An Energy-Efficient Method with Dynamic GPS Sampling Rate for Transport Mode Detection and Trip Reconstruction

Jonathan Milot[0000−0002−5470−7672], Jaël Champagne Gareau[0000−0002−1906−4157], and Éric Beaudry[0000−0002−4460−0556]

Université du Québec à Montréal
{milot.jonathan,champagne_gareau.jael}@courrier.uqam.ca
beaudry.eric@uqam.ca

Abstract. This paper presents a novel approach for trip reconstruction and transport mode detection. While traditional methods use a fixed GPS sampling rate, our proposed method uses a dynamic rate to avoid unnecessary sensing and waste of energy. We determine a time for each sampling that gives an interesting trade-off using a particle filter. Our approach uses as input a map, including transit network circuits and schedules, and produces as output the estimated road segments and transport modes used. The effectiveness of our approach is shown empirically using real map and transit network data. Our technique achieves an accuracy of 96.3% for a 15.0% energy consumption reduction (compared to the existing technique that has the closest accuracy) and an accuracy of 85.6% for a 56.0% energy consumption reduction.

Keywords: Particle filter · Transport mode detection · Trip reconstruction · Energy efficiency · Mobile device · GPS · Dynamic sampling

1 Introduction

The popularity of smartphones has brought out many services and applications based on their sensors (GPS, accelerometer, gyroscope, etc.) such as activity recognition [7], trip reconstruction (TR) [8,11] and transport mode detection (TMD) [3,13]. TR and TMD are both used for traffic study in urban planning to automate the process of data collection [12] instead of using the traditional survey method that is less precise and more expensive. Most GPS sensing are usually made at a fixed predetermined rate ranging from 1 second to 60 seconds [3,12]. However, these algorithms of reconstruction suffer from major drawbacks.

One problem is that a GPS device consumes a significant amount of energy. Using the GPS at 1 Hz consumes 143.1 mW on the HTC Dream and Google Nexus One models [5]. According to our experiments, the consumption is 439.3 mW for a Samsung Galaxy S8 and 397.44 mW for an Asus Zenphone 4 Max. The battery life of these smartphones ranged from 8 to 10 hours for normal daily use (cellular and Wi-Fi enabled, 4G web navigation when travelling in

public transit, etc.) when continuous geolocation was enabled. Sensing at a rate of 1 Hz is therefore hardly acceptable for users, since it drastically reduces their smartphone's autonomy. On another side, a lower rate decreases the accuracy of TR and TMD algorithms. Hence, there is a trade-off between the quality of estimation of the algorithm and the energy consumption (EC) of the GPS.

This ideal compromise highly depends on the road network and the smartphone's state after a certain time. Indeed, in a city's downtown, the number of different possible paths and the possible change of transport mode (e.g., bus stops, metro stations, etc.) makes it harder to do TR and TMD, because many paths and transport mode combination can explain the transition between the last two GPS sensing. In comparison, on a highway there is usually only one possibility (transport by *car* and shortest path between two points). The interval Δt between sensing should thus dynamically change depending on the position to achieve the optimal trade-off between precision and EC.

This paper presents a dynamic GPS sampling rate technique for path reconstruction and transport mode detection based on a particle filter that dynamically determines the moment to use the GPS sensor in order to get a compromise, depending on preferences, between energy consumption and accuracy.

2 Related Works

Since we address two related problems at once, TR and TMD, this section presents previous works related to one or both of these problems. Furthermore, we discuss their impacts on energy consumption.

2.1 Trip Reconstruction

The simplest TR approaches use a fixed GPS sensing rate. Usually, such approaches are tested with different fixed frequencies to show the decrease in accuracy when the rate increase. One approach is iterative based [11], where each node must be within a certain Euclidean distance from the corresponding GPS reading and directly linked to the previous associated node. If no node can be associated to a reading, the maximum distance is increased and another iteration begins. Another approach [10] uses a Hidden Markov Model (HMM) [4] to determine the most likely road segment for each GPS reading. The authors limit the EC of their algorithm by updating the sampling rate according to the mobile device state (stopped, normal road, highway). However, they do not test their approach with sampling rates higher than 30s to avoid an *arc-skipping* situation, which could greatly increase the error according to the authors. Finally, a model to generate a set of true potential paths and to associate likelihood to each of them is used in [1]. The model uses the speed, the time and the bearing of the mobile device to determine these likelihood. All these techniques highlight the same problem: an higher sampling rate to reduce EC directly leads to an increased error due to the *arc-skipping* problem. In this situation, the road

segments associated with two consecutive GPS readings may not be directly connected and a shortest-path algorithm must be used to link them [11,12]. This can induce errors, since some users may not have used the shortest path for some reason (e.g., construction site, personal preference, traffic, etc.).

2.2 Transport Modes Detection

TMD algorithms are generally based on machine learning (ML), since the goal is to classify data (GPS readings) among a defined and limited set of class (transport mode). While these approaches can achieve a good accuracy on average (more than 90%), they rely on a high GPS sampling rate (from 1s to 15s) and do not attempt to reconstruct the smartphone's trip. Some approaches also use data from other sensors, such as the smartphone's accelerometer or field sensor, to improve their accuracy [3]. Many TMD algorithms use ML techniques such as neural network (NN) [3,15] and derivatives such as convolutional neural network (CNN) [9]. Other ML techniques such as Random Forest [13] and Support Vector Machine [2] have also been experimented.

Table 1: Transport Mode Detection accuracy in related works

Approach	Sampling Rate (s)	Walk	Car	Bus	Average
[13]	15	98.9%	80.8%	93.0%	93.8%
[2]	60	93.8%	88.5%	58.3%	88.0%
[3]	1	95.0%	72.0%	84.0%	83.8%
[15]	1	98.5%	94.2%	88.4%	94.4%
[9]	1-5	95.7%	67.4%	81.1%	84.8%

The general accuracy of these techniques ranged from 84.8% for the CNN [9] to 94.4% for the NN with particle swarm optimization [15]. However, these average accuracies hide a deeper phenomenon that occurs in all reviewed study: distinguishing a *walk* is generally easier than a *car* or a *bus*. Table 1 presents the accuracy for the different papers previously cited. The *walk* accuracy is always the highest (more than 93.8%), while the *car* and *bus* can vary a lot (from 67.4% to 94.2% and 58.3% to 93.0% respectively). This is mainly due to the resemblance between the motorized transport modes (*car* and *bus*) regarding speed and acceleration. Thus, they are easily mixed up and wrongly classified. A note on the results of [2]: they claim to have an average accuracy of 88.0% for a sampling rate of 60s. While this seems impressive for such a low rate, it is important to note that 52.4% of their data is labelled as *walk*. Since this is the easiest transport mode to detect, this greatly increases their average accuracy.

2.3 Combined Approach

A technique to do both TR and TMD has been proposed by [8]. They use a conventional GIS-based map-matching algorithm to reconstruct the path and a

rule-based algorithm to identify transport modes (*walk*, *bicycle*, *bus* and *car*). Their average error on TR is 21% for a sample rate of 1s, which is worse than algorithms previously cited who exclusively reconstruct path. Their accuracy for transport mode detection is 92%, which is similar to other approaches.

3 Model and Algorithm

The goal of our method is to estimate the paths and the transport modes used during a trip with the GPS sensor of a smartphone while minimizing the EC. To achieve this, we use a particle filter to estimate the smartphone's state according to GPS readings made at a dynamic rate. This approach novelty resides in the use of the GPS sensor only when really needed, unlike other methods that make GPS readings at a given fixed rate. The general outline goes as follows:

1. Make a GPS reading to estimate the smartphone's state s.
2. Simulate the evolution of s until a time t in the future.
3. Determine a time $t^\star \in]0, t]$ offering an interesting compromise between accuracy and energy consumption. When $t^\star$ is reached, return to 1.

We model the space in which the smartphone evolves as a graph $G = (N, A)$. A node is a point $n = (n.\text{Lat}, n.\text{Long}) \in N$ on the map, and an arc is a road segment $a = (n_{\text{from}}, n_{\text{to}}, v_{\max}, T_e) \in A$ containing respectively the start and end vertex of the segment as well as the maximum possible speed on the segment and the set of all transport modes that can cross a. Each arc thus represents a way to move between two vertices by using a transport mode (e.g., a road segment, a subway tunnel, a train rail, etc.). The smartphone's state at time t is a tuple $s_t = (p, m, v, P)$, where p, m, v and P are respectively the smartphone's current position (a point lying on an arc in A), current transport mode, current speed and actual path travelled (list of nodes and transport modes used).

The smartphone's state s is not directly observable, but is estimated from an external sensor (GPS). Since this sensor is not perfect and contains noise, $s.p$ is modelled as a Gaussian distribution $\mathcal{N}(x, \sigma^2)$, where x is the projection of the coordinate returned by the GPS on the nearest arc a and σ^2 is the accuracy of the GPS. It is important to note that the smartphone's position distribution may overlap other allowed arcs according to the current transport mode if x is near the extremity of a. The evolution of $s.p$ after a time interval Δt is given by

$$s_{t+\Delta t}.p = s_t.p + \mathcal{N}(s.v, \sigma^2) \times \Delta t. \tag{1}$$

We need to take into account the fact that $s.v$ can vary during Δt (due to traffic, road elevation, etc.). Over a long Δt, we consider that a Gaussian distribution is a good speed approximation at which the smartphone travels.

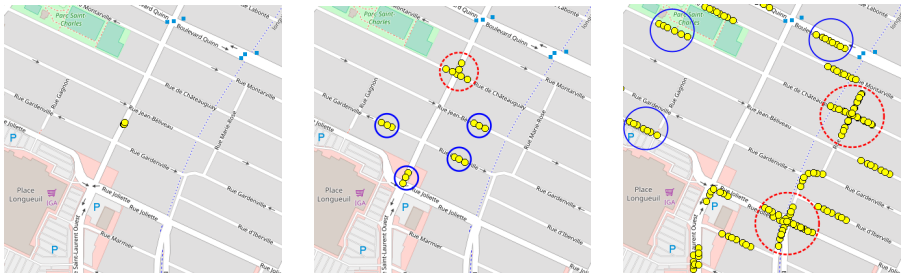
Using this formalism, the goal of our algorithm is to determine the list of nodes and transport modes $P = \langle (n_0, m_0), (n_1, m_1), \ldots, (n_{k-1}, m_{k-1}), (n_k, m_k) \rangle$ taken during a trip, where m_i is the transport mode used to reach n_i.

Since the evolution of the state s can hardly be modelled with parametric functions (smartphones can be at different places after Δt, each with multiple

Algorithm 1 The particle filter algorithm

function PARTICLE FILTER($\mathcal{X}_{t-1}$, u_t, z_t)
  $\bar{\mathcal{X}} = \mathcal{X}_t = \emptyset$
  for m = 1 to M **do**
    sample $x_t^{[m]} \sim p(x_t|u_t, x_{t-1}^{[m]})$
    $w_t^{[m]} = p(z_t|x_t^{[m]})$
    $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
  for m = 1 to M **do**
    draw i with probability $\propto w_t^{[i]}$
    add $x_t^{[i]}$ to $\mathcal{X}_t$
  return $\mathcal{X}_t$

different uncertainties, paths and transport modes), we use a particle filter. A generic implementation [14] is presented in Algorithm 1. Simply put, a particle filter's goal is to approximate a belief state bel(x_t) by a set of particles $\mathcal{X}_t$ of size M randomly constructed by the control data u_t and the sensor data z_t. In our problem, u_t is simply the time passed since the last sampling and z_t, the data returned by the GPS sensor. Figure 1 shows a visual example of the particle filter and the evolution of $\mathcal{X}_t$ between two GPS readings. Compare the particles circled in blue (resp. red) in Fig. 1b and in Fig. 1c. We see that many instances of $s_{30}.p$ are overlapping with different path.



(a) $\boldsymbol{t = 1}$. The uncertainty on $s_1.p$ is low; few particles are needed.

(b) $\boldsymbol{t = 10}$. $s_{10}.p$ can be 5 different positions.

(c) $\boldsymbol{t = 30}$. More particles needed to represent the increased uncertainty

Fig. 1: Evolution of $\mathcal{X}_t$ over 30 seconds for the *car* transport mode

One of the challenges of the particle filter is to determine how and when to resample $\mathcal{X}_t$. After a certain amount of time, the distribution of $\mathcal{X}_t$ may become too coarse to adequately approximate bel(x_t). Therefore, a resample is eventually needed. In our case, a resampling implies a new sensor observation z_t (which incur energy consumption). We implement two methods to determine the next resampling time and avoid unnecessary sensing.

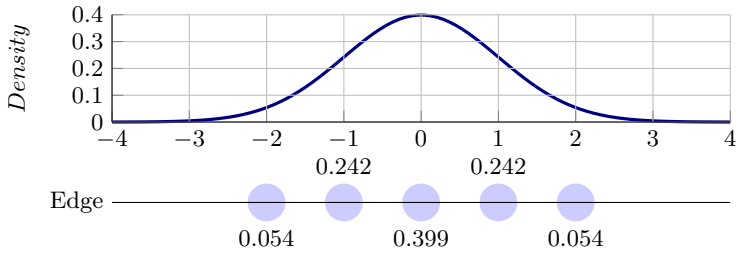


Fig. 2: Discretization of $\mathcal{N}(0,1)$ at every σ on $[-2\sigma, 2\sigma]$.

Firstly, to avoid too coarse particle distribution, we discretize all possible states with particles instead of sampling a fixed M particles from $p(x_t|u_t, x_{t-1}^{[m]})$ (Line 4 of Algorithm 1). Rather than calculating the density probability of $\text{bel}(x_t)$ with the density of particles, we associate a weight to each particle (see Fig. 2). When a smartphone reach position n_{to} of its current arc, the particle splits itself on all outgoing edges of n_{to}. An example of this splitting is shown in Fig. 1b. This ensures that all possible states are covered by the state space.

Secondly, we compute a score for $\mathcal{X}_t$ with the following formula:

$$\text{score}(\mathcal{X}_t) = \sum_{x \in \mathcal{X}_t} \Big(P(x) \times x_{acc} \Big) - \frac{2}{1 + e^{\lambda t}}. \tag{2}$$

This score represents the trade-off between the EC and the average estimated accuracy (TR and TMD) the algorithm would achieve if a GPS sensing was done at t, the elapsed time since the last sensing. The λ parameter in Eq. (2) controls the trade-off between EC and accuracy. A small λ saves more energy, while a bigger one gives an higher accuracy. Fig. 3 shows an example of the evolution of $\text{score}(\mathcal{X}_t)$ for different values of λ. The next sensing and resampling are done at the time t that maximizes $\text{score}(\mathcal{X}_t)$. The variable x_{acc} in Eq. (2) is defined by

$$x_{acc} = \sum_{x' \in \bar{\mathcal{X}}_t} P(x') \times \frac{|x'_{Path^+}| + |x'_{Path^-}|}{|x_{Path}|}, \tag{3}$$

where x is the particle currently computed, $|x'_{Path^+}|$ is the length of path that x' has, but not x, $|x'_{Path^-}|$ is the length of path that x' doesn't have, but x has and $\bar{\mathcal{X}}_t$ is the set of particles at a lower distance from x than the GPS sensing error. x_{acc} represents the average error on the state estimated when x is the true smartphone's state and a GPS sensing is made. Due to noise induced by the GPS sensor, the true smartphone's state could be mistaken for any particles of $\bar{\mathcal{X}}_t$ (which can have different paths and transport modes).

The resampling is pretty straightforward: $\text{score}(\mathcal{X}_t)$ is computed on $[0, T[$, where T is a time limit after which no better trade-off can be obtained. The time $t^\star$ offering the ideal compromise between accuracy and energy consumption

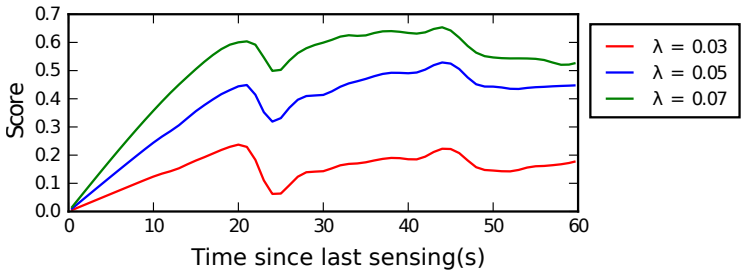


Fig. 3: score($\mathcal{X}_t$) for 3 different λ. With $\lambda = 0.03$, the next sensing is done after 21s. With the other two λ, the next sensing is done after 44s.

is retained (i.e., $t^\star = \text{argmax}_t \, \text{score}(\mathcal{X}_t)$). Then, a GPS sensing is done at time $t^\star$. The particles in a radius of $2\sigma^2$ are simply kept instead of generating new particles (Line 8 of Algorithm 1), where σ^2 is the radius (in meters) of 68% confidence given by the GPS sensor. In the event that no particles are in this radius, it is gradually extended by step of one σ^2. The weight of each particle is then updated according to their distance from the GPS sensing.

The change of transport mode during a trip is considered in two circumstances:

1. **Special nodes.** Some transport mode changes can only happen on predetermined nodes, such as *walk*→*bus*. When a particle p reaches these nodes, a copy of p is instantiated with a predefined different transport mode, changing its transition behaviour between nodes.
2. **During resampling.** Some other transport mode changes can occur practically anywhere, such as *car*→*walk*. Since those transport mode changes don't occur often, we only consider this change at resampling. Based on our results, the error induced by this simplification is acceptable compared to the combinatorial explosion that would arise if a transport mode change was considered every second. Therefore, after each resampling, a new state is added to $\mathcal{X}_t$ for every transport modes not already present in $\mathcal{X}_t$.

4 Experimentations

Experimentations were done in the metropolitan region of Montréal, Canada due to its range of different road network configurations (e.g., dense downtown, highways, suburbs and country roads on the peripheral region). Moreover, the presence of multiple bus and subway lines allows us to consider multiple transport modes. In this experimentation, the transport modes considered are *walk*, *car*, *bus* and *subway*. Also, score($\mathcal{X}_t$) has been evaluated on the interval $[0, 300[$.

4.1 Data For the Model

The map data comes from the OpenStreetMap (OSM) project, while the transit network schedule comes directly from the transport societies (STM, RTL, AMT,

STL) that cover the region of Montréal. Bus and subway stations were integrated to the OSM data by being connected to the nearest node. To obtain the average speed on every road segment for the *car* transport mode, we used data collected by the City of Montréal. With them, we computed an average speed of 28.60 ± 9.49 km/h. The *walk* average speed has also been found to be between 3.46 ± 0.65 km/h to 4.93 ± 0.68 km/h depending on the place he is and his gender [6].

4.2 GPS Data Collection

Table 2: GPS data collected

Transport mode	Runtime (hours)	Number of GPS readings
Walk	8.03	28514
Car	11.94	42768
Bus	5.13	17290
Subway	2.06	390
Total	27.16	88962

For two months, we recorded daily trips with an Android application on an Asus Zenphone 4 Max smartphone (model ZC554KL). All trips were collected at different times of the day and have different path. This ensures a variety regarding the traffic, weather conditions and transit network schedules. During these trips, the GPS sensor made readings at a rate of 1 Hz and the user was asked to manually specify its current transport mode (*walk*, *car*, *bus* or *subway*). A summary of the collected data is presented in Table 2. Few GPS readings have been recorded for the *subway* transport mode because GPS signal can't be received underground (but some stations are close enough to the surface to allow some signals to pass through).

5 Results

Table 3: Algorithm accuracy with different λ

λ	ASR (s)	Walk	Car	Bus	Subway	Avg TMD	TR Error
0.01	20.9	0.996	0.988	0.936	0.932	0.963	0.023
0.03	32.5	0.920	0.982	0.871	0.902	0.919	0.028
0.05	48.6	0.908	0.969	0.806	0.890	0.893	0.031
0.07	59.2	0.876	0.934	0.772	0.885	0.867	0.039
0.09	66.0	0.859	0.932	0.758	0.876	0.856	0.041

Table 4: Transport mode detection matrix confusion for $\lambda = 0.05$

Detected \ Real	Walk	Car	Bus	Subway
Walk	0.908	0.030	0.049	0.013
Car	0.001	0.969	0.021	0.009
Bus	0.048	0.055	0.806	0.091
Subway	0.048	0.018	0.044	0.890

Using the collected data, our approach was tested with different λ values to analyze the trade-offs between accuracy and EC. Table 3 shows for every tested λ the average sampling rate (ASR), the accuracy obtained with our technique for the detection of every transport mode and the average on all of them, as well as the trip reconstruction error. As expected, a higher λ implies a lower GPS rate. With $\lambda = 0.01$, a reading is made at an average sampling rate of 20.9s, while with $\lambda = 0.09$, the average is at 66.0s.

5.1 Transport Mode Detection

In order to make a fair comparison between our results and those in the related works, we looked for public implementation of related works' algorithms to test them on the same datasets. However, these datasets are, to the best of our knowledge, not publicly available. Given the usage of machine learning in these techniques, a reimplementation was hardly possible, since those algorithms are very sensivitive to the input dataset. Hence, we directly compare our results to those found in the related works.

The TMD accuracies we obtained are similar to those found in related works. It ranges from 96.3% with a $\lambda = 0.01$ to 85.6% with a $\lambda = 0.09$. The result for $\lambda = 0.01$ had a better average accuracy than any other approach cited (the closest average accuracy being 94.4% [15]) and was achieved with a lower sampling rate. For example, with $\lambda = 0.01$, the average sampling rate is 20.9s while [15] had a sampling rate of 1s. In comparison to [2], who used the highest sampling rate of 60s and had an accuracy of 88.5% for *car* and 58.3% for *bus* [2], our technique had an accuracy of 93.2% (*car*) and 75.8% (*bus*) for an average sampling rate of 66.0s with $\lambda = 0.09$. The TMD errors for $\lambda = 0.05$ are shown in a confusion matrix in Table 4.

The lowest accuracy is for the *bus* transport mode. It is falsely detected as *car* 5.5% of the time, which is understandable since buses and cars drive at a similar speed. However, *car* are less often mistaken for *bus*, only 2.1% of the time. This is explained by the fact that buses always share the road with cars (except some rare bus-only lanes) but cars can often be on roads with no buses route. *Bus* and *subway* transport can be confused with *walk* when the algorithm has difficulty to determine the exact bus stop or subway station a user has taken. Furthermore, *subway* are often (9.1%) confused with *bus*. This is due to the presence of a bus line running parallel to a subway line. Often, both routes can

explain the transition between the last GPS location before entering a subway station and the first one after exiting the other one. Currently, our algorithm doesn't consider an absence of GPS reading as being underground. However, the presented approach's accuracy could be improved by increasing the weight of underground states when a weak GPS signal is detected.

5.2 Trip Reconstruction

Ground truth for paths taken was generated from our GPS readings and corrected by hand. We compared the path estimated by our technique to the ground truth. The path error was computed with the formula [11]:

$$E = \frac{|P^{+}_{Estimated}| + |P^{-}_{Estimated}|}{|P_{Real}|}, \tag{4}$$

where P_{Real} is the real path, $P^{+}_{Estimated}$ is the part of the estimated path in extra compared to P_{Real} and $P^{-}_{Estimated}$ the part lacking from P_{Real}. TR error ranged from 2.3% with $\lambda = 0.01$ (ASR of 20.9s) to 4.1% with $\lambda = 0.09$ (ASR of 66.0s). Compared to their respective sampling rate, this is better than the accuracy found in the related works.

5.3 Energy Consumption

The energy efficiency of our approach is demonstrated by running our algorithm on smartphones and measuring the energy consumption using Android's *BatteryManager* API. Before every test, the battery was fully charged and all other applications were closed. Wifi was disabled and 4G enabled. Running the algorithm directly on the phone would consume more energy than the amount saved by making less GPS sampling. Hence, the 4G connection is required to communicate with a server running the algorithm. We also measured the EC for fixed GPS sampling rates. Table 5 shows the results obtained.

Obviously, for an equal ASR, the presented approach uses more energy than those that uses a fixed sampling rate because of the 4G usage (e.g., 286.28 mW versus 272.49 mW for a sampling rate of around 20s). However, we can have a higher or equivalent accuracy with our approach while using less energy. Most other approaches use a sampling rate of 1s resulting in an EC of 336.82 mW, the corresponding average accuracy ranging from 84.0% to 93.0%. In comparison, the presented approach consumes only 286.28 mW for an accuracy of 96.3% or 210.50 mW for an accuracy of 91.9%. This means a 15.0% EC reduction for a 3.3% higher accuracy and a 37.5% EC reduction for an equivalent accuracy. Furthermore, accuracy remains acceptable with greater sampling rate. With an ASR of 66.0s, the presented approach still achieves an average accuracy of 85.6% for an EC of 148.18 mW, a 56.0% EC reduction for an equivalent accuracy compared to [15].

Table 5: Energy consumption according to the GPS sampling rate

Method	λ	ASR (s)	EC (mW)
Fixed GPS sampling rate	-	1	336.82
	-	20	272.49
	-	40	172.67
	-	60	146.74
	-	80	114.02
Dynamic GPS sampling rate	0.01	20.9	286.28
	0.03	32.5	210.50
	0.05	48.6	170.79
	0.07	59.2	151.26
	0.09	66.0	148.18

6 Conclusion

In this paper, a novel approach for trip reconstruction (TR) and transport mode detection (TMD) has been presented. It reduces significantly smartphone energy consumption by using the GPS sensor only when necessary while achieving similar or higher accuracy compared to state-of-the-art methods. This approach uses a particle filter that estimates the smartphone's state evolution and the average resulting error if a GPS sampling was made at every moment. These average estimated errors are then weighted in a smartphone energy consumption model to determine the optimal time to do the next sampling. This is to the best of our knowledge the first approach using dynamic GPS rate depending on the underlying road and transit network. Finally, field tests demonstrated the approach's accuracy and energy saving compared to other methods. In the best case, the presented approach allowed an increase of 3.3% in the average accuracy and a 15.0% energy consumption reduction compared to other approaches and a 37.5% to 56.0% energy consumption reduction for an equivalent accuracy.

Our experimentation only looked at one smartphone model (i.e. Asus Zenphone 4 Max smartphone, model ZC554KL). More tests should be done to compare the power consumption saving with different models, since each part can have a different power consumption, i.e., GPS sensor, 4G antenna, CPU, etc. This could lead to different ratios of energy consumption saving on models where other parts would consume drastically more than the GPS sensor.

Currently, our transport model uses historic data regarding travel speed on certain road segments to determine an *a priori* travel speed. However, it does not consider the possible punctual slowdowns due to traffic. Because real-time data on traffic is hard to obtain, research has already been made toward predicting and modelling traffic on a road network. Such methods would improve our transport model in order to better predict the user's state evolution when travelling by transport mode relying on road segments (e.g., *car* and *bus*), increasing the proposed approach accuracy.

References

1. Bierlaire, M., Chen, J., Newman, J.: A probabilistic map matching method for smartphone GPS data. Transportation Research Part C: Emerging Technologies **26**, 78–98 (2013)
2. Bolbol, A., Cheng, T., Tsapakis, I., Haworth, J.: Inferring hybrid transportation modes from sparse GPS data using a moving window SVM classification. Computers, Environment and Urban Systems **36**(6), 526–537 (2012). https://doi.org/10.1016/j.compenvurbsys.2012.06.001
3. Byon, Y.J., Liang, S.: Real-Time Transportation Mode Detection Using Smartphones and Artificial Neural Networks: Performance Comparisons Between Smartphones and Conventional Global Positioning System Sensors. Journal of Intelligent Transportation Systems **18**(3), 264–272 (2014). https://doi.org/10.1080/15472450.2013.824762
4. Cappé, O., Moulines, E., Rydén, T.: Inference in hidden markov models. In: Proceedings of EUSFLAT Conference. pp. 14–16 (2009)
5. Carroll, A., Heiser, G., et al.: An analysis of power consumption in a smartphone. In: USENIX annual technical conference. vol. 14, pp. 21–21 (2010)
6. Chandra, S., Bharti, A.K.: Speed Distribution Curves for Pedestrians during Walking and Crossing. Procedia - Social and Behavioral Sciences **104**, 660–667 (2013). https://doi.org/10.1016/j.sbspro.2013.11.160
7. Cheng, W., Erfani, S.M., Zhang, R., Ramamohanarao, K.: Markov Dynamic Subsequence Ensemble for Energy-Efficient Activity Recognition. Proceedings of MobiQuitous, Australia, 2017 p. 10 (2017). https://doi.org/10.475/123_4
8. Chung, E.H., Shalaby, A.: Transportation Planning and Technology A Trip Reconstruction Tool for GPS-based Personal Travel Surveys. Transportation Planning and Technology **28**(5), 381–401 (2005)
9. Dabiri, S., Heaslip, K.: Inferring transportation modes from GPS trajectories using a convolutional neural network. Transportation Research Part C: Emerging Technologies **86**, 360–371 (2018). https://doi.org/10.1016/j.trc.2017.11.021
10. Fang, S., Zimmermann, R.: EnAcq: Energy-efficient GPS trajectory data acquisition based on improved map matching. Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems pp. 221–230 (2011). https://doi.org/10.1145/2093973.2094004
11. Li, X., Yuan, F., Lindqvist, J.: Feasibility of duty cycling gps receiver for trajectory-based services. 13th IEEE Annual Consumer Communications & Networking Conference (CCNC) (2016). https://doi.org/10.7282/T3VM4F56
12. Patterson, Z., Fitzsimmons, K.: DataMobile: Smartphone Travel Survey Experiment. Transportation Research Record Journal of the Transportation Research Board **15**(2594), 35–43 (2016)
13. Stenneth, L., Wolfson, O., Yu, P.S., Xu, B.: Transportation mode detection using mobile phones and GIS information. Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems p. 54 (2011). https://doi.org/10.1145/2093973.2093982
14. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics (Intelligent Robotics and Autonomous Agents). MIT Press (2005)
15. Xiao, G., Juan, Z., Gao, J.: Travel Mode Detection Based on Neural Networks and Particle Swarm Optimization. Information **6**(3), 522–535 (aug 2015). https://doi.org/10.3390/info6030522