UrbanEye: An Outdoor Localization System for Public Transport

Rohit Verma*, Aviral Shrivastava*, Bivas Mitra*, Sujoy Saha‡,
Niloy Ganguly*, Subrata Nandi†, Sandip Chakraborty*
*Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, INDIA 721302
‡Department of Computer Application, National Institute of Technology Durgapur, INDIA 713209
†Department of Computer Science and Engineering, National Institute of Technology Durgapur, INDIA 713209
Email: rohitverma.kgp@gmail.com, aviralshrivastava93@gmail.com, bivas@cse.iitkgp.ernet.in, sujoy.ju@gmail.com,
niloy@cse.iitkgp.ernet.in, subrata.nandi@gmail.com, sandipc@cse.iitkgp.ernet.in

***Abstract*—Public transport in suburban cities (covers 80% of the urban landscape) of developing regions suffer from the lack of information in Google Transit, unpredictable travel times, chaotic schedules, absence of information board inside the vehicle. Consequently, passengers suffer from lack of information about the exact location where the bus is at present as well as the estimated time to be taken to reach the desired destination. We find that off-the-shelf deployment of existing (non-GPS) localization schemes exhibit high error due to sparsity of stable and structured outdoor landmarks (anchor points). Through rigorous experiments conducted over a month however, we realize that there are a certain class of volatile landmarks which may be useful in developing efficient localization scheme. Consequently, in this paper, we design a novel generalized energy-efficient outdoor localization scheme - UrbanEye, which efficiently combines the volatile and non-volatile landmarks using a specialized data structure, the probabilistic timed automata. UrbanEye uses speed-breakers, turns and stops as landmarks, estimates the travel time with a mean accuracy of ± 2.5 mins and produces a mean localization accuracy of 50 m. Results from several runs taken in two cities, Durgapur and Kharagpur, reveal that UrbanEye provides more than 50% better localization accuracy compared to the existing system Dejavu [1], and consumes significantly less energy.**

***Index Terms*—Navigation; Localization; Public Transport**

I. Introduction

Alice during her visit to Durgapur wants to travel to the city museum from her hotel using city bus. But Alice is surprised to find that there is hardly any information about the public bus routes, schedules, fares, travel time estimates available in Google Transit system for Durgapur. Through enquiry she locates the nearby bus stoppage; however, she cannot find any stoppage sign around and certainly no information display board. On boarding a bus, much to her amazement she finds that even the bus itself does not have any electronic information system. She remains anxious throughout her trip about the timing to get down, as she cannot afford to continuously use energy hungry GPS navigation. Alice notices that the bus skips few stops to reach a particular busy stop before another bus to ensure that it picks up a larger pool of passengers. Finally, she reaches her destination, the travel time overshoots by 30% than expected and is barely able to get out with the help of a friendly commuter. Being frustrated by the hassles, on her way back to hotel Alice decides to pay 10 times more to hire a cab.

The above scenario though imaginary unfortunately represents the state of public transport in various suburban cities of developing countries (India is a glaring example [2]). Here a major bulk of the public transport is provided via the buses run by multiple private owners. As a result, unhealthy competition, unplanned traffic and poor infrastructure lead to chaotic stoppage patterns [3], schedules and unpredictable speed variations, thus causing extreme inconvenience to commuters and visitors alike. A simple help in this context would be to provide travelers with a mobile app which will act as a virtual electronic board to trace the bus route and to locate destination - the facility bus should otherwise have provided. GPS assistance may be a trivial solution, however that may prove extremely energy inefficient. In this paper, we take a deeper look into the underlying technical challenges in developing an energy efficient navigation system to assist people like Alice navigate smartly while onboard a public bus.

The basic outline of a general navigation system would be to identify landmarks using the sensors present in the mobile phones and localize the route using the landmarks as checkpoints. There exist several inertial navigation systems (INS) specially catering to cars/cabs [1], [4], bus [5], [6], underground metro [7], [8], pedestrians [9]–[11]. A recent navigation system Dejavu [1], experimented on the roads of Alexandria, Egypt specifically identifies landmarks and localize a route accordingly. However, when Dejavu is tested on bus routes in Durgapur and Kharagpur, it shows a mean deviation greater than 120m. Certainly, the result is not acceptable for implementing any practical location-based services. The fundamental reason behind the poor performance of Dejavu lies in its landmark fixing schemes - it mostly relies on the presence and identification of the virtual landmarks (landmarks specified by the presence of certain Wi-Fi zone and the area where GSM handover occurs) along with a variety of physical landmarks. However, our feasibility experiments (in the next section) illustrates the fact that, in one hand identifying virtual landmarks is a major challenge, on the other hand some of the physical landmarks are volatile in nature. Unlike few physical landmarks such as turn and speed breaker, landmarks such as bus stops are indeed volatile; buses occasionally skip the designated stop.

In this paper, we develop UrbanEye, an energy-efficient outdoor localization system for route navigation and travel time prediction for the city commuters (§III to §V). Our extensive feasibility study uncovers the failure of the virtual landmarks (Wi-Fi signal strength, GSM handoff) in the localization process (§II). Consequently, the core of the system is based on a novel framework for accurately identifying both volatile and non-volatile physical landmarks. In order to handle the uncertainty of the volatile landmarks and process navigation queries, we introduce a specialized probabilistic timed automata (PTA) [12] (§IV). Extensive experimentation at two different cities reveal that UrbanEye provides more than 50% better accuracy compared to the existing outdoor localization system, Dejavu, while consuming substantially less amount of energy. Moreover, UrbanEye surprisingly gives a little better estimate of travel time compared to popular vis-a-vis power hungry Google map service (§VI).

II. Feasibility Study

The main motivation of this work arises due to the fact that the existing state-of-the-art algorithm on outdoor localization and navigation performs poorly while applied in public buses in aforesaid cities. A possible reason behind such performance can be improper detection of underlying landmarks (anchor-points). Hence we launch an extensive study to assess the feasibility of using various types of landmarks - we consider physical landmarks (turns, speed breakers and bus-stops) and virtual landmarks (Wi-Fi hotspots, GSM handoffs).

Experiment: Volunteers traveled in buses conducting war driving on one route each in Durgapur and Kharagpur carrying five different types of smart-phones. They conducted the experiment for a month (60 trails each covering around 75 Kms across two cities). From the sensor trails signatures corresponding to different landmarks are extracted for analysis.

Observation: We found the turns and speed-breakers are detected with nearly 100% accuracy, hence they are considered to be non-volatile landmarks. However, in few occasions, the sensory signature obtained when the bus was overtaking another vehicle is exactly like a turn. Similarly, on some occasions, potholes are falsely identified as speed breakers, however the buses normally try to avoid them. Some other interesting observations are as following:

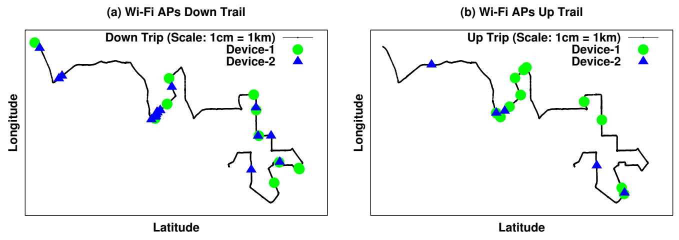

Fig. 1: Bus trail of a route in Durgapur and the positions where Wi-Fi hotspots are detected by different devices

Wi-Fi: Fig. 1(a) and Fig. 1(b) respectively show a representative plot of detected Wi-Fi signals along up and down trails on a schematic 2-D map (the trail is of a particular bus route in Durgapur). It can be observed that the availability of Wi-Fis is very sparse along a route, also the Wi-Fi hotspot observed by the two different devices are mostly not in the same place, making Wi-Fi hotspot an unreliable landmark to consider. We also found on three occasions (out of 60) users boarded the bus with their Wi-Fi hotspot enabled which completely disturbed the Wi-Fi spot identification process. We calculated the correlation between the locations of the detected hotspots by two different devices on same trips, which is 0.35 for the down trip and 0.11 for the up trip respectively. Also the correlation of the detection by the same device calculated over all (up and down) trips are 0.27 and 0.11 respectively for Device-1 and Device-2.

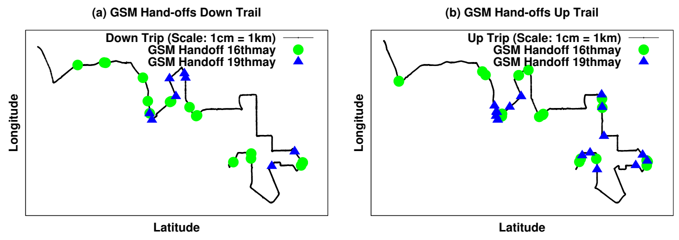

Fig. 2: Bus trail of a route in Durgapur and the positions where GSM handoffs are experienced in two different days

GSM handoff: Fig. 2(a) and Fig. 2(b) respectively show a representative plot of detected GSM handoffs along up and down trails (same bus route as previous case) on a schematic 2-D map on two different days. Similar to Wi-Fi hotspots, it is seen that the positions, where GSM hand-off occurs, differ from day to day. GSM hand-offs too show low correlation even when calculated over the same network. We have calculated the correlation between the place detected for hand-offs by the same device with same network provider on the same route on different days. This value is 0.08 for the down trip and 0.35 for the up trip. The difference occurs not only due to varying weather conditions but we believe due to unplanned installation of cell towers, thus handoffs occur under a region and not at a specific point as signal strength of towers vary.

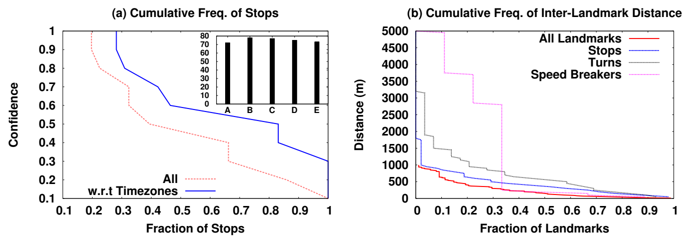

Fig. 3: (a) Cumulative fraction of stops w.r.t. confidence (Inset: Percentage of stops detected by different devices), (b) Cumulative fraction of landmarks (speed breakers, turns, stops) available at a given distance

Stops: A point (x, y) in Fig. 3(a) represents the fraction of stops (x) that has appeared in y-fraction of trails or more. The ‘red (lower)’ curve is drawn considering all possible trails, whereas we bin the trails into four bins according to the time of the day. For each stop, we take the highest fraction among the bins and plot the ‘blue’ curve. The ‘red’ curve shows that in general a good fraction (40%) of bus stops are detected with fair (0.6) degree of confidence. The ’blue’ curve shows that if the time of travel is known, the overall detection accuracy improves significantly, say, 70% detected with 0.6 confidence. It is because a bus occasionally skips some designated stops depending on a particular time of the day (non-busy hours). Hence stoppage patterns of buses has strong temporal features. Fig. 3(a) inset shows that the accuracy in detecting a bus stop is almost independent of the quality of the smart-phone sensors used (A (Xiaomi Mi4i), B(Moto G2), C(Google Nexus7), D (OnePlusOne), E(Yureka)).

Lessons learned: Road-side Wi-Fis and GSM handoffs show very high degree of unreliability, hence cannot be used as landmarks for outdoor localization. Stops though volatile show certain patterns. Turns and speed-breakers are always recognized but there are certain amount of false positives present. Fig. 3(b) shows that the distance in-between two bus stops is the least. Hence, although volatile, using the information of stops can significantly increase the density of landmarks which is a primary requirement for designing accurate localization scheme. Fig. 3(b) also shows that the mean distance between any two landmarks is reduced drastically (almost 500m) with the inclusion of stops. So the challenges are to design a novel efficient model to represent and utilize both the volatile and non-volatile landmark information and minimize (eliminate) the cases of false detection of landmarks.

III. System Overview

The general framework of UrbanEye has two broad modules – (1) creation of landmark database through war-driving, and (2) processing of database and on-line navigation.

A. Creation of Landmark Database

As mentioned earlier, we consider two types of landmarks - *volatile* (stops) and *non-volatile* (speed breakers and turns). Each landmark is essentially a signature which is computed from the smart-phone sensor readings. We use three primary sensors for landmark detection - accelerometer, gyroscope and compass. UrbanEye maintains a landmark database for storing and maintaining landmark information over different bus routes, as collected through war-driving. During war-driving, we collect the *landmark position* information via GPS, and store it in a landmark database server. It can be noted that the position is an one time information which is collected for localizing the bus position during navigation. Apart from the position information, we collect sensor signatures for a landmark, time zone when the landmark is detected and mean and deviation of travel time between two landmarks.

B. Processing of Landmark Database and On-line Navigation

Due to multiple bus routes in a city, the size of the landmark database may be very large, although we have to process it in real time. For this purpose we require an efficient data structure to pre-process and store the information in a format which can be useful for fast processing of navigation queries. Because of the volatility in the system, the navigation queries require complex probabilistic processing to find out the *probable location of the user* and the *travel time to the destination* (TTD). We use PTA to process the landmark information and to execute navigation queries. PTA combines the temporal as well as the probabilistic nature of the transition between two states, that make it a suitable data structure for our system.

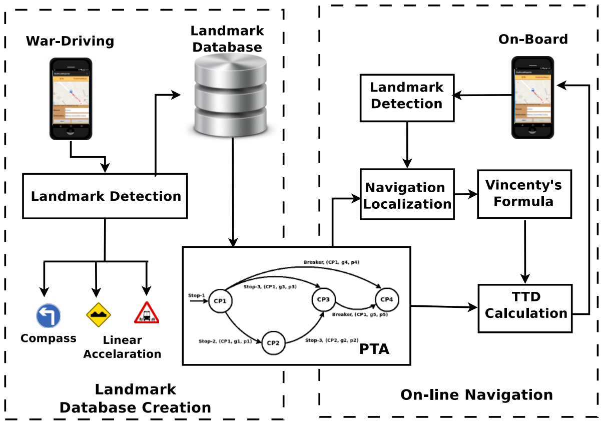

Fig. 4: UrbanEye Architecture with three design components

In UrbanEye, the user fixes the $\{$source, destination$\}$ pair, and the server generates a PTA of the bus routes between the specified source and destination bus-stops. The navigation module enumerates the PTA on detection of a landmark, and finds out the position of the bus over the route. We apply Vincenty’s formula [13] to approximate the location of a bus based on the previous landmark detected. Furthermore, the TTD is adjusted on-the-fly based on the running statistics of the bus and the historical statistics available with the PTA. During on-line navigation, the user also contributes in updating few information in the landmark database, like the mean travel time between two landmarks, and the landmark confidence values.

Fig. 4 shows the general architecture of UrbanEye system. It has three primary components - landmark database creation from war-driving, generation of the PTA from the landmark database, and on-line navigation. The detailed design of the system has been discussed in the subsequent sections.

IV. UrbanEye: Landmark Detection and Processing of Landmark Database

In this section, we discuss the procedures for landmark detection, landmark database creation and formation of PTA from the landmark database.

A. Landmark Detection

We use three primary sensor readings for landmark detection - accelerometer, gyroscope and compass. The system computes direction from compass, and the linear acceleration is internally computed by accelerometer and gyroscope. In a global axis space, we assume that the bus moves at the Y-axis direction, and the vertical axis is the Z-axis. Based on the sensor readings across different axes, the detection of turns, speed breaks and stops works as follows.

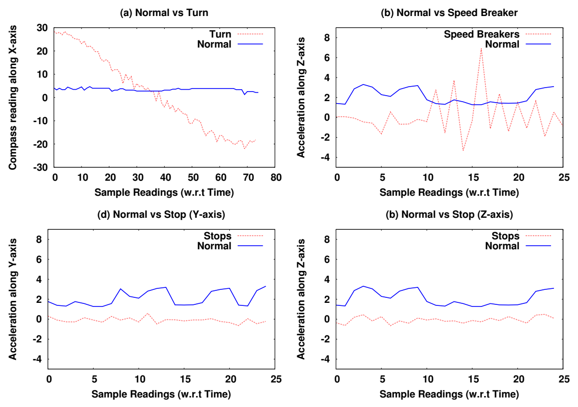

Fig. 5: Detection of Landmarks from Sensor Readings

*1) **Turns**:* We observe that very high variance for the X-axis reading of the compass signifies a turn. Fig. 5(a) shows a comparison between compass reading for a turn and the normal movement of the bus. Starting from the initial readings, the compass trace is processed in form of sub-traces where the duration of every sub-trace is $\mathcal{L}$ seconds. Suppose C_0^X and $C_\mathcal{L}^X$ are the compass readings along the X-axis at the beginning and at the end of a sub-trace, respectively. $\mathcal{T}_C$ is the compass reading threshold for detecting a turn. The system detects a turn if $|C_\mathcal{L}^X - C_0^X| > \mathcal{T}_C$ and $\frac{C_\mathcal{L}^X}{C_0^X} < 0$.

*2) **Speed Breakers**:* Speed breakers are also non-volatile landmarks that show a peak in the acceleration value and high variance at Z-axis. Fig 5(b) shows a comparison between linear acceleration for a speed breaker and that during the normal movement. Similar to the previous case, the accelerometer trace is divided into continuous sub-traces of length $\mathcal{L}$. Let $\mathcal{A}_p^Z$ and $\mathcal{A}_n^Z$ be the accelerometer peaks at positive and negative Z-axis directions, respectively. Then a speed breaker is detected if $|\mathcal{A}_p^Z - \mathcal{A}_n^Z| \geq \mathcal{T}_A$, where $\mathcal{T}_A$ is the accelerometer threshold for detecting a speed breaker.

*3) **Stops**:* Stops are the volatile landmarks and most chaotic of all. We make use of the linear acceleration values in the Y-axis and Z-axis for detecting the stops. Fig. 5(c) and Fig. 5(d) show a comparison between linear acceleration for a stop and that during the normal movement, across Y-axis and Z-axis. When there is a stop, the linear acceleration value in the Y-axis goes near to zero, and the Z-axis shows little deviation near zero (because of the bus vibration). These signatures help us to detect a stop.

Let there be n entries in the acceleration sub-trace, and $\mathcal{A}_i^Y$ and $\mathcal{A}_i^Z$ be the i^{th} entry in the linear acceleration values across Y-axis and Z-axis, respectively. The system detects a stop if $\mathcal{A}_i^Y \approx 0; 0 < \mathcal{A}_i^Z < 1, \forall\, i \in [1, n]$.

*4) **Setting the Parameters $\mathcal{T}_C$, $\mathcal{T}_A$ and $\mathcal{L}$**:* The impact of $\mathcal{T}_C$, $\mathcal{T}_A$ and $\mathcal{L}$ on the detection of turns and speed breakers is shown in Table I where the table values show the accuracy

TABLE I: Threshold calculation

$\mathcal{L}$ (Sec)	Detection % For Different Threshold Used						
	Turns ($\mathcal{T}_C$)				Breaker ($\mathcal{T}_A$)		
	10	20	25	30	3	4	5
2	0	9.38	40.63	65.63	33.33	44.44	33.33
3	3.13	43.75	59.38	**96.8**	**88.89**	77.78	44.44
4	21.88	56.25	65.63	87.5	88.89	77.78	33.33
5	31.25	62.5	71.88	81.25	77.78	77.78	44.44

for a particular landmark detection. Each value is an average of all the instances of that event observed during war-driving. The table indicates that at $\mathcal{T}_C = 30$ and $\mathcal{L} = 3sec$, turn is most accurately detected, whereas speed breakers can be detected best at $\mathcal{T}_A = 3$ and $\mathcal{L} = 3sec$. Therefore, we use these values of thresholds in UrbanEye.

*5) **Initial Axis Orientation for Sensor Measurement**:* In a global axis space, we assume that the bus moves in the Y-axis, with other axes accordingly oriented. However the orientation of the smart-phone and that of the bus may not be in sync. We resolve this problem by taking the components of gravity into consideration. We observed that, whatever be the orientation, the axis which has the component of gravity in the acceleration value, has a mean value close to the "acceleration due to gravity" (usually around $9.5m/s^2$). Considering this observation, we decide the Z-axis from the acceleration data of the user traces.

B. Creation of Landmark Database

The landmark database is created based on the sensory detection of different landmarks through war-driving. During war-driving, we collect the landmark position information via GPS, and store this in a landmark database server. The database stores the following information about a landmark;

- `landmark` : The actual landmark.
- `landmark-from` : The landmark from which this landmark was reached.
- `position` : The position of the landmark, obtained through war-driving.
- `time-zone` :We follow a simple rule of dividing the day into four time-zones of six hours each. This information is required to capture the temporal behavior of volatile bus-stops.
- `tt-mean` : The mean travel time between two consecutive landmarks detected in a trail during war-driving. This is computed as the mean of multiple trail collections.
- `tt-sd` : This field stores the standard deviation of the travel time between two consecutive landmarks detected during war-driving over the multiple trails.
- `confidence` : This parameter keeps track of the detection probability of a volatile landmark. The confidence is defined as a conditional probability $\mathcal{P}(l_i|l_j)$ that denotes the probability of detecting landmark l_i, given that the previous landmark detected was l_j. Numerically, this is computed as $\mathcal{P}(l_i|l_j) = \mathcal{P}(l_i \wedge l_j)/\mathcal{P}(l_j)$. From the war-driving trails, we compute the instances when both l_i and l_j are detected as successive landmarks ($n(l_i \wedge l_j)$), and

the instances when only l_j is detected ($n(l_j)$). Let n_t be the total number of trails. Then,

$$\mathcal{P}(l_i \wedge l_j) = \frac{n(l_i \wedge l_j)}{n_t}; \;\; \mathcal{P}(l_j) = \frac{n(l_j)}{n_t}$$

It can be noted that although the initial database information is generated via war-driving, every detection of a landmark during the application usage also updates the database if there is at-least 5% change in a parameter. This is required to capture the long term effect of the landmark information changes.

C. From Landmark Database to PTA

A PTA is an automaton with a set of states and transitions where every transition is associated with a probability and guarded by a time value. Formally a PTA $\mathcal{P}$ is a six tuple automaton, $\mathcal{P} = \{S, s_{\text{init}}, \mathcal{X}, inv, prob, \mathcal{L}\}$, where S is a set of states with the initial state $s_{\text{init}} \in L$, a finite set $\mathcal{X}$ of clocks, a function $inv : L \rightarrow \Xi_{\mathcal{X}}$, where $\Xi_{\mathcal{X}}$ is a set of clock constraints associating a time guard with each location, a finite set $prob$ with probabilities associated with each transitions, and a labeling function $\mathcal{L} \rightarrow 2^L$. Graphically, a PTA looks like a state transition diagram with every transition associated with three tuples $< l, g, p >$ where l is the input label for the transition, g is the guard time and p is the transition probability. If there is a transition between two states S_1, S_2 with this tuple defined as $< l_{12}, g_{12}, p_{12} >$, then on observation of l_{12}, the transition is fired from S_1 to S_2 with probability p_{12}, only if the elapsed time is in-between $g_{12} \pm \Delta$ where Δ is a tolerance factor.

It is interesting to observe that PTA has a natural similarity with the navigation system we are considering. The states can be represented as different checkpoints[1] over the road and encountering a landmark may trigger a transition between two consecutive states. However this transition occurs with some probability (`confidence`) in case the subsequent landmark is volatile and with complete certainty for non-volatile landmarks. Furthermore, the mean time to travel between two successive landmarks (`tt-mean`) can be used as the timer guard to avoid false transition.

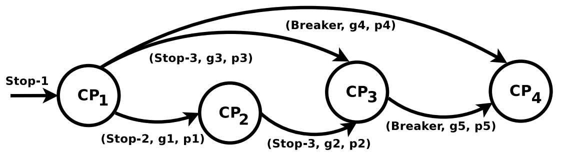

Fig. 6: An example PTA of a route

An example instance of the PTA is given in Fig. 6. Let, in a route there be sequence of landmarks {stop-1, stop-2, stop-3, breaker}. The user boards the bus from stop-1, and initiates the PTA from CP_1. Stop being a volatile landmark, the user may skip either or both of stop-2 and stop-3, and may directly reach the next checkpoint of detecting a breaker. The PTA is formed accordingly. From, CP_1, we have three possible options: CP_2 (detect stop-2), CP_3 (skip stop-2 and detect stop-3) and CP_4 (skip stop-2, stop-3 and detect the breaker).

[1]The checkpoints are the positions associated with every landmark. Once we detect a landmark, we say that the bus has reached at a particular checkpoint on its travel route.

Given a {source, destination} pair, the system builds the PTA as follows;

- The source is taken as the first checkpoint and the initial state whereas the destination is considered as the final state or the 'goal' state of the PTA.
- Between a {source, destination} pair, the system enumerates all the bus routes from the landmark database, and every landmark position is considered as a new checkpoint.
- Between two checkpoints with non-volatile landmarks, if there exists one or more volatile landmarks, then the system generates all possible transitions considering that the volatile landmarks may remain undetected, as shown in Fig. 6.

Once we have the backbone system in hand, we proceed for the navigation module as discussed in the next section.

V. URBANEYE: NAVIGATION AND TRAVEL TIME ESTIMATION

The navigation is initiated by a user by specifying the {source, destination} pair via the UrbanEye user interface. The {source, destination} pair is forwarded to the UrbanEye server, and the server builds the PTA and offloads it to the user's smart-phone.

A. On-line Localization of Bus Navigation

The detection of a bus navigation path is basically a satisfiability problem of the PTA to match an input landmark sequence $\{l_1, l_2, l_3, ...\}$ starting from the initial state (checkpoint corresponding to the originating bus stop) to the final state (checkpoint corresponding to the destination bus stop). Let, after crossing checkpoint CP_i (which corresponds to landmark l_k, say), the user detects another landmark l_{k+1} after time τ. Due to the volatility in landmarks, there may be more than one out-going edges from CP_i. Let there be q outgoing edges, and the checkpoints associated with those edges are $CP_{j1}, CP_{j2}, ..., CP_{jq}$. The system makes a transition from $CP_i \rightarrow CP_{jr}$; $r \in [1...q]$ (CP_{jr} is detected as the checkpoint corresponding to landmark l_{k+1}), if the following conditions are satisfied;

C-I. the label of the edge $CP_i \rightarrow CP_{jr}$ is l_{k+1}; and
C-II. $g_{i,jr} - g^{\text{sd}}_{i,jr} \leq \tau \leq g_{i,jr} + g^{\text{sd}}_{i,jr}$, where $g_{i,jr}$ is the guard interval of the edge $CP_i \rightarrow CP_{jr}$, and $g^{\text{sd}}_{i,jr}$ is the standard deviation of the guard interval obtained from `tt-sd` stored in the landmark database.

Handling sensory and environmental noise: However, due to sensory or environmental noise, there may be error in landmark detection. There can be two possibilities,

- The system may detect a false positive landmark due to sensory or environmental noise (ex., a turn may be detected due to overtaking)
- The actual travel time of the bus (τ) may significantly differ from the mean, due to environmental noise (ex. traffic congestion)

Significant deviation of τ from the mean travel time may result in the triggering of a false transition in the PTA. To eliminate such false transitions, we have to adjust the τ value based on the observation from bus running statistics. It can be noted that such behaviors are not regular and are outliers for the system. Therefore, these data points should not contribute in `tt-mean`, although we need to adjust τ for the detection of the next checkpoint. As the bus velocity becomes zero at every stop, therefore the deviation in acceleration results in the deviation of τ value.

The acceleration deviation is not regular, and we assume that it follows some unknown distribution with mean $-1 \ll \epsilon \ll 1$. The ϵ value is calculated from the normalized deviation in acceleration of the target patch. Let, τ_{adj} be the adjusted travel time and a be the average linear acceleration measured in a patch during run-time from the accelerometer and gyroscope. Assume that d is the distance of that target patch, which can be computed from the `position` of the landmarks. As the initial velocity is zero, we can write;

$$\frac{1}{2}a(1+\epsilon)\tau_{\text{adj}}^2 - d = 0 \tag{1}$$

Then with first order approximation of ϵ, τ_{adj} is obtained as,

$$\tau_{\text{adj}} = \sqrt{\frac{2d}{a}}\left(1 - \frac{\epsilon}{2}\right) \tag{2}$$

We compare the value of τ_{adj} with the guard interval, as given in condition (C-II) above, to detect a checkpoint corresponding to a landmark. This eliminates the problem of detecting volatile landmarks in the presence of chaotic bus movements. Further, the guard interval also eliminates false positive landmarks detected as a result of sensory or environmental noise.

B. Estimation of Bus Position

Whenever the bus reaches a checkpoint, the present position of the bus is updated as the location of that checkpoint. To estimate the bus position between two consecutive checkpoints, we use the *Vincenty's formula* [13]. Vincenty's formula takes the displacement of any moving object, its bearing angle (i.e. the orientation) and the previous position in terms of latitude and longitude as input. In our scenario, the initial point is the location of the last checkpoint of the bus. The bearing angle is calculated from the sensor readings. Further, we use the double integration approach over the acceleration readings to calculate the distance. The Vincenty's formula returns the new position of the bus which is updated in UrbanEye user interface.

Further, it is possible that the bus may skip a volatile landmark. In such a case the bus position is readjusted on detection of next non-volatile landmark. The in-between location is approximated using Vincenty's formula, as discussed above.

C. Estimation of Expected TTD

With the localization work done, the next task that remains is estimating the TTD from the current checkpoint. Assume that there are n number of checkpoints between the initial checkpoint and the final checkpoint. Further assume that the bus has already passed m checkpoints. Then the estimated TTD ($\mathcal{T}_{est}$) is derived as follows;

$$\mathcal{T}_{exp} = \sum_{i=m}^{n-1} p_{(i,i+1)} * g_{(i,i+1)} \tag{3}$$

TABLE II: Characteristics of two routes in the two cities. Kharagpur is richer in terms of non-volatile landmarks, has a better landmark density and hence low mean localization error.

City	Durgapur	Kharagpur
Distance Covered(in kms)	23	10
Total Landmarks	89	50
Turns (Detected in Avg.)	32(31)	9(9)
Speed Breakers (Detected in Avg.)	10(9)	33(31)
Stops (Detected in Avg.)	49(38)	8(7)
Avg. Landmark density (per km)	3.8	5
Avg. Localization Error (mts)	50	42

VI. Evaluation

In this section, we explain the experiments in details along with the developed applications and the competing algorithm. Next we show the experimental results demonstrating the performance of UrbanEye.

A. Experimental setup

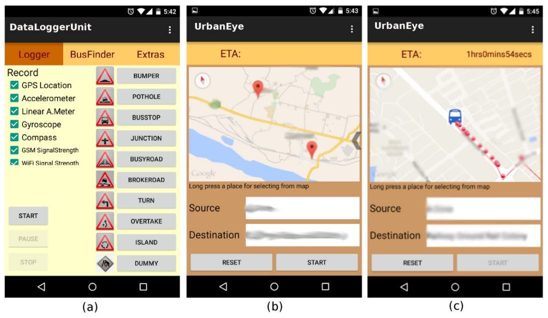

Fig. 7: Application Screen-shots: (a) Data Logging Application (b) and (c) UrbanEye

In our experiment, one of the key components is the landmark data acquisition, which we implement by war-driving. We develop an android application which logs data from different inertial sensors available on the phone along with GSM hand-offs and Wi-Fi APs. The application contains separate buttons for manual annotation of the landmarks while taking trail on a bus. In the experiment, we take a bus and start logging data using the application, and whenever we see that the bus is encountering a landmark (such as a speed breaker or a turn), we annotate the corresponding landmark through the application. We collect a month time sensor trails for up and down trips (60 trails) for two routes; one each in Durgapur and Kharagpur. Total coverage of the routes is ≈ 75 kms. These data have been further used for the creation of the database and also for learning the signatures for different landmarks. The trails were taken with the help of multiple smartphones namely, Google Nexus4, Micromax A092, Samsung Galaxy

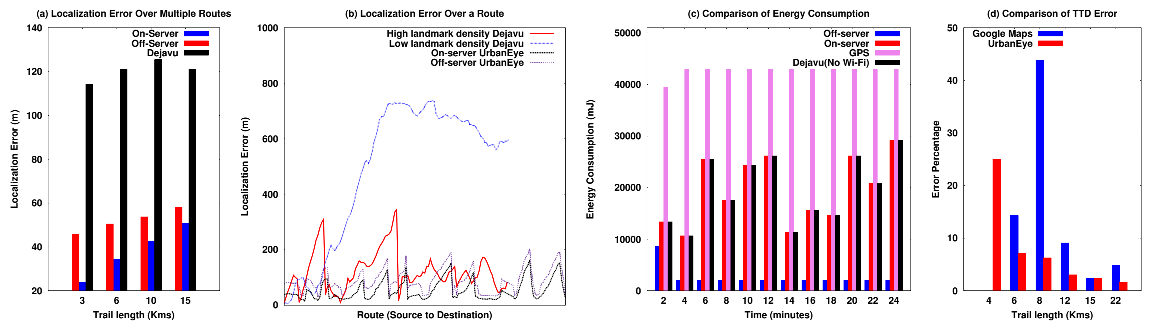

Fig. 8: Evaluation Results: (a) and (b) Evaluation of localization error, (c) Evaluation of energy consumption and (d) Evaluation of TTD error

Tab 3, Xiaomi Mi4i, Motorola MotoG2, Google Nexus7, OnePlus One and Yu Yureka . Fig. 7a shows the user interface of this application.

UrbanEye basically works in two phases. The first being the initialization of the user. Fig. 7(b) shows the interface of this phase. The user selects her source and destination from the Google map embedded in the application. UrbanEye calculates the stops closest to these locations (to create the corresponding PTA) and reports the TTD to the user. Next, in the second phase, navigation starts; we develop two versions of the application depending on the choice of availing the Google API assistance. First of all, the application downloads the PTA from the server to the local device. Next the device samples the user traces every 3 seconds and locally runs the landmark detection and searching algorithm. If the algorithm detects a landmark, then its corresponding location (obtained from the PTA) and the modified TTD is reported to the user. But if no landmark is detected, then the next position is computed using the Vincenty's Formula [13]. In this version, which we call *off-server version*, we do not use Google Map API. In the second version which we term *on-server version*, to make the new position less error prone, the application invokes the snapToRoads Google API to further calibrate the predicted position. Fig. 7(c) shows a screen-shot of this phase. The icon of the bus shows the current location and the red dots show the previous locations.

Competing heuristics: We have taken Dejavu [1] as a competing algorithm. Since Dejavu is not publicly available we have re-implemented the Dejavu system. As a sanity check, we have verified the performance of the implemented system under proposed (in Dejavu's paper) level of density of landmarks and have found it to exhibit similar range of error as reported in the paper.

Dejavu heuristics: Dejavu is a localization approach implemented for the private car transport system. It performs localization on the basis of anchor points located all over the route. It uses inertial sensors say Accelerometer, Gyroscope and compass for the detection of physical anchors like speed breakers, turns, tunnels, bridges etc. and uses Wi-Fi Access Points and GSM towers as virtual anchors. However tunnels and bridges are very difficult to detect in our scenario as they heavily rely on road conditions and GSM signal strength which are extremely poor in the aforesaid cities. Dejavu algorithm starts with an initial latitude and longitude, takes readings from different sensors on the smart-phone for a time window and dispatches it to the server whereby the server deduces the next latitude and longitude.

B. Experimental results

In experimental section, we first present UrbanEye's performance vis-a-vis Dejavu and then analyze various features of UrbanEye to understand the reason behind its superior performance.

1) Comparison in terms of Localization: We run UrbanEye on one route each in Durgapur and Kharagpur respectively, from which we have also taken sub-routes of varying lengths of 3, 6, 10 and 15 kms to illustrate our results. Fig. 8(a) shows that UrbanEye clearly outperforms Dejavu in term of localization error. However, the localization error increases with the increase in the length of the route, the worst case average localization error is around 50m for the 15km route. The result also illustrates the fact that assistance of Google API reduces the localization error, nevertheless, absence of the API also leads to a decent performance; absence of Google API in a 15km trail results in a localization error of around 57m. Fig. 8(b) shows the variation of localization error over a complete route. This is interesting to observe that UrbanEye performs exceptionally well (localization error around 45m) in a route with low landmark density, even without Google API assistance (localization error around 60m) whereas in Dejavu, average localization error is found to be around 120m **(**50% **less than UrbanEye off-server)** in the dense patch (with more number of landmarks per km) and around 524m in the sparse patch in the route.

2) Comparison in terms of Energy Consumption: We illustrate the performance of UrbanEye in the light of energy consumption. Since Dejavu makes use of same sensors as UrbanEye, in the context of energy, the GPS based system becomes a major contender. We measure the cumulative consumption after every 2 minutes. Fig. 8(b) shows that both on-server version of UrbanEye and Dejavu consume significantly less amount of energy compared to GPS at every interval.

Overall the energy consumed by our system was 235.25J over the trail while GPS consumes almost double, i.e. 508.61J. Important to note that the off-server version of UrbanEye consumes really mini-scale amount of energy (decreases by 86%) and we do not sacrifice too much on accuracy.

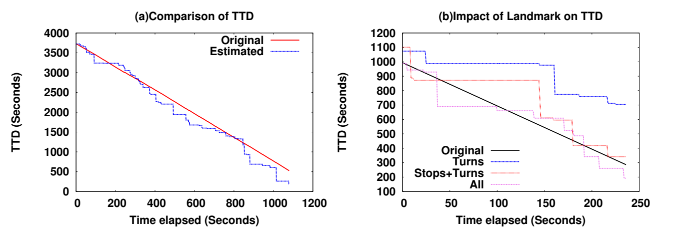

Fig. 9: Evaluation of TTD: (a) w.r.t. actual data and (b) Impact of inclusion/exclusion of landmarks

3) Comparison in terms of Time to Destination (TTD): We evaluate UrbanEye in terms of TTD against two different metrics (a) service offered by the Google map (b) ground truth: the actual time taken to reach the destination. Google map is a resource sensitive but popular tool to estimate the TTD. In Fig. 8(d), we show that UrbanEye gives a little better estimate of public navigation compared to Google map. The Google map surprisingly performs poorly because it cannot estimate the wait-time of a bus in various stops which is sometimes absolutely erratic - UrbanEye maintains history of that and accordingly adjusts.

Fig. 9a shows that UrbanEye quite accurately estimates the actual travel time to destination. The maximum deviation is observed to be around 7 minutes in a small stretch and the mean error is about 2.5 minutes. The deviation of 7 minutes that we see was actually observed in a patch in the route, due to scarcity of landmarks.

The above results shows superior performance of UrbanEye compared to Dejavu and Google maps in terms of achieving much better localization accuracy and TTD estimate, respectively. In the following subsections (VI-C to VI-E) we study the impact of different category of landmarks in achieving the performance gains. Especially we are interested to ponder over the importance of volatile landmark stops over the non-volatile ones.

C. Performance evaluation - Detection of landmarks

The accuracy of UrbanEye depends upon how accurately it identifies both non-volatile and volatile landmarks. In Table III, we show the accuracy of the detection strategies evaluated over a route. As can be seen, turns and speed-breakers have very high detection percentage. However, note that the number of false positive cases whereby other events wrongly identified as turns or speed-breakers are also high. This problem is efficiently tackled by the PTA construction where the concept of guard time enables the elimination of such false positives, we see that almost all the false positives are removed in case of turns and speed-breakers. As there are many volatile stops so it is expected that not all the stops in a trail will be detected - the accuracy is around 80%. On the other hand, several noisy stops or jams add up to the false positive cases. Since already there are misses, the systems performance regarding removal of false positives in case of stops slightly deteriorates.

TABLE III: Accuracy of Landmark Detection; FP: False positives, FPE: False positives after elimination by PTA guard intervals

Landmark	Actual	Detected (%)	FP (%)	FPE (%)
Turn	32	31(96.8)	10(31.2)	0(0)
Speed Breaker	9	8(87.5)	3(37.5)	0(0)
Stops	42	34(80.9)	13(38.2)	2(4.76)

D. Performance evaluation - Impact of Landmarks in TTD estimation

We analyze the impact of individual landmark on the TTD estimation and on the localization error and extract the most important landmarks for localization. First we consider only turns in a trail and perform the experiments over a smaller stretch of the trail. We then repeat the same experiment considering turns and stops together, and so on. Fig. 9(b) shows a comparison of TTD for these 4 trails. As per intuition, it can be easily observed that inclusion of landmarks improves the time estimation considerably. Considering only turns, the mean error in time estimation is more than 10 minutes which reduces to 7 minutes with inclusion of stops. This further reduces to 4 minutes on including all three (turns, stop and speed breakers) together.

E. Performance Evaluation - Localization Error between two landmarks

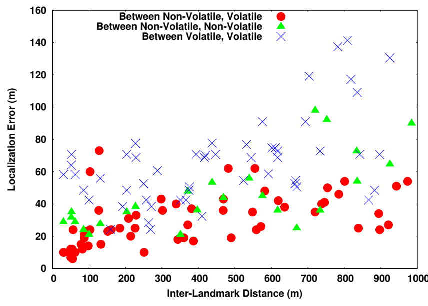

Fig. 10: Impact of Inter-landmark Distance over Localization Error

We analyze two aspects here (a) the impact of inter-landmark distance on error and (b) the impact of the nature of consecutive landmark pair. Fig. 10 draws the scatter plot considering three types of pair (both landmarks volatile (V-V), both being non-volatile (NV-NV) or a volatile-non-volatile pair (V-NV)). The impact of inter-landmark distance is clearly visible; the localization error increases with distance. The performance rank of the three sets are - V-NV pair being the best, but NV-NV pair performs a little worse. The reason is a bus doesn't stop in speed-breakers or turns as a result the speed is not exactly adjusted in the database. The performance of V-V is worst because there are several false negatives whereby a quick stoppage is missed accumulating large error.

The insights drawn from the above results can be summarized as following: we observe superior performance of UrbanEye (both on-server and off-server) compared to Dejavu in terms of achieving much better localization accuracy both in on-server and off-server versions. However, UrbanEye (on-server) and Dejavu consumes almost same energy. But UrbanEye (off-server) consumes significantly less energy than Dejavu with a slightly reduced accuracy. Secondly, compared to Google maps, UrbanEye provides a much better estimate of TTD, which is almost close to the actual time taken. It shows that the PTA successfully modeled both the temporal and spatial irregularities. Thirdly, the guard time is able to completely eliminate the cases of false detection for non-volatile landmarks and significantly reduce the number for volatile landmarks (stops). Fourthly, the inclusion of volatile landmarks (a novel feature of our framework) has significant impact in performance gains.

VII. State of the Art

There are number of works to resolve the issues of GPS through the use of inertial sensors. Works like *Compacc* [9], *smartNavi* [11] have made good use of the inertial sensors for outdoor pedestrian localization. While Compacc proposes use of accelerometer, compass with Google maps API to help pedestrian localization, smartNavi offers an efficient GPS-independent application which detects every step made and maps the position in corresponding direction. However, both the techniques fail miserably for other mode of transportation due to the step-based algorithm. *HowLongToWait* [5] uses cell-phone tower information (cellid) along with the inertial sensors for arrival time prediction in vehicular navigation. *Dejavu* [1] includes Wi-Fi in addition to other sensors previously mentioned for private navigation systems.

Landmark detection, which is one of the foundations of our work, has been worked upon by a few works like *Pothole patrol* [14] and *Wolverine* [15]. Pothole patrol analyses road conditions to detect potholes using external sensors which are pretty accurate but bring in extra hardwares. Whereas, Wolverine detects road congestion and speed breakers through the use of sensors available in smart-phones. *Easytracker* [6] uses GPS logs for identifying the landmarks but it will fail as well for developing countries where local ground truth information is missing on the map. *Bumping* [16] method is used to improve the positioning accuracy by exploiting the information of speed bumps readily available in parking garages which can be used as a means for the inertial navigation systems to correct its position and velocity. Bump matching algorithm provides an accuracy of around 5 meters, which matches the detected landmark to the right bumps.

VIII. Conclusion

The public good for developing countries clearly demands an assisting technology for transport navigation. In this paper, we exploit the advantages of growing smart-phone penetrations in such countries to design an assisting technology for bus localization and travel-time prediction. We designed a system based on inertial sensors only which is significantly less power-hungry than GPS. The proposed technology, UrbanEye, constructs a navigation map considering different physical landmarks at the routes, both volatile and non-volatile. The non-volatility and irregular bus movements are tackled using probabilistic timed automata. The extensive test runs of UrbanEye at two different cities show that the off-server version of UrbanEye performs significantly better than Dejavu but also requires much less energy.

Acknowledgement

The authors would like to thank Information Technology Research Academy (ITRA), Department of Electronics and Information Technology (DeitY), Government of India, for supporting this work under the research project "Post disaster situation analysis and resource management using delay tolerant peer to peer wireless networks (DISARM)", sanction letter number and date ITRA/15(58)/MOBILE/DISRAM/01, Dt. 19-09-2013.

References

[1] H. Aly and M. Youssef, "Dejavu: An accurate energy-efficient outdoor localization system," in *Proc. of SIGSPATIAL'13*. New York, NY, USA: ACM, 2013, pp. 154–163.

[2] "Transport in India," https://en.wikipedia.org/wiki/Transport_in_India.

[3] R. Mandal, N. Agarwal, S. Nandi, P. Das, A. Anvit, S. Sanyal, and S. Saha, "Stoppage pattern analysis of public bus GPS traces in developing regions," in *Proc. of PerCom '15*, 2015, pp. 276–279.

[4] M. Youssef, M. Yosef, and M. El-Derini, "Gac: Energy-efficient hybrid gps-accelerometer-compass gsm localization," in *Proc. of IEEE GLOBECOM '10*, Dec 2010, pp. 1–5.

[5] P. Zhou, Y. Zheng, and M. Li, "How long to wait? predicting bus arrival time with mobile phone based participatory sensing," *IEEE Transactions on Mobile Computing*, vol. 13, no. 6, pp. 1228–1241, June 2014.

[6] J. Biagioni, T. Gerlich, T. Merrifield, and J. Eriksson, "Easytracker: Automatic transit tracking, mapping, and arrival time prediction using smartphones," in *Proc. of SenSys '11*. New York, NY, USA: ACM, 2011, pp. 68–81.

[7] T. Stockx, B. Hecht, and J. Schöning, "Subwayps: Towards smartphone positioning in underground public transportation systems," in *Proc. of ACM SIGSPATIAL '14*. New York, NY, USA: ACM, 2014, pp. 93–102.

[8] A. Thiagarajan, J. Biagioni, T. Gerlich, and J. Eriksson, "Cooperative transit tracking using smart-phones," in *Proc. of ACM SenSys '10*. New York, NY, USA: ACM, 2010, pp. 85–98.

[9] I. Constandache, R. Roy, and C. I. Rhee, "Compacc: Using mobile phone compasses and accelerometers for localization," in *Proc. of INFOCOM '10*, 2010.

[10] H. Wang, S. Sen, A. Elgohary, M. Farid, M. Youssef, and R. R. Choudhury, "No need to war-drive: Unsupervised indoor localization," in *Proc. of MobiSys '12*. New York, NY, USA: ACM, 2012, pp. 197–210.

[11] "SmartNavi," http://smartnavi-app.com/home.

[12] M. Kwiatkowska, G. Norman, J. Sproston, and F. Wang, "Symbolic model checking for probabilistic timed automata," in *Modelling and Analysis of Timed and Fault-Tolerant Systems Formal Techniques*. Springer, 2004, pp. 293–308.

[13] T. Vincenty, "Transformation of co-ordinates between geodetic systems," *Survey Review*, vol. 18, no. 137, pp. 128–133, 1965.

[14] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan, "The pothole patrol: using a mobile sensor network for road surface monitoring," in *Proc. of MobiSys '08*. ACM, 2008, pp. 29–39.

[15] R. Bhoraskar, N. Vankadhara, B. Raman, and P. Kulkarni, "Wolverine: Traffic and road condition estimation using smartphone sensors," in *Proc. of COMSNETS '12*, Jan 2012, pp. 1–6.

[16] G. Tan, M. Lu, F. Jiang, K. Chen, X. Huang, and J. Wu, "Bumping: A bump-aided inertial navigation method for indoor vehicles using smartphones," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, pp. 1670–1680, 2014.