# CityDrive: A Map-Generating and Speed-Optimizing Driving System

Yiran Zhao*, Yang Zhang†, Tuo Yu*, Tianyuan Liu*, Xinbing Wang*, XiaohuaTian*‡, and Xue Liu§
*Department of Electronic Engineering, Shanghai Jiao Tong University, Shanghai, China
†Department of Computer Science, Shanghai Jiao Tong University, Shanghai, China
‡State Key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications)
§School of Computer Science, McGill University, Canada
{zhaoyiran0522, jeffreyzhangyang, yutuo, jjlty_yuan, xwang8, xtian}@sjtu.edu.cn, xueliu@cs.mcgill.ca

***Abstract*—There have been many traffic light control systems around the globe, but the high cost of infrastructure and maintenance hinders their wide deployment. However, speed-advisory systems enabled by on-vehicle devices are much cheaper and easier to deploy. The first challenge of such systems is to get the traffic signal schedule in complex intersections. The second challenge is to get map information and calculate the distance. Facing these challenges we devise and implement a speed-advisory driving system called CityDrive, which harnesses the sensor and GPS data from a wide participation of smartphones to suggest proper speed for drivers so that they arrive at intersections in green phase. CityDrive first generates a road map and then infers traffic signal schedules, using only smartphones and a server. CityDrive does not eliminate stops at intersections, but it tries to maximize the probability that vehicles cruise through intersections in green phase. Both simulation and real test show that this continuous speed advisory service effectively smoothes traffic flow and significantly reduces energy consumption.**

## I. Introduction

Nowadays the traffic lights dominate city traffic, coordinating vehicles from different roads to safely transfer to other roads. But absent of traffic light information, drivers can hardly adopt an appropriate speed and thus they often come to a halt reaching intersections. This stop-and-go driving pattern causes increased fuel consumption, air pollution, traffic congestions and road accidents. According to Texas Transportation Institute, in 2011, congestion caused urban Americans to travel 5.5 billion hours more and to purchase an extra 2.9 billion gallons of fuel, resulting in additional cost of $121 billion [1].

Therefore, it is desired either to dynamically adapt traffic signal timing to traffic flow, or to inform drivers about traffic signal schedules and suggest a proper speed. In the first case, extensive studies and implementations on intelligent traffic signal control have been made. Many studies view the traffic signal control problem as a job scheduling problem [2], [3]. They use vehicular ad hoc networks (VANETs) [3] or vehicle to infrastructure (V2I) communication scheme [4] or combined [5] to collect and aggregate real-time speed and position information on individual vehicles to optimize signal control at intersections. As for real-world implementations, systems like SCATS [6], SCOOT [7], RHODES and its successor MILOS [8] have been smoothing traffic flow in more than a hundred cities around the globe. But overall, the high cost of infrastructure deployment and maintenance makes it difficult to implement in a wide range of cities. In the second case, the benefits that Intelligent Speed Adaptation (ISA) yields to the environment is evaluated in [9]. And a system using a decision tree based classification model is proposed to predict the future traffic and help drivers to schedule their activities [10]. The current Green Light Optimal Speed Advisory (GLOSA) system is largely based on roadside message signs or countdown timers that allow drivers to assume a proper speed [12]. Unfortunately, most of these systems are costly and difficult for wide deployment. With the advancement of mobile phone technology, smartphone now assumes a promising role in traffic coordination. An evaluation of a cost-efficient traffic light assistant with smartphones as human-machine interface is presented in [11]. By proper design, future systems based on smartphone application should yield more results.

In this paper, we devise and implement a cost-effective, infrastructure-less system to dynamically infer traffic signal schedule and advise proper speed for drivers. Our proposed CityDrive system only runs on smartphones and an Internet server. The server of CityDrive first mines GPS and smartphone sensor data to establish a road-intersection topological map, and then gathers vehicle acceleration time and location to infer and calibrate traffic signal phase and timing information at each intersection. By requesting such information and combining local map database, smartphones are able to deduce the best driving speed. The use of reliable connection (GSM, 3G, LTE, etc.) enables continuous GLOSA on all smartphones running CityDrive at any location on road segments.

Our work has the following contributions:

1) Our system generates a road-intersection map, including detailed connections within intersections. We also take into account one-way roads.
2) Our system enables GLOSA even at large complex intersections with various traffic signal schedules.
3) Our system tolerates user fiddling, and is easy to deploy in any vehicle.

On the other hand, our system also relies on the following assumptions:

1) We assume that most vehicles are equipped with smartphones capable of running CityDrive.
2) We assume that all drivers obey traffic rules, i.e., no acceleration at red light, and no halt at green light.

978-1-4799-3360-0/14/$31.00 ©2014 IEEE

3) Right turns are always allowed, if such traffic movement exists.
4) Smartphones have ample power supply in vehicles.

Our CityDrive can also provide foundations for many other applications:

1) *Commercial map revision and refinement.* Since our system collects GPS traces to construct road-intersection map, the fine-grained road segments and intersection structure can be used to refine commercial maps [16].
2) *Traffic signal planning advisory service.* CityDrive's central server collects average vehicle speed of a particular road. If the average speed is lower than a threshold, the system can infer road congestions and provide traffic signal adjusting suggestion to the transportation department.
3) *Driving behavior and road condition estimation.* Our system calculates and records vehicle acceleration and heading direction. Thus it's possible to infer driver's behavior and suggest better commuting safety. In addition, the bumps and potholes can be detected by the smartphone sensors [15] and thus proper road maintenance advice can be obtained.
4) *Red Light Violation Advisory (RLVA).* RLVA service warns the drivers when they try to speed up and squeeze through the intersection when the green light is about to turn red. When the smartphone detects acceleration while suggesting the driver to slow down, warning will be given and driving safety record will be discredited.

The rest of this paper is organized as follows: In section II we discuss related work. Then in section III we detail the system architecture and methodologies. In section IV the simulation result is presented. Also we evaluate the performance of our system in real-world experiments in section V. Finally section VI offers the conclusion and discusses our future work.

## II. Related Work

There have been some infrastructure-less GLOSA systems that use information gathered from sensors or cameras on wireless devices to collaboratively infer traffic signal schedule.

The most straightforward approach to get the traffic light state is through processing video images from cameras [17]-[20]. But generally speaking, for a relatively complex situation, such as an urban road environment, confusion between traffic lights, traffic signs and tail lamps will appear easily. Despite difficulties in image processing, SignalGuru [12] successfully utilizes cell phone camera and adhoc network to collaboratively predict traffic signal phase and advise drivers to maintain a reasonable speed. However, this approach may be problematic when the traffic light is deformed, blurred by bad weather or obstructed by other vehicles. Furthermore, it is difficult for SignalGuru to pick the correct traffic light when facing a complex group of traffic lights. Later, a probabilistic prediction of traffic signal timings based on partial phase timing information is introduced in [13]. But in reality, such information is limited. In [21] the authors designed an Adaptive Driving Speed Guiding system that calculates optimal speed from traffic light timing downloaded from the database

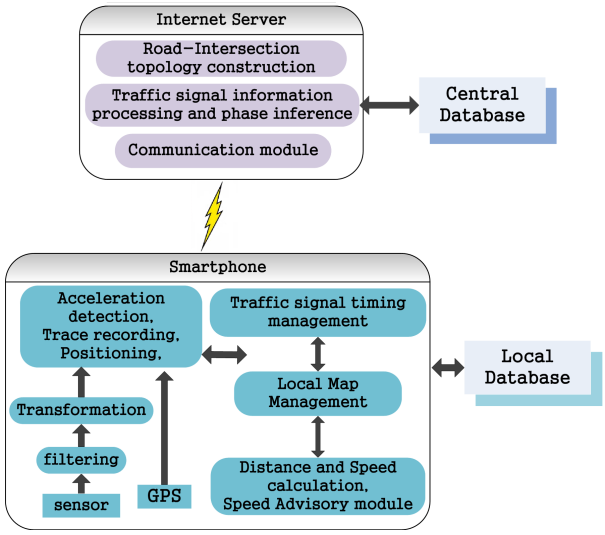



Fig. 1: CityDrive System Architecture.

owned by the ministry of transportation. However, such data may be inaccurate. Another way of using velocity profiles to estimate traffic signal schedule is presented in [14], but the prediction accuracy could be improved by using acceleration data. Finally, all above works didn't formulate a way to accurately get the distance to the next intersection.

To address the above problems, our proposed system is able to calculate distance by extracting road-intersection topological information, and is able to infer phase information in complex intersections with multiple sets of traffic lights. In addition, CityDrive does not require careful positioning of smartphones, and allows passengers to hold them in hands since our system tolerates normal user fiddling. In a word, our proposed system is more likely to be implemented in real world.

## III. System Implementation and Algorithm

The general system architecture is illustrated in Figure 1. We exploit smartphone capabilities and configure a computer as the server to implement the whole system.

### A. Android smartphone capabilities exploited in our system:

First, the concept of device's body coordinate system and local NED coordinate system is introduced in [25]. Then we explore the following capabilities on smartphones.

1) *3-axis Accelerometer.*
The on-device accelerometer measures the acceleration applied to the device in body coordinate system, including the force of gravity. Apart from acceleration sensor, the android system also provides gravity sensor which estimates the direction of gravity using the same accelerometer.
2) *3-axis Magnetometer.*
Smartphones also integrate magnetometer to measure the geomagnetic field in body coordinate system. We use geomagnetic field and gravity as the bridge to transform the acceleration vector from the body coordinate system to local NED coordinate system.

3) *Global Positioning System (GPS).*
GPS accuracy is greatly improved following the advent of DGPS. Smartphones used in our test show that the average accuracy of position is about 10-15 meters, with a minimum of 4 meters when the conditions fit. The GPS provides smartphones with information about devices' position (longitude and latitude), speed (in m/s), bearing (heading direction, from 0 to 360 degrees), UTC time (in milliseconds since 1 Jan, 1970), accuracy (in meters), etc.

### B. Acceleration module

Since the measurements from smartphone's 3-axis accelerometer contain a lot of noise, the acceleration data is first filtered with a cut off frequency of 2Hz. Then we transform the acceleration vector into local NED coordinate system.

*1) Coordinate system transformation:*

By coordinate system transformation the acceleration can be projected onto the vehicle's travelling direction. The transformation follows Z-Y-X rotation sequence. Denote $\theta_z,\ \theta_y,\ \theta_x$ as the angle of rotation about intermediate Z, Y, X-axis in sequence. Thus the rotation matrix $\boldsymbol{R_{nv|b}}$ from the body frame to the local NED frame is given by [25]:

$$\boldsymbol{R_{nv|b}} =$$

$$\left[\begin{array}{c|c|c} C(\theta_y)C(\theta_z) & C(\theta_y)S(\theta_z) & -S(\theta_y) \\ \hline S(\theta_x)S(\theta_y)C(\theta_z) & S(\theta_x)S(\theta_y)S(\theta_z) & \\ -C(\theta_x)S(\theta_z) & +C(\theta_x)C(\theta_z) & S(\theta_x)C(\theta_y) \\ \hline C(\theta_x)S(\theta_y)C(\theta_z) & C(\theta_x)S(\theta_y)S(\theta_z) & \\ +S(\theta_x)S(\theta_z) & -S(\theta_x)C(\theta_z) & C(\theta_x)C(\theta_y) \end{array}\right]$$

Where $C(\theta),\ S(\theta)$ denote $\cos(\theta),\ \sin(\theta)$, respectively.

To calculate the rotation matrix, we have to use the two reference vectors, i.e. the gravity and the geomagnetic field. Note that we ignore the difference between magnetic north and geodetic north. Thus, the acceleration vector $\vec{\boldsymbol{a}}_{\boldsymbol{nv}}$ in local NED coordinate system can be expressed by:

$$\vec{\boldsymbol{a}}_{\boldsymbol{nv}} = \begin{pmatrix} a_{x_{nv}} \\ a_{y_{nv}} \\ a_{z_{nv}} \end{pmatrix} = \boldsymbol{R_{nv|b}} \begin{pmatrix} a_{x_b} \\ a_{y_b} \\ a_{z_b} \end{pmatrix} = \boldsymbol{R_{nv|b}} \cdot \vec{\boldsymbol{a}}_{\boldsymbol{b}}$$

Where $\vec{\boldsymbol{a}}_{\boldsymbol{b}}$ is the acceleration vector in body coordinate system.

*2) Vehicle acceleration detection:*

We project the acceleration vector onto the direction which the vehicle is heading. The vehicle's direction is gained from GPS information and is frozen once the vehicle's speed is lower than a threshold. This is because the bearing reading from GPS data is inaccurate when the speed is zero. The projection naturally filters away noise or user fiddling acceleration perpendicular to the vehicle's heading direction. And to further reduce false positive produced by user fiddling, we set up a time window to calculate the aggregated acceleration in duration of about one second. In the cases like screen touching, moving or even shaking, the movement is usually less than one second so that the aggregated acceleration is neutralized by deceleration of approximately the same magnitude.

In real-world tests, five android phones (three LG Nexus 4, one LG F160, one Samsung Nexus S) experienced 416 times of normal acceleration in a vehicle (Corolla, Toyota). Acceleration detection delay after hitting the gas is also recorded. Results are:

1) The number of missing detection is 15 (3.6%).
2) The number false positive is 8 (1.9%).
3) Mean detection delay is 1.014 second, with standard deviation 0.334 second.

The acceleration detection success rate is relatively high (94.5%), which lays good foundation for further tests.

### C. Road-intersection topology construction

Since the length of road segments and the structure of intersections are not available in commercial maps, the first step is to construct the road-intersection topological map. Volunteers are needed to gather acceleration data and GPS traces in real driving, and transmit the data to the central server when wifi is available.

But not all vehicle acceleration is recorded. Only if the GPS speed reading falls to zero (or smaller than a threshold like 1 m/s) and this zero-speed state lasts more than 10 second will the succeeding acceleration be valid and recorded. Thus, most acceleration on road segments is ignored. To reduce the impact of inaccurate data, we discard acceleration vectors or GPS traces that have an GPS accuracy worse than 30 meters.

*1) Locating candidate intersections:*

To locate candidate intersections we use an approach similar to mean shift [22], in which successive computations of the mean shift yield a path leading to a local acceleration density maximum. The shifting circle reduces its size every iteration and ultimately stops shifting if the circle size is approximately the size of a typical intersection and the mean shift distance is less than a threshold. This process is shown in Figure 2. We call the circle of the last iteration final circle, shown in red color.

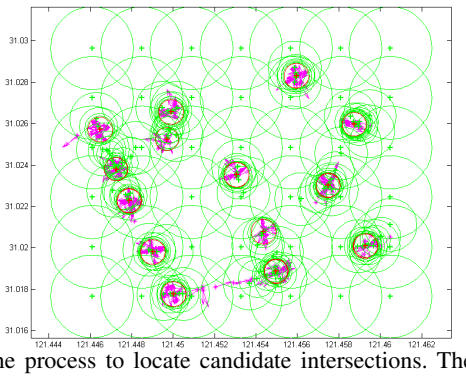



Fig. 2: The process to locate candidate intersections. The magenta arrows are acceleration vectors. The green circles are the shifting circles, and the red ones are final circles (candidate intersections).

Analysing the vector pattern inside the final circles can filter out some false intersections in parking lots or residential areas. If the acceleration vectors point outwards (like in Figure 3), the final circles are false intersections. Otherwise, they indicate valid intersections (like in Figure 4).

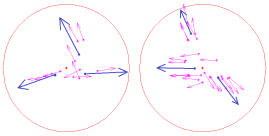


Fig. 3: False intersections.

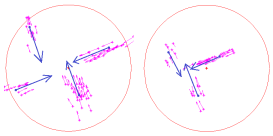
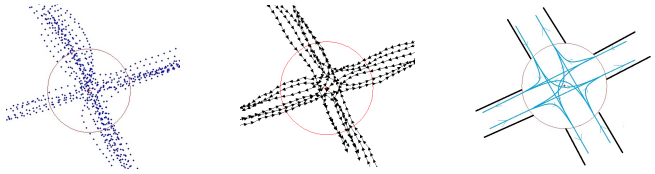


Fig. 4: Valid intersections.

*2) Clarifying intersection structure:*

Then GPS trace information is analysed to identify branches. Directed GPS trace segments that share the same start and end final circles, or share the same incoming direction are grouped into one branch. If the the number of branches is less than three or more than five, the intersection is discarded. Branches within the same final circle are also connected by GPS traces, resulting in detailed intersection structure which supports one-way road (Figure 5).



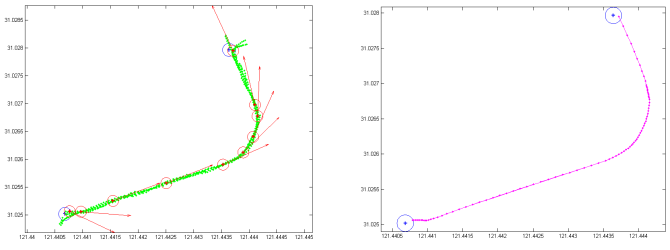
(a) (b) (c)

Fig. 5: (a) shows raw GPS traces, (b) shows the typical traces that connect the branches, (c) shows the logical connection of branches in the intersection.

Every resulting intersection is given a unique ID. And associated with each ID are the indexed branches and their connectivity information. Then the system uses GPS traces to link the intersections and establish a road map.

*3) Linking intersections:*

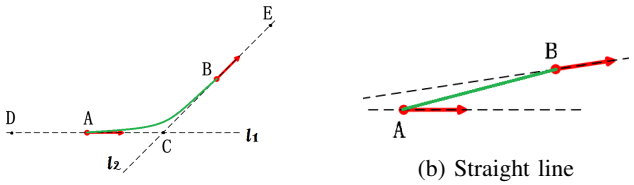
To generate a series of directed points that represent a road segment, we use an algorithm similar to weighted mean shift that finds the centroid point and mean direction of a cluster of GPS points sharing approximately the same direction. The weight of each GPS point is related to its GPS accuracy. The shifting direction is restricted to be orthogonal to the mean heading direction, and the final circle has a radius of a typical road width. The center of the final circle is called the anchor point (with direction). We perform weighted mean shift every 10 meters, but only if the mean direction changes more than 20 degrees or the distance to the last anchor point becomes larger than a threshold will it produce a new anchor point (Figure 6(a)).



(a) (b)

Fig. 6: (a) shows the raw GPS traces (green) and generated anchor points (red point with an arrow). (b) shows the representation of the road segment in (a).

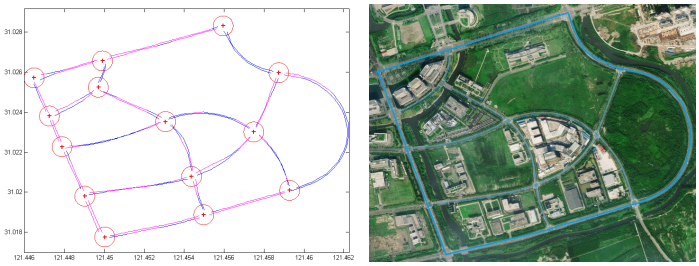
Then we use second order B-spline curves that represent road segments as piecewise defined polynomials with first order continuity at the anchor points. For every two successive anchor points (A and B in Figure 7(a)), the B-spline curve has to pass through them. Note that the B-spline control points are not anchor points. To derive the control points, we extend the lines along the direction of the anchor points ($l_1$ and $l_2$) and the joint point is denoted as C. Then, the control points are C, D (on $l_1$) and E (on $l_2$) such that $\overline{DA} = \overline{AC}$ and $\overline{CB} = \overline{BE}$. If the road is straight, B-spline may not apply, so a straight line is used instead (Figure 7(b)). We calculate the B-spline curves or straight lines between each consecutive pair and the result of Figure 6(a) is shown in Figure 6(b). After all roads between any pair of intersections are represented as piecewise polynomials, the distance can be calculated, thus the complete topological graph of a region is established.




(a) B-spline curve (b) Straight line

Fig. 7: (a) shows the case using a B-spline curve to represent the road segment, (b) shows the case using a straight line instead.

The generated map of our testbed is shown in Figure 8(a) (connections within intersections are omitted). The blue segments are B-spline curves and the magenta segments are straight lines.



(a) (b)

Fig. 8: (a) shows the generated map of our testbed. (b) shows the region in Google Map.

*4) Constructed central database:*

The intersection ID, its position, the indexed branches in this intersection, the distance and each anchor point's (A.P.) information is stored in the server's central database and can be downloaded on smartphones, as is illustrated in Table I.

### *D. Traffic signal schedule inference and update*

Based on the existing road-intersection map, future vehicle acceleration information will be associated with the corresponding intersection and a certain branch. When a vehicle stops at an intersection, the smartphone records the start time and end time of the zero-speed state. On detection of valid acceleration, the smartphone monitors the GPS information. Once the leaving branch is identified, the smartphone will send a message containing the intersection ID, the index of in and out going branches, the time interval from acceleration to transmission, the zero-speed state time interval, and the vehicle

TABLE I: Constructed database.

| Intersection ID | longitude | latitude | Branch $R_n$ angle | $R_n$ connected intersection ID | length of $R_n$ | $A.P.$ number | $A.P._i$ longitude | $A.P._i$ latitude | $A.P._i$ direction |
|---|---|---|---|---|---|---|---|---|---|
| 001 | 121.34632 | 30.13543 | 15 | 002 | 210 | 10 | 121.45334 | 30.32343 | 353 |

ID to the server. The recording of zero-speed state time is to identify vehicles that waited for the same traffic signal phase in a queue, so that only the first acceleration event is valid and will be stored in the server. The server thus utilizes the phase transition time to infer traffic signal phases and their time length. To illustrate the process, here we take a four-leg intersection (Figure 9) as an example and construct the in-out branch table (Table II).

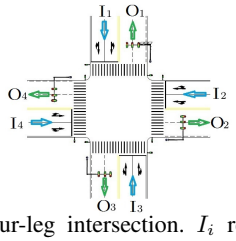



Fig. 9: A typical four-leg intersection. $I_i$ represents the $i'th$ incoming branch, and $O_i$ represents the $i'th$ out-going branch.

TABLE II: In-out branch table.

| | $I_1$ | $I_2$ | $I_3$ | $I_4$ |
|---|---|---|---|---|
| $O_1$ | \ | \ | 1 | 4 |
| $O_2$ | 2 | \ | \ | 3 |
| $O_3$ | 1 | 4 | \ | \ |
| $O_4$ | \ | 3 | 2 | \ |

Let $(O_i, I_j)$ $(i, j = 1, 2, 3, 4)$ denotes the movement from $I_j$ to $O_i$, and is shortened to $(i, j)$. Note that some elements are backslashes, because we don't consider turn-back or right-turn. The numbers in the Table II represent the movement's phase, which will be explained later.

*1) Traffic signal phase and phase sequence inference:*

This is the initial phase of establishing traffic signal schedule. We assume that every movement $(O_i, I_j)$ happens only once in one traffic signal cycle, and that the traffic signal cycle length, the length of each phase, and the sequence of the phases are stable within a relatively long period.

First the traffic signal cycle length (denoted as $T$) should be determined. Here are some notations to make. Each movement $(O_i, I_j)$ is a called a state, denoted as $S_{ij}$, and states that start at approximately the same time are merged into one state. Let $\boldsymbol{S}$ denotes the set of states of a intersection, and $N_s$ denotes the total number of states (either merged or not). For convenience, we give a short subscript number for each state regardless of their sequence (i.e. $S_i$, $i = 1, 2, ...N_s$), and let "event" represents a valid acceleration event. The event time associated with $S_i$ is denoted as $t_i^k$, where $k$ is the index of that event in $S_i$'s event array.

The event array for $\{S_i, S_i \subset \boldsymbol{S}\}$ is organized in Table III, in which $N_i$ $(i = 1, 2, ...N_s)$ denote the total event number of $S_i$ at a particular time. Upon receiving another event of $S_n$, the time $t_n^{N_n+1}$ is compared to $\{t_i^j, i \neq n, 1 \leq j \leq N_i\}$ to see if $S_i$ and another state occurring almost at the same time (within 5s of time difference) can be merged into one. If no merge happens, $t_n^{N_n+1}$ will be added to the end of array of $S_n$. If $S_n$ and $S_l$ are merged, the elements in the two event arrays are also merged in time order, and $N_s$ decreases by one. Note that the relationship between $S_i$ and $S_{ij}$ is recorded.

TABLE III: Event table for each state.

| | | | | |
|---|---|---|---|---|
| $S_1$ | $t_1^1$ | $t_1^2$ | ... | $t_1^{N_1}$ |
| $S_2$ | $t_2^1$ | $t_2^2$ | ... | $t_2^{N_2}$ |
| ... | ... | ... | ... | ... |
| $S_{N_s}$ | $t_{N_s}^1$ | $t_{N_s}^2$ | ... | $t_{N_s}^{N_{N_s}}$ |

We use Table III to infer $T$ and traffic signal phases. For each row of $S_i$, the minimum of $\{t_i^j - t_i^{j-1}, j = 2, 3, ...N_i\}$ is calculated and denoted as $\Delta t_i$. We want to know how many events are enough to estimate an approximate value of $T$, since the value $T$ is important to later phase inference. Say the probability of an event of $S_i$ in one traffic signal cycle is $p_i$, and the probability that $T$ is *not* equal to $\Delta t_i$ from an event array experiencing $n$ traffic signal cycles is $X_i(n)$. Then probability $X_i(n+2)$ is equal to probability $X_i(n+1)$ if the $(n+2)^{th}$ cycle has no event plus probability $X_i(n)$ if $(n+2)^{th}$ cycle has an event and $(n+1)^{th}$ cycle has no event. So we have the following equation:

$$X_i(n+2) = (1-p_i)X_i(n+1) + p_i(1-p_i)X_i(n)$$

Solving this equation yields:

$$X_i(n) = C_1(\frac{1-p_i+\sqrt{\triangle}}{2})^n + C_2(\frac{1-p_i+\sqrt{\triangle}}{2})^n$$

Where $\triangle = -3p_i^2+2p_i+1$, $C_1 = \frac{1+p_i}{2\sqrt{\triangle}}+\frac{1}{2}$, and $C_2 = 1-C_1$. Then the probability that $T$ is equal to $\Delta t_i$ is $P_i(n) = 1 - X_i(n)$.

We analyse the relationship between $P_i(n)$, $p_i$, and $n$ and derive the Figure 10. The goal is to maximize the probability that the minimum of $\{\Delta t_i, i = 1, 2, ...N_s\}$ is $T$. For simplicity, we assume that $p_i = p_j = p$ $(1 \leq i, j \leq N_s)$. We want $P_i(n)$ for all $N_s$ states to be greater than 0.8, so that the probability that $min\{\Delta t_i, i = 1, 2, ...N_s\}$ equals $T$ approaches to 1, which is $P = 1 - (1 - 0.8)^{N_s}$. Typically, if by merging concurrent states, $N_s = 4$, then $P = 1 - (1 - 0.8)^4 = 0.9984$. Thus we can almost make sure that the approximate value of $T$ is correctly obtained. To estimate $p_i$ and $n$, we suppose the traffic signal cycle length is two minutes, and use the average number of events of $S_i$ every two minutes to represent $p_i$. Then by solving $n$ in equation $P_i(n) = 0.8$ the system knows that it should calculate $T$ after $2 \times n$ minutes.

Once the traffic signal cycle length $T$ is obtained (though not accurate), states in sequence in $\boldsymbol{S}$ are linked into loop chains. In Table III, to find the state closest following $S_k$, we calculate $min\{t_j^i - t_k^u\}$, $(1 \leq i \leq N_j, 1 \leq j \leq N_s, t_j^i > t_k^u)$. If $S_l$ closest follows $S_k$ after $\Delta t_{kl}$, this pair forms a link $S_k \rightarrow S_l$. We concatenate all links into loop chains $\{C_i\}$. In each loop chain, say $C_i$ : $S_k \rightarrow S_l \rightarrow ... \rightarrow S_m \rightarrow S_k$,

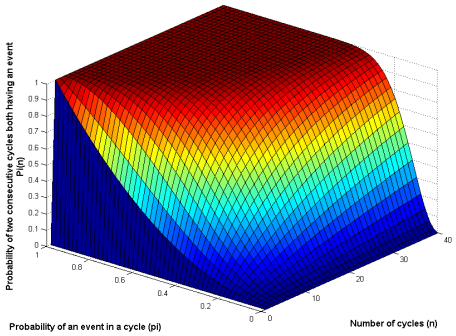
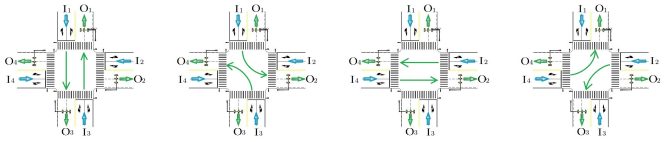



Fig. 10: $P_i(n)$ as a function of $p_i$, and $n$.

TABLE IV: Set of data for calibration.

| | | | | |
|---|---|---|---|---|
| $t_1^1$ | $ns_1$ | $ns_2$ | ... | $ns_N$ |
| $t_1^2$ | $ns_1'$ | $ns_2'$ | ... | $ns_N'$ |
| $t_2^1$ | $ns_1''$ | $ns_2''$ | ... | $ns_N''$ |
| $t_3^1$ | $ns_1'''$ | $ns_2'''$ | ... | $ns_N'''$ |
| ... | ... | ... | ... | ... |

if $\mid \sum_{p,q} \Delta t_{pq} - T \mid < 10s,\ (S_p \rightarrow S_q \subset C_i)$, then $C_i$ is a valid loop chain. Note that there may be a number of valid loop chains associated with one intersection, and that one state itself may also form a loop chain ($S_k \rightarrow S_k \rightarrow S_k ...$). Each time a merging happens, states in two loop chains are interposed or merged, forming one valid loop chain. States in a valid loop chain are called phases. For a four-leg intersection, states should be finally merged into four concatenate phases (i.e. $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_1$, $N_s = 4$). And typically, the four phases are shown in Figure 11, which explains the phase numbers in Table II.



(a) Phase 1 (b) Phase 2 (c) Phase 3 (c) Phase 4

Fig. 11: Typical phases in a four-leg intersection with $N_s = 4$.

*2) Traffic signal timing calibration and update:*

Each valid loop chain can produce a traffic signal time schedule for each phase. Specifically, consider a loop chain with $N$ phases ($S_1 \rightarrow S_2 \rightarrow ... \rightarrow S_N \rightarrow S_1$), and let $dts_i$ denote the duration of phase $S_i$, ($dts_i = \Delta t_i$ for the first time). Without loss of generality, we suppose that the system starts to infer traffic signal schedule at an event of $S_1$, at time $t_s$. Upon receiving an event, the system calculates the number of times each phase occurred (e.g. $S_i$ occurred $ns_i$ times), and stores them in Table IV.

As received events continue to add new rows in the Table IV, we dynamically make a calibration when the size of rows reaches a threshold ($i.e.$ $M$, this value empirically ranges from 20 to 40). Resolve the Table IV into $(N+1)$ columns, each forming a vector, namely, $\boldsymbol{t, ns_1, ns_2, ..., ns_N}$. With these vectors, we are going to find the calibrated starting time ($t_0$) and calibrated phase length ($dts_i,\ i = 1, 2, ...N$).

We formulate a target function $F$ that represents the mean square error (MSE) of predicted time:

$$F = (t_0 - t_s)^2 + \sum_{i=1}^{M}(t_0 + \boldsymbol{ns_1}(i) \cdot dts_1 + \boldsymbol{ns_2}(i) \cdot dts_2 + ... + \boldsymbol{ns_N}(i) \cdot dts_N - \boldsymbol{t}(i))^2 \quad (1)$$

Where $M$ is the size of the vectors $\boldsymbol{ns_i, t}$. Minimize this target function by taking partial derivative of each variable in Equ.(1):

$$\frac{\partial F}{\partial t_0} = 0,\ \ \frac{\partial F}{\partial dts_i} = 0,\ \ i = 1, 2, ...N$$

Solving the above equations yields calibrated $t_0$, $dts_i$, ($i = 1, 2, ...N$). After each calibration, the first five rows are discarded, and the system will make the next calibration after receiving five new rows of events. In most of our tests, the MSE algorithm utilizes a fair amount of history data to predict phase length accurately.

### *E. Routing and speed advising*

*1) Route planning:*

To pick the right signal phase of the next intersection, the smartphone has to gain prior knowledge of travel route. So Dijkstra algorithm is employed on smartphones to calculate a route with the least travel time. For real-time routing, smartphones request average vehicle speed information of interested roads that lie in the searching region. The searching region is a circle with diameter specified by the starting point and the destination point. If the current average speed on a certain road is not available, the smartphone assumes that the speed is up to the legal limitation (typically 60 km/h). To improve efficiency, only road segments that lie inside the searching region will be considered in the Dijkstra algorithm. And only if there is no existing route will the system enlarge the circle to search more pathes. Smartphones also has to send average speed information to the server when it finishes one road segment, so that the server can provide other vehicles with better real-time routing.

*2) Best speed calculation:*

Once the route is determined, the smartphone requests traffic signal schedule information of the next intersection. The server returns the traffic signal cycle length ($T$) of the intersection ahead, the remaining time for the desired phase to turn green ($t_g$), and the remaining time to turn red ($t_r$). Thus the time of reaching the intersection should be $t(n)$, such that $t_g + t_{sg} + n \times T < t(n) < t_r - t_{sr} + n \times T$ (n=0,1,2...). $t_{sg}$ and $t_{sr}$ are time intervals used to reduce the impact of prediction error, and are empirically set to 10 seconds.

To calculate the remaining distance, the smartphone turns to the local database. Given the anchor points on that road segment and the vehicle's current location and heading direction, the distance to the next intersection is calculated by integrating the length of the B-spline curves or straight lines formed in the way explained in Section III, C(3).

Once the distance ($d$) and travel time is obtained, the optimal speed should be $v_{opt}$ such that $\frac{d}{t_r - t_{sr} + n \times T} < v_{opt} < \frac{d}{t_g + t_{sg} + n \times T}$. And $n$ should be the least integer such that $v_{opt}$ does not exceed speed limit. The smartphone should send

a request every time the vehicle passes by an intersection or every minute on a road segment. Also, the distance is calculated every GPS location update.

## IV. Simulation

After the inference of traffic signal phase number and phase sequence, the server enables GLOSA service on vehicles using our system. The vehicles' acceleration helps the server to calibrate traffic signal prediction, but the system tries to reduce the occurrence of acceleration. To find this dynamic equilibrium and to test our system's effectiveness, we carry out a Matlab simulation with the following empirical parameters.

1) We test our system on a map of $4 \times 4$ intersections (Figure 12). To extend the map, we connect node 1,2,3,4 with node 13,14,15,16, respectively, and also node 1,5,9,13 with node 4,8,12,16.
2) Let the 16 intersections be identical, with the same 4-phase traffic signal schedule as Figure 11 shows. Suppose phase 1,3 last 50 seconds, and phase 2,4 last 30 seconds.
3) Suppose the speed limit is 17 m/s, and that drivers will randomly choose a speed between 12 to 17 m/s if there is no advisory speed.
4) The acceleration detection time plus the data transmission time is a random variable uniformly distributed in the range from 1 to 5 seconds.
5) Each vehicle chooses a random route and keeps running all the time.
6) Assume that the GPS data is $100\%$ accurate without latency.

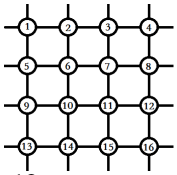


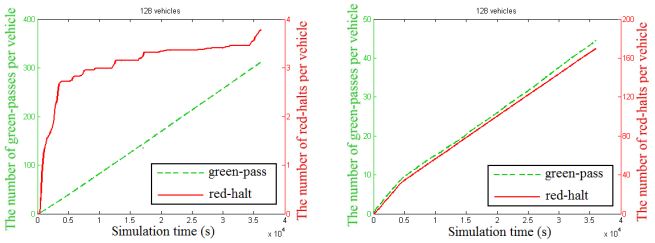

Fig. 12: Simulation Map.

### A. Simulation results with and without our system

We first place 128 vehicles with randomly chosen routes (on average 2 vehicles per road segment per direction). The initial stage of this simulation is after the inference of traffic signal phases, so there are 16 sets of initial values of phase length and traffic signal cycle length (with $-2$ to $+2$ seconds of random deviation from accurate values). During the travel, each vehicle requests the next intersection's traffic signal schedule only once when it passes the previous intersection.

The simulation time is 10 hours (36000s), and we record the time spent waiting and the number of times of arriving at intersections in green phase (green-pass) and in red phase (red-halt). To compare the total distance traveled, we prolonged the simulation time to 65000s.

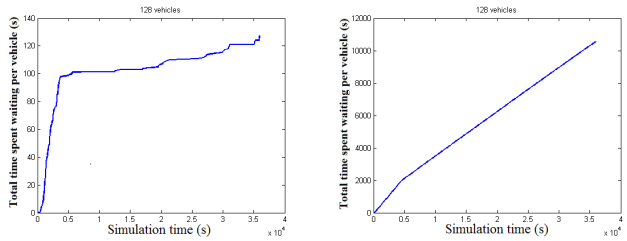
Figure 13 shows that without our system, there will be on average 170 red-halts every 44 green-passes (not including right turns). But using our system, there are only 3.8 red-halts every 311 green-passes. This is reasonable since a very small number of vehicles' acceleration data will help a big number of other vehicles pass through in green phase.




(a) With CityDrive (b) Without CityDrive

Fig. 13: (a) and (b) show the number of green-passes and red-halts with and without CityDrive, respectively.

Figure 14 shows that without our system the total waiting time per vehicle is 10557s (about 3 hours), while with our system the number is 127s ($98.8\%$ reduction). Also, Figure 15 shows the distance traveled per vehicle with speed advice also increased $49.8\%$. Because our system suggests the maximum speed below the speed limit to cruise through intersections, the actual time spent travelling will not increase. On the contrary, without speed advice, the drivers choose a speed ranging from 12 to 17 m/s. Such unwise speed will miss more green phases, causing more waiting time.




(a) With CityDrive (b) Without CityDrive

Fig. 14: (a) and (b) show the waiting time per vehicle with and without CityDrive, respectively.

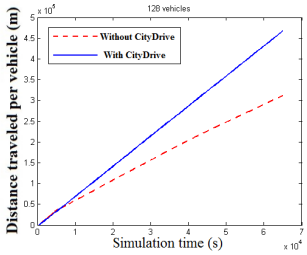



Fig. 15: The distance traveled per vehicle with and without our system.

### B. Influence of the number of vehicles using our system

To find out the impact of reducing the number of vehicles using CityDrive, we simulate 128, 64, 32 vehicles with the same initial condition. Intuitively, the more vehicles using our system, the more acceleration data to calibrate the traffic signal time prediction. So we record the length of time of the four phases in 16 intersections after each calibration, and calculate the phase duration error. Results in Table V show that there is only a slight increase in phase length error if the number of vehicles drops.

Apart from the above difference, we also find out the differences of average waiting time and the number of green-

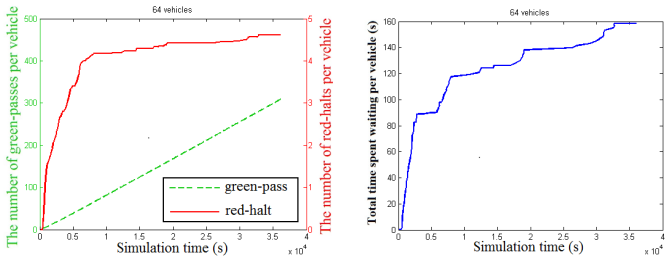
TABLE V: The influence of vehicle number on phase length error.

| Number of vehicles | 128 | 64 | 32 |
|---|---|---|---|
| Mean error(s) | 0.9804 | 1.1080 | 1.2380 |
| Standard deviation(s) | 0.7592 | 1.1406 | 1.1220 |

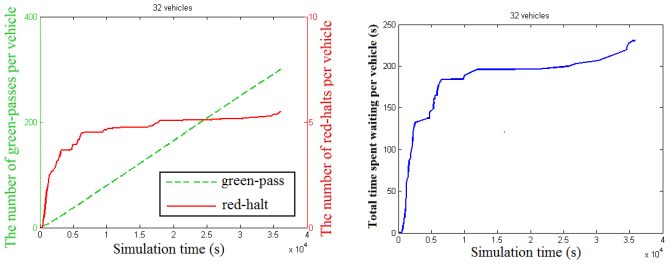
TABLE VI: The influence of vehicle number on waiting time and on the number of green-passes and red-halts.

| Number of vehicles | 128 | 64 | 32 |
|---|---|---|---|
| Waiting time per vehicle(s) | 127 | 158 | 230 |
| Number of green-passes per vehicle | 311 | 309 | 300 |
| Number of red-halts per vehicle | 3.8 | 4.6 | 5.5 |

passes and red-halts. The case of 128 vehicles is shown in Figure 14(a) and Figure 13(a), and the cases of 64 and 32 vehicles are shown in Figure 16. The comparison is in Table VI. There is indeed a slight increase in waiting time and the number of red-halts from 64-vehicle case to 32-vehicle case, while the number of green-passes is relatively stable.




(a) 64 vehicles




(b) 32 vehicles

Fig. 16: The influence of vehicle number on waiting time and on the number of green-passes and red-halts.

To gain information about speed distribution and fuel saving using our system, we carry out a real-world experiment and evaluate our system performance in the next section.

## V. Real world experiment

We target 14 intersections with 24 road segments in ZiZhu Hi-Tech Industrial Development Zone, Shanghai, China. We first test state merging and loop chain formation of individual intersections. Then we deploy a diving test and record speed and acceleration data to analyse.

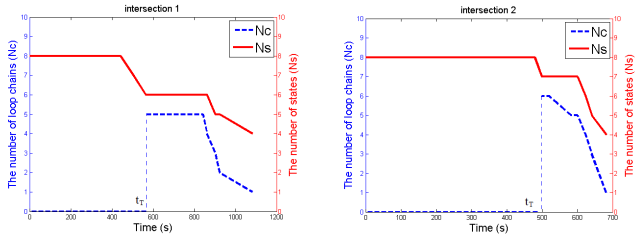
### A. Individual intersection test

To evaluate the performance of state merging and phase inference, we experiment 10 four-leg intersections. At the initial phase of establishing traffic signal schedule, the signal timing information is not yet available to drivers. Thus, we deploy three vehicles to repeatedly approach the tested intersection without speed-advisory service. The number of states ($N_s$) and the number of valid loop chains ($N_c$) are recorded. Two typical four-leg intersections with different traffic signal schedules are presented, whose phase numbers are shown in Table VII. The pair $[a, b]$ in $(i, j)$ means that the movement $(O_i, I_j)$ in the first and second intersection is in the phase $a$, $b$, respectively.

TABLE VII: In-out branch table of two intersections.

| | $I_1$ | $I_2$ | $I_3$ | $I_4$ |
|---|---|---|---|---|
| $O_1$ | \ | \ | [1,1] | [4,3] |
| $O_2$ | [2,2] | \ | \ | [3,3] |
| $O_3$ | [1,1] | [4,4] | \ | \ |
| $O_4$ | \ | [3,4] | [2,2] | \ |

The process of state merging and loop chain formation of the two intersections is shown in Figure 17.




(a) (b)

Fig. 17: The process of state merging and loop chain formation.

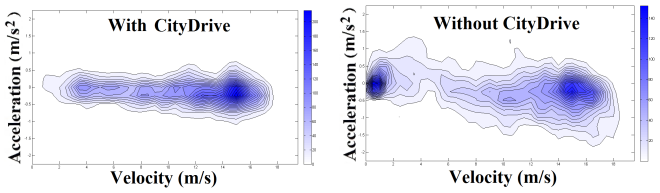
The calculation of traffic signal cycle length $T$ takes place at $t_T$, after which the number of loop chains ($N_c$) is determined. The state number $N_s$ decreases to 4 through state merging, and $N_c$ finally decreases to 1 by merging of loop chains. This means that the phase inference is successful.

### B. Driving test

Due to limited number of vehicles, we use 3 vehicles and 21 bicycles. Those bicyclers use a modified system application that sets the speed limit to 18 km/h. And the system application on three real vehicles has a speed limit of 60 km/h. We only gather speed and acceleration data from three real vehicles, while the acceleration events from bicyclers at intersections still help the server to infer traffic signal schedule. The three vehicles start travelling after the traffic signal schedules of all intersections are available. They repeatedly choose a destination in a random manner and run for about 2 hours, with a total travel distance of 180 kilometers. The GPS speed information and acceleration data are recorded, and the speed-acceleration histogram is shown in Figure 18(a).

In comparison, three vehicles without CityDrive have also run for the same total distance along random routes. The speed-acceleration histogram without CityDrive is shown in Figure 18(b).

The speed-acceleration histograms show that without CityDrive, drivers are prone to assume a maximum speed. But such driving habits incur increased probability of a complete halt at red lights. Thus the number of samples around zero speed is large, with significant acceleration measurements. Also, deceleration at high speed is obvious in Figure 18(b), probably because of the traffic light transition from green to




(a) With CityDrive (b) Without CityDrive

Fig. 18: Comparison of speed-acceleration histogram with and without CityDrive.

red when the drivers are arriving at intersections. Such sudden deceleration is usually lager in magnitude than acceleration, and is dangerous to vehicles behind. However, with CityDrive, such acceleration and deceleration is much less frequent. The number of samples around zero speed is significantly reduced, and acceleration readings are less in magnitude. Also, the speed is more evenly distributed along the speed axis, and excessive speed is avoided.

In addition, the acceleration module record the number of acceleration from zero speed. And velocity information from traces of GPS data is used to calculate the energy consumption. The calculation of kinetic energy is according to the formula $E_k = \frac{1}{2}mv^2$. Let $m = 1$ for simplicity, and the comparison of energy consumption per vehicle per kilometer and the number of acceleration from zero speed per vehicle per kilometer is in Table VIII.

TABLE VIII: Comparison of energy consumption and number of acceleration from zero speed per vehicle per kilometer.

| | With CityDrive | Without CityDrive |
|---|---|---|
| Energy consumption | 13.9 | 33.7 |
| Number of acceleration | 0.09 | 1.67 |

Finally, the results show that with our speed-advisory service throughout the travel, $58.8\%$ of kinetic energy can be saved. And the number of red-halts is significantly reduced.

## VI. Conclusion

In this paper, we have devised and implemented CityDrive, a software service that utilizes a collection of smartphone sensor and GPS data to continuously provide advisory speed for drivers. The server of CityDrive takes advantage of history signal transition data to calculate real-time traffic signal timing accurately, achieving an equilibrium in which most vehicles don't encounter red lights. Such speed-advisory service significantly reduces energy consumption and the number of complete halts.

But CityDrive requires a large proportion of running vehicles to use our system, and it may not work if the GPS signal is blocked by dense high buildings. Also, to avoid too much load on a single server, the central server system should be distributed to different regions.

Future work should figure out a way to identify flyovers, and to properly assign road lanes to vehicles moving with different speeds. In addition, incentives to use our service has to be explored, and proper punish mechanism is needed if drivers run the red light. Finally, we hope that this work will motivate further research in intelligent transportation system.

## VII. Acknowledgement

This work is supported by NSF China (No.61325012, 61271219, 61221001, 61202373); SRF for ROCS by SEM; Shanghai Basic Research Key Project (No.13510711300, 12JC1405200, 11JC1405100); China Ministry of Education New Century Excellent Talent (No.NCET-10-0580); China Postdoctoral Science Foundation Grant (2012T50417); National Key Project of China (No. 2012ZX03001009); Doctoral Fund of Ministry of Education of China (No.13005160); SEU SKL project (No.2012D13) and Open Foundation of State Key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications) (No.SKLNST-2013-1-16).

## References

[1] T. T. Institute, 2012 Annual Urban Mobility Report. Available: http://mobility.tamu.edu/ums/.
[2] N. Shah, F. B. Bastani, I.-L. Yen, "A Real-Time Scheduling Based Framework for Traffic Coordination Systems," in *Proc. IEEE SUTC*, 2006.
[3] K. Pandi, D. Ghosal, H. M. Zhang, C.-N. Chuah, "Adaptive Traffic Signal Control With Vehicular Ad hoc Networks," in *Proc. IEEE TVT*, vol. 62, no. 4, pp. 1459-1471, May 2013.
[4] C. Priemer, B. Friedrich, "A Decentralized Adaptive Traffic Signal Control Using V2I Communication Data," in *Proc. IEEE ITSC*, St. Louis, MO, USA, October 3-7, 2009.
[5] V. Gradinescu, C. Gorgorin, R. Diaconescu, V. Cristea, L. Iftode, "Adaptive Traffic Lights Using Car-to-Car Communication," in *Vehicular Technology Conference*, 2007. VTC 2007-Spring. IEEE 65th, pp. 21-25.
[6] Akcelik R., Besley M., and E. Chung, "An evaluation of SCATS master isolated control," in *Proc. 19th ARRB Transport Research Conference (Transport 98)*, pp. 1-24.
[7] N. Hounsell, J. Landles, R. D. Bretherton, and K. Gardener, "Intelligent systems for priority at traffic signals in London: The INCOME project," in *Road Transport Information and Control, 1998. 9th International Conference on (Conf. Publ. No. 454)*. IET, 1998: 90-94.
[8] Head, K. Larry, Pitu B. Mirchandani, and Dennis Sheppard, "Hierarchical framework for real-time traffic control," No. 1360. 1992.
[9] O. Servin, K. Boriboonsomsin, M. Barth, "An Energy and Emissions Impact Evaluation of Intelligent Speed Adaptation," in *Proc. IEEE ITSC'06*, Toronto, Canada, September 17-20, 2006.
[10] K. Perera, D. Dias, "An Intelligent Driver Guidance Tool using Location Based Services," in *Proc. IEEE ICSDM'11*, June 29-July 1, 2011.
[11] M. Krause, K. Bengler, "Traffic Light Assistant - Driven in a Simulator," in *Proc. IEEE IV'12*, 2012.
[12] E. Koukoumidis, L.-S. Peh, and M. R. Martonosi, "SignalGuru: leveraging mobile phones for collaborative traffic signal schedule advisory," in *Proc. MobiSys'11*, New York, NY, USA: ACM, 2011, pp. 127-140.
[13] G. Mahler, A. Vahidi, "Reducing idling at red lights based on probabilistic prediction of traffic signal timings," in *American Control Conference (ACC)*, 2012 (pp. 6557-6562).
[14] M. Kerper, C. Wewetzer, A. Sasse, M. Mauve, "Learning Traffic Light Phase Schedules from Velocity Profiles in the Cloud," in NTMS'12, pp. 1-5, 2012.
[15] P. Mohan, V. N. Padmanabhan, R. Ramjee, "Nericell: Rich Monitoring of Road and Traffic Conditions using Mobile Smartphones," in *Proc. ACM SenSys*, pp. 323-336, 2008.
[16] S. Schroedl, K. Wagstaff, S. Rogers, P. Langley, C. Wilson, "Mining GPS traces for map refinement," in *Data mining and knowledge Discovery*, 2004, 9(1): 59-87.
[17] Y.C. Chung, J.M. Wang, S.W. Chen, "A vision-based traffic light detection system at intersections," in *Journal of Taiwan Normal University: Mathematics, Science and Technology*, Vol. 47(1), pp. 67-86, 2002.
[18] J. Gong, Y. Jiang, G. Xiong, C. Guan, G. Tao, H. Chen, "The recognition and tracking of traffic lights based on color segmentation and camshift for intelligent vehicles," in *IEEE IV'10*, pp. 431-435, 2010.
[19] Tae-Hyun, Hwang, Joo In-Hak, and Cho Seong-Ik, "Detection of traffic lights for vision-based car navigation system", in *Advances in Image and Video Technology*, Springer Berlin Heidelberg, pp. 682-691, 2006.
[20] M. Omachi, S. Omachi, "Traffic light detection with color and edge information," in *IEEE ICCSIT'09*, pp. 284-287, 2009.
[21] G. Ning, and L. Cai, "Adaptive Driving Speed Guiding to Avoid Red Traffic Lights," in *Proc. 2nd International Conference on Computer Science and Electronics Engineering*, 2013.
[22] D. Comaniciu, P. Meer, "Mean shift analysis and applications," in *IEEE Computer Vision*, Vol. 2, pp. 1197-1203, 1999.
[23] L. Sun, W. Wang, "On latency distribution and scaling: from finite to large Cognitive Radio Networks under general mobility," in *Proc. IEEE Infocom*, Orlando, FL, March 2012.
[24] Zheng Yang, Yunhao Liu, "Understanding Node Localizability of Wireless Ad-hoc Networks," *Mobile Computing, IEEE Transactions on 11.8 (2012),* pp. 1249-1260.
[25] G. Cai, B.M. Chen, T.H. Lee, "Unmanned rotorcraft systems," *Springer*, 2011.