Characterizing Road Segments Using Compass Sensors to Predict Approaching Bus Stops

Danila Chenchik†, Jia Chen†, Stephen Yan†, Shahriar Nirjon
Undergraduate Mobile Computing Systems Research Group
Department of Computer Science
University of North Carolina at Chapel Hill
Email: {chenchik, jiac, sjyan, nirjon}@cs.unc.edu

***Abstract*—We devise an inexpensive and intuitive system for bus route navigation for locales where public transportation may serve as a prevalent mode of commute but where technologies that make arrival predictions through tracking vehicles in transit through GPS or other means do not exist. These systems typically require real-time monitoring of traffic variations. We provide a personalized approach where in a world of pervasive smart phone use, users may take advantage of sensor data to learn and personalize their bus routes, and alert them on time when a bus stop is approaching. We accomplish this through the development and implementation of two algorithms: 1) turn detection using on-board compass sensor of a smart phone, and 2) characterizing road segments in terms of turns and thereby predicting approaching bus stops. We conduct field experiments on a route with four selected bus stops in the town of Chapel Hill. Results show that the accuracy of turn detection and detection of approaching bus stops are 95.7% and 83%, respectively.**

I. Introduction

The built-in sensors on smart phones has encouraged the development of various convenient applications that users can interact with to aid them in different aspects of their lives. One such usage for smart phone sensors is route mapping and tracking. Map and GPS are some of the most vital features on a smart phone, and many public transportation systems in different cities have published their own bus route and alert systems. One helpful feature that is still lacking on many of these applications is the ability to recognize a user's routine bus route and alert the user when his/her usual bus stop is approaching. Often on bus rides, passengers might not be totally attentive on the road and/or bus stop. Missing a bus stop because of a moment of inattentiveness can be a frustrating thing. Realizing this issue, we have worked to develop a bus stop alerting system on Android that includes this feature for public bus routes.

Our system contains a set of built-in, pre-identified bus routes. It operates by learning a user's daily bus route, paying attention to details such as where the user gets on and get off regularly. This information, along with the compass sensor data, is collected and analyzed in order to characterize and recognize approaching bus stops. The goal of this system is to send an alert to the user when his or her usual bus stop is approaching.

†The first three authors are equal contributors to this paper.

Our system delves into several areas of research, one being accelerometer-based activity recognition. Many works have been published on context and activity detection using sensor readings; our work is based on such context recognition. Another aspect of this bus stop alert system deals with bus stop arrival information. Various applications exist for bus arrival prediction using a mixture of GPS and pre-determined bus arrival information. Such systems have some drawbacks upon which we wish to improve. Human attention is an area of research we briefly examine since our system aims to alert users of their particular bus stops, an application of which would be most pertinent when someone is not paying direct attention to the current route conditions.

To create a functional and efficient system, our major milestone is to be able to characterize and learn road segments on a specific bus route by using compass sensor values collected by a smart phone. This process is broken down into two steps in order to accurately classify a route – 1) accurately detecting turns, and 2) using turn information to represent a road segment between consecutive bus stops. Such characterizations based upon raw compass data is challenging due to inaccuracies and noise in the data. Normalization is required to smooth the raw data before classification is performed using different features we derive from the data. A benefit of this solution is that it avoids the use of GPS, which would be a major source of energy consumption for such a system that is expected to run for an extended period of time.

More specifically, this process is performed via a sequence of four steps: 1) data collection, 2) turn detection, 3) encoding road segments, and 4) classification. Data is collected during road testing using smart phones, and then used as inputs to the turn detection algorithm. This process identifies the performance of a turn while performing normalization on the data. Such information are encoded into route-specific features to be used in a simple 1-nearest neighbor classifier.

Our system has been tested against a route system with several bus stops defined on the route. Each test run includes both sensor values and real route conditions for testing purposes. Upon evaluation, our turn detection design results in 95.7% accuracy, while the overall bus stop detection performs at 83% accuracy.

This paper makes the follow contributions:

- We have collected a wide variety of sensor data on mobile devices on multiple routes in the town of Chapel Hill. We share our code and data with the community [1].
- We devise two algorithms – 1) to detect turning vehicles using noisy compass sensor data, and 2) encoding road segments using turn information to characterize road segments.
- We implement an Android application that reads in compass data and implements the algorithms. Our evaluation shows that the overall accuracy of the system in bus stop prediction is 83%.

II. Motivation and Problem

Public transits serve a vital transportation role in cities where they are readily available. Many people rely on buses and trains as their everyday mode of transportation. One trouble people might face riding the bus daily is missing their bus stop due to inattentiveness during the bus ride. Anyone would find this to be frustrating along with the precious time wasted during transit. Because of this, we want to discover a system to detect a user's routine bus stop so the user can be alerted in a timely manner at his or her usual stop.

Some similar systems often utilize a variety of built-in sensors on smart phones. Some of these most effective applications, however, often make use of high energy consuming sensors such as GPS. This would not be ideal for an application intended for regular long-period use on a smart phone. Another option would be to use the built-in compass and IMU on smart phones. This alternative would provide us a less expensive way to track a user's bus activity. While there are robust solutions that predict bus arrival time, no existing solution exists to connect our problem with this approach. We thus want to introduce a bus stop alert system that can provide route tracking functionality dependent only on a smart phone's compass and IMU.

In order to accomplish this, the main challenge we've identified is fingerprinting the route and specific bus stops. Route and bus stop detection would be a simple problem with the use of GPS. However, to rely solely on accelerometer and compass information provides some difficulty. Given a set of accelerometer and compass values collected over a period of time, we need to be able to characterize and classify each section of a bus stop with a section defined as the path from one bus stop to the next.

III. System Design

In this section, first, we introduce the system architecture, and then detail its specific components.

A. System Overview

The current system uses a smart phone's built-in sensors to read in accelerometer values and compass data as the bus is in motion. The data is then extracted and processed so that it can be used as an input to a turn detection module, which we rely on as a basis for path and bus stop recognition. For this step, a module is used which is able to detect left and right turns. The detailed road information is further encoded as bus route specific data and used as training data for classifying individual stops. Figure 1 shows the overall system.

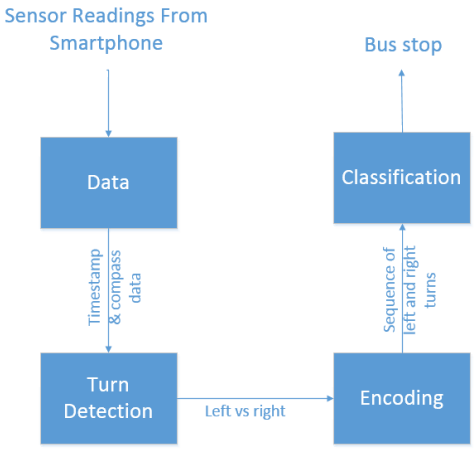

Fig. 1. The four building blocks – data collection, turn detection, segment encoding, and classification modules, and their relationships are shown.

B. Reading Sensor Data

In order to properly classify bus movements, sensor data collection is performed by following a bus route on a car. The route is shown in Figure 2. Instead of collecting data on a bus, we used a car to expedite the data collection process and to have more control over it. We made sure that the speed, route taken, and stops are as close to an actual bus ride as possible.

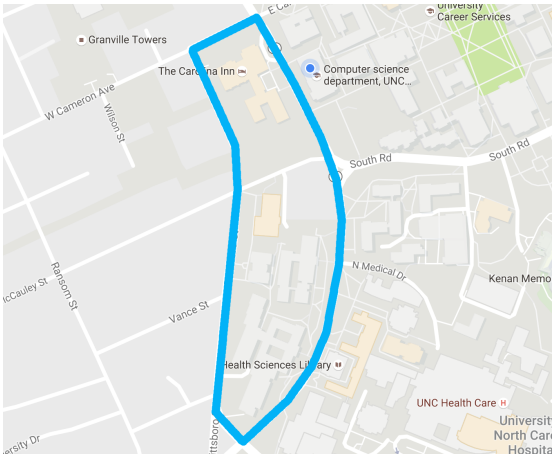

Fig. 2. Route taken during our initial data collection phase.

Multiple rounds of driving were completed and sensor data were collected throughout the entirety of the ride. All available sensors were used and tracked, but the time stamp along with accelerometer, magnetometer, gyroscope, and compass data were the most relevant parameters related to our problem. For testing and comparison purposes, the GPS information was also collected along with the bus stop markers, which we took note of when we reached a bus stop.

C. Detecting Turns

We design an algorithm for detecting turns which is reliant upon a smart phone's magnetometer and accelerometer sensors. These two sensors are used in conjunction to produce 'compass' readings that are represented in degrees ranging from 0 to 360.

We employ a simple interval and threshold based strategy to understand when a vehicle makes a turn. The interval decides the length of an array, which is used to keep the last n degree values. In our experiments, we found that $n = 50$ is ideal for turn detection. Our sensor readings come in at a rate of about 100 ms each, i.e., 10 readings/second, so a threshold of 50 means that our algorithm looks for degree values from the last 5 seconds. We find that a liberal view on how many degrees define a turn is the best approach. We settle on the number of 30 degrees after trying several values.

The proposed algorithm executes every time a new sensor reading comes in and decides whether or not a vehicle is making a turn at every execution. First, we shift all elements in our array of compass values by 1 unit to the left, and then fill the last space in the array with the newly read degree value. If the last element and the first element in the array have a difference of greater than 30, our predetermined threshold, it is considered a turn. Specifically, if the first element in the array subtracted from the last element in the array has a difference of greater than 30, the current point in time is considered a right turn. If the last element in the array subtracted from the first element in the array has a difference of greater than 30, the algorithm determines that the user is making a left turn. If the algorithm detects that a turn is being made, it classifies it as 'LEFT' or 'RIGHT'. If the algorithm detects that a turn is not being made, a special value of '-9999' is put in the turn column.

An edge case we have to consider is the shifting of compass readings between the 0 and 360 degree mark. In order to handle this edge case, we simply set the first element in the degree array with the newly read degree value and repeat the turn algorithm with new limits on what is considered the interval (or the end of the array). We then increment the new interval at every new degree reading until it reaches our original interval, 50. A more detailed view of the algorithm is provided by Algorithm 1.

The most significant issue we have encountered when implementing this algorithm has been the instability of the magnetometer sensor at crucial times. This caused the compass readings to fluctuate significantly, resulting in the detection of false turns. Figure 3 shows a map of a route we took with a car while collecting sensor data. The plot in Figure 4 shows the degree values that were converted from the accelerometer and magnetometer sensor readings.

As seen in Figure 4, degree fluctuations occur at periodic time intervals when the car was not turning. An example can be seen between times 160,000 and 320,000. Fluctuations between this time interval results in two false turn detections. In order to combat these fluctuations, we implemented an averaging technique that was applied to each of the incoming new degree values. To implement this, the array holding previous degree values is shifted to the left one element, then the sum of every element in the array plus the new degree value is added to the total. Then this running total is divided by the length of the array of the interval for previous degree values collected. This results in smoothing of the overall data trend and minimizes fluctuations that war responsible for false turns detected by the algorithm. The result is shown in Figure 5.

Algorithm 1 Detect Left and Right Turns

```
arr ← SHIFT_LEFT(arr)
turn ← 30
arr[interval − 1] ← newDegree
from ← arr[0]
to ← arr[interval − 1]
beforeTo ← arr[interval − 2]
if abs(beforeTo − to) > 180 then
    arr[0] ←to
    interval ← 0
else
    if  to − from > 0 then
        if  to − from > 30  then
            // Right Turn Detected.
        else
            // Going Straight.
        end if
    else
        if from − to > 30 then
            // Left Turn Detected.
        else
            // Going Straight.
        end if
    end if
end if
if interval < 50 then
    interval++
end if
```

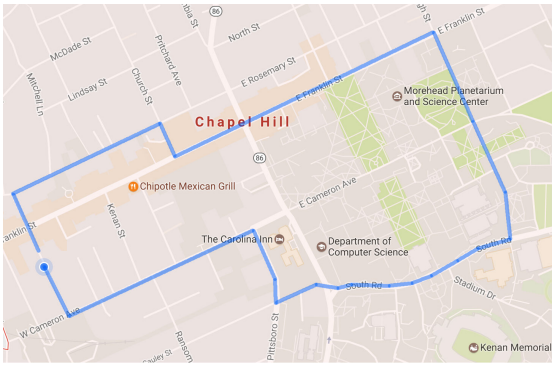

Fig. 3. Another route that produces false turn detections due to fluctuations in magnetometer readings, unless appropriate measures are taken.

$$avgValue = \frac{value + \sum_{i=0}^{n-2} arr[i]}{n} \quad (1)$$

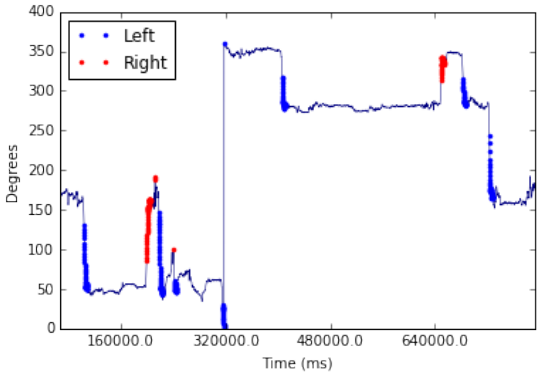

Fig. 4. Degree values over time without smoothing results in an accumulation of false turn detections.

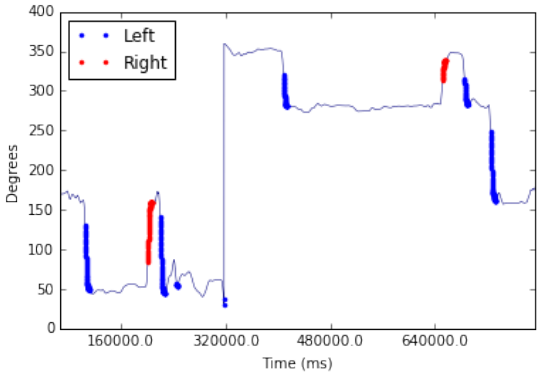

Fig. 5. False turn detections eliminated after smoothing.

Algorithm 2 Encode Segment

```
// global_len ≡ length of segment string
// global_r, global_l ≡ number right/left turns
//  global_n ≡ number of trials for given segment
```

$global\_len, global\_r, global\_l, global\_n = 0$

for each pass of segment **do**
  Read count from
  $global\_len \leftarrow global\_len + len$
  $global\_r \leftarrow global\_r + r$
  $global\_l \leftarrow global\_l + l$
  $global\_n$++
end for

$global\_len \leftarrow global\_len/global\_n$
$global\_l \leftarrow global\_l/global\_n$
$global\_r \leftarrow global\_r/global\_n$

return $(global\_len, global\_l, global\_r)$

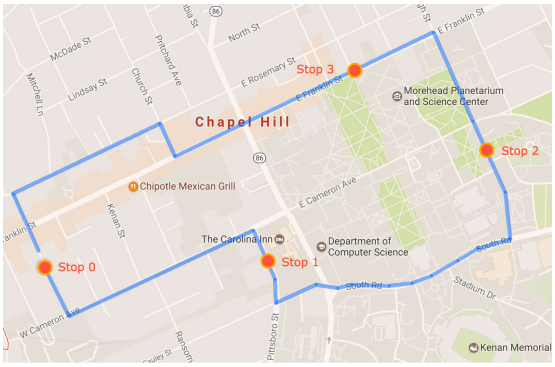

Fig. 6. An example route to illustrate segment encoding of four road segments separated by four bus stops on a loop.

D. Encoding Road Segments

The goal of this step is to encode a road segment between two consecutive bus stops into a suitable format so that 1) the encoded representation is consistent when we drive a vehicle along the same road segment at different times, and 2) it is capable of distinguishing different road segments.

We employ a simple encoding scheme that is computationally efficient and meets the above two criteria. Each road segment is represented with a triplet of the form (τ, λ, ρ), where each element represents the total length of the encoded string (τ), number of left turns (λ), and the number of right turns (ρ), respectively. The algorithm is presented in Algorithm 2.

To illustrate the encoding scheme, we show an example of a bus loop that contains four road segments that are connected at four bus stops. The loop is shown in Figure 6. As a first step of encoding, we convert all the raw compass readings into turns using the turn detection algorithm described in the previous section. Then a long series of 'LEFT' and 'RIGHT' values are expressed in a compressed form by bundling together consecutive 'LEFT' and 'RIGHT' as 'L' and 'R' respectively. Table I shows this form of encoding for each of the four segments of the road for three different passes through the same segment. We observe that the encoding of the same segment for different passes are highly consistent with each other, and encodings of different segments are clearly distinct.

TABLE I
ENCODING SEGMENTS AS STRINGS

	Pass 1	**Pass 2**	**Pass 3**
Segment 0	LRL	LR	LR
Segment 1	RLRL	RLRRLRL	RLRRLRLL
Segment 2	L	L	L
Segment 3	RLL	RLL	RLL

While the above string-based encoding meets our criteria, the encoding length is not uniform (e.g., segment 2 has a

longer length than others) and hence, computing the similarity between two different encodings is non-trivial. Hence, we convert each of these strings into a triplet (τ, λ, ρ) and use an average of multiple passes to empirically determine the mean values of τ, λ, and ρ for each road segment. This process is illustrated in Table II, where the last column represents the final encoding of each segment.

TABLE II
ENCODING SEGMENTS AS TRIPLETS

	Pass 1	**Pass 2**	**Pass 3**	**Final Encoding**
Segment 0	(3,2,1)	(2,1,1)	(2,1,1)	(2.33, 1.33, 1.00)
Segment 1	(4,2,2)	(7,3,4)	(8,4,4)	(6.33, 3.00, 3.33)
Segment 2	(1,1,0)	(1,1,0)	(1,1,0)	(1.00, 1.00, 0.00)
Segment 3	(3,2,1)	(3,2,1)	(3,2,1)	(3.00, 2.00, 1.00)

Algorithm 3 Classify Segment

S_i: encoding of the i^{th} segment.
GET a sequence of turns
count Num_{Length}, Num_{Left}, Num_{Right}
Form encoding E(Num_{Length}, Num_{Left}, Num_{Right})
$c = \underset{i}{\operatorname{argmin}} \|S_i - E\|$

E. Segment Classification

The final step of our system is to classify an encoded segment as one of the n segments of a route. For this, we use a simple 1-nearest neighbor algorithm (Algorithm 3) to classify an encoded segment $E(\tau, \lambda, \rho)$ based on its Euclidean distance from the previously encoded mean encoding of all the segments $S_i(\tau, \lambda, \rho)$. We use the following equation to classify a segment:

$$\underset{i}{\operatorname{argmin}} ||S_i(\tau, \lambda, \rho) - E(\tau, \lambda, \rho)|| \quad (2)$$

where, the operator $||x||$ denotes the l^2-norm of the variable x. As an illustration, we consider the values in Table II. We take each of the 12 entries in the middle and compare it with the four final encodings to determine its class. Out of the 12 entries, this approach correctly classifies 10 of them. The two entries (Segment 0 Pass 1, and Segment 0 Pass 2) are incorrectly classified. Such errors can be eliminated using more training of the classifier.

IV. EVALUATION

A. Experimental Setup

In all of our experiments we use a LG G4 smart phone with an Android Marshmallow operating system to collect and log sensor data. The G4 has a Hexa-core (4x1.4 GHz Cortex-A53 2x1.8 GHz Cortex-A57) CPU, an Adreno 418, 32 GB of internal storage, and 3 GB of RAM. It comes with a suite of sensors including an accelerometer, a gyroscope, a proximity sensor, a compass, a barometer, a color spectrum sensor, and an A-GPS.

To prepare the data for our experiment, we create CSV files with each row representing time slices of 100 ms and columns representing accelerometer, magnetometer, and degree values as well as left and right maneuver detections, when they were actually being made, and when we were stopped at certain bus stops. We produced a new CSV file for each run we made with the phone. The bus route we drove is shown in Figure 4 and Figure 6.

We use the magnetometer and accelerometer sensors to compute compass degree values, which are used as inputs in the turn detection algorithm. The biggest challenge to our turn detection algorithm has been the instability of the magnetometer sensor as environmental factors heavily influenced its readings. This issue has been especially prevalent in times of high traffic on our bus route. Typically, high degrees of traffic are encountered between the hours of 4:00 PM and 6:30 PM. A work around to that is to create separate models for different road conditions and/or times of the day.

B. Performance of Turn Detection

To test our turn detection algorithm, we take three runs with four stops each while running an Android application that collects compass degree values. The accuracy of the algorithm, when compared to the ground truth values for turns recorded by a human operator is 95.7%. Figure 7 shows the number of detected and missed left and right turns. We observe that except for only one left turn, the algorithm has been successful in detecting all the turns.

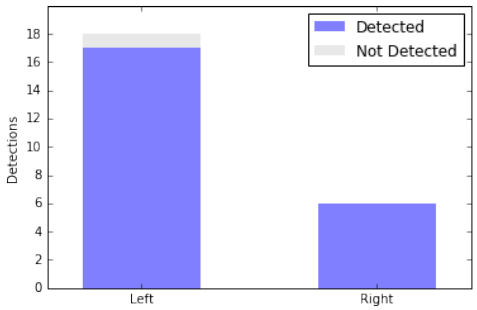

Fig. 7. The accuracy of left and right turns detection is over 95.7%.

Sometimes, the turn detection algorithm is susceptible to detecting false turns, especially when a route has a high degree of curvature. For example, the path between bus stops 1 and 2, called segment 1, displayed in Figure 6 has this characteristic. False turns encoded in segment 1 are seen because of this. Table III shows the distributions of true and false detected left and right turns. 'True Left' or 'True Right' mean the vehicle has made an explicit left or right turn that is detected correctly by the phone. 'Actual' represents turns recorded by the user. 'False Left' or 'False Right' means that the phone detected a turn while the user was not making one.

C. Performance of Segment Detection

Fortunately, false turns proved to be a characteristic of particular segments. If a large amount of false turns were

TABLE III
DISTRIBUTION OF TRUE AND FALSE TURN DETECTIONS

	Actual	Pass 1	Pass 2	Pass 3
True LEFT	6	5	6	6
True RIGHT	2	2	2	2
False LEFT	0	2	1	2
False RIGHT	0	2	4	4

detected in a segment, it was interpreted as a path with a road with a high degree of curvature. It could then be more easily differentiated from a segment with a more regular path. Our process of encoding the turns, transforming them into useful values, and classifying segments is described in Section III-E. Figure 8 is a chart displaying the runs per segments and the ones that are correctly detected by the classifier. Unlike our turn detection algorithm, our segment detection algorithm is not run in real time; we evaluated the algorithm offline after the trips. We leave it as a future work to integrate this step into the main smart phone application and perform a user study to understand its full potential.

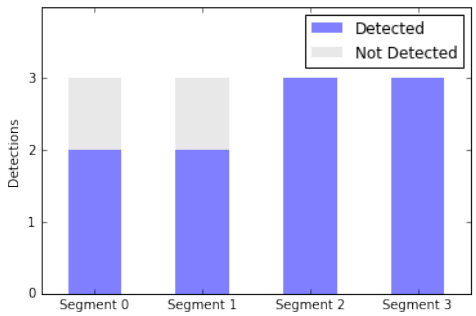

Fig. 8. Detections of total runs per segment

V. RELATED WORK

There have been plenty of research done on the subject of activity recognition utilizing some combination of accelerometer based sensor readings. [3] presented a system for context recognition relying on GPS and accelerometer. Similarly, [5] also worked on context recognition, but their system incorporates audio for classification and provides a solution that optimizes power consumption. Some more specific instances of activity recognition include Nuricell [6], a system that detects specific road conditions such as bumps and breaks using accelerometer, microphone, GSM, and GPS.

Another category of works we looked into was bus arrival predictions. Many of these rely solely on the bus arrival and running data to make their predictions. [2] examined bus arrival prediction using methods such as support vector machine (SVM), artificial neural network (ANN), and k-nearest neighbours algorithm (k-NN) with bus arrival information. GPS information is another method that can be used for these research, as presented in [7]. Their system uses GPS to display bus location, and they have also worked out an algorithm that can calculate the corresponding bus delay. [8] developed a GPS based system that helps cognitively disabled people to know when to get off a public transportation system.

The last area of research we have studied is the works that addresses driving patterns and habits such as inattentiveness, drowsiness, and aggression on the road. Many such systems require the use of camera [9], [10]. [9] uses camera information for eye closure detection as their primary method for drowsiness detection, while the CarSafe app [10] collects both camera and sensor readings from accelerometer and gyroscope to identify a common set of dangerous driving behaviors and road conditions. Their method uses computer vision and machine learning algorithms to detect road conditions and facial expressions simultaneously. Another system [4] makes use of sensor fusion (accelerometer and gyroscope readings) and Dynamic Time Warping for driving event recognition and aggression categorization and elaborates more on detection of various driving maneuvers.

VI. CONCLUSION

In this paper we present a system for bus route navigation and notification that can effectively work in city environments to detect a user's regular bus routine and make bus stop predictions. Our design avoids dependence on GPS functionality and instead relies on compass sensors, which are far more energy efficient. We perform a system evaluation and showed our design produces 83% accuracy in bus route and stop detection. Our future work includes performing additional in-depth testing on various bus routes, real-time integration of segment detection, and improving upon turn detection and segment detection accuracy.

REFERENCES

[1] UNC BusStopAlert Project. https://github.com/uncmobile/BusStopAlert.

[2] W. H. L. Bin Yu and M. L. Tam. Bus arrival time prediction at bus stop with multiple routes. In *Transportation Research Part C: Emerging Technologies*, 2011.

[3] J. J. Hao Xia, Yanyou Qiao and Y. Chang. Using smart phone sensors to detect transportation modes. In *Sensors*, 2014.

[4] D. A. Johnson and M. M. Trivedi. Driving style recognition using a smartphone as a sensor platform. In *International IEEE Conference on Intelligent Transportation Systems*, 2011.

[5] Y.-K. L. Manhyung Han, La The Vinh and S. Lee. Comprehensive context recognizer based on multimodal sensors in a smartphone. In *Sensors*, 2012.

[6] P. Mohan, V. N. Padmanabhan, and R. Ramjee. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *SenSys*, 2008.

[7] T. S. Naoki Kanatani and T. Kawamura. Development of bus location system using smart phones. In *SICE Annual Conference 2010, Proceedings of*, 2010.

[8] R. W. H. Richard F. Fabiano. Bus passenger alerting system, 06 1991.

[9] M. S. N. S. Shreya.P.Patel, Bhumika.P.Patel and Hinaxi.M.Patel. Detection of drowsiness and fatigue level of driver. *International Journal for Innovative Research in Science and Technology*, 1(11):133–138, 2015.

[10] C.-W. You, N. D. Lane, F. Chen, R. Wang, Z. Chen, T. J. Bao, M. Montes-de Oca, Y. Cheng, M. Lin, L. Torresani, and A. T. Campbell. Carsafe app: Alerting drowsy and distracted drivers using dual cameras on smartphones. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 13–26, New York, NY, USA, 2013. ACM.