Automatic Construction of Garage Maps for Future Vehicle Navigation Service

Qian Zhou*, Fan Ye*, Xiaoge Wang†, Yuanyuan Yang*
*Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11794, USA
{qian.zhou, fan.ye, yuanyuan.yang}@stonybrook.edu
†Institute for Cyber-Enabled Research, Michigan State University, East Lansing, MI 48824, USA
wangx147@msu.edu

Abstract—**Digital garage maps are the basis for future vehicle navigation services such as smart parking management that displays the availability of parking spaces. It can direct drivers to empty ones, avoiding any searching, circulating in large, complex parking structures. However, such maps are not currently available, making it impossible to deploy smart parking management. Conducting manual survey incurs tremendous amount of human efforts, and cannot scale to large numbers of garages. In this paper, we propose three algorithms, Sequential Merging, Points Clustering and Segments Matching that can automatically construct complete and accurate garage maps using data crowdsensed from drivers. Upon entering and leaving the garage, the driver's smartphone collects inertial data, which are used to generate the vehicle's trajectory. Our algorithms fuse together these trajectories to recreate the size, layout of the garage. We compare the performance of the three algorithms using different garages. We find that Points Clustering is robust to trajectory errors, with F-score above 0.95 for trajectory length error up to 2 meters, Segments Matching can handle partial trajectories with arbitrary start/end locations, and it constructs the same map using trajectories much shorter than those needed by the other two algorithms.**

I. Introduction

Due to the prevalence of digital maps, detailed navigation instructions have become standard for vehicles. However, when one drives into indoor environments, such as large underground parking lots, the world suddenly turns dark because no maps exist. Without such maps a lot of future vehicle services, such as smart parking [1], [2] that can display available parking spots, and direct drivers to empty ones, will be impossible. Currently there has been little effort addressing the problem. Google Indoor Maps [3] have started to cover indoor buildings, but not specifically for large parking structures. Some recent researches [4], [5] create algorithms to construct indoor floor maps using sensing data from mobile users. However, they are intended for buildings occupied by humans where richer categories of data (WiFi, images, etc.) can be leveraged and more flexible human operations are allowed (e.g., users can move freely and take photos.). Thus, those techniques are not suitable for the particular garage environment.

In this paper, we propose a crowdsensing [6] based approach to reconstruct parking structure maps using mobile data from drivers' smartphones. A driver's phone can record the inertial data as he drives through the entrance, follows certain paths and eventually parks the car at a spot. Each driver's data can be used to generate a corresponding trajectory. Given enough numbers of such trajectories, we can assemble these fragments to construct the complete map of the parking structure.

Such a crowdsensing based approach has several advantages: 1) It gathers data from many drivers who come in and out of garages every day, thus there is no need to hire dedicated personnel. 2) It does not require special hardware infrastructure; the driver's smartphone can collect the data needed for trajectories; 3) There is no need for special user operation. A driver only needs to allow a data recording app run in the background.

However, creating complete, accurate garage maps from crowdsensed data is not trivial. First, exactly how to assemble the fragmented, individual trajectory samples into a complete parking structure map? Second, the trajectory samples from the real world always contain inevitable errors and noises. How can we ensure the robustness such that final maps are accurate?

We propose three algorithms that can automatically construct garage maps from such trajectory samples. The Sequential Merging and Points Clustering methods take full, complete trajectories that start from a common entrance point. Sequential Merging is the simplest; Points Clustering identifies common turning points in different trajectories by more advanced clustering. Thus it achieves better accuracy and robustness. To accommodate incomplete trajectories that may start from any location in the garage, which can be common among casual drivers, we design a Segments Matching method that takes trajectories starting/ending at arbitrary locations. It leverages the lengths and orientations in segments to fuse trajectories.

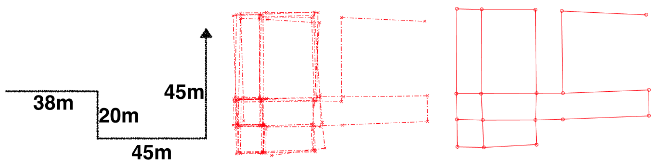
We make the following contributions in this paper: First, we propose two algorithms that can create garage maps using trajectories starting from the same entrance point. They handle cases where enough data are collected to generate complete trajectories, from entrance to the parking spot. Second, we propose a third algorithm that can use partial trajectories with arbitrary starting and ending points. This accommodates more challenging real world cases where drivers may only collect incomplete data, thus partial trajectories. Third, we evaluate the three algorithms systematically under different parking structures. We find that Points Clustering is more robust to errors in trajectory with F-score above 0.95 when trajectory length error is within 2 meters, and Segments Matching can handle trajectories with arbitrary start/end locations, and it constructs the same map using trajectories much shorter than those needed by the other two algorithms. To the best of our knowledge, we are the first to develop algorithms that can construct garage maps automatically from crowdsensed driver data.

II. Models and Assumptions

We assume that the vehicle's movement trajectories are made available to our algorithms. Such trajectories can be constructed from inertial and OBD vehicle speed data [7], [8] that a smartphone can easily collect. Each trajectory is a polygonal line, made up of multiple segments. From the OBD speed data, the length of each segment can be computed [9]; the gyroscope of the phone can tell the turning angle between two consecutive segments. Thus a trajectory (Figure 1. (a)) can be presented in the following format: $A_1, L_1, A_2, L_2, ..., A_n, L_n$, where n is the number of line segments, A_i the ith angle and L_i the ith length ($1 \le i \le n$).

A parking lot can be abstracted as a graph consisting of vertices and edges, with a vertex corresponding to a turn, or entrance/exit. Thus the map can be drawn given all vertices' coordinates and the paths connecting them (Figure 1. (b) and (c)), which are produced by our algorithms.

The OBD speed measurements are relatively reliable [10], but some errors in segment length estimations are possible. The gyroscope can measure the vehicle turning angle quite accurately [7]. Due to construction code [11], the turning angle between driving paths in a garage is usually a few limited candidates (e.g., $90°, 45°$). Thus the measured angles can be rectified to the closest candidate values.



(a) a trajectory (b) input set (c) output

Fig. 1. (a) a trajectory represented as: $0°, 38, -90°, 20, 90°, 45, 90°, 45$. (b) and (c): the input set is 10 trajectory samples from a parking lot, and the output is the reconstructed map.

The key insight for fusing trajectories into maps is to identify the overlapping relationship among trajectories. It imposes a constraint on how one trajectory can be placed relative to another in a global coordinate system. To support underground parking lots where GPS signals cannot penetrate, we assume GPS is available only for the entrance/exit of the garage, but not inside. Thus a trajectory has at most one point with known location. For partial trajectories that start/end inside the garage, no point has known GPS location. The challenge is how to find enough, reliable overlapping relationships among trajectories, so that all of them can be placed correctly relative to each other. We will present three algorithms that can use full or partial trajectories to form garage maps.

III. Algorithm 1: Sequential Merging

Sequential Merging assumes that all trajectories have the same starting point - the entrance of the parking lot. It chooses one trajectory randomly from the input set as the initial map, merge the remaining trajectories into the map one at a time, until all of them are processed. Each merging evolves the map to better completeness.

To merge a trajectory into the current map, we place its starting point at the common, known entrance, and rotate the trajectory so the first segment is oriented the same as the entrance path on the map. For each vertex on the trajectory, if it is close enough to an existing vertex on the map (e.g., $\le T_m$, a threshold), it is considered the same point and merged. Otherwise it is considered a new one and will be added to the map. T_m can be chosen empirically to be smaller than the minimum segment length in the parking garage. We finally set T_m to 15m based on our field trips which find that the segment length in a real garage is usually above 20m.

A. Steps

1) Build the initial map: Choose one trajectory from the set, assign coordinates $(0, 0)$ to the starting point, and calculate the coordinates for every vertex along the trajectory following their lengths and turning angles.

2) Merge another trajectory: Pick another trajectory, place its starting point at $(0, 0)$, rotate it around $(0, 0)$ such that the first segment orients the same as the entrance path into the garage. Calculate the coordinates for every vertex along the trajectory.

3) For each vertex V_t at (x_t, y_t) in this trajectory, find which vertex V_m at (x_m, y_m) in current map is closest. If the distance is below T_m, merge V_t and V_m into one new vertex whose coordinates are:

$$x'_m = \frac{x_t + x_m * w}{1 + w}$$

$$y'_m = \frac{y_t + y_m * w}{1 + w}$$

where w is the number of the vertices which have been merged to get the coordinates of V_m. If the distance is above T_m, V_t is considered a new vertex, and added to the map.

4) Repeat 2), 3) until all trajectories are merged.

IV. Algorithm 2: Points Clustering

Sequential Merging incorporates trajectories one at a time and reconstructs the map progressively. One critical limitation is that its performance depends on the merging order. If a low-quality trajectory is picked first, it can influence subsequent merging adversely. We propose a second algorithm that eliminates this drawback.

Points Clustering also assumes all trajectories share the same starting point. However, it merges all trajectories simultaneously. All trajectories are placed to start at the same origin and their entrance path segments oriented at the same angle. Vertices corresponding to the same turning point appear as clusters. We use k-means algorithm [12] to compute the centers of those clusters and use them as vertices in the map.

The original k-means algorithm requires the right k value (the number of clusters) as an input for good performance. However, k is initially unknown. Two mistakes can happen when k is incorrect: "over" estimation when one cluster is incorrectly recognized as two or more, "under" estimation vice versa. Furthermore, k-means' clustering effect is also affected by the distribution of cluster centers, which are usually randomly picked and always have some randomness. Thus "under" and "over" estimations can still happen even with the right k (Fig. 2. (a)).

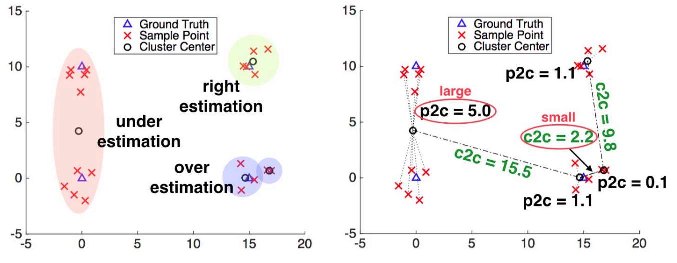


Fig. 2. (a) "under" or "over" estimations can still happen when k is correct. (b) "under" estimation leads to an abnormally large $p2c$ while "over" estimation leads to an abnormally small $c2c$.

We use a search strategy to find the right k: first increase k to get rid of all "under" estimations, then decrease to eliminate "over" estimations. We detect "under" and "over" estimations using two values (Fig. 2. (b)): 1) $p2c$, the average distance between a cluster center and all points within that cluster. We compute $p2c$ of all clusters, and see if the maximum one is abnormally large. Two or more clusters incorrectly recognized as one will cause a "cluster" spanning wide area, thus large $p2c$. 2) $c2c$, the distance between a cluster center and the nearest other center. We compute all pairwise $c2c$ values and see if the minimum one is abnormally small. One cluster incorrectly recognized as two or more will cause "tiny" clusters extremely close to each other, thus small $c2c$.

A. Steps

1) Place all trajectories from the same starting point $(0, 0)$ and rotate them so the first segments are oriented the same. Pick an initial value of k of the average number of vertices in trajectories. Since the algorithm adjusts k, the initial value does not impact the final result.

2) Eliminate all "under" estimations. Uses k-means [12] to identify k clusters and compute coordinates of their centers. To generate better initial cluster centers, k-means++ [13] is used for the very first time because it tends to choose k points far away from each other, while k-means picks them randomly and may result in clusters too close to each other. Then calculate $p2c$ for all clusters. If the maximum $p2c$ is above a threshold d_1, an "under" estimation is thought to have happened, and one random position in that cluster is added as a new center. Increase k by 1, call k-means using the new k value and cluster center set as input. Repeat this step until the maximum $p2c$ is less than d_1.

3) Eliminate all "over" estimations. Call k-means and compute $c2c$ for all pairwise cluster centers. If the minimum $c2c$ is below a threshold d_2, an "over" estimation is thought to have happened. One of the two corresponding cluster centers is removed from the center set. Decrease k by 1, call k-means using the new k value and cluster center set as input. Repeat this step until the minimum $c2c$ is above d_2.

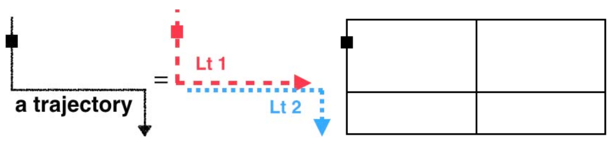
The two thresholds d_1, d_2 can be chosen based on building code and empirical knowledge of garages [11]. For example, if straight path segments are at least 20 meters, two centers much closer than $20m$ (say half at $10m$) is very unlikely. To tolerate errors in path length measurements, we use $15m$ as threshold d_2. For a straight path segment, points corresponding to one end should be closer to this end than the other. Thus the $p2c$ should not exceed half the segment length. To tolerate errors, we use a value slightly larger (e.g., $15m$).

We also tried other criteria to stop the elimination iterations. One common method is to observe the rate of changes of these parameters over iterations. E.g., when there are "over" estimations, the minimum $c2c$ will be always small, thus the rate of change is small. As the last "over" estimation is eliminated, the minimum $c2c$ will suddenly jump to a much larger value, causing a big change rate. Thus we can detect when "over" estimations are gone. We find this method has similar performance, but the two thresholds method is simpler and so we decide to use it.

V. Algorithm 3: Segments Matching

The previous two methods require complete trajectories that start from the entrance point to parking spots. In reality this may not always be possible. Users may collect only partial trajectories starting inside the garage, and ending at any other location in the garage. We propose an algorithm that can deal with partial trajectories.

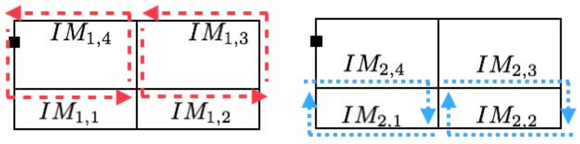
Segments Matching chooses the longest trajectory in the set as the initial map, then merges remaining trajectories one at a time. Because trajectories no longer start at the same point, a new trajectory cannot be placed directly on the map. Thus comparing distances to find which vertices correspond to the same turning point becomes infeasible. We detect segment overlap between trajectories as constraints, and give each one a certain position and orientation on the common plane, so the total overlap is maximized.



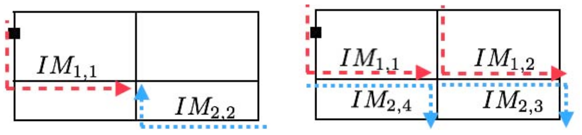
(a) break up a trajectory (b) existing map

Fig. 3. (a) break up a trajectory sample into multiple L-components (L_t). (b) existing map can also break into multiple L-components (L_m) and then match with L_t. A square represents a speed bump detected.

Specifically, we first break up a trajectory into multiple "L-components", each with two consecutive segments in shape like "L" (Fig. 3. (a)). If one L-component L_t from a trajectory has similar angle to another L_m from the map (Fig. 3. (b))(e.g., the turning angles differ less than 5°), then a match is declared, and a matching score is computed: $\frac{|L_t^{head} - L_m^{head}|}{L_m^{head}} + \frac{|L_t^{tail} - L_m^{tail}|}{L_m^{tail}}$, where $L_t^{head}, L_t^{tail}, L_m^{head}, L_m^{tail}$ are the lengths of the head, tail segment of L_t, L_m respectively. Usually, an L_t can match multiple L_ms in the map, and we use $IM_{i,j}$ to denote the jth match of the ith L_t (Fig. 4. (a)).



(a) all possible matches between a L_t and the map



(b) find out simultaneous matches

Fig. 4. (a) L_t1 can independently match to the map in 4 ways, so is L_t2. (b) $IM_{1,1}$ and $IM_{2,2}$, for instance, cannot happen simultaneously, otherwise the geometrical shape of the trajectory will be unable to maintain. Finally, only two combinations are left: ($IM_{1,1}$, $IM_{2,4}$) and ($IM_{1,2}$, $IM_{2,3}$).

Since each trajectory is a rigid object, the relative positions and orientations among its L-components are fixed. When the trajectory is placed onto the map to create a match $IM_{i,j}$, the positions and orientations of other L-components are determined (Fig. 4. (b)). Some of them may also match the map in different ways. Intuitively, we want to maximize such simultaneous matches, so the trajectory overlaps the "most" with the map.

To this end, we detect which simultaneous matches can occur. Then we sum up the scores of these matches as the score of the placement, and select the placement with the maximum score to merge the trajectory. We detect simultaneous match as follows: connect the middle vertices of two L-components on the trajectory, and compute d, the length of connecting line; α, β, the angle between the connecting line and the arrow head of two L-components. We also measure the same d', α', β' for the matching L-components on the map. If respective values differ less than a threshold (e.g., 5% for d, d', $5°$ for angles), we consider them a simultaneous match.

A. Steps

1) Choose the trajectory with the most segments as the initial map.

2) Pick one trajectory from the remaining set, break it up into L-components. For each L-component L_t^i, find all its matches $IM_{i,j}$, and compute their scores.

3) Find out all simultaneous matches: for each $IM_{i,j}$, use the method described earlier to find other simultaneous L-component matches. Sum up the scores of simultaneous matches for a placement score. The placement with the maximum score is adopted to merge the trajectory into the map. Repeat the same to merge each remaining trajectory.

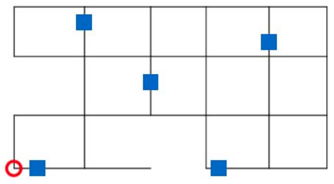
The above uses only the segment lengths and angles as constraints. Sometimes other constraints are possible, e.g., when landmarks such as speed bumps exist. Such bumps can be detected reliably through accelerometer data [7]. The number and locations of bumps on L-components have to be the same to declare a match (e.g., same number, location difference less than a threshold). In Fig. 4. (b), if speed bump information is leveraged, ($IM_{1,1}$, $IM_{2,4}$) can be easily determined as the unique simultaneous match. We will evaluate how this can help improve matching performance.

VI. Evaluation

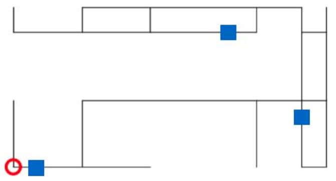
A. Methodology

To evaluate the algorithms' performance, we develop a map generator that generates "ground truth" maps. The sets of trajectories are generated from such maps. The trajectories are then fed to the algorithms to produce its reconstructed map. We compare the reconstructed ones with the "ground truth" for performance.

We do not use real garage maps because obtaining them is too time-consuming and effort-intensive for test to be taken at scale. Instead, we generalize garages' common actual parameters through field trips and consulting construction code [11]. A map generator based on those parameters is developed which can produce a lot of realistic ground truth garage maps. For each map, multiple trajectories are generated following a random walk. The starting point is either the entrance, or an arbitrary vertex of the map, then each time a random neighboring vertex is picked to form the next segment on the trajectory. We add length errors to segments to simulate measurement inaccuracy. The turning angles can have errors. We assume a small number of candidate angles are possible in a garage, and a pre-processing stage will rectify them to the nearest candidate angle. We test 10 regular garage maps and 10 irregular ones. For each garage, we generate two sets each containing 10 trajectories, either starting from the entrance, or randomly inside the garage.



(a) an exemplar regular map



(b) an exemplar irregular map

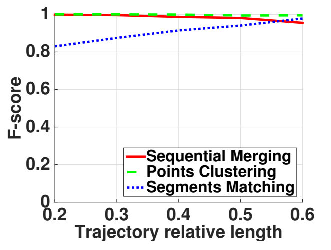
Fig. 5. The parameters of map can be configured: (a) a regular map consists of similar small grids arranged neatly (red circle represents the entrance and blue square for a speed bump). (b) an irregular map has more missing parts and its grids have various sizes.

We evaluate the impact of two map factors on the performance of the algorithms: 1) Map regularity. A regular map (Fig. 5. (a)) is close to a grid with evenly distributed rows and columns. It has at most 5 segments in width and 3 in height. The length of each segment is about 50 meters. In an irregular one (Fig. 5. (b)) some grid segments are missing, thus the distribution of rows and columns are more random. We use two maps of high and low regularity. 2) Bump ratio: the fraction of segments with speed bumps. We start from no bumps, and gradually increase the fraction of segments with bumps. There are techniques [7] that detect bumps reliably along trajectories.

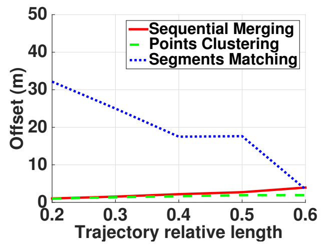
There are two trajectory factors impacting the performance: 1) Relative length: the ratio of the number of segments in the trajectory to the total number of segments in the map. 2) Length error. We add a zero-mean Gaussian error to each segment and use the standard deviation as the parameter. We do not add angle errors for two reasons: 1) The gyroscope can measure turning angles very accurately (e.g., $1° \sim 2°$); 2) Possible path orientations are usually limited. So angle errors can be removed by rectifying the path to the nearest candidate orientation.

We use three metrics to evaluate the performance of algorithms: 1) $F\text{-}score = \frac{2*Precision*Recall}{Precision+Recall}$, where precision is the ratio of the number of the correctly reconstructed vertices to the number of all constructed vertices, and recall the ratio of the number of the correctly constructed vertices to the number of all vertices in the part of ground truth map that has been travelled by the trajectories. F-score quantifies the balance between reconstruction accuracy and completeness. An "aggressive" algorithm may include all true segments, but also non-exist ones; a "conservative" one does the contrary, including some true ones, but missing other true ones. An ideal algorithm should achieve a balance. 2) Coverage: the ratio of the number of the correctly constructed vertices to the number of all vertices in the whole ground truth map. 3) Offset: Directed Hausdorff Distance [14], the greatest of all pairwise distances from a vertex in the travelled ground truth map to the closest constructed vertex.

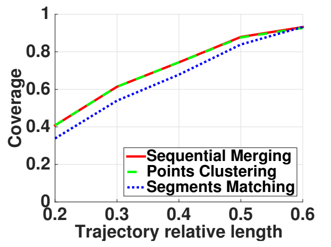
B. *Impact of trajectory length (entrance starting point)*



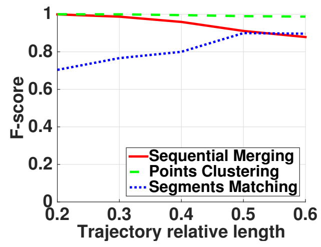
(a) using map of high irregular rate



(b) using map of high irregular rate



(c) using map of high irregular rate



(d) using map of low irregular rate

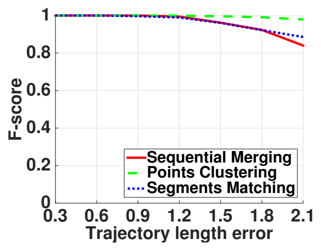
Fig. 6. The performances of three algorithms all change as the trajectory relative length increases, but change differently.

We evaluate how the trajectory length impacts the performance. As is shown in Fig. 6. (a) and (b), for a highly irregular map, the F-scores of Sequential Merging and Points Clustering fall slightly when trajectories grow longer, and their offsets increase. This is because longer trajectories accumulate more errors for vertices, especially those later in the sequence. Points Clustering shows better performance than Sequential Merging because it processes all trajectories simultaneously instead sequentially.

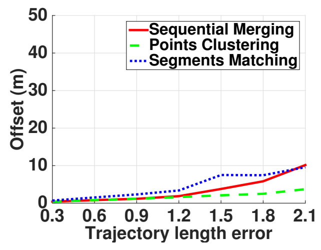
Segments Matching has low F-score (0.8) for short trajectories, and gradually reaches almost 1 for longer ones. This is because short trajectories do not contain sufficient constraints to determine the correct match. For long trajectories, its offset is comparable with the other two algorithms. All three algorithms have similar coverage increase as longer trajectories are used (Fig. 6. (c)). When the relative length is 0.6, a constructed map covers more than 90% of the whole parking structure. This is simply because longer ones cover more segments in the garage.

We repeat the experiments with regular map (Fig. 6. (d)). Sequential Merging and Points Clustering are not affected remarkably, but Segments Matching deteriorates: its F-score is lower than 0.8 before the relative length reaches 0.4. This is because a regular map has segments and L-components of similar angles and lengths. A trajectory (especially a short one) can match in too many ways. There are not enough constraints to find the correct placement. But as trajectories grow longer, more constraints are brought in and finally the F-score becomes acceptable (0.9). For coverage and offset, there is not much change for the three algorithms compared to those in irregular map. The only exception is Segments Matching's offset, which roughly doubles. It is because segment lengths are similar in regular map, thus many possible matching can happen. There are not enough constraints to produce the correct result. We omit the figures due to space limit.

C. *Impact of length error*



(a)



(b)

Fig. 7. The performances of three algorithms under different length errors.

We evaluate how length error impacts the performance (Fig. 7). To avoid the influence of segments of similar lengths in regular map (especially Segments Matching), we use irregular map. As the length error increases, the F-score of each algorithm falls gradually and offset increases. Among the three, Points Clustering is the most robust because it processes all trajectories simultaneously. When the length error reaches 2 meters, the F-score of Points Clustering is still above 0.95, with offset less than 5 meters. Sequential Merging and Segments Matching have F-scores above 0.8 and offsets below 10 meters when length error is within 2 meters. We find that coverage is nearly 80% and does not change much for all algorithms, so we omit the figure.

D. *Impact of trajectory length (arbitrary starting point)*

We evaluate the impact of trajectory length when trajectories have arbitrary start/end locations. For similar reasons, we use irregular map. Fig. 8. (a), (b) and (c) show that as longer trajectories are used as input, Segments Matching's performance improves. Its F-score reaches 0.9, offset is within

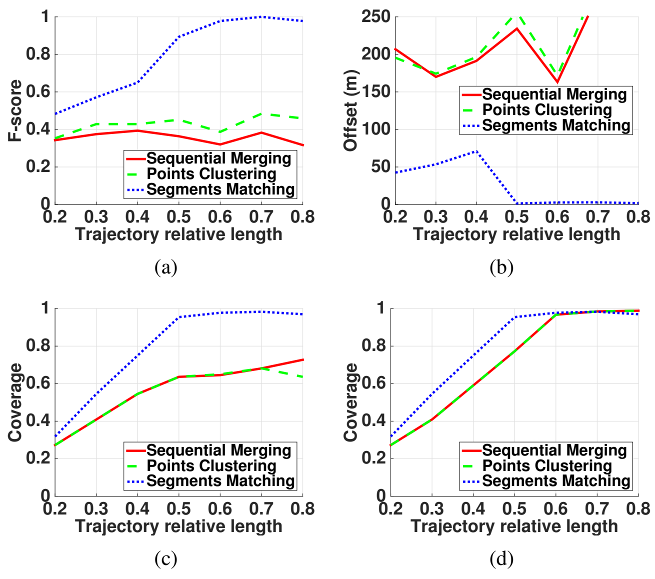


Fig. 8. Segments Matching is the only method that can take trajectories with arbitrary starting points as input.

10 meters and coverage exceeds 90% when the relative length is 0.5. In contrast, Sequential Merging and Points Clustering have much worse or even unacceptable performances (F-score 0.4, offset 200m, coverage 0.6), because they require common starting points to identify which vertices are actually the same turning point.

We also compare the coverage of Segments Matching using trajectories with arbitrary starting points with that of Sequential Merging and Points Clustering using trajectories starting from the entrance (Fig. 8. (d)). We find that Sequential Merging and Points Clustering need trajectories of relative length 0.6 to reach more than 90% coverage, while Segments Matching needs only 0.5. The reason is, to reach any destination in the garage, a random location inside can be much closer to the destination than the entrance point. However, Segments Matching does need relatively longer trajectories (e.g. more than 0.3) to obtain enough constraints for correct match.

E. Impact of the number of bumps

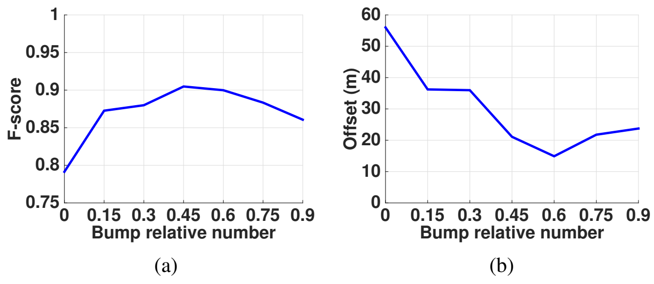


Fig. 9. The number of bumps affects the performance of Segments Matching

Segments Matching is the only one among the three that leverages speed bumps to further improve the performance. As is shown in Fig. 9, as the number of speed bumps increases, the performance is first improved but begins to deteriorate after it exceeds 0.6. This is because speed bumps serve as additional constraints to help distinguish the right match from other possible ones. However, when all segments have bumps, they become similar and lose the distinction. Thus excessive bumps lower the matching accuracy.

F. Time complexity analysis

The complexity of Sequential Merging is $O(m^2n^2)$, where m is the number of trajectories, and n is the average number of segments in a trajectory. Points Clustering calls Lloyd's k-means algorithm which is often considered linear complexity and the worst case can be $O(m^3n^3)$. Segments Matching's running time is $O(m^3n^4)$. All three algorithms are feasible in reality for the following reasons: 1) Generating garage map is not a real-time task, so there is sufficient time to run the algorithms and produce the map. 2) The algorithms run on the backend, which has abundant computing resources. 3) For a actual garage, m and n are usually within 20 and 10 respectively.

VII. Discussion

In our current design we assume that the turning angles between driving paths take a limited number of candidate values (e.g., $90°, 45°$). The measured angles can be rectified to the closest candidate value, thus eliminating the angle error. Although this assumption is true for most garages, we plan to enhance our algorithms so they can tolerate angle errors.

Currently we depend on OBD data to obtain the vehicle speed, thus computing the length of traveled segment. OBD adaptors are cheap (e.g., about \$20) and do not require wired connection (e.g., Bluetooth or WiFi). There exist many smartphone apps [15] that can stream OBD data to a phone. Thus the data collection does not incur much cost or efforts. To make it more convenient, we plan to develop methods that use only smartphone data to estimate travelled distance.

Our garage map generator is built based on survey of real garages and construction code [11]. It helps us evaluate the performance over many maps without intensive survey. In the future we plan to test our algorithms on real garages of different internal structures to ensure they are widely applicable.

GPS signal state can be used to tell whether a vehicle is inside a garage. The loss of the signal indicates that the vehicle is entering the garage. Generally a vehicle will slow down when it enters a garage. Thus the system can check GPS signal state when such inertial events happen to efficiently detect the entrance to a garage.

The map we produce consists of driving paths, but not parking spaces. Although many paths have parking spaces on both sides, some path may have spaces on one side, or no space at all. So we cannot simply add standard size parking spaces along each reconstructed path. We plan to solve the problem by detecting vehicle parking from trajectories: a vehicle makes a turn (e.g., usually $90°$) into a parking space, and stops after moving a vehicle's distance. These special patterns can be recognized to find parking spaces, thus adding them to make complete maps.

The maps we generate are one-level only. For garages of multiple levels, the critical issue is how to detect the movement across levels. Since the vehicle has to drive up slope to climb to the next level, the pitch onto such slopes can be detected from the phone's gyroscope [7]. Thus the pattern of level ground to slope then level again, and some distance on the slope, can be used to detect new levels. We will gather data in real garages and develop robust algorithms to recognize such patterns.

VIII. Related Work

There exist many GPS-based techniques for outdoor road map construction. Mahmud Ahmed et al. [16] present a simple and practical incremental algorithm based on partial matching of trajectories to the map. James Biagioni et al. [17] use KDE algorithm to automatically infer maps from large collections of opportunistically collected GPS traces. However, all of them depend on GPS signals, which are suitable outdoors but not indoor environments (e.g., underground parking structures) where GPS may not penetrate.

Peng Zhuang [18], Yiming Ji [19] and Huimin Wang et al. [20] leverage radio signals to construct indoor maps. However, they require pervasively deployed WiFi APs, which are usually not present in garages. Our work uses inertial data from devices carried by the driver, and does not require any pre-existing infrastructure.

Some researchers including R.C. Luo [21] and A Ohya [22] use intelligent robots to map the indoor environments. Although they can produce highly accurate maps with details, they incur special hardware and high expenses. It is difficult to scale to large numbers of garages in short time at low costs. Our work leverages commodity mobile devices such as smartphones that drivers already carry everywhere, thus our solution can scale at small costs.

Indoor floor plan construction has attracted the attention of mobile community and some pioneering work (e.g., CrowdInside [4], Jigsaw [5]) constructs such plans using inertial, image, radio data crowdsensed from smartphones users. We follow the same crowdsensing paradigm [6] to leverage pervasively available smartphones to eliminate dedicated infrastructure, and we target a quite different environment of garages for which such work are not designed.

IX. Conclusion

The lack of garage maps is a fundamental obstacle to future vehicle navigation services such as smart parking management. In this paper we propose and compare three algorithms that can take the vehicle movement trajectories crowdsensed from drivers to construct garage maps automatically. Sequential Merging and Points Clustering require trajectories all start from the same entrance point. They can create maps of high precision, recall and low offset. Points Clustering is more robust to errors in trajectories because it incorporates all trajectories simultaneously and does not depend on the merging order. Segments Matching is the only one that can handle partial trajectories that may start or end at arbitrary locations inside the garage. It can produce high quality maps given trajectories of sufficient lengths, and leverage additional information such as speed bumps for better results.

X. Acknowledgement

This research work was supported in part by the grant from the US National Science Foundation under grant number CSR 1513719.

References

[1] R. Lu, X. Lin, H. Zhu, and X. S. Shen, "Spark: a new vanet-based smart parking scheme for large parking lots," in *INFOCOM 2009, IEEE*. IEEE, 2009, pp. 1413–1421.

[2] S. Liniger and B. Stiller, "Parking prediction techniques in an iot environment," Ph.D. dissertation, Master Thesis University of Zurich, Zurich, Switzerland, 2015.

[3] "Google maps," https://www.google.com/maps/about/partners/indoormaps/.

[4] M. Alzantot and M. Youssef, "Crowdinside: automatic construction of indoor floorplans," in *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*. ACM, 2012, pp. 99–108.

[5] 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 *Proceedings of the 20th annual international conference on Mobile computing and networking*. ACM, 2014, pp. 249–260.

[6] R. K. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: current state and future challenges," *Communications Magazine, IEEE*, vol. 49, no. 11, pp. 32–39, 2011.

[7] M. Zhao, T. Ye, R. Gao, F. Ye, Y. Wang, and G. Luo, "Vetrack: Real time vehicle tracking in uninstrumented indoor environments," in *Proceedings of the 4th ACM Conference on Embedded Networked Sensor Systems (ACM SenSys)*, November 2015.

[8] R. K. Ganti, N. Pham, H. Ahmadi, S. Nangia, and T. F. Abdelzaher, "Greengps: a participatory sensing fuel-efficient maps application," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 151–164.

[9] A. Chowdhury, T. Chakravarty, and P. Balamuralidhar, "A novel approach to improve vehicle speed estimation using smartphone?s ins/gps sensors."

[10] A. Goodwin, "A brief intro to obd-ii technology," http://www.cnet.com/news/a-brief-intro-to-obd-ii-technology/.

[11] "Parking requirements," http://chicago47.org/wp-content/uploads/Building-Codes-for-Garage-Construction.pdf.

[12] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Applied statistics*, pp. 100–108, 1979.

[13] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[14] G. Rote, "Computing the minimum hausdorff distance between two point sets on a line under translation," *Information Processing Letters*, vol. 38, no. 3, pp. 123–127, 1991.

[15] "Obd2 diagnostics for windows and android and ios," https://www.obdsoftware.net.

[16] M. Ahmed and C. Wenk, "Constructing street networks from gps trajectories," in *Algorithms–ESA 2012*. Springer, 2012, pp. 60–71.

[17] J. Biagioni and J. Eriksson, "Map inference in the face of noise and disparity," in *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*. ACM, 2012, pp. 79–88.

[18] P. Zhuang, D. Wang, and Y. Shang, "Smart: Simultaneous indoor localization and map construction using smartphones," in *Neural Networks (IJCNN), The 2010 International Joint Conference on*. IEEE, 2010, pp. 1–8.

[19] Y. Ji, S. Biaz, S. Pandey, and P. Agrawal, "Ariadne: a dynamic indoor signal map construction and localization system," in *Proceedings of the 4th international conference on Mobile systems, applications and services*. ACM, 2006, pp. 151–164.

[20] H. Wang, L. Ma, Y. Xu, and Z. Deng, "Dynamic radio map construction for wlan indoor location," in *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2011 International Conference on*, vol. 2. IEEE, 2011, pp. 162–165.

[21] R. C. Luo and C. C. Lai, "Enriched indoor map construction based on multisensor fusion approach for intelligent service robot," *Industrial Electronics, IEEE Transactions on*, vol. 59, no. 8, pp. 3135–3145, 2012.

[22] A. Ohya, Y. Nagashima, and S. Yuta, "Exploring unknown environment and map construction using ultrasonic sensing of normal direction of walls," in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*. IEEE, 1994, pp. 485–492.