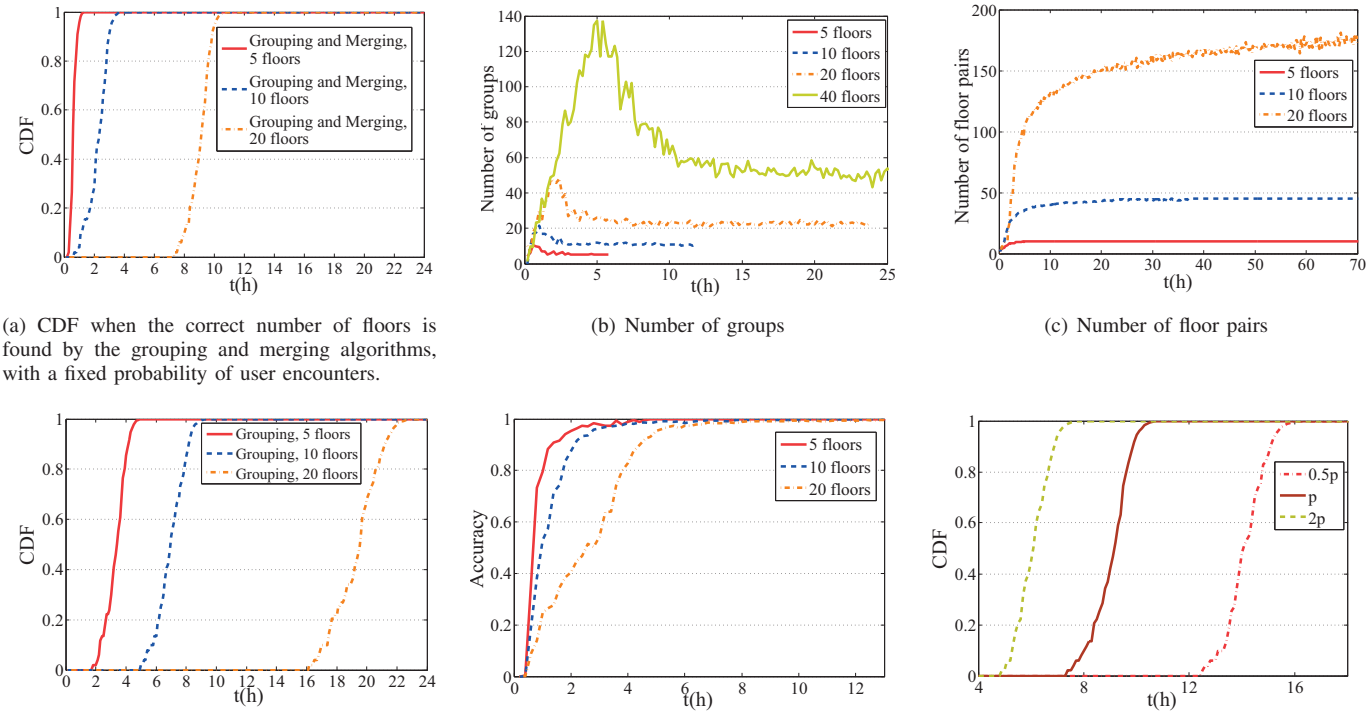
Fig. 6(b) shows the number of groups computed over time for different building cases. We observe a similar trend for all the cases. In the beginning, the number of groups grows since more people travel between different levels (i.e., more encounters). When more groups are merged, the number of groups decreases and converges to a value which is equal to the maximum number of floors for each case.

Fig. 6(c) shows the time spent by our algorithms on creating the complete mapping table. The maximum number of tuples (i.e., floor pairs) for a building with f floors can be calculated by $f(f-1)/2$. For example, we have 10, 45 and 180 tuples for a 5-, 10-, and 20-floor buildings, respectively. With enough time, we can obtain all the tuples. Higher buildings require much longer time because 1) more tuples to be computed, and 2) some tuples may be rare (e.g., the elevator rarely goes up from the ground floor to the 20th floor without a stop in the middle). Fig. 6(c) also shows that our algorithms are capable of computing most of the tuples quickly.

Fig. 6(e) shows the accuracy of FTrack. The accuracy increases with more tuples computed, for example, FTrack achieves an accuracy of over 90% after 2.2 hours in our simulation. When the complete set of tuples is obtained, the accuracy achieves 100%. While the merging algorithm is effective, its efficiency is significantly influenced by user encounters. We run the merging algorithm with different encounter probabilities and the result is shown in Fig. 6(f), where t is the time when the correct number of floors is found in a 20-floor building. The figure shows that the result is obtained faster with a higher probability of user encounters.

C. Field Study

Our simulation results show that the performance of FTrack is influenced by the probability of user encounters. We are interested to know how FTrack behaves under a real-world encounter probability. We conduct a field study which involves 10 mobile users for three hours in a 10-floor building with two elevators. We use different Android smartphones (e.g., HTC, Moto and Samsung). Each smartphone is equipped with an embedded 3-axis accelerometer. The field study is conducted as follows. We first load a program with a simple



(a) CDF when the correct number of floors is found by the grouping and merging algorithms, with a fixed probability of user encounters.

(b) Number of groups

(c) Number of floor pairs

(d) CDF when the correct number of floors is found by the grouping algorithm only, with a fixed probability of user encounters.

(e) Accuracy of FTrack

(f) CDF when the correct number of floors is found ($f = 20, p = 0.01$).

Fig. 6. Simulation Results

user interface to each smartphone. This program gives a random destination floor where the user is supposed to go, and the user can choose to get there by either taking the elevator or walking on the stairs as they wish. If the user encounters someone (e.g., on a floor level, or in the elevator), she/he should record the encounters by pressing a button on the screen for the ground-truth. Mobile client is implemented using the Android platform, and it performs functions such as logging acceleration readings, performing user state detection, collecting and transferring the user trail. All the users are told to put their phones in their front trouser pockets, unless when they record for the ground-truth. We assume normal walking on stairs for each user (i.e., one stair for each step). FTrack computes the mapping table periodically using the user trails collected, and the process runs recursively until the end of experiment. We report our results in the next section.

D. Field Study Results

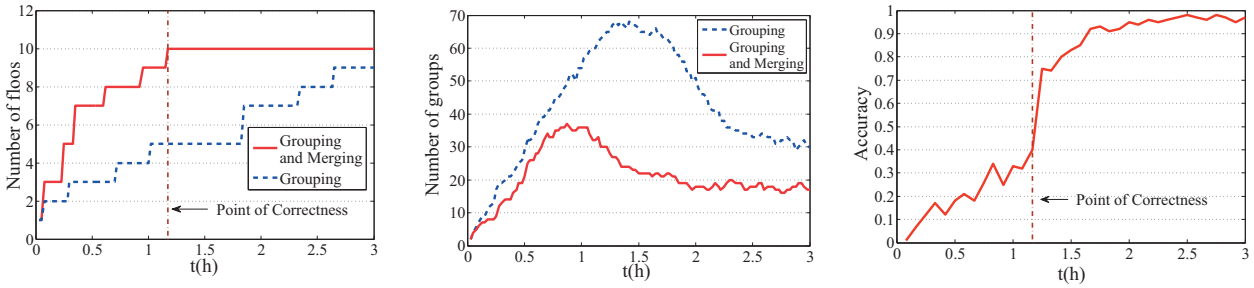
Fig. 7(a) shows that FTrack detects the correct number of floors after about 70 minutes when the *complete trails* are obtained, indicated by the point of *correctness*. The time required to detect the correct number of floors is less than that in our simulation (i.e., 3.5 hours) due to more encounters occur in our field study, especially in the elevator. The figure also shows that by the merging algorithm, FTrack obtains the *complete trails* more effectively than the grouping algorithm.

Fig. 7(b) shows the number of groups generated over time. While we observe a similar trend to that seen in our simulation, the number of groups in the field study takes longer time to converge as compared to that in our simulation. One explanation is that we observe that users in our field study do not stay on a floor for a long time, and they do not encounter each other more often, resulting in many individual, discrete groups which require a longer time to merge.

Fig. 7(c) shows the accuracy of FTrack. At the point of *correctness*, the accuracy increases steeply. The other observation is that the accuracy approaches 90% in two hours, and reaches 97% in three hours.

VI. Related Work

Existing techniques for localization rely on deployed radios (e.g., WiFi access points, GSM base stations, etc) and make different assumptions about infrastructure and calibration. Radar [5] operates on WiFi fingerprints, and is capable of achieving high accuracy in indoor deployments. However, Radar needs to calibrate WiFi signal strengths at many physical locations in a building. The calibration process is time-consuming and may not scale over larger areas. PlaceLab [4] uses WiFi and GSM signals. A radio map is created by war-driving car-accessible roads and mapping the WiFi access points or GSM based stations to GPS coordinates. The radio map is distributed to mobile devices which localize themselves by comparing overheard access points or GSM based stations with those recorded in the map. Different from these systems,



(a) Number of floors detected (b) Number of groups (c) Accuracy of FTrack

Fig. 7. Field Study Results

FTrack does not require any pre-installed infrastructure.

Sensor-assisted localization methods have been proposed with the popularity of smartphones. These systems typically make use of accelerometer and electronic compass on the smartphone. The basic idea is to use acceleration data to measure the walking speed, and electronic compass to compute the orientation of a user. To overcome inherent noise in sensor readings, Constandache et al. [9], [10] propose a better approach by identifying an acceleration signatures in human walking patterns (i.e., the nature up and down bounce), and then multiplying step count with the user's step size to obtain the location. However, Escort [9] leverages fixed beacons for calibration, and CompAcc [10] makes use of possible walking paths extracted from Google Maps [13]. While FTrack makes use of accelerometer on the mobile phone, we do not require any calibration and any prior knowledge of the building, and yet we achieve high accuracy.

SkyLoc [16] uses a GSM fingerprinting-based approach to address the floor localization problem using mobile phones. It identifies the floor correctly in up to 73% of the cases and is within 2 floors in 95% of the cases. It leverages on GSM signals which may vary significantly in indoor environments. The training process in SkyLoc is time-consuming. In addition, the solution may not be scalable since training is required for each building.

VII. Conclusion and Future Work

In conclusion, this paper presents a novel, scalable floor localization scheme. FTrack uses the mobile phone's accelerometer only, and it requires neither any infrastructure nor any prior knowledge of the building. By analyzing user trails, FTrack is able to map the traveling time to the number of floors, which can be then used by mobile users to pinpoint their current floor levels. Through our simulation and field studies, we demonstrate the effectiveness of FTrack and its feasibility. FTrack can complement any existing indoor localization system to achieve better performance.

For our future work, we intend to further develop our algorithms to improve the robustness, especially when there are false detections of state recognition or user encounters. We will also study a complex graph model for user encounters, and design advance algorithms to infer more knowledge from user trails. Finally, we plan to offer FTrack as an free application to the Android market or the Apple store, and further test FTrack in real-life environments.

Acknowledgment

This work was supported by the Fabrikant Mads Clausens Fund of Denmark, the National 973 project of China under Grant 2009CB320702, the NSFC under Grants 61021062, 60736015 and 61073031.

References

[1] S. Gaonkar, J. Li, R. Choudhury, L. Cox, and A. Schmidt, "Micro-blog: sharing and querying content through mobile phones and social participation," in *Proceeding of the 6th international conference on mobile systems, applications, and services*. ACM, 2008.

[2] G. Bruner II and A. Kumar, "Attitude toward location-based advertising," *Journal of Interactive Advertising*, vol. 7, no. 2, pp. 3–15, 2007.

[3] P. Mohan, V. Padmanabhan, and R. Ramjee, "Nericell: rich monitoring of road and traffic conditions using mobile smartphones," in *Proceedings of the 6th ACM conference on embedded network sensor systems*, 2008.

[4] A. LaMarca, Y. Chawathe, S. Consolvo, J. Hightower, I. Smith, J. Scott, T. Sohn, J. Howard, J. Hughes, F. Potter *et al.*, "Place lab: device positioning using radio beacons in the wild," *Pervasive Computing*, pp. 301–306, 2005.

[5] P. Bahl and V. Padmanabhan, "Radar: an in-building rf-based user location and tracking system," in *INFOCOM 2000. Israel.*

[6] M. B. Kjærgaard, "Indoor location fingerprinting with heterogeneous clients," *Pervasive and Mobile Computing*, pp. 31–43, 2011.

[7] G. V. Zàruba, M. Huber, F. A. Kamangar, and I. Chlamtac, "Indoor location tracking using rssi readings from a single wi-fi access point," *Wirel. Netw.*, vol. 13, pp. 221–235, April 2007.

[8] V. Otsason, A. Varshavsky, A. LaMarca, and E. De Lara, "Accurate gsm indoor localization," *UbiComp 2005*, pp. 141–158, 2005.

[9] I. Constandache, X. Bao, M. Azizyan, and R. Choudhury, "Did you see bob?: Human localization using mobile phones," in *Proceedings of the 16th annual international conference on mobile computing and networking*. ACM, 2010.

[10] I. Constandache, R. Choudhury, and I. Rhee, "Towards mobile phone localization without war-driving," in *INFOCOM2010*. IEEE.

[11] A. Ofstad, E. Nicholas, R. Szcodronski, and R. Choudhury, "Aampl: Accelerometer augmented mobile phone localization," in *Proceedings of the first ACM international workshop on mobile entity localization and tracking in GPS-less environments*. ACM, 2008, pp. 13–18.

[12] C. Jekeli, *Inertial navigation systems with geodetic applications*. Walter De Gruyter Inc, 2000.

[13] Google maps. [Online]. Available: http://maps.google.com/

[14] N. Ravi, N. Dandekar, P. Mysore, and M. Littman, "Activity recognition from accelerometer data," in *Proceedings of the National Conference on Artificial Intelligence*. AAAI Press, 2005.

[15] P. Erdos and A. Rényi, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, pp. 17–61, 1960.

[16] A. Varshavsky, A. LaMarca, J. Hightower, and E. de Lara, "The skyloc floor localization system," in PerCom 2007.