APT: Accurate Outdoor Pedestrian Tracking with Smartphones

Xiaojun Zhu*†, Qun Li†, Guihai Chen*§
*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210024, China
†Department of Computer Science, the College of William and Mary, Williamsburg, VA 23185, USA
§Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University, Shanghai 200240, China
Email: gxjzhu@gmail.com, liqun@cs.wm.edu, gchen@nju.edu.cn

***Abstract*—This paper presents APT, a localization system for outdoor pedestrians with smartphones. APT performs better than the built-in GPS module of the smartphone in terms of accuracy. This is achieved by introducing a robust dead reckoning algorithm and an error-tolerant algorithm for map matching. When the user is walking with the smartphone, the dead reckoning algorithm monitors steps and walking direction in real time. It then reports new steps and turns to the map-matching algorithm. Based on updated information, this algorithm adjusts the user's location on a map in an error-tolerant manner. If location ambiguity among several routes occurs after adjustments, the GPS module is queried to help eliminate this ambiguity. Evaluations in practice show that the error of our system is less than 1/2 that of GPS.**

I. Introduction

We have noticed several occurrences of blind people losing their way in our university. It is disconcerting to see those visually impaired people disoriented on campus even though the routes are comparably familiar and thus simple to follow. We realize that the fundamental reason why blind people lose their way is that they do not know their current location and easily miss a turn or go toward a wrong direction. We aim to build a system to assist the blind people with smartphones by providing accurate location information and, combined with a digital map, offering guidance audio messages. Compared with blind navigation systems such as RFID based technology, it is our belief that this system can be rapidly and inexpensively deployed. Toward this goal, this paper solves the localization problem by designing an accurate outdoor pedestrian tracking system.

GPS, the most accurate localization service for smartphones at present, has inherent limitations in its accuracy. Our GPS measurements show error up to 15 meters in a clear-sky-view environment and up to 20 meters when trees are around. Such accuracy restricts the development of our system.

Our goal is to improve localization accuracy to a few meters. This touches the fundamental limitation of GPS and enables accuracy-demanding applications. To improve the accuracy of the GPS module on smartphones, we tried several approaches including taking the average for consecutive GPS readings and consulting GPS location history data. Unfortunately, the improvement is negligible. The underlying reason is that GPS error does not have a useful pattern. It can be arbitrary from time to time and from location to location.

The work was done when the first author was visiting the College of William and Mary.

To realize our goal, we base our solution on three observations. First, pedestrians have regular movements that provide opportunities for an accurate dead reckoning system. Second, though the accuracy of GPS is unsatisfactory, it works well in distinguishing between distant routes. Third, we can easily generate augmented maps on a smartphone. Based on these observations, we design a dead reckoning algorithm and a map-matching algorithm. The dead reckoning algorithm detects steps and turns, and triggers our map-matching algorithm. When map-matching fails to uniquely determine a route, it requests location information from the GPS component. The main purpose of GPS is to help distinguish between distant routes, thus GPS is requested infrequently. Consequently, the increased energy efficiency is another advantage of our system (the accelerometer and gyroscope are known to consume much less energy than GPS [1]). However, there are various challenges in implementing the ideas above, such as different phone placements, phone axes reorientation and error-tolerant map-matching. Tackling these challenges constitutes our contributions, which can be enumerated as follows.

First, we propose a robust dead reckoning algorithm (Section V). This algorithm features two designs. First, instead of detecting walking steps as usual, we manage to find acceleration patterns that can reflect travel distance and be easily detected. This design simplifies the detection problem, and the resulting algorithm can also tolerate different placement of the phone. Second, we do not require the user to hold the phone flat out when using the system. Usually, this requirement is for walking direction estimation. We relax this requirement by re-orienting the phone's axes robustly so that different ways to hold the phone yield the same direction estimation.

Second, we propose an error-tolerant map-matching algorithm (Section VI). The novelty is that it tolerates possible errors of the dead reckoning algorithm and GPS. This tolerance is achieved by trying all possible routes that are within our error tolerance threshold. In some cases, this scheme finds several possible routes and cannot determine which one is correct. Then, GPS can be used to help eliminate this ambiguity.

Third, we evaluate our approach using real-world measurements conducted around the campus (Section VII). The results show that our approach can achieve a localization accuracy within 5 meters, while GPS-based solutions have error up to 15 meters.

The rest of this paper is organized as follows. We review related works in Section II, and report GPS observations

in Section III. Section IV overviews our solution, followed by detailed description of the dead reckoning algorithm in Section V and the map matching algorithm in Section VI. We conduct evaluation in Section VII, discuss related issues in Section VIII and conclude our paper in Section IX.

II. Related Works

Many indoor localization systems can achieve localization accuracy to within a few meters. Radar [2] is a classic indoor localization system using wireless signal fingerprinting to localize a user. It requires a training period that incurs labor-intensive data collection. Recently, researchers try to eliminate this training period by solving a system of equations to find the wireless propagation model [3]. SparseTrack [4] is an indoor pedestrian tracking system, where a user carries a smartphone and a special sensor mote (Cricket mote). The smartphone is for dead reckoning where the accelerometer and the digital compass are for distance estimation and direction estimation respectively. The error accumulation problem of the dead reckoning system is tackled by the special sensor mote, which has ultrasound ranging capability and can communicate with other sparsely deployed special sensor motes in the infrastructure. Our dead reckoning system differs in the type of sensors used and in the techniques. Additionally, the error accumulation problem in our system is solved by the map-matching component. Localization is also an important research problem in wireless sensor networks [5]–[8].

Smartphones have become pervasive in the past several years. Among the numerous applications for smartphones, many require position information [9]–[11]. For these applications, GPS is the most popular choice, but it is power-hungry. Much research effort has been devoted to improving the energy efficiency issue of GPS [12]–[18]. One common approach is to substitute power-hungry GPS with energy-efficient localization schemes such as WiFi and cellular-tower. Following this idea, EnTracked [14] focuses on position tracking, and EnTracked$_T$ [16] focuses on trajectory tracking. A-Loc [12] tries to track positions for mobile search applications with dynamic accuracy. RAPS [15] turns on GPS in a rate-adaptive manner. CAPS [13] is a cell-ID based positioning system that leverages position history for localization. If multiple applications request location service simultaneously, then a unifying middleware layer can help reduce the use of GPS [17]. Another idea for saving energy is to modify GPS in hardware. LEAP [18] shifts the position calculating step of the GPS hardware to the cloud, providing energy efficient trajectory service. Other localization methods are also proposed for smartphones. EV-Loc [19] integrates electronic and visual signal for accurate localization and tracking. E-Shadow [20] has a localization component using direction information.

Several works consider off-line localization. AutoWitness [21] is a system for tracking stolen personal property. In the system, a special tag is embedded inside an asset to be protected. The tag contains an accelerometer and a gyroscope for dead reckoning, and a GSM/GPRS model for reporting data to a server. The server performs map-matching to recover the traveled route. We differ in both the dead reckoning system and the map-matching algorithm. First, our dead reckoning system is designed specifically for pedestrians, and theirs is for cars. The solutions are not applicable to each other due to different movement properties. Second, we focus on online tracking, while the off-line map-matching algorithm in AutoWitness cannot be easily extended to this scenario. Another work considers utilizing temporal stability and low-rank structure for improving localization accuracy [22].

III. Error characteristics of smartphone GPS

We consider improving GPS accuracy based solely on GPS. This requires the knowledge of GPS error characteristics. For this purpose, we conducted GPS measurements around Sunken Garden, Williamsburg, VA (N37.270851°,W76.711682°). The smartphone we use is the Samsung Galaxy S II on the android platform. It is programmed to collect GPS data once a second. Our first encountered issue is that at any location, if the GPS coordinate stabilizes, then it will never change, or will not change for at least several hours. Experiments with different phones, locations and durations all lead to the same result. One possible explanation is that this is a strategy of the GPS hardware and/or Android system for saving energy. But this phenomenon implies that *staying in one place longer does not help improve GPS accuracy*. Thus, the natural and seemingly possible solution, staying in a place and taking the average of all GPS readings, will not work well in practice. Finally, we resort to the following strategy for data collection. Every time the readings stabilize, we change the phone's location briefly. Among the measurements we conducted, we report two typical ones.

The first consists of data collected at locations shown in Figure 1. The data are organized into tuples. Each tuple consists of three stabilized GPS readings with each at a different location. In December 2011, we collected 7 datasets at different times with each dataset containing 10-20 tuples. We find that the tuples are different from each other, even for consecutive ones. We plot all the tuples in Figure 2 differentiating datasets by color. We can see that GPS readings at the same location can differ up to 15 meters. It is worth noting that all three locations have a clear view of the sky. Further study shows that it is hard to find any obvious temporal/spatial correlation among tuples. Consequently, we believe that in our scenario, historical GPS data or neighborhood GPS data do not help much in improving GPS accuracy.

The second measurement is for testing pedestrian tracking. Note that continuously updating GPS is currently the most accurate methodology for smartphone-based tracking. We selected a route around Sunken Garden, and a large portion of this route is covered by trees. A user holding the smartphone walks along the route five times. The GPS coordinates are recorded every second. Figure 3 shows the five walks differentiated by color. We can see that the accuracy of GPS is worse than that of the previous scenario. Even after removing obvious outliers, the error can still be more than 20 meters. More importantly, there is no obvious error pattern in different repetitions. (If we can find a useful pattern, we may use it to improve accuracy.)

As a result, we find that it is unlikely to improve localization accuracy based solely on GPS. To achieve our goal, other information is required. For this reason, we design a dead reckoning system to refine the user's location, as well as a map matching system to map a user's location onto a map.

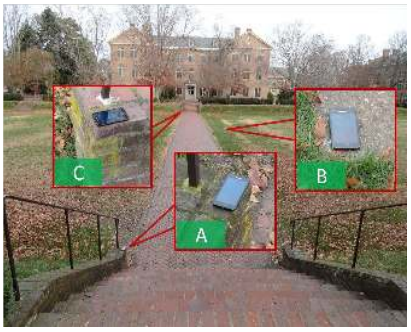


Fig. 1. Three locations in Sunken Garden, Williamsburg, VA. Each location has a clear view of the sky.

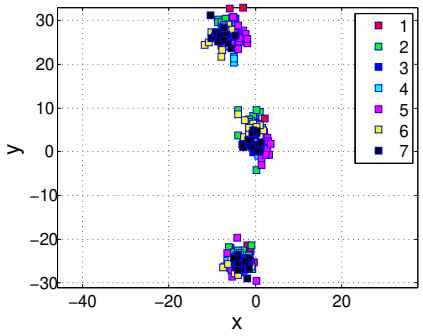


Fig. 2. GPS coordinates (in meters) for the three locations. From bottom to top, the three clusters correspond to locations A, B and C.

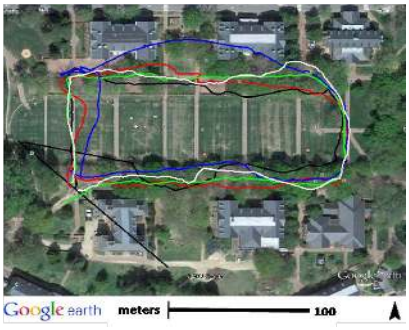


Fig. 3. GPS coordinates for five walks distinguished by color. A large portion of the route is covered by trees.

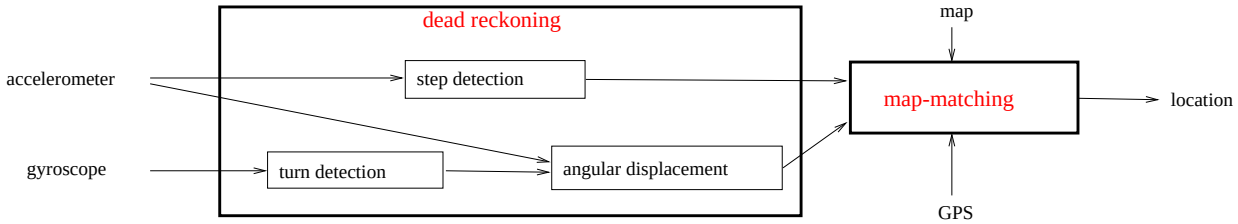


Fig. 4. Flowchart of our system.

GPS is still required, but its use is limited to help reduce route ambiguity.

IV. Our solution overview

In this section, we briefly introduce our system. In subsequent sections, we will motivate each component individually, followed by detailed descriptions.

Our system can be organized into the framework in Figure 4. The system consists of two components, a dead reckoning component and a map-matching component. It takes as input four pieces of information: accelerometer, gyroscope, map and GPS, and outputs user's current location.

When the user walks, both the accelerometer and the gyroscope of the phone keep sampling and sending the samples to the dead reckoning algorithm. The dead reckoning algorithm processes the data in real time. On one hand, it monitors the acceleration data to detect a step and reports new steps to the map-matching component. On the other hand, it monitors gyroscope data for possible turns. If a turn is detected, then the dead reckoning algorithm uses the acceleration data to reorient the phone's axes and compute the angular displacement of the turn. It then reports the angular displacement to the map-matching component. An example of the reported angular displacement is "turn, $80°$, left".

The map-matching component combines all information to produce the final location of the user. Using map information, it maintains all possible routes of the user, and updates the user's location once a new step or a new turn is reported by the dead reckoning algorithm. Whenever a unique location of the user is required but the system cannot tell between several routes, then the GPS component is queried to eliminate ambiguity.

As we can see, among the four pieces of information, the map information can be queried once and used many times and the GPS is queried only when the system cannot tell which route is correct. The other two pieces of information need frequently updating, but it is well known that the accelerometer and gyroscope consume much less energy than GPS. Therefore, our solution is also energy efficient.

V. Dead reckoning

There are two parts of our dead reckoning system: travel distance estimation and direction estimation. The challenge in this section is to deal with complex pedestrian movements and tolerate the different ways pedestrians carry their phones. In our system, the user does not need to hold the phone flat out in front of them.

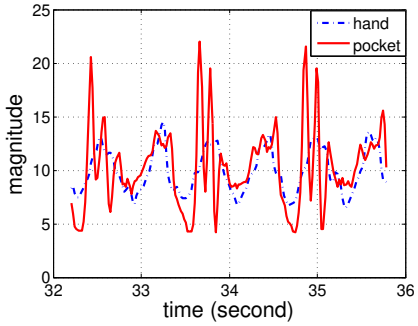
A. Estimating distance

For pedestrians, estimating travel distance by taking the double integral of acceleration results in large error due to complex human movement. Instead, a common approach is to count the number of walking steps and then multiply it by the stride length (e.g., [4]). This can be explained by the following simple formula

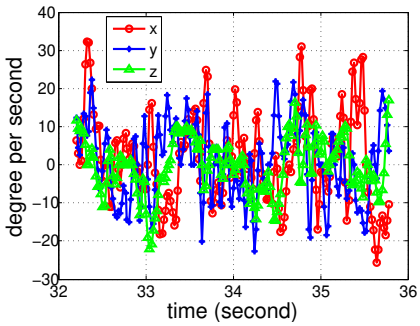
$$d = n_s \cdot l_s \tag{1}$$

where d is the distance traveled, n_s is the number of walking steps and l_s is the stride length trained beforehand. To this end, step detection is critical. This can be done by observing and identifying acceleration patterns during a typical walking step [23]. There are quite a few step-counter apps in Google Play (previous Android Market).

Unfortunately, we find that different placement of the phone has a large impact on the accuracy of each step counter.



(a) Acceleration magnitude



(b) Gyroscope readings

Fig. 5. A 6-step walk along a straight line. The readings in (b) is from the phone in hand.

To illustrate, we carry out an experiment where a user holds one phone in front and carries another phone in his pants' pocket with both phones recording acceleration data. Both phones are synchronized manually by starting the program simultaneously. We show in Figure 5(a) the acceleration magnitudes (i.e., $\sqrt{x^2+y^2+z^2}$ where x, y, z are accelerometer readings in three dimensions) of the two phones over a 6-step duration. The difference in the patterns is evident. Having the phone in hand produces desired data, which contain 6 recurring patterns; while keeping the phone in the pocket produces only 3 recurring patterns. This 3-pattern phenomenon also arises if the pedestrian swings the phone in their hand.

This is caused by the nature of walking. While walking, there is always one foot touching the ground. For the pocket phone, left foot touching the ground is different from right foot touching the ground. Thus, each pattern in this case actually corresponds to two steps of pedestrian. The case where the phone swings in the user's hand is similar. It seems that we need to differentiate between different placement of the phone for accurate distance estimation, which is complicated.

Fortunately, no matter how the phone is placed, we find that acceleration always shows some recurring patterns. Those patterns only differ in the number of human steps one pattern corresponds to. For convenience, we refer to a pattern as a period. Thus, we can generalize (1) to

$$d = n_p \cdot l_p$$

where d is the travel distance, n_p is he number of periods and l_p is the travel distance within a period. Here, a period does not necessarily correspond to one human step—it may correspond to two steps, or only a half. As long as the number of periods is proportional to the travel distance, we can define any pattern as a "period" and count its number. Relaxing "human step" to "period" significantly simplifies our problem. The travel distance within a period can be trained, as before, or can be estimated by the method in Section VIII. For ease of presentation, we still refer to the newly defined period as a step. Thus, we refer to detecting a period as step detection, and travel distance within a period as step length.

Following this idea, we define an up-down pattern as a step, which is captured by the following level crossing algorithm. Suppose the series of acceleration magnitudes up to now are $x_1, x_2, \ldots, x_n$ where x_n is the most recent. We map reading x_n to a bit according to

$$Q(x_n) = \begin{cases} 1 & \text{if } x_n > \mu_n + \sigma_n \\ 0 & \text{if } x_n < \mu_n - \sigma_n \\ \wedge & \text{otherwise} \end{cases}$$

where μ_n is the average of the series, σ_n is the corresponding standard deviation, and $\wedge$ is an undefined state. The two thresholds $\mu_n + \sigma_n$ and $\mu_n - \sigma_n$ are two levels for characterizing "up" and "down" respectively. Both μ_n and σ_n can be updated incrementally based on new incoming acceleration magnitudes. This mapping yields a sequence of bits. Then we merge consecutive 1s into a single bit 1, 0s to 0, and $\wedge$s into $\wedge$. A pattern "10" or "$1 \wedge 0$" is defined as a step. Whenever a step is detected, dead reckoning system will report it to the map-matching component. Note that the two thresholds $\mu_n + \sigma_n$ and $\mu_n - \sigma_n$ can be adjusted for better performance. This simple algorithm also appears in the secret key extraction literature [24].

It is worth mentioning that using acceleration magnitude, instead of acceleration in a certain direction, can tolerate different ways pedestrians carry the phone, because the acceleration magnitude is the same no matter which direction the phone is pointing.

B. Estimating direction

This task is more challenging. Previously, the magnetic digital compass was the only choice for determining direction for smartphones. However, it is easily influenced by the environment (simply putting two phones together will influence the compasses). Recently, gyroscopes are equipped in more and more smartphones, e.g, iPhone 4S, Google Galaxy Nexus, HTC EVO 3D, etc. Though its main purpose is for games, we will use it for direction estimation.

Different ways to hold the phone put a challenge on direction estimation. On one hand, the gyroscope data are with respect to the Cartesian frame of reference of the phone itself, which is represented by the orthogonal xyz axes with the x-axis pointing to the right side of the phone, the y-axis pointing to the top of the phone and the z-axis leaving the screen. On the other hand, the Cartesian frame of reference we need should be the XYZ axes system with the X-axis pointing to right-hand side of the user, the Y-axis pointing to the direction the user is facing and the Z-axis pointing to the sky. (We intentionally use capitalized XYZ to distinguish between the two references.) To allow different ways to hold the phone, the two references can be different, as shown in Figure 6. In fact,

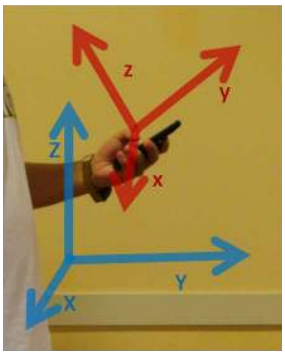


Fig. 6. Usual way of holding a phone and the two Cartesian references.

the two references are different in most cases. We can obtain xyz angular velocity data, but we need Z-axis data.

This problem also arises in the vehicular context [21], [25], where the solution is to find the rotation matrix between the two references by acceleration measurement. This may work for cars in a state of uniform motion in a straight line and for static human, but not for pedestrians. Any sinlge rotation matrix is unable to characterize all the rotations during walking, because the rotation keeps changing in a single human step. We show in Figure 5(b) the gyroscope readings during the 6-step experiment. The angular velocity varies over time.

We resort to average rotation as an approximation. Our method is based on two observations. First, the angular velocity at each axis follows a recurring pattern. Therefore, integrating them along time may reduce the fluctuations. Second, when a pedestrian walks in a straight line, the average acceleration in any axis does not fluctuate much. To illustrate the two observations, we perform another experiment where the user walks in a straight line, makes a 90° left turn, and then walks in a straight line again. We plot in Figure 7(a) the angular displacement in each axis by integrating gyroscope readings along time. Observe that the displacement around any axis remains roughly the same before/after the turn. Figure 7(b) shows the sliding-window-smoothed acceleration data. The acceleration does not fluctuate much before/after the turn, but is quite unusual during the turn. Thus, if we can determine when a turn is occurring and exclude the data during the turn, we can approximately treat the phone as in static case by using the average acceleration and integrated angular displacement.

Following these observations, our method works as follows. First, we integrate gyroscope readings around the phone's xyz axes similar to how we produce Figure 7(a). Second, we monitor angular displacement around all three axes to determine straight walking. Within a time window, if all three angular displacements do not exceed a pre-determined threshold, then we believe the user walked along a straight line. Third, for the duration when the user walks in a straight line, we average the acceleration readings in each direction. Suppose the obtained averages are μ_x, μ_y, μ_z respectively. Then, the adjusted angular displacement around Z-axis for an incoming turn is calculated as $(\alpha\mu_x + \beta\mu_y + \gamma\mu_z)/\sqrt{\mu_x^2 + \mu_y^2 + \mu_z^2}$ where α, β, γ are the angular displacements computed in the first step. The rationale behind this formula is that the average acceleration during a straight walk should approximate gravity. Therefore, we can infer how a Z-axis vector (the gravity) is decomposed into three components: μ_x in x-axis, μ_y in y-axis and μ_z in z-axis. Reversing this process back properly on angular measurements will recover the angular velocity and angular displacement around the Z-axis. Using this procedure, we recover the Z-axis angular displacement in Figure 7(c). The result is promising. The angular displacement is 91.56° in this case. It is worth emphasizing that error is inevitable. Even though all sensors report perfect measurements, it is nearly impossible to find the exact rotation due to complex human movement as mentioned before. This is in contrast to the vehicular scenario.

There are two issues left. The first is the threshold for detecting a turn. We find from experiments that the angular displacement in a straight line walk can change up to $\pm 10^\circ$. Therefore, the detection threshold cannot be less than 20°. We set the threshold to be 30° in our experiments. Note that this threshold is purely for detecting turns, and is for angular displacement around the phone's axes. This setting results in the second issue, i.e., we may not be able to detect shallow turns, and the reported angular displacements contain non-negligible errors. This issue poses a challenge on the map-matching algorithm, which needs to tolerate such errors.

VI. Map Matching

After the dead reckoning algorithm reports new steps and new turns, the map-matching component will incorporate this information to refine the user's current location. This section describes how this is done. We will first consider the scenario with perfect information, then adjust the process to tolerate various errors. There are several existing works on mapping GPS coordinates to a map (e.g., [26]), but the technique is not directly applicable to our scenario.

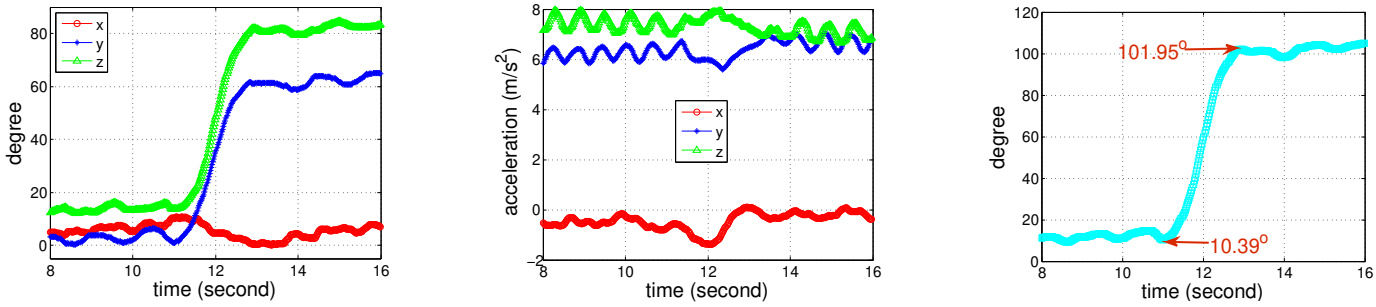
We give several definitions. Since our target is to match a walk with a map, we define three terms: map, walk and map matching.

Definition 1 (map)**.** *A map $\mathcal{M}$ is a subset of points in 2-D Euclidean space. It is characterized by a tuple (V, L, E) where V is a set of vertices, $L : V \longrightarrow \mathbb{R}^2$ describes vertex locations, and $E \subseteq V^2$ is the set of edges characterizing permissible straight-line paths between vertices. A point is in the map if and only if the point is one of the vertices or lies on an edge.*

It is worth mentioning that we do not include "curves" in our definition, though the inclusion is possible by extending the definition of "tuple". There are two reasons why we did not include this. First, it is hard to characterize the map if arbitrary curves are allowed. Second, most curves can be approximated quite well by several straight lines. Note that our definition can be directly extended to 3-D scenarios.

Definition 2 (walk)**.** *A walk $\mathcal{W}$ is a sequence of 2-D points and the associated timestamps. It is represented by $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ with timestamps $t_1, t_2, \ldots, t_n$.*

Though the output of the dead reckoning system is in terms of steps and turns, it is straightforward to rewrite them in the above form. Sometimes we omit the timestamps, but they are implicit. This definition is discrete for practical concerns. Though a walk should be continuous in nature, any dead reckoning system can only report locations at discrete



(a) Angular displacements around xyz axes (b) Sliding window smoothing for acceleration (c) Resulting Z-axis angular displacement

Fig. 7. A 90°-left-turn experiment. The window size in (b) is the number of readings within 1 second.

times. Locations at any time in between are interpolated. Additionally, direction information is implicit in the sequence due to timestamps.

Definition 3 (matching)**.** *A matching of a walk $\mathcal{W}$ to a map $\mathcal{M}$ is a function f that maps every points of $\mathcal{W}$ to a point in $\mathcal{M}$. A matching is an exact matching if the mapped walk $f(\mathcal{W})$ can be obtained from $\mathcal{W}$ by a combination of the following transformations: (1) translation, (2) rotation around any line perpendicular to the 2-D surface of the map. A matching is correct if the mapped walk is equal to the ground-truth.*

The requirement on rotation axis is because of the direction information in the walk. Our dead reckoning system can tell the diference between left turn and right turn. This information is kept by putting the restriction on rotation axis.

A. Perfect information

We consider perfect information in this subsection. We assume dead reckoning involves no error, i.e., the distance estimation and direction estimation are perfect and the map is perfect. Unfortunately, even under this assumption, we may still not be able to match a walk correctly.

The cause is ambiguity. It is possible to find an exact matching, but multiple may exist. One simple example is that the walk is short and is along a straight line. Many edges of the map can lead to exact matching. Additionally, the computational overhead can be intolerable in the worst-case where every edge corresponds to one or several exact matching(s). To this end, we introduce *position fix*, which is realized by GPS receivers.

Definition 4 (position fix)**.** *A position fix P is a pair of points (p_1, p_2) where $p_1 \in \mathcal{W}$ and $p_2 \in \mathcal{M}$. A position fix is satisfied if the user's position p_1 is matched with the point p_2 in the map.*

The intuition of a position fix is to "pin" the user's position to the map. In practice, this can be done using the GPS component of the phone. Note that we again assume perfect information, i.e., the position fix is perfect without error. One natural question arises, how many position fixes are required for eliminating ambiguity? We find that two are enough. The underlying reason is the direction information in the walk, which is reflected in the rotation axis requirement. If we cannot tell a left turn from a right turn, then any finitely many position fixes cannot guarantee ambiguity elimination.

Theorem 1. *Given two position fixes $P' = (p'_1, p'_2)$ and $P'' = (p''_1, p''_2)$ where $p'_1 \neq p''_1$ and $p'_2 \neq p''_2$, there exists at most one exact matching from walk $\mathcal{W}$ to map $\mathcal{M}$ that satisfies both P' and P''.*

Proof: Recall that only two operations are allowed for obtaining an exact matching. After satisfying P', we cannot perform the translation operation any more, thus we may only carry out rotation around the line that is through p'_2 and is perpendicular to the map. In this case, there is at most one angular displacement for satisfying P'', which completes the proof. ■

We consider how to find the exact matching given two position fixes. There is a straightforward solution that finds the transformation matrix by analytically solving a system of equations according to the two position fixes. However, this solution cannot be easily extended to cope with errors. Instead, we propose an adaptive algorithm that is easily extended.

The basic idea is trial and error. Starting from one position fix, we find out all possible routes. Then we use subsequent points in the walk to test and extend these routes. We rule out a route from consideration if either case happens: (1) the route fails to map some point of the walk, e.g., the walk indicates a turn but there is no corresponding turn in the map by following the route; (2) we encounter another position fix which cannot be satisfied by the route. Such a rule will remove all routes except the correct one due to Theorem 1.

This idea is described in Algorithm 1. We first map subsequent (with respect to the starting point) walk points, then reverse the routes and map all previous walk points. We clarify several issues. First, a *route* not only records the previous matching information as history, but also the current route segment (edge) and the user's walking direction in this route segment. Following this, during initialization, each map edge containing p'_2 will introduce one route if p'_2 is an end point, or two routes if p'_2 lies on the edge. Second, to refine routes, we need to add a walk point p to a route t (Line 3 in Algorithm 2). This is done by checking whether the mapped position is on the map. If it is on the map, then the process reports a success status, adds p to the route, and then updates current route segment and direction. Otherwise, the process reports a failure status and the corresponding route will be removed from consideration. It is worth emphasizing that though the algorithm is written in an off-line manner, it is easy to rewrite it in an online form.

Algorithm 1: Map matching with perfect information

Input: map $\mathcal{M}$; walk $\mathcal{W}$; two position fixes P', P''
Output: $f : \mathcal{W} \longrightarrow \mathcal{M}$, a matching

1 **begin**
2 $start \longleftarrow$ the index of p'_1 in $\mathcal{W}$;
3 $\mathcal{T} \longleftarrow$ find all possible routes through p'_2;
4 $i \longleftarrow start + 1$;
5 **while** $i \leq n$ **do**
6 $p \longleftarrow (x_i, y_i)$;
7 $refine\_routes(\mathcal{T}, p)$;
8 $i \longleftarrow i + 1$;
9 reverse the direction of all routes in $\mathcal{T}$;
10 $i \longleftarrow start - 1$;
11 **while** $i \geq 1$ **do**
12 $p \longleftarrow (x_i, y_i)$;
13 $refine\_routes(\mathcal{T}, p)$;
14 $i \longleftarrow i - 1$;
15 $f \longleftarrow$ pick one route from $\mathcal{T}$;
16 **end**

Algorithm 2: Subroutine for Algorithm 1: refine_routes

Input: $\mathcal{T}$; p;

1 **begin**
2 **foreach** $t \in \mathcal{T}$ **do**
3 $status \longleftarrow$ add p to t;
4 **if** $status = SUCCESS$ **then**
5 **if** $p \in P''$ **then**
6 delete all other tacks in $\mathcal{T}$, return;
7
8 **else**
9 delete t from $\mathcal{T}$;
10 **end**

B. Dealing with errors

In practice, various errors, including distance estimation error, direction estimation error, and GPS error, are inevitable, which requires adjustment of Algorithm 1. To introduce our error tolerance mechanism, we reformulate the representation of a walk. Previously, a walk was formulated as a set of data points with timestamps. Here we formulate it as a series of incremental changes, where a change can be a step or a turn.

The basic idea is to threshold the possible errors and consider all possible routes. A route is considered as a possible route if following this route does not yield errors exceeding our error thresholds. Our algorithm updates the set of possible routes whenever new information is available. The new information can be a new step, a new turn, or a GPS update. We require GPS information because, as mentioned before, GPS is necessary even if dead reckoning is perfect. Next, we will describe how we maintain the set of possible routes when different new information is available. The following operations can be incorporated to Algorithm 1 with the first operation corresponding to Line 3 of Algorithm 1 and the rest implementing the sub-routine $refine\_routes$.

1) Initial routes: At the beginning of our algorithm, we request a GPS coordinate and initialize the set of possible routes. At this stage, we do not know the walking direction of the user, thus we try all possible directions. We enumerate all possible locations of the user on the map by considering GPS error. Then we enumerate all possible routes through the possible locations. Here, each route consists of one "current position" and one "current route segment". The current route segment and the walking direction on it are represented by its two end points, the "from" end point and the "to" end point. To cope with GPS error, we set the distance error tolerance of each route as the error of GPS. This error will be set to 0 after we successfully match a new turn in the route.

2) A new step: When the dead reckoning algorithm detects a new step, we try to add one step to all possible routes. This is done by adding a step length offset to the "current position" of each possible route towards the "to" end point. For any possible route, if the new position is still on the route's current route segment, then this route remains in the set of possible routes. Otherwise, we search over all adjacent route segments (indicated by the neighbors of the "to" end point) to find possible ones. An adjacent route segment is possible if walking to it only requires a shallow turn within angular error tolerance. (Maybe our dead reckoning fails to detect this shallow turn.) For each possible route segment, we replicate a new route, extend it by this possible route segment, and add the resulting route to the set of possible routes. If no possible route segments are found, then we still keep this route in the set of possible routes but decrease its error tolerance by the step length. If its error tolerance is decreased to 0, then we remove it from the set of possible routes.

3) A new turn: After a new turn is available, we add this turn to all possible routes. For each possible route, we check whether its "current position" is within a certain distance (distance error tolerance) to any end point of the "current route segment". This is because, possible turns may exist at these end points. If the distance to both end points is beyond the distance error tolerance, then this route is removed from the set of possible routes. If the distance to either end point is within the distance error tolerance, then we check all this end point's neighbors to find out all route segments that are reachable by a turn within the range: the reported angular displacement plus/minus angular error tolerance. For each satisfying route segment, we make a duplicate of the route, extend it by the satisfying route segment, and add it to the set of possible routes. On the other hand, if no route segment satisfies, then the route will be removed from the set of possible routes.

4) A new GPS coordinate: When a new GPS coordinate is available, we check each possible route by verifying whether the new GPS coordinate is within a certain distance (distance error tolerance plus GPS error) to the "current position" of the route. If not, then the route will be removed from the set of possible routes.

If no possible route exists, the system will restart by requesting a new GPS coordinate. Two cases need further consideration. The first is when a step and a turn arrive simultaneously. This actually happens in practice when the user makes a turn slowly in several steps. We solve this problem by ignoring the steps during a turn, because we find that the displacement during a turn is very short and ignoring it incurs only small distance errors. The second is when the number of possible routes becomes intolerable due to our error tolerance

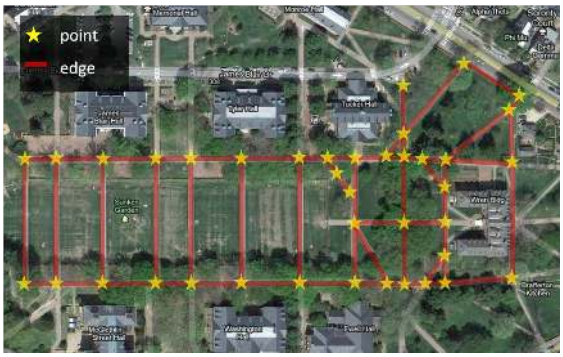


Fig. 8. Evaluation map. This area is around Sunken Garden, Williamsburg, VA. The stars are vertices and the route segments connecting them are edges.

mechanism. In this case, we request a GPS coordinate, which will reduce the number of possible routes.

VII. Evaluation

We conduct our experiments in the area shown in Figure 8. The map consists of anchor points as vertices and route segments as edges. The ground-truth coordinates of the anchor points are manually found from GoogleMap. The ground-truth should not be obtained from the GPS receiver due to the GPS error mentioned before. All vertices and edges are input information for any algorithm using map information.

We compare our algorithm to two GPS-based solutions. The first is the raw GPS coordinates reported by the GPS component of the smartphone. The second is to combine the raw GPS coordinates with the map information. This combination is done by mapping the raw GPS coordinates to the nearest point on the map.

Three routes are selected for our experiments, as shown in Figure 9. For each route, a user walks along the route with a smartphone held out in front as in Figure 6. In each second, the phone records three pieces of information: 50 accelerometer readings, 50 gyroscope readings, and one GPS reading. All readings are recorded with timestamps. These data are retrieved later from the phone for evaluation. To establish the ground-truth locations during the walk, whenever an anchor point is encountered, the user presses a button in our program, which records the instant timestamp into a file. In this way, we have the ground-truth $(time, location)$ pair. Later, we query each localization method for the user's location at that time, and compare the returned location with the ground-truth.

The parameters of our algorithm are set as follows. The threshold for detecting a turn is $30°$, since the angular displacement during a single step can be up to $20°$. The tolerance for angular error is also $30°$. The tolerance for distance error is $20m$. These settings are based on experience and have not been optimized. Our algorithm uses one GPS coordinate as a position fix during initialization. Route ambiguity is eliminated automatically at the end of each walk.

Figure 10 shows the result. Though the three routes are selected from simple to complicated with route 1 being the simplest and route 3 being the most complicated, it turns out that the error of our algorithm does not necessarily increase. On the contrary, to some extent, the error decreases. At quite a few anchor points, the error of our algorithm is close to 0. This surprising accuracy comes from the turn information. If there is a turn during the route and our algorithm successfully detects it, then the user's location is adjusted to that turning point, resulting in 0 error. The most complicated route, route 3, contains more turns, and the error is 0 at most anchor points. The error at non-turn anchor points is at most $5m$. This occurs at anchor point 9 in route 1. In all three routes, our algorithm have consistently less error than GPS-based solutions. GPS+Map performs slightly better than pure GPS, while both have unstable error of about 15 meters. The GPS-based solutions have different error among three routes. We suspect that this is caused by time differences and weather, since we have observed quite large error (up to 30 meters) in cloudy weather in roughly the same place.

VIII. Discussion and Future Work

At the first stage of this project, we evaluated a method similar to Radar [2] due to the reported meter-level accuracy of Radar. This is done by collecting WiFi signal information at different locations as training data and later distinguishing locations by WiFi information. Though the number of APs in the experimental area is promising, around ten, it turns out that the localization accuracy is much worse than GPS. This may be caused by the fact that outdoor environments do not have much multi-path effect so that a location can no longer be fingerprinted well. For our current solution, we also tried to use more GPS updates to improve the accuracy. This does not help if we already know the current route segment.

In our current implementation, the step length is obtained by training. This training period can be avoided by using GPS and map information. We may query GPS to determine a traveled route segment on the map and then find the step length via dividing the route length by the number of steps. The drawback is that we are not able to track the user accurately during this period. Note that finding step length by computing the distance between two GPS coordinates yields unsatisfactory step length accuracy. Therefore, we propose to use GPS to determine route segment only, and then use map information to find step length.

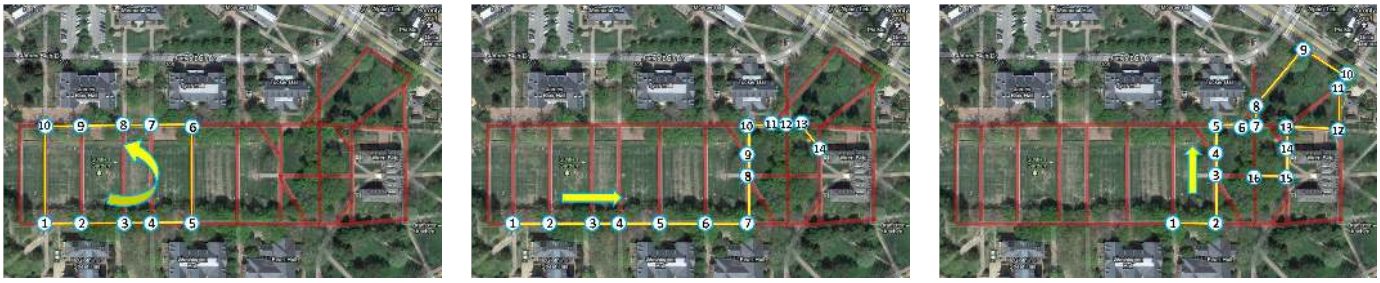
In the future, we plan to implement our solution in different platforms and evaluate the system in complicated routes and environments.

IX. Conclusion

In this paper, we present APT, a system targeting at accurate pedestrian localization. It uses the accelerometer, gyroscope and GPS component of modern smartphones, and integrates them with external map information. The system can tolerate GPS error and the different ways to hold the phone. Measurements from real-world show that its accuracy is significantly higher than GPS-based solutions.

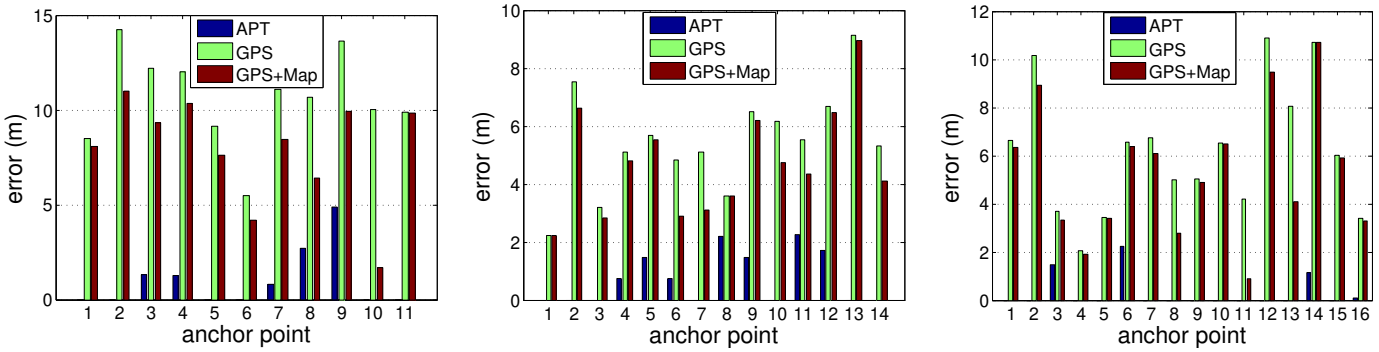
Acknowledgment

The authors would like to thank Jacqulyn Johnson for suggesting this problem, and all the reviewers for their helpful comments. This project was supported in part by US National Science Foundation grants CNS-1117412, CAREER Award CNS-0747108, China NSF grants (61073152, 61133006) and China 973 project (2012CB316200).



(a) Route 1 (b) Route 2 (c) Route 3

Fig. 9. Ground-truth routes.



(a) Route 1 (b) Route 2 (c) Route 3

Fig. 10. Localization errors at anchor points in the route.

REFERENCES

[1] F. B. Abdesslem, A. Phillips, and T. Henderson, "Less is more: energy-efficient mobile sensing with senseless," in *MobiHeld*, 2009.

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

[3] K. Chintalapudi, A. Padmanabha Iyer, and V. N. Padmanabhan, "Indoor localization without the pain," in *MobiCom*, 2010.

[4] Y. Jin, M. Motani, W.-S. Soh, and J. Zhang, "Sparsetrack: Enhancing indoor pedestrian tracking with sparse infrastructure support," in *INFOCOM*, 2010.

[5] B. Zhang, X. Cheng, N. Zhang, Y. Cui, Y. Li, and Q. Liang, "Sparse target counting and localization in sensor networks based on compressive sensing," in *INFOCOM*, 2011.

[6] M. Ding and X. Cheng, "Fault tolerant target tracking in sensor networks," in *MobiHoc*, 2009.

[7] X. Zhu, X. Wu, and G. Chen, "Refining hop-count for localisation in wireless sensor networks," *International Journal of Sensor Networks*, vol. 12, no. 4, 2012.

[8] X. Zhu and G. Chen, "Spatial ordering derivation for one-dimensional wireless sensor networks," in *ISPA*, 2011.

[9] S. Gaonkar, J. Li, R. R. Choudhury, L. Cox, and A. Schmidt, "Micro-blog: sharing and querying content through mobile phones and social participation," in *MobiSys*, 2008.

[10] J. Froehlich, M. Y. Chen, S. Consolvo, B. Harrison, and J. A. Landay, "Myexperience: a system for in situ tracing and capturing of user feedback on mobile phones," in *MobiSys*, 2007.

[11] Foursquare. [Online]. Available: http://www.foursquare.com

[12] K. Lin, A. Kansal, D. Lymberopoulos, and F. Zhao, "Energy-accuracy trade-off for continuous mobile device location," in *MobiSys*, 2010.

[13] J. Paek, K.-H. Kim, J. P. Singh, and R. Govindan, "Energy-efficient positioning for smartphones using cell-id sequence matching," in *MobiSys*, 2011.

[14] M. B. Kjaergaard, J. Langdal, T. Godsk, and T. Toftkjaer, "Entracked: energy-efficient robust position tracking for mobile devices," in *MobiSys*, 2009.

[15] J. Paek, J. Kim, and R. Govindan, "Energy-efficient rate-adaptive gps-based positioning for smartphones," in *MobiSys*, 2010.

[16] M. B. Kjaergaard, S. Bhattacharya, H. Blunck, and P. Nurmi, "Energy-efficient trajectory tracking for mobile devices," in *MobiSys*, 2011.

[17] Z. Zhuang, K.-H. Kim, and J. P. Singh, "Improving energy efficiency of location sensing on smartphones," in *MobiSys*, 2010.

[18] H. S. Ramos, T. Zhang, J. Liu, N. B. Priyantha, and A. Kansal, "Leap: a low energy assisted gps for trajectory-based services," in *UbiComp*, 2011.

[19] B. Zhang, J. Teng, J. Zhu, X. Li, D. Xuan, and Y. F. Zheng, "Ev-loc: integrating electronic and visual signals for accurate localization," in *MobiHoc*, 2012.

[20] J. Teng, B. Zhang, X. Li, X. Bai, and D. Xuan, "E-shadow: Lubricating social interaction using mobile phones," in *ICDCS*, 2011.

[21] S. Guha, K. Plarre, D. Lissner, S. Mitra, B. Krishna, P. Dutta, and S. Kumar, "Autowitness: locating and tracking stolen property while tolerating gps and radio outages," in *SenSys*, 2010.

[22] S. Rallapalli, L. Qiu, Y. Zhang, and Y.-C. Chen, "Exploiting temporal stability and low-rank structure for localization in mobile networks," in *MobiCom*, 2010.

[23] H.-J. Jang, J. Kim, and D.-H. Hwang, "Robust step detection method for pedestrian navigation systems," *Electronics Letters*, 2007.

[24] X. Zhu, F. Xu, E. Novak, C. C. Tan, Q. Li, and G. Chen, "Extracting secret key from wireless link dynamics in vehicular environments," in *INFOCOM*, 2013.

[25] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: using mobile smartphones for rich monitoring of road and traffic conditions," in *SenSys*, 2008.

[26] C. White, D. Bernstein, and A. Kornhauser, "Some map matching algorithms for personal navigation assistants," *Transportation Research Part C: Emerging Technologies*, vol. 8, no. 1-6, pp. 91–108, Dec. 2000.