BluePark: Tracking Parking and Un-parking Events in Indoor Parking Lot

Sonia Soubam
IIIT Delhi
sonias@iiitd.ac.in

Dipyaman Banerjee
IBM Research Lab, India
dipyaban@in.ibm.com

Vinayak Naik
IIIT Delhi
naik@iiitd.ac.in

Dipanjan Chakraborty
Mosix Technologies
dipanjan.chakraborty@gmail.com

ABSTRACT

Finding a parking spot in a busy indoor parking lot is a daunting task. Retracing a parked vehicle can be equally frustrating. We present BluePark, a collaborative sensing mechanism using smartphone sensors to solve these problems in real-time, without any input from user. We propose a novel technique of combining accelerometer and WiFi data to detect and localize parking and un-parking events in indoor parking lot. We validate our approach at the basement parking of a popular shopping mall. The proposed method out-performs Google Activity Recognition API by 20% in detecting drive state in indoor parking lot. Our experiments show 100% precision and recall for parking and un-parking detection events at low accelerometer sampling rate of 15Hz, irrespective of phone's position. It has a low detection latency of 20 seconds with probability of 0.9 and good location accuracy of 10 meters.

1. INTRODUCTION

Shopping malls in India are one of the largest shopping and entertainment zones with about 50 Million footfalls per month [2]. The mall sector is growing in India with a current count of approximately 570 malls [2]. Several parking facilities in malls in India are underground and they have heavy traffic. It is difficult to detect free parking spots as driver have limited visibility range and parking lots are quite large. Advanced parking systems like that in developed nations have limited adoption in India due to several practical issues such as cost, maintenance, mischievous users tampering with sensors, etc. Furthermore, for retrieving parked cars, the only aid available is marked pillars. Users often forget to note down manually the pillar numbers leading to a frustrated experience where they have to walk around with heavy shopping bags to find the parked car. In this paper, we propose *BluePark*, a smart parking system for indoor parking facility, that can detect when and where a car is parked and un-parked using smartphone sensing in real-time, without any user input. While detection and localization of parking events helps users to retrace back to their vehicles, un-parking events information help other users to find empty parking spots. The same BluePark system can also be used by parking administration to monitor the state of the indoor parking space. The sharing of information about parking and un-parking can be done anonymously as users only need to know which parking spots are currently vacant and which are occupied, thus preserving privacy of the users.

Real-time parking systems can be classified into two categories (a) infrastructure based and (b) crowdsource based. In infrastructure based systems additional hardware, such as occupancy sensors and wireless transceivers, are installed either on the parking spots [16, 9] or vehicles [10]. High cost involved in procuring, installing and maintaining these equipment have deterred their use. Crowdsource based systems use information collected from users [11, 3] through web applications and mobile phones. Crowdsource based approaches can be further classified into two, manual reporting and automatic reporting. In manual reporting[1], users report about empty parking spot using a mobile or web application. The collected data is transmitted to a central system, from which other drivers can find recently vacated parking spots. The burden of manual tagging, although incentives are in place [11], is one of the main disadvantages of such approaches. Mischievous users may not want to share information or provide false information to divert competition from area of interest [7]. In crowdsource based approaches using automatic reporting, one need not deal with the problem of incentives. Using sensors present on smartphone sensors, occupancy status is automatically detected and reported to a system. Drivers can use this information to find a parking spot, and are not bothered to report their status manually. Our work falls in this category. GPS systems do not work indoors which restricts us from locating the parked vehicle or tracking its trajectory. This makes automatic parking detection systems for outdoors [12] cannot be used for indoors.

The aim of this paper is to build a smartphone-based approach to detect "when and where have I parked" and with "zero" cognitive load on the user. We note that any parking or un-parking event, almost all the time, is associated with a change of locomotive state of the user. Parking events imply a transition from a $drive$ state to a $walk$ state of the user, whereas un-parking events imply the opposite transition [12]. We use the same definitions in this paper. We propose a novel approach of combining accelerometer and WiFi data to accurately detect $walk$ and $drive$ states in indoor parking environment. We further design a light-weight algorithm that quickly detects the state transitions, which translates to parking and un-parking event, from user's state history with high precision and minimum detection latency. We define detection latency as the time difference between actual time of the event and that reported by our algorithm. We use the automatically detected parking event to find out the parking location. Latency in detecting the parking event directly affects localization accuracy. The latency is an important reliability factor to ensure the parking availability information being provided to the driver is not obsolete.

Use of WiFi is not limited to data communication only, it is widely used for localization on smartphone. We use an indoor-localization system based on the Horus method [18] that uses WiFi signals present at the time of event occurrence to derive the location of the parked/un-parked vehicle. BluePark uses WiFi for two purposes, (a) in an innovative manner, by fusing its data with that of accelerometer to detect locomotive state of the users, and (b) in localizing users. BluePark's localization method is discussed in detail in subsection 5.3.

The state detection and state transition detection algorithms run entirely on the phone, and localization of the parking spot require communication with an external server. Continuous monitoring for parking and un-parking events leads to unnecessary phone power usage. It leads to false positives, where state transitions similar to that of parking and un-parking events occurring outside the indoor parking facility are reported to our central system. To avoid such instances, we use Geo-fencing technique, which triggers event detection module only when the user is inside the targeted indoor parking region. Localization query to external server is a one time request and it is not power hungry. The whole system is automatic and does not require any intervention from the user. BluePark was deployed in the indoor parking area of a popular shopping mall in New Delhi. It utilized the existing WiFi APs deployed in the facility and did not require any extra infrastructure.

The primary contributions of this paper are as follows:

1. An algorithm, for detecting drive and walk states in indoor parking areas, which outperforms Google Activity Recognition API's accuracy by more than 20%. Our approach is independent of how the phone is held. We achieve this accuracy by way of proposing
    (a) A rule-based fusion of WiFi and accelerometer data
    (b) Adaptive DBSCAN, a modification of traditional DBSCAN algorithm, that allows unsupervised real-time detection of states using accelerometer data
    (c) A WiFi-based metric that differentiates between moving and not moving in indoor environments
2. Based on our state-detection algorithm, we propose an algorithm for detecting and localizing parking and un-parking events that has
    (a) 100% precision and recall even at low sampling frequency of 15Hz
    (b) Low detection latency of 20 seconds with probability of 0.9
    (c) Good parking localization accuracy of 10 meters

The rest of the paper is organized as follows: Section 2 discusses related work. Section 3 describes challenges in detecting and localizing parking and un-parking events in indoor parking spaces. The system architecture and working of BluePark is discussed in Section 4. Section 5 describes the algorithms used in BluePark. We compare the performance of BluePark with other approaches in Section 6. Finally, future work and conclusion are described in Section 7.

2. RELATED WORK

Although we are looking at the problem of parking, it overlaps with the problem of detecting mode of transportation. In this section, we will look at existing work in both the areas.

2.1 Parking Detection System

Lan et al [8] used smartphone sensor based personal dead reckoning system to localize and track the trajectory of driver, given the floor plan of the building. Their system is based on the assumption that: if a driver is approaching the area where he has parked his vehicle, then he is most likely to un-park, and the parking space will soon be vacated. Our work detects both parking and un-parking automatically.

ParkSense [13] uses parking payment smartphone application provided by the parking management to detect parking event and captures WiFi signature of the location where the vehicle is parked for outdoor environments. This WiFi signature is used to detect the driver's return to the parked vehicle location. They used Jaccard Index of visible WiFi APs to determine if the driver has started driving to detect un-parking event. Our work is partly inspired from the WiFi based approach of ParkSense with additional challenges associated with indoor parking facility. For an outdoor environment, the visible WiFi APs while driving are unique. While driving in an enclosed indoor parking facility, often slower as compared to outdoors, the visible APs may or may not repeat. Jaccard Index is not able to utilize this information of repeatedly visible WiFi APs. So, we propose a modified version of Jaccard Index, which is discussed in the Approach Section 6.

Nandugudi et al [12] proposed PocketParker, a crowd-sourcing system using smartphone to predict parking lot availability. They used existing activity detection approaches to detect user state, and transitions between states to detect parking and un-parking. The main focus of their work is to create a prediction model, based on parking and un-parking events of user and how drivers not using PocketParker affects the parking estimation. The focus of our paper is on accuracy and latency of the event detection. Our work complements their work.

2.2 Transportation Mode Detection Using Smartphone Based Sensors

Ubiquity of accelerometer in contemporary smartphones, and its low power consumption has made it a popular choice for context detection. While some of these studies were offline because of complexity of the classifier [17], some others were real-time classifiers [14] which were trained offline. Hemminki et al. [6] proposed a hierarchical accelerometer-based fine grained identification of transportation mode to distinguish walking/stationary and type of motorized vehicle (bus, train, metro, or train). The existing classifiers are supervised classifiers, which require extensive training to account for dependence of accelerometer data on user and phone position. Our proposed accelerometer based walking state identifier is un-supervised, real-time, and independent of the phone position and user.

Transportation mode detection using features extracted from GPS measurements has been proposed [19]. Reddy et al. [14] used a combination of GPS and accelerometer to determine whether user is stationary, walking, running, biking, or in a motorized vehicle, with an accuracy of over 90%. A disadvantage of GPS based approach is its high power consumption, and unreliability in indoor environments. In this paper, our design scenario is an indoor parking system and therefore, GPS information cannot be used. As mentioned before, Jaccard Index of the visible WiFi APs has been used for transportation mode detection in ParkSense [13]. RSSI is a potential metric for detecting motion in WiFi based system. Previous study [15] have shown that factors, such as multi-path propagation, influence the WiFi signal strength in indoor environments and therefore require complicated solutions. In this paper we present a simple approach to identify *drive* state. We find that taking frequency of occurrences of WiFi APs is sufficient for our purpose.

3. PROBLEM DESCRIPTION

In this paper, we describe a smartphone based sensing system, which enables users to accurately track the time and location of parking and un-parking events of their vehicles, in real time, without any input from the user and irrespective of the position and orientation of the phone.

As discussed in Section 1, parking and un-parking events are associated with change of locomotive states of the user. While parking is typically characterized by the transition $drive \rightarrow still \rightarrow walk$, un-parking is characterized by the transitions $walk \rightarrow still \rightarrow drive$. Detecting these two state transitions will give us a solution to identify parking and un-parking events. Moreover, as the vehicle location does not change during the $still$ state, it is sufficient to only detect and localize the $drive \rightarrow walk$ and $walk \rightarrow drive$ transitions. This approach has an added advantage. It can discard the $drive \rightarrow still$ and $still \rightarrow drive$ transitions, which may occur due to frequent stops a vehicle makes in an indoor parking space and can lead to false positives. Given the above observations, we divide our problem into three sub-problems:

- **State detection**: To detect locomotive state of user in real time, e.g., *drive*, *still*, and *walk* using their smartphone based sensors. In order to accurately detect the time and location of user's locomotive state transition in an indoor area using smartphone sensors, it is imperative to detect the locomotive state of the user correctly and with minimum latency. Incorrect detection of user state will have an adverse effect on the accuracy of detecting the state transitions. Similarly, delay in detecting will lead to miscalculating the time of the state transition, which in turn can result in an erroneous localization of the parking. Therefore, locomotive state detection must be done accurately, continuously and at a high frequency.

  There exists solutions to detect driving state, for example Google Activity Recognition API. However, in an indoor parking facility due to low speed and acceleration of the vehicle, existing solutions often cannot accurately detect *drive* state. This observation is verified in Table 3, which shows accuracy of Google Activity Recognition API for driving state detection in indoor parking.

- **Event detection**: To detect time of parking or un-parking event for user by detecting the $drive \rightarrow walk$ and $walk \rightarrow drive$ transitions respectively. While we want to be as responsive as possible in detecting such state transitions, we observe that such transitions occur seldom in an indoor parking space for a given user. Therefore, besides detecting state transitions accurately and with minimum latency, we need to consider history of states before and after such transitions to ensure they signify actual parking and un-parking events and discard transitions, which are transient. For example, sudden jerk of the phone or presence of speed-breakers can often lead to faulty detection of *walk* or *drive* state by the sensors resulting in such false state transitions.

- **Event localization**: To determine location of the parking and un-parking events for the user. In order to find vacant parking spots and to retrace back to the parked vehicle, we need to know the locations, where parking and un-parking event occured. With the absence of GPS signals in indoor spaces, we have to use other smartphone sensors, such as WiFi, to localize these events. However, while in a moving state, *drive* or *walk*, the

WiFi RSSI recorded by user's smartphone can change significantly with time. The accuracy of detection depends on latency of event detection, as any delay in detecting the state transitions can erroneously localize the parking or un-parking events to a wrong parking spot.

4. ARCHITECTURE OF THE SYSTEM

In this section, we give an overview of BluePark's system architecture. It has two components: a mobile and a cloud service as shown in Figure 1.

4.1 Mobile Client

The mobile client, an Android application running on the smartphone, is responsible for the state detection and the event detection. State detection is done using a combination of accelerometer and WiFi based sensing techniques. An event detection module monitors the detected state transitions corresponding to parking and un-parking events, which is reported to the cloud service. It runs parallel to the state detection module and uses a novel change detection technique to detect the transition with minimum latency. WiFi signals received by users' smartphones at the time of the transition is sent to the cloud service to identify the location of the parked or un-parked vehicle.

4.2 Cloud Service

The cloud service provides event localization and parking occupancy status. WiFi fingerprint information of the various parking spots are stored in the service's database. Given a WiFi signature, the cloud service returns the most probable parking spot. The cloud service maintains occupancy status of each parking spot in a *Parking Entries* log using the parking and un-parking events reported by the mobile client.

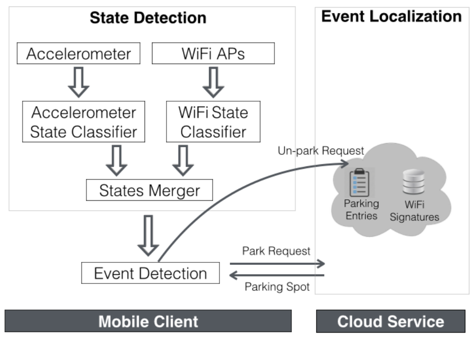


Figure 1: System Architecture of BluePark

5. APPROACH

In this section, we describe the algorithms used in state detection, event detection, and event localization modules in detail.

5.1 State Detection

The state detection module has three parts: Accelerometer State Classifier, WiFi State Classifier, and State Merger. Accelerometer State Classifier uses accelerometer to distinguish $walk$ state from other states, while WiFi State Classifier uses WiFi to identify $drive$ state. The State Merger module combines the output of these two classifier along with previous merged state to find out the current merged state of the user.

5.1.1 *Notation and Definitions*

We define the following time series to denote user's states:

1. $A(1), A(2), \ldots, A(t)$ is a time series resulting from the Accelerometer State Classifier, where $A(t)$ is the accelerometer based user state at time t. $A(t)$ has three possible classes: a_{walk}, $a_{\neg walk}$, and $a_{unknown}$.
2. $W(1), W(2), \ldots, W(t)$ is a time series resulting from the WiFi State Classifier, where $W(t)$ is the WiFi based user state at time t. $W(t)$ has two possible classes: w_{moving} and $w_{\neg moving}$.
3. $C(1), C(2), \ldots, C(t)$ is a time series resulting from the State Merger, where $C(t)$ is the state of the user at time t derived by combining $A(t)$ and $W(t)$. $C(t)$ has four possible classes: $walk$, $drive$, $still$, and $unknown$.

Algorithm 1 AdaptiveDBScan

```
INPUT: Accelerometer feature values, $V_i = \{v_1, v_2, ...v_n\}$
INITIALIZE: $NeighborPts \leftarrow \{\}$, list of potential cluster points
procedure ADAPTIVEDBSCAN($v_i$)
    if $NeighborPts$ is empty then
        append $v_i$ to $NeighborPts$;
        $\alpha \leftarrow v_i$
    else
        if REGIONQUERY($v_i, \alpha$) then
            append $v_i$ to $NeighborPts$;
            $\alpha \leftarrow mean$(NeighborPts);
            if $sizeof(NeighborPts) \geq MinPts$ then
                $A(i) \leftarrow getState(\alpha)$;
                declare $A(i)$;
        else
            if $sizeof(NeighborPts) < MinPts$ then
                $A(i) \leftarrow unknown$;
                declare $A(i)$;
            empty $NeighborPts$;
            append $v_i$ to $NeighborPts$;
            $\alpha \leftarrow v_i$;
procedure REGIONQUERY($v_j$,$\alpha$)
    if $v_j > \tau$ then $\epsilon \leftarrow \theta * \alpha$ ;
    else
        $\epsilon \leftarrow \epsilon_0$;
    if $|v_j - \alpha| < \epsilon$ then return true;
    else
        return false;
```

5.1.2 *Accelerometer State Classifier*

We developed an online unsupervised learning approach to identify $walk$ states of a user from his accelerometer data. We propose *Adaptive DBSCAN*, a modified version of DBSCAN

[5], which can cluster accelerometer data in real time. The Accelerometer State Classifier uses a threshold value on each cluster's centroid α to determine if the accelerometer traces in that cluster correspond to a $walk$ state of the user.

DBSCAN [5] is one of the most popular density based clustering algorithm. It requires only two parameters and can deal with noise: (1) ϵ, a fixed threshold which determines if two points are close enough to be put together in the same cluster i.e. if the distance between them is less than ϵ and (2) $MinPts$, the minimum number of points required to form a cluster. DBSCAN is known to perform not so accurately for data set with large differences in densities. This is because it becomes difficult to choose a $MinPts$ and ϵ combination, which can discover all the clusters. It requires complete data set to perform clustering and hence, is not appropriate for clustering real-time data. For BluePark we need a classifier which can process incoming data in real time. Moreover, the accelerometer data for different user states is observed to have large density differences, which DBSCAN cannot cluster. We modified DBSCAN to serve our needs and proposed an adaptive form of the DBSCAN algorithm.

Input to the *AdaptiveDBSCAN* are features that are computed on accelerometer traces collected over non-overlapping time windows of a fixed duration. It is a sequence of features $\{v_1, v_2, ...v_n\}$, where v_i is the average variance of FFT energy for accelerometer window at time i.

Algorithm 1 describes the steps of how *Adaptive DBSCAN* is used for determining $walk$ or $\neg walk$ states. At any given time instance, the algorithm keeps a list of potential cluster points in $NeighborList$ and a current ϵ value. The *regionQuery* function determines if an incoming data point should be included in the current cluster in $NeighborList$. If the incoming data is within ϵ distance of α, the current cluster centroid, it is included in the cluster. A cluster is discarded as noise if it has less points than $MinPts$ as in traditional DBSCAN and the accelerometer state is declared as $unknown$.

DBSCAN has a fixed ϵ value, which is suitable for non-real time and historical data. In *Adaptive DBCAN*, the ϵ value is adapted based on the incoming data. For each new incoming data point the ϵ value is recomputed as a fraction of that data point. The fractional adapting coefficient is denoted as $0 \leq \theta \leq 1$. This enables *Adaptive DBSCAN* to work on real time incoming data and helps to accommodate large density differences among different user locomotive states. For data points with small values ($\leq \tau$) we use a default threshold ϵ_0. The $getState$ function returns a_{walk} or $a_{\neg walk}$ using a threshold on the value of α.

5.1.3 WiFi State Classifier

The WiFi State Classifier determines if a user is currently moving and changing location continuously. As the user changes location, the set of WiFi APs (APs) visible on the user's phone changes. We define a metric β to measure this change and it is used to determine the WiFi based state $W(t)$ by the WiFi State Classifier.

The WiFi signals are scanned continuously and are grouped into non-overlapping time windows of fixed duration Δ called as *WiFi scan window*. Each *WiFi scan window* consists of multiple WiFi scans. Let $S(t) = \{s_1, s_2, \ldots, s_N\}$, represent a set of unique SSIDs present in *WiFi scan window* starting at time t where s_i is SSID of a unique AP. We measure β between successive *WiFi scan windows*.

The ParkSense [13] paper uses the following Jaccard Index Index formula to compute the rate at which a vehicle is moving away to detect un-parking in an outdoor environment. It compares the number of unique APs visible in successive *WiFi scan windows*. Jaccard Index at time t is defined in Equation 1, where $t - \Delta$ is the preceding WiFi scan window.

$$J(t) = \frac{|S(t-\Delta) \cap S(t)|}{|S(t-\Delta) \cup S(t)|} \tag{1}$$

We observe that the above equation only counts the number of APs and does not consider the frequency of their occurrences. It assumes the signal received by a receiver from an AP will disappear as soon it moves away from that AP. While such an assumption may be valid in outdoors due to absence of reflecting surfaces, for indoors, such a metric can be misleading for computing the rate of movement. As indoor parking areas have multiple reflecting surfaces like walls, ceilings, pillars, etc., a signal from a far away AP can reach the receiver due to multi-path effect. Hence, it will be erroneous if one estimates rate of movement by only comparing the count of APs that are visible from successive positions of a vehicle. Moreover, the same AP may be visible multiple times in the same scanning window due to slow driving speed and restricted driving lanes.

Figure 2 shows histograms of Jaccard Index for *walk* and *drive* states in the indoor parking facility. The index value is spread over the entire range of 0-1 for both *walk* and *drive*, unlike that of in outdoor environment [13]. This leads to the conclusion that in an indoor environment, Jaccard Index is not sufficient to identify states. We propose a modified version of the index, which takes into account frequency of occurrence of APs in a scanning window.

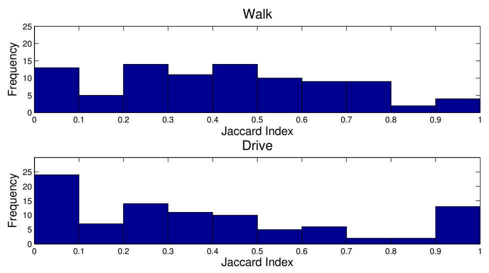


Figure 2: Histogram of Jaccard Index for walking and driving in the indoor parking facility. Using the Jaccard Index it is difficult to distinguish $drive$ state from $walk$ state.

For indoors, we argue, one needs to consider the frequency of occurrence of visible APs. Though multi-path may cause a far away signal to reach the receiver, such a signal traverses over a longer path compared to a signal from the AP, which is in direct line of sight. As a result, APs which are closer to the receiver will occur more frequently during the scanning window than the APs, which are distant. To capture the effect of multi-path, we propose a modified version of the Jaccard Index β, which includes frequency of occurrence of APs in its formulation. We define β for successive *WiFi scan windows* staring at time $t - \Delta$ and t as:

$$\beta(t) = \frac{\sum\limits_{s \in S'} min\{f_t(s), f_{t-\Delta}(s)\}}{\sum\limits_{s \in S'} max\{f_t(s), f_{t-\Delta}(s)\}} \qquad (2)$$

where $S' = S(t-\Delta) \cup S(t)$, and $f_t(s)$ is the number of times AP s occurs in *WiFi scan window*. The above formulation computes the ratio of occurrences of the APs between two successive windows. Lower the ratio, higher the difference in occurrences indicating a possible displacement of the user. On the other hand, higher ratio implies less difference between the number of AP occurrences, indicating that the user is stationary. We use a threshold on $\beta(t)$ values to determine the WiFi based states w_{moving} or $w_{\neg moving}$ which are sent to the States Merger module.

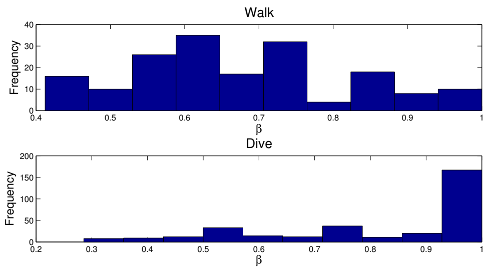


Figure 3: Histogram of β for walking and driving in the indoor parking facility. The left skewed histogram of $drive$ state shows that β is a better metric to distinguish $drive$ from $walk$ state.

5.1.4 States Merger

The Accelerometer State Classifier and WiFi State Classifier are independent and computed asynchronously. States Merger merge the output of these two classifiers to determine the current state of the user. Suppose at any time t, the current Accelerometer state is $A(t)$ and WiFi state is $W(t)$. The State Merger determines the current state of the user $C(t)$ by combining $A(t)$, $W(t)$, and previous state $C(t-1)$. There are four possible classes for $C(t)$: $walk, drive, still$, and $unknown$.

As the accelerometer window and *WiFi scan window* can have different duration, $A(t)$ and $W(t)$ states can span across different time intervals. The state merger uses a wait-and-proceed approach to find time overlap between incoming $A(t)$ and $W(t)$ states and determines $C(t)$ for the overlapped interval. The time overlap between two corresponding states, where each state has a start and an end time, is the time interval between the most recent of the two start times and the earliest of the two end times.

The States Merger combines $A(t)$, $W(t)$, and $C(t-1)$ using the rules given in Table 1, where $*$ denotes any possible state. The rule table helps to distinguish the $walk$ and $drive$ states of the user, which is the primary goal of the state detectionmodule. As mentioned in the table, if WiFi State Classifier detects the user to be in a $moving$ state but the Accelerometer State Classifier does not detect a $walk$ state then we infer the user's current state to be $drive$. Similarly, we conclude a *still* state if the user is neither $moving$ and nor $walk$. A $walk$ state is detected whenever the Accelerometer State detects a $walk$ irrespective of all other sates whereas the previous state $C(t-1)$ is used when the Accelerometer based state is $unknown$. The rule table approach is a lightweight state detection mechanism compared to more sophisticated approaches like HMM and helps in minimizing latency which, as discussed earlier is critical for accuracy in detecting state transitions and event localization.

$A(t)$	$C(t-1)$	$W(t)$	$C(t)$
a_{walk}	$*$	$*$	$walk$
$a_{\neg walk}$	$*$	$w_{\neg moving}$	$still$
		w_{moving}	$drive$
$a_{unknown}$	$unknown$	$*$	$unknown$
	$walk$	$w_{\neg moving}$	$unknown$
		w_{moving}	$walk$
	$drive$	$w_{\neg moving}$	$unknown$
		w_{moving}	$drive$
	$still$	$w_{\neg moving}$	$still$
		w_{moving}	$unknown$

Table 1: Rule for combining Accelerometer and WiFi based states to obtain the final state.

For Accelerometer State Classifier, time taken to detect a state for the first time, is a function of $MinPts$ and window size of Accelerometer State Classifier. Whereas, for WiFi State Classifier, it is a function of the *WiFi Scan Window* duration. The parameter values used in our experiments are given in Table 2. Time taken to detect state for the first time is the maximum of the time taken by the two classifiers, which may not be periodic.

5.2 Event Detection

The goal of the event detection module is to detect state transitions, which corresponds to parking and un-parking events. Algorithm 2 describes the steps for detecting parking and un-parking events. For each state transition between *walk* and *drive*, the algorithm checks if the user states, before and after the transition remains unchanged for more than ν amount of time to mark it as a legitimate transition. If the state changes within ν amount of time, we discard the corresponding state transition. *still* and *unknown* states are not considered in this algorithm as we only look for transitions between walk and *drive* states.

The mobile client reports the detected events to the cloud service by sending two types of requests. (1) Park Request that reports a parking event by sending WiFi signature seen at the time of detected parking event. The cloud service processes the WiFi signature and returns the parking spot L_{id} to the mobile client. It marks L_{id} as occupied. (2) Un-park

Algorithm 2 State transition detection algorithm

```
INPUT: Current state $C(k) = \{type, duration, endTime\}$
INITIALIZE: $sL \leftarrow \{\}$, a list of merged state
procedure DETECTSTATETRANSITION($C(k)$)
    $matchFound \leftarrow$ false
    if $sL$ is not empty then // Check history
        for $L_i$ in $sL$ do
            if $L_i.type = C(k).type$ then
                remove $L_i$ from $sL$
                $C(k).dur \leftarrow C(k).dur + L_i.dur$
                append $C(k)$ to $sL$
                $matchFound$ = true
    if not $matchFound$ then
        append $C(k)$ to $sL$
    if $sizeof(sL) = 2$ then
        CHECKTRANSITION(sL(0),sL(1));
        if $(sL(1).dur \geq \nu)$ then
            // Keep only the last stable state
            remove $sL(0)$ from $sL$;
procedure CHECKTRANSITION(prevState,curState)
    if $(prevState.dur \geq \nu)$ and $(curState.dur \geq \nu)$ then
        declare $prevState.type \rightarrow curState.type$
        transition at time $prevState.endTime$
```

Request that reports an un-parking event by sending the L_{id}. The cloud service marks L_{id} as empty.

5.3 Event Localization

Event Localization module is responsible for determining parking spot L_{id} for the WiFi signature sent by the mobile client in a Park Request. We used a WiFi based indoor localization system which was hosted on the cloud service. It is based on the well known Bayesian Inference technique called Horus [18]. While indoor localization is widely studied using various methods, we choose a WiFi based localization system due to its ubiquity and low cost. Moreover, BluePark already uses WiFi for locomotive state detection, this choice allows us to use the same for localization without any additional sensor, thereby eliminating other overheads.

As a prerequisite to apply Horus, we collected WiFi fingerprints from the parking area in the region highlighted in Figure 6a. Using the Horus method, these fingerprints were later matched offline with the WiFi data collected at the time of parking and un-parking events for event localization. The pillar numbers adjacent to parking spaces served as parking spots. We observed that WiFi coverage in the parking are was not uniform due to decentralized placement of APs by the retailers on the floors above. We tuned our localization system to be robust enough to handle this non-uniformity.

	Parameters	Classifier Threshold
ASC	window size = 6s $MinPts = 2$ $\theta = 0.35$	$\alpha < 2$, then $A_{state} = a_{\neg walk}$ $2 \leq \alpha \leq 20$, then $A_{state} = a_{walk}$ $\alpha > 19$, then $A_{state} = a_{unknown}$
WSC	sampling interval = 5s window size = 20s	$\beta(t) \geq 0.34$, then $w_{state} = w_{moving}$ $\beta(t) < 0.34$, then $w_{state} = w_{\neg moving}$
ED	$\nu = 10$	NA

Table 2: List of parameters used in Accelerometer State Classifier (ASC), WiFi State Classifier (WSC), and Event Detection(ED) and their values used for evaluation of BluePark

5.4 Tuning of Parameter

In this section, we discuss about how we determined values for parameters used in our Accelerometer State Classifier and WiFi State Classifier. We collected accelerometer and WiFi traces for $walk$, $drive$, and $\neg moving$ states to determine the values.

We assume the phone is stationary relative to the motion of the vehicle, and therefore, we do not consider on-body position of the phone while collecting $drive$ data. We collected 43 minutes of drive data inside the parking area. For $walk$ data, we collected data for three common on-body phone positions, trouser pocket, hand, and backpack. We collected continuous walking data for 43 minutes, for each on-body phone positions.

5.4.1 *Accelerometer State Classifier Parameters*

For tuning the parameters of Accelerometer State Classifier, we conducted experiments with different value of θ in the range 0.2 to 0.5, and loosely found the optimal value as 0.35. For $MinPts$ value, we experimented with values in the range of 2 to 5. Higher value of $MinPts$ have higher chance of discarding a collection as noise. For our BluePark system, we set $MinPts = 2$.

Using these setting, we computed the α values of the collected accelerometer data. Figure 4 shows a plot of cumulative distribution of α for $walk$ and $drive$ states. In the plot, we observe that $walk$ and $\neg walk$ states ($driving$ and $loitering$) are clearly distinguishable. However, as described earlier $driving$ and $loitering$ cannot be distinguished using the α value. In the plot, we observe that $drive$ has an α lower than 2 with a probability of 0.99, whereas for $walk$ with different phone positions α has value between 2 and 20. The Accelerometer State Classifier uses these values to determine the threshold values for α for classifying $walk$ and $\neg walk$ states.

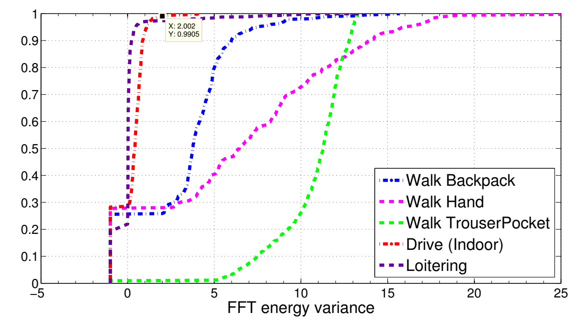


Figure 4: Cumulative distribution of α for walk and drive states. $walk$ state at all phone positions are distinguishable whereas $drive$ and $loitering$ states are not distinguishable from each other.

5.4.2 *WiFi State Classifier Parameters*

We use a *WiFi scan window* of 20 seconds in which WiFi scanning is done at 5 seconds interval. Figure 5 shows the cumulative distribution of β for $walk$, $drive$, and $\neg moving$

states. In the plot, we observe that the non-stationary states, $walk$ and $drive$, are indistinguishable from each other using the β values but can be distinguished from $\neg moving$ state where $\beta \leq 0.34$ with probability of 0.9. We use this as the threshold to detect $\neg moving$ state in the WiFi state classifier. Table 2 consolidates the parameters used in BluePark.

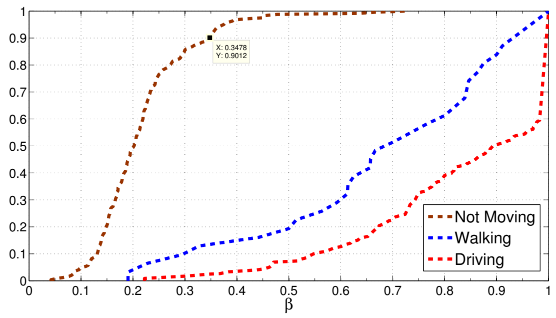


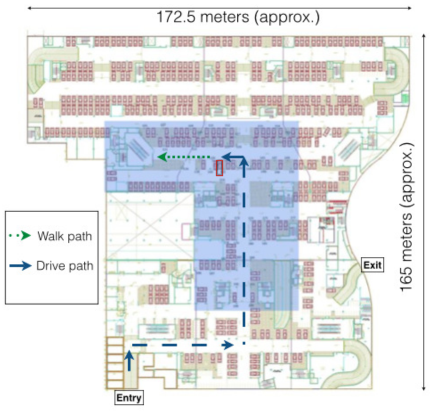
Figure 5: Cumulative distribution of β for $walk$, $drive$, and $\neg moving$ states

6. EVALUATION

In this section, we evaluate BluePark in terms of accuracy and latency. We compare BluePark's state detection with Google's Activity Recognition API (GARA) [4], which is the most popular activity detection service on a smartphone. It is a part of the Google Play Service and is capable of detecting STILL, ON_FOOT, IN_VEHICLE, TILTING, ON_BICYCLE, and UNKNOWN states. Application developer has to specify an update interval to run the activity detection. We used version 6.1 of the Activity Recognition API in Google Play Services. To lower event detection latency, we require update interval of 3 seconds for version 6.1. Unless otherwise specified, in our evaluation the update interval is set to 3 seconds.

6.1 Setup for Experiments

We conducted our experiments in an underground parking area of a popular shopping mall in New Delhi. Figure 6a shows a map of the parking area. The highlighted region in the figure represents the area, which was WiFi fingerprinted. Figure 6b shows a representative picture of the underground parking area, where the experiments were conducted. Each pillar in the parking area is assigned a pillar number. These numbers are used in BluePark to identify a location, where a vehicle is parked. We collected WiFi fingerprints for 56 parking spots. A total of 53 unique APS are visible from these parking spots.The total area of the parking area is 172.5 meters$\times$165 meters. The standard size of a vehicle parking spot is 2.5 meters$\times$5 meters. For localization accuracy, distance between estimated and ground truth location is computed using this standard size and map of the basement parking area. Parkings at non-fingerprinted areas are not considered for measuring the accuracy.



(a)

(b)

Figure 6: (a) Map of the parking area where BluePark system was deployed. The WiFi fingerprinted area is highlighted in blue. The arrows indicate an example route of a vehicle while parking. (b) Representative picture of the underground parking area.

Five accelerometer sampling rates 5Hz, 10Hz, 15Hz, 50Hz, and 99Hz were used. We tested BluePark on three on-body phone positions, in hand, in trouser pocket, and in backpack. These are the commonly used phone positions. GARA has its internal way of deciding sampling frequency.

6.2 Accuracy of State Detection

We compare the performance of our state detection module with that of GARA. We consider on VEHICLE ($drive$) and ON_FOOT ($walk$) states for comparison. As we only look for transitions between walk and drive states, we do not evaluate the remaining states. Table 3 shows accuracy in percentage of the two approaches in the three on-body phone positions for $drive$ state. A similar comparison of $walk$ state detection is shown in Table 4.

GARA	**BluePark**				
	5Hz	**10Hz**	**15Hz**	**50Hz**	**99Hz**
72.10%	93.38%	93.90%	95.27%	95.08%	95.98%

Table 3: Accuracy of $drive$ detection using BluePark and Google Activity Recognition API (GARA). BluePark detects $drive$ state better than GARA.

From Table 3, we observe BluePark performs better than GARA for $drive$ detection by more than 20% at all sampling rates. It validates efficacy of our modified Jaccard Index formulation. During $drive$ state, the phone is moving relative to the motion of the car and is at a fix position relative to the body of the user. Therefore, we do not consider the on-body position of the phone while driving. From Table 4, we ob-

Phone position while walking	**GARA**	**BluePark**					
		5Hz	**10Hz**	**15Hz**	**50Hz**	**99Hz**	**Average**
Trouser Pocket	100.00%	98.60%	98.14%	98.78%	98.56%	98.55%	98.52%
Hand	96.45%	98.05%	98.72%	98.93%	98.95%	98.74%	98.68%
Backpack	99.57%	85.00%	96.75%	96.86%	98.21%	98.21%	95.07%
Average	98.65%	94.41%	98.10%	98.48%	98.65%	98.56%	

Table 4: Accuracy of $walk$ using BluePark and Google Activity Recognition API (GARA)

Event	**Phone position while walking**	**GARA**		**BluePark**									
				5Hz		**10Hz**		**15Hz**		**50Hz**		**99Hz**	
		P	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**	**R**	**P**	**R**
Parking	Trouser Pocket	100	100	100	100	100	100	100	100	100	100	100	100
	Hand	100	100	100	100	**80**	100	100	100	100	100	100	100
	BackPack	100	100	**80**	100	100	100	100	100	100	100	100	100
Unparking	Trouser Pocket	100	100	**83.33**	100	100	100	100	100	100	100	100	100
	Hand	100	100	100	100	**80**	100	100	100	100	100	100	100
	BackPack	100	100	**80**	100	100	100	100	100	100	100	100	100

Table 5: Precision (P) and Recall (R) of Activity Detection using BluePark and Google Activity Recognition API (GARA)

serve that BluePark detects $walk$ state with high accuracy event at a low sampling rate of 10Hz. In trouser pocket and backpack positions, GARA performs better than BluePark, whereas at hand phone position BluePark has better accuracy. Detection accuracy of $walk$ state by BluePark, at 5Hz, reduces significantly. The overall performance of $walk$ detection of the two approaches are not significantly different when the sampling rate is higher than or equal to 10Hz.

For detecting parking and un-parking, we need a system, which can correctly detect both $walk$ and $drive$ states. The better accuracy of BluePark in $drive$ state detection results in lower event detection latency than that of GARA. This is discussed in Subsection 6.4.

6.3 Accuracy of Event Detection

We measure accuracy of event detection in terms of precision and recall. Precision measures the probability that a detected event is truly an event. It is computed as $tp/(tp + fp)$, where tp is true positive and fp is false positive. Recall measures the probability that a true event is actually detected by the system. It is given by $tp/(tp + fn)$ where fn is false negative. False positive is falsely detecting an event will effect the precision value, while false negative is failing to detect a true event will effect the recall value.

Table 5 shows a comparison of precision and recall of parking and un-parking event detectionfor BluePark and GARA. The precision and recall of GARA is 100% for all phone positions for both parking and un-parking events. For BluePark, there are cases of false positives at 5Hz and 10Hz, resulting in less than 100% precision and recall. At sampling rate of 15Hz or higher, the precision and recall are all 100%. Therefore, we will use 15Hz and higher sampling rate for further evaluations.

In our experiments, precision and recall imply detection of a parking and un-parking but are not indicative of how responsive the system is. Responsiveness of the system is measured by the detection latency. For the automatic event localization and validity of the parking information, latency of detecting parking or un-parking event is important. Next, we discuss the latency of event detection.

6.4 Latency in Event Detection

Latency in event detection is computed as time difference between actual time and detected time of an event. The actual time of events were noted down manually at the time of data collection. Latency in event detectionis computed for both BluePark and GARA.

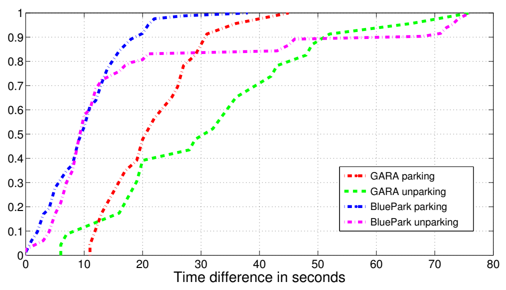


Figure 7: Cumulative distribution of time difference between ground truth and detected time for parking and un-parking activity. BluePark detects parking within 20 seconds with probability of 0.9, while GARA takes around 30 seconds for the same probability.

Cumulative distribution function of latency in event detection for BluePark in comparison to GARA is shown is Figure 7. BluePark detects parking transition within 20 seconds delay with probability of 0.89, whereas GARA takes around 30 seconds for the same. The higher event detection latency of GARA is due to its low detection accuracy of $drive$ state as seen in Table 3. WiFi signature seen at the time of parking event is used to determine location of the parking spot. A high detection latency will result in an incorrect WiFi signature, thereby introducing an error in localization. Therefore, latency in event detection directly affects the event location accuracy.

6.5 Accuracy of Event Localization

Event localization accuracy is important for identifying where the vehicle was parked. Apart from detection latency,

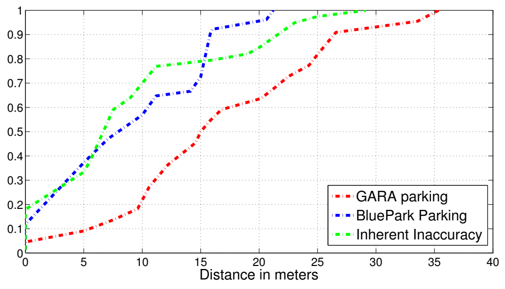


Figure 8: Cumulative distribution of location error in terms of meters. BluePark closely follows the inherent latency whereas GARA is less accurate due to higher detection latency.

which we discussed in the previous section, location accuracy depends on localization approach used in the system. The inherent inaccuracy of the used localization approach limits overall accuracy event localization. Lesser the inherent inaccuracy, better the overall event localization accuracy. With a probability of more than 0.9, the location inaccuracy of BluePark is less than 15.81 meters, which is approximately 6 parking spots away from where the car was actually parked. For the same probability, the location inaccuracy of GARA is 26.7 meters, is approximately 10 parking spots away.Figure 8 shows comparison of cumulative distribution of distance error in meters. For measuring the location accuracy of GARA, we used our modular design by replacing the state detection module with GARA and the rest of the system remains the same. The set of locations considered in computing BluePark and GARA location accuracy is a subset of the locations considered for inherent accuracy calculation. The median location error of BluePark is 10 meters and that of GARA is 15.4 meters.

7. CONCLUSION

In this paper, we presented BluePark, a collaborative sensing mechanism, using smartphone sensors, that can detect and localize parking and un-parking of vehicles in indoor parking lots in real time, without any user input. We proposed a combination of accelerometer and WiFi sensors to detect $drive$ and $walk$ states. It is a modular system and its state state detection and localization modules can be replaced by algorithm of user's choice. We presented our own algorithm for state detection and showed that our algorithm gives better accuracy, even at low sampling rate, as compared to accelerometer based Google Activity Recognition API (GARA). As our state detection algorithm uses unsupervised training for detection, it does not require per-user calibration.

We proposed an event detection algorithm that is based on state transitions. In our experiments, we found that the overall precision and recall of GARA is better than that of BluePark. However, BluePark's state detection outperforms GARA in terms of detection latency. As accuracy of localization is directly proportional to latency, accuracy of BluePark using our state detection algorithm is better than that using GARA. The experimental results probe that BluePark is robust to common on-body phone positions.

8. REFERENCES

[1] Google OpenSpot. **http://techcrunch.com/2010/07/09/google-parking-open-spot/** Accessed: 2015-02-12.

[2] http://outdoormediaplan.com/malls-media-kit.php. **http://outdoormediaplan.com/malls-media-kit.php**Accessed:2015-08-3.

[3] Parkitekt Bangalore. **https://goo.gl/oitu6f** Accessed : 2015-02-12.

[4] Recognizing the User's Current Activity. **http://goo.gl/5fm314** Accessed : 2014-08-16.

[5] Ester, M., peter Kriegel, H., S, J., and Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. AAAI Press (1996), 226–231.

[6] Hemminki, S., Nurmi, P., and Tarkoma, S. Accelerometer-based transportation mode detection on smartphones. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, SenSys '13, ACM (New York, NY, USA, 2013), 13:1–13:14.

[7] Kokolaki, E., Kollias, G., Papadaki, M., Karaliopoulos, M., and Stavrakakis, I. Opportunistically-assisted parking search: a story of free riders, selfish liars and bona fide mules. In *Wireless On-demand Network Systems and Services (WONS), 2013 10th Annual Conference on*, IEEE (2013), 17–24.

[8] Lan, K.-C., and Shih, W.-Y. An intelligent driver location system for smart parking. *Expert Systems with Applications 41*, 5 (Apr. 2014), 2443–2456.

[9] Lee, S., Yoon, D., and Ghosh, A. Intelligent parking lot application using wireless sensor networks. In *Collaborative Technologies and Systems, 2008. CTS 2008. International Symposium on* (May 2008), 48–57.

[10] Mathur, S., Kaul, S., Gruteser, M., and Trappe, W. Parknet: A mobile sensor network for harvesting real time vehicular parking information. In *Proceedings of the 2009 MobiHoc S3 Workshop on MobiHoc S3*, MobiHoc S3 '09, ACM (New York, NY, USA, 2009), 25–28.

[11] MONKEYPARKING : on-street parking on demand. **http://monkeyparking.strikingly.com**.

[12] Nandugudi, A., Ki, T., Nuessle, C., and Challen, G. Pocketparker: Pocketsourcing parking lot availability. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '14 Adjunct, ACM (New York, NY, USA, 2014), 963–973.

[13] Nawaz, S., Efstratiou, C., and Mascolo, C. Parksense: A smartphone based sensing system for on-street parking. In *Proceedings of the 19th Annual International Conference on Mobile Computing &#38; Networking*, MobiCom '13, ACM (New York, NY, USA, 2013), 75–86.

[14] Reddy, S., Mun, M., Burke, J., Estrin, D., Hansen, M., and Srivastava, M. Using mobile phones to determine transportation modes. *ACM Trans. Sen. Netw. 6*, 2 (Mar. 2010), 13:1–13:27.

[15] Sharma, P., Chakraborty, D., Banerjee, N., Banerjee, D., Agarwal, S., and Mittal, S. Karma: Improving wifi-based indoor localization with dynamic causality calibration. In *Sensing, Communication, and Networking (SECON), 2014 Eleventh Annual IEEE International Conference on* (June 2014), 90–98.

[16] Tang, V., Zheng, Y., and Cao, J. An intelligent car park management system based on wireless sensor networks. In *Pervasive Computing and Applications, 2006 1st International Symposium on* (Aug 2006), 65–70.

[17] Wang, S., Chen, C., and Ma, J. Accelerometer based transportation mode recognition on mobile phones. In *Proceedings of the 2010 Asia-Pacific Conference on Wearable Computing Systems*, APWCS '10, IEEE Computer Society (Washington, DC, USA, 2010), 44–46.

[18] Youssef, M., and Agrawala, A. The horus wlan location determination system. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, MobiSys '05, ACM (New York, NY, USA, 2005), 205–218.

[19] Zheng, Y., Chen, Y., Li, Q., Xie, X., and Ma, W.-Y. Understanding transportation modes based on gps data for web applications. *ACM Trans. Web 4*, 1 (Jan. 2010), 1:1–1:36.