# Indoor Tracking using Crowdsourced Maps

Jiang Dong[1], Yu Xiao[1], Zhonghong Ou[2], Yong Cui[3], and Antti Ylä-Jääski[1]

[1]Department of Computer Science, Aalto University, Finland
[2]School of Computer Science, Beijing University of Posts and Telecommunications, China
[3]Department of Computer Science and Technology, Tsinghua University, China

***Abstract*—Using crowdsourced visual and inertial sensor data for indoor mapping has attracted much attention in recent years. Nevertheless, the opportunities and challenges of indoor tracking using crowdsourced maps have not been fully explored. In this work, we aim at tackling the challenges due to incomplete obstacle information in crowdsourced indoor maps, especially at the initialization stage of crowdsourcing. We propose a novel solution for particle-filtering-based indoor tracking, using the crowdsourced maps derived from image-based 3D point clouds. Our solution enhances particle filtering with density-based collision detection and history-based particle regeneration. Evaluation with real user traces demonstrates that our solution outperforms the state-of-the-art. In particular, it reduces the average distance error of indoor tracking by 47% when using crowdsourced 3D point clouds.**

## I. Introduction

Indoor maps play an essential role in many indoor location applications, e.g., indoor positioning, real time tracking, and location based analytics. Recently, with the popularity of sensor-rich mobile devices, utilizing crowdsourced visual and ineritial sensor data for indoor mapping has attracted much attention. Examples include *Jigsaw* [7] and our previous work, *iMoon* [4]. These solutions adopt a similar approach which reconstructs 3D point clouds of indoor environment from videos/images, and then creates floor plans and/or navigation meshes from the point clouds. Along this line of research, we focus on smartphone-based indoor tracking using crowdsourced image-based 3D point clouds.

The quality of crowdsourced 3D point clouds, in terms of coverage and precision, depends on the quality and quantity of the data contributed by the crowd [5]. In the practice of crowdsourcing, it usually takes a long time to bootstrap a well established system. As a consequence, the generated 3D point clouds may not cover every corner of the indoor space at initialization stage. This is a common issue in crowdsourced indoor mapping, which poses challenges to the performance of indoor tracking using crowdsourced maps.

Particle filtering is one of the state-of-the-art techniques for indoor tracking [2, 8]. The existing solutions assume that fine-grained indoor maps, which contain complete and precise obstacle information, are available. Nevertheless, when using crowdsourced indoor maps, this assumption may not hold. Consequently, as we observe from our experiments (see Section IV-B), the accuracy of indoor tracking drops significantly when applying conventional particle filtering on crowdsourced 3D point clouds.

To tackle the aforementioned problem, we propose an Enhanced Particle Filtering (EPF) algorithm which improves the design of collision detection and particle regeneration. Our solution first classifies collisions into three types based on the root causes of collisions. It then conducts density based collision detection on an occupancy grid map [14], which illustrates the placements of obstacles at grid granularity. Furthermore, upon the occurrence of collisions, our solution regenerates particles based on the history of collisions, instead of the positions of the live particles nearby.

We evaluate the performance of EPF with a crowdsourced 3D point cloud of a public building and real user traces collected from there. The results show that EPF outperforms the state-of-the-art [9]. Particularly, it increases the accuracy of indoor tracking by 47%.

Our major contributions are summarized as follows: (1) To the best of our knowledge, this work is the first one that tackles challenges related to indoor tracking using crowdsourced maps, specifically the crowdsourced 3D point clouds. (2) We develop an enhanced particle filtering algorithm for indoor tracking, which takes into account the potential obstacle information missing from the crowdsourced maps. (3) We implement and evaluate our solution using real user traces.

## II. Background and Related Work

In this section, we will introduce the background of crowdsourced indoor mapping, with the focus on the solutions that utilize crowdsourced visual sensor data. After that, we will give an overview of the state-of-the-art of particle filtering based indoor tracking algorithms.

**Indoor mapping using crowdsourced visual sensor data.** SLAM (Simultaneously Localization And Navigation) [10, 11] techniques have been widely adopted in robotic mapping. The concept of SLAM is to build a map of an unknown environment by a mobile robot while at the same time navigating the environment using the map. To implement SLAM, there is a need for a mobile robot and a range measurement device, e.g., laser scanner. These devices are more expensive and difficult to operate, compared with more widely available mobile devices like smartphones.

With the wide availability of cameras on mobile devices, utilizing crowdsourced visual sensor data for indoor mapping has attracted much attention, as visual sensor data like videos

[1]This work was supported by the Academy of Finland (grant number 278207 and 268096) and NSFC 61422206.

978-1-5090-0802-5/16/$31.00 ©2016 IEEE

and images provide abundant information about indoor environments. *CrowdMap* [3] and *Jigsaw* [7] are example solutions of crowdsourced indoor floor plan reconstruction. The first one uses crowdsourced video and inertial sensor data, while the latter takes crowdsourced photos as input. Similar to *Jigsaw*, *iMoon* [4] creates indoor maps based on 3D point clouds that are built from crowdsourced photos using SfM [6, 13] techniques. With SfM-based 3D point clouds, *iMoon* also provides the feature of image-based localization, which allows users to precisely locate themselves by taking photos. In this work, we focus on indoor tracking using crowdsourced 3D point clouds, and will refer to *iMoon* for the implementation of 3D point cloud reconstruction and image-based localization.

**Particle filtering based indoor tracking.** Particle filtering has been a de facto approach for indoor tracking. A particle filtering based indoor tracking algorithm defines how particles move under the constraints of the map, and how to regenerate particles when collisions occur. Its performance can be affected by the accuracy of step detection and stride length estimation. Previous works [9, 12, 15] assume that a fine-grained indoor map including all the obstacle information is available. Based on this assumption, Rai et al. [12] designed an augmented particle filtering algorithm that simultaneously estimates location and user-specific walk characteristics such as stride length. Li et al. [9] developed novel algorithms that provide reliable step detection and accurate stride length estimation. In this work, we implemented the step detection and stride length estimation based on [9]. Our contributions focus on the enhancement of particle filtering, in terms of collision detection and particle regeneration, in order to provide accurate indoor tracking using crowdsourced maps in which obstacle information may be partly missing.

## III. System Design and Implementation

This work focuses on indoor tracking using crowdsourced 3D point clouds of indoor environments. As illustrated in Figure 1, we propose to enhance the conventional particle filtering based approaches by applying occupancy grid mapping, density-based collision detection, and history-based particle regeneration. The details are presented in this section.

### A. Occupancy Grid Mapping

This work utilizes the 3D point clouds generated from crowdsourced photos using VisualSFM [1], a state-of-the-art SfM tool chain. Given a 3D point cloud, our system detects the 3D points representing floors and ceilings based on the z-axis value, which stands for the height of the object. The threshold for floors is set to 0.2 m, while the one for ceilings is set to 2.5 m. After removing all the points representing floors and ceilings, the remaining ones represent the obstacles in the space. We project the remaining 3D points onto a ground plane, and discretize the domain by a grid of cells. Herein a cell is defined as a square with the length of each side equal to 0.15 m.

Denote a grid cell by $c_{i,j}$ and index it with a unique coordinate $(i,\ j)$. For each point $(x,\ y)$, the coordinate of the grid cell

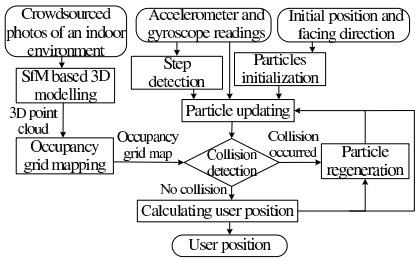



Fig. 1: A system overview.

it belongs to can be calculated by $i = \lfloor (y - MIN_y)/0.15 \rfloor + 1$ and $j = \lfloor (x - MIN_x)/0.15 \rfloor + 1$, where $MIN_x$ and $MIN_y$ refer to the minimum x-axis and y-axis values of the area.

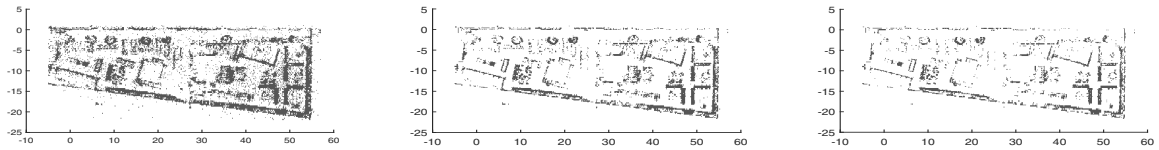
We define an occupied grid cell as one that is covered by an obstacle. In order to separate the noise from the signal, we measure the number of 3D points projected inside each cell, and define the minimum number of points in an occupied grid cell as a threshold $T$. The principle of setting $T$ is to remove as much noise as possible with minimum amount of signal loss. As shown in Figure 2, when $T$ is set to 1, most of the noise still remain in the 3D point cloud. On the contrary, when the value of $T$ is too big, e.g. 5, some obstacle information starts fading out undesirably. Based on this trade-off, we empirically set the value of $T$ to be 3 in our experiments.

### B. Enhanced Particle Filtering

We propose to apply an enhanced particle filtering algorithm on the occupancy grid map described in Section III-A. As illustrated in Algorithm 1, our algorithm takes as input the accelerometer and gyroscope readings collected from the user's mobile device, and provides an estimate of the user's position at step granularity. Our algorithm is built on the assumption that the user's initial position and facing direction are known. The pipeline of our algorithm is described as follows.

**Particle initialization**. The first step is to initialize $N$ particles based on the user's initial position, denoted by ($x^0$, $y^0$), and the facing direction, denoted by $\theta^0$. For each particle $p_i$, we define its position and direction at time $t$ as ($x_i^t$, $y_i^t$) and $\theta_i^t$, respectively. The initial position of each particle is determined by the initial position of the user and a pair of noise elements, denoted by ($\Delta x$, $\Delta y$). The noise elements follow standard normal distribution. The same rules apply to the definition of $\theta_i^0$, as presented in line 4 of Algorithm 1. In our experiments, we utilize the feature of image-based localization for obtaining the initial position and facing direction. As reported in [4], the error in distance and facing direction is on average 2 m and 6 degrees, respectively. Accordingly, we set the standard deviation of distance to be 0.5 m, and that of angle to be 1.5 degrees.

For each $p_i$, we also give an estimate of the user's stride length (see line 2 of Algorithm 1). It is computed based on the step frequency model proposed in [9]. As the stride length varies between people and the actual value is unknown when initializing the particles, we set the default values of the coefficients, denoted by $a_i$ and $b_i$, as a constant plus a noise following standard normal distribution. The coefficient



(a) $T = 1$ (b) $T = 3$ (c) $T = 5$

Fig. 2: Comparison of occupancy grid maps with different settings of the minimum amount of points in each cell. The minimum number is defined by the value of $T$.

**Algorithm 1** Enhanced particle filtering.

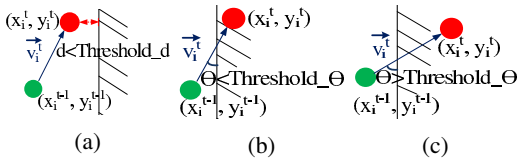
**Input:** User's initial position and facing direction: $(x^0, y^0)$, $\theta^0$;
Accelerometer and gyroscope readings: $acc$, $gyro$;
Occupancy grid map: $M$;
**Output:** User's position at time $t$: $(x^t, y^t)$;

1: //Define $l_i$ as the estimate of stride length, and $f$ as step frequency
2: $l_i = a_i \times f + b_i$;
3: **for** $i \in N$ // Initiate $N$ particles
4: $(x_i{}^0, y_i{}^0, \theta_i{}^0, a_i, b_i) = (x^0 + \Delta x, y^0 + \Delta y, \theta^0 + \Delta\theta, a + \Delta a, b + \Delta b)$;
5: **if** one step is detected
6: **for** $i \in N$
7: //Update $N$ particles
8: $x_i{}^t = x_i{}^{t-1} + (l_i + \Delta l) \times cos(\theta_i{}^t + \theta(t) + \Delta\theta)$;
9: $y_i{}^t = y_i{}^{t-1} + (l_i + \Delta l) \times sin(\theta_i{}^t + \theta(t) + \Delta\theta)$;
10: //Collision detection
11: $collision_i{}^t = CollisionDetect(\vec{v_i}^t, M)$;
12: $CSS_i[t] = collision_i{}^t$
13: //Particle weighting
14: $w_i = 1/2^{\sum_{t-n+1<j\leq t} CSS_i[j]}$
15: **for** $i \in N$ //Particle weight normalization
16: $w_i' = w_i / \sum_{j \in N} w_j$;
17: $(x^t, y^t) = (\sum_{i \in N} w_i' \times x_i{}^t, \sum_{i \in N} w_i' \times y_i{}^t)$;//Estimate user position
18: //Particle regeneration
19: **for** $i \in N$
20: **if** $collision_i{}^t \neq 0$
21: $j = SelectLiveParticle(x_i{}^t, y_i{}^t, CSS)$;
22: **if** $TURN == 1$ AND $collision_i{}^t == 3$
23: $(x_i{}^t, y_i{}^t, a_i, b_i) = (x_j{}^t + \Delta x, y_j{}^t + \Delta y, a_j + \Delta a, b_j + \Delta b)$;
24: **else**
25: $(x_i{}^t, y_i{}^t, \theta_i{}^t) = (x_j{}^t + \Delta x, y_j{}^t + \Delta y, \theta_j{}^t + \Delta\theta)$;

values ($a_i$, $b_i$) will be adjusted when a turn or a certain type of collision is detected.

**Particle updating**. We apply the step detection algorithm proposed in [9] to detect steps based on the readings of accelerometer. When a step is detected, all the particles are updated, as described in lines 8 and 9 of Algorithm 1. Note that $\Delta l$ follows standard normal distribution; $\theta(t)$ is the accumulated direction offset between two successive steps, which is calculated from the gyroscope readings.

**Collision detection**. Every time after updating the particles, the system checks if any collision has occurred. In previous works [9, 12], obstacle information is assumed to be complete and precise in the indoor map. Based on this assumption, collision occurs only when the particle in question hits a wall or any other obstacle. In case of using crowdsourced maps, the obstacle information may be partly missing. For example, a wall may be fragmented into pieces so that the narrow gaps between fragments become "walkable" areas. When a particle tries to go through any of these gaps, the collision would not be caught if using conventional particle filtering algorithms. To increase the chances of catching such collisions, we propose to check also the density of occupancy grid cells along user's moving path. The detailed rules of collision detection are described as follows.




(a) (b) (c)

Fig. 3: Three types of collisions. (a) Type 1: too close to an obstacle. (b) Type 2: crossing an obstacle due to accumulated error of gyroscope. (c) Type 3: crossing an obstacle due to the error of stride length.

We classify collisions into 3 types. In Type 1 (see Figure 3a), the particle is getting too close to an obstacle. Namely, the distance between the particle and the obstacle is less than a safe distance for walking. In the other two types, a particle is trying to get into or cross an obstacle. We define a vector $\vec{v_i^t}$, which indicates the moving path from $(x_i{}^{t-1}, y_i{}^{t-1})$ to $(x_i{}^t, y_i{}^t)$. If the angle between $\vec{v_i^t}$ and the outline of the obstacle is smaller than a threshold, as illustrated in Figure 3b, a Type 2 collision is identified; otherwise, a Type 3 collision is detected.

We argue that Type 2 collisions are caused by the accumulated error of gyroscope readings. Imagine that a user is walking straight along a corridor. Her walking trace can be considered to be parallel to the walls. Biased gyroscope readings cause error in the estimate of the walking direction, with the result that the calculated walking trace is eventually intersected with one of the walls. Type 3 collisions are usually caused by the error in the estimate of stride length. For example, when the stride length is over-estimated, the walking trace may give a wrong impression that the user has bumped her head on the wall in front.

The process of identifying different types of collisions consists of four steps. Firstly, it forms a set of neighbouring grid cells that includes all the cells crossed by the particle along $v_i^t$. After that, for each of the cell included in the set, their neighbouring cells within a distance of 0.3 m are added to the set as well. Secondly, as shown in Figure 4, the set of neighbouring grid cells are partitioned into four zones based on their positions relative to $\vec{v_i^t}$ . The numbers of the occupied grid cells corresponding to each zone are denoted by $N1$, $N2$, $N3$ and $N4$, respectively. Thirdly, based on the number

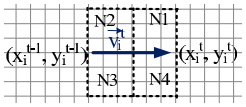



Fig. 4: Zoning of the neighbouring grid cells along $\vec{v_i^t}$.

of occupied cells in each zone, it checks whether collision exists. In principle, if the occupied grid cells are sparse, the probability of causing collision is low and can be safely set to zero. Otherwise, collision is assumed to exist. In practice, we calculate the sum of occupied cells (defined as $N_s$) in the four zones, and claim a collision if the sum is bigger than a threshold $T_1$. We assume that an obstacle always occupies more than 1 grid cell, and therefore set the value of $T_1$ to 1 in our experiments. Fourthly, if a collision is detected, we further identify the type of the collision. If most of the occupied cells stay on the same side of $\vec{v_i^t}$, i.e., $(N1 + N2)/N_s \geq 0.8$ or $(N3+N4)/N_s \geq 0.8$, it is recognized as a Type 1 collision. If it does not belong to Type 1, we check if either $N1/(N1+N4)$ or $N2/(N2+N3)$ falls within the range of (0.2, 0.8)[1]. If this condition is satisfied, we consider the particle is heading to an obstacle and the collision belongs to Type 3; otherwise, the collision belongs to Type 2.

Each particle keeps a collision state sequence (CSS), which records the collisions that happened in the previous steps. At each step, if no collision is detected, the collision status is 0; otherwise, the collision status is indicated by the type of collision, such as 1 for Type 1 collision. CSS is used for particle weighting and regeneration.

**Particle weighting**. Every particle has a weight, which reflects the confidence in the result of collision detection. The weight value of each particle is updated when one step is detected. At time $t$, the weight $w_i$ for particle $p_i$ is calculated as illustrated in line 14 of Algorithm 1. Instead of simply setting the weight to 0 or 1 based on the collision status of the current step, we take the collision status of last $n$ steps into consideration. This is based on the fact that a particle that survives for longer time usually has a higher probability of being in the right position. In our experiment, we empirically set $n$ as 3. After that, the weights of all the particles are normalized and then used for calculating the user's position.

**Particle regeneration.** Any particle that encounters collision will be regenerated. In conventional particle filtering algorithms, a particle is regenerated based on the positions of the live ones nearby. Differently from that, our approach takes into account the history of collisions. It first sorts all the particles by the number of successive steps without any collision. This number is obtained from CSS. After that, the first 15% of the particles which have the biggest numbers are selected and categorized as long live particles. When a particle $p_i$ dies, our algorithm selects one long live particle that stays nearby as reference for generating a new particle. When a Type 3 collision is detected while the user is making a turn, the position and the coefficients of the step frequency model

[1]We tried various combinations, but this simple setting of range produces comparable results.

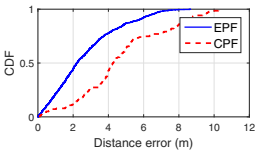



Fig. 5: Comparison of distance error between EPF and CPF.

are adjusted based on line 23 of Algorithm 1[2]. Otherwise, the position and direction of the new particle are calculated following line 25 of Algorithm 1.

## IV. Evaluation

We evaluate the performance of our indoor tracking algorithm with real user traces, and compare it with the state-of-the-art [9]. In this section, we will first describe our experimental setup, and then discuss our experiment results.

### A. Experimental Setup

We collected in total 50 traces, including over 5,000 steps, between September and October 2015. The traces were collected by 5 volunteers from a public indoor space, approximately 1,110 $m^2$, including a library and a cafeteria. The volunteers were requested to walk through the routes which had been tagged with precise coordinates. These coordinates provided the ground truth of user traces. While walking, the volunteers always carried a smartphone on which the accelerometer and gyroscope readings were being collected. In addition to the traces, a 3D point cloud of the indoor space in question was generated from around 2,000 photos collected in advance by different users[3] We created an occupancy grid map from the 3D point cloud and used it for experiment. The map is visualized in Figure 2b.

We implemented our EPF algorithm based on the occupancy grid map, and visualized the experiment results, including the estimated user traces, using Matlab. A demo video that shows how our algorithm updates user position on the move is available online[4].

To compare the performance of our proposal with previous work, we also implemented the Conventional Particle Filtering (CPF) algorithm according to the descriptions in [9]. We applied CPF on an obstacle map compiled from the 3D point cloud in 3 steps. First, we removed the 3D points representing floors and ceiling, as described in Section III-A. Second, we projected the remaining 3D points on the X-Y plane and grouped the points by distance. Third, each group of points were wrapped with a non-convex polygon, representing the outline of an obstacle.

In our implementation of CPF, a collision is detected when the vector $\vec{v_i^t}$, defined in Section III-B, intersects with the outline of any obstacle present in the map. Upon the occurrence of collision, the weight of the particle is reduced

[2]Li et al. [9] proposed to adjust the coefficient values of the step model upon the detection of turns. We tighten the constraint in order to differentiate the collisions due to inaccurate stride length.

[3]Detailed description of these photos and the generated 3D point cloud is presented in our previous work [4].

[4]Demo video URL: https://youtu.be/WU96VXzWkrQ.

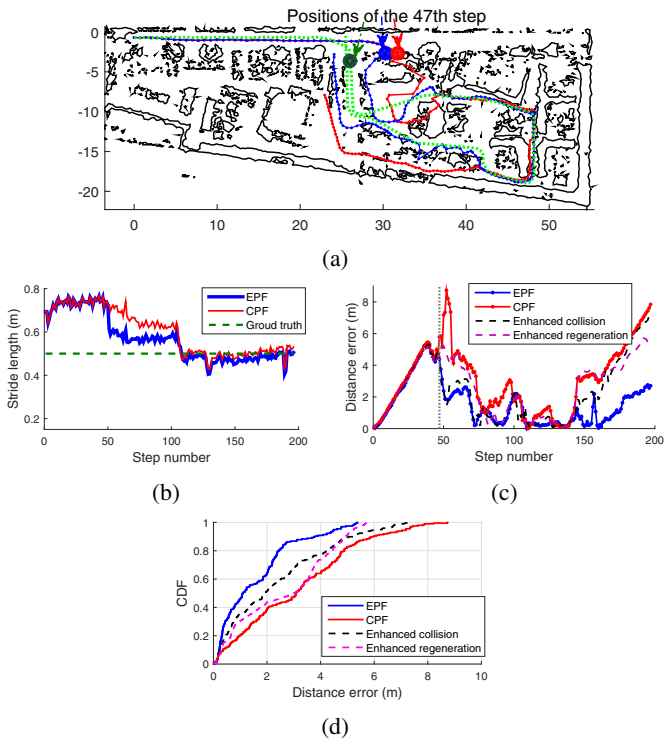



Fig. 6: (a) Comparison between the ground truth (green) of a walking trace and the traces estimated by EPF (blue) and CPF (red). (b) Estimates of stride length along the walking trace. (c) Comparison of distance error at step granularity between CPF, EPF, and the scenarios where only the density-based collision detection (black dashed line) or the history-based particle generation (purple dashed line) is applied. The grey dotted line refers to the 47th step. (d) Comparison of the overall distance error along the trace.

from 1 to 0. Meanwhile, a new one will be generated based on the live particles nearby. In practice, the new one is configured according to line 23 of Algorithm 1 if the collision happens while the user is making a turn. Otherwise, it is regenerated based on line 25 of Algorithm 1.

### *B. Performance Evaluation*

We compare the performance of indoor tracking, in terms of distance error, between EPF and CPF. The distance error is calculated as the difference between the estimated user position and the ground truth corresponding to each step. From Figure 5 we can clearly see that EPF outperforms CPF. In general, the accuracy of EPF is 47% higher than CPF. The average distance error is 2.4 m for EPF and 4.5 m for CPF.

To provide more detailed analysis of the results, we choose one user trace as example and compare in Figure 6a the respective traces created by EPF and CPF. To ease the comparison, we visualize the traces on the same obstacle map used by CPF. Based on the same set of accelerometer and gyroscope readings, as we can see from Figure 6a, the trace estimated by EPF is closer to the ground truth in most of time. If we take a closer look at the distance error corresponding to each step along the trace, as illustrated in Figure 6c, the distance errors are relatively big during the first and the last 40 steps. The bigger errors are mainly because of the variance of stride length and the information missing from the map in use.

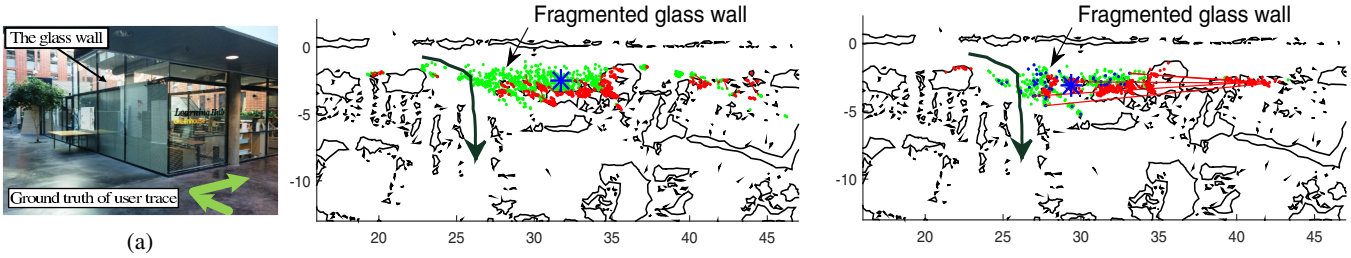
**Variance of stride length**. The distance error accumulated in the first 40 steps when using either EPF or CPF. This is mainly due to the variance of stride length. Unless the stride length is known beforehand, by default, particle filtering based approaches apply a general step model [9] for estimating the stride length. The values of the coefficients are adjusted when a turn is detected. In case the user keeps walking along a straight path, the coefficient values remain the same that the error would accumulate throughput the whole way. This is an inherited limitation of particle filtering based approaches. In practice, the more turns the user has made, the better the estimate of the stride length to the ground truth becomes. We compare the estimates of stride lengths between EPF and CPF in Figure 6b. When using EPF, the estimate of stride length decreases from around 0.7 m to 0.6 m after the first turn is made. After multiple turns, the variance of the estimates becomes smaller, and the estimated stride length stays around 0.5 m, which is close to the ground truth. Although the estimated stride length using CPF is getting close to the ground truth as well, the distance error does not decrease as fast as using EPF. The reason arises from the difference in the ways of handling the potential information missing from the crowdsourced maps, which is discussed as follows.

**Incomplete obstacle information**. In an extreme case where the obstacle information in a certain area is completely missing from the map, walking in that area is treated in the same way as walking in an open space without any constraint. From particle filtering point of view, no collision will occur as no obstacle exists. Namely, user position is estimated solely from the accelerometer and gyroscope readings. Thus, the accumulated error cannot be eliminated through enhancement of particle filtering itself. This explains why the distance error accumulates during the last 40 steps of the example trace (see Figure 6c), when the user was walking in an area with little obstacle information available in the map.

**Collision detection and particle regeneration**. In most cases, there is a certain amount of obstacle information available, although some is still missing. For example, part of the glass wall shown in Figure 7a was missing from the 3D point clouds. In such scenarios, EPF performs better than CPF because of the different strategies of collision detection and particle regeneration principles, as explained below.

CPF assumes that the map is complete and accurate, and collision occurs only when a particle hits a wall or other obstacles. When the obstacle information is partially missing, CPF is not able to capture all the collisions. When a collision is detected, CPF tries to locate the live particles close by and regenerate a new one based on it. Figure 7b depicts the snapshot of the particles when the 47th step in the user trace (see Figure 6a) is detected. The actual moving path of the user is indicated in Figure 7a. In fact, the user first turned right, and then walked along the glass wall.

Due to the lack of data, the glass wall was not fully




(a) (b) CPF (c) EPF

Fig. 7: Comparison of particle regeneration between EPF and CPF, taking the snapshots of particles corresponding to the 47th step as examples. (a) The user's view while taking the 47th step. (b) and (c) Snapshots of particles in case of CPF and EPF. The dark green arrow lines indicate the ground truth of user trace, corresponding to the one indicated in Figure 7a. The green, red and blue dots indicate the positions of the live, dead and long lived particles, respectively. The blue star shows the estimated user position, and the red lines shows how far the new particles are placed away from the dead ones.

reconstructed in the crowdsourced map. In other words, the wall was fragmented, as we can see the obstacle map. In case of CPF, when some particles try to get through the narrow gaps between fragments, the collisions may not be caught, as the conditions of collision are not satisfied. In case of EPF, the system checks the density of the neighbouring grid cells along the moving path, through which the narrow gaps and further the particles trying to go through the naps can be detected.

When collisions are caught, CPF regenerates the particles in question and relocate them near the previous positions. The short red lines in Figure 7b show where the particles are relocated relative to their previous locations. Obviously, this regeneration method increases the probability of collisions in the following steps. EPF takes a different strategy and determines the location of the new particle based on the history of collisions. In principle, the new particles will be placed close to the long lived particles, as visualized in Figure 7c. Compared with CPF, our approach helps to reduce the number of subsequent collisions.

**Effectiveness of each enhancement.** We compared the distance error between the scenarios where particle filtering is enhanced by density-based collision detection, history-based particle regeneration, or their combination (a.k.a EPF). Density-based collision detection is designed for solving the issue of obstacle fragmentation. In the first scenario, as shown in Figure 6d, the enhanced particle filtering achieves lower distance error than CPF. However, in the last 40 steps (see Figure 6c), the distance error of EPF is much lower. In the second scenario, dead particles can quickly converge to the long-lived particles, which are considered as the most trustful ones. Nonetheless, without density-based collision detection, some particles may penetrate fragmented obstacles, with the result of having more particles converging to those wrong particles. We can see from Figure 6d that the distance error increases in this scenario. Therefore, it is necessary to combine both enhancements in order to achieve the best performance.

## V. Conclusions

We presented in this paper a novel approach of particle filtering based indoor tracking using crowdsourced maps. We enhanced particle filtering by improving the design of collision detection and particle regeneration, in order to maintain the accuracy of indoor tracking in the scenarios where the obstacle information is partly missing from the map. We utilized the map created from crowdsourced 3D point clouds, and evaluated the performance of our approach with real user traces. Benefited from the enhancement of particle filtering, our approach has proved to provide more accurate and robust results than the conventional one. In the future, we plan to evaluate it also with other types of crowdsourced maps.

## References

[1] Visualsfm: A visual structure from motion system. http://ccwu.me/vsfm/.
[2] K. Al Nuaimi and H. Kamel. A survey of indoor positioning systems and algorithms. In *Proc. of Innovations '11*, pages 185–190, April 2011.
[3] S. Chen, M. Li, K. Ren, and C. Qiao. Crowdmap: Accurate reconstruction of indoor floor plans from crowdsourced sensor-rich videos. In *Proc. of ICDCS '15*, pages 1–10, Columbus, Ohio, USA.
[4] J. Dong, Y. Xiao, M. Noreikis, Z. Ou, and A. Ylä-Jääski. imoon: Using smartphones for image-based indoor navigation. In *Proc. of SenSys '15*, pages 85–97, Seoul, Korea.
[5] J. Dong, Y. Xiao, Z. Ou, and A. Ylä-Jääski. Utilizing internet photos for indoor mapping and localization - opportunities and challenges. In *Proc. of SmartCity '15*, pages 636–641, Hong Kong.
[6] J.-M. Frahm, P. Fite-Georgel, D. Gallup, T. Johnson, R. Raguram, C. Wu, Y.-H. Jen, E. Dunn, B. Clipp, S. Lazebnik, and M. Pollefeys. Building rome on a cloudless day. In *Proc. of ECCV'10*, pages 368–381, Berlin.
[7] 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 *Proc. of MobiCom '14*, pages 249–260, Maui, Hawaii, USA.
[8] R. Harle. A survey of indoor inertial positioning systems for pedestrians. *IEEE Communications Surveys and Tutorials*, 15:1281–1293, 2013.
[9] 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 *Proc. of UbiComp '12*, pages 421–430, Pittsburgh, Pennsylvania.
[10] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Proc. of ISMAR 2011*, pages 127–136, Oct 2011.
[11] B. E. Okorn, X. Xiong, B. Akinci, and D. Huber. Toward automated modeling of floor plans. In *Proc. of 3DPVT 2010*, Paris, France.
[12] A. Rai, K. K. Chintalapudi, V. N. Padmanabhan, and R. Sen. Zee: Zero-effort crowdsourcing for indoor localization. In *Proc. of Mobicom '12*, pages 293–304, Istanbul, Turkey.
[13] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: Exploring photo collections in 3d. *ACM Trans. Graph.*, 25(3):835–846, July 2006.
[14] S. Thrun. Learning occupancy grid maps with forward sensor models. *Auton. Robots*, 15(2):111–127, Sept. 2003.
[15] O. Woodman and R. Harle. Pedestrian localisation for indoor environments. In *Proc. of UbiComp '08*, pages 114–123, Seoul, Korea.