Mobile Applications for Cities

by

Mark J. Yen

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

May 2012

Author: _____
Department of Electrical Engineering and Computer Science
May 21, 2012

Certified by: _____
Gabriela Gomes, Lab Manager, SENSEable City Lab, on behalf of Carlo Ratti, Director, SENSEable City Lab, Thesis Supervisor May 21, 2012

Accepted by: _____
Prof. Dennis M. Freeman, Chairman, Masters of Engineering Thesis Committee

Abstract

A smartphone user today carries around in his or her pocket more computing power than all of NASA had when it sent a man to the moon in 1969 (Miller, 2012). These devices allow us to extend our sense of our surroundings in ways never before imaginable. With smartphone usage on the rise all around the world, there is room for new innovative mobile apps to improve how citizens and city officials go about their business. There are two areas in particular that have been identified as areas I am researching further.

The first pertains to the best way for motorists to find parking in cities, and then pay for it and refill it once they have arrived. We are developing a mobile app that will hopefully streamline this process for drivers, while replacing costly infrastructure for cities and reducing enforcement costs, and minimizing the amount of wasted time, frustration, and carbon emissions that come from drivers searching for parking spots.

The second applies to the growing amount of e---waste in the world, and the challenge of its proper disposal. By tagging trash with GPS sensors, we are able to better understand the process of its removal, discover where inefficiencies lie, and identify any improper handling along the removal chain.

Introduction

While cities make up only two percent of the Earth's crust, they account for 50 percent of the world's population, 75 percent of its energy consumption, and 80 percent of its carbon emissions. In today's society, cities are the hubs of innovation and constitute an interesting ecosystem of creativity and new ways of thinking. Their high

concentration of ideas and technology make cities the natural choice as a test tube for new research and new ways of doing things. A project that can make an impact on cities is a project that can make an impact on the world.

Another striking trend is the rapid adoption of smartphones, particularly across the U.S. and Europe. In May 2011, 35 percent of American adults owned a smartphone. By February 2012, that figure had jumped to nearly half (46 percent) of Americans. In fact, there are now more smartphone users than there are users of more basic feature phones according to a report by the Pew Internet & American Life Project (Smith, 2012). The rise of the smartphone has brought with it a dramatic shift in how we think about computing. Computing is no longer confined to the desk, but is now pervasive, accompanying us in our pockets and going with us wherever we go.

It is the intersection of these two trends that I am interested in. The rise of smartphones has turned our cities into data gold mines, with computing and sensing resources unimaginable ten years ago now abundant. There are many ways in which city life and city planning can be better informed through the creative utilization of mobile devices, but there are two areas in particular that I focused on.

The first area is parking. Cars today spend 95% of their lifetime parked, and the growing number of drivers is leading to a persistent shortage of spots, particularly in downtown areas. Furthermore, an average of 30% of traffic in central business districts around the world is due to drivers cruising for an open on-street parking spot (Shoup & American Planning Association., 2005). This is largely a problem of lack of real-time information. If drivers knew where the freest and cheapest areas to park were, they

would be able to get to their destination in less time and with less gas. Cities would benefit with fewer cars in traffic on the road, and with less CO_2 and other pollutants in the air. We are building a mobile application that we hope can eliminate this information asymmetry and make the parking process smoother for everyone. We also have some proposals on how we can use technologies such as license plate recognition in order to cut parking enforcement costs.

The second applies to the growing amount of e-waste in the world, and the challenge of its proper disposal. In particular, certain types of e-waste have been chronically mishandled. Though there are strict regulations in place regarding the proper procedures for recycling e-waste, many recyclers find it easier to illegally ship their waste off to third world countries that will pay them under the table. In these places, the waste is melted down (often in unsafe conditions), and the raw materials are sold as scrap. By tagging waste with GPS sensors, we are able to better understand the process of its removal, discover where inefficiencies lie, and identify any improper handling along the removal chain.

Smart parking systems

Parking meters were first introduced in the 1930s in order to prevent workers in downtown Oklahoma City from hogging all the street parking away from others who wished to do business there. Early parking meters were based on coin acceptors that would mechanically trigger a pointer to indicate how much time was remaining. This design remained unchanged for over 40 years, and the general principle of fixed coin-operated parking meters has persisted well into the 21st century.

Figure 1: A parking meter in 1940

The traditional parking meter leaves much to be desired. Many accept only quarters, sometimes leaving drivers in a tough situation where they can't leave their car without having paid, but having to leave their car to get change. Nearly all of them require you to return to the spot to refill the meter, interrupting the driver and forcing him or her to stay close to the car if there is a risk of running over. And finally, they are unwieldy and expensive to install, typically costing several hundred dollars per space to purchase and install.

And if you take a step back, parking meters have in many ways failed at their original purpose of encouraging turnover in downtown areas. Spaces are perpetually in

short supply, forcing drivers to circle endlessly, searching for an open spot. This affects urban quality of life by increasing traffic congestion, pollution and driving hazards, and contributing to the reduction of public space. According to a study by Donald Shoup of 15 city blocks in Westwood Village, Los Angeles, the average driver cruised for 3.3 minutes searching for parking. In a year, this results in 47,000 gallons of wasted gas, 950,000 wasted vehicle miles, and 730 tons of carbon dioxide in the atmosphere. An average of 30% of traffic in central business districts has been drivers cruising for a cheaper spot, with the figure reaching as high as 74% in some places (Shoup & American Planning Association., 2005). Additionally, the wasted time increases driver frustration and when drivers are paying attention to parking spots rather than the road, the risk of accidents also increases. On the city's side, the status quo in parking meters is not a rosy picture either. It costs a city $32,400 a year per meter technician and $36,090 a year per parking enforcement officer in wages and salaries, not including benefits (Bureau of Labor Statistics, 2011). This adds up to a cost of $6 million per year for enforcement and meter maintenance, assuming a city of approximately 20,000 parking spots.

Traditionally, the response to a lack of parking spots was answered with the construction of more parking lots. Guidelines were distributed to property developers, mandating enough spaces to sustain peaks in parking demand. However, since these peak conditions are reached infrequently, the parking lots are left unoccupied most of the time. As more and more parking lots appear, in many cases taking more space than the shops and businesses drivers are visiting in the first place, the pedestrian experience is worsened, leading to more cars on the road and even more parking requirements.

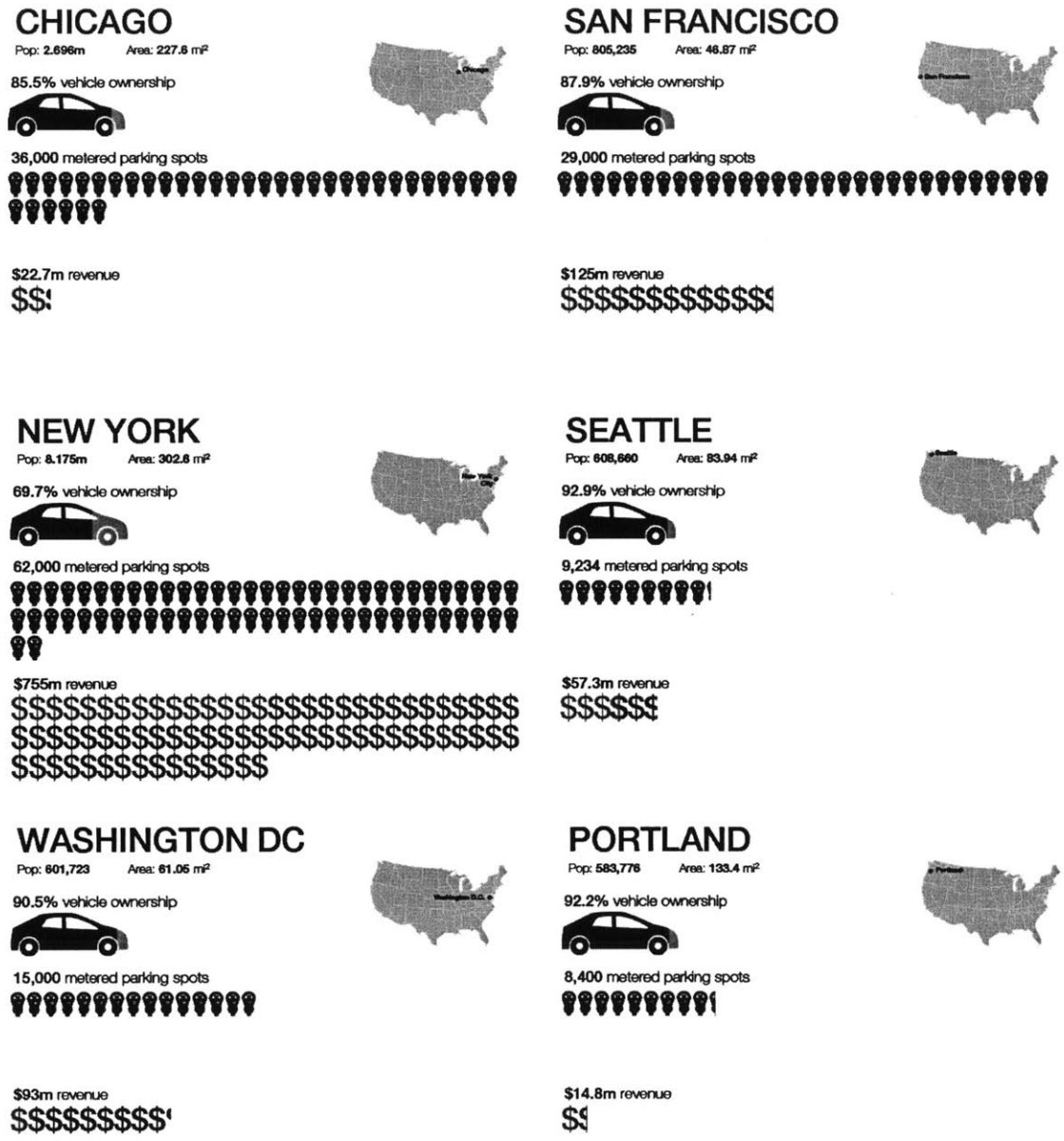


Figure 2: Parking statistics for selected cities

In recent times, there have been a number of systems that have attempted to mitigate some of the problems above. Portland, OR replaced 7,000 traditional meters with 1,130 multispace meters between 2005 and 2007 and saw its revenue jump by more than $2 million. In 2009, New York City trialed its "PARK Smart" program in

Greenwich Village. The program involved increasing meter rates by a dollar during the busiest part of the day, a very rough form of dynamic pricing. Dynamic pricing, or the practice of varying parking prices in response to real-time congestion data, has the promise of fixing many of the problems we see in parking today. Studies have found that drivers, particularly those driving for non-business purposes, are price sensitive enough that their behavior is influenced by changes in parking charges (Kelly & Clinch, 2006). Thus the price of parking at a spot can be the lever a city planner uses in order to effect changes in transportation choices and parking locations.

Perhaps most ambitious system to date, however, is the SFpark system in San Francisco. The project is comprised of several components: first, the meters in the project were revamped to accept credit card payments in additional to coins, a convenience that has been shown to typically increase revenues by 20–30% in other cities that have implemented this same change. The meters also have screens that are able to adjust their prices dynamically, something done once a month in San Francisco. Secondly, there is a sensor embedded in each space that is able to detect whether a car is present in each spot, making the system aware of precisely how many spaces are free on any block at any given time. This information is fed into the third component, a mobile application available for iOS and Android that displays parking availability and parking prices in real time.

Figure 3: SFpark meters

Despite the advances SFpark has brought, there are still several areas where it could be pushed further. At its base, the system is still based on paying at a meter, just like the original Oklahoma City system of the 1930s, and still carries along with it much of the unnecessary costs that come along with such a system. That there is a meter at every spot is not an insignificant cost, and adding in the sensor and the network connectivity makes it even less of one. Enforcement is still done in the traditional way, with parking enforcement workers checking each individual meter to see if it is expired. Presumably, this could be made easier by the sensors that detect the presence of a car, but seeing as the city retained 327 parking enforcement officers in 2011, it does not appear this advantage has been fully realized yet.

Apart from city-based initiatives, there are also several standalone apps, some of which work in several cities. They fall under three broad categories: parking directories, crowd-sourced data apps, and real-time parking apps. Parking directories, such as Primo

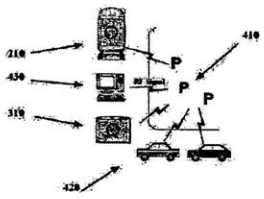
Spot and Park Place, allow users to find nearby on-street and garage parking, and view their opening hours and time limits. However, they don't allow users to determine whether spots are available or not at any given moment in time. Crowd-sourced data apps, such as SpotSwitch and Roadify, attempt to remedy this by prompting users to report when they are leaving a spot so that others can take it. Unfortunately, these apps are often unreliable due to low activity, since users are not incentivized to input information when they are leaving a spot. The last category of apps is real-time parking apps, which include ParkMe, Parker, and SFpark. These apps allow the user to view a map of all available spots and can find the closest free spot to the user's current location. But, since they are not linked with payment for any of the spots, they rely on in-ground sensors and other costly infrastructure in order to get their data. SFpark, for instance, cost $25 million to implement before it launched in April 2011.

Figure 4: Selected other parking apps

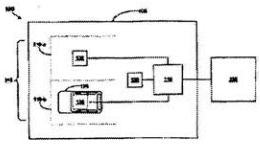
Also in the prior art are a number of patents related to parking and parking regulation systems. In 2006, Mehndelson described a "parking detector" embedded in each parking space, whose data could be transmitted to cell phones, PCs, and in-vehicle navigation systems. In 2005, Marin described a system of enforcing parking meters that involves a parking sensor to determine whether or not a space is occupied, and a vehicle identifier that can uniquely recognize vehicles in violation. A system for garage parking was patented in 2002 by Trajkovic et al. that identifies free spots using image processing and automatically directs drivers to the nearest free spot. Finally, Kulju et al. described

various methods of transmission of parking-related data, such as accrued parking fee and remaining time in 2004.

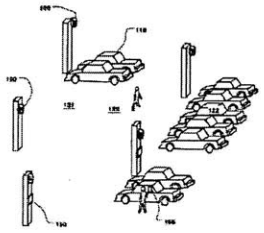
- **Parking Detector** (US 2006/0267799 A1)
- "Parking Detector" embedded in each space
- Detector data transmitted to cell phone, PC, and in-vehicle navigation systems
- Enables navigation to empty spaces, as well as back to your own parked vehicle



- **Automated Enforcement of Parking Meters** (US 2005/0068196 A1)
- "Parking sensor" detects whether or not a car is parked in each space
- "Vehicle identifier" captures an image of the license plate or VIN of vehicles parked too long
- "Violation manager" in each meter sends out notices of violation automatically



- **Smart Parking Advisor** (US 2002/6426708 B1)
- Uses a camera and image processing to determine which spots are free
- Provides a terminal at the entry gate that direct drivers to the nearest free spot



- **Transmission of parking-related data to user** (US 2004/0260712 A1)
- A parking fee register keeps maintains the parked state of all managed spaces
- User checks a mobile station for data such as accrued parking fee and remaining time in real time

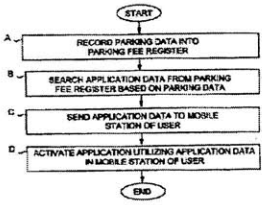


Figure 5: Selected related patents

The system that we propose has a number of advantages over existing ones. First, by replacing individual meters with multispace meters, we can demonstrate a savings in installation, maintenance, and downtime costs. Fewer meters means less to install and lower maintenance, with amount of maintenance decreasing from 8 hours per week to 2 hours per week in Dover, NH. Next, since all of the meters are linked, when one is broken, motorists can simply go down the street to use the next one.

	Traditional	Smart Parq
Meter operational costs	$200,000/year	$0/year
Meter maintenance costs	$610,000/year	$152,000/year
Enforcement personnel	$5.8m/year	$1.23m/year
Enforcement equipment	$266,000	$703,000

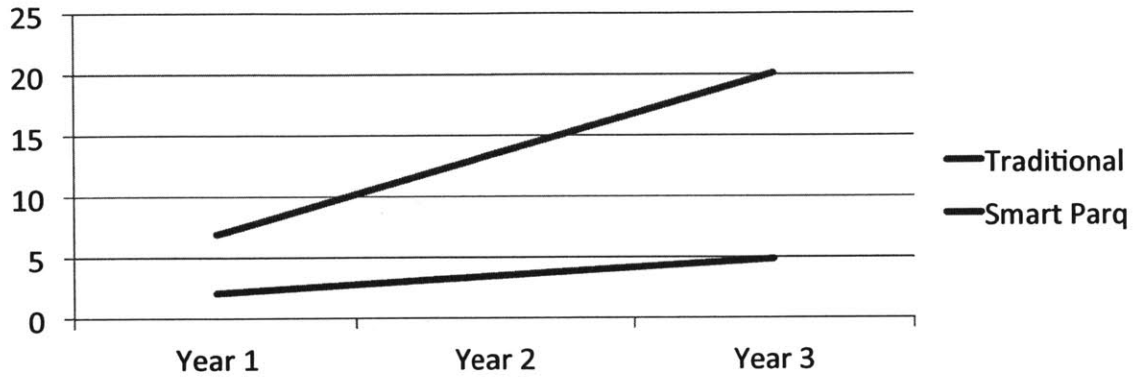


Figure 6: Projected savings for a city of 20,000 parking spaces

Next, we will collect users' license plate information when they sign up, so we can save cities millions of dollars per year on their enforcement costs. We propose using a technology known as License Plate Recognition (LPR). LPR is widely deployed in the UK (where it is known as ANPR, or Automated Number Plate Recognition), with some 50 million reads per day in 2006, and most likely more than double that number today. It was originally implemented in order to detect criminal movements on the roads, and is linked with a database of stolen number plates. By putting an LPR-enabled camera onboard a moving vehicle, enforcement capacity can be increased by as much as 40%, meaning enforcement costs can be cut dramatically. Furthermore, we plan to give users who arrive at a free spot only to discover a car parked there unpaid the ability to report it, further reducing the need for active enforcement patrols.

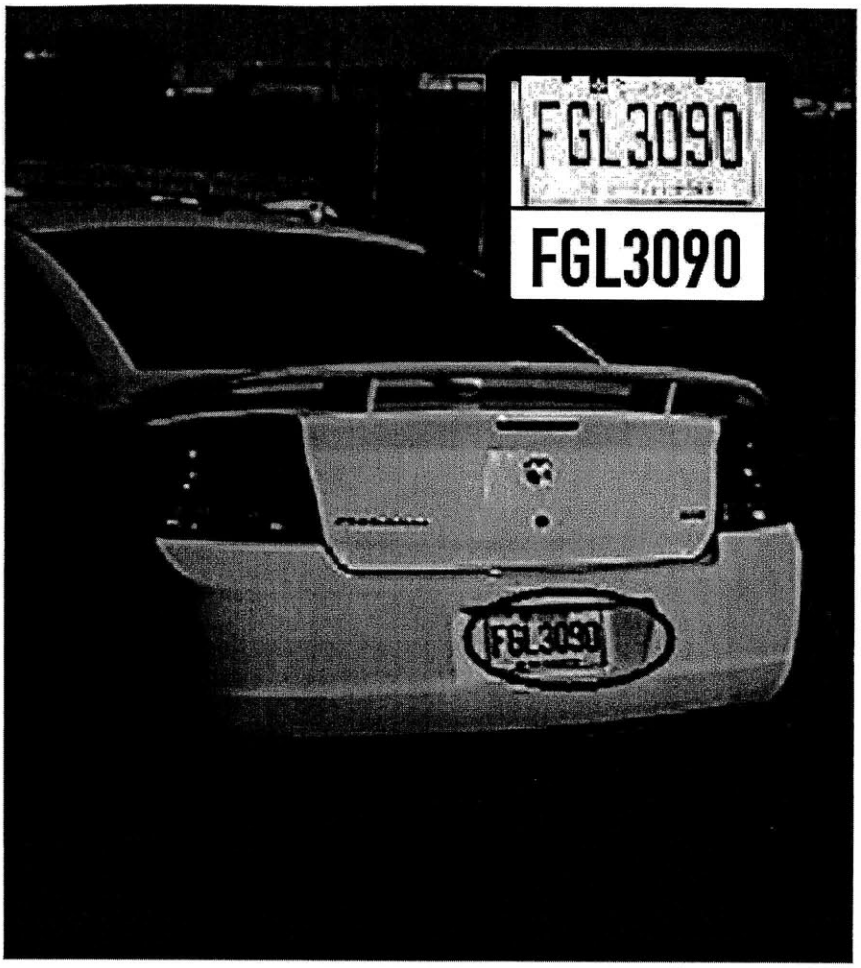


Figure 7: LPR license plate reading technology

We also have some ideas on how to make the system easier to use from the user's perspective. First of all, we will have a smartphone application that allows the user to find available parking, based on payment data that tells us whether a spot has been paid for or not. Based on the zoom level of the area the user is currently looking at, we display availability and price information either in a grid format or street by street.

Figure 8: User interface mockup showing parking availability in grid view (left) and street view (right)

We will also allow the user to pay for the parking directly from his or her phone with a credit card. The user will be notified remotely when time is about to expire, and can also extend the reservation remotely, without having to return to the car. Or, if the user is unsure about how long they will be parked, they can simply choose to have the meter count upwards and they will only pay for the amount of time they use. Finally, in situations where parking is likely to be in short supply, the user can put in a reservation ahead of time so that he or she knows there will be a spot waiting upon arrival.

Figure 9: Meter interface mockup in Pay As You Go mode before paying (left), and in Prepaid mode after paying (right)

We wanted to demonstrate that such a system with the design decisions we have made for enforcement and user experience will be feasible and will meet the needs of the cities we are designing it for. In order to accomplish this, we have created an agent-based simulation, which is made up of an environment and a number of agents, whose behavior is governed by a series of simple rules. Each agent, when spawned, has a destination it is trying to reach, and a duration for which it would like to park there. It also has a randomly assigned level of price sensitivity and threshold distance from the intended destination at which it will accept a spot. Under the traditional random model, an agent will drive towards its destination until it is able to find a spot within its price and distance guidelines. Under the model in which the user

has our app, he or she will know the location of the nearest spot and drive there directly. We hope to prove through this simulation that with better information, we can produce a better system in which drivers spend less time searching for parking, and the city generates more revenue from the more efficient use of available parking spots and a better distribution of cars throughout the city.

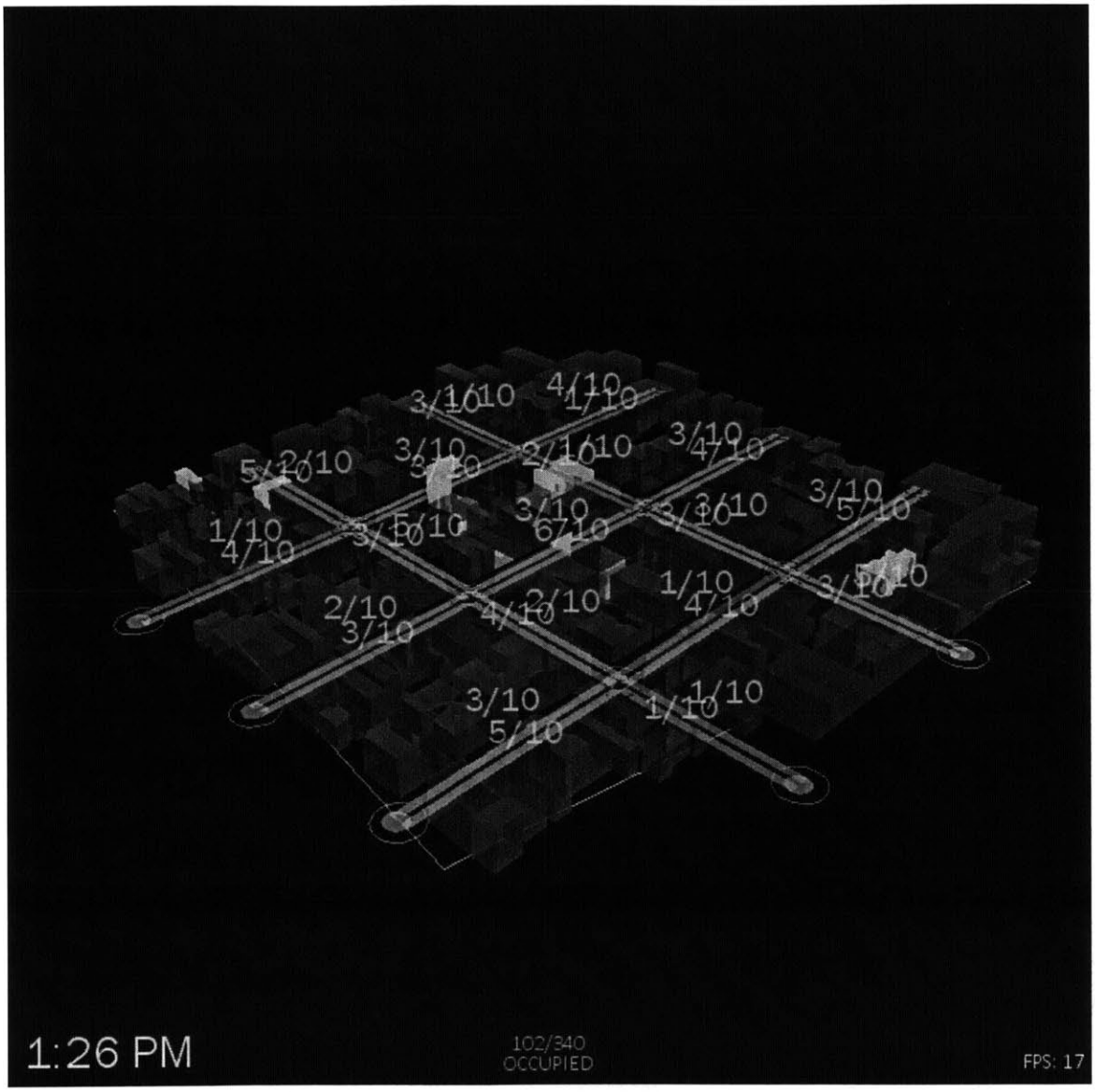


Figure 10: Parking simulation. Differently colored buildings represent different land uses and differently colored streets represent different levels of occupancy

We also wanted the system to be able to vary the price of parking at each block in an intelligent way. The ideal situation is one in which each block has one free spot on it remaining at all times. We attempt to stay as close to this ideal as possible by pricing the blocks of spots according to the following formula:

$$price = base\ price^{\%\ spots\ occupied * k}$$

where *k* is an adjustment factor based on how popular the block is likely to be (factoring in the centrality of the block, its proximity to points of interest, previous patterns, etc), and how popular the day is likely to be (factoring in weather, day of the week, holidays, special events, etc).

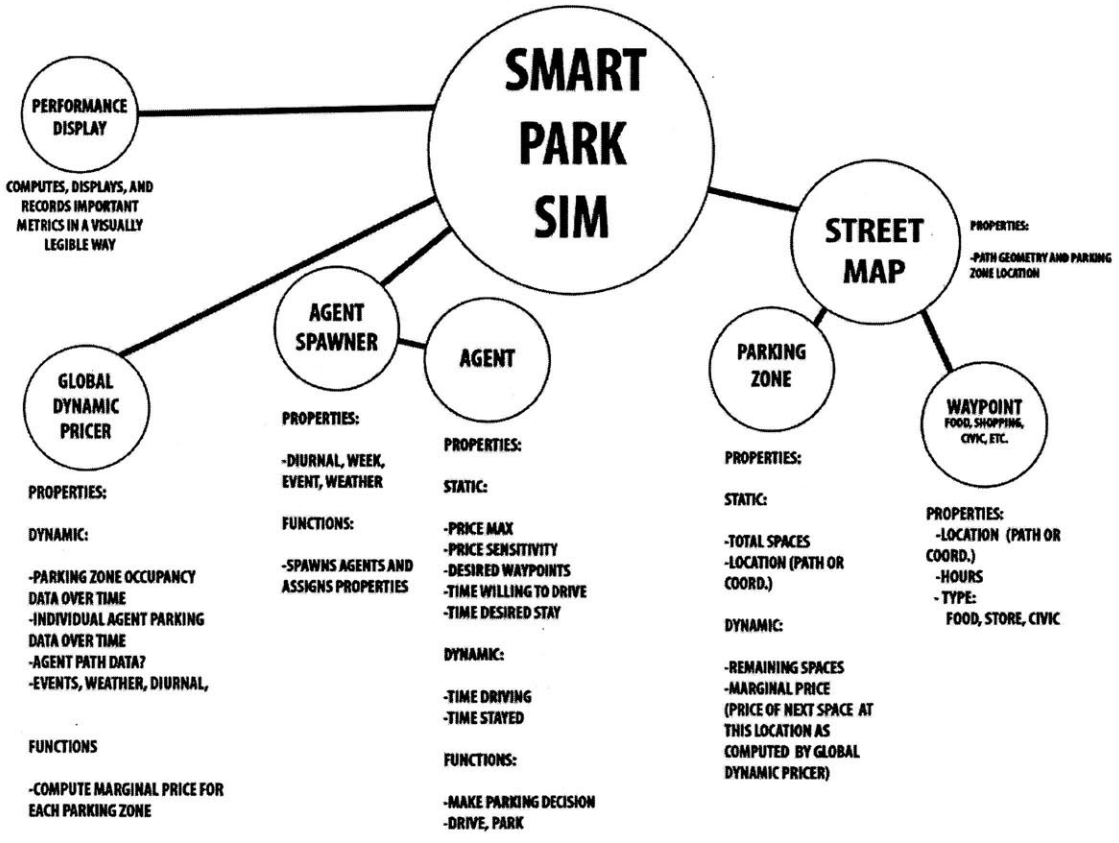


Figure 11: Overview of the parking simulation

We currently have plans to give a demonstration of our system in July 2012 in the Prenzlauer Berg neighborhood of Berlin in partnership with the BMW Guggenheim Lab. We have also been in talks with the Lord Mayor of the city of Copenhagen about launching an initial deployment there.

E-waste tagging and tracking

Every year Americans throw out 2.2 million tons of e-waste, or waste from discarded electronic devices. These include old cell phones, computers, and televisions. While the companies that manufacture these devices spend lots and lots of money understanding the supply chain behind their products; however, no one has taken the time or effort to examine the "removal chain" of these products after they have outlived their usefulness. This information can be used to expose the challenge of waste management and point to ways in which we can build a more efficient and sustainable infrastructure.

Previous research

It is not a simple task to pull off the trick of tracking items through the waste removal stream. The conditions the tags will have to survive and transmit through are highly variable, in temperature, humidity, pressure, orientation, and occlusion from the open sky. In 2004, Lee and Thomas proposed the use of radio transmitters to send GPS coordinates for the purpose of ensuring that hazardous materials are handled properly and don't lead to environmental damage (Lee, Thomas, & ieee, 2004). RFID (radio frequency identification technology), which is widely used in supply chain monitoring, could also be used for our situation (Binder, Quirici, Domnitcheva, & Staubli, 2008).

We currently have plans to give a demonstration of our system in July 2012 in the Prenzlauer Berg neighborhood of Berlin in partnership with the BMW Guggenheim Lab. We have also been in talks with the Lord Mayor of the city of Copenhagen about launching an initial deployment there.

E-waste tagging and tracking

Every year Americans throw out 2.2 million tons of e-waste, or waste from discarded electronic devices. These include old cell phones, computers, and televisions. While the companies that manufacture these devices spend lots and lots of money understanding the supply chain behind their products; however, no one has taken the time or effort to examine the "removal chain" of these products after they have outlived their usefulness. This information can be used to expose the challenge of waste management and point to ways in which we can build a more efficient and sustainable infrastructure.

Previous research

It is not a simple task to pull off the trick of tracking items through the waste removal stream. The conditions the tags will have to survive and transmit through are highly variable, in temperature, humidity, pressure, orientation, and occlusion from the open sky. In 2004, Lee and Thomas proposed the use of radio transmitters to send GPS coordinates for the purpose of ensuring that hazardous materials are handled properly and don't lead to environmental damage (Lee, Thomas, & ieee, 2004). RFID (radio frequency identification technology), which is widely used in supply chain monitoring, could also be used for our situation (Binder, Quirici, Domnitcheva, & Staubli, 2008).

However, because the tags are only detectable at short range, the cost of deploying electronic readers would be significant.

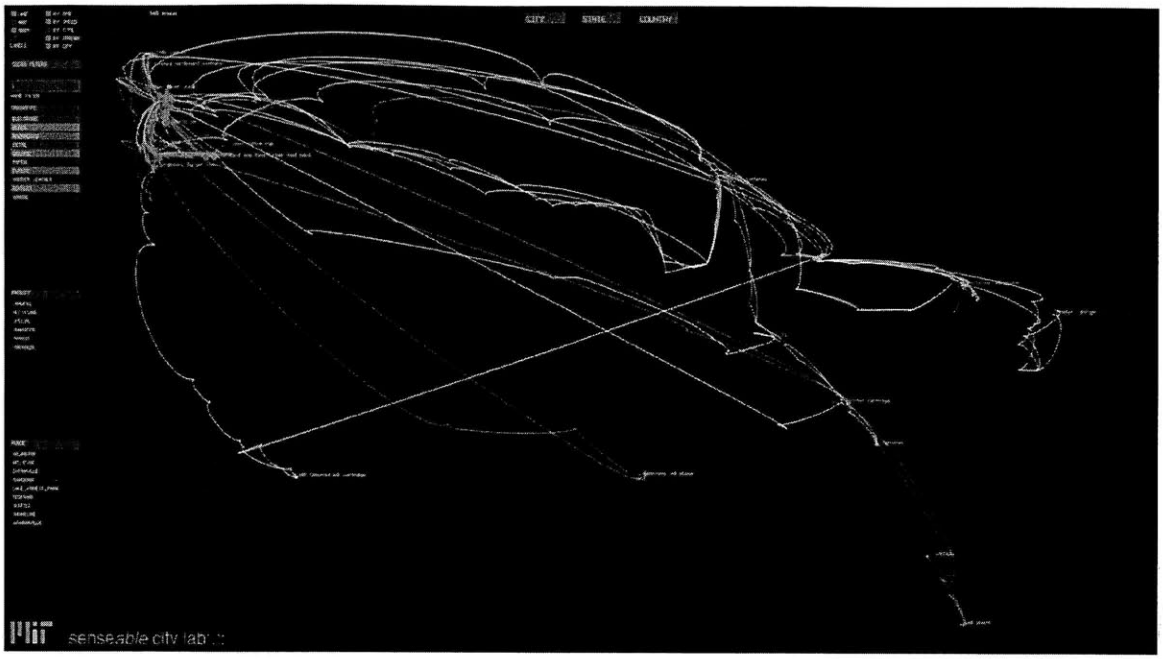


Figure 12: Tracking trash to its final destination

First iteration: GSM tags

The first iteration of the TrashTrack project used small GSM tags attached to the trash being tracked. The tags compare signal strengths from nearby cell towers in order to get a rough fix on its own location, a method known as cell tower triangulation. The tags also include a GPS unit, which provides a more precise location, but at the cost of reduced reliability when the trash is buried and there is no line of sight with the sky. In this case, the cellular technology takes over.

One problem that had to be solved was the question of how to guarantee that the tag carries with it enough power to continue transmitting all the way until the piece of trash has reached its final destination. In our solution, we developed algorithms that could keep the tag off for most of the time, waking it up only from time to time in order

to report its location. The tag included a motion sensor, which allowed us to wake up the tag based on when the sensor is in motion, and allow it to keep hibernating if it hasn't moved. In addition, the rate at which the samples are sent is varied according to the specific conditions. If previously unseen cell towers are observed, or if the tag seems to be moving, then the sampling rate is increased. The tags and their components are completely RoHS compliant, and can legally be introduced into waste streams in the US and the EU.

Figure 13: The tag used to track pieces of trash

These tags were deployed in Seattle, WA as the first phase of the TrashTrack project. Two thousand waste items, provided by volunteers based on a prioritized list of waste categories, were tagged and then disposed of normally. Items smaller than the tags themselves were excluded, in order to preserve the shape of the trash, as were

organic waste, in order to prevent unwanted contamination of composting piles with the e-waste of the tags. In order to strike a balance between the lifetime of the battery (and thus the useful lifetime of the tag) and the granularity of its readings, we send about half of the tags out set to send data every six hours, and set the other half to report every three to four hours. The location data that came back was corroborated with data from the EPA's Facility Registry System and from waste management companies working with city of Seattle.

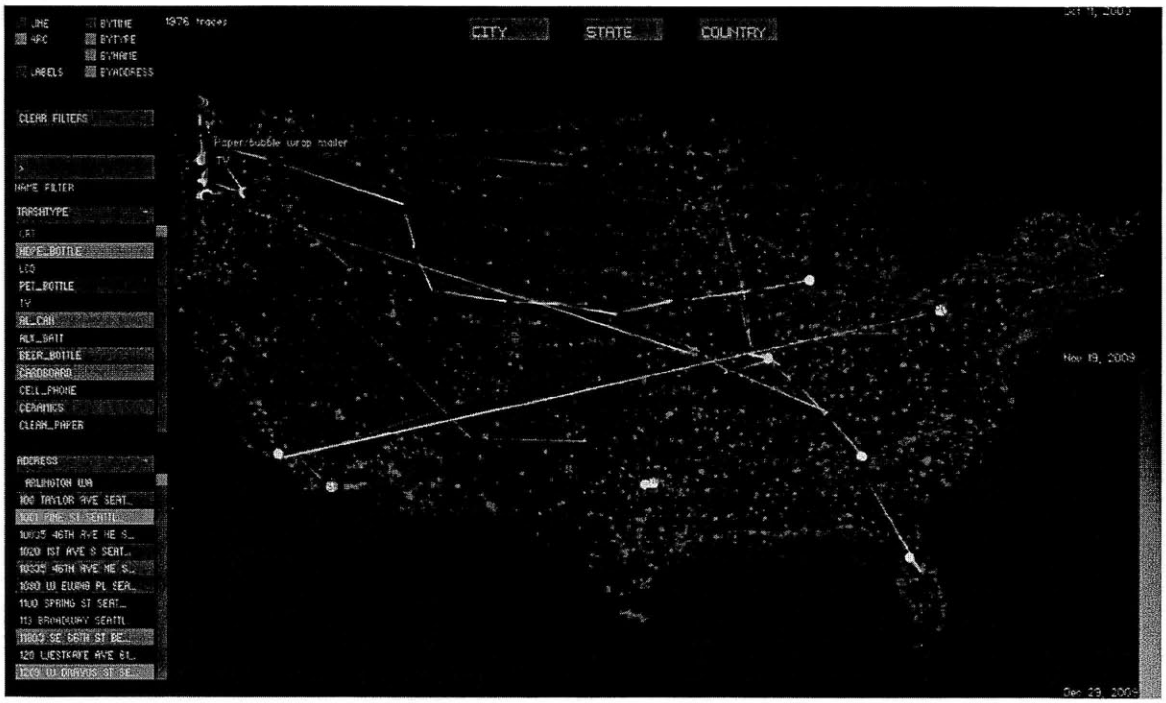


Figure 14 Traces with waste processing facilities from the EPA overlaid

When the traces came back, they gave us some very interesting results. They showed that household hazardous waste and electronic waste were the most erratic, often traveling across the country before coming to a final resting places. We did an analysis on the impact that the transportation of waste after it was disposed of had on the environment, and found that some of the longest traces tracked were producing on

the order of 0.3–0.8 metric tons of greenhouse gases, depending on the mode of transportation. This represents a significant penalty to the environmental benefits realized from recycling.

This first phase was one of the first cases in which researchers were able to directly observe the trajectory of waste as it moves through the removal chain. Previously, the only data available was on the volume of material that is goes through each waste processing facility, rather than how these different waypoints and facilities are related to each other.

One interesting result was the relationship between the distance a specific piece of waste travelled and the commodity value of the piece of waste. As it turns out, the pieces of waste that travelled the furthest were those worth the most and those that were worth the least. This is consistent with the specialized treatment required for waste that contains potentially valuable components or materials, such e-waste, and also the specialized treatment needed for hazardous wastes that could be potentially dangerous if disposed of normally.

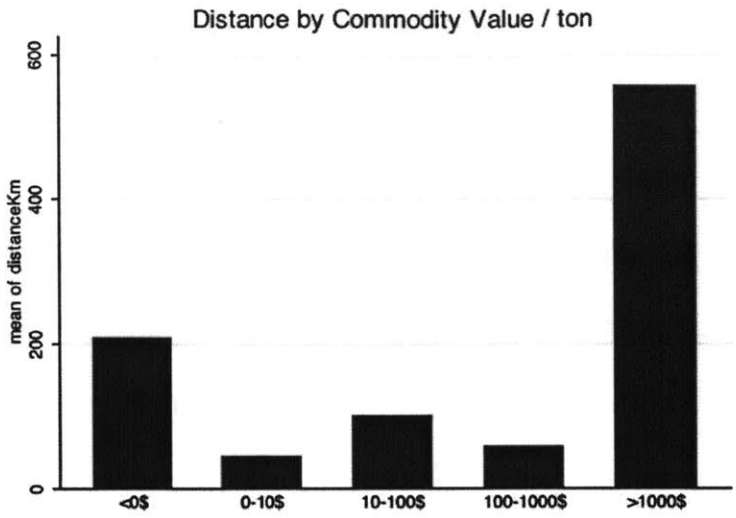


Figure 15: Distance by commodity value

Second iteration: Mobile phones

In the next phase of the project, we focused on recycling, and in particular the illegal transport of certain materials to developing countries. One area in which we have identified a problem is in recycling cathode ray tube (CRT) television monitors. While televisions based on CRT technology have passed their peak in the United States, they are still very popular in the developing world. In places like these, even a cheap television set is a luxury to have, and is often one of the first luxury purchases made with their disposable income. However, CRTs contain a significant amount of lead, which can be hazardous to the health of those exposed to it if it is simply left in a landfill. As a result, there are stringent guidelines regarding the disposal of CRTs that must be obeyed in many countries.

Despite these guidelines, the best practices are not always being followed, with disastrous results. Instead of disposing of the hazardous materials safely, some recyclers have found that they can make additional income by illegally selling the toxic materials to smelters in China, India, Ghana, and Pakistan. There, the lack of regulation means

that workers melt down the materials in an attempt to recover usable metals in open air, with no protection, posing a danger to themselves as well as to the environment.

While products such as cell phones, printers, and routers contain enough gold, copper, and silver to be recycled for a profit by large recyclers, these CRTs do not, instead costing $35 each to recycle. However, the temptation is to export the CRTs instead to smelters, for whom the value of the materials abroad outweighs the cost of shipping and cheap labor.

We have, in partnership with Qualcomm, developed a lightweight Android application that can run on low-end phones to transform the phones into sophisticated tracking tags. The application is designed to operate the phone in a low-power mode that only wakes up when it needs to gather sensor readings or when it needs to transmit these readings back to the central server. The phones wake up periodically, and take GPS traces and photographs that are queued to be sent back to the home server. The added functionality of the camera, and the fact that the phones will be able to run and transmit data across borders give us two advantages over the system used in the first deployment.

We would like these phones, once deployed, to continue reporting back to us for approximately 3 months, so we had to make some modifications. Instead of using the stock battery, we have retrofitted the phones with high-capacity batteries that can meet our specifications.

In order to estimate how long the phone would last in the field, we had to make some assumptions about the power consumption characteristics of the phone running

our tracking software. The program is set to wake up every four hours and do some work. It will snap a picture with its camera, attempt to get a location fix, and attempt to upload all of the data it currently has to the server (this could be data from previous sessions). The search for a location fix will timeout after ten minutes, and the upload attempt will timeout after two minutes, unless a location fix is still in progress.

Figure 16: An LG Thrive, the phone we have been using for our testing

Our assumptions then, are based on the power consumption characteristics of two states: wake and sleep. The phone will sleep for four hours every time it enters that phase, and its measured power consumption in this phase is around 3 mA. The wake phase can last for a variable amount of time (between 5 seconds and 10 minutes), depending on how readily accessible cellular signal is, but we have been working with an assumption that the average wake length is approximately 2 minutes. The power consumption in this mode was measured to be about 200 mA. Over the course of one

duty cycle (one sleep phase and one wake phase), we can therefore expect the phone to exhibit an average current consumption of 4.63 mA. If we wanted these batteries to last for 90 days, we would need about 10 Ah in our battery or battery pack.

We considered two methods of connecting our external battery to the phone. The first, and perhaps more elegant solution, was to use an external USB charger kit that includes a boost regulator to step up the voltage of the battery pack to the 5V, the voltage the phone expects for charging, then connects to the phone's USB charging port. We left the stock battery in its slot and connected the external battery through the USB port to charge the stock battery with. This solution was very modular in that it allowed either the battery or the phone to be connected or disconnected quickly and easily, but it had problems as well. Unfortunately, when the phone is connected to a power source over USB, as it was whenever the USB kit is connected, it will not clock itself down into sleep mode, and it won't hit the 3 mA consumption mode we were relying on in our calculations.

Instead, we wound up going with the more direct approach of soldering the battery directly to the positive and negative terminals of the phone. The third battery terminal on the phone, used for determining the battery level and temperature, was left unused. This approach, being the most direct, uses the battery to its full potential and minimizes waste due to inefficiencies in voltage or current transformation.

That left to us the task of selecting the battery that would give us the best performance. Our first pass at this selection process was to find the batteries rated for the highest number of amp-hours. The battery that we selected based on these criteria

was the ER34615, a lithium thionyl chloride battery that—at 19 Ah per D-sized unit—was advertised as the highest energy density of any practical lithium battery, three times greater than zinc manganese dioxide batteries.

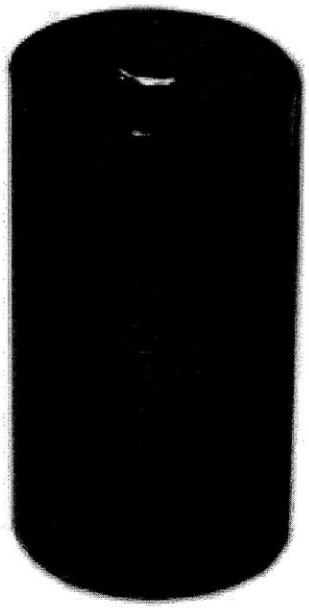


Figure 17: ER34615 battery

However, once we received a set of batteries to test in our system, we found that these batteries didn't perform anywhere near their nominal capacity. We determined this was due to the fact that we were not operating these batteries near their optimal current draw, giving us poor power efficiency. While the 19 Ah figure was arrived at assuming a constant 2 mA current draw, our real world system would fluctuate in current draw, peaking as high as 200 mA (as noted above).

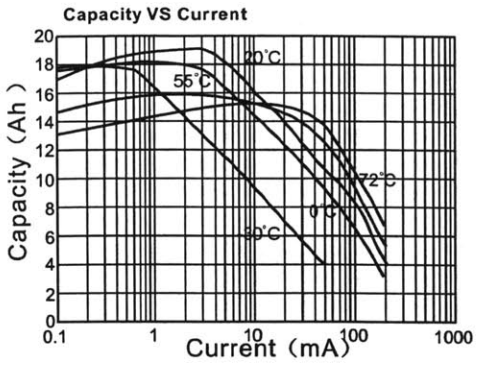


Figure 18: Capacity curves for ER34615 battery

Our second test setup involved an 18650 lithium-ion battery. At 1600-3100 mAh, these batteries don't have enough capacity to reach our target lifespan, but if several of the batteries are wired together in parallel, the desired capacity can be achieved. We looked at the capacity of several 18650 batteries down to a threshold voltage of 3.1 V, the voltage at which we experimentally determined that the phone shuts down from low voltage.

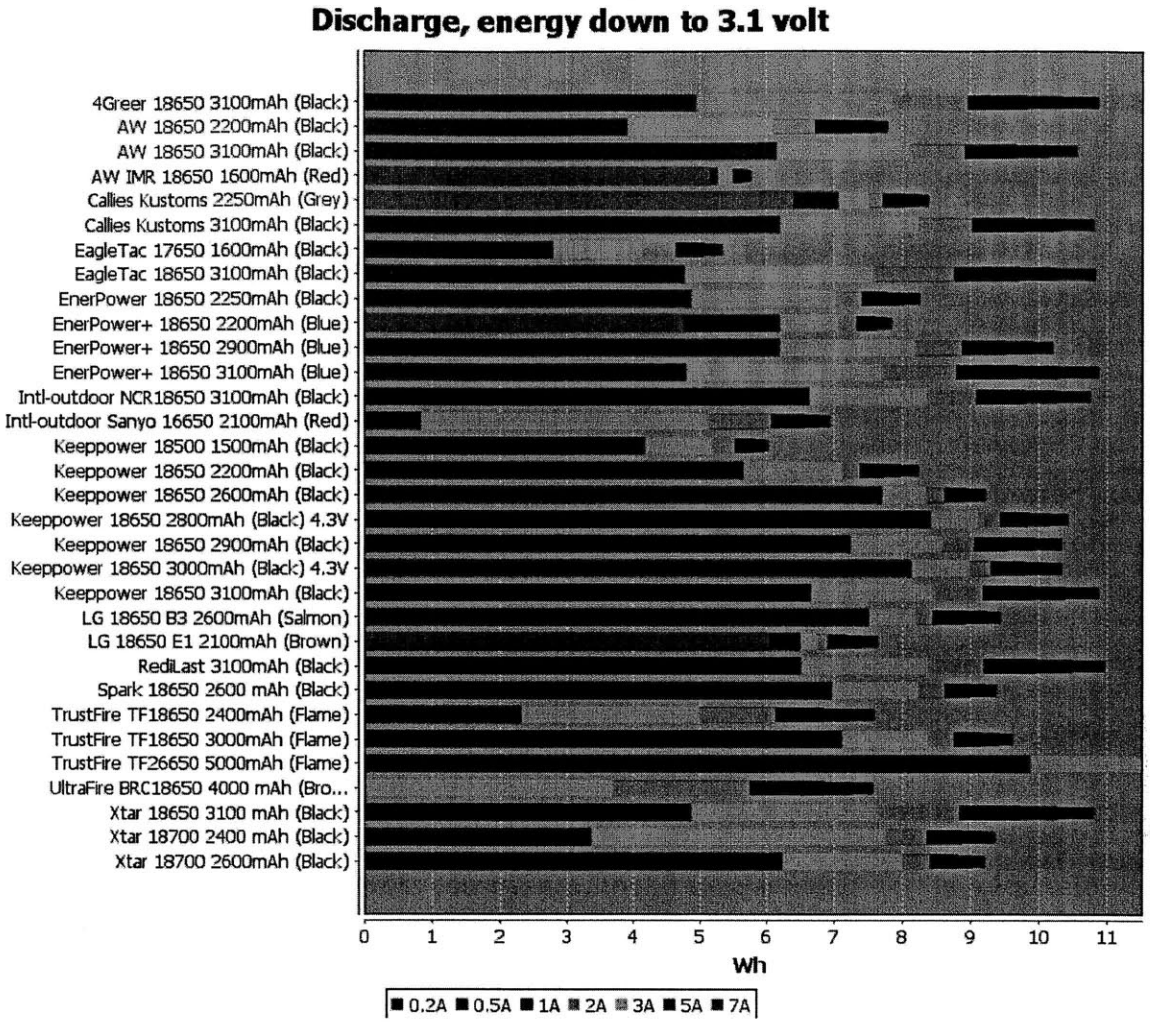


Figure 19: Discharge of several 18650 batteries to 3.1V

We found that the 18650 significantly outperformed the ER34615 for our purposes, despite having a much lower nominal capacity. We attribute this to the fact that the 18650 is designed to operate at the level of current we draw, whereas the ER34615 is not.

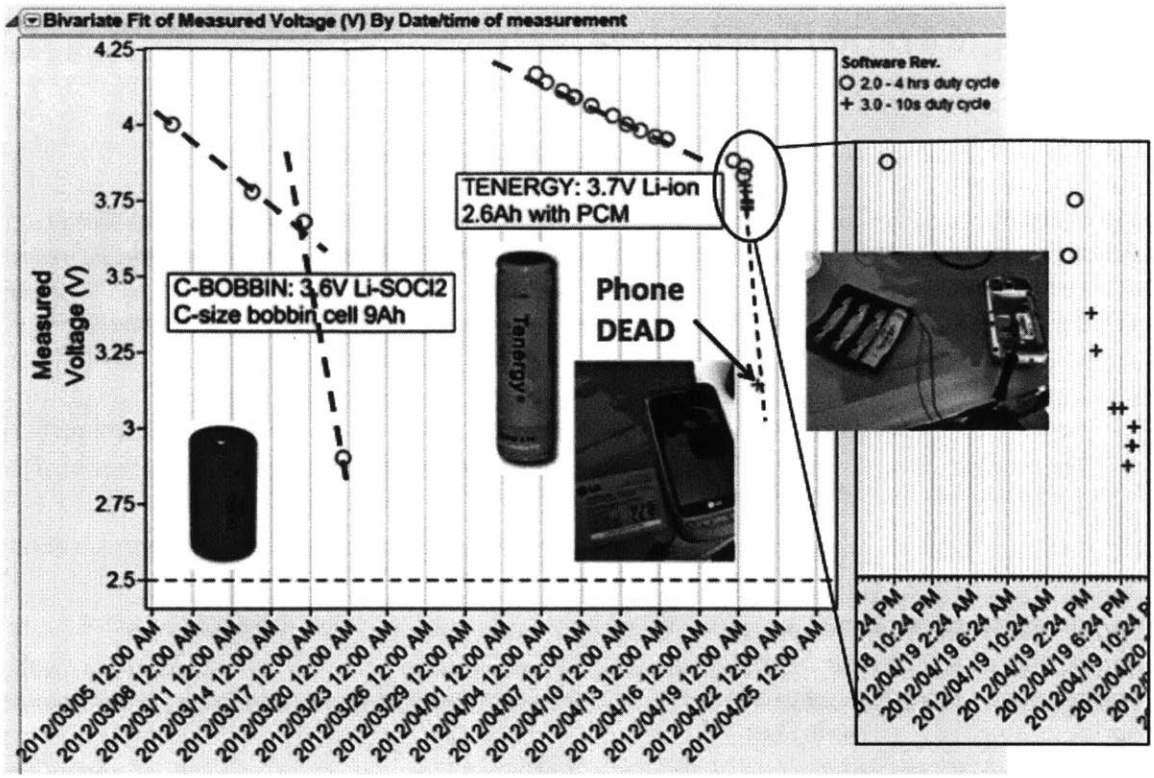


Figure 20: Performance of ER34615 and 18650 batteries

We are now in the process of discussing plans for a trial deployment of our phones with the BMW Guggenheim Lab this July in Berlin. After an initial deployment has been proven and validated, we would also like to look at other applications of the same technology. The very same tracking system used to tag CRT monitors could also be used for monitoring, enforcement or for coordinating fleets or crowds in real-time.

Appendix A: TrackerService.java

```
/*==========================================================================
========
FILE:           TrackerService.java
PROJECT:        MIT MoMA 2011
DISCLAIMER:     AS-IS. Use at own risk. No warranties. No liabilities.
LICENSE:        Sample code for illustration/research/educational
purpose.
==========================================================================
======*/

package com.qualcomm.mig.mit.moma;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;
import java.util.List;

import org.apache.http.client.HttpClient;
import org.apache.http.client.ResponseHandler;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.mime.HttpMultipartMode;
import org.apache.http.entity.mime.MultipartEntity;
import org.apache.http.entity.mime.content.ByteArrayBody;
import org.apache.http.impl.client.DefaultHttpClient;

import android.app.AlarmManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.hardware.Camera;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.media.AudioManager;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.os.IBinder;
import android.os.SystemClock;
import android.provider.Settings;
import android.telephony.NeighboringCellInfo;
```

```
import android.telephony.TelephonyManager;
import android.telephony.gsm.GsmCellLocation;
import android.text.format.DateFormat;
import android.util.Log;
import android.view.SurfaceView;

/**
 * TrackerService is an Android service that does all of the tracking
work.
 *
 * Tracking work is divided into "sessions" or "iterations", which are
spaced
 * apart by some hours. Each session attempts to obtain GPS/network
location,
 * take a picture, upload logs/pictures to server. For battery savings,
 * airplane mode is turned OFF at the start of a session, and turned ON
just
 * before ending the session, so that radios are off for those few
hours. The
 * CPU wake lock is released at the end of an iteration also, so that
the phone
 * can go to sleep.
 *
 * Normally, if a location fix was obtained, and all files have been
uploaded,
 * then we can end the iteration; if not, we stay awake at least 10
minutes to
 * try to obtain a location fix (GPS or network); also, we may stay
awake beyond
 * that if we are still making progress uploading the backlog of files;
at any
 * time if an upload attempt fails, and it has been more than 2 minutes
since
 * the last upload was successful, then we consider ourselves no longer
making
 * much progress uploading; but if the phone stays awake for some
reason (waiting
 * for GPS fix) and somehow after more than 2 minutes of failure in
uploading an
 * upload subsequently succeeds, of course then the 2 minute counter is
pushed out
 * and we would now need to fail uploads for 2 minutes consecutively
again before
 * considering ourselves unable to make any upload progress.
 *
 * Note that there is no "maximum" session time limit, so as long as
the backlog
 * still exists, and as long as we continue to make progress uploading,
then
 * we will continue to stay awake.
```

```
 */
public class TrackerService extends WakefulService {

	// TAG string for Android logcat logging.
	private static final String TAG = "MIT_MoMA_Tracker";

	// How long to wait (in milliseconds) between tracker
iteration/session?
//	private static final long WAIT_BETWEEN_SESSIONS_MS = 4 * 3600 *
1000;
	private static final long WAIT_BETWEEN_SESSIONS_MS = 10 * 1000;

	// How long to wait for camera to warm up? If too short, images
may be dark.
	private static final long WARMUP_TIME_MS = 5000;

	// wait at least 10 minutes before giving up on obtaining any
location fix
	private static final long MINIMUM_TRY_TIME_FOR_LOCATION = 10 * 60
* 1000;

	// do not log more than 10 fixes, no need to spam the log with
too many fixes in a session
	private static final int MAX_FIXES_TO_LOG_PER_SESSION = 10;

	// fail consecutively for 2 minutes before we consider data
network unavailable;
	// even if data network is deemed unavailable, we would continue
to retry as long
	// as some other reason cause us to remain awake (e.g. location
fix's 10 minute
	// has not reached); any time a successful upload occurs, we
consider data network
	// available again, and we also push back the consecutive time by
2 minutes again
	private static final long CAN_STOP_UPLOAD_RETRY_AFTER_TIME = 2 *
60000;

	// session should stay open at least this amount, even if
location is available
	// and upload is done (just to allow more location fixes).
	private static final long MINIMUM_SESSION_TIME_MS = 30000;

	// do not upload any picture files that are too small (probably
black image anyway)
	private static final int MINIMUM_JPEG_FILE_SIZE_TO_UPLOAD =
20000;

	// how long to wait between upload retries (if an upload failure
occurred)
```

```
      private static final long WAIT_BETWEEN_UPLOAD_RETRIES_MS = 10000;
// 10 seconds

      // how often to check if we are done (i.e. uploads done and
location fixes done)
      private static final int CHECK_DONE_INTERVAL_MS = 5000;

      // These objects are simply created once onCreate.
      private TelephonyManager telephonyManager = null;
      private AlarmManager alarmManager = null;
      private AudioManager audioManager = null;
      private int ringerModeBeforeTakePicture = 0;
      private PendingIntent pendingIntent = null;
      private LocationManager locationManager = null;
      private LocationListener locationListener = null;

      private String deviceId = null;
      private String subscriberId = null;
      private String line1Number = null;

      private boolean sessionInProgress = false;
      private Camera camera=null;
      private int sessionNumber = 0;
      private long sessionStartTime = 0;
      private boolean cameraPreviewing = false;
      private boolean uploadsCompleted = false;
      private int numFixesObtained = 0;

      // how many times have we dumped TelephonyManager info in this
session
      private int tmdumpTotal = 0;
      // down counter which must reach 0 before we dump
TelephonyManager info again
      private int tmdumpDowncounter = 0;

      private void log(String msg)
      {
          PrintWriter pw;
          try {
            String nowString = DateFormat.format("yyyy-MM-dd kk:mm:ss
zz", new Date()).toString();
            String logName = String.format("/sess%04d.log",
sessionNumber);
            pw = new PrintWriter(new
FileWriter(Environment.getExternalStorageDirectory()+logName, true));
              pw.printf("%s -- %s\n", nowString, msg);
              pw.flush();
              pw.close();
          } catch (IOException e) {
              e.printStackTrace();
```

```
        }
        Log.d(TAG, msg);
    }

    private void restoreSessionNumber() {
        try {
            BufferedReader br = new BufferedReader(new
FileReader(Environment.getExternalStorageDirectory()+"/session.dat"));
            String line = br.readLine();
            sessionNumber = Integer.parseInt(line);
            br.close();
        } catch (Exception e) {
            // Sure hope this doesn't happen.
            sessionNumber = 0;
        }
    }

    private void saveSessionNumber() {
        try {
            PrintWriter pw = new PrintWriter(new
FileWriter(Environment.getExternalStorageDirectory()+"/session.dat",
false));
            pw.printf("%d\n", sessionNumber);
            pw.flush();
            pw.close();
        } catch (Exception e) {
            // not much we can do
        }
    }

    @Override
    public void onCreate() {
        super.onCreate();

        restoreSessionNumber();
        ++sessionNumber;
        saveSessionNumber();

        log("TrackerService onCreate, session " + sessionNumber);

        telephonyManager =
(TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
        deviceId = telephonyManager.getDeviceId();
        subscriberId = telephonyManager.getSubscriberId();
        line1Number = telephonyManager.getLine1Number();

        alarmManager = (AlarmManager)
getSystemService(Context.ALARM_SERVICE);
        Intent i = new Intent(this, OnAlarmReceiver.class);
        pendingIntent = PendingIntent.getBroadcast(this, 0, i, 0);
```

```
            audioManager = (AudioManager)
getSystemService(Context.AUDIO_SERVICE);

            locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);
            locationListener = new LocationListener() {
                  public void onLocationChanged(Location location) {
                        String provider = location.getProvider();
                        long   time     = location.getTime();
                        double lat      = location.getLatitude();
                        double lon      = location.getLongitude();
                        float  accuracy = location.getAccuracy();
                        double altitude = location.getAltitude();
                        float  bearing  = location.getBearing();
                        float  speed    = location.getSpeed();
                        String info =
                              "prov:" + provider + "," +
                              "time:" + time     + "," +
                              "lat:"  + lat      + "," +
                              "lon:"  + lon      + "," +
                              "acc:"  + accuracy + "," +
                              "alt:"  + altitude + "," +
                              "bear:" + bearing  + "," +
                              "spd:"  + speed;
                        ++numFixesObtained;
                        if (numFixesObtained <=
MAX_FIXES_TO_LOG_PER_SESSION) {
                              log("LocationListener - onLocationChanged
- " + info);
                        }
                  }
                  public void onStatusChanged(String provider, int
status, Bundle extras) {
                        log("LocationListener - onStatusChanged - " +
provider + ":" + status);
                  }

                  public void onProviderEnabled(String provider) {
                        log("LocationListener - onProviderEnabled - " +
provider);
                  }

                  public void onProviderDisabled(String provider) {
                        log("LocationListener - onProviderDisabled - "
+ provider);
                  }
            };
      }
```

```
	@Override
	public void onDestroy() {
		super.onDestroy();

		log("TrackerService onDestroy, session " + sessionNumber);

		audioManager = null;
		alarmManager = null;
		pendingIntent = null;
		locationManager = null;
		locationListener = null;
		telephonyManager = null;

		getLock(this).release();
	}

	@Override
	public int onStartCommand(Intent intent, int flags, int startId)
{

		if (sessionInProgress) {
			return START_STICKY;
		}

		sessionInProgress = true;

		sessionStartTime = SystemClock.uptimeMillis();
		log("TrackerService onStartCommand STARTING");

		// Cancel the current alarm. We will set it again just
before leaving this session.
		alarmManager.cancel(pendingIntent);

		// Turn OFF airplane mode. This returns immediately, but
takes a while to be effective!
		setAirplaneMode(false);

		// Open the camera, and begin preview immediately, camera
takes a while to warm up.
		// If not warmed-up, picture will be taken, but either
black or very dark (LG Thrive).
		cameraOpenAndPreview();

		// Request for location updates immediately, this will also
trickle in over time.
		numFixesObtained = 0;
		startLocationUpdate();

		// Reset TelephonyManager dump variables.
		tmdumpTotal = 0;
```

```
            tmdumpDowncounter = 0;

            // All the remaining operations (upload, take picture) will
defer for a few seconds.
            Handler handler = new Handler();
            handler.postDelayed(new Runnable()
            {
                  public void run() {
                        delayedOperationsForWarmup();
                  }
            }, WARMUP_TIME_MS);

            log("TrackerService onStartCommand ENDING");
            return START_STICKY;
      }

      protected void delayedOperationsForWarmup() {
            log("delayedOperationsForWarmup called.");

            // Because camera takes a bit of time to "warm up", wait
some time before taking picture.
            cameraTakePicture();

            // Start a separate thread to do upload whatever has not
been uploaded yet.
            uploadsCompleted = false;
            new UploadToServerTask().execute(new byte[1]);

            // Compute how long to wait to achieve MINIMUM session time
(for GPS fix collection).
            long nowTime = SystemClock.uptimeMillis();
            long waitTime = MINIMUM_SESSION_TIME_MS - (nowTime -
sessionStartTime);
            if (waitTime <= 0) {
                  waitTime = 1;
            }

            Handler handler = new Handler();
            handler.postDelayed(new Runnable()
            {
                  public void run() {
                        shutdownWhenUploadsComplete();
                  }
            }, waitTime);
      }

      private String nciToString(NeighboringCellInfo nci) {
            int cid = nci.getCid();
            int lac = nci.getLac();
            int nettype = nci.getNetworkType();
```

```
            int psc = nci.getPsc();
            int rssi = nci.getRssi();
            return
                  ""
                  + cid + ","
                  + lac + ","
                  + nettype + ","
                  + psc + ","
                  + rssi;
      }

      private void dumpTelephonyManagerInfo() {
            TelephonyManager tm = telephonyManager; // shorter name
            int callState = tm.getCallState();
            GsmCellLocation loc = (GsmCellLocation)
tm.getCellLocation();
            String cellidstr = "(null)";
            String lacstr = "(null)";
            if (loc != null) {
                  int cellid = loc.getCid();
                  int lac = loc.getLac();
                  cellidstr = "" + cellid;
                  lacstr = "" + lac;
            }
            String operatorname = tm.getNetworkOperatorName();
            String operatorcode = tm.getNetworkOperator();
            String operatoriso = tm.getNetworkCountryIso();
            int networkType = tm.getNetworkType();

            log(
                  "TM|"
                  + callState + "|"
                  + cellidstr + "|"
                  + lacstr + "|"
                  + operatorname + "|"
                  + operatorcode + "|"
                  + operatoriso + "|"
                  + networkType
            );

            List<NeighboringCellInfo> cellinfo =
tm.getNeighboringCellInfo();
            if (cellinfo != null && cellinfo.size() > 0) {
                  String allNci = "NCI";
                  for(NeighboringCellInfo info : cellinfo){
                        allNci = allNci + "|" + nciToString(info);
                  }
                  log(allNci);
            }
      }
```

```
	protected void shutdownWhenUploadsComplete() {

		if (tmdumpDowncounter <= 0) {
			dumpTelephonyManagerInfo();
			++tmdumpTotal;
			tmdumpDowncounter = tmdumpTotal;
		}
		else {
			--tmdumpDowncounter;
		}

		if ((numFixesObtained > 0 || (SystemClock.uptimeMillis() -
sessionStartTime > MINIMUM_TRY_TIME_FOR_LOCATION))
				&& uploadsCompleted) {

			setAirplaneMode(true);
			stopLocationUpdate();
			cameraShutdown();

	alarmManager.set(AlarmManager.ELAPSED_REALTIME_WAKEUP,
					SystemClock.elapsedRealtime() +
WAIT_BETWEEN_SESSIONS_MS,
					pendingIntent);
			sessionInProgress = false;
			stopSelf();
		}
		else {
			Handler handler = new Handler();
			handler.postDelayed(new Runnable() {
				public void run() {
					shutdownWhenUploadsComplete();
				}
			}, CHECK_DONE_INTERVAL_MS);
		}
	}

	private void startLocationUpdate() {
		log("startLocationUpdate called.");
		locationManager.requestLocationUpdates(
				LocationManager.NETWORK_PROVIDER, 0, 0,
locationListener);
//		locationManager.requestLocationUpdates(
//				LocationManager.GPS_PROVIDER, 0, 0,
locationListener);

	}

	private void stopLocationUpdate() {
		log("stopLocationUpdate called.");
```

```
            locationManager.removeUpdates(locationListener);
            locationListener = null;
            locationManager = null;
      }

      private void cameraOpenAndPreview() {
            log("cameraOpenAndPreview called.");
            if (camera == null) {
                  camera = Camera.open();
            }
            if (camera != null) {
                  try {
                        SurfaceView view = new SurfaceView(this);
                        camera.setPreviewDisplay(view.getHolder());
                        camera.startPreview();
                        cameraPreviewing = true;
                  }
                  catch (IOException e) {
                  }
            }
      }

      private synchronized void cameraShutdown() {
            log("cameraShutdown called.");
            if (camera != null) {
                  if (cameraPreviewing) {
                        camera.stopPreview();
                  }
                  camera.release();
            }
            camera = null;
            cameraPreviewing = false;
      }

      private void cameraTakePicture() {
            log("cameraTakePicture called.");
            if (camera != null && cameraPreviewing) {

                  // Here we handle the "shutter" sound when taking a
picture.
                  // Basically, we want to mute that sound, but it
appears the best
                  // way to do so is to brute-force mute the audio
altogether!
                  // For now, we will also restore ringer mode to
original value
                  // after the picture is successfully taken.
                  ringerModeBeforeTakePicture =
audioManager.getRingerMode();
```

```
      audioManager.setRingerMode(AudioManager.RINGER_MODE_SILENT);

                  camera.takePicture(null, null, null,
                        new Camera.PictureCallback() {
                              public void onPictureTaken(byte[] data,
Camera camera) {
                                    String info = "Camera
onPictureTaken PostView callback, data is ";
                                    if (data == null) {
                                          info += "null";
                                    } else {
                                          info += data.length + "
bytes";
                                    }
                                    log(info);

                                    // if we have some data, spawn
separate thread to
                                    // write data into storage.
                                    if (data != null) {
                                          new
SavePhotoTask().execute(data);
                                    }

      audioManager.setRingerMode(ringerModeBeforeTakePicture);
                              }
                        });
            }
      }

      private void setAirplaneMode(boolean bMode) {
            int nMode = (bMode == false ? 0 : 1);
            log("setAirplaneMode(" + nMode + ") called.");
            Settings.System.putInt(getContentResolver(),
Settings.System.AIRPLANE_MODE_ON, nMode);
            Intent intent = new
Intent(Intent.ACTION_AIRPLANE_MODE_CHANGED);
            intent.putExtra("state", (nMode != 0));
            sendBroadcast(intent);
      }

      @Override
      public IBinder onBind(Intent arg0) {
            return null;
      }

      class SavePhotoTask extends AsyncTask<byte[], String, String> {
            @Override
            protected String doInBackground(byte[]... jpeg) {
```

```
                try {
                        File img = new
File(Environment.getExternalStorageDirectory(),
                                        String.format("IMG%04d.JPG",
sessionNumber));
                        if (img.exists()) {
                                img.delete();
                        }
                        FileOutputStream fos = new
FileOutputStream(img.getPath());
                        fos.write(jpeg[0]);
                        fos.close();
                        // we can go ahead to shutdown the camera at
this point
                        cameraShutdown();
                }
                catch (IOException e) {
                }
                return null;
        }
    }

    class UploadToServerTask extends AsyncTask<byte[], String,
String> {

        @Override
        protected String doInBackground(byte[]... dummy) {
                log("UploadToServerTask doInBackground called.");

                long retryUntil = SystemClock.uptimeMillis() +
CAN_STOP_UPLOAD_RETRY_AFTER_TIME;

                HttpClient httpClient = new DefaultHttpClient();

                String[] filenames = null;
                if (Environment.getExternalStorageDirectory() !=
null) {
                        filenames =
Environment.getExternalStorageDirectory().list();
                }
                if (filenames != null) {

                        for (String filename : filenames) {

                                if (filename.startsWith("sess") &&
filename.endsWith(".log")) {

                                        String currentName =
String.format("sess%04d.log", sessionNumber);
                                        if (filename.equals(currentName)) {
```

```
                                    continue;
                                }

                                File file = null;
                                byte[] data = null;
                                try {
                                    file = new
File(Environment.getExternalStorageDirectory(), filename);
                                    data = new byte[(int)
file.length()];
                                    FileInputStream fis = new
FileInputStream(file);
                                    fis.read(data);
                                    fis.close();
                                }
                                catch (Exception e) {
                                    continue;
                                }

                                int unixTime = (int)
(System.currentTimeMillis() / 1000);
                                HttpPost httpPost = new
HttpPost("http://forager.xvm.mit.edu/qualcomm/track.php?t="
                                        + unixTime
                                        + "&devid=" + deviceId
                                        + "&subid=" +
subscriberId
                                        + "&line1=" +
line1Number
                                        );
                                MultipartEntity entity = new
MultipartEntity(HttpMultipartMode.BROWSER_COMPATIBLE);
                                ByteArrayBody bab = new
ByteArrayBody(data, filename);
                                entity.addPart("uploaded", bab);
                                httpPost.setEntity(entity);

                                ResponseHandler<byte[]>
responseHandler = new ServerResponseHandler();

                                long nowTime =
SystemClock.uptimeMillis();
                                while (nowTime < retryUntil) {
                                    try {

    @SuppressWarnings("unused")
                                        byte[] responseBody =
httpClient.execute(httpPost, responseHandler);
                                        break;
                                    }
```

```
                                    catch (IOException e) {
                                          log("Failed upload: " +
e);
                                    }
                                    log("Waiting " +
WAIT_BETWEEN_UPLOAD_RETRIES_MS
                                                + " milliseconds
to retry...");
                                    try {
Thread.sleep(WAIT_BETWEEN_UPLOAD_RETRIES_MS); } catch
(InterruptedException e) { }
                                    nowTime =
SystemClock.uptimeMillis();
                              }

                              if (nowTime < retryUntil) {
                                    log("Upload of " + filename +
" succeeded");
                                    file.delete();
                                    retryUntil =
SystemClock.uptimeMillis() + CAN_STOP_UPLOAD_RETRY_AFTER_TIME;
                              }
                              else {
                                    log("Upload failed, retry
duration exceeded");
                              }

                        }

                        if (SystemClock.uptimeMillis() >
retryUntil) {
                              break;
                        }
                  }

                  for (String filename : filenames) {

                        if (SystemClock.uptimeMillis() >
retryUntil) {
                              break;
                        }
                        if (filename.startsWith("IMG") &&
filename.endsWith(".JPG")) {

                              File file = null;
                              byte[] data = null;
                              try {
                                    file = new
File(Environment.getExternalStorageDirectory(), filename);
```

```
                                        long filesize =
file.length();
                                        if (filesize <
MINIMUM_JPEG_FILE_SIZE_TO_UPLOAD) {
                                              file.delete();
                                              log("Deleting " +
filename + " without upload. Too small. Size is " + filesize);
                                              continue;
                                        }

                                        data = new byte[(int)
file.length()];
                                        FileInputStream fis = new
FileInputStream(file);
                                        fis.read(data);
                                        fis.close();
                                  }
                                  catch (Exception e) {
                                        continue;
                                  }

                                  int unixTime = (int)
(System.currentTimeMillis() / 1000);
                                  HttpPost httpPost = new
HttpPost("http://forager.xvm.mit.edu/qualcomm/track.php?t="
                                              + unixTime
                                              + "&devid=" + deviceId
                                              + "&subid=" +
subscriberId
                                              + "&line1=" +
line1Number
                                              );
                                  MultipartEntity entity = new
MultipartEntity(HttpMultipartMode.BROWSER_COMPATIBLE);
                                  ByteArrayBody bab = new
ByteArrayBody(data, filename);
                                  entity.addPart("uploaded", bab);
                                  httpPost.setEntity(entity);

                                  ResponseHandler<byte[]>
responseHandler = new ServerResponseHandler();

                                  long nowTime =
SystemClock.uptimeMillis();
                                  while (nowTime < retryUntil) {
                                        try {

      @SuppressWarnings("unused")
                                              byte[] responseBody =
httpClient.execute(httpPost, responseHandler);
```

```
                                        break;
                                    }
                                    catch (IOException e) {
                                        log("Failed upload: " +
e);
                                    }
                                    log("Waiting " +
WAIT_BETWEEN_UPLOAD_RETRIES_MS
                                            + " milliseconds
to retry...");
                                    try {
Thread.sleep(WAIT_BETWEEN_UPLOAD_RETRIES_MS); } catch
(InterruptedException e) { }
                                    nowTime =
SystemClock.uptimeMillis();
                                }

                                if (nowTime < retryUntil) {
                                    log("Upload of " + filename +
" succeeded");
                                    file.delete();
                                    retryUntil =
SystemClock.uptimeMillis() + CAN_STOP_UPLOAD_RETRY_AFTER_TIME;
                                }
                                else {
                                    log("Upload failed, retry
duration exceeded");
                                }

                            }

                            if (SystemClock.uptimeMillis() >
retryUntil) {
                                break;
                            }
                        }
                    }

                    httpClient.getConnectionManager().shutdown();
                    uploadsCompleted = true;
                    return null;
                }
            }
}
```

Appendix B: PQMapViewController.m

```
//
//  PQMapViewController.m
//  parq-ios2
//
//  Created by Mark Yen on 3/16/12.
//  Copyright (c) 2012 Massachusetts Institute of Technology. All
rights reserved.
//

#import "PQMapViewController.h"
#import "MKShape+Color.h"
#import "UIColor+Parq.h"
#import "CalloutMapAnnotation.h"
#import "CalloutMapAnnotationView.h"

#define METERS_PER_MILE 1609.344

#define STREET_MAP_SPAN 0.0132
#define SPOT_MAP_SPAN 0.0017

#define GPS_LAUNCH_ALERT 10
#define MAX_CALLOUTS 8
typedef enum {
    kGridZoomLevel,
    kStreetZoomLevel,
    kSpotZoomLevel
} ZoomLevel;

@interface PQMapViewController ()
@property (strong, nonatomic) UIView *disableViewOverlay;
@property (strong, nonatomic) UIBarButtonItem *leftBarButton;
@property (nonatomic) ZoomLevel zoomState;
@end

@implementation PQMapViewController
@synthesize gCircle;
@synthesize callouts;
@synthesize calloutLines;
@synthesize map;
@synthesize topSearchBar;
@synthesize availabilitySelectionView;
@synthesize bottomSpotSelectionView;
@synthesize topSpotSelectionView;
@synthesize navigationBar;
@synthesize bottomSpotSelectionBar;
@synthesize topSpotSelectionBar;
@synthesize geocoder;
@synthesize disableViewOverlay;
```

```
@synthesize leftBarButton;
@synthesize zoomState;

-(void) tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath{
    NSLog(@"hello\n");
}

- (NSArray*)loadGridData {
    return [NSArray arrayWithObjects:
            [[NSDictionary alloc] initWithObjectsAndKeys:@"42.350393,-
71.104159", @"nw_corner", @"42.360393,-71.114159", @"se_corner",
[NSNumber numberWithInt:0], @"color", nil],
            [[NSDictionary alloc] initWithObjectsAndKeys:@"42.360393,-
71.104159", @"nw_corner", @"42.370393,-71.114159", @"se_corner",
[NSNumber numberWithInt:4], @"color",nil],
            [[NSDictionary alloc] initWithObjectsAndKeys:@"42.370393,-
71.104159", @"nw_corner", @"42.380393,-71.114159", @"se_corner",
[NSNumber numberWithInt:2], @"color",nil],
            [[NSDictionary alloc] initWithObjectsAndKeys:@"42.350393,-
71.114159", @"nw_corner", @"42.360393,-71.124159", @"se_corner",
[NSNumber numberWithInt:3], @"color",nil],
            [[NSDictionary alloc] initWithObjectsAndKeys:@"42.360393,-
71.114159", @"nw_corner", @"42.370393,-71.124159", @"se_corner",
[NSNumber numberWithInt:3], @"color",nil],
            [[NSDictionary alloc] initWithObjectsAndKeys:@"42.370393,-
71.114159", @"nw_corner", @"42.380393,-71.124159", @"se_corner",
[NSNumber numberWithInt:0], @"color",nil],
            [[NSDictionary alloc] initWithObjectsAndKeys:@"42.350393,-
71.124159", @"nw_corner", @"42.360393,-71.134159", @"se_corner",
[NSNumber numberWithInt:1], @"color",nil],
            [[NSDictionary alloc] initWithObjectsAndKeys:@"42.360393,-
71.124159", @"nw_corner", @"42.370393,-71.134159", @"se_corner",
[NSNumber numberWithInt:2], @"color",nil],
            [[NSDictionary alloc] initWithObjectsAndKeys:@"42.370393,-
71.124159", @"nw_corner", @"42.380393,-71.134159", @"se_corner",
[NSNumber numberWithInt:3], @"color",nil], nil];

}

- (NSArray*)loadBlockData {

    return [NSArray arrayWithObjects:
            [[NSDictionary alloc] initWithObjectsAndKeys:@"42.36441,-
71.113901;42.365203,-71.104846", @"line", [NSNumber numberWithInt:0],
@"color", nil],
            [[NSDictionary alloc] initWithObjectsAndKeys:@"42.365399,-
71.110897;42.364751,-71.110771", @"line", [NSNumber numberWithInt:4],
@"color", nil], nil];
```

```
}

-(NSArray*) loadSpotData {
    return [NSArray arrayWithObjects:
            @"42.365354, -71.110843, 1, 1410",
            @"42.365292, -71.110835, 1, 1412",
            @"42.365239, -71.110825, 1, 1414",
            @"42.365187, -71.110811, 0, 1416",
            @"42.365140, -71.110806, 1, 1418",
            @"42.365092, -71.110798, 0, 1420",
            @"42.365045, -71.110790, 1, 1422",
            @"42.364995, -71.110782, 0, 1424",
            @"42.364947, -71.110768, 0, 1426",
            @"42.364896, -71.110766, 0, 1428",
            @"42.364846, -71.110752, 0, 1430",
            @"42.364797, -71.110739, 0, 1432",

            @"42.365348, -71.110924, 1, 1411",
            @"42.365300, -71.110916, 0, 1413",
            @"42.365251, -71.110905, 0, 1415",
            @"42.365203, -71.110900, 0, 1417",
            @"42.365154, -71.110892, 1, 1419",
            @"42.365104, -71.110876, 0, 1421",
            @"42.365049, -71.110868, 1, 1423",
            @"42.364993, -71.110860, 1, 1425",
            @"42.364943, -71.110849, 1, 1427",
            @"42.364894, -71.110846, 1, 1429",
            @"42.364846, -71.110835, 0, 1431",
            @"42.364799, -71.110830, 1, 1433",
            nil];
}

- (void)alertView:(UIAlertView *)alertView
clickedButtonAtIndex:(NSInteger)buttonIndex {
    if(alertView.tag==GPS_LAUNCH_ALERT &&
alertView.firstOtherButtonIndex==buttonIndex){
        //this URL kinda works.
http://maps.google.com/maps?daddr=Spot+1412@42,-
73&saddr=Current+Location@42,-72

        //if yes, store the destination's lat/lon for return launch and
start gps app.
        NSString* destination =[NSString
stringWithFormat:@"http://maps.google.com/maps?daddr=Spot+%d@%1.2f,%1.2
f&saddr=Current+Location@%1.2f,%1.2f", 1412, 41.343, -74.115, 43.124, -
72.31552];
        [[UIApplication sharedApplication] openURL:[NSURL
URLWithString:destination]];
    }
}
```

```
- (double) dot_prodX1:(double)x1 Y1:(double)y1 X2:(double)x2
Y2:(double)y2 {
    return x1*x2 + y1*y2;
}

-(double) dot_relative_to_P:(CLLocationCoordinate2D*)p
V:(CLLocationCoordinate2D*)v W:(CLLocationCoordinate2D*)w{
    return [self dot_prodX1:(*p).latitude - (*v).latitude
                        Y1:(*p).longitude - (*v).longitude
                        X2:(*v).latitude - (*w).latitude
                        Y2:(*v).longitude - (*w).longitude];
}

- (double) l_sqrdV:(CLLocationCoordinate2D*)v
W:(CLLocationCoordinate2D*) w{
    return ((*v).latitude-(*w).latitude)*((*v).latitude-(*w).latitude)
    + ((*v).longitude - (*w).longitude)*((*v).longitude -
(*w).longitude);
}

- (NSArray *)calloutBubblePlacement:(CLLocationCoordinate2D
*)selectionCenter withR:(CLLocationDistance) radius{
    //using the street information, snap to the street via geometric
projection


    CLLocation* center = [[CLLocation alloc]
initWithLatitude:(*selectionCenter).latitude
longitude:(*selectionCenter).longitude];

    //look through list of points, check spot distanceFromLocation
(coord) vs radius.
    NSArray* data = [self loadSpotData];
    //keep track of some stuff
    float avgLat=0, avgLon;

    for(id spot in data){
        NSArray* point = [spot componentsSeparatedByString:@","];
        CLLocation* spot_loc = [[CLLocation alloc]
initWithLatitude:[[point objectAtIndex:0] floatValue] longitude:[[point
objectAtIndex:1] floatValue]];

        CLLocationDistance dist = [spot_loc
distanceFromLocation:center];

        if(dist<radius){
            //inside the circle
            avgLat += spot_loc.coordinate.latitude;
            avgLon += spot_loc.coordinate.longitude;
```

```
        }
    }
    //compute the average point
    avgLat /= data.count;
    avgLon /= data.count;

    //project bubbles using this average and the spot's coordinates.


    return [[NSArray alloc] initWithObjects:
            [[NSDictionary alloc] initWithObjectsAndKeys:
             [[CalloutMapAnnotation alloc]
initWithLatitude:selectionCenter->latitude+0.0002

andLongitude:selectionCenter->longitude-0.0005
                                                    andTitle:@"1104"

andCorner:kBottomRightCorner], @"callout",
             [[CLLocation alloc] initWithLatitude:42.365154 longitude:-
71.110892], @"spot", nil],
            [[NSDictionary alloc] initWithObjectsAndKeys:
             [[CalloutMapAnnotation alloc]
initWithLatitude:selectionCenter->latitude-0.0001

andLongitude:selectionCenter->longitude-0.0005
                                                    andTitle:@"1106"

andCorner:kTopRightCorner], @"callout",
             [[CLLocation alloc] initWithLatitude:42.365049 longitude:-
71.110868], @"spot", nil],
            [[NSDictionary alloc] initWithObjectsAndKeys:
             [[CalloutMapAnnotation alloc]
initWithLatitude:selectionCenter->latitude-0.0004

andLongitude:selectionCenter->longitude-0.0005
                                                    andTitle:@"1108"

andCorner:kTopRightCorner], @"callout",
             [[CLLocation alloc] initWithLatitude:42.364993 longitude:-
71.110860], @"spot", nil],
            [[NSDictionary alloc] initWithObjectsAndKeys:
             [[CalloutMapAnnotation alloc]
initWithLatitude:selectionCenter->latitude+0.0003

andLongitude:selectionCenter->longitude+0.0005
                                                    andTitle:@"1101"

andCorner:kBottomLeftCorner], @"callout",
```

```
            [[CLLocation alloc] initWithLatitude:42.365140 longitude:-
71.110806], @"spot", nil],
            [[NSDictionary alloc] initWithObjectsAndKeys:
             [[CalloutMapAnnotation alloc]
initWithLatitude:selectionCenter->latitude

andLongitude:selectionCenter->longitude+0.0005
                                                    andTitle:@"1105"

andCorner:kBottomLeftCorner], @"callout",
            [[CLLocation alloc] initWithLatitude:42.365045 longitude:-
71.110790], @"spot", nil], nil];
}

- (void)clearMap {
    [self.map removeOverlays:self.map.overlays];
    [self.map removeAnnotations:self.map.annotations];
}

- (void)showSelectionCircle:(CLLocationCoordinate2D *)coord {
    // Assumes overlays and annotations were cleared in the calling
function
    [self.map setCenterCoordinate:*coord animated:YES];

    MKCircle *greyCircle = [MKCircle circleWithCenterCoordinate:*coord
radius:12];
    [greyCircle setColor:-1];
    [self.map addOverlay:greyCircle];


    NSArray *placement = [self calloutBubblePlacement:coord
withR:greyCircle.radius];

    if(calloutLines==NULL || callouts == NULL){
        calloutLines = [[NSMutableArray
alloc]initWithCapacity:MAX_CALLOUTS];
        callouts = [[NSMutableArray alloc]
initWithCapacity:MAX_CALLOUTS];

    }else{
        [self.map removeAnnotations:callouts];
        [self.map removeOverlays:calloutLines];
        [self.map removeOverlay:gCircle];

        [calloutLines removeAllObjects];
        [callouts removeAllObjects];
    }
    gCircle = greyCircle;
    for (NSDictionary *bubble in placement) {
        CLLocationCoordinate2D endpoints[2];
```

```
        CalloutMapAnnotation *callout = [bubble
objectForKey:@"callout"];
        endpoints[0] = callout.coordinate;
        endpoints[1] = ((CLLocation *)[bubble
objectForKey:@"spot"]).coordinate;
        MKPolyline *calloutLine = [MKPolyline
polylineWithCoordinates:endpoints count:2];
        [calloutLine setColor:-1];
        [callouts addObject:callout];
        [calloutLines addObject:calloutLine];
        [self.map addOverlay:calloutLine];
        [self.map addAnnotation:callout];

    }
}

- (void)showGridLevelWithCoordinates:(CLLocationCoordinate2D *)coord {
    [self clearMap];

    NSArray* data = [self loadGridData];

    for (id element in data) {
        NSArray *array = [[element objectForKey:@"se_corner"]
componentsSeparatedByString:@","];

        CLLocationCoordinate2D nw_point =
CLLocationCoordinate2DMake([[array objectAtIndex:0] floatValue],
[[array objectAtIndex:1] floatValue]);
        array = [[element
objectForKey:@"nw_corner"]componentsSeparatedByString:@","];
        CLLocationCoordinate2D se_point =
CLLocationCoordinate2DMake([[array objectAtIndex:0] floatValue],
[[array objectAtIndex:1] floatValue]);

        //calculate actual se_point using haversine.

        int color = [[element objectForKey:@"color"] intValue];

        CLLocationCoordinate2D ne_point =
CLLocationCoordinate2DMake(nw_point.latitude ,se_point.longitude);
        CLLocationCoordinate2D sw_point =
CLLocationCoordinate2DMake(se_point.latitude ,nw_point.longitude);


        CLLocationCoordinate2D testLotCoords[5]={nw_point, ne_point,
se_point, sw_point, nw_point};

        MKPolygon *commuterPoly1 = [MKPolygon
polygonWithCoordinates:testLotCoords count:5];
        [commuterPoly1 setColor:color];
```

```
        [self.map addOverlay:commuterPoly1];
    }
}

- (void)showStreetLevelWithCoordinates:(CLLocationCoordinate2D *)coord
{
    [self clearMap];

    NSArray* data = [self loadBlockData];

    for (id line in data) {
        NSArray *raw_waypoints = [[line objectForKey:@"line"]
componentsSeparatedByString:@";"];
        CLLocationCoordinate2D waypoints[raw_waypoints.count];
        int i=0;
        for (id raw_waypoint in raw_waypoints) {
            NSArray *coordinates = [raw_waypoint
componentsSeparatedByString:@","];
            CLLocationCoordinate2D coordinate =
CLLocationCoordinate2DMake([[coordinates objectAtIndex:0] floatValue],
[[coordinates objectAtIndex:1] floatValue]);
            waypoints[i++] = coordinate;
        }

        MKPolyline *routeLine = [MKPolyline
polylineWithCoordinates:waypoints count:raw_waypoints.count];

        int color = [[line objectForKey:@"color"] intValue];
        [routeLine setColor:color];

        [self.map addOverlay:routeLine];
    }
}


- (void)showSpotLevelWithCoordinates:(CLLocationCoordinate2D *)coord {
    [self clearMap];
    NSArray* data = [self loadSpotData];
    for(id spot in data){
        NSArray* point = [spot componentsSeparatedByString:@","];
        CLLocationCoordinate2D coord =
CLLocationCoordinate2DMake([[point objectAtIndex:0] floatValue],
[[point objectAtIndex:1] floatValue]);
        int color = [[point objectAtIndex:2] intValue];
        MKCircle* circle;
        if(color==0){
            circle  = [MKCircle circleWithCenterCoordinate:coord
radius:2];
        }else if(color==1){
```

```
            circle = [MKCircle circleWithCenterCoordinate:coord
radius:3];
        }
        [circle setColor:color];
        [self.map addOverlay:circle];
    }
}

- (void)showAvailabilitySelectionView {
    self.availabilitySelectionView.hidden = NO;
    self.topSpotSelectionView.hidden = YES;
    self.bottomSpotSelectionView.hidden = YES;
}

- (void)showSpotSelectionViews {
    self.topSpotSelectionView.hidden = NO;
    self.bottomSpotSelectionView.hidden = NO;
    self.availabilitySelectionView.hidden = YES;
}

- (void)mapView:(MKMapView *)mapView
regionDidChangeAnimated:(BOOL)animated {
    CLLocationCoordinate2D coord = mapView.centerCoordinate;
    if (mapView.region.span.latitudeDelta>=STREET_MAP_SPAN) {
        NSLog(@"GRID:currSpan: %f\n",
mapView.region.span.latitudeDelta);
        zoomState = kGridZoomLevel;
        [self showGridLevelWithCoordinates:&coord];
        [self showAvailabilitySelectionView];
    } else if (mapView.region.span.latitudeDelta>=SPOT_MAP_SPAN) {
        NSLog(@"Block:currSpan: %f\n",
mapView.region.span.latitudeDelta);
        zoomState = kStreetZoomLevel;
        [self showStreetLevelWithCoordinates:&coord];
        [self showAvailabilitySelectionView];
    } else {
        NSLog(@"currSpan: %f\n", mapView.region.span.latitudeDelta);
        zoomState = kSpotZoomLevel;
        [self showSpotSelectionViews];
    }
}

- (MKOverlayView *)mapView:(MKMapView *)myMapView
viewForOverlay:(id<MKOverlay>)overlay {
    if ([overlay isKindOfClass:[MKPolygon class]]) {
        MKPolygon* polygon = (MKPolygon*) overlay;
        MKPolygonView *view = [[MKPolygonView alloc]
initWithOverlay:polygon];
        view.lineWidth=1;
        view.strokeColor = [UIColor whiteColor];
```

```
        switch (polygon.color) {
            case 0:
                view.fillColor = [[UIColor veryLowAvailabilityColor]
colorWithAlphaComponent:0.2];
                break;
            case 1:
                view.fillColor = [[UIColor lowAvailabilityColor]
colorWithAlphaComponent:0.2];
                break;
            case 2:
                view.fillColor = [[UIColor mediumAvailabilityColor]
colorWithAlphaComponent:0.2];
                break;
            case 3:
                view.fillColor = [[UIColor highAvailabilityColor]
colorWithAlphaComponent:0.2];
                break;
            case 4:
                view.fillColor = [[UIColor veryHighAvailabilityColor]
colorWithAlphaComponent:0.2];
                break;
        }
        return view;
    } else if ([overlay isKindOfClass:[MKPolyline class]]) {
        MKPolylineView *view = [[MKPolylineView alloc]
initWithOverlay:overlay];
        view.lineWidth = 8;
        MKPolyline *polyline = (MKPolyline *)overlay;
        switch (polyline.color) {
            case -1:
                view.strokeColor = [UIColor blackColor];
                view.lineWidth = 1;
                break;
            case 0:
                view.strokeColor = [UIColor veryLowAvailabilityColor];
                break;
            case 1:
                view.strokeColor = [UIColor lowAvailabilityColor];
                break;
            case 2:
                view.strokeColor = [UIColor mediumAvailabilityColor];
                break;
            case 3:
                view.strokeColor = [UIColor highAvailabilityColor];
                break;
            case 4:
                view.strokeColor = [UIColor veryHighAvailabilityColor];
                break;
        }
        return view;
```

```
    } else if ([overlay isKindOfClass:[MKCircle class]]) {
        MKCircleView *circleView = [[MKCircleView alloc]
initWithCircle:(MKCircle *)overlay];
        MKCircle* circle = (MKCircle*) overlay;
        switch (circle.color) {
            case -1:
                // Grey selection circle
                circleView.fillColor = [[UIColor blackColor]
colorWithAlphaComponent:0.3];
                circleView.strokeColor = [UIColor whiteColor];
                //51 204 0

                circleView.lineWidth = 6;
                break;
            case 0:
                //taken
                circleView.fillColor = [[UIColor redColor]
colorWithAlphaComponent:0.9];
                circleView.strokeColor = [UIColor whiteColor];
                circleView.lineWidth = 2;
                break;
            case 1:
                //free
                circleView.fillColor = [UIColor colorWithRed:51.0/255
green:224.0/255 blue:0 alpha:0.8];;
                circleView.strokeColor = [UIColor whiteColor];
                circleView.lineWidth = 2;
                break;
        }
        return circleView;
    }
    return nil;
}

- (MKAnnotationView *)mapView:(MKMapView *)mapView
viewForAnnotation:(id <MKAnnotation>)annotation {
     if ([annotation isKindOfClass:[CalloutMapAnnotation class]]) {
            CalloutMapAnnotationView *calloutMapAnnotationView =
(CalloutMapAnnotationView *)[self.map
dequeueReusableAnnotationViewWithIdentifier:@"CalloutAnnotation"];
            if (!calloutMapAnnotationView) {
                  calloutMapAnnotationView = [[CalloutMapAnnotationView
alloc] initWithAnnotation:annotation


reuseIdentifier:@"CalloutAnnotation"];

            }
            calloutMapAnnotationView.mapView = self.map;
            return calloutMapAnnotationView;
```

```
    }

    return nil;
}

- (void)handleGesture:(UIGestureRecognizer *)gestureRecognizer
{
    if (gestureRecognizer.state != UIGestureRecognizerStateEnded)
        return;

    CGPoint touchPoint = [gestureRecognizer locationInView:self.map];
    CLLocationCoordinate2D coord = [self.map convertPoint:touchPoint
toCoordinateFromView:self.map];
    MKMapPoint mapPoint = MKMapPointForCoordinate(coord);

    if (zoomState==kGridZoomLevel) {
        for (id gridOverlay in self.map.overlays) {
            MKPolygonView *polyView = (MKPolygonView*)[self.map
viewForOverlay:gridOverlay];
            CGPoint polygonViewPoint = [polyView
pointForMapPoint:mapPoint];


            if (CGPathContainsPoint(polyView.path, NULL,
polygonViewPoint, NO)) {
                MKCoordinateRegion reg = [self.map
convertRect:polyView.bounds toRegionFromView:polyView];
                [self.map setRegion:[self.map regionThatFits:reg]
animated:YES];
                [self showStreetLevelWithCoordinates:&(reg.center)];
            }
        }
    } else if (zoomState==kStreetZoomLevel) {
        MKCoordinateRegion viewRegion =
MKCoordinateRegionMakeWithDistance(coord, METERS_PER_MILE/16,
METERS_PER_MILE/16);
        [self.map setRegion:[self.map regionThatFits:viewRegion]
animated:YES];
        [self showSpotLevelWithCoordinates:&coord];
        /*
         //ask user if that's desired destination
         UIAlertView* warningAlertView = [[UIAlertView alloc]
initWithTitle:@"Spot 4102 Selected!" message:@"Launch GPS to this
spot?" delegate:self cancelButtonTitle:@"Cancel"
otherButtonTitles:@"Launch" , nil];
         warningAlertView.tag = GPS_LAUNCH_ALERT;
         [warningAlertView show];
         */
    } else if (zoomState==kSpotZoomLevel) {
        if([gestureRecognizer numberOfTouches]==1){
```

```
            [self showSelectionCircle:&coord];
        }
    }
}

- (IBAction)topSpotBarTapped:(id)sender {
    if (((UISegmentedControl *)sender).selectedSegmentIndex>0 &&
((UISegmentedControl *)sender).selectedSegmentIndex<5) {
        NSLog(@"%@", @"Attempting segue");
        [self performSegueWithIdentifier:@"parkingSegue"
sender:sender];
    }
}
- (IBAction)bottomSpotBarTapped:(id)sender {
    if (((UISegmentedControl *)sender).selectedSegmentIndex>0 &&
((UISegmentedControl *)sender).selectedSegmentIndex<5) {
        NSLog(@"%@", @"Attempting segue");
        [self performSegueWithIdentifier:@"parkingSegue"
sender:sender];
    }
}

#pragma mark - Search bar and UISearchBarDelegate methods

- (void)setSearchBar:(UISearchBar *)searchBar active:(BOOL)visible {
    if (visible) {
        if (zoomState == kSpotZoomLevel) {
            self.disableViewOverlay.frame =
CGRectMake(0.0f,88.0f,320.0f,372.0f);
        } else {
            self.disableViewOverlay.frame =
CGRectMake(0.0f,44.0f,320.0f,416.0f);
        }
        [self.view addSubview:self.disableViewOverlay];

        [UIView animateWithDuration:0.25 animations:^{
            self.navigationBar.leftBarButtonItem = nil;
            self.disableViewOverlay.alpha = 0.8;
            if (zoomState == kSpotZoomLevel) {
                searchBar.frame = CGRectMake(0, 0, 320, 88);
            } else {
                searchBar.frame = CGRectMake(0, 0, 320, 44);
            }
        }];
    } else {
        [searchBar resignFirstResponder];

        [UIView animateWithDuration:0.25 animations:^{
            self.navigationBar.leftBarButtonItem = self.leftBarButton;
            self.disableViewOverlay.alpha = 0;
```

```
            searchBar.frame = CGRectMake(45, 0, 275, 44);
        } completion:^(BOOL s){
            [self.disableViewOverlay removeFromSuperview];
        }];
    }
    [searchBar setShowsScopeBar:(visible &&
zoomState==kSpotZoomLevel)];
    [searchBar setShowsCancelButton:visible animated:YES];
}

- (void)handleSingleTap:(UIGestureRecognizer *)sender {
    [self setSearchBar:(UISearchBar *)self.topSearchBar active:NO];
}

- (void)searchBarSearchButtonClicked:(UISearchBar *)searchBar {
    [self clearMap];

    [geocoder geocodeAddressString:searchBar.text inRegion:nil
completionHandler:^(NSArray *placemarks, NSError * error){
        CLLocation* locationObject = [[placemarks objectAtIndex:0]
location];

        MKCoordinateRegion viewRegion = [self.map
regionThatFits:MKCoordinateRegionMakeWithDistance(locationObject.coordi
nate, 0.5*METERS_PER_MILE, 0.5*METERS_PER_MILE)];
        [self.map setRegion:viewRegion animated:YES];
    }];

    [self setSearchBar:searchBar active:NO];
}

- (void)searchBar:(UISearchBar *)searchBar
selectedScopeButtonIndexDidChange:(NSInteger)selectedScope {
    if (selectedScope == 1) {
        searchBar.keyboardType = UIKeyboardTypeNumberPad;
    } else {
        searchBar.keyboardType = UIKeyboardTypeDefault;
    }
    [searchBar resignFirstResponder];
    [searchBar becomeFirstResponder];
}

- (void)searchBarBookmarkButtonClicked:(UISearchBar *)searchBar {
    //load recently parked spots.

}

- (void)searchBarTextDidBeginEditing:(UISearchBar *)searchBar {
    [self setSearchBar:searchBar active:YES];
}
```

```
- (void)searchBarCancelButtonClicked:(UISearchBar *)searchBar {
    [self setSearchBar:searchBar active:NO];
}

#pragma mark - Debug button actions
- (IBAction)gridButtonPressed:(id)sender {
    NSLog(@"Grid Button Pressed\n" );
    zoomState = kGridZoomLevel;

    //set the zoom to fit 12 grids perfectly
    CLLocationCoordinate2D point =
CLLocationCoordinate2DMake(42.365045,-71.118965);

    MKCoordinateRegion viewRegion =
MKCoordinateRegionMakeWithDistance(point, METERS_PER_MILE,
METERS_PER_MILE);  //this is essentially zoom level in android

    MKCoordinateRegion adjustedRegion = [map
regionThatFits:viewRegion];

    [self.map setRegion:adjustedRegion animated:YES];
    [self showGridLevelWithCoordinates:&point];
}

- (IBAction)streetButtonPressed:(id)sender {
    CLLocationCoordinate2D point = {42.364854,-71.109438};

    MKCoordinateRegion viewRegion =
MKCoordinateRegionMakeWithDistance(point, METERS_PER_MILE/2,
METERS_PER_MILE/2);
    [self.map setRegion:[self.map regionThatFits:viewRegion]
animated:YES];
    [self showStreetLevelWithCoordinates:&point];
}
- (IBAction)spotButtonPressed:(id)sender {
    CLLocationCoordinate2D point = {42.365077, -71.110838};

    MKCoordinateRegion viewRegion =
MKCoordinateRegionMakeWithDistance(point, METERS_PER_MILE/16,
METERS_PER_MILE/16);
    [map setRegion:[map regionThatFits:viewRegion] animated:YES];

    [self showSpotLevelWithCoordinates:&point];
}
- (IBAction)noneButtonPressed:(id)sender {
    [self clearMap];
}
```

```
- (void)locationManager:(CLLocationManager *)manager
didUpdateToLocation:(CLLocation *)newLocation fromLocation:(CLLocation
*)oldLocation{
    if (MAX(newLocation.horizontalAccuracy,
newLocation.verticalAccuracy) < 100) {
        //One location is obtained.. just zoom to that location
        /*
         MKCoordinateRegion region;
         region.center=newLocation.coordinate;
         //Set Zoom level using Span
         MKCoordinateSpan span;
         span.latitudeDelta=.005;
         span.longitudeDelta=.005;
         region.span=span;

         [map setRegion:region animated:TRUE];
         */

        [self spotButtonPressed:nil];

        [locationManager stopUpdatingLocation];
    }
}

#pragma mark - Memory

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Release any cached data, images, etc that aren't in use.
}

#pragma mark - View lifecycle

- (void)viewDidLoad{
    [super viewDidLoad];

    //check the current zoom level to set the ZOOM_STATE integer.
    if (self.map.bounds.size.width > STREET_MAP_SPAN) {
        zoomState = kGridZoomLevel;
        //above middle zoom level
    } else if (SPOT_MAP_SPAN) {
        //above spot zoom level
        zoomState = kStreetZoomLevel;
    } else {
        //inside spot zoom level.
        zoomState = kSpotZoomLevel;
    }
    self.disableViewOverlay = [[UIView alloc]
```

```
initWithFrame:CGRectMake(0.0f,88.0f,320.0f,372.0f)];
    self.disableViewOverlay.backgroundColor=[UIColor blackColor];
    self.disableViewOverlay.alpha = 0;
    UITapGestureRecognizer *singleTap = [[UITapGestureRecognizer alloc]
initWithTarget:self action:@selector(handleSingleTap:)];
    [self.disableViewOverlay addGestureRecognizer:singleTap];

    ((UISearchBar *)self.topSearchBar).scopeButtonTitles = [[NSArray
alloc] initWithObjects:@"Place name", @"Spot number", nil];

    self.leftBarButton = self.navigationBar.leftBarButtonItem;

    [self.topSpotSelectionBar setWidth:36 forSegmentAtIndex:0];
    [self.topSpotSelectionBar setWidth:36 forSegmentAtIndex:5];
    [self.bottomSpotSelectionBar setWidth:36 forSegmentAtIndex:0];
    [self.bottomSpotSelectionBar setWidth:36 forSegmentAtIndex:5];

    // Do any additional setup after loading the view, typically from a
nib.
    self.map.delegate=self;
    self.map.showsUserLocation = YES;

    //setup gesture recognizer for grids and blocks and spots.
    UITapGestureRecognizer *tgr = [[UITapGestureRecognizer alloc]
                                   initWithTarget:self
action:@selector(handleGesture:)];
    [map addGestureRecognizer:tgr];

    //SETUP LOCATION manager
    locationManager=[[CLLocationManager alloc] init];
    locationManager.delegate=self;

locationManager.desiredAccuracy=kCLLocationAccuracyNearestTenMeters;

    [locationManager startUpdatingLocation];
}

- (void)viewDidUnload
{
    [self setMap:nil];
    [self setNavigationBar:nil];
    [self setTopSearchBar:nil];
    [self setBottomSpotSelectionBar:nil];
    [self setTopSpotSelectionBar:nil];
    [self setAvailabilitySelectionView:nil];
    [self setBottomSpotSelectionView:nil];
    [self setTopSpotSelectionView:nil];
    [super viewDidUnload];
    // Release any retained subviews of the main view.
```

```
    // e.g. self.myOutlet = nil;
}

- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];
}

- (void)viewDidAppear:(BOOL)animated
{
    [super viewDidAppear:animated];
    //prepare geocoder upon view load.
    if(geocoder ==nil){
        geocoder = [[CLGeocoder alloc] init];
    }
}

- (void)viewWillDisappear:(BOOL)animated
{
      [super viewWillDisappear:animated];

    [locationManager stopUpdatingLocation];
}

- (void)viewDidDisappear:(BOOL)animated
{
      [super viewDidDisappear:animated];
}

-
(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)in
terfaceOrientation
{
    // Return YES for supported orientations
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}

@end
```

Appendix C: PQParkingViewController.m

```
//
//  PQParkingViewController.m
//  Parq
//
//  Created by Mark Yen on 4/26/12.
//  Copyright (c) 2012 Massachusetts Institute of Technology. All
rights reserved.
//

#import "PQParkingViewController.h"
#import "UIColor+Parq.h"

@interface PQParkingViewController ()
@property (nonatomic) BOOL timerStarted;
@property (strong, nonatomic) UIBarButtonItem *cancelButton;
@property (strong, nonatomic) UIBarButtonItem *doneButton;
@end

@implementation PQParkingViewController
@dynamic rate;
@dynamic address;
@synthesize rateNumeratorCents;
@synthesize rateDenominatorMinutes;
@synthesize limit;
@synthesize limitUnit;
@synthesize addressLabel;
@synthesize startButton;
@synthesize unparkButton;
@synthesize paygFlag;
@synthesize prepaidFlag;
@synthesize hours;
@synthesize minutes;
@synthesize paygCheck;
@synthesize prepaidCheck;
@synthesize prepaidAmount;
@synthesize paygView;
@synthesize addressView;
@synthesize prepaidView;
@synthesize seeMapView;
@synthesize datePicker;
@synthesize rateNumerator;
@synthesize rateDenominator;
@synthesize limitValue;
@synthesize cancelButton;
@synthesize doneButton;
@synthesize timerStarted;

#pragma mark - Main button actions
```

```
- (IBAction)startTimer:(id)sender {
    timerStarted = YES;
    paygView.hidden = YES;
    addressView.hidden = NO;
    prepaidView.hidden = YES;
    seeMapView.hidden = NO;
    ((UIButton *)sender).hidden = YES;
    unparkButton.hidden = NO;
    [self.tableView cellForRowAtIndexPath:[NSIndexPath
indexPathForRow:0 inSection:0]].userInteractionEnabled = NO;
    [self.tableView cellForRowAtIndexPath:[NSIndexPath
indexPathForRow:1 inSection:0]].accessoryType =
UITableViewCellAccessoryDisclosureIndicator;
    [self.tableView reloadData];
}

- (IBAction)unparkNow:(id)sender {
    timerStarted = NO;
    paygView.hidden = NO;
    addressView.hidden = YES;
    prepaidView.hidden = NO;
    seeMapView.hidden = YES;
    startButton.hidden = NO;
    ((UIButton *)sender).hidden = YES;
    [self.tableView cellForRowAtIndexPath:[NSIndexPath
indexPathForRow:0 inSection:0]].userInteractionEnabled = YES;
    [self.tableView cellForRowAtIndexPath:[NSIndexPath
indexPathForRow:1 inSection:0]].accessoryType =
UITableViewCellAccessoryNone;
    [self.tableView reloadData];
}

#pragma mark - Date Picker control
- (void)durationChanged {
    int totalMinutes = (datePicker.countDownDuration-1)/60+1;
    if (totalMinutes > limit) {
        datePicker.countDownDuration = limit*60;
        totalMinutes = limit;
    }
    int hoursPart = totalMinutes/60;
    int minutesPart = totalMinutes%60;
    hours.text = [NSString stringWithFormat:@"%02d", hoursPart];
    minutes.text = [NSString stringWithFormat:@"%02d", minutesPart];

    int totalCents = self.rate * totalMinutes;
    int dollarsPart = totalCents/100;
    int centsPart = totalCents%100;

    if (hoursPart == 0) {
```

```
        prepaidAmount.text = [NSString stringWithFormat:@"%dm
($%d.%02d)", minutesPart, dollarsPart, centsPart];
    } else {
        prepaidAmount.text = [NSString stringWithFormat:@"%dh %02dm
($%d.%02d)", hoursPart, minutesPart, dollarsPart, centsPart];
    }
}

- (void)activatePicker {
    [UIView animateWithDuration:.25 animations:^{
        self.tableView.frame = CGRectMake(0, -131, 320, 611);
        datePicker.frame = CGRectMake(0, 2, 320, 216);
    }];
    self.navigationItem.title = @"Enter Amount";
    if (cancelButton == nil) {
        cancelButton = [[UIBarButtonItem alloc]
initWithBarButtonSystemItem:UIBarButtonSystemItemCancel target:self
action:@selector(cancelPicker)];
    }
    if (doneButton == nil) {
        doneButton = [[UIBarButtonItem alloc]
initWithBarButtonSystemItem:UIBarButtonSystemItemDone target:self
action:@selector(doneWithPicker)];
    }
    self.navigationItem.leftBarButtonItem = cancelButton;
    self.navigationItem.rightBarButtonItem = doneButton;
    [self durationChanged];
}

- (void)resignPicker {
    [UIView animateWithDuration:.25 animations:^{
        self.tableView.frame = CGRectMake(0, 0, 320, 416);
        datePicker.frame = CGRectMake(0, 89, 320, 216);
    }];
    self.navigationItem.title = @"1106, Cambridge, MA";
    self.navigationItem.leftBarButtonItem = nil;
    self.navigationItem.rightBarButtonItem = nil;
}

#pragma mark - Table View actions
- (void)paygSelected {
    paygCheck.image = [UIImage imageNamed:@"check.png"];
    prepaidCheck.image = [UIImage imageNamed:@"check_empty.png"];
    paygFlag.hidden = NO;
    prepaidFlag.hidden = YES;
    prepaidAmount.text = @"Enter Amount";
    prepaidAmount.textColor = [UIColor disabledTextColor];
    hours.text = @"00";
    minutes.text = @"00";
    [self resignPicker];
```

```
}

- (void)prepaidSelected {
    paygCheck.image = [UIImage imageNamed:@"check_empty.png"];
    prepaidCheck.image = [UIImage imageNamed:@"check.png"];
    paygFlag.hidden = YES;
    prepaidFlag.hidden = NO;
    prepaidAmount.textColor = [UIColor whiteColor];
    [self activatePicker];
}

#pragma mark - Bar Button actions
- (void)cancelPicker {
    NSIndexPath *paygIndexPath = [NSIndexPath indexPathForRow:0
inSection:0];
    [self.tableView selectRowAtIndexPath:paygIndexPath animated:NO
scrollPosition:UITableViewScrollPositionNone];
    [self paygSelected];
    [self.tableView deselectRowAtIndexPath:paygIndexPath animated:YES];
}

- (void)doneWithPicker {
    [self.tableView deselectRowAtIndexPath:[NSIndexPath
indexPathForRow:1 inSection:0] animated:YES];
    prepaidAmount.textColor = [UIColor activeTextColor];
    [self resignPicker];
}

#pragma mark - UITableViewDelegate
- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
    if (!timerStarted) {
        if (indexPath.section == 0) {
            if (indexPath.row == 0) {
                [self paygSelected];
                [tableView deselectRowAtIndexPath:indexPath
animated:YES];
            } else if (indexPath.row == 1) {
                [self prepaidSelected];
            }
        }
    } else {
        [tableView deselectRowAtIndexPath:indexPath animated:YES];
    }
}

- (NSString *)tableView:(UITableView *)tableView
titleForHeaderInSection:(NSInteger)section {
    if (!timerStarted) {
        return @"Select payment method";
```

```
    } else {
        return @"Your car\u2019s parking location";
    }
}

- (UIView *)tableView:(UITableView *)tableView
viewForHeaderInSection:(NSInteger)section {
      UIView *containerView = [[UIView alloc]
initWithFrame:CGRectMake(0, 0, 320, 40)];
      containerView.backgroundColor = [UIColor
groupTableViewBackgroundColor];
      CGRect labelFrame = CGRectMake(20, 2, 320, 30);
      UILabel *label = [[UILabel alloc] initWithFrame:labelFrame];
      label.backgroundColor = [UIColor clearColor];
      label.font = [UIFont boldSystemFontOfSize:17];
      label.shadowColor = [UIColor colorWithWhite:1.0 alpha:1];
      label.shadowOffset = CGSizeMake(0, 1);
      label.textColor = [UIColor colorWithRed:0.265 green:0.294
blue:0.367 alpha:1.000];
      label.text = [self tableView:tableView
titleForHeaderInSection:section];
      [containerView addSubview:label];

    if (!timerStarted) {
        UIButton *abutton = [UIButton buttonWithType:
UIButtonTypeInfoDark];
        abutton.frame = CGRectMake(288, 12, 14, 14);
        //    [abutton addTarget: self action: @selector(addPage:)
        //      forControlEvents: UIControlEventTouchUpInside];
        [containerView addSubview:abutton];
    }
      return containerView;
}

- (CGFloat)tableView:(UITableView *)tableView
heightForHeaderInSection:(NSInteger)section {
    return 36;
}

#pragma mark - Dynamic properties
- (double)rate {
    return (double)rateNumeratorCents/rateDenominatorMinutes;
}

- (NSString *)address {
    return addressLabel.text;
}

- (void)setAddress:(NSString *)addressString {
    addressLabel.text = addressString;
```

```
}

#pragma mark - Memory

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Release any cached data, images, etc that aren't in use.
}

#pragma mark - View lifecycle

- (void)viewDidLoad{
    [super viewDidLoad];

    // Hardcoding the rate and limit for now
    rateNumeratorCents = 50;
    rateDenominatorMinutes = 30;
    limit = 600; // 10 hours * 60 min/hr

    timerStarted = NO;

    [startButton setBackgroundImage:[[UIImage
imageNamed:@"green_button.png"]
resizableImageWithCapInsets:UIEdgeInsetsMake(0, 14, 0, 14)]
forState:UIControlStateNormal];
    [unparkButton setBackgroundImage:[[UIImage
imageNamed:@"red_button.png"]
resizableImageWithCapInsets:UIEdgeInsetsMake(0, 14, 0, 14)]
forState:UIControlStateNormal];

    [datePicker addTarget:self action:@selector(durationChanged)
forControlEvents:UIControlEventValueChanged];

    hours.font = [UIFont fontWithName:@"OCR B Std" size:60];
    minutes.font = [UIFont fontWithName:@"OCR B Std" size:60];

    // rateNumerator
    int dollarsPart = rateNumeratorCents/100;
    int centsPart = rateNumeratorCents%100;
    if (dollarsPart == 0) {
        rateNumerator.text = [NSString stringWithFormat:@"%d¢",
centsPart];
    } else if (centsPart == 0) {
        rateNumerator.text = [NSString stringWithFormat:@"$%d",
dollarsPart];
    } else {
        rateNumerator.text = [NSString stringWithFormat:@"$%d.%02d",
dollarsPart, centsPart];
    }
```

```
    // rateDenominator
    int hoursPart = rateDenominatorMinutes/60;
    int minutesPart = rateDenominatorMinutes%60;
    if (hoursPart == 0) {
        rateDenominator.text = [NSString stringWithFormat:@"/%dmin",
minutesPart];
    } else {
        if (minutesPart == 0) {
            if (hoursPart == 1) {
                rateDenominator.text = @"/hour";
            } else {
                rateDenominator.text = [NSString stringWithFormat:@"/%d
hrs", hoursPart];
            }
        } else {
            rateDenominator.text = [NSString stringWithFormat:@"/%dh
%02dm", hoursPart, minutesPart];
        }
    }

    // limitValue and limitUnit
    if (limit<60) {
        limitValue.text = [NSString stringWithFormat:@"%d", limit];
        limitUnit.text = @"mins";
    } else if (limit>60) {
        if (limit%60 == 0) {
            limitValue.text = [NSString stringWithFormat:@"%d",
limit/60];
        } else {
            limitValue.text = [NSString stringWithFormat:@"%.1f",
limit/60.0];
        }
        limitUnit.text = @"hours";
    } else {
        limitValue.text = @"1";
        limitUnit.text = @"hour";
    }
}

- (void)viewDidUnload
{
    [self setPaygFlag:nil];
    [self setHours:nil];
    [self setMinutes:nil];
    [self setStartButton:nil];
    [self setPrepaidFlag:nil];
    [self setPaygCheck:nil];
    [self setPrepaidCheck:nil];
    [self setPrepaidAmount:nil];
```

```
    [self setPaygView:nil];
    [self setAddressView:nil];
    [self setPrepaidView:nil];
    [self setSeeMapView:nil];
    [self setUnparkButton:nil];
    [self setDatePicker:nil];
    [self setRateNumerator:nil];
    [self setRateDenominator:nil];
    [self setLimitValue:nil];
    [self setLimitUnit:nil];
    [self setAddressLabel:nil];
    [super viewDidUnload];
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}

- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];
}

- (void)viewDidAppear:(BOOL)animated
{
    [super viewDidAppear:animated];
}

- (void)viewWillDisappear:(BOOL)animated
{
      [super viewWillDisappear:animated];
}

- (void)viewDidDisappear:(BOOL)animated
{
      [super viewDidDisappear:animated];
}

-
(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)in
terfaceOrientation
{
    // Return YES for supported orientations
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
@end
```

Bibliography

Binder, C. R., Quirici, R., Domnitcheva, S., & Staubli, B. (2008). Smart labels for waste and resource management - An integrated assessment. [Article]. *Journal of Industrial Ecology, 12*(2), 207-228. doi: 10.1111/j.1530-9290.2008.00016.x

Bureau of Labor Statistics. (2011). Occupational Employment Statistics. Retrieved from http://www.bls.gov/oes/

Kelly, J. A., & Clinch, J. P. (2006). Influence of varied parking tariffs on parking occupancy levels by trip purpose. [Article]. *Transport Policy, 13*(6), 487-495. doi: 10.1016/j.tranpol.2006.05.006

Lee, J. A., Thomas, V. M., & ieee. (2004). GPS and radio tracking of end-of-life products. [Proceedings Paper]. *Proceedings of the 2004 Ieee International Symposium on Electronics & the Environment, Conference Record*, 309-312.

Miller, M. J. (2012). Intel Enters Smartphone Chip Race For Real. Retrieved from http://forwardthinking.pcmag.com/ces/292745-intel-enters-smartphone-chip-race-for-real

Shoup, D. C., & American Planning Association. (2005). *The high cost of free parking*. Chicago: Planners Press, American Planning Association.

Smith, A. (2012). 46% of American adults are smartphone owners (pp. 9). Washington, D.C.: Pew Research Center's Internet & American Life Project.