VJAĠĠ – A THICK-CLIENT SMART-PHONE JOURNEY DETECTION ALGORITHM

PREPRINT

Michael P. J. Camilleri
School of Informatics
University of Edinburgh, UK

Adrian Muscat
Faculty of ICT
University of Malta, Malta

Victor Buttigieg
Faculty of ICT
University of Malta, Malta

Maria Attard
Institute for Climate Change & Sustainable Development
University of Malta, Malta

April 3, 2020

ABSTRACT

In this paper we describe *Vjaġġ*, a battery-aware journey detection algorithm that executes on the mobile device. The algorithm can be embedded in the client app of the transport service provider or in a general purpose mobility data collector. The thick client setup allows the customer/participant to select which journeys are transferred to the server, keeping customers in control of their personal data and encouraging user uptake. The algorithm is tested in the field and optimised for both accuracy in registering complete journeys and battery power consumption. Typically the algorithm can run for a full day without the need of recharging and more than 88% of journeys are correctly detected from origin to destination, whilst 12% would be missing part of the journey.

***K*eywords** Location tracking · Journey-identification · GPS · Smartphone · Android development

1 Introduction

Typically, transport planners, researchers and service providers rely on infrequent and expensive travel diary surveys to collect information relative to journeys made during the day, their purpose, mode used and other useful information to derive population travel behaviour. Traditional pen and paper travel diary surveys suffer from a number of limitations. From a logistical perspective, surveys entail a determined commitment on behalf of participants to either recall or continuously track their activities and nevertheless, data is typically inaccurate due to errors in human judgement [1, 2]. Fortunately, the availability of low-cost GPS trackers and the subsequent boom in the smartphone market bootstrapped research and development in the automation of travel data collection that promises to automatically retrieve (i) accurate time-stamps and locations for journey origin and destination, (ii) Trip Path (route) with velocities along the journey and (iii) travel mode and trip purpose, [3, 4, 5, 2]. Additionally, to maximize uptake [6] (especially in the case of voluntary data collection), it is important to (i) avoid costs associated with data transfers; (ii) maintain anonymity and privacy; and (iii) consider battery energy consumption if personal mobile phones are being used.

In this paper we define and implement an algorithm, *Vjaġġ*[2], as an independent automated journey segmentation algorithm embedded as a mobility data collector, for both Android and iOS devices. *Vjaġġ* is able to anonymously and seamlessly collect travel data from participants, using the device's GPS receiver and accelerometer. Mainly, it

We acknowledge financial support from the Vodafone Malta Foundation, under the "Connecting for Good" project. The work was done while Michael P. J. Camilleri was employed with the Faculty of ICT at the University of Malta.

[2] "*vjaġġ*" is the Maltese word for "journey"

automatically segments the GPS trace into whole end-to-end journeys. Additionally, participants are given a summary of all journeys tracked and only those selected are uploaded to the server, guaranteeing full control on personal data. The algorithm is tested for accuracy in the journey detection task and for battery energy efficiency. The algorithm source code is released as open source under GNU GPL-v3.0 license[3].

Prelipcean et. al. [7] summarizes the available technology in automated travel diary collection, in terms of development, distribution and operations costs for both dedicated GPS receivers and smartphones. In the past, dedicated GPS receivers proved to be more cost effective than the use of smartphones, mainly due to the large development cost associated with the latter. However it is argued that open source solutions can lower these costs. MEILI [8] is probably the only open source system available right now. MEILI is a versatile research tool for setting up travel data surveys and computes the trip segmentation and classification on the server side. On the other hand the intention of *Vjaġġ* is to support demand responsive transport (DRT) service providers, such as (with the customer's consent) quantifying missed opportunities. The main focus in developing *Vjaġġ* is therefore on accurate journey detection and on scaling up the collection of daily data for demand forecasting and extending services to new areas and corridors. *Vjaġġ* is therefore implemented and executed on the mobile device, such that data transfer is limited to journeys automatically suggested to and selected by the customer. Notwithstanding, we added trip purpose and mode manual functions such that *Vjaġġ* (embedded as a mobility collector) can be used by transport researchers for travel behaviour studies.

The rest of the paper is organised as follows. We first review related work in the area of automated and battery-aware journey segmentation algorithms (section 2) and then describe the problem of characterising journeys, Section 3. This is followed by a detailed discussion of our battery-aware journey segmentation algorithm, section 4. We then discuss our test setups and results (Section 5), and finally give our conclusions and suggestions for future work.

2 Related Work

We define *journey* (or trip) as the path taken to travel from origin to destination, and a *segment* as a part of a complete journey. In this section we review methods that detect journeys and segments in a GPS trace as well as algorithms that minimise battery energy consumption.

2.1 Detection of Journeys and Segments

Intuitively, given a GPS trace, complete journeys are detected by identifying stops (i.e. GPS segments with zero speed). However stops do occur along journeys, (for example when queuing at junctions, or when changing mode) rendering the task a non trivial one. Various methods have therefore been proposed. The first attempt in detecting journeys and trips solely from a GPS trace is described in [9]. The authors define and make use of *Change Point* segmentation, i.e. where commuters change the transportation mode. Statistically it is shown that walking is engaged during most changes and therefore the algorithm detects walking (based on velocity and acceleration thresholding) as the change point. This method is further improved using knowledge of the underlying transportation network [10], and adding change of magnitude on heading and single travel-mode pattern-classifiers [11]. Lee et. al. [12] design a Variable-Rate-Localisation algorithm based on two key components: standstill detection, which uses a three-phase finite-state-machine based on GPS, accelerometer and Wi-Fi sensing, and an indoor/outdoor classification scheme. In [13], journeys are first extracted from the GPS trace, using features such as signal shortage and long periods of idle time and then segmented into modes. All stops are considered as potential transition points and may result in a single-mode trajectory to be segmented into many shorter pieces, but consecutive segments of the same modes are eventually merged. Similar experiments are reported in [14, 15, 16]. In most of the reported work, the parameters and thresholds are chosen from experience. On the other hand, the thresholds are determined by a K-means algorithm in [17] and [18] carry out a parameter search over a discrete grid to optimise the accuracy in detecting trip ends. In general, the literature lacks a comparison of algorithms mainly due to the lack of a standard and suitable method to compare them [19].

2.2 Battery-Aware Algorithms

In this section we review the literature on battery-aware computing related to our work, i.e. in location tracking, where the power-hungry GPS sensing module is used. In general, GPS battery-aware methods can be classified as either those that make use of two or more sensing modes [20, 21, 22, 12, 5], or those that make use of past history or spatial-maps [23, 24, 25, 26].

[3] `https://github.com/michael-camilleri/vjagg`

A single modal Location-Aware State Machine is used in [20] to throttle the GPS sensor when the user appears stationary, with limited power-saving results, whereas in [21, 22, 12] the schemes employ the accelerometer to detect periods of motion that triggers or throttles the GPS. Finally, in [5] the authors utilise an 'equidistance' tracking scheme, to predict velocity and dynamically adjust the rate at which GPS samples are taken, thereby saving power even while the GPS is in motion. Accelerometer readings are used to turn off the GPS when no motion is detected. Unfortunately, the authors do not quantify the savings due to their algorithm, focusing their contribution on the mode-detection instead.

From the survey it was clear that historical and spatial-map based methods are efficient in the use of battery consumption. However these schemes work well for coarse user-localisation, where the emphasis is on identifying where the user is at different periods of the day, often with respect to general key places and most of the time the detailed traces are compromised. In our case we want to preserve accurate origin and destination points as well as a detailed and accurate trip trace. Our algorithm is inspired from works that make use of various sensors, namely the GPS and accelerometer sensors. However whilst in the works reviewed above, the algorithms sample the accelerometer continuously, our algorithm makes use of one sensor at a time, i.e. while journey tracking is active, it is the GPS itself that identifies periods of no motion and turns itself off.

3 Problem Definition

In this section we give a mathematical representation of the problem and our proposed solution and describe the issues considered in the design of the algorithm.

3.1 Characterising Journeys

The location sensor (in our case, the device's GPS receiver) returns a sequence $L = (l_i)_{i=1,2,\ldots,N}$ of locations represented by the vector $l_i = (t_i, \phi_i, \lambda_i) : t_{i+1} > t_i$, denoting respectively the time-stamp, latitude and longitude coordinates. We will often summarise the spatial components of l_i by $X_i = (\phi_i, \lambda_i)$. We also define L_a^b to be a subsequence of L given by

$$L_a^b = (l_a, l_{a+1}, \ldots, l_b) \tag{1}$$

with $1 \leq a < b \leq N$. Additionally, we define the distance function $d(L_a^b)$ to be the sum of individual distances between each of the sub-sequence locations:

$$d(L_a^b) = \sum_{n=a}^{b-1} hav\left(X_{n+1}, X_n\right) \tag{2}$$

where $hav(a, b)$ denotes the haversine distance operator between two locations a and b [27]. We can now define a journey J as a sub-sequence L_a^b which satisfies the following two conditions:

$$d(J) \geq D \tag{3}$$

$$\nexists\, (a^*, b^*) : a^* < a < b < b^*,\ d(J) \approx d(L_{a^*}^{b^*}) \tag{4}$$

The first condition, (3) enforces that within a time-frame there is 'significant motion', represented by the threshold D. On the other hand, (4) ensures that no stationary periods at the ends of the journey contribute to the journey itself. Figure 1(a) shows an idealised sample-set L where the subject is initially idle, then moves for some distance before stopping. In this case, we wish to identify the 'discontinuities' in the samples as the start and end of the journey. It follows that a journey is defined by those sections of the data where the velocity is above a threshold.

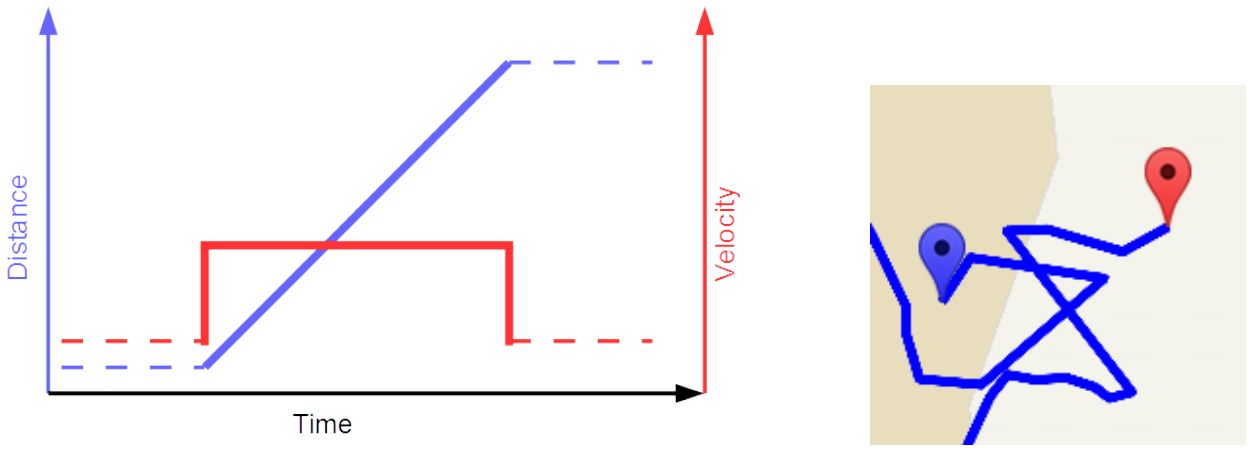
Mathematically, we can achieve this by numerically differentiating the location sample set L. We define the instantaneous speed of a point l_i to be:

$$\left|\frac{\partial(X_i)}{\partial(t)}\right| \approx v\left(X_i, X_{i-1}\right) = \frac{hav(X_i, X_{i-1})}{t_i - t_{i-1}} \quad \forall\, 1 < i \leq |L| \tag{5}$$

Note that in some cases, we shorten the notation as $v_i = v\left(X_i, X_{i-1}\right)$ and that the way we chose to approximate the derivative implies that we have one less velocity sample than position samples because v_1 is undefined. We can then rewrite our journey conditions, (3) and 4, as the contiguous sub-sequence of points where the speed is larger than our threshold:

$$J = \{l_n\} : v_n > V \quad \forall\, a \leq n \leq b \tag{6}$$

where the symbols have the same meaning as before.



(a) (b)

Figure 1: (a) Idealised Journey Characterization, in terms of distance travelled (blue) and velocity profile (red). The solid component is what we would classify as a journey/segment. (b) Typical example of noise at the start/end of a journey.

In implementing the above scheme, we have to deal with a number of practical issues. First off, the GPS signal is characterised by numerous inaccuracies: we assume this to be Additive White Gaussian Noise (AWGN). This is exacerbated by a 'canyoning effect' in locations surrounded by high-rise buildings [28].This phenomenon can lead to spurious jumps in locations, especially when the GPS signal is temporarily lost (see Fig. 1(b)). This is particularly significant since the proposed speed-based thresholding and averaging may not easily mitigate this form of noise, due to the high-magnitude jumps.

In our discussion in Section 3.1 we assumed that there is a definite starting and ending point of a journey, identified by clear changes in velocity. In practice this is rarely the case. Events such as stopping at junctions, bus-stops, pedestrian crossings, etc all trigger the end of a trip or segment and a simple velocity threshold is inadequate, yielding too many false-positives (in terms of end-triggers). Additionally, our definition of J in section 3.1 applies more to a single leg of a journey, rather than a necessarily complete journey [8]. Hence, we require methods for concatenating consecutive segments into single journeys.

3.2 Algorithmic Overview

The *Vjaġġ* journey Identification Algorithm (VIA) provides the core implementation to address the goal of storing faithful GPS traces of whole journeys, whilst taking the following goals into consideration: (i) minimise device memory usage, (ii) avoid intensive computations during logging, (iii) minimising battery energy consumption.

In order to achieve the above, we adopt a hierarchical approach, and implement a multi-level Finite-State-Machine (FSM), fig. 2, making use of multiple sensing modalities. At the highest level, the FSM operates in one of three states: (i) **OFF** : The Tracking Service is off (default state). Once tracking is enabled (manually by the user or automatically via an alarm), we transition to **GPS**. (ii) **GPS** : The Tracking Service is based solely on GPS sensing. In this mode, the algorithm is actively tracking journeys. The algorithm may exit this state if the user stops tracking or if it detects that the user is not travelling, in which case it switches to the **ACC** state. (iii) **ACC** : The Tracking Service is solely using the Accelerometer (GPS is off), and looking for the presence of 'significant motion' which potentially is the start of a journey. If motion is detected, the state goes back to **GPS**. These components are defined in the next section.

4 Active Journey Tracking

The main component of VIA is the GPS-based tracking of journeys. In order to get the best balance between journey segmentation accuracy, data storage requirements and computational efficiency, the implementation follows a two-stage process.

1. An *optimistic* online algorithm is employed during the active GPS sensing process. This consists of a finite-state-machine which identifies periods of idle locations (no-motions), journey start/stop triggers and the periods in between.

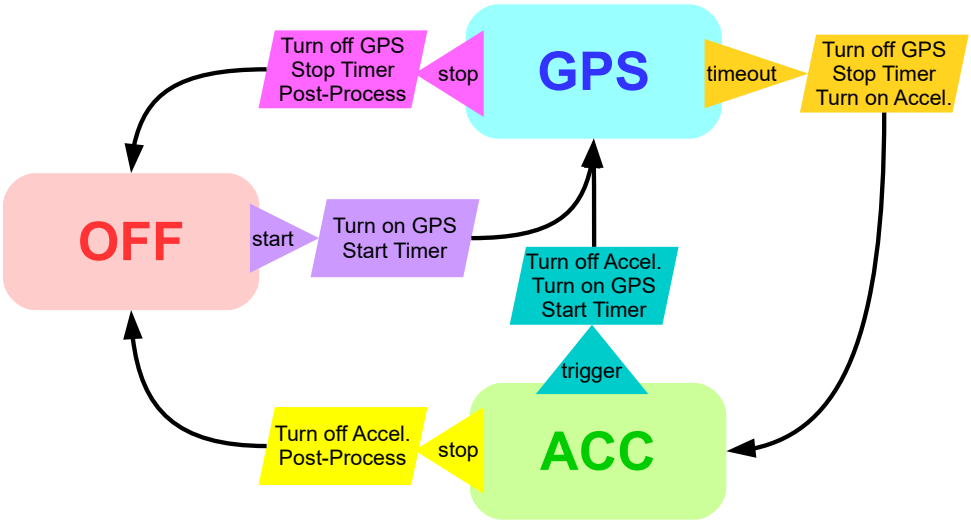


Figure 2: Global State Transitions

2. The generated segments (i.e tentative journeys) are offloaded to temporary file storage, which are then post-processed using a multi-pass filtering scheme once the **GPS**-state is terminated.

The emphasis of the online phase is to identify *tentative* start and end of journeys, with a bias towards over-detecting journeys – these can then be corrected using the offline post-processor which has a wider context. The two-stage process also avoids the storage and post processing of the full GPS trace signal at once, including long entries of stationary points (which can be easily picked up by the online algorithm). Note that the offline phase is still executed on the device!

4.1 Algorithmic Motivations

4.1.1 Filtering out Noise in GPS data

All the state changes of the on-line portion of the algorithm are governed (flow control) by the filtered (averaged) data samples $\hat{L}$, although we store all the raw GPS samples in J for the detected journey. We choose to filter AWGN noise (see section 3.1) with an averaging down-sampler (eqn. 7) , which worked well for the Markovian journey-start identifier (see section 4.1.2).

$$\hat{X}_i = \frac{1}{W} \sum_{n=i+1-W}^{i} (X_n) \quad \forall i \in \{Wk-1\}\,,\ 1 \leq k \leq \frac{N+1-W}{W} \tag{7}$$

In equation (7), the average over a window size of W samples is defined at every W^{th} sample starting from $W-1$, N is the sample size and X_n is as defined before. The choice of W is a trade-off between mitigating noise and guaranteeing a minimum data rate that is sufficient to identify start/end triggers.

4.1.2 Identifying Start Triggers

The start trigger is primarily identified with a velocity threshold defined in (6). Additionally, to mitigate the second form of noise ("canyoning effect", section 3.1), we opt for a Markov-Chain decision process, whereby: (i) the velocity value must exceed the threshold for a number of successive windows (the MC state), and (ii) the aggregate motion (the difference between the first and last data point in the MC state) must also exceed a threshold. The first condition seeks to reduce false-positives due to insignificant motions (for example shifting position while outdoors, resulting in significant velocity over a single or a few samples) while the second deals with the noise problem just mentioned (where multiple high-velocities are present, but generally no significant motion).

Formally, for a set of M consecutive samples from $\hat{L}$, a journey start is identified at down-sampled index j iff these two conditions are met:

$$\begin{aligned} v_n &> V^i \quad \forall\, j \leq n < j+M \\ \frac{hav\left(X_{j+M-1}, X_j\right)}{t_{j+M-1} - t_j} &> V^c \end{aligned} \tag{8}$$

where V^i is the individual velocity threshold and V^c the cumulative velocity threshold.

4.1.3 Locating End Triggers

Similarly, journey ending points are identified when the velocity falls below a threshold and a hysteresis approach is employed to cater for traffic congestion and stops. This entails that we do not search for an end-trigger until the *displacement* within a number of samples H (estimated as the displacement between the first and last point in the window) falls below a conservative threshold D^H. Once this happens we assume the journey segment has ended and attempt to locate a stop-trigger (at j) by running the Markov-Chain over the window H, as given by the conditions:

$$\begin{aligned} v_n &< V^i \quad \forall\, j < n \leq j+M \\ \frac{hav\left(X_{j+M}, X_{j+1}\right)}{t_{j+M} - t_{j+1}} &< V^c. \end{aligned} \tag{9}$$

In general, the displacement windowing scheme works well, including when u-turns are present. However the presence of stops when queuing at junctions and the passing through tunnels require further post-processing. Due to our conceptual definition of a journey, another *sufficient* condition for identifying a journey as having ended is when the user enters a building (i.e. we ignore motion within buildings). This can be detected from the loss of the line-of-sight signal of GPS satellites. VIA uses a time-out whereby if the GPS signal is lost for an extended period of time (to mitigate the effect of loss due to tunnels or high-rise buildings), the markovian-trigger-search scheme of (9) is initiated.

4.1.4 Concatenating Segments into Journeys

In the process of concatenating segments into journeys we first consider what can lead to journey segmentation in the first place, and from there work our way towards concatenating them. The most obvious reasons, which are typically mode dependent, include: (i) *On Foot* : taking a rest, meeting an acquaintance or waiting to cross the road. (ii) *Personal Vehicle*: idling in traffic congestion, waiting at junctions/stops/lights or dropping off/picking up (iii) *Public Transport* : idling in traffic congestion, waiting at junctions/stops/lights or regular scheduled stops. A key realisation in all these instances above is that the sense of continuity is indicated by a short punctuation, both in time (period between stopping and restarting) and in space (distance between stopping and restarting point). Our joining-algorithm is based on these heuristics. Basically, two successive journey segments, J_1 and J_2 are deemed to be part of a larger whole *iff*:

$$t_1^{J_2} - t_{|J_1|}^{J_1} < T \tag{10}$$

$$\frac{K}{M} \sum_{m=1}^{M} v_{|J_1|-m}^{J_1} < v\left(J_{1,|J|}, J_{2,0}\right) \tag{11}$$

where T is a time-threshold, K a distance threshold, and M is the size (length) of an average. The first condition enforces the time-constraint, while the second one ensures that it is realistic that given the speed of the user just prior to losing the signal, the second journey leg is a continuation of the previous one and the operation is recursive.

Finally, since we are not interested in very short commutes, we discard any journeys whose length is less than a threshold. In order to provide the best measure of journey length, we consider the journey bounds, indicated by a bottom-left (bl) and top-right (tr) corner pair rather than individual points:

$$X^{bl} = \left(\min_{n\in|J|}\left(x_n\right), \min_{n\in|J|}\left(y_n\right)\right), \quad X^{tr} = \left(\max_{n\in|J|}\left(x_n\right), \max_{n\in|J|}\left(y_n\right)\right), \tag{12}$$

where x and y are the cartesian co-ordinates of a location.

4.2 Online GPS Logger

The online portion of VIA is structured as a Finite-State Machine (FSM), triggered through GPS updates (handled by the underlying smartphone OS). The finite-state formulation serves to synchronise the otherwise asynchronous callback mechanisms provided, while allowing efficient use of the single-threaded implementation. The VIA FSM

triggers on downsampled points, i.e. state-changing decisions take place on the reconstructed $\hat{X}$ points rather than on the raw data at a base rate of 0.5Hz. The emphasis in journey identification is on reducing False Negatives (at the expense of increased False Positives, which are handled by the off-line component). The state of the FSM is governed by the following variables: (i) **Window Buffer**: H holds the down-sampled point averages. (ii) **Markov Chain**: M keeps track of successive velocities within the H Buffer (iii) **Start_ptr**: Keeps track of the sample (within H) at which a start of journey trigger was found. (iv) **Journey**: J conceptualises a journey segment, storing it to file as required. The FSM itself consists of four states. In all states, the GPS is turned *ON* and points are buffered, however not necessarily logged to file. The states are (in order of typical flow): (i) **Idle** : The M Buffer is still filling up. This state is required because the Markov Chain logic requires at least two-samples to commence. (ii) **Searching** : The Markov Chain is searching for a start trigger. (iii) **Found** : The Markov Chain has found a start trigger, but not enough time-points have been retrieved to allow determination of end-triggers (H buffer is filling up). (iv) **Logging** : The journey is being logged to file.

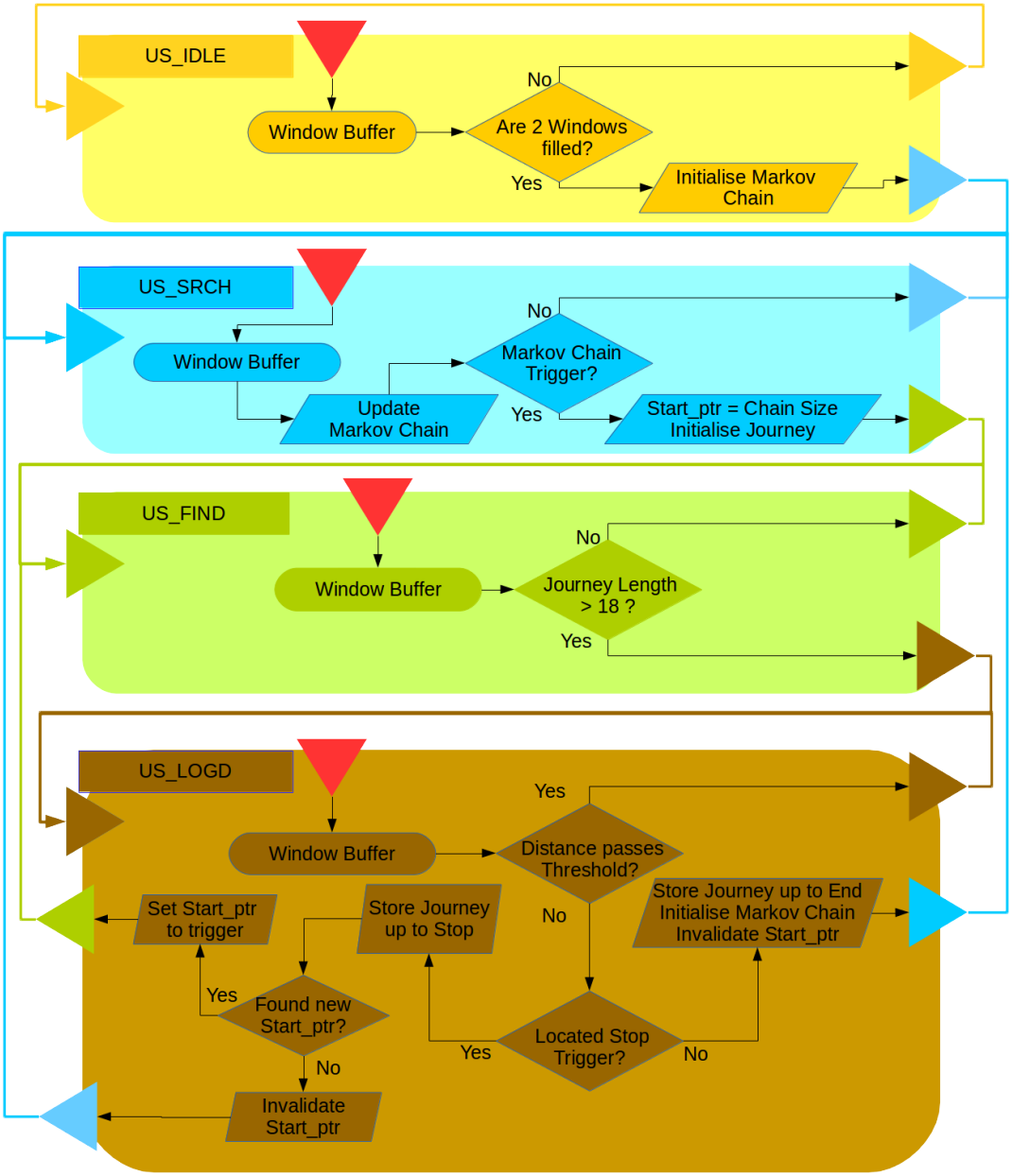


Figure 3: Flowchart of the GPS Finite State Machine showing state progression under normal circumstances.

Figure 3 depicts the flow of control between states. Each state is represented by its colour, such that the colour of the operation itself serves to identify the state with which it is associated (including the transition to the next state). All operations with the same colour and within the same background signify completion within the same down-sample callback. Transitions take place on any down-sample callback cycle (but not within the same callback) and are indicated by the coloured arrow-lines extending outside the state border. Each state has an entry point (of its own colour) and a set of exit points (with the colour of the state that the machine will transition to on the next cycle). Furthermore, the entry point for each state is not connected to the first operation to be performed while in that state: instead, this comes from the red-coloured entry point (down-sample callback), to emphasize that the state change happens on a callback boundary.

The State Machine starts in the **Idle** state (***US_IDLE***), with the H/M buffers flushed. Once the buffer is full, the machine transitions to the **Searching** state. The Markov Chain M is initialised with the velocity between the first two windows. Once the trigger is found, we set the ***Start_ptr*** to the beginning of the appropriate window and transition to the **Found** state. The **Found** state (***US_FIND***) serves to buffer samples until there are enough for the end-trigger calculation. Once a journey reaches the H length, VIA transitions to the **Logging** state. Finally, in the **Logging** state (***US_LOGD***), VIA buffers the samples while checking for the distance threshold over a period of H windows. If the distance threshold is met, then we stay in the same state. Otherwise, we test for an end of journey. We run the Markov Chain M' in reverse (starting from the oldest window and moving forwards in time) until we locate a stop trigger or we reach the present location. If no stop trigger is found, we simply store the journey up till the latest point and transition to the **Searching** state. Otherwise, we attempt to find another start trigger, since a new journey could have started in the mean-time. If one is found, we set the ***Start_ptr*** and move to the **Found** state. Otherwise, we retain the Markov Chain state (with a potential partial trigger) and switch to the **Searching** state.

In addition, due to time-outs and the stop-event, a further asynchrony is introduced. The asynchrony refers to the fact that although the actual transition does happen when the down-sampler is idle (i.e. it does not interrupt an in-state operation as illustrated by a filled background in Fig.3), it can happen any time in between calls (it is state independent). In fact, more often than not, it happens between successive GPS updates, implying only a partial down-sample (which must be explicitly catered for, since the algorithm runs at the down-sampled rate).

Besides indicating a potential end of journey, long gaps between GPS fixes could pose a problem for thresholds. The location callback itself is triggered only when there is a fix, and hence, if not called, will halt the FSM. A watchdog is thus employed to identify when the signal is lost for an extended period of time. If the watchdog triggers, it checks whether the currently active state indicated a journey was being logged (i.e. we were in state **Logging**). If this is the case, then we attempt to find a stop trigger within our buffer and store the journey up to either the location of the stop trigger (if one is found) or to the end of the buffer. If no journey was active we flush the buffer (and transition to the **Idle** state).

4.3 The Offline Post-Processor

The Post-Processing algorithm is triggered when the user presses the Stop Tracking Button (after the on-line algorithm terminates) or the **ACC** trigger kicks in. The first task is to load all stored journeys from the temporary file generated by the FSM. The algorithm then executes a number of distinct routines:

Threshold based on Distance (low): Initially, all journeys whose length is less than 50m are discarded. The low 50m threshold ensures that if a valid journey consists of multiple small parts (perhaps due to being stuck in traffic), it does not get eliminated in this first step.

Journey Concatenation: The main aim of post-processing is to allow individual journey segments to potentially be joined into a single journey. Starting from the next to last journey and moving backwards, if the duration between the two journeys is less than a threshold and the distance between the two journeys is such that the average velocity at the end of the first journey (up to a tolerance factor, currently set at 120%) indicates that the starting point of the second one is a viable continuation, then the journeys are concatenated.

Threshold based on Distance (high, 500m): Another threshold on distance is performed. This together with the initial thresholding operation, provides a hysteresis threshold.

End Trimming: Finally, in order to mitigate the spurious jumps which occur while the GPS system is achieving a stable fix, the ends are trimmed for points with velocities in excess of 20m/s, up to a limit of three eliminations (to prevent eliminating the entire journey).

4.4 Battery-Aware Algorithm

In this section we describe the battery energy consumption savings features of the VIA. The algorithm depends on the availability of a sensory input, which yields a distinct output value when the device is idle (motionless) and when it is in motion (travelling). We use the accelerometer, which is found in virtually all smartphones ($> 99\%$[4]).

From an information perspective, the GPS is redundant when the user is stationary and when the GPS cannot obtain a reliable fix, such as when the user is indoors, the latter being dealt with in the context of the journey detection portion of the algorithm. We deal with the former by means of another watchdog time-out condition, which triggers when no journey has been active (VIA state is US_IDLE or US_SRCH) for an extended period of time (in our case five minutes). When this happens, the GPS is switched off, and the algorithm transitions to looking for significant motion by way of the accelerometer.

The problem of *identifying significant motion* is difficult due to (a) the presence of noise (which is especially pronounced in the accelerometer), *and* (b) the term stationary may not necessarily mean perfectly motionless. In short we wish that no motion is signalled when the device (a) is on a table, (b) is in the user's bag/garment pocket who is sitting or standing, (c) vibrates while in the user's pocket, *or* (d) is briefly checked by the user. In particular handling conditions (c) and (d) reduces false-positives and the GPS turning on unnecessarily. Conversely, we wish to detect motion when the user walks or drives with device in hand, bag or garment pockets. In VIA the emphasis is on reducing false negatives, since we seek to pick up the starting point of a journey as accurately as possible.

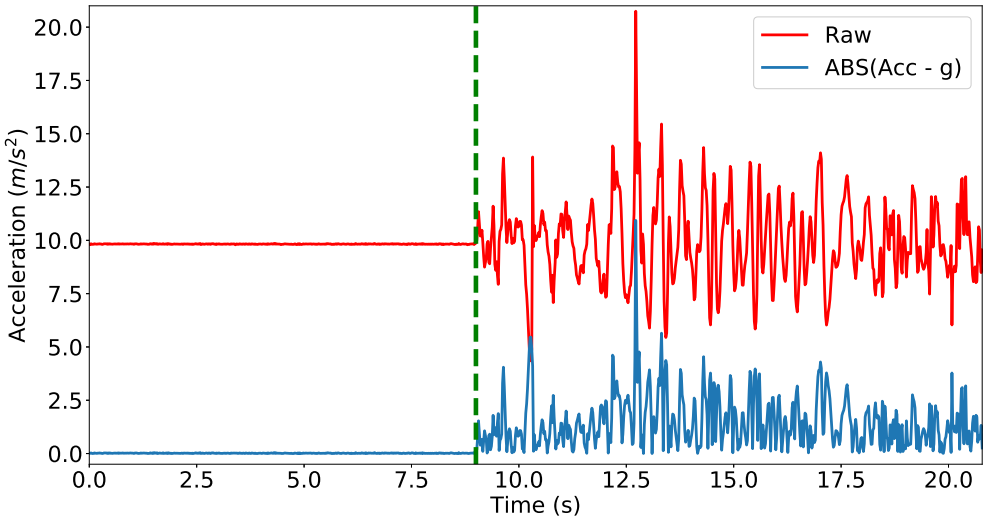


Figure 4: Typical Acceleration Profile: the dotted line indicates the transition from standstill to motion. The raw acceleration is shown in red, with blue being the filtered version (subtracting $g = 9.81$ and taking absolute value).

Fig. 4 displays a typical acceleration-magnitude profile (blue trace) of a Samsung Tablet, transitioning between static to hand-held, where we note that while the raw average appears to remain the same, the average deviation from the nominal $9.81 m/s^2$ is detectable. Fig. 5 gives acceleration profiles in a number of scenarios, while the decision logic is given in Fig. 6. Optimisation of thresholds/parameters was achieved through simulated annealing and random search (described further below). The Finite-State-Machine (FSM) goes through three distinct states. (i) **INIT** : This is an intermediary initialisation stage which is executed when the GPS is turned off. (ii) **CHECK** : This state provides a low threshold for early detection of motion (based on a Markovian threshold) which can then later be verified in the **EXTRA** state. (iii) **EXTRA** : This state consists of an extended sample run, using a different threshold to confirm or reject the original hypothesis arrived at in state **CHECK**. The decision to use a two-stage checking scheme follows from the intuition that even in motion-less scenarios, there may be occasional spikes, albeit of short duration (see Fig.5(f), when compared to (c)) which can throw off a single threshold. This also spurred the option to test with multiple successive averages in the **EXTRA** state.

[4]https://opensignal.com/sensors/library/accelerometer

Table 1: List of Android Test Devices. Specifications retrieved from *www.gsmarena.com*

Device	**Tab E 9.6”**	**Tab A 10.1”**	**Pulp 4G**	**J5**
Device Code	T1	T2	S1	S2
Quantity	1	1	1	3
Manufacturer	Samsung	Samsung	Wiko	Samsung
Model	T561	T585	–	J510FN
OS Version	4.4.4	6.0.1	5.1.1	6.0.1
CPU Cores	4	8	4	4
CPU Speed	1.3GHz	1.4/1.0GHz	1.2GHz	1.2GHz
Battery (mAh)	Li-Ion 5000	Li-Ion 7300	Li-Po 2500	Li-Ion 3100

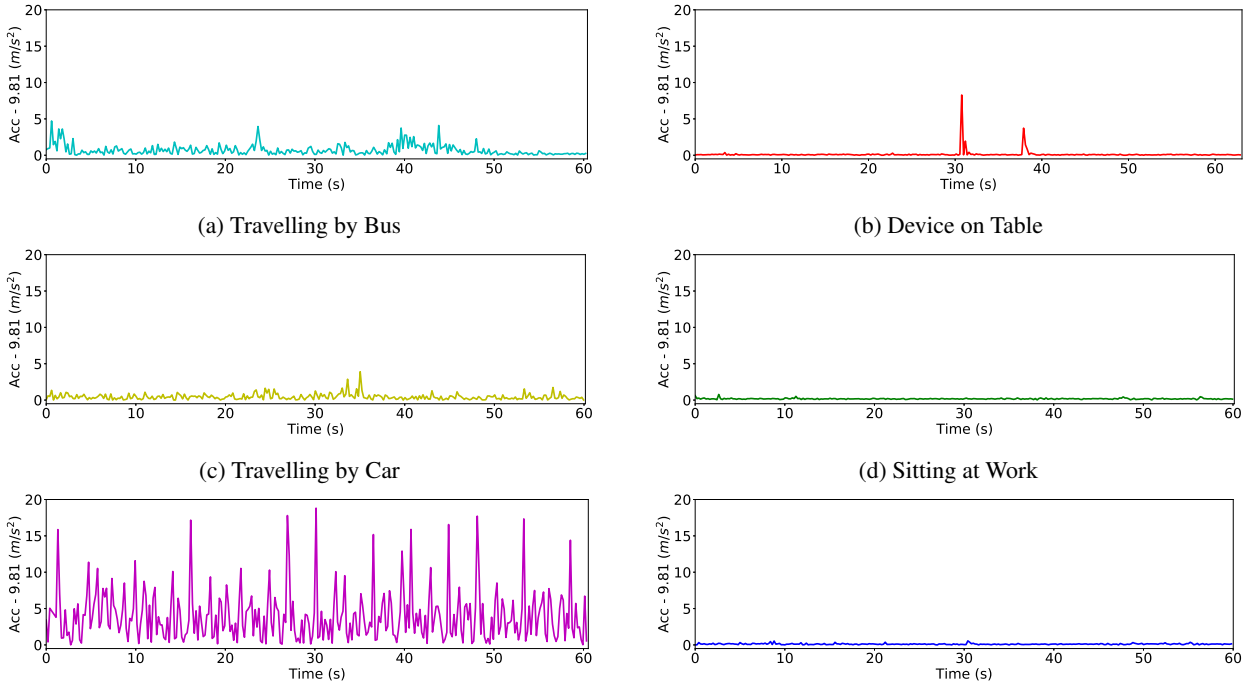
5 Tuning and Evaluation

In this section we discuss the tuning process of the various free parameters and their experimental validation, and describe and evaluate the empirical experiments carried out to determine the efficacy of VIA on real-world data.

5.1 Experimental setup

Our tests were carried out on a range of devices from different manufacturers and running on various OS versions, as noted in Table 1. We collected data from three main modalities, namely the raw sensors, battery-usage and pen-and-paper diaries for validation.

Sensor Data: Verification of the algorithmic operation required multiple runs with different parameters on the datasets. To this end, we incorporated an option in our wrapper application to save all sensor data (GPS fixes, satellite and accelerometer readings to file storage for later retrieval. All readings were time-stamped, and offloaded to an



(a) Travelling by Bus

(b) Device on Table

(c) Travelling by Car

(d) Sitting at Work

(e) Travelling on Foot

(f) Sitting/Standing at meeting

Figure 5: Acceleration profiles for Moving (Left) and Motion-Less (Right) scenarios.

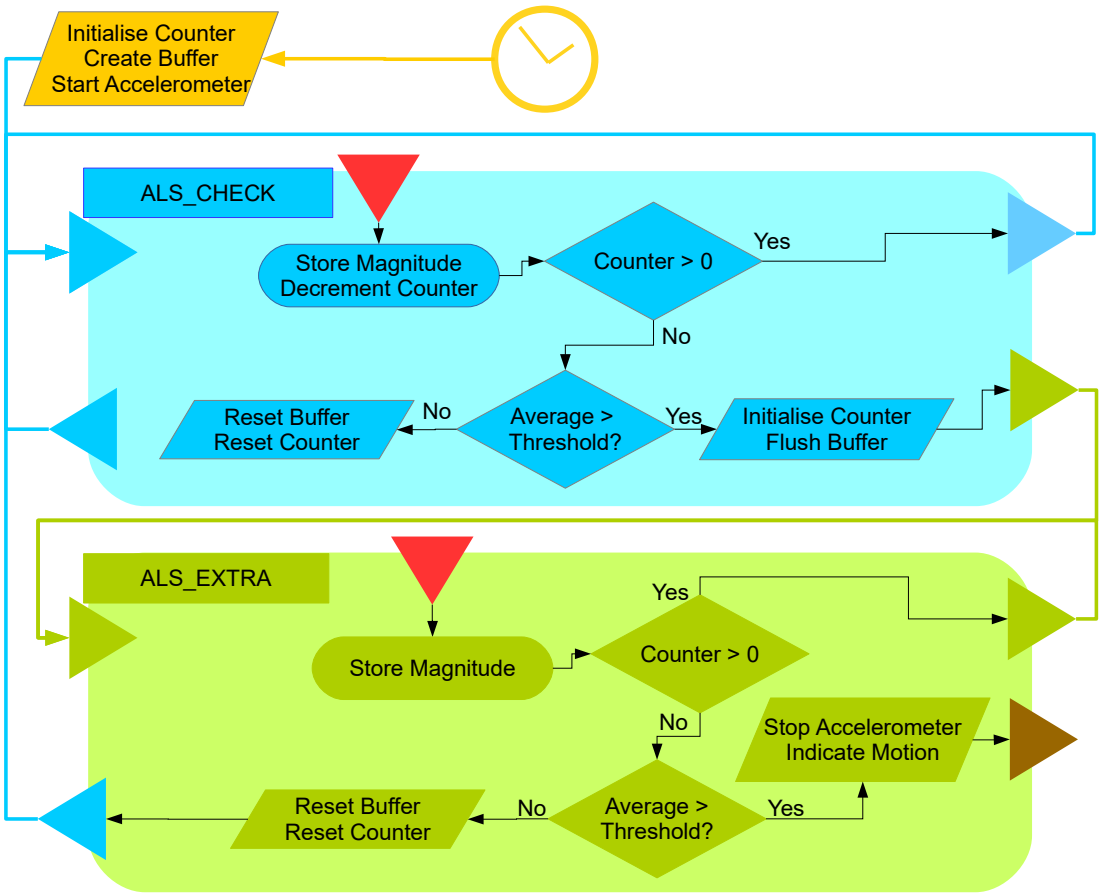


Figure 6: Detection of Significant Motion (trigger from Accelerometer to GPS). (The colour codes here are independent of those in Fig. 3)

external text file, which we then retrieve directly through the phone's file system. In the interest of efficiency and file-size considerations, we cap the sampling rate at 0.5Hz for the Location and 5Hz for the acceleration, as indicated in literature, e.g. [5].

Diaries: In order to fine-tune the parameters and validate the algorithm we required the addition of annotated journey data with clear starting and stopping points, kept by the annotators. To aid this, we also employ a 'ping' feature within the 'debug' application to allow the annotator to explicitly indicate the start or stop of a journey.

Battery Consumption: Typically, power data is collected using elaborate custom-made hardware, e.g. [29], however, given the constraints of our project, we opted for a software based approach. In this case, we collected battery percentage and voltage levels, and for all but T1 the instantaneous/average current drawn. Data was sampled at either 15 (IDLE-battery tests) or 1 (for all other tests) minute intervals using the *AlarmManager*.

5.2 GPS Noise

Although it was not our intention to fully characterise the noise process within the Location framework, we nonetheless ran a number of tests to determine whether the amount of noise would be significant in our algorithm. We ran two types of tests, in a static and dynamic setting.

Static Noise: To quantify the magnitude of noise in different static scenarios, we left the device in a position with a view of the sky (namely in direct view, partially obstructed by a building and under foliage) and computed the difference from a mean recorded location for a period of 10 minutes. This setup is based on the assumption that the noise is Gaussian distributed around the actual true value. Results showed that, apart from a certain amount of drift with time in the recorded locations, there were no significant errors beyond 10^{-4} in latitude and 1.4×10^{-4} in

longitude. In fact, the satellite count as recorded by the framework was barely affected in each of the tested scenarios, varying between 11 and 16 satellites. Figure 7 displays a sample run.

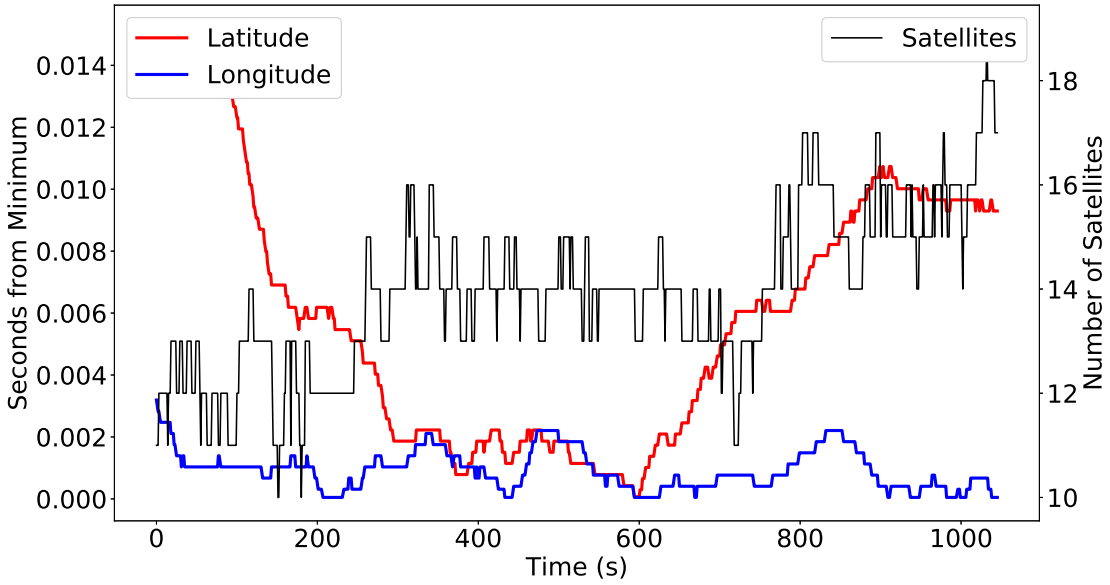


Figure 7: Typical example of static noise for Device S2 under extreme foliage.

Dynamic Noise: Potentially, in the VIA algorithm, the greatest detriment of noise would be in causing false-positive triggers for the start/stop detection algorithm. We therefore sought to characterise the effect of the noise on velocity computations from successive positions. It was also desirable to test the system under severe canyoning effects which provide the worst case scenario for GPS tracking. The tests where carried out in Valletta, characterised by a Manhattan Grid street pattern, narrow streets and relatively high buildings. The experiments involved walking at a moderate constant pace alongside a block. Multiple circumnavigations of more than one block at a time were carried out, with the annotator pinging at each corner. The experiments were repeated around different blocks. The data, collected at intervals of 1s, was then filtered as follows. First, the straight-line motions between corners of the blocks, were separated into individual runs R_i. For each of these runs, the mean velocity V_i was calculated by dividing the straight-line distance between the corners (obtained online from Google Maps) by the total travel time (obtained from the pings registered by the researcher). The individual points are then averaged over windows of varying sizes $w \in \{1...10\}$ to yield a down-sampled run $D_{i,w}$. In order to increase the number of samples, for reliability of the computation, in windowing schemes, multiple runs D^k were computed started at successive points. Finally for each pair of points $d_{i,w}^{j,j+1}$ within $D_{i,w}$, the first-order approximation of the velocity was computed, and the deviation from the nominal velocity V_i recorded. In summary, the mean discrepancy for a particular window size w was computed as follows:

$$\tilde{V}_w = \frac{1}{|W||I|} \sum_{k=0}^{w-1} \left\{ \sum_{i=1}^{|I|} \left(\frac{1}{|D_{i,w}^k|} \sum_{j=1}^{|D_{i,w}^k|-1} \left| \frac{d_{i,w}^{k,j+1} - d_{i,w}^{k,j}}{t_{i,w}^{k,j+1} - t_{i,w}^{k,j}} - V_i \right| \right) \right\} \tag{13}$$

Figure 8 illustrates the effect of window-size on the estimated velocity '*error*'. As can be seen (and mostly from the maximum error), a window size of 3 should provide adequate filtering (with the 95^{th} percentile lying within 1.1 m/s from the nominal) while not delaying triggers excessively (due to too large windows).

5.3 Identifying Journey Starts

The parameters of interest here are the three related to the aforementioned Start Markov Chain: i.e. the number of velocity points to consider, the instantaneous velocity threshold and the total velocity threshold. The thresholds here are set for typical walking speeds: given the average speed of 1.3m/s, we cap both the instantaneous and total velocity at 1m/s, which must both be exceeded to start recording data. This is a conservative threshold, but we chose it because we prefer to generate False Positives (i.e. starting tracking when no journey actually exists), which can be handled by the offline post-processor, rather than False Negatives (which would miss out on a journey). The choice of three time-steps was based on visual inspection of the GPS traces during start/stops of journeys as well as typical motion

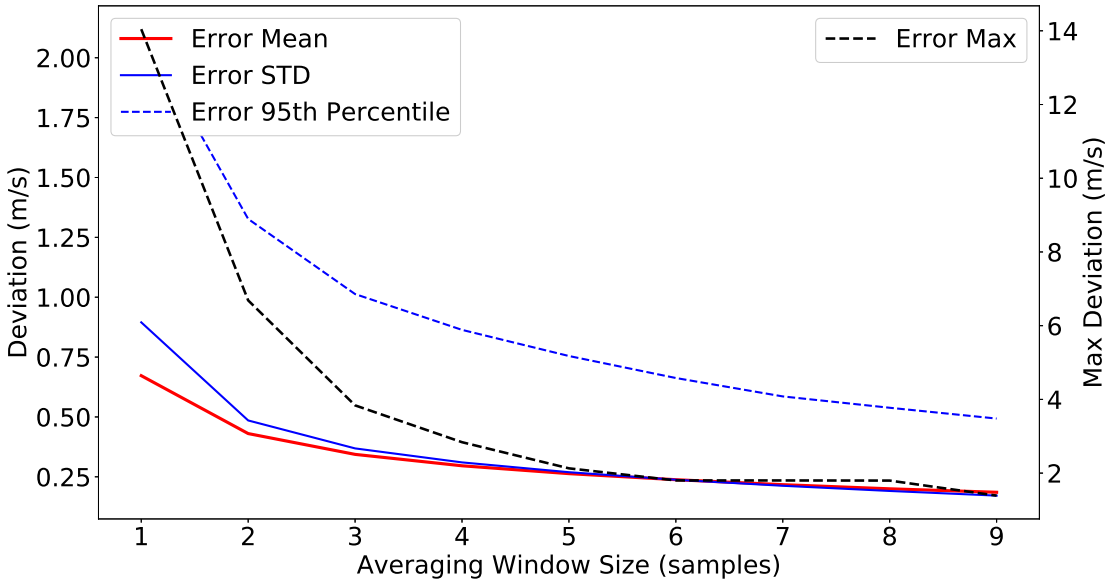


Figure 8: Evolution of deviation from Nominal Velocity due to noise for varying average-window sizes. Note that *max error* is on a different scale (right).

patterns for individuals in open spaces. Specifically, given the 0.5Hz sample rate and the further down-sampling by 3, the length of the chain corresponds to a time-period of 24 seconds which is adequate to filter out short motions but can capture intent to start a journey.

5.4 Determining Journey Stops

In the stop-detection, preference is given to False Negatives (i.e. not detecting a stop) rather than False Positives. With regard to the detection window, we employ 2.5 minutes worth of data (25 down-sampled points), and a threshold distance (i.e. distance between first and last point in the set) of 30m. The motivation behind this scheme is targeted mostly at vehicular travel: specifically, it seeks to avoid detecting slow traffic or stopping at junctions/lights as journey termination. The values were intuitively set based on traces of journey data. Given this, we then retroactively seek to find the actual point of journey termination, using the same Markovian scheme as for journey starts, although with the conditions reversed. In this case it was found adequate to use the same thresholds as above: i.e. 3 velocity windows, with instantaneous and total velocity thresholds set at 1m/s.

5.5 Satellite Indications

The number of satellites in line-of-sight are a key indication of journey termination (due to the user having entered a building), but at the same time, can generate False Negatives (such as tunnels). Theoretically a GPS fix can be achieved with a minimum of 4 satellites: however, this threshold also has to consider the difference between typical indoor and outdoor receiving status. In tests carried out it emerged that in good view of the sky, the satellite count could be as high as 11 or 12 satellites, while indoors this falls to 0 or sometimes 1. Hence, we decided to cut-off at a value of less than 5.

The determination of the time-out at which to signal such a journey end has to do with the tunnel problem. This is hard to quantify, as it depends not only on tunnel lengths but also the vehicle speed and the presence of traffic. We employed a time-out of 40 seconds which worked well for our use-case. At the same time, we choose to mitigate the problem using the off-line post processor which is able to join together journey segments.

5.6 Concatenating Journeys

The journey concatenation scheme is mainly designed to address the problem of vehicular journeys being divided into smaller segments, due to temporary signal loss. From the data, we observed that the velocity magnitudes at the splitting points are similar. This condition forms the basis for our concatenation algorithm.

More specifically, concatenation depends on two parameters; (i) the time difference between successive journey segments, manually set to 2 minutes, and (ii) the end/start velocity ratio, set to 1.2. It should be noted that this scheme does not handle the case where the segmentation is due to being stuck in traffic or at junctions. This is because, in this case the conditions are typically the opposite (short wait times, and excessive velocity differences). Instead, this is designed to be catered for by the stopping hysteresis in the on-line algorithm.

5.7 Filtering out irrelevant journey segments

Very short journeys (e.g. moving between buildings on a small campus) are typically non-informative for the scope of our use case and we delete journeys that are less than 500 metres. We also trim portions of journeys that are due to spurious jumps in GPS traces, that occur due to the location service making use of both GPS and other less accurate sensors (such as Wi-Fi access-point information or mobile cells), and typically characterised by a rapid jump, with velocities in excess of 20 m/s. While this velocity is itself perfectly normal for vehicular travel, these typically happen towards the beginning or the end of a trip (when there is GPS signal loss), and where typically, velocities are still low mostly due to the first mode of transport, i.e. walking. Hence, we discard such points at the beginning (first three points) and end of the journey (last three points) whose velocity exceeds this threshold.

5.8 Battery Consumption Profiling

The battery-aware sub-system of our base Journey Segmentation algorithm comes at a price in data accuracy. Extensive testing was carried out to find the best trade-off between battery efficiency (which would impact on user uptake) and accuracy of data.

5.8.1 Idle Battery Consumption

We ran tests with the device in standby to identify the power-consumption of the device under idle conditions (no applications running, Wi-Fi/3G off, GPS switched on but not polling). We also allowed the OS to handle all sleep control, including DOZE[5]. The importance of these tests is especially marked for *T1*, which has no measure of current drawn. Since in general the discharge may not follow a linear decay [30], the resulting profiling (from 100% to 80% over a period of 5 days) supported our decision to extrapolate from a linear idle consumption. Figure 9 shows the evolution of the voltage with time as the devices discharge slowly (typically over a number of days). For brevity's sake, only the discharge for device *T1* is shown. The charge level shows periodic oscillations but a general linear trend can be inferred (dotted line). The linear estimate was fitted through least-squares of degree 2 and the squared term was actually 6-7 orders of magnitude less than the linear term, meaning that the discharge can be assumed linear. The discharge rates were estimated to be 1.79, 1.06, 5.02 and 2.17 units per hour for *T1*, *T2*, *S1* and *S2* respectively. The rate is higher for smartphones as opposed to tablets (due to the different battery capacities) but also shows a trend towards higher efficiency with the DOZE-feature in Android 6.0.

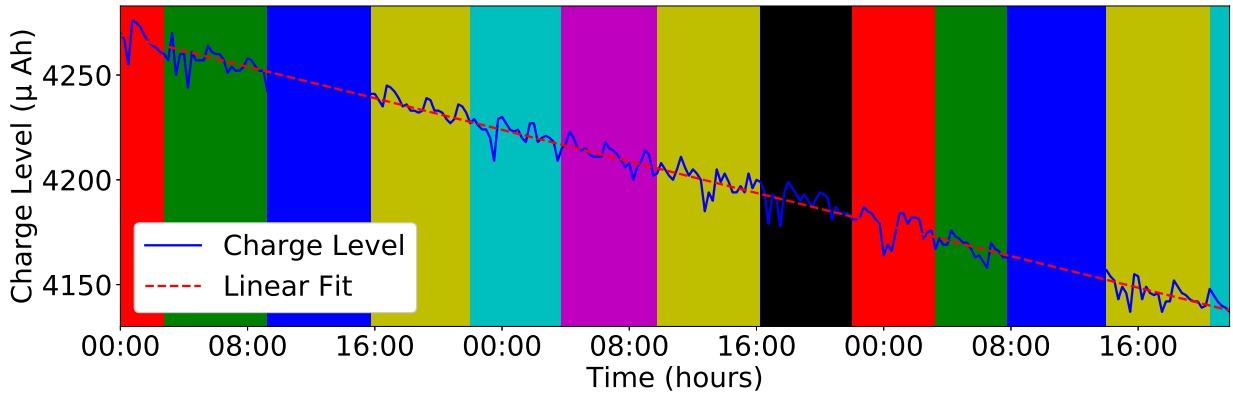


Figure 9: Idle Discharge for T1. The coloured bars indicate different battery-% readings. Note how due to their inconsistent width, they are not directly indicative of consumption.

[5]Note that our algorithm actually disables DOZE, and hence, this is a tougher baseline to compare to, and illustrates the capability of our algorithm much more clearly.

5.8.2 Active Battery Consumption

We studied power consumption of the GPS (base algorithm), accelerometer and the use of the combined wake-lock by recording and comparing the battery discharge rate, rather than by generating a detailed map of power usage (c.f. [29]). The results, tabulated in Table 2 and displayed graphically (for *T2* and *S1*) in Fig. 10, show significant differences between the various algorithms. While turning on the accelerometer consumes an average of three times the idle rate, the GPS increases the rate by an order of magnitude (nine to thirteen times the idle rate).

Device	T1	T2	S1	S2
IDLE	1.79	1.06	5.02	2.17
Wake-Lock	6.16	–	–	6.54
Accel. Cont.	6.71	3.22	6.70	7.42
Accel./Sleep	6.58	3.20	–	7.22
GPS	20.13	10.22	47.97	28.26

Table 2: Discharge Rates under various conditions (first-order coefficient of quadratic polynomial)

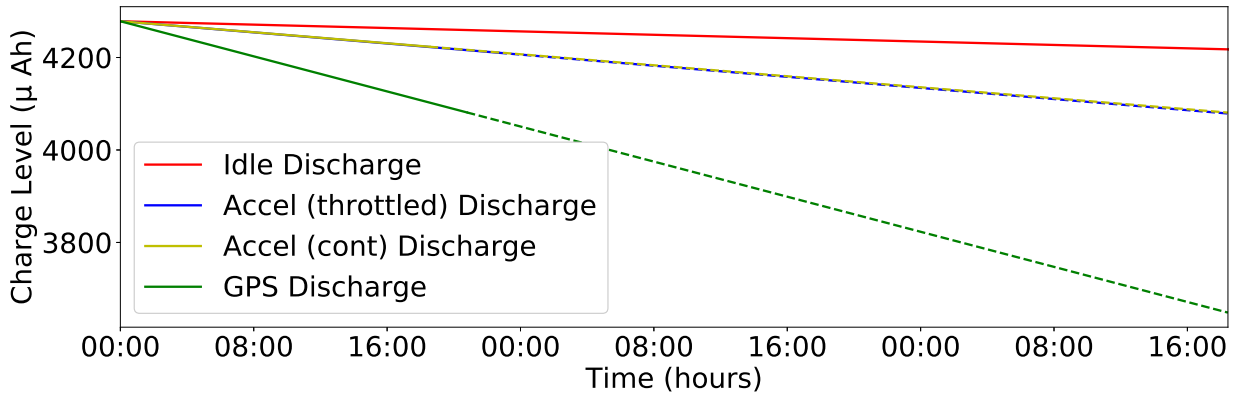


Figure 10: Discharge profile for one of the devices under test (T2). The solid lines are the fitted discharge which is extrapolated linearly (dotted lines).

In the literature there is hardly any consensus as to whether achieving a GPS fix or not affects power consumption, [21, 29, 5]. To study this, a four-way test was set up, with identical devices turned on under the conditions of a clear view of the sky and indoors, as well as in different android location modes utilising either the GPS only or with assistance from Wi-Fi/Cell information. The average measured discharge rates when outdoors are 20.23 (GPS only) and 22.75 (assisted mode), whilst for indoors rates are 26.37 (GPS only) and 28.87 (assisted mode), or approximately 30% higher. Additionally, assisted location services (which mitigate GPS signal loss) add to further energy consumption, unless the algorithm uses these as another mode to turn the GPS off.

5.8.3 Motion-Detection FSM Tuning

The choice of appropriate window-sizes and distance thresholds for the motion-detection FSM were determined using simulated annealing. The five free parameters are: (i) length of initial sample buffer (1 to 20), (ii) motion threshold for the initial buffer 0 to 5), (iii) size of second sample buffer (1 to 20), (iv) motion threshold for the second buffer (0 to 5), and (v) size of sample-windows over which to average the second buffer (1 to 5). The accelerometer traces were used for training the identification algorithm (classifies: motion/no motion) and the parameters optimised using a cost function (Eq 14) based on False-Negative Rate (FNR) and False-Positive Rate (FPR), number of samples till detection of a True Negative (N_N), and number of samples till detection of a True Positive (N_P), to regularise the process and avoid overfitting.

$$C = 12 \times FNR + 4 \times FPR + 0.02 \times N_N + 0.04 \times N_P \tag{14}$$

In line with the needs of the application, the FNR is weighted three times the cost of the FPR (since it is more significant), while cost for sample-delays are two orders of magnitude less than the error rates. The algorithm was

Predicted \Actual	Negative	Positive
Negative	**8104 (96.0%)**	295 (7.3%)
Positive	338 (4.0%)	**3738 (92.7%)**

Table 3: Optimal Classification Results (training data)

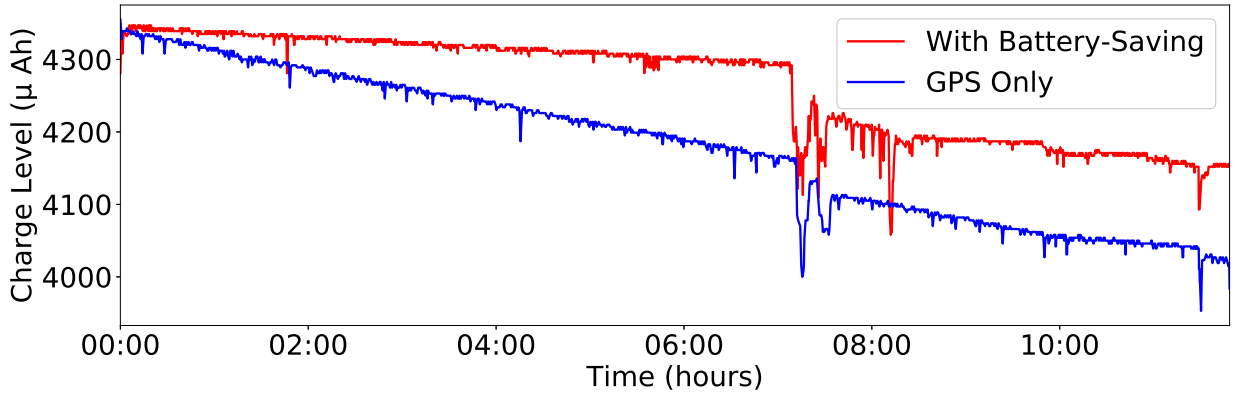


Figure 11: Typical Battery Discharge with (red) and without (blue) battery-saving

executed for 10000 epochs, with the experiment repeated multiple times. The optimal parameters indicated a first sample buffer of size 5 samples (about 1s) with a conservatively low-threshold of 0.18 followed by a single sample buffer of size 7, with a threshold of 4.78. In this scenario, the performance on the training data achieved appears in Table 3 (validation is discussed next). Interestingly, the search converges to a low-threshold followed by a high one. This follows from the intuition that the first threshold serves to 'feel' the acceleration and hence there is no need to investigate further if there is no motion. Also, counter-intuitively, the second run always converges to a single averaging window rather than multiple small ones.

5.8.4 Performance Validation

Finally, we ran tests for typical usages throughout a single day with the baseline algorithm and the battery-aware version. We use these results to demonstrate the efficacy of our system. In this case, participants were given two identical devices (*S2*), one running the base algorithm and the other the battery-aware adaptation, as they went about their normal day routine. The device running the battery-aware scheme detected 88% of all journeys recorded using the base algorithm (i.e. without the Battery-Saving scheme): 9% of the journeys were clipped at the start or end of the journey and the rest involved a trip on a ferry which was not detected, possibly due to minimal acceleration. The battery-aware algorithm exhibited savings of between 50% and 70% in total battery consumption, compared to the base algorithm (see Fig. 11). Furthermore participants who donated their data reported that the version with the battery-saving algorithm is significantly better compared to the one without, in the sense that they were not worried of running out of battery power towards the end of the day, contributing to sustained uptake.

6 Conclusions

In this paper we reported on the development of a smartphone based trip detection and mobility data collection application. We defined what constitutes a journey and identified a number of issues which may arise in naively using GPS traces. We described in detail the trip segmentation algorithm, its implementation and its battery-saving schemes, which allowed the phone to track a full day's worth of journeys without the need of re-charging. We tuned the free parameters of the algorithms by optimising on captured data, and verified that our technique is able to capture most of the trips in a real-world mixed-mode scenario (97%, if considering all trips logged) with significant battery savings of up to 70%.

Future work is required to further fine-tune many of the thresholds with more more in-the-field experiments to verify and understand their effects. It is also interesting to look at using map data to enhance the concatenating scheme, especially as regards transport mode changes: other outlets include measuring the bias in the rate of under/over-reporting of the measurement method.

References

[1] P. R. Stopher, L. Shen, W. Liu, A. Ahmed, The challenge of obtaining ground truth for GPS processing, Transportation Research Procedia 11 (2015) 206 – 217, transport Survey Methods: Embracing Behavioural and Technological Changes Selected contributions from the 10th International Conference on Transport Survey Methods 16-21 November 2014, Leura, Australia. doi:https://doi.org/10.1016/j.trpro.2015.12.018.

[2] K. T. Geurs, T. Thomas, M. Bijlsma, S. Douhou, Automatic trip and mode detection with move smarter: First results from the Dutch mobile mobility panel, Transportation Research Procedia 11 (2015) 247 – 262, transport Survey Methods: Embracing Behavioural and Technological Changes Selected contributions from the 10th International Conference on Transport Survey Methods 16-21 November 2014, Leura, Australia. doi:https://doi.org/10.1016/j.trpro.2015.12.022.

[3] M. Berger, M. Platzer, Field evaluation of the smartphone-based travel behaviour data collection app "smartmo", Transportation Research Procedia 11 (2015) 263 – 279, transport Survey Methods: Embracing Behavioural and Technological Changes. Selected contributions from the 10th International Conference on Transport Survey Methods 16-21 November 2014, Leura, Australia. doi:https://doi.org/10.1016/j.trpro.2015.12.023.

[4] F. Zhao, A. Ghorpade, F. C. Pereira, C. Zegras, M. Ben-Akiva, Stop detection in smartphone-based travel surveys, Transportation Research Procedia 11 (2015) 218 – 226, transport Survey Methods: Embracing Behavioural and Technological Changes Selected contributions from the 10th International Conference on Transport Survey Methods 16-21 November 2014, Leura, Australia. doi:https://doi.org/10.1016/j.trpro.2015.12.019.

[5] A. C. Prelipcean, G. Gidofalvi, Y. O. Susilo, Mobility Collector, Journal of Location Based Services 8 (4) (2014) 229–255. doi:10.1080/17489725.2014.973917.

[6] C. Cottrill, F. Pereira, F. Zhao, I. Ferreira Dias, H. Beng Lim, M. Ben-Akiva, C. Zegras, Future mobility survey, Transportation Research Record: Journal of the Transportation Research Board 2354 (2013) 59–67. doi:10.3141/2354-07.

[7] A. C. Prelipcean, Y. O. Susilo, G. Gidofalvi, Collecting travel diaries: Current state of the art, best practices, and future research directions, Transportation Research Procedia 32 (2018) 155 – 166, transport Survey Methods in the era of big data:facing the challenges. doi:https://doi.org/10.1016/j.trpro.2018.10.029.

[8] A. C. Prelipcean, G. Gidofalvi, Y. Susilo, Meili: A travel diary collection, annotation and automation system, Computers, Environment and Urban Systems 70 (July 2018) (2018) 24–34, qC 20180605. doi:10.1016/j.compenvurbsys.2018.01.011.

[9] Y. Zheng, L. Liu, L. Wang, X. Xie, Learning transportation mode from raw gps data for geographic applications on the web, in: Proceedings of the 17th International Conference on World Wide Web, WWW '08, ACM, New York, NY, USA, 2008, pp. 247–256. doi:10.1145/1367497.1367532.

[10] L. Stenneth, O. Wolfson, P. S. Yu, B. Xu, Transportation mode detection using mobile phones and gis information, in: Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS '11, ACM, New York, NY, USA, 2011, pp. 54–63. doi:10.1145/2093973.2093982.

[11] L. Zhang, S. Dalyot, D. Eggert, M. Sester, Multi-stage approach to travel-mode segmentation and classification of gps traces, ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XXXVIII-4/W25. doi:10.5194/isprsarchives-XXXVIII-4-W25-87-2011.

[12] C. Lee, G. Yoon, D. Han, A context-based energy optimization algorithm for periodic localization in smartphones, in: Proceedings of the First ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems, MobiGIS '12, ACM, New York, NY, USA, 2012, pp. 86–92. doi:10.1145/2442810.2442826.

[13] F. Biljecki, H. Ledoux, P. van Oosterom, Transportation mode-based segmentation and classification of movement trajectories, International Journal of Geographical Information Science 27 (2013) 385–407.

[14] T. K. Rasmussen, J. B. Ingvardson, K. Halldrsdttir, O. A. Nielsen, Using wearable gps devices in travel surveys: A case study in the greater copenhagen area, in: Proceedings from the Annual Transport Conference at Aalborg University, 2013, pp. 1603–9696.

[15] P. Nitsche, P. Widhalm, S. Breuss, N. Brändle, P. Maurer, Supporting large-scale travel surveys with smartphones – A practical approach, Transportation Research Part C: Emerging Technologies 43 (2014) 212 –221, special Issue with Selected Papers from Transport Research Arena. doi:https://doi.org/10.1016/j.trc.2013.11.005.

[16] H. Safi, B. Assemi, M. Mesbah, F. Luis, H. Mark, Design and implementation of a smartphone-based system for personal travel survey: Case study from new zealand, in: Presented at Transportation Research Board 94th Annual Meeting, 2015.

[17] L. Stenneth, K. Thompson, W. Stone, J. Alowibdi, Automated transportation transfer detection using gps enabled smartphones, in: 2012 15th International IEEE Conference on Intelligent Transportation Systems, 2012, pp. 802–807. doi:10.1109/ITSC.2012.6338603.

[18] G. Xiao, Z. Juan, J. Gao, Inferring trip ends from gps data based on smartphones in shanghai, in: Proceedings of the Transportation Research Board 94th Annual Meeting, 2015, pp. 11–15.

[19] A. Prelipcean, G. Gidofalvi, Y. Susilo, Measures of transport mode segmentation of trajectories, International Journal of Geographical Information Science 30 (2016) 1–22. doi:10.1080/13658816.2015.1137297.

[20] S. Barbeau, M. A. Labrador, A. Perez, P. Winters, N. Georggi, D. Aguilar, R. Perez, Dynamic management of real-time location data on gps-enabled mobile phones, in: 2008 The Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, 2008, pp. 343–348. doi:10.1109/UBICOMM.2008.83.

[21] F. Ben Abdesslem, A. Phillips, T. Henderson, Less is more: Energy-efficient mobile sensing with senseless, in: Proceedings of the 1st ACM Workshop on Networking, Systems, and Applications for Mobile Handhelds, MobiHeld '09, ACM, New York, NY, USA, 2009, pp. 61–62. doi:10.1145/1592606.1592621.

[22] T. O. Oshin, S. Poslad, A. Ma, Improving the energy-efficiency of gps based location sensing smartphone applications, in: 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications, 2012, pp. 1698–1705. doi:10.1109/TrustCom.2012.184.

[23] I. Constandache, S. Gaonkar, M. Sayler, R. R. Choudhury, L. Cox, Enloc: Energy-efficient localization for mobile phones, in: IEEE INFOCOM 2009, 2009, pp. 2716–2720. doi:10.1109/INFCOM.2009.5062218.

[24] Z. Zhuang, K.-H. Kim, J. P. Singh, Improving energy efficiency of location sensing on smartphones, in: Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, MobiSys '10, ACM, New York, NY, USA, 2010, pp. 315–330. doi:10.1145/1814433.1814464.

[25] J. Paek, J. Kim, R. Govindan, Energy-efficient rate-adaptive gps-based positioning for smartphones, in: Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, MobiSys '10, ACM, New York, NY, USA, 2010, pp. 299–314. doi:10.1145/1814433.1814463.

[26] D. H. Kim, Y. Kim, D. Estrin, M. B. Srivastava, Sensloc: Sensing everyday places and paths using less energy, in: Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems, SenSys '10, ACM, New York, NY, USA, 2010, pp. 43–56. doi:10.1145/1869983.1869989.

[27] C. C. Robusto, The cosine-haversine formula, The American Mathematical Monthly 64 (1) (1957) 38–40.
URL http://www.jstor.org/stable/2309088

[28] P. Misra, P. Enge, Global Positioning System: Signals, Measurements, and Performance, Ganga-Jamuna Press, 2011.
URL https://books.google.com.mt/books?id=5WJOywAACAAJ

[29] A. Carroll, G. Heiser, An analysis of power consumption in a smartphone, in: USENIX, 2010.

[30] L. Hruska, Smart batteries and lithium ion voltage profiles, The Twelfth Annual Battery Conference on Applications and Advances (1997) 205–210doi:10.1109/BCAA.1997.574104.