Accurate, Low-Energy Trajectory Mapping for Mobile Devices

Arvind Thiagarajan, Lenin Ravindranath, Hari Balakrishnan, Samuel Madden, Lewis Girod
MIT Computer Science and Artificial Intelligence Laboratory
$\{$*arvindt, lenin, hari, madden, girod*$\}$*@csail.mit.edu*

Abstract

CTrack is an energy-efficient system for *trajectory mapping* using raw position tracks obtained largely from cellular base station fingerprints. Trajectory mapping, which involves taking a sequence of raw position samples and producing the most likely path followed by the user, is an important component in many location-based services including crowd-sourced traffic monitoring, navigation and routing, and personalized trip management. Using only cellular (GSM) fingerprints instead of power-hungry GPS and WiFi radios, the marginal energy consumed for trajectory mapping is zero. This approach is non-trivial because we need to process streams of highly inaccurate GSM localization samples (average error of over 175 meters) and produce an accurate trajectory. CTrack meets this challenge using a novel two-pass Hidden Markov Model that sequences cellular GSM fingerprints directly without converting them to geographic coordinates, and fuses data from low-energy sensors available on most commodity smart-phones, including accelerometers (to detect movement) and magnetic compasses (to detect turns). We have implemented CTrack on the Android platform, and evaluated it on 126 hours (1,074 miles) of real driving traces in an urban environment. We find that CTrack can retrieve over 75% of a user's drive accurately in the median. An important by-product of CTrack is that even devices with no GPS or WiFi (constituting a significant fraction of today's phones) can contribute and benefit from accurate position data.

1 Introduction

With the proliferation of sensor-equipped smartphones, the decades-long promise of location-based mobile services and mobile sensing applications is finally becoming real. Many location-based applications periodically probe the device's position sensor to obtain a stream of position samples, and then process this stream to obtain a trajectory. Examples include crowd-sourced traffic and navigation applications [15, 33], personalized trip management applications [28, 15], fleet management applications [21], and mobile object/asset tracking [11, 34, 7, 19, 25]. The fundamental problem in these applications is *trajectory mapping*, where the goal is to produce the most likely trajectory—a sequence of map segments—traversed by the mobile device.

If each device could always use a GPS sensor, this problem is straightforward because the majority of the position samples would usually be accurate to within a small number of meters. For applications that require positions to be monitored continuously, however, GPS has some significant practical limitations. First, GPS chipsets on today's mobile devices consume a non-trivial amount of energy, causing a significant reduction in battery life (§2). Second, in many embedded tracking applications, objects are packaged deep inside vehicles and do not have a clear line-of-sight to GPS satellites e.g., anti-theft systems on vehicles (often hidden under layers of metal), systems that track couriered packages [11] and systems like TrashTrack [34] for tracking waste and recycled materials. Most of these tracking applications also face energy and cost constraints. Third, antenna limitations on commodity mobile devices cause poor GPS performance in "urban canyons" and near high-rise buildings. Finally, a large number of phones today simply do not have GPS on them—85% of phones shipped in 2009, and projected to be over 50% for the next five years [6]. The users of these devices, a disproportionate number of whom are in developing regions, are largely being left out of the many new location-based applications.

This paper describes the design, implementation, and experimental evaluation of *CTrack*, a system for mapping the trajectory of mobile devices without using GPS. The noteworthy aspect of CTrack is that it uses much less energy than current approaches, which use GPS, WiFi localization [32, 8], or a combination of the two. CTrack processes a stream of raw, highly inaccurate position samples from mobile devices obtained by fingerprinting cellular GSM base stations, and matches them to segments on a known map in a way that achieves high accuracy. The marginal energy cost of gathering a fingerprint (a list of nearby GSM towers and their signal strengths) is zero on mobile phones because the cellular radio is usually always on. CTrack optionally augments GSM fingerprints with data from one or more of a phone's accelerometer, compass, and gyro, all of which consume tiny amounts of energy, using these sensor hints

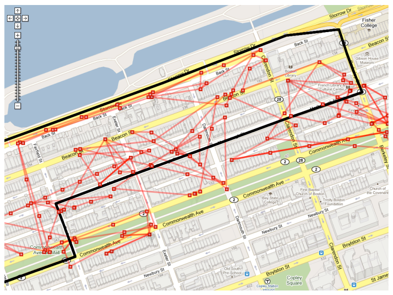


Figure 1: GSM Localization Errors. Raw location samples are in red and the true driving path is in black.

to identify the kind of movement and improve the accuracy of trajectory mapping.

GSM localization using, for example, the Placelab [8] approach, leads to errors of 100–200 meters in dense urban areas, and as much as 1 km in some areas. Such errors are too large for many applications, which require results with sufficient accuracy to pinpoint a specific road segment or route driven by a user. Figure 1 illustrates the problem with existing GSM localization. The red points are raw locations obtained from our implementation of cellular positioning as used in Placelab [8]. The actual roads traversed (ground truth) are shown in black. Directly reporting raw positions or matching locations to the nearest segments in the road map would result in unacceptably low accuracy for the applications mentioned at the beginning of this section.

CTrack makes it possible to use GSM fingerprints for accurate trajectory mapping using two novel ideas. Like previous approaches e.g. VTrack [32], CTrack matches a sequence of GSM tower observations, rather than a single point at a time, using constraints on the transitions a moving vehicle can make between locations. However, unlike VTrack, which first converts radio fingerprints to (lat, lon) coordinates, CTrack matches cellular fingerprints *directly* to a map *without* first converting them into (lat, lon) coordinates, an insight critical to achieving high accuracy. Instead, CTrack uses a two-pass algorithm. The first pass is a Hidden Markov Model (HMM) that divides space into grid cells, and determines the most likely sequence of traversed grid cells. The second pass uses a different HMM to match the traversed grid cell sequence to road segments.

The second idea in CTrack is to (optionally) *fuse* information from two low-energy phone sensors: the *accelerometer* and a *compass* or *gyroscope*. *CTrack* uses the compass/gyro to detect if the driving path took a turn, and the accelerometer to determine if the user is stopped or moving. These sensor hints can correct some common systematic errors that arise in GSM localization.

We implemented *CTrack* on the Android smartphone platform, and evaluated it on nearly 125 hours of real drives (1,074 total miles) from 20 Android phones in the Boston area. We find that:

1. *CTrack* is good at identifying the sequence of road segments driven by a user, achieving 75% precision and 80% recall accuracy. This is significantly better than state-of-the-art cellular fingerprinting approaches [8] applied to the same data, reducing the error of trajectory matches by a factor of 2.5×.
2. Although *CTrack* identifies the exact segment of travel incorrectly 25% of the time, trajectories produced by *CTrack* are on average only 45 meters away from the true trajectory. This implies that our system is useful for applications like route visualization. In this respect, CTrack is 3.5× better than map-matching raw cellular fingerprints, which results in 156 meters median error.
3. *CTrack* has a significantly better energy-accuracy trade-off than sub-sampling GPS data to save energy, reducing energy cost by a factor of 2.5× for the same level of accuracy.

2 Why Cellular?

One of the key motivations for *CTrack* is that it uses substantially less energy than GPS. This is to be expected from a theoretical standpoint because of the difference in effective radiated power (ERP) for the two systems. GPS satellites fly in an orbit 11,000 miles above the earth, with a transmission power of 50 W, resulting in 2×10^{-11} mW/m^2 at the receiver; in contrast, typical cellular systems register an ERP of up to 10 mW/m^2 [14]. This difference of 117 dB translates directly into energy consumption at the receiver, as the difference must be compensated by additional processing gain and amplification. The ERP difference also explains why GPS signals cannot be acquired without relatively unobstructed line-of-sight to orbiting satellites, and why they are more sensitive to weather conditions than GSM signals.

2.1 Energy Measurements

We performed a simple experiment to quantify the energy consumption of each of the sensors of interest — GPS, WiFi, GSM, the compass and the accelerometer on an Android G1 phone. For each sensor, we wrote an Android application to continuously sample the sensor at some given frequency, as well as continuously query the battery level indicator. We charged the phone to 100%, configured the screen to turn off automatically when idle (the default), and started the application. We used the Android telephony API to retrieve nearby cell towers and their associated signal strength values.

Figure 2 shows the reported battery life as a function of time for four configurations: GPS sampled every second, GPS sub-sampled every two minutes, WiFi scanned every second, and the configuration used by *CTrack* — scanning GSM cell towers every second, and the com-

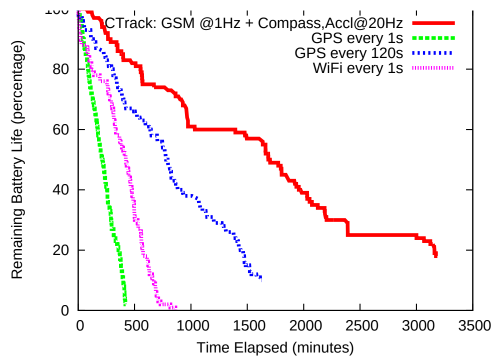


Figure 2: Energy Consumption: GPS vs WiFi vs CTrack on an Android Phone.

pass and accelerometer at 20 Hz. CTrack results in a saving of approximately 10× in battery life compared to GPS every second over 6× compared to WiFi every second. Also, although sub-sampling GPS ever 2 minutes saves energy over continuously sampling it, we show later that sub-sampling also hurts accuracy. The battery drain curves look irregular because the G1 phone estimates remaining battery life poorly – the same experiment on a Nexus One (a later model) showed a similar trend, but looked like a straight line for all sensors.

2.2 Other Energy Studies and Discussion

The numbers above are consistent many previous studies conducted on a range of phones. For example, we found [32, 31] that continuously sampling GPS on iPhone 3G and 4 resulted in 3–10 hours total battery life (iPhone 3G has lower battery life, and screen brightness varied in the different papers, resulting in different run times even without GPS). Leaving the phone on (with screen on) resulted in 10–18 hours of lifetime (this would be higher if we could turn the phone's screen off, but at the time, non-jailbroken iPhones did not support background applications.)

In [23], the authors showed that Nokia N95 phones use about 370 mW of power when GPS is left on, versus 60 mW when idling, and that continuous (once a second) GPS sampling results in 9 hours of total battery life. Several other papers [36, 16, 5, 9, 13] suggest similar numbers for N95 phones (battery life in the 7–11 hour range) with regular GPS sampling. On a more recent AT&T Tilt phone [18], the authors found that continuous GPS sampling used 400 mW, a single GPS fix costs 1.4-5.7 J of energy (depending on whether previous seen satellite information is cached or not) and a WiFi scan consumed about 0.55 J of energy.

The energy cost of GPS is rooted in the need for processing gain to acquire the positioning signals. As signal quality degrades due to obstructions or weather conditions, the energy cost of recovering the signal increases. In contrast, because phones continuously track cell towers as a part of normal operation, the marginal energy cost of CTrack is driven by CPU load. Processing a cell tower signature might require at most 100,000 instructions, which costs 5 nJ on a current generation 1 GHz Qualcomm Snapdragon processor.

In embedded (non-phone) applications that don't need the radio on, it is possible to track only the signal quality and cell ID portions of the GSM protocol. This requires observing only the BCH slots of the GSM beacon channel, which are 4.6 ms long and are transmitted once per each 1.8 second cycle. A 10% GSM receiver duty cycle should be adequate to track the strongest towers. Assuming a GSM receiver uses 17 mA at 100% duty cycle, this represents an additional power consumption of 5 mW (1.7 mA @ 2.7 V)amortized cost assuming 17 mA cost for receiver circuitry [1, 30].

Accelerometers and compasses (magnetometers) also have low overhead—for example ADXL 330 accelerometers use about 0.6 mW when continuously sampling, and at 10 Hz can be idle about 90% of the time, suggesting a power overhead of around .06 mW for sampling the accelerometer [2]. The MicroMag3 compass uses about 1.5 mW in continuous sampling, suggesting a power consumption of .15 mW or less at 10 Hz [24].

In summary, the power consumption of cellular scanning plus sensors on phones is less than 5 mW, and the power consumption of sensors alone if cellular is free—as is typical—is less than 1 mW, low enough that it does not reduce the phone's overall lifetime even when in standby mode, when it consumes 20–30 mW of power. In contrast, the best case for GPS is 75 mW in tracking mode when a fix is already acquired, but in practice is closer to 400 mW when including the energy to periodically re-acquire fixes, and is similar for WiFi scans every second or two. The power differential is thus significant.

2.3 Embedded Low-Power Applications

CTrack can also be applied outside the smartphone context to embedded low-power tagging applications. For these applications, minimizing cost and battery requirements is essential. These applications benefit from using GSM in place of GPS because of increased flexibility of antenna placement for cellular systems, and resilience to obstructed environments.

One such application is cold-chain management where the focus is on monitoring the temperature of a package during its shipping. A low-power passive cellular receiver can be used to record cellular fingerprints during transport. Upon arrival, CTrack can be run on the fingerprints to compute the shipment's trajectory and map temperature readings on to it.

Another embedded application of CTrack is Trash-Track [34, 7], where items of trash were tagged with active tags that traced the items through the path along

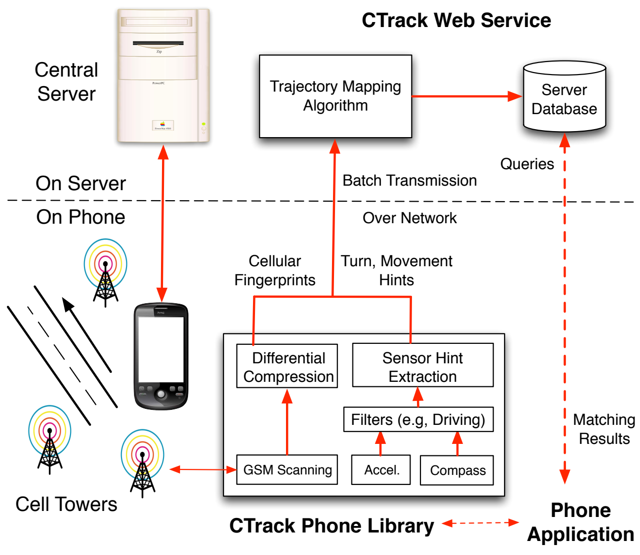


Figure 3: CTrack System Architecture.

the "disposal chain". Because the tag will eventually be destroyed, this system needs cellular communication capabilities; using the same technology for trajectory mapping consumes lower power, has lower cost, and is more robust than adding a GPS receiver to the tag.

3 System Overview

We now describe the design of CTrack. Figure 3 shows the system architecture. It consists of two software components, the *CTrack Phone Library*, and the *CTrack Web Service*. The library collects, filters, and scans for GSM and sensor data on the phones, and transmits it via any available wireless network (3G, WiFi, etc.) to the web service, which runs the trajectory mapping algorithm on batches of sensor data to produce map-matched trajectories. The mapping algorithm runs on the server to avoid storing complete copies of map data on the mobile device, and to provide a centralized database to which phone or web applications can connect to view and analyze matched tracks (e.g., for visualizing road traffic or the path taken by a package or vehicle).

Phone Library: The phone library collects a list of GSM towers and optionally, if accelerometer, compass, or gyro are available on the phone, current sensor hints. These sensor hints are binary values indicating if the phone is moving and/or turning; Section 5 describes how we extract sensor hints. The phone library also filters accelerometer data to detect if the user is stationary or walking (as in [27, 31]), for applications that want data only from moving vehicles. The library may also be configured to periodically collect GPS data for use in the training phase of our algorithm from users who wish to contribute.

Our implementation collects about 120 bytes/second of raw ASCII data on average. This quantity varies because the number of cell towers visible varies with location. We use simple gzip compression, which on our test drives resulted in just 11 bytes/second of data to be delivered. We batch this data and upload a batch every t seconds. At 11 bytes/sec, with even small batches, using a 3G uplink with an upload speed of 30 kBytes/s (typical of most current 3G networks in the US) results in very low 3G radio duty cycles—for example, setting t to 60 seconds results in the radio being awake only 0.03% of the time, which consumes a negligible amount of additional power. Once-per-minute ($t = 60$) reporting is sufficient for most applications we are concerned with, including traffic reporting, package tracking, and vehicular theft detection.

We chose not to run trajectory matching on the phone because it results in a negligible space savings, while consuming extra CPU overhead and energy. For low data rates, the primary determinant of 3G or WiFi transmission energy is the transmitter duty cycle [4], making batch reports a good idea. However, we do extract sensor hints on the phone because the algorithms for hint extraction are simple and add negligible CPU overhead, while significantly reducing data rate. The raw data rate from sampling the accelerometer/compass without compression or hint extraction is about 1.3 MBytes/hour, which means that an application collecting this data from a user's phone for two hours a day could easily rack up a substantial bandwidth bill without on-phone filtering.

CTrack Web Service: The web service receives GSM fingerprints and converts them into map-matched trajectories using the trajectory mapping algorithm. These matched trajectories are written into a database. Optionally, the user's current segment can be sent directly back to the phone. A detailed description of the trajectory mapping algorithm is given in the next section.

4 Trajectory Mapping Algorithm

CTrack's algorithm for map-matching a sequence of GSM cell tower observations ("cellular fingerprints") differs from previous approaches in two key ways. First, we do not convert cellular fingerprints into (lat, lon) coordinates before matching them to segments. We find that reducing a fingerprint to a single geographic location loses a lot of information because a given cellular fingerprint is often seen from multiple locations quite far apart. This situation is unlike the WiFi map-matching in VTrack [32], where this spread is small, and the approach of converting to centroids worked well. Second, CTrack optionally fuses sensor hints from the accelerometer and the compass to improve matching accuracy. We show that turn hints can help remove spurious turns and kinks from GSM-mapped trajectories, and movement hints can

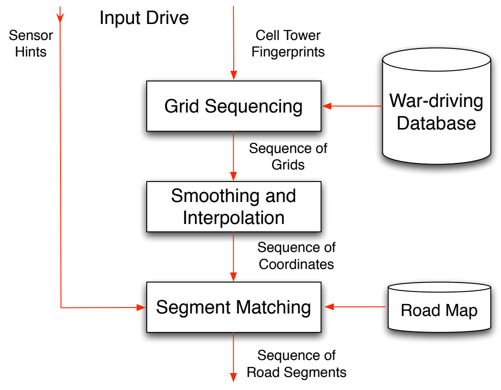


Figure 4: Trajectory Mapping Algorithm.

help remove loops, a common problem with GSM localization when a vehicle is stationary.

4.1 Algorithm Outline

The goal of the algorithm is to associate a sequence of cellular fingerprints to a sequence of road segments on a known map. Our algorithm takes as input:

1. A series of GSM fingerprints from the phone, one per second in our implementation. In our paper, the term *GSM fingerprint* refers to a set of observed IDs of cell towers and their associated received signal strength (RSSI) values. In our implementation, the Android OS gives us the cell ID and the RSSI of up to 6 neighboring towers in addition to the associated cell tower. Each RSSI value is an integer on a scale from 0 to 31 (higher means higher signal-to-noise ratio).

2. If available, time series signals from accelerometer, compass, and gyroscope sampled at 20 Hz or higher. These are converted to "sensor hints" using on-phone processing as explained below.

3. A known map database that contains the geography of all road segments in the area of interest, such as OpenStreetMaps [22], NAVTEQ, or TeleAtlas.

The output is the likely sequence of road segments traversed, one for each time instant in the input.

Figure 4 shows the components of the algorithm. *Training* builds a training database, which maps ground truth locations from GPS to observed cell towers and their RSSI values. *Grid Sequencing* uses a Hidden Markov Model (HMM) to determine a sequence of spatial grid cells corresponding to an input sequence of GSM fingerprints. The output of grid sequencing is smoothed, interpolated, and fed to *Segment Matching*, which matches grid cells to a road map using a different HMM.

Figure 5 illustrates our algorithm by example. The input "raw points" in Figure 5(a) are shown only to illustrate the extent of noise in the input data. They are not actually used by CTrack. They are computed by using the Placelab fingerprinting algorithm [8], where a cell tower fingerprint is assigned a location equal to the centroid of the closest k fingerprints in the training database (we used $k = 4$).

Next, we describe each stage of the algorithm.

4.2 Training

We divide the geographic area of interest into uniform square grid cells of fixed size g_s. We associate with each cell an ordered pair of positive integers (x, y), where $(0, 0)$ represents the south-west corner of the area of interest. We use $g_s = 125$ meters, chosen to balance running time, which increases with smaller grid size, against accuracy.

We train CTrack for the area of interest using software on mobile phones that logs a timestamped sequence of ground truth GPS locations and associated cell tower fingerprints. For each grid G in the road map, our training database stores F_G, the set of distinct fingerprints seen from G. Training can be done out-of-band using an approach similar to the Skyhook [29] fleet. Once the training database is built, it can be used to map-match or track any drive, and needs to be updated relatively infrequently. We can also collect new training data in-band from consenting participating phones that use the CTrack web service whenever the user has enabled GPS.

4.3 Grid Sequencing

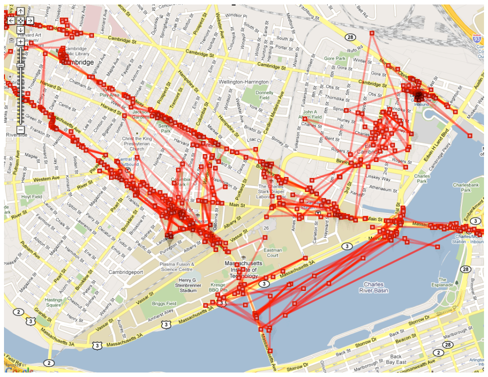
Grid sequencing uses a Hidden Markov Model (HMM) to determine the sequence of grid cells corresponding to a timestamped sequence of cellular fingerprints. An HMM is a discrete-time Markov process with a set of *hidden states* and *observables*. Each state *emits* an observable, whose likelihood is given by an *emission score*. An HMM also permits transitions among its hidden states at each time step. These transitions are governed by a different set of likelihoods called *transition scores*.

In our (first) HMM, the hidden states are grid cells and the observables are GSM fingerprints. The emission score, $E(G, F)$ captures the likelihood of observing fingerprint F in cell G. The transition score, $T(G_1, G_2)$, captures the likelihood of transitioning from cell G_1 to G_2 in a single time step.

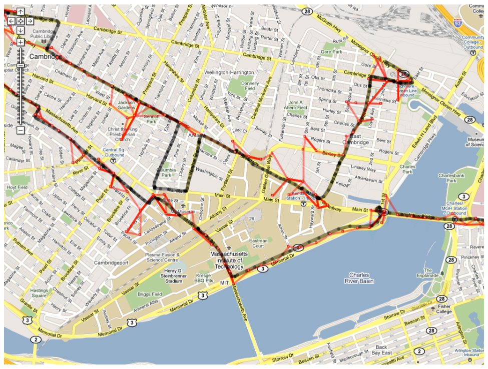
We first process the input GSM fingerprints using the *windowing* technique described below. We then use Viterbi decoding [35] to find the maximum likelihood sequence of grid cells corresponding to the windowed version of the input sequence. The maximum likelihood sequence is defined to be the sequence that maximizes the product of emission and transition scores.

We now describe the four parts of this HMM: windowing, hidden states, emission score, and transition score.

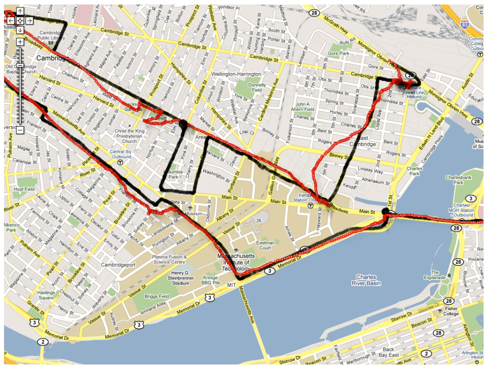
Windowing. Because it is common for a single cell tower scan to miss some of the towers near the current location, we group the fingerprints into *windows* rather than use the raw fingerprints captured once per second.



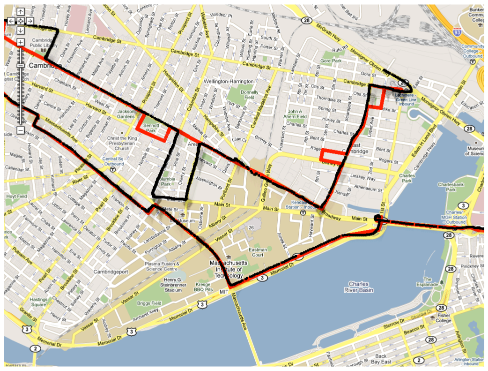
(a) Raw points before sequencing



(b) After Grid Sequencing



(c) After smoothing



(d) Final map-matched output

Figure 5: CTrack map-matching pipeline. Black lines are ground truth and red points/lines are obtained from cellular fingerprints.

We aggregate the fingerprints seen over W_{scan} seconds of scanning. We chose $W_{scan} = 5$ seconds empirically: the phone typically sees all nearby cell towers within 3 scans, which takes about 5 seconds. In our evaluation, we show that windowing improves accuracy (Table 1).

Hidden States. The hidden states of our HMM are grid cells. Given an observed fingerprint F, a grid cell G is a candidate hidden state for F if there is at least one training fingerprint in G that has at least one cell tower in common with F. Note that we might sometimes omit a valid possible hidden state G if the training data for G is sparse. To overcome this problem, we use a simple wireless *propagation model* to predict the set of cell towers seen from cells that contain no training data. The model computes the centroid and diameter of the set of all geographic locations from which each cell tower is seen in the training data. The model draws a "virtual circle" with this center and diameter and assumes that all cells in the circle see the tower in question.

Emission Score. Our emission score $E(F,G)$ is intended to be proportional to the likelihood that a fingerprint F is observed from grid cell G. A larger emission score means that a cell is a more likely match for the observed fingerprint. Our emission score uses the following heuristic. We find F_c, the closest fingerprint to F seen in training data for G. "Closest" is defined to be the value of F_c that maximizes a pairwise emission score $E_P(F,F_c)$. Our pairwise score is inspired by RADAR [3]. It captures both the number of matching cell IDs, M, between two fingerprints, and the Euclidean distance d_R in between the signal strength vectors of the matching towers:

$$E_P(F_1,F_2) = M\lambda_{match} + (d_R^{max} - d_R(F_1,F_2)) \qquad (1)$$

where λ_{match} is a weighting parameter and $d_R^{max} = 32$ is the maximum possible RSSI distance. A higher number of matching towers, and a lower value of d_R, both correspond to a higher emission score. The maximum value of the pairwise emission score is normalized (described below) and assigned as the emission score for F.

As an example, consider the fingerprints $\{$(ID=1, RSSI=3), (ID=2, RSSI=5)$\}$ and $\{$(ID=1, RSSI=6), (ID=2, RSSI=4), (ID=3, RSSI=10)$\}$. The distance between them would be $2\lambda_{match} + (32 - \frac{\sqrt{(3-6)^2+(5-4)^2}}{2})$. The weighting parameter affects how much weight is given to tower matches versus signal-strength matches: we chose $\lambda_{match} = 3$.

We normalize all our emission scores to the range $(0,1)$ to ensure that they are in the same range as transition scores, which we discuss next.

Transition Score. Our transition score is given by:

$$T(G_1,G_2) = \begin{cases} \frac{1}{d(G_1,G_2)} & , G_1 \neq G_2 \\ 1 & , G_1 = G_2 \end{cases}$$

where $d(G_1, G_2)$ is the Manhattan distance between grid cells G_1 and G_2 represented as ordered pairs (x_1, y_1) and (x_2, y_2). The transition score is based on the intuition that, between successive time instants, the user either stayed in the same cell or moved to an adjacent cell. It is unlikely that jumps between non-adjacent cells occur, but we permit them with a small probability to handle gaps in input data.

Figure 5(b) shows the output of the grid sequencing step for our running example. As we can see, sequencing removes a significant amount of noise from the input data. In our evaluation, we demonstrate that the sequencing step is critical (Figure 11).

4.4 Smoothing and Interpolation

This component takes a grid sequence as input and converts it into a sequence of (lat, lon) coordinates that are then processed by the *Segment Matching* stage.

Smoothing filter. For each grid in the sequence, we calculate the centroid of the training points seen from the grid. The centroid has the following advantage: if there is only one road segment in a grid (a frequent occurrence) and the training points lie on it, so will the centroid. Typically, centroids from grid sequencing have high frequency noise in the form of back-and-forth transitions between grids (Figure 5(b)). Hence, we apply a smoothing low-pass filter with a sliding window of size W_{smooth} to the centroids calculated as described above. The filter computes and returns the centroid of centroids in each window. This filter helps us to accurately determine the overall direction of movement and filter out the high frequency noise. We chose the filter window size, $W_{smooth} = 10$, empirically.

Interpolation. Earlier, we windowed the input trace and grouped cellular scans over a longer window of W_{scan} seconds. As a result, the smoothing filter produces only one point every W_{scan} seconds. We linearly interpolate these points to obtain points sampled at a 1-second interval, and pass them as input to the *Segment Matching* step described in §4.5.

The reason for interpolation is that segment matching produces a continuous trajectory where each segment is mapped to at least one input point. The minimum frequency of input to the segment matcher is one that ensures that even the smallest segment has at least one point. The smallest segment in the OpenStreetMaps and NAVTEQ maps is roughly 30 meters; so assuming a maximum speed of 65 MPH = 105 km/h = 29 m/s, we need about once-a-second sampling or higher to ensure this condition. Higher speeds than that generally occur on freeways where segments are usually longer than 30 meters.

Figure 5(c) shows the example drive after smoothing and interpolation. This output is free of back-and-forth transitions and correctly fixes the direction of travel at each time instant. Our evaluation quantifies the benefit of smoothing (Table 1).

4.5 Segment Matching

Segment Matching maps sequenced, smoothed grids from the previous stages to road segments on a map. It takes as input the sequence of points from the *Smoothing and Interpolation* phase, and turn and movement hints from the phone, to determine the most likely sequence of segments traversed. We describe how movement and turn hints are extracted in Section 5.

For segment matching, we use a version of the VTrack algorithm [32] augmented to process sensor hints. This step also uses an HMM. In this case, the states are the set of possible triplets $\{S, H_M, H_T\}$, where S is a road segment, $H_M \in \{0,1\}$ is the current movement hint, and $H_T \in \{0,1\}$ is the current turn hint.

The emission score of a point (lat, lon, H_M, H_T) from a state (S, H'_M, H'_T) is zero if $H_M \neq H'_M$ or $H_T \neq H'_T$. Otherwise, we make it Gaussian, with the form e^{-D^2}, where D is the distance of (lat, lon) from road segment S.

The transition score between two triplets $\{S^1, H^1_M, H^1_T\}$ and $\{S^2, H^2_M, H^2_T\}$ is defined as follows. It is 0 if segments S^1 and S^2 are not adjacent, disallowing a transition between them. This restriction ensures that the output of matching is a continuous trajectory. For all other cases, the base transition score is 1. We multiply this score with a *movement penalty*, $\lambda_{movement} (0 < \lambda_{movement} < 1)$, if $H^1_M = H^2_M = 0$ and $S_1 \neq S_2$, to penalize transitions to a different road when the device is not moving. We also multiply with a turn penalty, $\lambda_{turn} (0 < \lambda_{turn} < 1)$ if the transition represents a turn, but the sensor hints report no turn. We used $\lambda_{movement} = 0.1$ and $\lambda_{turn} = 0.1$. Our algorithm is not very sensitive to these values, since the penalties are multiplied together and a small enough value suffices to correct incorrect turn/movement patterns.

Similar to VTrack, the HMM also includes a *speed constraint* that disallows transitions out of a segment if sufficient time has not been spent on that segment. The maximum permitted speed can be calibrated depending on whether we are tracking a user on foot or in a vehicle.

The output of the segment matching stage is a set of segments, one per fingerprint in the interpolated trace (which, on average, is the same periodicity as the original input). The output for the running example is shown in Figure 5(d).

When running online as part of the CTrack web service, the segment matcher takes turn hints and sequenced grids as input in each iteration and returns the current segment to an application querying the web service.

Running time. The run-time complexity of the entire algorithm, including all stages, is $O(mn)$, where m is

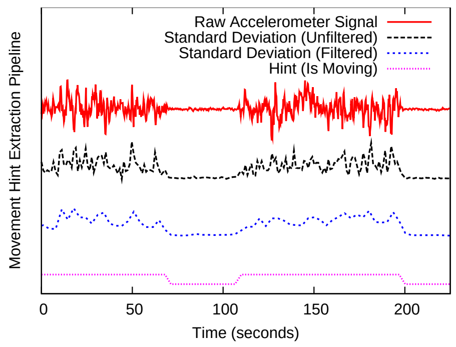


Figure 6: Movement hint extraction from accelerometer.

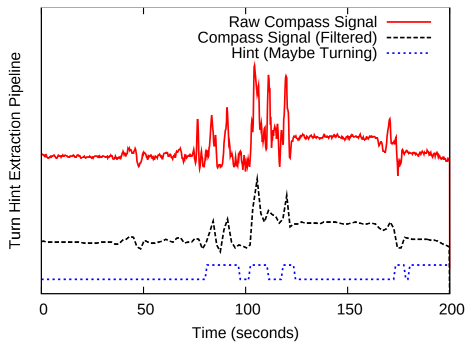


Figure 7: Turn hint extraction from compass.

the number of input fingerprints and n is the number of search states (the larger of the number of grid cells and road segments on the map). Our Java implementation on a MacBook Pro with 2.33 GHz CPU and 3 GB RAM map-matched an hour-long trace in approximately two minutes, approximately 30 times faster than real time. It is straightforward to reduce the run time by more aggressively pruning the search space, but we have not found the need to do so yet.

5 Sensor Hint Extraction

CTrack includes a sensor hint extraction layer that processes raw phone accelerometer readings to infer information about whether the phone being tracked is moving or not, and processes orientation sensor readings from a compass or a gyroscope to heuristically infer vehicular turns. These hints are transmitted along with the GSM fingerprint to the server for map matching.

Anomaly detection. Anomaly detection filters out periods when the user is lifting the phone, speaking on the phone, texting, waving the phone about, or otherwise using the phone. We want to use accelerometer and compass/gyro data only in periods where we have high confidence that the phone is more or less at rest relative to the moving object in which it is located (e.g., on a flat surface or in a user's pocket). We found empirically that when driving with the phone at rest in a vehicle or in a pocket, the raw accelerometer magnitude tends to be smaller than 14 ms^{-2}. Hence, we look for spikes in the raw accelerometer magnitude that exceed a threshold of 14 ms^{-2}. Whenever we encounter such a spike, we ignore all accelerometer and compass data in the map-matching algorithm until the phone comes back to a state of rest (this can be detected using standard deviation of acceleration, as explained below). On more recent phones such as the iPhone 4, the in-built gyroscope gives the exact orientation of the phone which can be directly read to determine if the phone is on a flat surface/in a user's pocket.

Having filtered out anomalous periods, the hint extraction processes stable periods to extract movement and turn hints, as explained below.

Movement Hints. Our algorithm uses accelerometer data sampled at 20 Hz. We extract a simple "static" or "moving" (1-bit hint) rather than integrating the accelerometer data to compute velocities or processing it in a more complex way, because accelerometer data is noisy and hard to integrate accurately without accumulating drift. In contrast, it is easy to detect movement with an accelerometer: within a stable (spike-free) period, the accelerometer shows a significantly higher variance while moving than when stationary.

Accordingly, we compute a boolean (true/false) movement hint for each time slot. We divide the data into one-second slots and compute the standard deviation of the 3-axis magnitude of the acceleration in each slot. Directly thresholding standard deviation sometimes results in spurious detections when the vehicle is static and the signal exhibits a short-lived outlier. To fix this, we apply an EWMA filter to the standard deviation stream to remove short-lived outliers. We then apply a threshold $\sigma_{movement}$, on the standard deviation to label each time slot as "static" or "moving". We used a subset of our driving data across multiple phones as training (where we do know ground truth from GPS), to learn the optimal value of $\sigma_{movement}$, which turned out to be approximately 0.15 ms^{-2} for one-second windows. Figure 6 illustrates our movement hint extraction algorithm on example data.

Turn Hints. The orientation sensor of a smartphone (compass/gyroscope) provides orientation about three axes. We are interested in the axis that provides the relative rotation of the phone about an axis parallel to gravity (called "yaw" on the iPhone 4).

Because the phone can be in any orientation to start with in a handbag or pocket, we do not use the absolute orientation in any of our algorithms. We have observed that irrespective of how the phone is situated, a true change in orientation manifests as a *persistent*, *significant*, and *steep* change in the value of the orientation sensor.

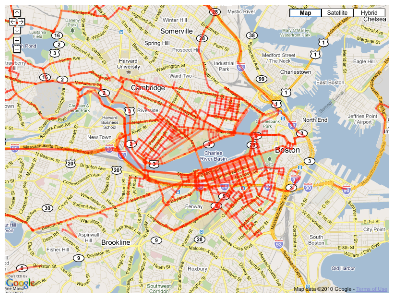


Figure 8: Coverage map of our driving data set.

With compass data, the main challenge is that the orientation reported is noisy because metallic objects nearby, or because the compass becomes uncalibrated. We solved this problem by applying a *median filter* with a three-second window on the raw orientation values, which filtered out non-persistent noise with considerable success (a mean filter would also remove noise, but would blur sharp transitions that we do want to observe). We then find transitions with a magnitude exceeding at least 20 degrees and slope exceeding a threshold, which we fixed at 1.5 by experimentation.

Figure 7 illustrates a plot of the compass data with the turn marked, and the processing steps required to generate a turn hint. We note that even after filtering, a true change in orientation can sometimes be produced by the phone sliding around within a pocket or a bag, or turning for reasons other than the car actually turning.

6 Evaluation

In this section, we show that the trajectory matches produced by *CTrack* are: (1) accurate enough to be useful for various tracking and positioning applications, (2) superior to sub-sampled GPS in terms of the accuracy-energy tradeoff, and (3) significantly better than strategies that reduce cellular fingerprints to point locations before matching. We investigate how much each of the four techniques used in CTrack —sequencing, windowing, smoothing, and sensor hints—contribute to the gains in accuracy.

6.1 Method and Metrics

We evaluate CTrack on 126 hours of real driving data in the Cambridge-Boston area, collected from 15 Android G1 phones and one Nexus One phone over a period of 4 months. We configured our phone library for the Android OS to continuously log the ground truth GPS location and the cell tower fingerprint every second, and the accelerometer and compass at 20 Hz. Our data set covers 3,747 road segments, amounts to 1,718 km of driving, and 560 km of distinct road segments driven. The data set includes sightings of 857 distinct cell towers. Figure 8 shows a coverage map of the distinct road segments driven in our data set.

From 312 drives in all, we selected a subset of 53 drives verified manually to have high GPS accuracy as *test drives*, amounting to 109 distinct km. We picked a limited subset as test drives to ensure each test drive was contained entirely within a small bounding box with dense training coverage. This is because evaluating the algorithm in areas of sparse coverage (which many of the other 259 drives venture into) could bias results in our favor by reducing the number of candidate paths to map-match to. For each test drive, we perform *leave-one-out* evaluation of the map-matching algorithm: we train our algorithm on all 311 drives excluding the test drive, and then map-match the test drive using CTrack. We do this to ensure enough training data for each drive, and at the same time to keep the evaluation fair.

We compare CTrack to two other strategies in terms of energy and accuracy:

1. *GPS k* gets one GPS sample every k minutes ($k = 2,4$), interpolates, and map-matches it using VTrack [32].
2. *Placelab-VTrack* computes the best static localization estimate for each time instant using Placelab's technique [8], and matches the static estimates using VTrack [32]. The VTrack paper shows that its HMM does much better than just matching each point to the nearest segment.

We use three metrics in our evaluation of accuracy: *precision*, *recall*, and *geographic error*. Our precision and recall are similar to conventional precision and recall, but take the order of matched segments in the trajectory into account. We say that a subset of segments in a trajectory T_1 that also appears in trajectory T_2 are *aligned* if those segments appear in T_1 in the same order in which they appear in T_2. Given a ground truth sequence of segments G and an output sequence X to evaluate (produced by one of the algorithms), we run a dynamic program to find the maximum length of aligned segments between G and X. We define:

$$Precision = \frac{Total\ length\ of\ aligned\ segments}{Total\ length\ of\ X} \quad (2)$$

$$Recall = \frac{Total\ length\ of\ aligned\ segments}{Total\ length\ of\ G} \quad (3)$$

We estimated the ground truth sequencing of segments by map-matching GPS data sampled every second with VTrack [32], and manually fixing a few minor flaws in the results.

Geographic Error. Precision and recall are relevant to applications that care about obtaining information at a segment-level, such as traffic monitoring. However, applications such as visualization do not need to know the exact road segments traversed, but may want to identify the broad contours of the route followed (e.g., mistaking a road for a nearby parallel road may be acceptable).

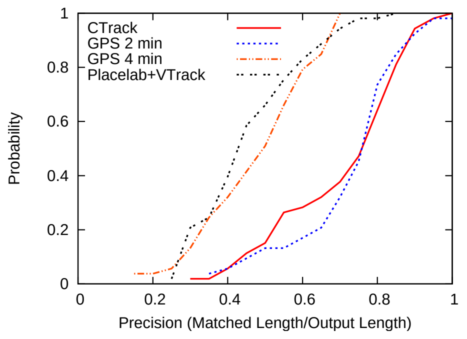


Figure 9: CDF of Precision: Comparison.

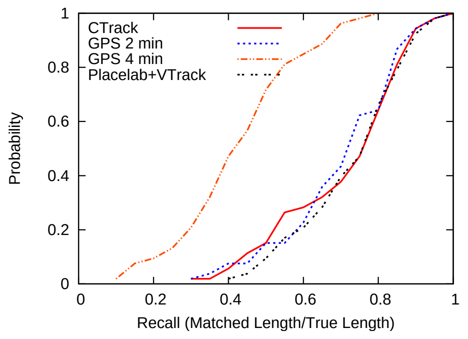


Figure 10: CDF of Recall: Comparison.

To quantify this notion, we compute a third metric, *geographic error*, which captures the spatial distance between the ground truth and the matched output. We compute the maximum alignment between the ground truth trajectory G and output trajectory X using dynamic programming. This alignment matches each segment S of X to either the same segment S on G (if CTrack matched that segment correctly) or to a segment $S_{wrong} \in G$ (if matched incorrectly). Define the *segment geographic error* to be the distance between S and S_{wrong} for incorrect segments, and 0 for correctly matched segments. The mean segment geographic error over all segments in X is the *overall geographic error*.

6.2 Key Findings

The key findings of our evaluation are:

1. CTrack has 75% precision and 80% recall in both the mean and median, and a median geographic error of 44.7 meters. We discuss what these numbers mean in the context of real applications below.

2. CTrack has 2.5× better precision and 3.5× smaller geographic error than *Placelab+VTrack*.

3. CTrack is equivalent in precision to map-matching GPS sub-sampled every 2 minutes while consuming over 2.5× less energy. It also reduces error ($1 - precision$) by a factor of over 2× compared to sub-sampling GPS every 4 minutes, consuming a similar amount of energy. CTrack is 6× better than continuous WiFi sampling in terms of battery lifetime on the Android platform.

4. The first step of CTrack, grid sequencing, is critical. Without sequencing, CTrack effectively reduces to computing a (lat, lon) estimate from the best fingerprint match, ignoring all other data. The median precision without sequencing is only 50%. See Section 6.4 for more detail.

5. We can extract movement and turn hints from raw sensor data with approximately 75% precision and recall. These hints improve accuracy by removing spurious loops and turns in the output. Using hints improves precision by 6% and recall by 3%. See Section 6.5 for more detail.

6.3 Accuracy Results

Figure 9 shows a CDF of the map-matching precision for CTrack, *GPS k* (for $k = 2, 4$ minutes) and *Placelab+VTrack*. CTrack has a median precision of 75%, much higher than the both the energy-equivalent strategy of sub-sampling GPS every 4 minutes (48%), and *Placelab+VTrack* (42%). In effect, CTrack has over 2× lower error ($1 - precision$) than sub-sampling GPS every 4 minutes, and over 2.5× lower error than map-matching cellular localization estimates output by the Placelab method. Also, CTrack has equivalent precision to map-matching GPS sub-sampled every two minutes, while reducing energy consumption by approximately 2.5× compared to this approach (Figure 2).

Figure 10 shows a CDF of the recall. All the strategies except *GPS 4* min are equivalent in terms of recall. Sub-sampling GPS every four minutes has poor recall (median only 41%) because a four-minute sampling interval misses significant turns in our input drives and finds the wrong path. The fact that *Placelab+VTrack* has identical recall shows that simple static cellular localization does manage to recover a significant part of the input drive. However, converting cellular fingerprints directly to points results in significant noise and long-lived outliers, and hence produces a large number of incorrect segments when map-matched directly (i.e., has low precision).

To understand what 75% precision might mean in terms of a an actual application, we refer readers to our work on VTrack [32], which studies the relationship between map-matching accuracy and the accuracy of two end-to-end applications: traffic delay monitoring and traffic hot-spot detection. We found that a median precision of 85% was still useful for accurate traffic delay estimation. Our results for cellular (75%) are only somewhat worse, and while not directly comparable, they suggest a significant portion of delay data from CTrack would be useful.

For applications such as route visualization, or those that aggregate statistics over paths (e.g., to compute histograms over which of n possible routes is taken), or

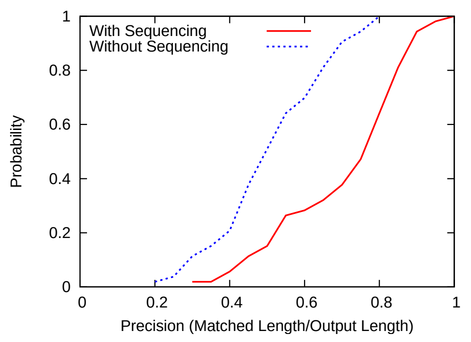


Figure 11: Precision with and without grid sequencing.

those that simply show a user's location on a map, getting most segments right with a low overall error is likely sufficient. Our median geographic error is quite low—just 45 meters—suggesting CTrack would have sufficient accuracy for such applications. In contrast, the median geographic error of the Placelab+VTrack approach is 156 meters, over 3.5× worse than CTrack.

Filtering using a confidence predictor. We investigated whether a confidence metric could be used to filter out drives on which CTrack does poorly, thereby trading-off some recall for substantially better precision, which would be useful for accuracy-sensitive applications. We found two predictors, both weakly correlated with map-matching accuracy: (a) the 90th percentile distance of smoothed grids from the segments they are matched to, and (b) the mean difference (over all points P) in emission score between the segment that P is matched to in the output, and the segment closest to P. The intuition is that a point far away from the road segment it is matched to, or closer to a different road segment, implies lower confidence in the match. When applying these confidence filters to our output drives, we currently improve the median precision from 75% to 86%, but lose substantially in terms of recall, whose median reduces from 80% to 35%). In future work, we plan to explore whether *boosting* [12] can combine these weak confidence predictors into a stronger one.

6.4 Benefit of Sequencing

We elaborate on one of our key technical contributions: the idea that the first pass of grid sequencing *before* converting fingerprints to geographic locations is crucial to achieving good matching accuracy. We provide experimental evidence supporting this idea. We also show that windowing and smoothing help improve matching accuracy, though to a lower degree.

Impact of Sequencing. Figure 11 is a CDF that compares the precision of CTrack with and without the first pass of grid sequencing. This figure shows that sequencing is critical to achieving reasonable accuracy: without sequencing, the median precision drops from 75% to 50%. The reason is that running CTrack without sequencing amounts to reducing each fingerprint to its best match in the training database, ignoring the sequence of points.

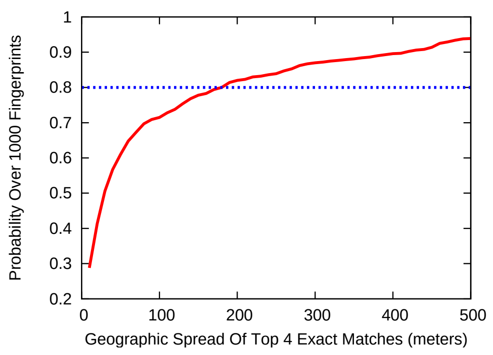


Figure 12: Geographic spread of exact matches. The dashed line shows the 80th percentile.

As mentioned earlier, reducing a fingerprint to a single geographic location loses information because a given cellular fingerprint is seen from multiple locations quite far apart. Figure 12 illustrates the CDF of this geographic spread. We selected 1000 fingerprints at random from our training data. For each fingerprint F, we found all the *exact matches* for F, i.e. fingerprints F' with the exact same set of towers in the training data as F. We ordered the matches by similarity in signal strength, most similar first, and computed the geographic diameter of the top k matches for each fingerprint (using $k = 4$).

The figure shows that over 20% of matching sets have a diameter exceeding 150 meters, and at least 10% have a diameter exceeding 400 meters. Recall that the methods in Placelab (and RADAR, if applied to cellular data) would simply compute the centroid of the top k matches. This approach does not work well for sets with a large geographic spread, and motivates the need for the fundamentally different approach used in CTrack in which we keep track of *all* possible likely locations, and then use a continuity constraint to *sequence* these locations in two steps.

Windowing and Smoothing. Table 1 shows the precision and recall of CTrack with and without windowing and smoothing, two other heuristics used in CTrack. We see that each of these features improves the precision by approximately 10%, which is a noticeable quantity. The recall does not improve because the algorithm without windowing/smoothing is good enough to identify most of the segments driven: the heuristics mainly help eliminate loops in the output.

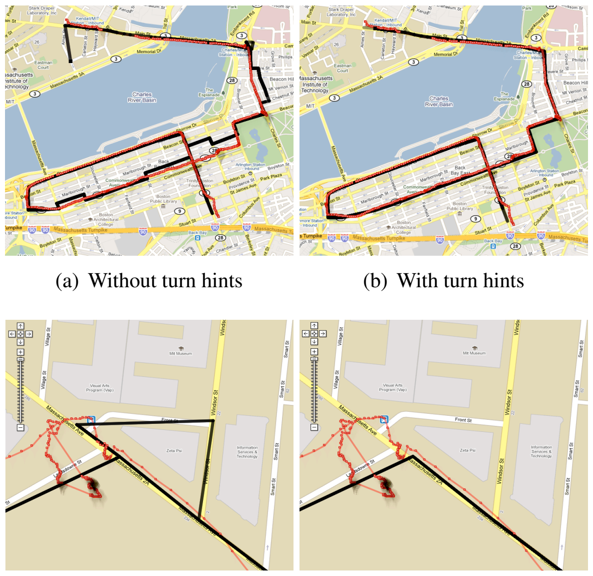
6.5 Do Sensor Hints Help?

Figure 13 illustrates by example how turn hints extracted from the phone compass help in trajectory matching. Without using turn hints (Figure 13(a)), our algorithm

	With		Without	
	Prec.	Recall	Prec.	Recall
Windowing	75.4%	80.3%	65.6%	82.3%
Smoothing	75.4%	80.3%	66.5%	82.5%

Table 1: Windowing and smoothing improve median trajectory matching precision.

finds the overall path quite accurately but includes several spurious turns and kinks, owing to errors in cellular localization. After including turn hints in the HMM, the false turns and kinks disappear (Figure 13(b)).



(a) Without turn hints (b) With turn hints

(c) Without movement hints (d) With movement hints

Figure 13: Sensor hints from the compass and accelerometer aid map-matching. Red points show ground truth and the black line is the matched trajectory.

In Figure 13(c), the driver stopped at a gas station to refuel, which can be seen from the cluster of ground-truth GPS points. Before using movement hints, errors from cellular localization were spread out, causing the map-matching to introduce a loop not present in the ground truth (Figure 13(c)). After incorporating movement hints, the speed constraint in our HMM eliminates this loop because it detects that the car would not have had sufficient time to complete the loop (Figure 13(d)). We note a limitation of the movement hint: this kind of stop detection works because the phone was placed on the dashboard: if it had been in the driver's pocket during refueling, the movement hints would not have helped had the driver gotten out of the car and been moving about.

Figure 14 is a CDF that compares the precision of CTrack with and without sensor hints (both movement and turn). This figure shows that sensor hints improve the median precision of matching by approximately 6%. While this may not seem huge, there exist several trajectories for which the hints do help significantly, suggesting that using them is a good idea when available. In our experience, the main benefit of the hints is in eliminating the several "kinks" and spurious turns in the matched trajectory, which our metrics don't adequately capture.

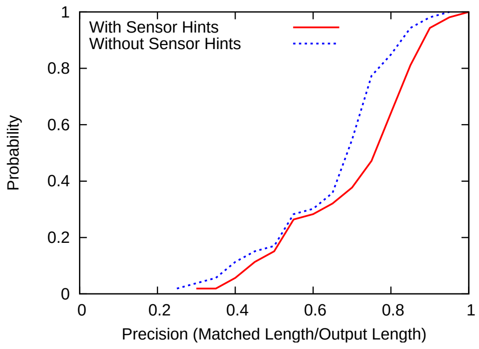


Figure 14: Precision with and without sensor hints.

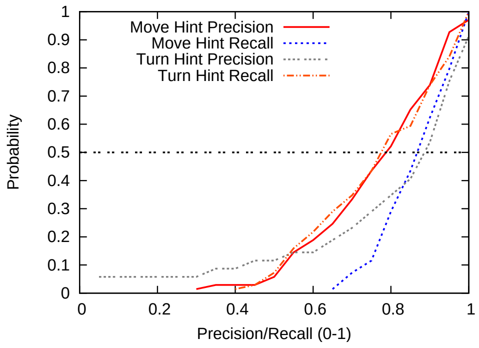


Figure 15: Precision/Recall CDF For Hint Extraction.

We used the ground truth GPS to measure how accurately our CTrack is able to extract individual movement and turn hints. We found that the median precision and recall of both motion and turn hint extraction exceeds 75%.

6.6 How Much Training?

To quantify the amount of training data essential to achieving good trajectory mapping accuracy with CTrack, we picked a pool of test drives at random, amounting to 5% of our data set (8 hours of data), and designated the remaining 95% as the training pool. We picked subsets of the training pool of increasing size, i.e. first using fewer drives for training, then using more. In each run, the training subset was used to train CTrack and then evaluated on the test pool. Figure 16 shows the mean precision and recall of CTrack on the test pool as a function of the number of drive hours of training data used to train the system. The accuracy is poor for very small training pools, as expected, but encouragingly, it quickly increases as more training data is available. The algorithm performs almost as accurately with 40 hours of training data as with 120, suggesting that 40 hours of training is sufficient for our data set.

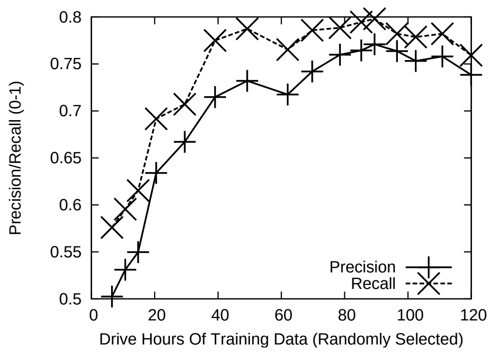


Figure 16: Prec./Recall vs Training Data Size.

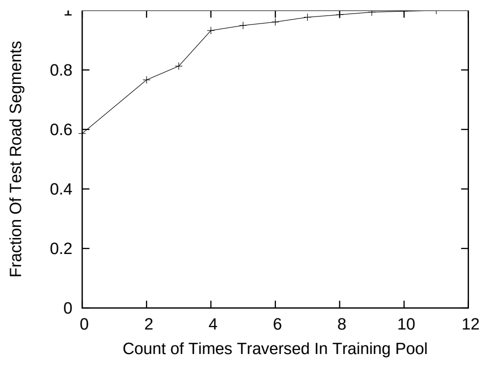


Figure 17: CDF of Drive Counts, 40 Hrs Training Data.

The 40-hour number, of course, is specific to the geographic area we covered in and around Boston, and to the test pool. To gain more general insight, we measure the *drive count* for each road segment in the test pool, defined as the number of times the segment is traversed by any drive in the training pool. Figure 17 shows the distribution of *test segment drive counts* corresponding to 40 hours of training data. While the mean drive count is approximately 3, this does not mean each road segment on the map needs to be driven thrice to achieve good accuracy. As the graph shows, about 60% of the test segments were not traversed even once in the training pool, but we can still map-match many of these segments correctly. The reason is that they lie in the same grid cell as some nearby segment that was driven in the training pool. This result promising because it suggests that training does not have to cover every road segment on the map to achieve acceptable accuracy.

7 Related Work

Placelab performed a comprehensive study of GSM localization and used a fingerprinting scheme for cellular localization [8]. RADAR used a similar fingerprinting heuristic for indoor WiFi localizations [3], and our map-matching emission score is inspired by these methods. However, neither Placelab nor RADAR address the problem of trajectory matching, and are concerned with the accuracy of individual localization estimates, rather than finding the optimal sequencing of estimates. As shown by our results, this sequencing step is critical: applying a map-matching algorithm directly to Placelab-style location estimates results in significantly worse accuracy (by a factor of over $2\times$) compared to CTrack.

Letchner et al. [17] and our previous work on VTrack [32] use HMMs for map-matching. However, these previous algorithms use and process (lat, lon) coordinates as input and use a Gaussian noise model for emissions, and are hence unsuitable and inaccurate for map-matching cellular fingerprints, as shown by our results. Nor do they use sensor hints.

CompAcc [10] proposes to use smartphone compasses and accelerometers to find the best match for a walking trail by computing directional "path signatures" for these trails. They do not use cell towers. However, from our understanding, the paper uses absolute values of compass readings. This approach did not work in our experiments, because the absolute orientation of a phone can be quite different depending on whether it is in a driver's pocket, on a flat surface, or held in a person's hand. For this reason, we chose to use boolean turn hints instead, which are more robust and can be accurately computed regardless of changes in the phone's initial orientation or position. For extracting motion hints and detecting walking and driving using the accelerometer, we use algorithms similar to those in [27, 31, 26].

Some previous papers [9, 23, 16] have proposed energy-efficient localization schemes that reduce reliance on continuously sampling GPS by using a more energy-efficient sensor, such as the accelerometer, to trigger sampling GPS. RAPS [23] also uses cell towers to "blacklist" areas where GPS accuracy is low and hence GPS should be switched off, to save energy. However, none of these papers address trajectory matching or propose a GPS-free, accurate solution for map-matching.

Skyhook [29] and Navizon [20] are two commercial providers for WiFi and Cellular localization, providing databases and APIs that allow programmers to submit WiFi access point(s) or cell tower(s) and look up the nearest location. However, to the best of our knowledge, they do not use any form of sequencing or map-matching, and focus on providing the best static localization estimate.

8 Conclusion

We described CTrack, an energy-efficient, GPS-free system for trajectory mapping using cellular tower fingerprints. The key lesson we learned was that sequencing cellular fingerprints before matching them is critical to achieving good accuracy. On smartphones, our CTrack implementation uses close to zero extra energy while achieving good mapping accuracy, making it a good way to distribute collaborative trajectory-based applications like traffic monitoring to a huge number of users without any associated energy consumption or battery drain concerns. A GPS-free approach to trajectory matching also opens up the possibility of providing more fine-grained location services on the world's most popular, cheapest phones that do not have GPS, but that do have GSM connectivity.

Acknowledgments

This work was supported by the National Science Foundation under grant CNS-0931550.

References

[1] Analog Devices AD9864 Datasheet: GSM RF Front End and Digitizing Subsystem. http://www.analog.com/static/imported-files/data_sheets/AD9864.pdf.

[2] Analog Devices, Inc. *ADXL330: Small, Low Power, 3-Axis +/-3 g iMEMS Accelerometer (Data Sheet)*, 2007. http://www.analog.com/static/imported-files/data_sheets/ADXL330.pdf.

[3] P. Bahl and V. Padmanabhan. RADAR: An In-building RF-based User Location and Tracking System. In *INFOCOM*, 2000.

[4] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In *IMC*, 2009.

[5] F. Ben Abdesslem, A. Phillips, and T. Henderson. Less is more: Energy-efficient Mobile Sensing with SenseLess. In *MobiHeld*, 2009.

[6] GPS and Mobile Handsets. http://www.berginsight.com/ReportPDF/ProductSheet/bi-gps4-ps.pdf.

[7] A. Boustani, L. Girod, D. Offenhuber, R. Britter, M. I. Wolf, D. Lee, S. Miles, A. Biderman, and C. Ratti. Investigation of the Waste Removal Chain Through Pervasive Computing. *IBM Journal of Research and Development*, 2010.

[8] M. Y. Chen, T. Sohn, D. Chmelev, D. Haehnel, J. Hightower, J. Hughes, A. Lamarca, F. Potter, I. Smith, and A. Varshavsky. Practical Metropolitan-scale Positioning for GSM Phones. In *UbiComp*, 2006.

[9] I. Constandache, S. Gaonkar, M. Sayler, R. Choudhury, and L. Cox. EnLoc: Energy-Efficient Localization for Mobile Phones. In *INFOCOM*, 2009.

[10] I. Constandache, R. Roy Choudhury, and I. Rhee. CompAcc: Using Mobile Phone Compasses and Accelerometers for Localization. In *INFOCOM*, 2010.

[11] Fedex intros Senseaware Sensor for Tracking Packages. http://www.electronista.com/articles/09/11/27/senseaware.sensor.sends.temps.drops.more.

[12] Y. Freund and R. E. Schapire. A Decision Theoretic Generalization of Online Learning and an Application to Boosting. In *EuroCOLT*, 1995.

[13] 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.

[14] Information On Human Exposure To Radiofrequency Fields From Cellular and PCS Radio Transmitters. http://www.fcc.gov/oet/rfsafety/cellpcs.html.

[15] iCartel. http://icartel.net/icartel-docs/.

[16] M. B. Kjærgaard, J. Langdal, T. Godsk, and T. Toftkjær. EnTracked: Energy-efficient Robust Position Tracking for Mobile Devices. In *MobiSys*, 2009.

[17] J. Krumm, J. Letchner, and E. Horvitz. Map Matching with Travel Time Constraints. In *SAE World Congress*, 2007.

[18] K. Lin, A. Kansal, D. Lymberopoulos, and F. Zhao. Energy-accuracy Trade-off for Continuous Mobile Device Location. In *MobiSys*, 2010.

[19] LoJack Car Security System For Stolen Vehicle Recovery. http://www.lojack.com.

[20] Navizon. http://www.navizon.com.

[21] Qualcomm Transportation: OmniTRACKS Mobile Communications System. http://www.qualcomm.com/products_services/mobile_content_services/enterprise/omnitracs.html.

[22] OpenStreetMap. http://www.openstreetmap.org.

[23] J. Paek, J. Kim, and R. Govindan. Energy-efficient Rate-adaptive GPS-based Positioning for Smartphones. In *MobiSys*, 2010.

[24] PNI Corporation. *MicroMag3 3-Axis Magnetic Sensor Module*. http://www.sparkfun.com/datasheets/Sensors/MicroMag3%20Data%20Sheet.pdf.

[25] Qualcomm inGeo Service. http://www.qualcomm.com/innovation/stories/ingeo.html.

[26] L. Ravindranath, C. Newport, H. Balakrishnan, and S. Madden. Improving Wireless Network Performance Using Sensor Hints. In *NSDI*, 2011.

[27] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava. Using Mobile Phones to Determine Transportation Modes. *Transactions on Sensor Networks*, 6(2), 2010.

[28] RunKeeper. http://runkeeper.com.

[29] Skyhook. http://www.skyhookwireless.com.

[30] Telit GE865 Datasheet. http://www.telit.com/module/infopool/download.php?id=1666.

[31] A. Thiagarajan, J. Biagioni, T. Gerlich, and J. Eriksson. Cooperative Transit Tracking Using GPS-enabled Smart-phones. In *SenSys*, 2010.

[32] A. Thiagarajan, L. Sivalingam, K. LaCurts, S. Toledo, J. Eriksson, S. Madden, and H. Balakrishnan. VTrack: Accurate, Energy-Aware Road Traffic Delay Estimation Using Mobile Phones. In *SenSys*, 2009.

[33] TomTom. http://www.tomtom.com.

[34] Trash Track. http://senseable.mit.edu/trashtrack.

[35] A. J. Viterbi. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. In *IEEE Transactions on Information Theory*, 1967.

[36] Y. Wang, J. Lin, M. Annavaram, Q. A. Jacobson, J. Hong, B. Krishnamachari, and N. Sadeh. A Framework of Energy Efficient Mobile Sensing for Automatic User State Recognition. In *MobiSys*, 2009.