Incrementally-deployable Indoor Navigation with Automatic Trace Generation

Yuanchao Shu[1,⋆], Zhuqi Li[2,⋆], Börje Karlsson[1], Yiyong Lin[1], Thomas Moscibroda[1], Kang Shin[3]
[1]Microsoft Research, [2]Princeton University, and [3]University of Michigan
⋆: Co-primary authors

Abstract—Despite years of research attention, localization-based indoor navigation has not found wide-spread practical use, largely due to the high burden on deployment and bootstrapping. Lightweight peer-to-peer navigation systems that use a leader-follower model have recently been proposed to alleviate these burdens. However, typical peer-to-peer navigation suffers from poor scalability and flexibility as navigation is only possible over pre-collected leader paths. In this paper, we present FollowUs, an easily-deployable (bootstrap-free) and scalable indoor navigation system. In addition to robust navigation through real-time trace-following, FollowUs integrates cloud services to process and combine traces at large scale. Optionally, it can also leverage floor plans to further enhance navigation efficiency. We design and implement FollowUs, including mobile app and cloud services. Experimental results from a company-internal beta release show that 91% of FollowUs' spatial errors on reaching destinations to be 3m or less, and 95% of navigation instructions are shown to users within a 4-step error margin during navigation.

I. Introduction

There has been substantial investment in indoor navigation solutions both in industry and academia. The traditional way of enabling indoor navigation is through localization and maps, and there have been numerous attempts to build and deploy such indoor localization services. One reason why such services are nevertheless not widely available in practice is that they rely on specialized infrastructure (e.g., beacons, LEDs) or incur high bootstrap costs [1–6]. For example, fingerprinting-based techniques e.g., Wi-Fi) require an onerous sample collection and map calibration process. Lack of accurate indoor maps often poses additional challenges to path planning and exacerbates navigation performance issues. For these reasons, navigation services based on localization and maps have proven difficult to deploy at scale.

To circumvent these difficulties, several peer-to-peer (P2P) navigation systems have recently been proposed [7–11]. These systems fundamentally differ from the above in that they do not require localization. Instead, they follow a leader-follower paradigm. P2P navigation compares favorably to its localization-based counterparts for two main reasons. First, it avoids the substantial setup effort and precision requirements of localization. Instead of building a full-fledged localization system, it offers a simple plug-and-play mechanism — a "leader" captures a path trace (i.e., a series of sensor data), and then any "follower" can navigate on that same path. Second, it does not require indoor maps for navigation. Irrespective of incomplete map information, followers are able to go to any point of interest (PoI) as long as a prior user has visited it before.

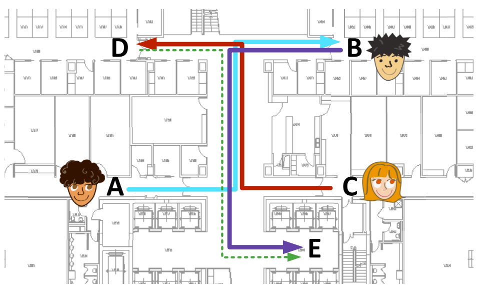


Figure 1. A peer-to-peer navigation example.

However, the flip-side of such incrementally deployable P2P navigation systems is clear: they come at the cost of low scalability, since they only enable navigation between the start and destination points of previously recorded paths. Consider a simple example in Figure1. Bob recorded a trace from A to B (blue), Alice recorded a trace from C to D (red), and Charlie recorded a trace from B to E (purple). In current peer-to-peer navigation systems, if another user Dan wants to navigate from A to E, he must first follow the blue trace to B, and then follow the purple trace to E. Moreover, there is no way for a subsequent user to move from D to E using typical P2P navigation techniques. In a nutshell, for an indoor environment with n PoIs, one needs to collect at least $n(n-1)/2$ traces to cover all navigation cases, thus incurring a heavy $O(n^2)$ bootstrap cost.

Existing peer-to-peer navigation systems can also suffer from low flexibility. They do not take additional data sources as input to improve performance and navigation results. For instance, while indoor maps (a.k.a. floor plan) may be available in some cases, no systematic way to leverage such additional resources is known. Clearly, a flexible P2P navigation system must lower the entry barrier for users as much as possible, should provide substantial value even to early-adopters, and should incorporate existing additional data sources such as maps, if available.

In this paper, we propose FollowUs, a new end-to-end indoor navigation system that combines the advantages of both localization-based and peer-to-peer navigation with increased flexibility and scalability. FollowUs can scale both horizontally — reaching as many users as possible without requiring much effort of them — as well as vertically by efficiently computing new paths from traces shared by users. As in typical peer-to-peer navigation, users can walk along a path, share it with others and follow paths shared by others via a mobile client. The cloud-service components of FollowUs are responsible for storing traces, and notably also for parsing and combining ("stitching together") traces at scale. Based on the analysis of motion events and sensor signals, FollowUs is able to split, concatenate, and generate new traces from different contributors. Therefore, the navigation is not restricted on the pre-recorded routes. Taking Figure 1 as an example, FollowUs is able to identify the overlapping segment from the blue and red traces, and the ones from the blue and purple traces, thereby providing navigation directly from D to E by following red and purple segments.

This way, FollowUs incrementally provides large-scale indoor navigation without prior knowledge of floor plans, nor the need for any infrastructure. As the data available in the system organically grows, FollowUs can automatically generate more possible paths. Moreover, it is also able to leverage any available floor plan to speed up the process and build trace/segment graphs, which further enhance navigation efficiency. In summary, this paper makes the following three contributions:

- We propose a new peer-to-peer navigation service to address the scalability and flexibility problems of existing previous indoor navigation systems.
- We design efficient and accurate matching, indexing, and segment mapping algorithms to enable indoor graph construction and automatic trace generation.
- We implement FollowUs as an Android app and a set of cloud services. The system has been released as a company-internal beta, and has been in use by 150+ beta users over a period of 3 months. Experimental results show that 91% of FollowUs' spatial errors are found to be 3m or less, and 95% of navigation instructions are shown to users within a 4-step error from the ideal timing during navigation.

II. Overview

FollowUs allows users to both record and follow paths using its mobile client app. Users of the app play two roles:

Recording (as "leader"). During recording, the FollowUs client queries sensors and detects user motion events including steps, turns, and level changes i.e., staircases, elevators, or escalators). When the leader arrives at the destination, sensor measurements and detection results are packed to build a reference trace. Leader is able to label a trace with text descriptions (e.g., from East Entrance to Room 1357). However, these highly diversified labels are not used as location identifiers. Finally, leader sends the trace to the cloud to be shared publicly or with specific groups of users.

Navigation (as "follower"). Users can follow traces over any pre-recorded path. Traces can either be pushed to the client app Inbox or pulled via Search functionality (e.g., by trace ID or labels). When following a trace, real-time navigation instructions including turns and floor/level changes are displayed in the app based on online synchronization between the reference trace and current ambient sensing information.

It is noted that leader's input on trace information (e.g., starting point and destination) is required for navigation. FollowUs assumes a certain degree of leader's initiative to create reasonable quality traces to help subsequent followers (e.g., finding a conference room or a restaurant in shopping mall). We argue that in the context of P2P navigation, such requirements can be met in most cases. The current client also allows users to report a public trace due to either bad quality or inappropriate content. Nevertheless, the mechanisms to prevent abuse of FollowUs are out of the scope of the paper.

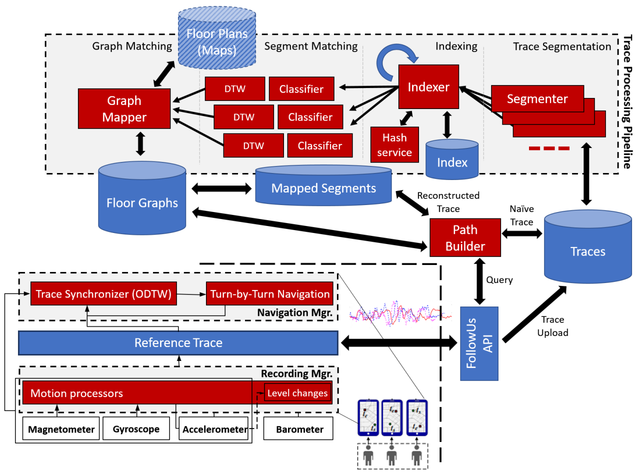


Figure 2. System diagram of FollowUs.

Figure 2 shows FollowUs' architecture and data pipeline. Data flows from client (as a leader) to cloud, and then from cloud to client (as a follower). The cloud component not only provides storage, communication interfaces, telemetry, and account management, but also features an intelligent trace processing pipeline that makes FollowUs flexible and scalable. In "naïve" navigation, where one can only follow the exact same paths from others, each path recording produces and uploads a trace. To allow "cross-trace" navigation, FollowUs employs a novel trace segmentation and indexing technique. Segments are then efficiently matched, indexed, and persisted to allow the construction of segment graphs. A segment graph is incrementally built from trace segments and can eventually provide users with an exponentially increasing number of paths. Additionally, floor plans can naturally be leveraged to improve both performance and efficiency of the graph generation process.

It is important to highlight that FollowUs is completely plug-and-play, meaning it imposes no infra-structure requirements on buildings, no boot-strapping effort such as building Wi-Fi fingerprint databases, and it requires minimal effort from users. This allows the system to scale out to reach as many potential users and to work in as many locations as possible. As such, FollowUs does not rely on collecting Wi-Fi- (or other wireless technology-) related data. FollowUs traces require only readings from smartphones' magnetometer, accelerometer, gyroscope, and barometer sensors. Nonetheless, other signals can easily be integrated into the system.

III. Segment

In this section, we first introduce the concept of a segment that serves as basis for the identification of intersections between traces and is key to scalability. Specifically, we look to answer two questions: (1) how does segmentation help FollowUs scale? and (2) what are the associated challenges and our solutions?

A. Trace segmentation

In FollowUs, users share traces for the purpose of navigation. In naïve navigation, users can merely follow a trace that was previously created by a leader. However, due to a huge number of paths connecting all PoIs in indoor environments, it is highly unlikely there to be a trace that exactly meets user requirements. A possible approach to address this problem is to divide all traces into segments, and concatenate these segments to generate new traces based on user needs. However, there are several challenges in doing so. Chief among them is how to design effective matching to identify shared segments between different traces. For example, in Figure 1, we can see that the key to build trace DE is to identify matching segments between trace AB and trace BE. Also, the matching method should be scalable to the number of pairs of segments that grows quadratically with the number of traces.

FollowUs leverages the previously-stored navigation events in each trace and divides a trace into segments based on the different user motion events (e.g., turns, staircases, or elevators). Although traces could be split at any arbitrary point, event-based segmentation captures the formation of indoor pathways and makes a good tradeoff between segment matching performance and efficiency. We represent a given trace U by a sequence of directed segments $S^U = \{s_i^U\}_{i=1}^{n}$ and a sequence of connectors $C^U = \{c_i^U\}_{i=1}^{n-1}$. Each segment also contains location-specific sensing data (e.g., magnetic field magnitude, RF signal strength) and motion detection information during the corresponding period. Considering different events, any connector has an event type and a corresponding attribute value. For example, for a turn event, the value can be measured by the turn angle.

B. Segment matching

Segment matching aims to identify pairs of trace segments, which are recorded along the same pathway. During matching, we exploit the similarity between the location-specific sensor data of two segments to tell whether they are a match. Although different types of location-specific data can be utilized in segment matching, FollowUs uses indoor geomagnetic magnitude to demonstrate the methodology (and reach its design goals from Figure II). Due to space limitation, we refer to [9, 12] for detailed analysis (e.g., robustness) and process on geomagnetic signals.

In FollowUs, different walking speeds and different smartphone sensor sampling rates during recording will make the recorded signals have widely different magnetic magnitude sequences. To cope with this problem, we take a two-step approach to identify whether two sequences match: we first use dynamic time warping algorithm (DTW) [9, 13] to synchronize two sequences, and then use a general model pre-trained by SVM to classify the synchronization result. To achieve a better accuracy, we carefully selected four features to train the SVM classifier – the length of the first segment, length of the second segment, DTW distance between the two segments, and area between the optimal DTW path and the linear mapping path in the warping cost matrix.

C. Segment indexing

The key to make FollowUs scalable is to identify matched segments in different traces and be able to "create" new traces. However, the overhead of directly comparing each pair of trace segments grows quadratically with the number of new traces, which is unacceptable for large-scale deployments. To efficiently select matching candidates, we designed a segment indexing mechanism, used in the matching, search, and comparison infrastructure.

In FollowUs, we design a hash structure for efficient segment indexing and query based on Distance-Based Hashing (DBH). This structure contains four basic operations: structure building, structure update, query, and re-hash.

1) Distance-based hashing: DBH is a family of hash functions that can map data under an arbitrary distance into real numbers. It is based on the FastMap technique [14]. The formal definition can be given as: given any distance space $\mathbb{D}$, choose two reference members $s_1, s_2 \in \mathbb{D}$. For any query point s in $\mathbb{D}$, the hash function is:

$$Hash_{s_1,s_2}(s) = \frac{d(s,s_1)^2 + d(s_1,s_2)^2 - d(s,s_2)^2}{2d(s_1,s_2)}.$$

If $\mathbb{D}$ is a Euclidean distance space, this hash function will project the query point onto the line determined by s_1 and s_2. If $\mathbb{D}$ is a non-Euclidean distance space, this hashing function can still approximately preserve distance relationship. After obtaining the hash value, we can binarize the value based on whether or not $Hash_{s_1,s_2}(s)$ lies in a certain range $[t_1, t_2]$.

$$Hash_{s_1,s_2}^{t_1,t_2}(s) = \begin{cases} 1 & t_1 \leq Hash_{s_1,s_2}(s) \leq t_2. \\ 0 & otherwise. \end{cases}$$

Optimally, the range $[t_1, t_2]$ should be chosen to make a random object in $\mathbb{D}$ have a 50% possibility to be mapped within the range and a 50% possibility to be mapped outside the range. The set of ranges can be defined as $\mathbb{T} = \{[t_1,t_2] | Pr_{\mathbb{D}}(Hash_{s_1,s_2}^{t_1,t_2}(s) = 1) = 0.5\}$. The set of optimal hash functions $OptHash_{s_1,s_2}$ can be defined as $OptHash_{s_1,s_2}(s) = \{Hash_{s_1,s_2}^{t_1,t_2}(s) | [t_1,t_2] \in \mathbb{T}\}$.

2) Hash structure building: Algorithm 1 demonstrates how FollowUs builds the structure from scratch. For every segment s_i in segment set S, FollowUs maintains a hash sequence h_i with the length of $\lceil \log_2 |S| \rceil$ bits. To generate $\lceil \log_2 |S| \rceil$ bits for every hash sequence, Algorithm 1 takes

$\lceil \log_2 |S| \rceil$ iterations. For each iteration, it first randomly chooses two reference segments s_{i1}, s_{i2}, and computes $Hash_{s_{i1},s_{i2}}(s_j)$ for every segment s_j under DTW distance. Then, it assigns a value to the hash bits according to whether $Hash_{s_{i1},s_{i2}}(s_j)$ is larger or smaller than the median. In total, $O(n \log n)$ comparisons between trace segments are conducted. Since the hash sequence has a length of $\lceil \log_2 |S| \rceil$, there are a total of $O(n)$ possible types of hash sequence. Due to the algorithm's balancing mechanism, the average number of segments that are hashed to a specific hash sequence is $O(1)$.

ALGORITHM 1: Index building.

```
Input  : The set of segments $S$
Output: The set of hash value $H$
for $i \leftarrow 1$ to $\lceil \log_2 |S| \rceil$ do
    random choose two reference segments $s_{i1}$, $s_{i2}$ from $S$;
    for $j \leftarrow 1$ to |S| do
        $Hash_{s_{i1},s_{i2}}(s_j) \leftarrow \frac{dtw(s_j,s_{11})^2 + dtw(s_{i1},s_{i2})^2 - dtw(s_j,s_{i2})^2}{2dtw(s_{i1},s_{i2})}$;
        $Queue_i$.append($Hash_{s_{i1},s_{i2}}(s_j)$);
    end
    $mid$ = mean($Queue_i$);
    for $j \leftarrow 1$ to |S| do
        if $Hash_{s_{i1},s_{i2}}(s_j) < mid$ then
            $h_i$.append(0);
        end
        else
            $h_i$.append(1);
        end
    end
end
```

For example, given a set of segment $S = \{s_1, s_2, s_3, s_4\}$, Algorithm 1 could choose s_1 and s_2 as the reference segments and compute the first hash bit for all the hash sequence. Let's say the result is $\{1, 0, 1, 0\}$. Then, it repeats the operation on different reference segments and gets the second hash bit, for example, $\{10, 00, 11, 01\}$.

3) Structure update: The hash structure is updated incrementally. Every time a new segment comes in, it will be compared with reference segments and get its own hash sequence. This step takes $O(\log n)$ comparisons. When the number of segments redoubles, the system chooses a new pair of reference segments and adds one more bit to every hash sequence. This step takes $O(n)$ comparisons. For the example shown above, if one more segment s_5 is added into the system, it should be compared with all the reference traces to get its hash sequence. If additional four segments $\{s_5, s_6, s_7, s_8\}$ are added, it will choose two more reference traces and extend all the hash sequence by one bit.

4) Query: The query operation is performed to find matching segments in the hash structure for an input segment. The new segment will first be compared with all reference segments to get its own hash sequence (in $O(\log n)$ comparisons). Then, it will be compared with all the segments whose hash sequence has at most one bit difference with itself to find a match case. Since there are a total of $O(\log n)$ hash sequences satisfying this requirement, and every sequence has an average of $O(1)$ corresponding segments, this step takes, on average, $O(\log n)$ comparisons. Therefore, a query operation also takes a total of $O(\log n)$ comparisons. For the example shown above, if a query to find a match segment for s'_1 comes in, FollowUs will compare s'_1 with all reference segments and get its hash sequence, say $\{10\}$ in this case. Then it finds s_1 and s'_1 matches to each other.

5) Re-hash: Although we have a balancing mechanism to map half of the bits to 0 and the other half to 1, the segments that come into the system later may break the previous mapping balance, making a disproportional distribution of hash values. To address this problem, we need a re-hash operation to re-balance. The re-hash operation is straightforward: for any hash digit, if the number of hash bits set to one is much more/less than that to zero, a re-hash operation will be triggered to reassign the hash bit value based on the median of $Hash_{s_{i1},s_{i2}}(s_j)$. The total complexity of the re-hash operation for any digit is $O(n)$. For the example shown above, if the first hash bit for all subsequent four segments $\{s_5, s_6, s_7, s_8\}$ is 1, the re-hash mechanism will be triggered and reassign the first hash bit to achieve a more balanced distribution.

6) Collision and branching: Collision and branching are two common problems for all hash structures. In FollowUs, collision happens when different segments (non-match) are hashed to the same hash sequence. While branching refers to the problem where matching segments are hashed to different hash sequences.

For the hash collision problem, we can run the pairwise matching algorithm for all segments with the same hash sequence. Since the average number of segments that are hashed to a specific hash sequence is O(1), this operation only takes O(1) comparisons for every hash sequence.

The hash branching problem can make the system miss several pairs of matched segments. This problem is more severe when a bad set of reference traces is chosen. To improve overall system robustness, FollowUs introduces a redundancy mechanism: given a duplication factor K, it builds K hash structures to index traces under different sets of reference segments. All the operations are applied to different copies of hash structures. Intuitively, even though a pair of matched trace segments can hash to diverse hash sequences in one hash structure, they tend to be hashed to the same sequence in other hash structures. This method significantly reduces the branching ratio in segment indexing with no increase in computational complexity (since K is a constant). In Section V, we demonstrate the performance of hash structure duplication in terms of both accuracy and efficiency.

IV. Graph

Identifying segment matches from different traces is the first step to generate new traces. Finding a satisfactory trace between any two PoIs requires global knowledge of segment matching from all traces and how each segment connects to each other. It would be time-consuming to check all pairs of matched segments for each trace request. To bridge the gap, FollowUs uses segment graphs as an intermediary representation to aggregate segment matching results and connections. This section shows how to construct such segment graph, both with and without available indoor maps.

A. Graph and map models

We first define the graph model before introducing its construction process. A segment graph G is a representation of overall segment connectivity, while a map M is a real indoor floor plan. In FollowUs, such a graph G is modeled using segments and connectors, i.e., we define G as a set of segments $S^G = \{s_i^G\}_{i=1}^n$ and a set of connectors $C^G = \{c_j^G\}_{j=1}^m$. For an indoor map M, we define it as a set of pathways $S^M = \{s_i^M\}_{i=1}^n$ and a set of connectors $C^M = \{c_j^M\}_{j=1}^m$. Extracting pathways and connectors from floor plans can be achieved based on computer graphics or computational geometry, which is outside the scope of this paper. A map pathway and a graph segment differ only in that the segment includes a representation of sensor data. Each connector has a type, a corresponding attribute value, and the ID list of the adjacent segments/pathways.

By default, a trace graph is constructed with only segment matching information. If optional indoor maps are available, FollowUs first casts traces into the map – as constraints on the trace shape – and then applies a hidden Markov model to get a graph representation.

B. Graph construction – without map

We first present how to construct the graph without any map information. Algorithm 2 shows the graph construction flow. Given a set of traces T and a set of buckets B, where each bucket is a set of segments that match each other as input, the algorithm uses trace level connectivity information to connect different buckets with distinctive connector values (lines 1-7). The bucket function returns the corresponding bucket for each segment, and the value function returns the attribute value for each connector. Each bucket keeps a list of connector values representing individual links from itself to other buckets. The algorithm then finds the most probable next segment for every connector value for each bucket using majority vote (lines 8-13). After obtaining the connectivity between different segments, the segment graph can be obtained from the buckets and the intermediate connectors between them (lines 14-16).

After obtaining the indoor segment graph, FollowUs associates PoIs (i.e., starting point, destination, and trace connectors) with the corresponding places in the graph. When a user requests navigation service, she selects the start and destination points from the PoI database, and FollowUs automatically generates a new reference trace by searching the graph and selecting appropriate segments.

C. Graph construction – with map

FollowUs is designed to leverage indoor floor plans as additional information if available. Such map information can facilitate the segment graph construction process. Specifically, it helps FollowUs: i) get a clear representation

ALGORITHM 2: Graph construction without maps.

```
Input  : The trace set $T$ and bucket set $B$
Output: The trace graph $G$
for $i \leftarrow 1$ to $|T|$ do
    for $j \leftarrow 1$ to $|S^{t_i}|$-1 do
        $b_j^{t_i}$ = bucket($s_j^{t_i}$); $b_{j+1}^{t_i}$ = bucket($s_{j+1}^{t_i}$);
        $cvalue$ = value($c_j^{t_i}$);
        $b_j^{t_i}.cvalue$.append($b_{j+1}^{t_i}$); $b_{j+1}^{t_i}.cvalue$.append($b_j^{t_i}$);
    end
end
for $i \leftarrow 1$ to $|B|$ do
    for each connector value $cv_j$ do
        $b_k$ = findMost($b_i.cv_j$);
        $b_i.cv_j.next = b_k$;
    end
end
$G.S = B$ for all pairs of buckets $b_i, b_j$ do
    $G.C$.append($c_{b_i,b_j}$);
end
```

for segment graphs, ii) provide ground truth for all events and trace segments, and iii) reduce the dimension of the segment indexing structure. To fully take advantages of a map, we need to design methods that cast each trace into the map, while in the meantime dealing with potential mapping conflicts from new coming traces.

It takes three steps to construct a segment graph with map information: (1) cast each trace into the map using a novel map mapping algorithm; (2) generate representative signals (sensor data) for every segment mapped in the map; (3) utilize the map geometry to represent how to connect segments.

1) Trace projection into map:

The first step is to cast every trace into the map. Finding the best-match pathway for an abstract trace in the floor plan can be modeled as a search problem. However, it faces two challenges: metric selection and computational overhead.

Metric selection: Lengths of recorded traces and pathways in the map are not measured by a uniform metric (steps and meters respectively). It is non-trivial to convert steps into meters as users' step length vary.

Computational overhead: Pathway search in the map has exponential complexity due to the large amount of possible match candidates.[1]

Since a fixed average step length cannot adapt to various walking patterns, FollowUs uses a relative value of average step length as a metric to find best-match pathways in the map. Intuitively, longer segments from the trace should be cast into longer pathways on the map, vice versa for shorter ones. That means, different segments from the same trace should have roughly the same average step length. Therefore, we use the variance of the average step length for different segments as a metric to measure match quality. Formally, the problem is defined as follows:

Given a recorded trace U and a map M, pathway P is a possible matching candidate for trace U in M. $S^U = \{s_i^U\}_{i=1}^n$ and $S^P = \{s_i^P\}_{i=1}^n$ is the sequence of

[1]Proof is omitted due to space limitation.

segments of trace U and pathway P, respectively. The function $length(\cdot)$ is used to obtain the length of a segment. If the segment comes from a recorded trace, the function returns the number of steps in the segment, otherwise it returns the length in meters. We use $L^{(U,P)} = \{l_i^{(U,P)} = \frac{length(s_i^P)}{length(s_i^U)}\}$ to denote the average step length set for the matching between U and P. $C^U = \{c_i^U\}_{i=1}^n$ and $C^P = \{c_i^P\}_{i=1}^n$ is the sequence of connectors of trace U and pathway P, respectively. Every connector c_i^U or c_i^P can also be represented by two adjacent segments $< s_i^U, s_{i+1}^U >$ or $< s_i^P, s_{i+1}^P >$. We use $c_i^U \sim c_i^P$ to denote two connectors that have the same event type and matched event value (defined in Section III-A).

To differentiate the effect of segments with different lengths, we consider a weighted variance of $L^{(U,P)}$ as the matching metric. To sum up, the goal of finding the best match pathway in the map can be expressed as $\min\limits_P \text{var}(L^{(U,P)})$, subject to $P \in M$; $\forall i, c_i^U \sim c_i^P$.

From the analysis of the computational challenge we know that the number of potential pathways that fit a given trace event pattern is exponential. In view of this large search space, FollowUs improves the search algorithm along two dimensions: i) reducing the computation within each step and ii) reducing the number of search steps.

For the first goal, FollowUs incorporates an incremental method to update the weighted variance. This method fully utilizes the previous result to compute the result of the next step, and only takes $O(1)$ time for each step state update.

For the second goal, FollowUs adopts a novel pruning method. As can be proved,

$$\text{var}(L_{1:k+1}^{(U,P)}) = \text{var}(L_{1:k}^{(U,P)}) + \sum_{i=1}^{k} len(s_i^U)(\overline{l_{1:k+1}^{(U,P)}} - \overline{l_{1:k}^{(U,P)}})^2 + len(s_{k+1}^U)(l_{k+1}^{(U,P)} - \overline{l_{1:k+1}^{(U,P)}})^2 \quad (1)$$

This characteristic gives us a good way to prune the search: if the weighted variance of the average step length for the first several segments in Pathway A is already larger than that for all the segments in Pathway B, the search for Pathway A can be pruned due to the monotonic non-decreasing characteristic of the weighted variance of average step length.

The performance of pruning also relies on search order: the earlier the best-match pathway is found, the more unnecessary searches can be avoided. Based on this observation, FollowUs tries to expand the most promising search pathways in every iteration. To identify how promising a pathway is, we use the normalized step length variance $\text{norm}(L^{(U,P)}) = \frac{\text{var}(L^{(U,P)})}{\sum length(s_i^U)}$ as metric. Thus the algorithm chooses the pathway with minimal $\text{norm}(L^{(U,P)})$.

Algorithm 3 illustrates in detail the search algorithm. The algorithm takes as input a trace U, a map M, and the starting point c_{sp} in the map. During search, it maintains the search state in a priority queue *pqueue*. The search state contains several domains: *key*, *index*, *var*, *next*, *seg*, which refer to the average weighted step variance, index

ALGORITHM 3: Heuristic pathway search.

```
Input  : trace $U = \{S^U, C^U\}$, map $M = \{S^M, C^M\}$, $c_{sp}$.
Output: best-match pathway $P$ in $M$ that can minimize
        $\text{var}(L^{(U,P)})$ subject to $c_i^U \sim c_i^P$
$pqueue$.push(makestate($\emptyset, c_{sp}$));
while $pqueue \neq \emptyset$ do
    $headstate = pqueue$.pop();
    $index = headstate.index$;
    if $headstate.var > var_{min}$ then
        continue; // pruning
    end
    if $index == |S^U|$ then
        $P$ = traceback($headstate$);
        $var_{min} = headstate.var$;
        continue; // search possible better result
    end
    for each $seg \in headstate.next$ do
        $c =< headstate.seg, seg >$;
        if $c \sim c_{index}^U$ then
            $pqueue$.push(makestate($headstate, seg$));
        end
    end
end
```

of current matching segment, weighted step variance, the next search segments, and the last matched segment, respectively. The priority queue compares states by *key* and returns the minimum-key state. The makestate function is used to update the search state information incrementally based on the previous search state. The key part of this function is to upgrade the weighted variance of step length according to Equation 1. In lines 5-7, the algorithm performs pruning with the information of weighted variance of step length to reduce the search space. Once the algorithm finds a feasible solution, it traces back the search route to get its pathway (lines 8-12). Lines 13-18 check whether a trace event qualifies and pushes one more search state to the queue. The algorithm stops when the priority queue becomes empty.

2) Graph generation: After casting traces into a map, we can extract the most probable bucket for every segment in the map and connect them with corresponding navigation event connectors. Upon receiving querys of the starting point and destination, FollowUs matches them in PoI database, finds the shortest path and concatenates signal sequences of its segments to build a reference trace.

V. Implementation and Evaluation

We implemented FollowUs as an Android app[15] and a set of backend cloud services on Azure. The entire system consists of $\approx$ 56k lines of code (loc), broken into: Mobile Core 8.9kloc; FollowUs App 29kloc; Trace Processing Pipeline 13.6kloc; Path Builder 1.7kloc, and FollowUs Service API 2.5kloc.

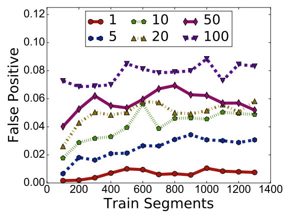
FollowUs has been released in a company-internal beta and thus far has more than 150 users (including developers, testers, and interns) on 52 different Android device models across tens of buildings in two continents. All participants recorded at least one trace and followed several other traces after watching a simple tutorial.

To fully test FollowUs in unfamiliar environments, all recorders are told to randomly choose a floor and intentionally create circuitous routes to ensure that followers are not following the path based on prior knowledge or building signage.

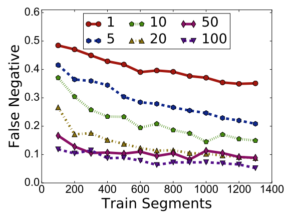
To evaluate FollowUs we utilize 373 indoor traces, with ground truth paths, covering an area of 6,000m^2. All traces were recorded with smartphone holding horizontal in front of the body. In what follows, we present the evaluation of the key components of FollowUs segment matching/indexing, graph construction/mapping, and finally end-to-end navigation (paths) with comparison to FollowMe [9].

A. Segment-level and Graph-level evaluation

FollowUs partitions traces into segments based on detected user motion events (e.g., turns, level changes). FollowUs successfully detects 97.7% of all events in the utilized dataset. Wrongly partitioned segments are then filtered out during segment matching as low-quality segments. Due to space limitations, we omit detailed segmentation results.



(a) False positive rate



(b) False negative rate

Figure 3. Segment matching results.

1) Segment matching: Segment matching is the basis for trace indexing and graph construction. We evaluate the performance of segment matching in terms of accuracy and efficiency.

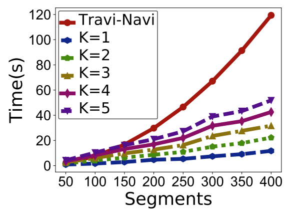
Accuracy: FollowUs uses SVM to classify DTW synchronization results between segments to determine whether segments are matched with each other. To better understand the effectiveness and limitations of this method, we evaluate the performance of the algorithm with different sizes of segment sets. Figure 3(a) and Figure 3(b) show both the false positive and false negative rates of FollowUs for different sizes of training data with different compression factors α, where $\alpha = n$ means we do down-sampling by averaging every n samples. A low compression factor (small α) results in a low false positive rate but a high false negative rate. This is because it preserves more distinct features of matched traces, thereby reducing false positives but at the same time including more noise. The accuracy is also found to be stable with more than 1000 training segments, indicating that the SVM model does not require a large amount of data to get trained well. Figure 3(a) and Figure 3(b) provide a guideline for the choice of α. Based on these results, we set $\alpha = 20$ in our implementation to optimize the trade-off between false positive and false negative rates.

Efficiency: The efficiency of segment matching is evaluated in terms of running time for different compression factors. To better measure the workload, we ran microbenchmarks locally on a server with an Intel core i7 3770 Processor (8M cache, up to 3.90 GHz) and a 16GB RAM. Results show that matching signals without compression takes more than 1s for segments with 2000 samples. Increasing α can substantially reduce computation time. For example, matching 2000-sample segments requires less than 10ms when $\alpha = 20$.

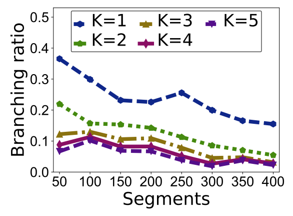
2) Segment indexing: We now evaluate the accuracy and efficiency of segment indexing. To demonstration the performance gain that we can obtain from the indexing mechanism in Section III-C, we compare the processing time with pair-wise comparison (the method that used by Travi-Navi to identify the matching relationship between segments). In addition, to understand the effect of duplication factor, K, we compare segment indexing performances under different K.

We first consider the time cost of segment indexing. Figure 4(a) shows the time of building the indexing structure with different methods for different size data sets. The pair-wise comparison that used by Travi-Navi increases quadratically, whereas FollowUs shows a nearly linear increase. The gap of processing time between the larger and larger as the number of traces increases. Therefore, the indexing technique used by FollowUs is advantageous over the pair-wise comparison used by Travi-Navi when the data volume becomes large, ensuring FollowUs' scalability.

In FollowUs, segment indexing uses a hash structure to determine segment matching, instead of exhaustively comparing every pair of segments. However, this may cause a lot of branching. Figure 4(b) shows the branching ratio of the system. In line with our design in Section III-C6, increasing the duplication factor K turns out to be highly effective at reducing the branching ratio. Considering both efficiency and accuracy, we set $K = 3$ in FollowUs.



(a) Time



(b) Branch ratio

Figure 4. Segment indexing results.

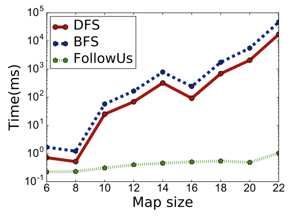
For the collision cases, we can run pairwise matching to identify un-matched pairs of segments. This will not cause any loss in accuracy but will incur time cost. During the overall experiments, comparisons to identify collision cases only takes up less than 5% of the overall segment indexing time, thus making it a small overhead to the entire system.

3) Trace mapping: Casting traces into a map helps FollowUs utilize an optional floor plan to improve the performance of trace graph construction. In Section IV, we use a heuristic search algorithm to find the corresponding

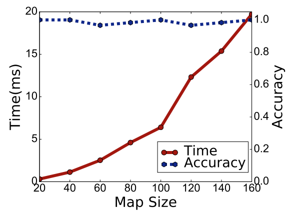
pathways on the map. To illustrate the advantage of this algorithm, we use traditional Depth-First Search (DFS) and Breadth-First Search (BFS) as comparisons and evaluate both accuracy and efficiency. In the evaluation, accuracy is measured by the percentage of trace segments that are correctly mapped.

For the real floor-plans in our testing environments, all three algorithms work well (equivalent accuracy within 100ms processing time on a local server). Hence, to demonstrate the advantage of our design, we run the three algorithms on large-scale simulated traces and maps. To this end, we first create a virtual map with $n \times n$ grids. Then, we remove some segments from the map based on a Bernoulli distribution $Bern(1, 0.1)$. To construct a virtual trace, we randomly select two points in the map as the starting point A and destination B, and search a path between A and B. At last, for every trace segment we synthesize its "step length" by sampling the number of steps from a normal distribution $N(l\mu, l\sigma^2)$, where l is the length for the trace segment, $\mu = 0.78, \sigma = 0.133$ are the average and standard error of step length for all traces in the experiment.

Figure 5(a) compares the trace-mapping times of different algorithms. The x-axis is map size. For example, if a map contains 3 horizontal paths and 2 vertical paths, the map size is 3+2=5. The figure shows that the time costs of both DFS and BFS increase exponentially, whereas our heuristic search grows much slower. To gain further insight into the performance of the heuristic search in FollowUs we also evaluate the correlation between computational time and accuracy (Figure 5(b)). The heuristic search in FollowUs is shown to have a processing time roughly linearly increasing with map size, but a high and stable mapping accuracy ($\approx 97.6\%$).



(a) Trace mapping time



(b) FollowUs performance

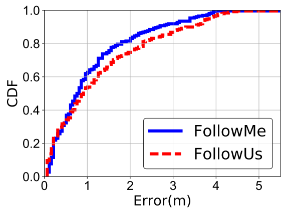
Figure 5. Trace mapping results.

B. End-to-end evaluation

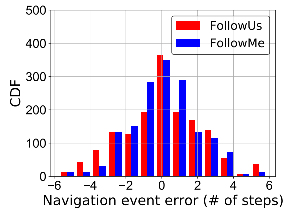
Finally, we evaluate the end-to-end navigation performance of FollowUs.

Reference trace: For the purpose of comparison, we choose the most similar indoor navigation system, FollowMe [9], as the baseline in the experiment. We use two types of reference traces in our experiments. One is directly recorded by users with FollowMe and the other is recorded and reconstructed by FollowUs. In total, 200 type-one traces as well as 200 type-two traces are used in this evaluation.

Metric: We use a spatial offset between the leader's locations (from linear interpolation) and the relative locations obtained by trace synchronization algorithm as an indicator of navigation performance. Since the primary concern of a navigation system lies in whether it can provide correct and timely navigation instructions to users, we also adopt a navigation metric called navigation event error. Specifically, it is defined as the difference (in number of steps) between the time when a navigation instruction e.g., turns, staircases, elevators, etc.) should be provided to user, compared to when it actually appears.



(a) Spatial error.



(b) Event error

Figure 6. End-to-end navigation performance.

We randomly choose 1000 navigation events from both type-one and type-two traces to calculate error distribution. In Figure 6(a), we find that spatial errors of FollowUs are close to FollowMe. For example, navigation with FollowUs and FollowMe achieves a 91 percentile accuracy of 3m (91% and 93.5%, respectively). The small gap between two CDF curves is caused by the glitch of magnetic signal at connectors between different segments. Similarly, Figure 6(b) shows that 95.1% and 98.8% of navigation event errors are below 4 steps for both FollowUs and FollowMe. From the distribution of navigation event errors, we can also see that both types of reference traces show comparable navigation performance. In summary, our end-to-end evaluation demonstrates adequate navigation performance, validating the design and confirming the robustness and accuracy of FollowUs.

VI. Related Work

Hardly any problem has been studied as extensively in the mobile computing community over the past decade as indoor localization. Many proposed devices [16, 17] and systems [1–4, 18, 19] using acoustic signals [20], Wi-Fi [1, 21], FM radio [6, 22], geomagnetism [8, 12, 23], GSM [24], RFID [25, 26], and light [27–30] achieve high localization accuracy. However, they usually incur large bootstrapping cost in terms of either hardware or data. Moreover, floor plans are usually required to enable navigation service beyond locations.

To ease the burden of such labor-intensive bootstrapping process, many crowdsourcing methods [31–36] have been proposed to assist localization and floor-plan generation. Despite the advances made in this direction, the high bootstrapping cost and dependence on infrastructure hinders these systems' wide-scale deployment in practice. FollowUs borrows from some of the ideas above, but differs from them in the following aspects.

First, as a location-free indoor navigation system, FollowUs does not rely on location coordinates and therefore circumvents obstacles during localization. Second, the data FollowUs uses to generate segment graphs comes from its incrementally-deployable navigation service. Hence, it successfully eliminates the Chicken-and-Egg dilemma between crowdsourcing data and navigation service (the navigation service is functional and useful even for the very first person storing a path), making the system easy to deploy. Third, in addition to the effectiveness, both segment matching and indexing, and graph construction are designed to be efficient, making FollowUs practical and scalable.

The works closest to FollowUs are Travi-Navi [10] and FollowMe [9]. FollowUs differs from these two systems as follows. First, as a navigation system, neither energy-hungry components such as camera and Wi-Fi, nor compute-intensive algorithms such as particle filtering are used in FollowUs. While retaining the merits of peer-to-peer navigation, FollowUs is also adaptive to maps and different modalities. These features make FollowUs easily deployable and practical. Second, in terms of scalability, FollowMe only allows navigation along a route created by one specific leader. Though Travi-Navi is capable of finding shortcuts and planning trips for users, it incurs $O(n^2)$ pairwise comparisons to find a potential trace match. In contrast, FollowUs adopts efficient indexing and segment matching algorithms with $O(n \log n)$ computational complexity, which allows it to scale. Third, FollowUs is the only system that constructs graphs based on user traces with or without floor plans. Instead of performing local optimization (to mitigate detours from overlapped traces) in Travi-Navi, segment graphs not only provide globally optimal paths for navigation, but they also play a key role in aggregating traces during FollowUs' incremental deployment.

VII. Conclusion

We presented FollowUs, an incrementally-deployable indoor navigation system with high flexibility and scalability. The system is powered by cloud services for accurate and efficient trace matching and indexing, and can also leverage optional floor plans to build trace graphs to further enhance navigation performance. We implemented FollowUs and our experimental results based on the beta dataset show that 91% of FollowUs' spatial errors are found to be 3m or less, and 95% of navigation instructions are shown to users within a 4-steps error margin for reconstructed traces.

References

[1] J. Xiong and K. Jamieson, "ArrayTrack: A Fine-Grained Indoor Location System," in NSDI, 2013.

[2] J. Wang, H. Jiang, J. Xiong, K. Jamieson, X. Chen, D. Fang, and B. Xie, "LiFS: Low Human Effort, Device-Free Localization with Fine-Grained Subcarrier Information," in MobiSys, 2016.

[3] M. Kotaru, K. Joshi, D. Bharadia, and S. Katti, "Spotfi: Decimeter level localization using wifi," in SIGCOMM, 2015.

[4] D. Vasisht, S. Kumar, and D. Katabi, "Decimeter-level localization with a single WiFi access point," in NSDI, 2016.

[5] P. Bahl and V. N. Padmanabhan, "RADAR: An In-Building RF-Based User Location and Tracking System," in INFOCOM, 2000.

[6] S. Yoon, K. Lee, and I. Rhee, "FM-based Indoor Localization via Automatic Fingerprint DB Construction and Matching," in MobiSys, 2013.

[7] I. Constandache, X. Bao, M. Azizyan, and R. R. Choudhury, "Did You See Bob?: Human Localization Using Mobile Phones," in MobiCom, 2010.

[8] T. H. Riehle, S. M. Anderson, P. A. Lichter, N. A. Giudice, S. I. Sheikh, R. J. Knuesel, D. T. Kollmann, and D. S. Hedin, "Indoor Magnetic Navigation for the Blind," in EMBC, 2012.

[9] Y. Shu, K. G. Shin, T. He, and J. Chen, "Last-Mile Navigation Using Smartphones," in MobiCom, 2015.

[10] Y. Zheng, G. Shen, L. Li, C. Zhao, M. Li, and F. Zhao, "Travi-navi: Self-deployable Indoor Navigation System," in MobiCom, 2014.

[11] Z. Yin, C. Wu, Z. Yang, N. Lane, and Y. Liu, "ppNav: Peer-to-Peer Indoor Navigation for Smartphones," in ICPADS, 2016.

[12] J. Chung, M. Donahoe, C. Schmandt, I.-J. Kim, P. Razavai, and M. Wiseman, "Indoor location sensing using geo-magnetism," in MobiSys, 2011.

[13] M. Müller, "Dynamic time warping," Information retrieval for music and motion, 2007.

[14] C. Faloutsos and K.-I. Lin, FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. SIGMOD, 1995.

[15] Microsoft, "Microsoft Path Guide," https://aka.ms/mspathguide.

[16] Apple, "iBeacon for Developers - Apple Developer."

[17] Intel, "Intel Wireless-AC 8x70 Product Brief (802.11mc featured)."

[18] Microsoft, "Microsoft Indoor Localization Competition – IPSN 2016."

[19] F. Li, C. Zhao, G. Ding, J. Gong, C. Liu, and F. Zhao, "A Reliable and Accurate Indoor Localization Method Using Phone Inertial Sensors," in UbiComp, 2012.

[20] K. Liu, X. Liu, and X. Li, "Guoguo: enabling fine-grained indoor localization via smartphone," in MobiSys, 2013.

[21] M. Youssef and A. Agrawala, "The Horus WLAN location determination system," in MobiSys, 2005.

[22] Y. Chen, D. Lymberopoulos, J. Liu, and B. Priyantha, "FM-based Indoor Localization," in ACM MobiSys, 2012.

[23] Y. Shu, C. Bo, G. Shen, C. Zhao, L. Li, and F. Zhao, "Magicol: Indoor Localization Using Pervasive Magnetic Field and Opportunistic WiFi Sensing," IEEE JSAC, vol. 33, no. 7, pp. 1443–1457, July 2015.

[24] V. Otsason, A. Varshavsky, A. LaMarca, and E. de Lara, "Accurate GSM Indoor Localization," in UbiComp, 2005.

[25] Y. Ma, X. Hui, and E. C. Kan, "3D Real-time Indoor Localization via Broadband Nonlinear Backscatter in Passive Devices with Centimeter Precision," in MobiCom, 2016.

[26] Y. Ma, N. Selby, and F. Adib, "Minding the Billions: Ultra-wideband Localization for Deployed RFID Tags," in MobiCom, 2017.

[27] Y.-S. Kuo, P. Pannuto, K.-J. Hsiao, and P. Dutta, "Luxapose: Indoor Positioning with Mobile Phones and Visible Light," in MobiCom, 2014.

[28] C. Zhang and X. Zhang, "Litell: robust indoor localization using unmodified light fixtures," in MobiCom, 2016.

[29] S. Zhu and X. Zhang, "Enabling High-Precision Visible Light Localization in Today's Buildings," in MobiSys, 2017.

[30] C. Zhang and X. Zhang, "Pulsar: Towards Ubiquitous Visible Light Localization," in MobiCom, 2017.

[31] A. Rai, K. K. Chintalapudi, V. N. Padmanabhan, and R. Sen, "Zee: Zero-effort Crowdsourcing for Indoor Localization," in MobiCom, 2012.

[32] S. Chen, M. Li, K. Ren, X. Fu, and C. Qiao, "Rise of the Indoor Crowd: Reconstruction of Building Interior View via Mobile Crowdsourcing," in SenSys, 2015.

[33] Z. Yang, C. Wu, and Y. Liu, "Locating in Fingerprint Space: Wireless Indoor Localization with Little Human Intervention," in MobiCom, 2012.

[34] M. Alzantot and M. Youssef, "CrowdInside: Automatic Construction of Indoor Floorplans," in ACM SIGSPATIAL GIS, 2012.

[35] R. Gao, M. Zhao, T. Ye, F. Ye, Y. Wang, K. Bian, T. Wang, and X. Li, "Jigsaw: Indoor Floor Plan Reconstruction via Mobile Crowdsensing," in MobiCom, 2014.

[36] G. Shen, Z. Chen, P. Zhang, T. Moscibroda, and Y. Zhang, "Walkie-Markie: indoor pathway mapping made easy," in NSDI, 2013.