TrafficTurk: a smartphone based turning movement counter for monitoring extreme congestion events

Mauricio Carvajal
Mauricio Carvajal Industrial Design, Bogotá, Colombia

Brian Donovan
Department of Civil and Environmental Engineering, University of Illinois at Urbana-Champaign

Mostafa Reisi Gahrooei
Department of Civil and Environmental Engineering, University of Illinois at Urbana-Champaign

Sudeep Gowrishankar
Department of Mechanical Science and Engineering, University of Illinois at Urbana-Champaign

Meng Han
Department of Civil and Environmental Engineering, University of Illinois at Urbana-Champaign

Jonathan Que
Department of Civil and Environmental Engineering, University of Illinois at Urbana-Champaign

Bhinav B. Sura
Department of Computer Science, University of Illinois at Urbana-Champaign

Camilo Vega
Department of Civil and Environmental Engineering, University of Illinois at Urbana-Champaign

Dan Work (corresponding author)
Department of Civil and Environmental Engineering and Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
dbwork@illinois.edu

5461 words + 7 figures + 1 table
ABSTRACT
This article describes TrafficTurk, a smartphone based turning movement counter developed on Android and free for download on Google Play\(^1\). Users of the application count vehicles by swiping gestures on the phone screen, and the data is streamed to a back-end processing engine in real-time. Because of the ubiquity of smartphones, TrafficTurk enables the quick and low cost deployment of temporary traffic sensors. To illustrate some of the potential use cases, TrafficTurk has been deployed on several extreme congestion events, such as a 100-sensor experiment for a football game, and an emergency deployment after Hurricane Sandy. This article summarizes the preliminary application design, supporting infrastructure, algorithms, and experiments to date, as well as highlighting future directions of the project.

\(^{1}\)a preliminary version (functional only at intersections in Urbana–Champaign), can be downloaded from the TrafficTurk website at: http://trafficturk.com/
INTRODUCTION

Need for Sensing During Extreme Congestion Events

The long term goal of this work is to develop a traffic monitoring platform to enhance traffic information during extreme congestion events, such as sporting events, political events, and natural disasters. In the US, the Federal Highway Administration estimates that pre-planned special events are responsible for 93–187 million hours of traffic delay annually, with direct costs ranging between $1.7 and $3.4 billion. Despite the existence of more than 24,000 large events drawing an estimated 600 million attendees annually (3), understanding and managing traffic during these events has not seen widespread attention by researchers. This is in part due to sensing limitations that prevent widespread monitoring of the traffic generated by these events.

With a few exceptions, dedicated traffic sensing infrastructure is extremely limited, especially on surface streets. To overcome this data deficiency, several researchers have proposed to use GPS data from smartphones, fleet vehicles, and personal navigation devices. Because this data is often sparse in time and space, statistical models (2, 6, 7) of traffic have been developed to predict real–time traffic conditions.

However, because the statistical models rely on historical priors when data rates are low, their performance may deteriorate significantly during extreme congestion events. This is because extreme congestion events may change the network topology (e.g. due to planned road closures, or unplanned infrastructure failures), change travel demands (e.g. spikes in numbers of trips near sporting venues, storm evacuations, etc.), and change traffic control devices (e.g. restrictions on travel, overriding traffic signal timings by traffic control police officers).

Motivation for the Development of TrafficTurk

To improve traffic monitoring during extreme congestion events, this article introduces a new approach to quickly and cheaply add temporary traffic sensors on surface streets. The application, known as TrafficTurk (13), is a turning movement counter implemented on a smartphone. The name of the application is inspired by the 18th century human–based chess playing “machine” (and later the human based web service from Amazon) called the Mechanical Turk (11), which was later revealed to be a human–chess player hidden inside the “machine”. In the same way, TrafficTurk leverages humans to provide streaming traffic data during extreme congestion events. The main motivation behind using humans to manually count traffic is twofold. First, most conventional traffic sensors have moderate to high installation costs. Because extreme congestion events may cause traffic congestion in areas that are typically uncongested, it is not often economically feasible to install a large number of sensors to monitor traffic during the event. Because the TrafficTurk application is free and on a common smartphone platform, the only cost of collecting data is the labor cost of the human counter. In our experience, $15-20/hr is sufficient to quickly recruit and train counters to collect data during events. The counters bring their own cellphones and use their own data plans, so the total cost of deployment is quite low.

The second reason for using humans to collect traffic data is based on a future plan to collect counts on vehicles, pedestrians, and bicyclists. Humans can easily distinguish between the various modes of transport, and TrafficTurk may eventually support multi-touch swiping (e.g. one finger swiping for a car, two for a bike, etc.) to encode counts across multiple modes. This may prove useful in areas where the traffic flows are highly heterogeneous. Again, this is a future direction of the project, and an aspect we have not yet explored in our current work.

To illustrate the potential of TrafficTurk, we have deployed the application during several
extreme congestion events. In the first experiment, we recruited more than 100 volunteers to collect traffic data for the 2012 Illinois–Indiana homecoming football game, which generated significant traffic in Urbana–Champaign, IL. For the three hours leading up to the game, volunteers using the TrafficTurk application temporarily turned the Urbana–Champaign community (and home of our University), into one of the most densely instrumented cities in the country. Although we originally planned TrafficTurk for use during pre-planned special events, a few days after our football game experiment, the TrafficTurk system was re-deployed in New York City, to measure traffic in the aftermath of Hurricane Sandy. This has also helped shape the design of the application to support future monitoring deployments in the aftermath of natural disasters.

Outline of the Article
The remainder of this article describes in detail the TrafficTurk smartphone application, as well as the backend computational infrastructure that supports it. First we describe some of the key design challenges both on the application side, as well as on the server side. Next, we provide an overview of some of the preliminary algorithms we have developed to process TrafficTurk data, and then our preliminary experiments are presented. Finally, we discuss some of the future directions of the application and the project.

TRAFFICTURK CLIENT APPLICATION
Overview of Existing Turning Movement Counters and their Limitations
The TrafficTurk application is an Android platform based turning movement counter (Figure 1a). The application is inspired by traditional turning movement counter boards, shown in Figure 1b. Traditionally, these devices have 12 or 16 counting buttons, one for each possible maneuver at a four-way intersection, and possibly one pedestrian button at each corner of the intersection. Users take the board to an intersection, input data, and offload the data later when it is connected to a PC. Then, it can be processed by software, for example to improve signal timings.

Unfortunately, traditional turning movement counters have several constraints that prevent them from being used for extreme congestion event monitoring. First, most boards do not have any wireless communication capabilities, which limits their applicability to real-time monitoring applications. Second, the boards can be expensive. A single board can cost over $300, which means it is very expensive to deploy them at multiple intersections simultaneously, and this limits the amount of data that can be collected.

The TrafficTurk application was designed to tackle the above mentioned issues, by enabling turning movement counts to be collected in real-time and over larger portions of the transportation network simultaneously. The development of a smartphone application is a natural direction due to the ever-growing ubiquity of smartphones and their inherent connectivity to the internet. The embedded communication capabilities allow for real-time data transfer to a centralized database. Moreover, excellent software development kits enable fast application development and debugging. The ubiquity of existing hardware that can run the application eliminates the need for expensive and rigid hardware solutions. Finally, the user-interface is adaptable, which enables custom counting screens for three-way intersections, one–way streets, etc., which can lead to improved counting accuracy at these intersections.
FIGURE 1: Turning movement counters: (a) TrafficTurk smartphone turning movement counter. User swipes (indicated by the dashed arrows) are recorded as left, right, and through movements. (b) Conventional turning movement counter

TrafficTurk Application Design Challenges
While the smartphone platform provides many hardware and communications benefits, it presents several new design challenges. Our first attempt at designing a smartphone based turning movement counter directly emulated the traditional turning movement counters, complete with a 12 button interface. However it was soon found that the interface was quite difficult to use due to the small size of smartphone screens, and the lack of a tactile response. Unlike dedicated boards, it was difficult to verify that the correct button was pressed, without looking at the phone screen.

Given that most smartphones are manufactured with high precision touchscreens, our second design focused on using swipes (gestures) on the screen to record turning movement counts. For example, by tracing a finger along a path indicated by the arrows in Figure 1a, the user is able to record each of the 12 movements, along with the timestamp of the count. Accurate swipe recognition is also a non-trivial problem, due to the wide variety of screen sizes manufactured on the Android platform. Moreover, the wide variety of hand sizes, combined with varying screen and resolution sizes, meant a reliable swipe recognizer needed to be developed (detailed later in this article). The result was a simple interface that allows fast and accurate data collection without the need to look at the screen.

A second design challenge was created by the desire to use the TrafficTurk for real-time monitoring applications. First, this means the application must communicate with a backend server regularly to transmit the latest counting information. Second, it means the application and the backend server must agree on exactly which intersection the application is recording counts. The simple solution to this problem was to have the server keep track of the network topology, and share with the application the locations where the phone may record counts. Once the phone receives a list of intersections at which counts can be collected (see Figure 2), the user selects the intersection, and the compass is used to determine the alignment of the counter, relative to the intersection.

Synchronization of this map and count information proved harder than one might expect, precisely because extreme congestion events also cause extreme congestion on the cell phone network infrastructure. In the case of a natural disaster, the network may be completely unavailable.
Designing an application to support synchronization of all information about intersections to display on the phone proved quite difficult, because of the wide range of communication network quality. Ultimately, we insisted that the application appear to be completely functional to users, regardless of the communication capabilities with the server. Moreover, the application is required to intelligently cache information about the network topology, and opportunistically transmit data to the server whenever a connection with the server can be established.

Because it is difficult to obtain information about traffic signal phases (8), and because signal phase timings are relevant for predicting how the system will respond to future traffic demands, we decided that TrafficTurk should also collect information on the traffic signal phases. In a preliminary version of the application, we created a separate data entry mode within the application, where users could directly record the traffic signal phase. This was problematic because it was impossible for users to simultaneously record vehicle counts and phase changes reliably. Instead, we made a design decision to estimate the signal phases directly from the turning movement counts, detailed later in this manuscript. In this way, we keep the application design simple, and we do not require the user to input any data that the application itself can deduce from the recorded data.

Another important aspect of the application design is the user interface. The application has been carefully designed to be as intuitive as possible. We attempt to present the smallest amount of written information, buttons, and interactions to the user at every step and provide feedback whenever necessary. The chosen aesthetics and graphic styles are designed to present information clearly and a game-like experience is provided to keep users engaged.

**TRAFFICTURK SYSTEM DESIGN**

In order to support all the features required by the application, a robust backend infrastructure is needed. This is accomplished with a server that is capable of managing connections from many users simultaneously, storing incoming traffic data (vehicle counts), and sending requested information (intersection names, number of streets at an intersection, one-way streets, etc.) to all users seamlessly. The server houses a database for long–term storage and retrieval of user generated data as well as map and network topology data. The client-side application is designed to serve as a convenient mechanism for users to access and create this data.
TrafficTurk Communication Protocol Design

The range of intended deployment environments for TrafficTurk creates some unique design challenges. One of these challenges is to be able to collect and transmit data reliably on a smartphone regardless of the immediate cellular network connectivity. This is because extreme congestion events typically also overload the cellular networks due to large demands on the network.

The solution to the intermittent connectivity issues was a carefully designed custom communication protocol. This protocol uses TCP sockets and a system of acknowledgements (ACK) to ensure data quality and fast transfer of data when a good communication network exists, and guaranteed eventual data transmission even when internet connections are unreliable. The communication protocol design is described in Figure 3. An outline of the protocol is as follows:

1. The server constantly listens for new clients (smartphone app) to connect, creating a new Connection Thread every time this occurs. Multiple Connection Threads can run in parallel on the server, and this ensures that it can efficiently handle many users.

2. The Connection Threads constantly read Control Messages (CTRL) sent by the client application. These messages are a way for the client application to request map data, or inform the server that subsequent messages will contain vehicle counts.

3. If the server has not heard from the client for some pre-specified amount of time, it closes the Connection Thread. If the Connection Thread is closed in error (due to poor network connectivity), the client will attempt to reconnect to the server. At this time, a new Connection Thread is created.

4. On the client side, the Update Thread is responsible for periodically sending turning movement counts to the server. It is constantly running when there is an active connection to the server. When this communication pipeline fails (the app cannot receive any messages in a timely manner), the Retry Connection Thread is created on the client. This thread periodically attempts to reestablish a connection with the server.

5. Regardless of the connectivity status of the communication protocol, the application’s User Interface Thread (UI) runs continuously, and allows the user to collect traffic data. This data is stored by the application locally in a persistent cache. Every 30 seconds, the client attempts to send the cached data to the server in small bundles. Data is only deleted from the phone cache once the client receives an acknowledgement from the server that the data has been received. In other words, this ACK serves as a guarantee that the server has received a given bundle of data and has stored it in the database. In case the ACK is not received, the data will continue to persist on the mobile phone, and it will continue attempts to send the data. In this way, we ensure that data is never lost, even if the connection is unreliable. A number of additional optimizations prevent excessive communication attempts, and allow for energy efficient data transmission.

TrafficTurk Map and Database Design

Another design challenge that was encountered was that the system had to be robust to changes in the map data. TrafficTurk’s internal representation of the map is based on the freely available OpenStreetMap (OSM) data (9), which allows users across the world to edit and improve maps
FIGURE 3: Communication protocol between the server and the smartphone client.

continuously. This data is free to use and share, it is simple to download, and it is constantly improving.

Identifying and fixing errors in the map is a continuous process, which is distributed across the many users of OSM. However, propagating the latest changes of the OSM map into our system poses the potential problem of invalidating data collected on older versions of the map that have since been updated. To overcome this, our system adapts by storing all previous map versions simultaneously. Thus, we have the ability to display the map as it appeared at any point in time, so that no data is lost. Only the latest version of the map is sent to the TrafficTurk application. This solution allows for a constantly evolving map that does not require explicit manual maintenance within our system, while avoiding the need for manual data roll-over every time the map changes.

Database Structure
This section outlines the database structure used to store map data and user-generated turning movement counts. To store our map and traffic data, we use an object-relational database called PostGres (10). PostGres allows the use of various types of indices for specialized data search and retrieval. For example, GiST indices allow us to retrieve information from small regions of a map without having to search through all of the map data.

Figure 4 visualizes the relationships between the tables of the database. As mentioned earlier, we use map data available on OpenStreetMap and process it into a form that is useful for TrafficTurk. The traffic network on the map is encoded as Nodes and Links, where Nodes represent individual intersections and Links represent stretches of road between intersections. We record the spatial information related to these Nodes and Links in terms of latitude and longitude. We also keep a record of the time range when a Node or Link existed in a particular form in our map. Each Link references two Nodes - the node it originates from and the node it ends at. Two-way streets are encoded as two separate Links facing in opposite directions.

To store the turning movement counts, we use a combination of the Maneuver table and the ManeuverCount table. Each row in the Maneuver table describes a possible maneuver, or in
FIGURE 4: Summary of the tables and relationships in the database. Arrows denote a foreign key relationship between two tables.

TABLE 1: An example of the ManeuverCount table

<table>
<thead>
<tr>
<th>maneuver_count_id</th>
<th>maneuver_id</th>
<th>node_id</th>
<th>count_timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>23</td>
<td>38140494</td>
<td>1374792521713</td>
</tr>
<tr>
<td>25</td>
<td>17</td>
<td>38140494</td>
<td>1374792521913</td>
</tr>
<tr>
<td>26</td>
<td>23</td>
<td>38140494</td>
<td>1374792522093</td>
</tr>
<tr>
<td>27</td>
<td>12</td>
<td>38140494</td>
<td>1374792522446</td>
</tr>
<tr>
<td>28</td>
<td>39</td>
<td>38014831</td>
<td>1374793434915</td>
</tr>
<tr>
<td>29</td>
<td>39</td>
<td>38014831</td>
<td>1374793435353</td>
</tr>
<tr>
<td>30</td>
<td>39</td>
<td>38014831</td>
<td>1374793436808</td>
</tr>
</tbody>
</table>

other words, a movement that a vehicle may make at an intersection (northbound straight-through, southbound right-turn, etc.). Each maneuver is encoded with references to the Link that it started from, the Node it passed through, and the Link it went to. In the ManeuverCount table, we store each maneuver that was recorded by a user of the application, i.e. each row in the ManeuverCount table corresponds to a swipe made by a user. The time of each swipe is recorded as the timestamp, the Node at which the data was collected is identified with a unique node_id and each maneuver in the database is given a unique maneuver_id. Table 1 shows an example of a small portion of the ManeuverCount table.

ALGORITHMS
Several algorithms have been developed to improve functionality of the TrafficTurk application, as well as to help generate meaningful data for future real-time traffic estimation algorithms, currently under development. This section outlines three algorithms currently implemented within our system. First, we explain how we solve the swipe recognition problem outlined earlier. Next, we provide an overview of the phase estimation algorithm used to recover the traffic signal phases from turning movement counter data. Finally, we summarize our work on estimating the control logic implemented at the intersection from TrafficTurk, using a technique known as inverse optimal control.
**Swipe Recognition**

One of the requirements of *TrafficTurk* is a highly accurate engine to recognize and correctly classify gestures as vehicle maneuvers. Given our prior failures with button based interfaces due to the lack of tactile feedback provided by smartphones, we decided to develop an accurate swipe recognizer. While the Android platform has an *Application Programming Interface* (API) to recognize many common gestures on the device (e.g. up-swipes and down-swipes can be recognized for scrolling web pages in a mobile browser), the current APIs do not support recognition of the complete gesture set (Figure 5) required to make a functional turning movement counter. Thus, a new swipe classifier based on a hierarchical *Support Vector Machine* (SVM) algorithm was designed and implemented within *TrafficTurk*.

![Figure 5: Possible maneuvers in TrafficTurk](image)

SVMs are a common approach to solve classification problems in a wide range of fields (1). The approach is a supervised learning method to classify a set of data points into one of two classes. Training the SVM requires defining a separating hyperplane that partitions the training data in such a way that the distance between the hyperplane and the nearest data points in each class is maximized. Classifying a data point with an SVM simply requires checking on which side of the hyperplane the data point is located.

In our implementation, we use four classification layers to classify each swipe into one of the twelve maneuvers, since the SVM algorithm is only a binary classifier. We use 4000 recorded swipe trajectories that were manually labeled (set of coordinates recorded by the phone’s touchscreen) to train the SVMs. The hierarchy of the classifier is shown in Figure 6. The numbers in the boxes are the maneuver numbers depicted in Figure 5. In the first and second layer, we examine the difference between the starting and ending coordinates on the swipe to determine the direction of the swipes in the horizontal and vertical directions. These two checks easily separate left–right maneuvers from right–left maneuvers and up–down maneuvers from down–up maneuvers. Some swipes, such as maneuver 5, may end up classified as down–up maneuvers, but can either point to the left or the right. Thus, it appears in multiple classes, and needs to be distinguished at lower levels of the tree.

In the third layer, we separate turning movements from through movements. Using information about the angle of the vector connecting the start and end coordinates of each swipe, an SVM was trained to effectively separate the turns from the through movements. Finally, the fourth layer separates two types of turns and also two types of through movements. This is achieved by training an SVM on a 4-dimensional feature composed of starting and ending coordinates. Al-
though this approach appears complex, it achieves above 99% percent accuracy on the labeled data and works quite well in experimental tests.

**Signal Phase Estimation**

One of the goals of the TrafficTurk project is to collect sufficient data to estimate traffic flows on a road network. This requires not only the turning movement counts, but also knowledge of traffic signal phases. This can help distinguish between the cases when the flow is zero because the light is red, from the case when the flow is zero because the light is green, but the road is empty. Although signal phasing information can be obtained from traffic agencies, obtaining the information can be quite difficult logistically. Additionally, collecting the signal phasing information along with turning movement counts on the TrafficTurk application is not viable from a usability perspective.

Considering these requirements and challenges, we decided to extract traffic signal phasing information directly from the turning movement counts. In order to do this, we model traffic flow through a signalized intersection as a Hidden Markov model (HMM). In this model, the turning movement counts are observations from different hidden states (phases), which are to be inferred. Using a learning and inference algorithm detailed in (4), we can infer signal phases at an intersection from the maneuver counts with high accuracy.

As a matter of illustration, we ran the algorithm described in (4) on a dataset (a sequence of vehicle maneuvers) collected by TrafficTurk application at the intersection of S. First St. and E. Springfield Ave., in Champaign, IL. This is a four-way intersection with an actuated traffic signal. Thus, the phase sequence depends on the traffic conditions. The data was collected over a 10 minute period during which 200 maneuvers were recorded.

Figure 7a depicts the distribution of the maneuvers, and Figure 7b shows the estimated and true phases that generated the maneuvers. Maneuvers are abbreviated as eastbound left (EBL) or eastbound through (EBT), with logical extensions to the various other movements. In Figure 7b Phase 1 allows all east–west maneuvers (EBT, EBR, EBL, WBT, WBR, WBL) and phase 2
FIGURE 7: Phase estimation using HMM: (a) Distribution of the maneuvers (b) Inferred phases.

allows all north–south maneuvers. Phase 3 and Phase 4 consist of EBL, WBL and NBL, SBL respectively. Phase 5 allows all maneuvers from west to east (EBT, EBR, EBL), and Phase 6 consists of all maneuvers from east to south. Phase 7 and Phase 8 allow all maneuvers from south to north and from north to south respectively.

As illustrated in Figure 7b, six of the 200 maneuvers are labeled with the incorrect phase. Among these, four of them appear at end of the phase and were labeled incorrectly because of the similarity of the next phase to the current phase (for example when switching from phase 8 to phase 2), and are difficult to distinguish because the maneuver is permitted in both phases. The other two errors (at 350 sec and 500 sec) are more critical because of the extra switches they introduce. However, these errors are easy to eliminate in a final naive post-processing algorithm.

Estimating Traffic Signal Controls With Inverse Optimal Control

The estimation of traffic controller strategies is an important problem when prediction or estimation of future traffic states is desired after the observation of the current state of traffic. The traffic controller strategies are an important part of the model of traffic evolution and have been traditionally hard to obtain in large scales. Additionally, since TrafficTurk is designed for extreme traffic events, we have encountered many intersections that are manually controlled by human traffic controllers.

Our method of estimation of traffic controller strategies using turning movement counter data uses the concept of inverse optimal control (IOC) (also called inverse reinforcement learning) to recover an objective function under which a traffic controller strategy is optimal. In other words, if we can find an objective function under which the traffic controller’s strategy is optimal, we can solve the optimal control problem with that objective function to mimic the traffic controller. We assume that we can create a linear combination of objective basis functions and use a convex optimization problem to solve for the weights. This objective function can then be used to solve the optimal control problem to predict how the future traffic will be controlled under the learned objective. Details of the algorithm can be found in (5).
EXPERIMENTS

Deployment at the 2012 Illinois–Indiana Homecoming Football Game

On October 27, 2012, more than a hundred students from the University of Illinois at Urbana-Champaign collected traffic data using the TrafficTurk application to monitor the atypical traffic congestion caused by the Illinois-Indiana Homecoming football game. Due to the changes in network topology (caused by road closures, manual control of traffic etc.), and increased travel demand, the collected data differs significantly from historical data in the region. In this 3-hour long experiment, more than 220,000 vehicles were counted with the average count of 852 veh/hr/intersection. The maximum number of counts appeared at W University Ave and N Lincoln Ave with 2088 veh/hr, which is a major intersection in the town.

One of the major challenges faced during this 3–hour experiment was the recruitment and training of more than one hundred students. We also had to organize the logistics in such a way that each volunteer collected data at a different intersection and knew how to reach that intersection. In order to do this, we included a feature in the application that allowed us to assign an intersection to each volunteer beforehand and used the capability of the Navigation application (in-built on most Android smartphones) to provide directions to their assigned intersection. Another challenge was to monitor the data streaming between the phones and our server. Using a real-time monitoring tool that displayed information about the incoming data as well as the state of each smartphone (remaining battery life, GPS location, etc.), we were able to quickly assess and predict any potential problems and solve them using a team of support staff that consisted of phone operators and bicycle riders. The support staff was able to resolve issues ranging from forgetting how to use the app (with phone support) to phones that were about to run out of battery (bicycle rider would swap a counter’s phone for a phone with a charged battery). Overall, this experiment proved to be the first successful large-scale network deployment of TrafficTurk.

Deployment Following 2012 Hurricane Sandy

On November 2, 2012, the TrafficTurk team traveled to New York City to deploy the traffic monitoring system in the aftermath of Hurricane Sandy. With traffic controllers not working, streets closed and chaotic traffic, it was the first real test of TrafficTurk in a disaster response setting. This test crystallized the issues of disaster zone traffic data collection to the team and helped spur further work to improve the system. Over the course of three days, out of which one was spent deploying the map, publicizing the need for volunteers to collect data and building an ad-hoc mission control center, the team collected over 30,000 traffic counts at nine different intersections. The counters were recruited on Columbia University’s campus in New York City and were sent to locations in the Garment district of the city where traffic conditions were atypical and severe.

Before this experiment, the application was only available to be used in Champaign-Urbana. However, the sudden and unexpected natural disaster required a completely new map deployment in New York City. Even though the map was freely available, the TrafficTurk database had to be populated with relevant intersections and streets, in order to build a sensor network. This spurred the team to re-evaluate the scalability and fast-deployment capability of the system and helped redesign the map to its current state.

Another lesson learned during this experiment was that phone network reception can be very poor or non-existent in some areas that are hit by natural disasters. This causes the real-time data transfer capability of TrafficTurk to not function. However, the team used this insight to design a robust data transfer protocol that allows data collection in the absence of a network and ensures
that all collected data is relayed to the server as soon as the phone receives an internet connection.

CONCLUSIONS AND FUTURE WORK
This article summarized the work to date on TrafficTurk, a smartphone based turning movement counter. While the application has been designed for the purpose of monitoring traffic during extreme congestion events, it could be easily used to replace conventional turning movement counters. To support this potential use case, we are currently extending the geographic areas where the application is functional. Because this requires loading OSM data on our server everywhere potential counts may be requested, some work remains to support this functionality. If the application is not functional in an area of interest to the reader, please contact the corresponding first author. We will gladly load the corresponding map to support users of the application. By January 2014, we hope to automate the deployment of the map in areas relevant to users of the application, directly within the application.

We are also working on the development of additional tools relevant for real–time monitoring applications (e.g. visualization, estimation tools, control algorithms, etc.), which is part of our future work. Moreover, we are actively seeking input from practitioners who would like additional data processing functionality within TrafficTurk (e.g. we are beginning to explore optimizing signal timings from TrafficTurk data).

References


