Data Collection and Management of a Mobile Sensor Platform

by

Abraham M. Rosenfeld
S.B., C.S. Massachusetts Institute of Technology, 2010

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
February 2013

Author: ______________________
Department of Electrical Engineering and Computer Science
February 1, 2013

Certified by: ______________________
[Prof. Sanjay Sarma] Thesis Supervisor
February 1, 2013

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

[THIS PAGE INTENTIONALLY LEFT BLANK]

Data Collection and Management of a Mobile Sensor Platform

by

Abraham M. Rosenfeld
S.B., C.S. Massachusetts Institute of Technology, 2010

February 2013
In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis explores the development of a platform to better collect and manage data from multiple senor inputs mounted on a car sensor platform. Specifically, focusing on the collection and synchronization of multiple forms of data across a single mobile sensor system. The project will be implemented for three versions of a light-sensing platform, and will cover the different methods of data collection and different types of sensor devices implemented in each version. It will also cover the different technical challenges faced when collecting and managing data across multiple mobile sensors.

[THIS PAGE INTENTIONALLY LEFT BLANK]

Acknowledgments

I would like to thank Professor Sanjay Sarma for providing me with the guidance and support throughout the entire project, as well as the MIT Mechanical Engineering Department and CSAIL Department for providing me with the resources and knowledge needed to complete this task. Sumeet Kumar, Stephen Ho, Benjamin Green, and Joshua Siegel also provided invaluable support and insight throughout the life of this project.

Finally I would like to thank my family and friends for their unconditional love, support, and guidance throughout my entire academic career.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

In the modern world, data has all but become a commodity. However, the collection and management of data is still an imperfect practice. Whether one is collecting data from residential homes, large buildings, cars, trains, public transport, or even entire cities, there are many ways to collect and manage data.

The collection of sensor data may appear to be a trivial solution. Sensors are made to be data collection devices. However, in practice, data collection from multiple sensors can be extremely challenging. Usually one is trying to collect data from an existing system. For example: the building has already been built, the car already made, and the city lights already standing; in these examples the infrastructure is already built. The systems from which data is being collected are primarily static and unchanging, making it difficult to integrate a data collection system with a potentially inflexible system.

These inflexibilities can arise in many different ways. In buildings they may arise from an existing archaic Building Management Software (BMS) that is nearly impossible to communicate with. In cars it could be that the On Board Diagnostic

(OBD) system doesn't allow collection of a data point one is looking for. In city street lighting, specific to this research, it could arise from lights just being widely distributed throughout a city.

Having lights spatially distributed brings up a major design decision for a data collection system. In a distributed system of lights, does one collect data by distributing a network of static sensors throughout the city, or does one create a mobile sensor platform and mount it to city vehicle for data collection? Both solutions have their pros and cons. A distributed network gives consistent data, but it is expensive to deploy and maintain. A mobile senor platform is cheaper and easier to upkeep, but hard to gain consistent and synchronized data.

1.2 Mobile Sensor Platform

In this project we chose to develop a mobile sensor platform as we considered it a more feasible approach for wide-area coverage needed for city sensing. The pros of having an easily upgradable and deployable system far outweigh the cons of the complexity in collecting and managing data within a mobile solution. Along with this, city municipalities often have fleets of vehicles to allow for this type of solution to be possible.

Mobile sensor platforms are easily upgradable, as opposed to spatially a distributed network of sensors, because there are fewer sensors to maintain. If one were to distribute light sensors throughout a city for every single streetlight, then upgrading the sensors would be just as cumbersome as initially deploying them. With a mobile solution one is only dealing with a small fraction of the sensors, and for the same reason, deploying the mobile system is far easier.

Mobile sensor platforms do complicate the collection and management of data. With a distributed network of sensors one would know exactly when and where the data was collected because the sensors are always in the same place. With a mobile sensor platform, other information is needed in addition to the data of interest. GPS location must be tagged and synchronized with the data, as well as any other information needed to gain a more accurate location measurement. On top of this, all data must be accurately time-synchronized to allow accurate post-processing of the data.

Even with the complexities of a mobile sensor platform, for city sensing projects, mobile sensor platforms prove to be the more robust solution. With mobile sensor solutions one could crowd-source multiple systems using fleet vehicles in a city to gain higher accuracy of street lighting levels, much like crowd-sourced geo-tagged social media can be used to gain insight in disaster relief operations (Gao et al 2011). Overall, the advantages of building an accurate mobile sensor solution are immense.

1.3 Objective

The objective of this thesis was to study and implement data management in a mobile sensor platform for assessing the quality of street lighting in cities. The platform was required to collect accurate and synchronized data. Along with the collection, data management methods had to be developed to allow easy integration with post collection analysis. A complete standalone system had to be developed in order to possibly transition to a real world application.

Chapter 2

Background

2.1 Mobile Sensors

Mobile sensing has grown exponentially over recent years. With the emergence of technologies like *Google Maps Street View,* massive amounts of data have been collected in all facets of life (Anguelov et al 2010). Specifically, in the Field Intelligence Lab (FIL) at MIT, there have been two projects that I have worked closely with. The first is the thermal imaging system headed by Long Phan and Jonathan Jesneck. The project was based on mounting infrared cameras on top of a car and building thermal images of cities to provide useful information on energy management for homeowners. The project's insight in data synchronization and location correction immensely aided in the research for this thesis. A lot of what was done with data management was bootstrapped from their successful project.

The second project from the FIL that I worked closely with was the CloudCar project headed by Joshua Siegel and Eric Wilhelm. The project was based on collecting car diagnostic information in real time and sending that information over the cell network to a central server. The project's insight in multiple sensor data collection and storage aided the direction I took for this thesis project.

2.2 Light Sensing Solutions

Street lighting is an area of high concern for cities experiencing financial setbacks. Street lighting is expensive, and it is important that cities are optimizing the amount of light they are providing while adhering to known lighting standards. Streetlight levels have been tied to enhanced safety levels in residential areas (Hamsa et al 2009). However, it has gotten to a point where cities just shut streetlights off for a period of time at night to save money, rather than attempting to better understand how much light they are actually supplying and then analyzing a form of optimization. The main reason they are unable to understand their current streetlight situation is that they do not have an easy way of measuring light levels.

Throughout the course of this project we have encountered a number of existing solutions to street light management in Spain and the United Kingdom. We have spoken with numerous city officials on the standards and current solutions employed to address street light management. The most common practices are as follows:

- Having a city official drive scheduled routes and observe the status of streetlights.
- Deploying static light sensors throughout the city on streetlight poles (Jing et all 2007).

- Having city officials block off a road and measure the light levels with lux meters at ground level. This method is used to measure light levels of roads against set standards (European Standard 2003).

The first practice is the most common. The human observation is used to measure the light levels in a city. This is not ideal for a number of reasons. First, it is tedious and expensive to hire somebody to monitor lights in a city. They must drive multiple routes over multiple days to cover the entire city. Secondly, it is an inconsistent and inaccurate measuring tool. Every person may perceive acceptable light levels differently. And lastly, it is not guaranteed to fit to the city/state standards of lighting levels. Every city must achieve a stated level of street lighting, however, it is nearly impossible for a human being to measure light to the degree of one or two lux consistently.

The second practice is better at accurately measuring light levels, but has its own drawbacks. Firstly, the light sensors are placed on the actual street light pole a few meters below the light source. Although this measures the light correctly at that point, we are interested in the lighting levels at street level. Secondly, the amount of upkeep it takes to make sure the sensors are working properly may be just as cumbersome as if they had somebody driving routes and analyzing the lights. There are systems that exist that allow for monitoring and controlling of streetlights (Huang et al 2004), however these systems tell us little about the performance of the streetlight's past whether it is on or off.

The third practice is the most accurate form of measurement. However, blocking off a road is costly and slow. It would take a city official many months to block off individual roads at night in order to measure the light levels with lux meters for the entire city. However, this is the current test city officials use to make sure streetlight levels are adhering to the standards (European Standard 2003).

There exists an opportunity for a fast and inexpensive way of measuring streetlight levels within high accuracy in cities. This is the opportunity our project tried to address.

2.3 Proposed Solution

This project aimed to develop a mobile sensor platform for measuring streetlight levels in a city. The mobile sensor platform had to be inexpensive and accurate enough to compare the measured data against city standards of streetlight levels. The prototype developed must be standalone and supply data in a way such that post-processing analysis could be done easily. If a cheap standalone solution could be developed, a crowd-sourced solution using a fleet of vehicles could be used to monitor the streetlight levels in a city.

Chapter 3

Design Methodology

3.1 Vision and Milestones

Creating a system that would be able to collect synchronized light, video, and location data was the overarching goal from the beginning of this project. The sensors needed to achieve this would be luminosity sensors that measured the light levels at a given point, video frames that could be analyzed to determine the type of lamp and location of a streetlight, and location sensors that could be used to collect very accurate information on where the other data points were taken. Initially, the system was to be developed as a proof of concept. Could multiple devices be mounted on a car to produce accurate and usable data? From the proof of concept, the two main goals were to achieve better accuracy and to create a standalone system. The broad milestones were as follows:

- Collecting and synchronizing GPS, IMU, OBD-II, video, and light data onto a PC
- Improving the location and light data accuracy
- Migrating the system to run through a microcontroller
- Storing the collected data on a SD card through a microcontroller

The initial milestone was to be a proof of concept. Improving the accuracy of the data was needed to achieve desired accuracy of the system. Microcontroller migration was needed to achieve a standalone system.

Chapter 4

Version 1: Proof of Concept

4.1 Design

Version 1 was the initial proof of concept. The goal was to design a car mounted platform that would collect video and light data and synchronize it with location data. In order to do so, we designed the platform to have 4 inexpensive cameras mounted in a semicircle to cover the 180-degree plane above the car, as well as 3 light sensors arranged to cover the same plane. **Figure 1** is a flowchart showing the dataflow of Version 1:

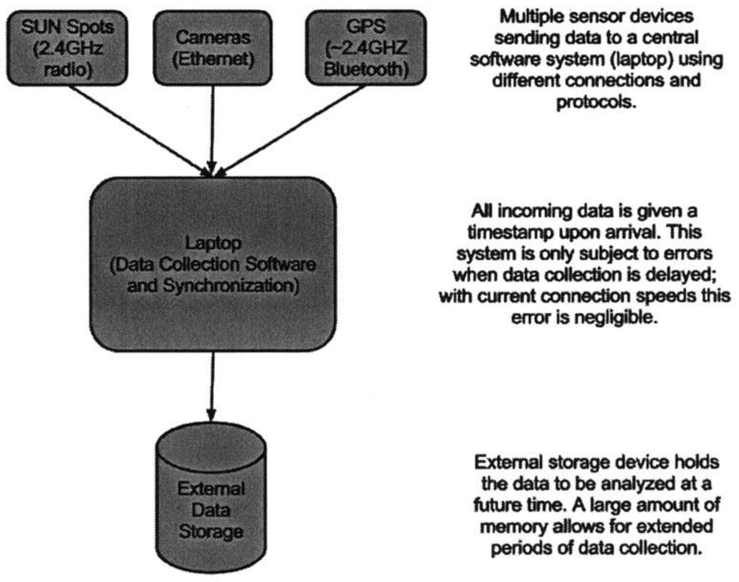

SUN Spots (2.4GHz radio)
Cameras (Ethernet)
GPS (~2.4GHZ Bluetooth)
Multiple sensor devices sending data to a central software system (laptop) using different connections and protocols.
Laptop (Data Collection Software and Synchronization)
All incoming data is given a timestamp upon arrival. This system is only subject to errors when data collection is delayed; with current connection speeds this error is negligible.
External Data Storage
External storage device holds the data to be analyzed at a future time. A large amount of memory allows for extended periods of data collection.

Figure 1: Version 1 flowchart

The light sensors (SUN Spots), cameras, and GPS were all connected by wireless, Ethernet, and Bluetooth protocols respectively. The devices would relay information over communication ports on a PC, and the PC would be running simple applications to collect and store the incoming data. **Figure 2** shows a picture of the actual mount:

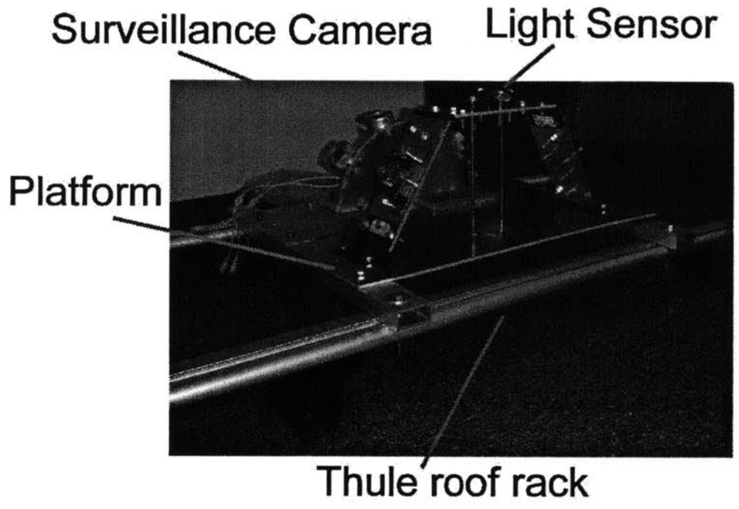


Figure 2: Version 1

4.2 Sensors

The sensors used are as follows:

GPS: BT-Q818XT

Accuracy: 3 – 15 m

Sampling rate: 10 Hz

Cost: $ 73.95/unit

Figure 3: QStarz BT-Q818XT (source: http://www.qstarz.com/Products/GPS%20Products/BT-Q818XT-F.htm)

This GPS unit communicates over Bluetooth at 10 Hz and has an accuracy range from 3-15 meters. The BT-Q818XT GPS unit was selected on its relatively inexpensive cost and level of accuracy. It is not the most accurate GPS on the market; however for the application it is very affordable and easily used as a proof of concept. It also has a high sampling rate to allow for greater spatial resolution.

Light Sensor: Sun Spot

Resolution: ~2 Lux

Sampling rate: 10 Hz

Cost: $ 100.00 /unit

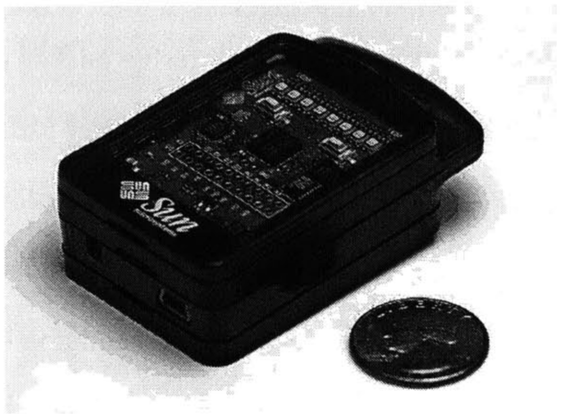


Figure 4: SunSpot (source: http://www.sunspotworld.com/)

In this version we had 3 of these light sensors to capture light from directly above and from each side. The light sensor has a resolution of about 2 lux and a sampling rate of 10 Hz. Again, for proof of concept this sensor allowed for fairly easy integration into our system. It was definitely overkill for what we needed a light sensor to achieve, but we didn't want to waste too much time integrating a light sensor with more integration overhead.

The camera system used was a very inexpensive security camera package including 4 video cameras and a DVR. The security camera system did not allow for the certain adjustments to be made in the system, such as light gain, and it really proved to be fairly ineffective in trying to locate the light fixtures, as there was too much noise in the video. In future versions, better camera systems were used and lamp detection was achieved.

4.3 Implementation

The data collection implementation of Version 1 was fairly straightforward. Each device had a dedicated communication port on the PC and a separate data collection application. Because each device pushes data automatically, the applications only needed to listen on a specific port and then handle the data accordingly. All data was stored in a separate time-stamped .csv file. Also, separate libraries for each device needed to be developed to parse and understand the different protocols of incoming data. In pseudo-code, the implementation looked like the following:

```
While(Com Port has data)
{
        collect one whole line of data;
}
add timestamp to the collected data
append data to the .csv file
```

The data management was the most complicated part of the Version 1 implementation, specifically data synchronization. It was decided that all incoming data be time-stamped using the local PC time. This would allow the addition of new devices to be added to the system and synchronized fairly easily. This solution was one that worked well when a PC was used, but in later versions this method had to be updated when a simple microcontroller was used. The thought process was that as long as the data was coming through a centralized point, that centralized point should be in charge of synchronizing the data. The results from this version were positive.

4.4 Outcome and Findings

Version 1 was a success and we were able to collect usable data. The main issue with Version 1 was that data was not accurate enough. This was a result of low-end sensors being used and not enough data being collected. The light sensors used could be upgraded for higher resolution, and the GPS data could be paired with IMU data in order to increase location accuracy (as seen in Version 2) (Krakiwsky et all 1988). However, this version was successful enough to be tested on our first trip to the UK and Spain.

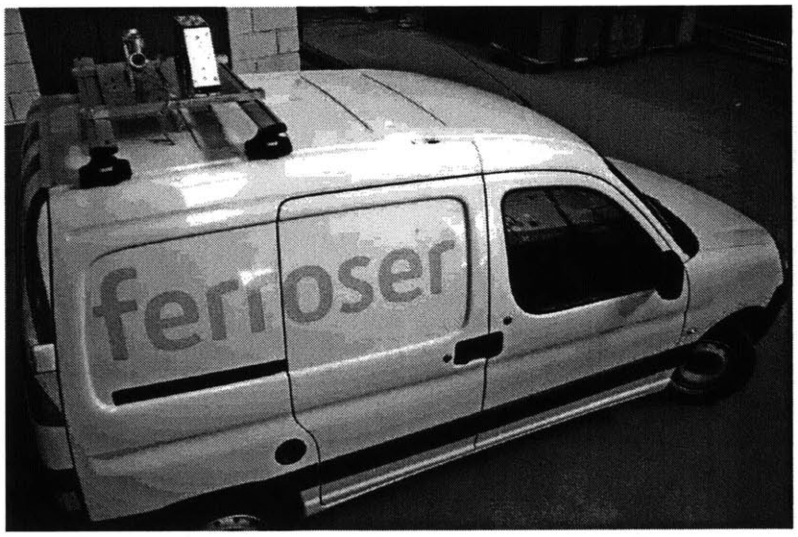


Figure 5: Version 1 being mounted in Spain.

On the trip we were able to test the mobile sensor platform in Birmingham UK, Soto De Real Spain, and Torrejon Spain. During the trip we met with many city officials interested in this new technology and passionate about their city streetlights. Our perspective of the overarching goal of the project was enhanced and a great deal was learned about the current state of streetlight monitoring and streetlights in general.

On our return we developed a visualization application that would display the information gathered in an informative and appealing way. What the application produced was luminosity maps in a Google Earth format. **Figure 6** is an example taken from Torrejon Spain:

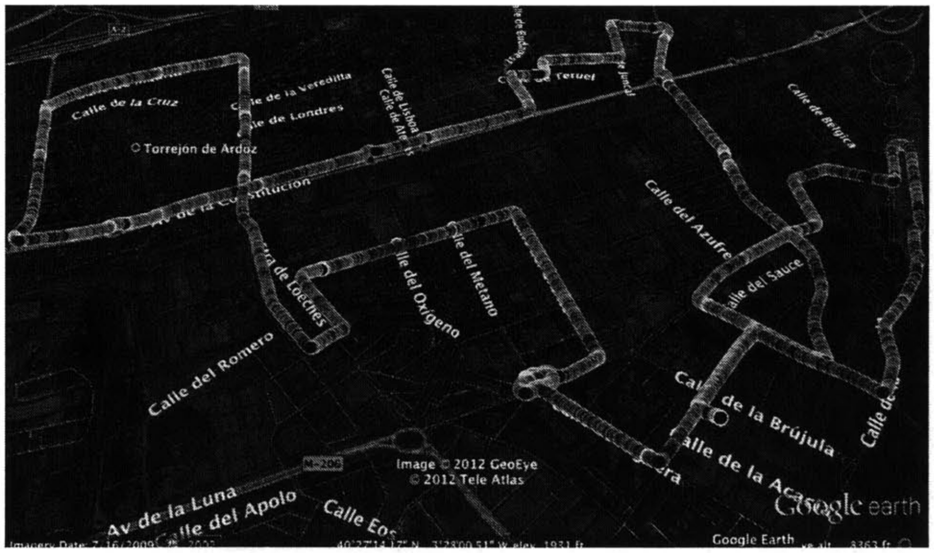


Figure 6 Luminosity map of Torrejon Spain

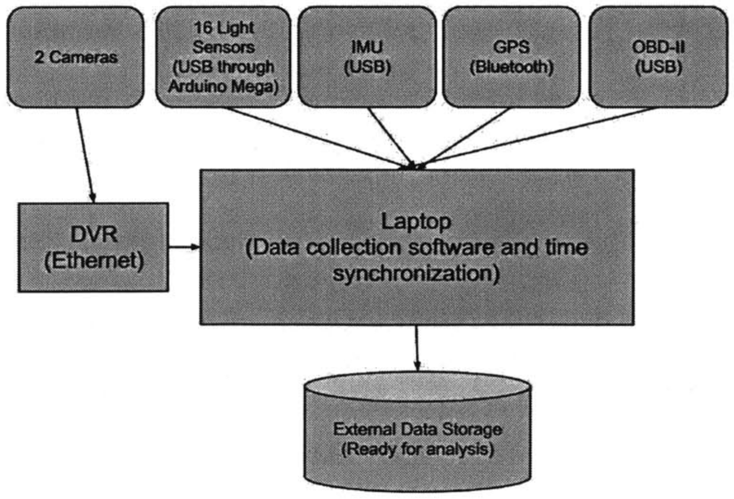
Version 1 taught us not only that this project was possible, but that it was also a needed tool desired by the industry.

Chapter 5

Version 2: Increasing Accuracy

5.1 Design

Version 2 was a step beyond the initial proof of concept. The main shortcoming of Version 1 was that it did not achieve the desired level of accuracy. The light sensors were not accurate enough, having just a lux range of 2 lux, and the location data were not accurate enough either, having a worst case accuracy range of 15 meters. In order to achieve higher accuracy, more research was done in order to find light sensors that could achieve the resolution desired. In terms of gaining higher location accuracy, an IMU and an OBD-II sensor were added to allow for location correction on analyzed data (Phan 2012). Overall, not much changed in the overall flow of data, but the number and types of sensors used was quite different from Version 1. **Figure 7** is a flowchart showing the dataflow of Version 2:



2 Cameras
16 Light Sensors (USB through Arduino Mega)
IMU (USB)
GPS (Bluetooth)
OBD-II (USB)
DVR (Ethernet)
Laptop (Data collection software and time synchronization)
External Data Storage (Ready for analysis)

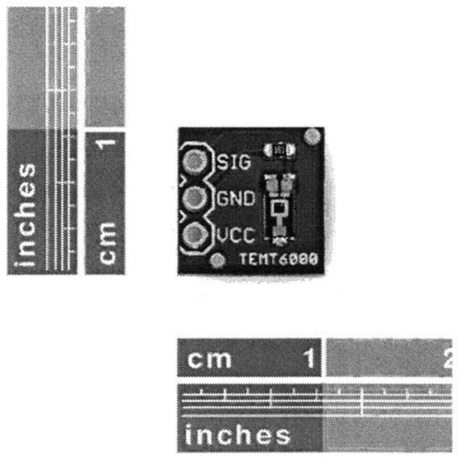
Figure 7: Data flowchart of Version 2.

The overall goal of this version was to achieve a level of accuracy needed to be a useful tool. If we could prove this was possible, then we could take even further steps in making a more standalone version.

5.2 Sensors

Light Sensor:

The light sensors used are the TEMT600 as shown in **Figure 8** below:



Resolution: 0.25 lux

Range: 0.25-250 lux

Sampling rate: 10-50 Hz

Cost:$4.95/unit

Figure 8: TEMT6000 (source: https://www.sparkfun.com/products/8688)

The TEMT6000 light sensor was selected on account of its relatively inexpensive cost and high resolution. In earlier versions of this project, light sensors with a resolution of 2 lux were used and were not accurate enough for the type of information the project required. Moving to a light sensor with a better resolution, as well as a high sampling rate better fit the requirements of our project.

16 TEMT6000s are assembled on a Plexiglas board in a 2 by 8 sensor array (see **Figure 9**).

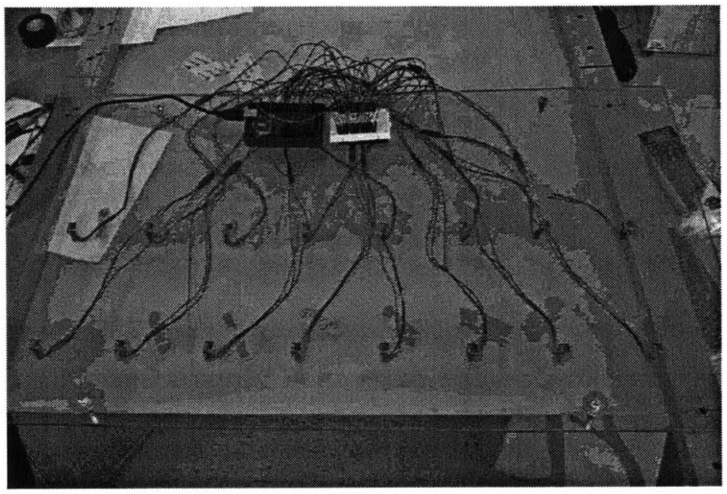


Figure 9: The linght sensor array of Version 2.

This setup allows for spatial resolution for cross-street analysis, as each light sensor measures vertical illumination at car-top level.

Data is collected through an Arduino Mega. Each of the 16 light sensors is connected to the Arduino Mega through analog input ports on the microcontroller. The Arduino Mega simply reads each analog input port and relays a batched message consisting of all sensor readings through an open serial port. A Java program running on a laptop connected to the Arduino Mega listens for communication over the specified serial port, timestamps the data collected, parses the relayed data, and the finally stores the data in a .csv file.

The time delay between analog input readings on the Arduino Mega is a matter of a few milliseconds; therefore the timestamp given by the Java program at the time of reception can be used for all 16 data points. This is an acceptable delay in this application because a few milliseconds between the first data point and last data point still yields accurate data analysis in our system. Using the PC time as the timestamp allows all sensors relaying information to the PC to be synchronized through a common time.

GPS:

The GPS unit used is the BT-Q818XT as seen in **Figure 10** below:

Accuracy: 3-15 meters

Sampling rate: 10 Hz

Cost: $73.95/unit

Figure 10: QStarz BT-Q818XT (source: http://www.qstarz.com/Products/GPS%20Products/BT-Q818XT-F.htm)

The BT-Q818XT GPS unit was again selected based on cost and accuracy. Data is collected either through USB or Bluetooth. The GPS relays data through a known communication port on the PC it is connected to. A Java program listens on the specified communication port and, whenever data is relayed, processes the GPS data. The Java program timestamps the data using its current PC time, parses the GPS data, and stores the data in a CSV file.

IMU:

The IMU unit used for this project is the CHR-UM6 as seen in **Figure 11**:

Range: Accelerometer: +/- 2g

Gyroscope: +/- 2000 °/s

Compass: ~5° accuracy

Sampling rate: 20 Hz

Cost:$199.00/unit

Figure 11: CHR-UM6 (source: http://www.pololu.com/catalog/product/1255).

The CHR-UM6 IMU unit was selected over 2 other tested IMUs based on its accuracy and low noise level. Through multiple experiments run on 3 different IMUs, the CHR-UM6 displayed excellent accuracy in accelerometer, gyroscope, and magnetometer data. It is reasonably priced and has a very high sampling rate.

Data is collected through a USB connection to a communication port on the PC. A Java program runs and listens to the open port, parsing and time-stamping incoming data as it is broadcasted to the PC. The data is then stored in a csv file.

OBD-II:

The OBD-II reader selected is the OBDLink SX as seen in **Figure 12**:

Supported Protocols: All legislated OBD protocols

Data Available: Speed, RPM, MPG, etc...

Cost: $50.00/unit

Figure 12: OBDLink SX (source: http://www.scantool.net/obdlink-sx.html)

The OBDLink SX was selected because of its price and ability to communicate through all legislated OBD protocols. It connects to the PC through a USB connection, and comes with proprietary software that saves OBD data in an excel file.

5.3 Implementation

The data collection implementation of Version 2 is again fairly straightforward. Each device had a dedicated communication port on the PC and a separate collection application. Each device had its own collection software running. The OBD-II reader and the IMU sensor data collection software was run on third party software, but the GPS and light sensors were run on data collection software developed as part of this project. The OBD-II and IMU were new to this system and the devices we used only allowed for third party software to be used to collect data.

All data was collected through a central PC so time synchronization was similar to Version 1. All data coming through the PC was tagged with a timestamp and stored in a local .csv file. We designed Version 2 to be centered on a PC again because were still trying to achieve a higher level of accuracy. It was in the trajectory of the project to go forth with a standalone solution once it was clear a feasible solution was achievable.

The main data collection issue we dealt with was addressing the latency between the time data is collected from the sensor, and the time that data is time-stamped. This latency was most studied with the light data once the 16 sensors were all being collected concurrently. Our studies showed that within the microcontroller, there were only a few microseconds between when the first light sensor was collected and the last, and just a few microseconds between when the data was collected by the PC program and time-stamped. Although some latency does exist, the time actually measured was not significant enough to really concern us.

5.4 Outcome and Findings

Version 2 was successful. We were able to achieve a higher level of accuracy with our data, and the methods used to collect and manage data were effective in being a real solution for monitoring streetlight levels in cities. Sumeet Kumar was able to use the extra IMU and OBD-II data to correct location inaccuracies in the GPS data, and the 16 light sensor array gave us a lot more usable information than the Sun Spots achieved in Version 1.

The biggest issue with Version 2 was that it was too user dependent. Ideally, we wanted to achieve a standalone system which we could mount on a car, plug in, and drive around a city. That was not the case with Version 2. Version 2 was very setup intensive. It would take some time to make sure all the sensors were working properly and logging properly. And the whole system had to be monitored using a PC inside the vehicle. There is too much user interaction to really have anybody but researchers use this version. We were able to take another trip to Spain to test the new version.

On our trip to Spain we were able to collect a lot of usable and accurate data. Again, too much time was needed to set up the system as well as run it. However, the experiments were executed properly. Along with running experiments, we met with city officials from Malaga and Santander, and even met the mayor of both cities. City officials were very excited about the system we were developing and showed extreme interest in using it in their cities. Santander had a high number light sensors placed below lampposts along parts of the city. This was the first time I actually saw a distributed network of light sensors and I noticed a few things about the way they set up the system that surprised me. Firstly the sensors were placed a few meters below the lights on the actual lamppost. This is fine for making sure the light is on and functional, but gives no information on what the lighting level is at street level. Secondly, there were far too many lampposts in Santander for this to be an effective deployment. The number of sensors they would need to cover the entire city would be a large investment in time and resources. Lastly, the operators really had no idea what to do with the information. It was all new to them as they were asked to install the light sensors as a trial. Their knowledge of how to use the information was limited. There seemed to be an opening for a mobile solution tied with automated analyses, so that city workers and officials could actually use the data collected by their light sensing system.

Figure 13: Myself with Version 2 mounted in Spain.

On our return we further developed the visualization application. The application now produced a higher-level luminosity map in a Google Earth format. Below is an example taken from Torrejon Spain:

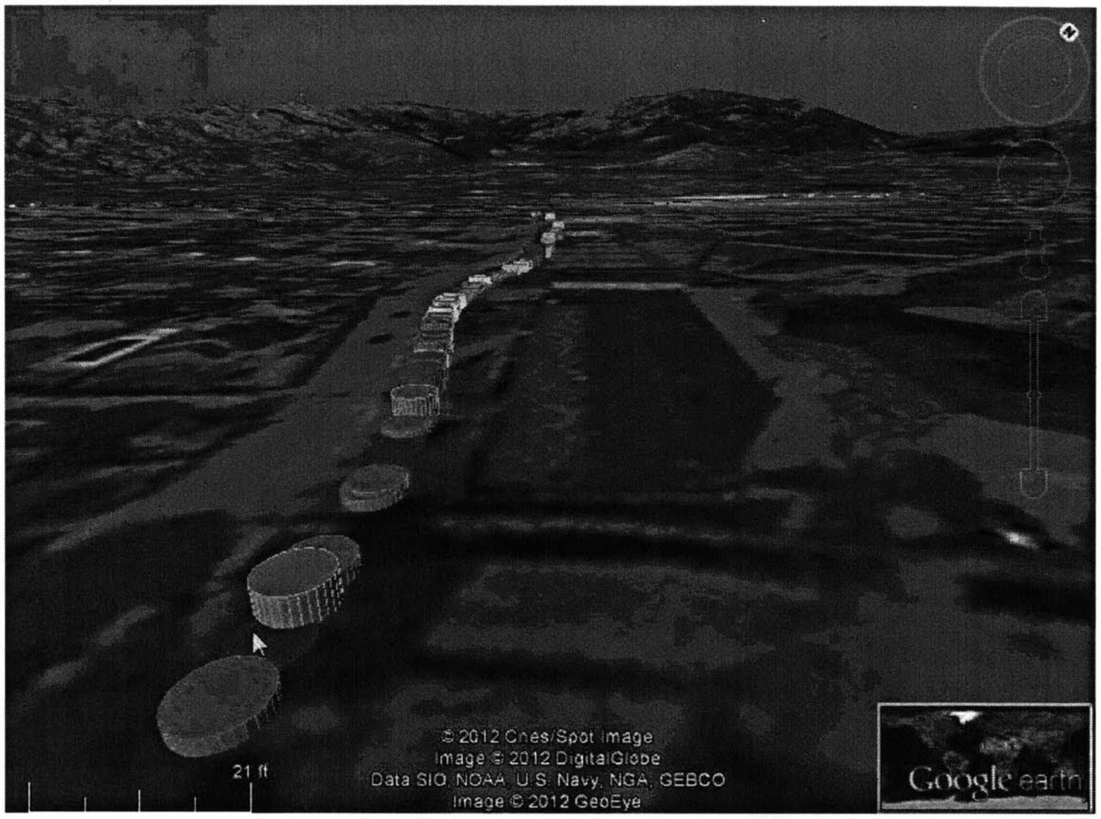

21 ft
© 2012 Cnes/Spot Image
Image © 2012 DigitalGlobe
Data SIO, NOAA, U.S. Navy, NGA, GEBCO
Image © 2012 GeoEye
Google earth

Figure 14: Version 2 luminosity map.

Chapter 6

Version 3: A Standalone System

6.1 Design

Version 3's main goal was to make a standalone data collection system. We wanted to design a collection system that, after initial setup, needed no human monitoring; a system that could be setup on top of a car, initialized, and sent for data collection. To do this, the major design change was exchanging a PC for a microcontroller. The microcontroller used in this system was the Arduino Mega 2560. The microcontroller had to be programmed to collect data from a GPS, OBD-II reader, IMU, and analog light sensors and store the collected data in some manner.

This version was a big step for the project because it was the first real prototype we designed for a potential standalone system. The biggest challenge, next to integrating the sensors with the microcontroller, was achieving the same level of accuracy and data quality achieved in Version 2. This was always a theme in the design choices we made for data collection, as much of the processing and parsing of data we could do on a PC would greatly reduce the speed at which the microcontroller could collect data. These choices, along with the extensive research done in choosing appropriate sensors, were the bulk of what drove the overall design of the new system. Below in **Figure 15** is a diagram of the Version 3's dataflow:

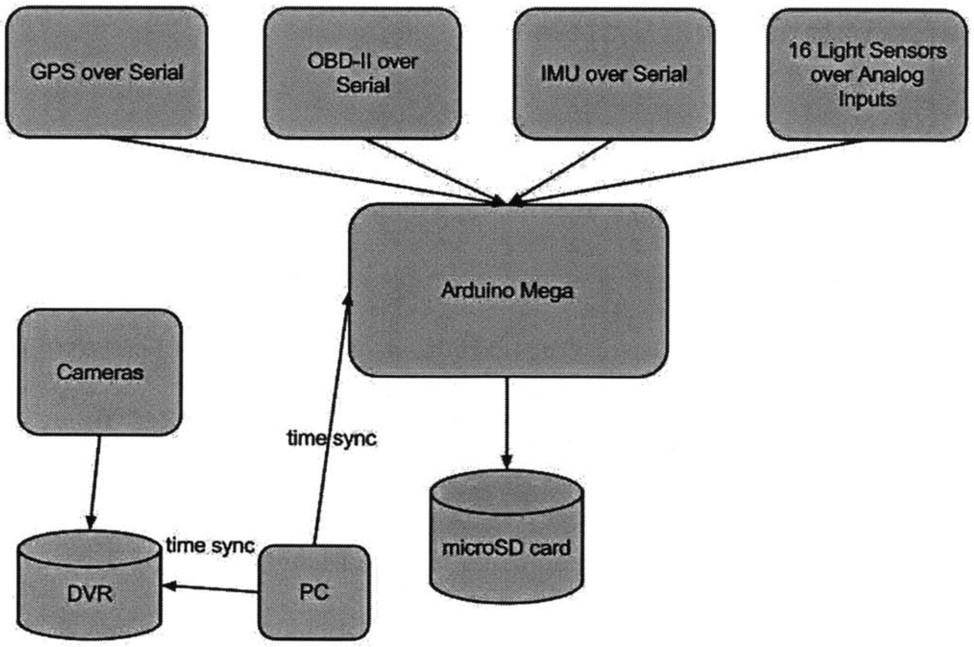


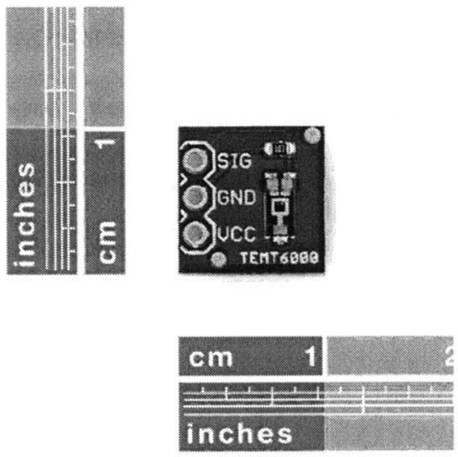
Figure 15: Version 3 data flowchart.

It can be seen that all the devices, except for the light sensors, are connected via Serial ports to the microcontroller. The microcontroller collects all the data, and stores it on a microSD card. There was discussion about transmitting the data via cellular data to a central server, however the time constraint proved storing data on a microSD card to be the more workable option (this is discussed in chapter 7). The camera system remains as a standalone system as it was in Version 2. Data synchronization is achieved by initializing both the microcontroller and DVR to be synched with a common timestamp served by a PC. This choice will be discussed in the implementation section of this chapter.

6.2 Sensors

Light Sensor:

The light sensors used are the TEMT6000 as seen in **Figure 16** below:



Resolution: 0.25 lux

Range: 0.25-250 lux

Sampling rate: 10-50 Hz

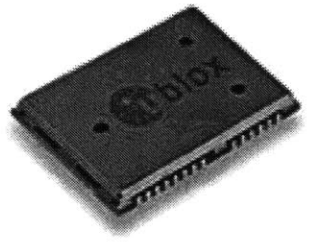
Cost: $4.95/unit

Figure 16: TEMT6000 (source: https://www.sparkfun.com/products/8688)

The same light sensors are used in Version 3 as were used in Version 2. They are also assembled in the same array system.

GPS:

The GPS used is the u-blox LEA-6R as seen in **Figure 17**:



Accuracy: 2 meters

Sampling Rate: 1-4 Hz

Price: $200.00

Figure 17: u-blox LEA-6R (source: http://www.u-blox.com/en/gps-modules/u-blox-6-dead-reckoning-module/lea-6r.html)

The LEA-6R was chosen for a number of reasons. Firstly, it has a higher accuracy range than the QStarz GPS used in previous versions. This would obviously lead to more accurate data. Secondly, it has UART capabilities, meaning it could be integrated through the microcontroller serial port. Integration was a main concern when choosing sensors, the more complex an integration system had to be, the slower data collection would be. Lastly, it achieved the sampling rate needed for our system. Using a 10 Hz GPS last version proved to be overkill, and our analysis only required 1-4Hz depending on the speed at which the vehicle was moving.

For the LEA-6R, data is collected through the Rx/Tx ports on the GPS. It is connected to a serial port on the microcontroller, and data is relayed. The GPS runs on a "push" system, where the microcontroller cannot simply request data when needed, the GPS pushes data whenever an interval has been reached. This was not ideal for our system as a polling system would be easier to manage. A polling system would run by our system polling data from the sensor when needed, and the sensor relaying that information back to our system; a call and response protocol. However the need for a polling system was realized after the choice had been made so the system had to adjust to get successful data collection without slowing down the entire system. In future versions, a GPS that allows interrupts should be used. The difference in implementation of a push versus polling system is shown below in pseudo-code:

```
PUSH:

While(data is available)
{
        collect the data;
}
```

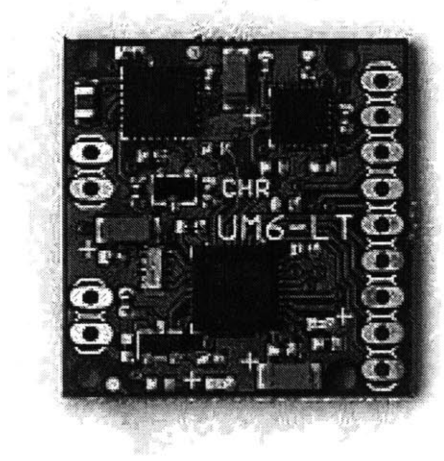
```
timestamp the data;
store the data is a .csv file;

POLL:
Request data from sensor;
Timestamp collected data;
Store the data in a .csv file;
```

As can bee seen above, the push solution must wait for data to be sent. This may cause a lot of wasted time if the data collection methods are not scheduled properly. However, the polling solution only has to wait for the data when it is requesting it.

Along with the data collection complications, setup and initialization would need to be done through the microcontroller. Luckily an existing library was available for the LEA-6R and a full solution was successfully implemented.

IMU:

The IMU used was CHR Robotics UM6-LT Orientation Sensor as seen in **Figure 18**:



Range: Accelerometer: +/- 2g

Gyroscope: +/- 2000 °/s

Compass: ~5° accuracy

Sampling rate: 20 Hz

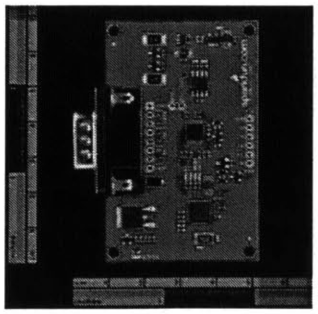
Cost: $149.00/unit

Figure 18: CHR UM6-LT (source: http://www.pololu.com/catalog/product/1256)

The UM6-LT IMU is essentially the same IMU used in Version 2, it isjust more programmable. The UM6-LT allows for easy integration and has a standard serial output that could be connected to one of the microcontroller serial ports. During implementation, a library had to be developed to decode packets sent by the UM6-LT, which took time and testing, but in the end, data collection was achieved and proved to be very fast and accurate. The library consisted of decoding methods that would wait for header bits that designated the type of information in the packet, and then step through each bit and convert it to a decimal value to be stored. It was thought of to just store the raw binary information to save on processing time. However it was realized that the processing did not take enough time to really be concerned with.

OBD-II Reader:

The OBD-II reader used in this system is the OBD-II UART from Sparkfun seen in **Figure 19**:



Supported Protocols: all legislated OBD II protocols

Data Available: Speed, RPM, MPG, etc...

Cost: $49.95/unit

Figure 19: OBD-II UART (source: https://www.sparkfun.com/products/9555)

This OBD-II reader has the same capabilities as the previous one used in Version 2. However it is more easily integrated with a microcontroller. This reader has standard Rx/Tx ports that connect to a microcontrollers serial port. A library had to be developed to send requests and parse data. This library consisted of a send method that would send the appropriate binary data to the reader, and a read method that would wait for data to be sent, and then convert the binary information into decimal data.

Microcontroller:

The microcontroller used was the Arduino Mega 2560 seen in **Figure 20:**

I/O: 54 digital,16 analog inputs,

4 UARTs (hardware serial ports)

Clock: 16 MHz crystal oscillator

Price: $58.95/unit

Figure 20: Arduino Mega 2560 (source: https://www.sparkfun.com/products/9949)

The Arduino Mega was chosen because of it's speed, I/O capabilities, and ease of use. Arduinos are used for many applications and there exists an extensive open source documentation and code base. We needed a microcontroller that could hold 16 analog inputs and at least 3 serial ports. The Arduino Mega was the obvious fit for this system and proved to be a successful choice.

6.3 Implementation

Implementation of Version 2 proved to be the most challenging. In other versions, each sensor had its own data collection program (either third party or developed in lab). This allowed for very easy scheduling; every program's data collection method was started and ended independent of each other. However, previous versions relied so heavily on user interaction. Data collection programs needed to be started, stopped, monitored, configured, and at times debugged. It would be impractical to assume we could train operators to have this level of knowledge of the system. The previous versions achieved accurate data collection, but as a product, they were really only useful in a research laboratory and not in a real world application. The goal of this version was to achieve a level of robustness within the system so that virtually no user interaction was needed after initial setup.

To achieve this, the microcontroller had to be programmed to collect data from all devices. First, independent initialization methods were developed. The GPS, IMU, and OBD-II reader were all programmable. And before data was collected, initialization had to be done to ensure consistent output formats from these devices. For example, the GPS was configured so the only GPGGA (Global Positioning System Fix Data) was relayed from the GPS as to not flood the microcontroller with numerous GPS data strings. Similar settings had to be configured on the IMU and OBD-II reader as well.

Then independent data collection methods had to be developed for each device. The microcontroller does not have a multithreaded processor, therefore the methods had to be as simple as possible as only one could be run at a time (Kreuzinger et all 2003). The more the data is processed in the data collection method, the slower the entire system would run, and the slower the data rate for each device would be. These methods were developed and tweaked as the system was tested. There were two types of methods developed for the sensors: a push method and a poll method. The IMU and GPS ran on the push method as they were not able to accept data requests. The light sensors and OBD-II reader ran on the poll method. These methods are described above in the previous section.

After the collection methods were developed, the outputs from these methods had to be stored in some way. The system uses a standard microSD shield mounted on top of the microcontroller. The shield can be seen in **Figure 21**:

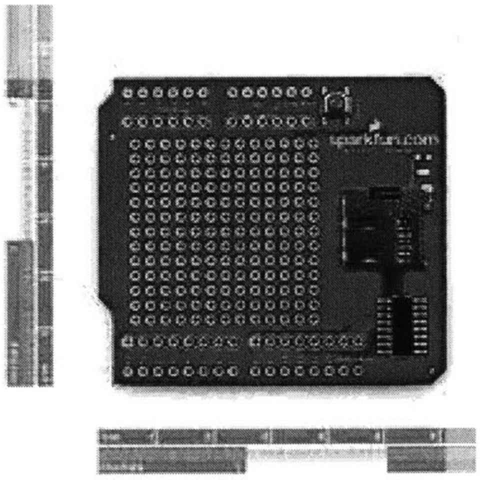


Figure 21: microSD Shield (source: https://www.sparkfun.com/products/9802?)

This microSD shield allowed us to save data directly from the microcontroller to files on a microSD card. The shield was easily integrated with the microcontroller, and each data collection method would save it's output to individual .csv files on a microSD card.

Lastly, the data collection methods had to be scheduled in such a way that the optimal amount of data could be collected at any given point. For example, there would be no point in waiting for GPS data to be relayed if there was IMU data available. The methods had to be interleaved in such a way that no time was left vacant within a scheduling cycle (Siegel 2011). Also, it would not be ideal to collect IMU data when GPS was available because then you would only collect partial GPS data. This took a lot of experimentation and understanding of how each device relayed information. The GPS was the slowest device in terms of sampling rate. Therefore, we designed the system to get as much done as possible in between GPS data samples. While the solution is not perfect, it does achieve a high enough data rate for our application. The scheduling system is as follows in pseudo-code:

```
While(GPS available)
{
        collect GPS data;
}
collect IMU;
collect light;
collect OBD-II;  //repeat IMU, light, and OBD-II until GPS is available.
repeat process;
```

The scheduling must be timed so that the system is waiting for GPS data just as GPS data is being relayed so that no time is wasted waiting. If this is not done correctly, then either the GPS is scheduled too early and time is wasted waiting, or the GPS is scheduled too late and only partial GPS data is collected.

Once the data initialization and collection was implemented, the system had to be able to synchronize all the data. The microcontroller does have an internal clock. However it is restarted every time the microcontroller is restarted, and there exists no frame of reference outside the microcontroller itself. This proved to be a large issue because we needed the data to be analyzed with the video collected through the DVR. The DVR was time synched with a PC, so we went the same route when time synching the microcontroller. On start up, the microcontroller waits for a time synch request. An application on the PC sends a current timestamp to the microcontroller over USB, and then the data collection starts. All data is time-stamped with the synchronized PC time just as the video on the DVR. This solution worked well for our application, however it required the user to manually synchronize both the microcontroller and DVR. This is an area that will be optimized in future versions.

6.4 Outcome and Findings

Data collection with this system is vastly more manageable for the user. After initialization, the user just has to ensure cables and wires do not disconnect. We felt we achieved a very accurate standalone system with Version 3, although not perfect by any means.

We analyzed the data latency between data points on a single sensor system (i.e. GPS, IMU, light sensor, and OBD-II) for Version 3. Below is a histogram of the latency between light sensor data collection:

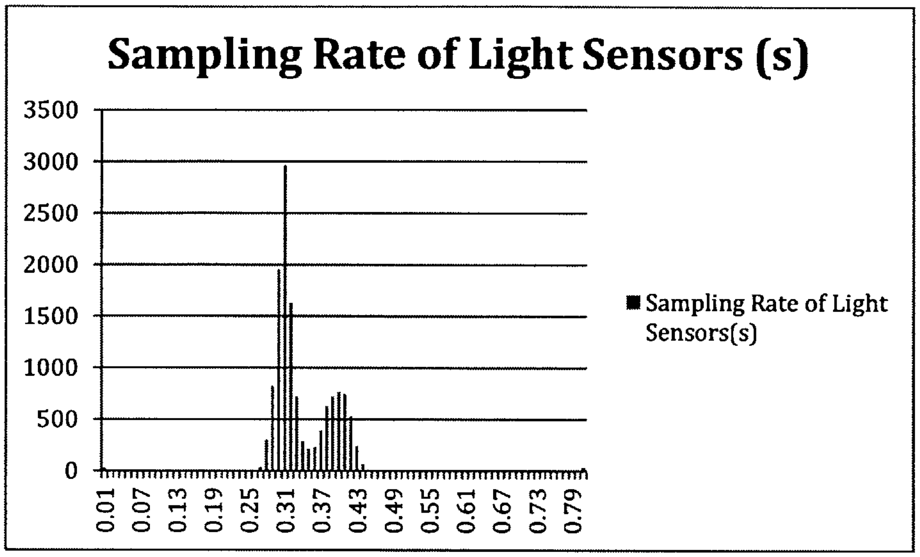


Figure 22: Histogram showing the sampling rate of light sensors in Version 3

As seen here the latency between data points of the light sensors appears to be bimodal between 0.3 seconds and 0.4 seconds. This can be explained by the scheduling the system used. The scheduling cycle was centered on the GPS data rate: 1Hz as seen in the histogram below:

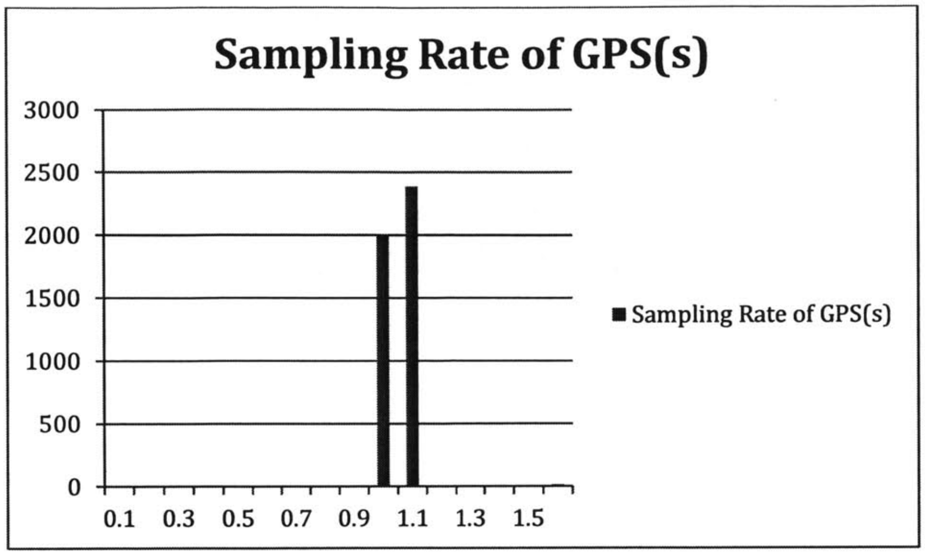


Figure 23: Histogram showing the sampling rate of the GPS in Version 3

In between GPS data points, the maximum possible amount of data collection for IMU, OBD-II, and light sensors was scheduled while still allowing some time buffer so the GPS 1Hz cycle was not missed. This in turn allowed for IMU, OBD-II, and light data to be collected every 0.3 seconds in between GPS data collection, with an average delay of 0.1 seconds delay while waiting for the scheduled GPS data point to be collected; for a total of 0.4 seconds every fourth data collection period that included the GPS collection. **Figure 24** describes the process more clearly:

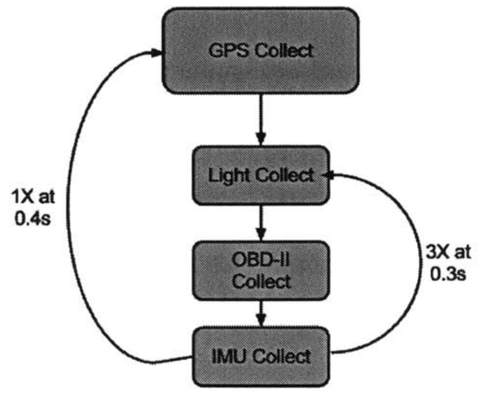


Figure 24: A flowchart demonstrating the scheduling in Version 3

This data shows that our scheduling was not perfect. There was still enough leftover time to possibly schedule more IMU, OBD-II, and light data to be collected, and could have produced a more consistent latency in between data points for these sensors. However, data loss was extremely rare and for light sensors was at a rate of 0.001805 loss per data point. Similar results for latency and data loss can be found for the IMU and OBD-II data collection in this version.

Data collection methods are not consistent in this system. The different devices all relay information to the microcontroller in different ways; some push data and some can be polled for data. Ideally, all devices would allow data polling, which would make scheduling a lot simpler. If the microcontroller could request data when needed, it would never have to wait for data to be available on the serial port. This would allow us to schedule the data collection methods in any manner we wished. For this to be achieved more research would have to be done on devices and ways of communicating with devices.

Chapter 7

Further Work

7.1 Data Collection Upgrades

There are a number of data collection upgrades possible with this system.

7.1.1 Polling vs. Pushing

With limited resources on the microcontroller, it is not optimal for the system to have to wait for data to be pushed from a sensor device. Waiting for data wastes valuable time and resources. Instead, the microcontroller should be able to poll the device for data and receive the data with limited delay. This may be difficult however, as the delay for polling data may be as long, if not longer, than the delay in waiting for data to be pushed from a device. However, there two ways I believe will solve this. Both need to be verified.

First, one could just buy better sensors which are quicker to respond to requests. This could solve the problem if all the sensors were able to be polled as described above. Unfortunately, this may not be possible with the budget of the entire system. Sometimes one will have to work with lower quality sensors that are not capable data polling.

The second solution would be to tie each device to its own microcontroller that acts as a data manager. That soulution would take care of waiting for data to come in, buffer the most recent data, and when interrupted by the main microcontroller, send the most recent data (Brinkschulte et al 1999). This could be a possible solution and would lower the strain on the main microcontroller as all it would need to do is poll other microcontrollers for data and then store the data.

There definitely exists an opening for further research and testing on how to create a poll-only system. In theory it should be a better system for collection and management of data across many sensors.

7.1.2 Latency Issues and Testing

With the focus of this project on successful and accurate collection of data in a mobile sensor system, not enough research was done in understanding the latency issues with the data collection. Like any sensor system, latency exists, however there needs to be research done on the threshold for acceptable latency in this system (Kaempchen et al 2003). If the delay between data collection and timestamp is too great, there may be inconsistencies in the analyzed data. This needs to be addressed and analyzed in future work.

7.1.3 Error Checking and Handling

Errors are inevitable in many systems, like the one described, and error checking and handling must be accounted for. Using LEDs, a future version of this system should be able to notify the operator if errors are occurring. Currently the only way of knowing is looking at the data stream to ensure all devices are working properly. There is a need for a more user-friendly notification system for error handling.

7.1.4 Graphic User Interface for Commercial Usage

Before non-researchers use this system, a Graphic User Interface (GUI) must be developed for the configuration of the system. Currently the only way to change configuration settings is by changing variables in the microcontroller code. This is not optimal for somebody who doesn't have a high understanding of the actual code. A GUI for setup would be integral in deploying a successful system to be used outside of a research lab.

7.2 Data Management Upgrades

7.2.1 Sending Data of a Cellular Network to a Remote Server

Currently this system stores data on a microSD card mounted on the microcontroller. In future systems it would be more beneficial to send data over a cellular network to a central server to be stored in near real-time (Siegel 2011). This would allow for more applications for real-time monitoring of the fleet collecting data. If a user had access to real-time data, they would know where every car in the fleet was located as well as the light levels they have already observed. It would also lead to faster notification of unsafe light levels in a city. A user could implement a flag system, where at certain light levels, notification is sent to the driver to further inspect an area. This would lead to even more accurate observations as the human and monitoring systems would combine their observations for a better analysis.

7.2.2 Database Storage

Rather than storing all the data in csv files, it would be a better practice to store all data in a database. This would allow easier post-processing integration with the data, as well as the capabilities to develop more applications for the data. This is a more feasible solution to data storage once data is being transmitted to a remote server over a cellular network as discussed in the previous section. If a database was available with all the streetlight level data of a city, outsiders could use the data to create display applications and analysis to further understand the data. It could also be paired with other studies to extrapolate on streetlight levels and their implications on other issues (pedestrian safety, accident occurrence, etc...).

7.2.3 Time Synchronization Improvements

Although the time synchronization solution in this system is functional and accurate, it is not automatic. It still requires the user to manually synchronize the time of the microcontroller and DVR. It would be a better solution to use the GPS timestamp to automatically set the microcontroller time as well as the DVR. This has been done with microcontrollers before, however it was not possible with our current DVR. A future system should use a DVR where this is possible.

Chapter 8

Contributions

In this project, we developed three separate versions of a mobile sensor platform system for collecting data on streetlight levels of a city.

The first version was a simple system consisting of a GPS, 3 lux meters, and a camera system. Data was collected and stored on a PC using software developed in the lab. Visualizations were developed to further understand the type of data we were collecting and how useful it could be. This version was a success and proved that this type of data collection could be done successfully.

The second version was a more complex system using a GPS, IMU, ODB-II reader, 16 lux meters, and a camera system. Data was collected and stored on a PC using third party software and software developed in lab. Visualizations were improved to better display the data. This version was a success and proved this type of data collection could be done accurately.

The last version of the system consisted of a GPS, IMU, ODB-II reader, 16 lux meters, a microcontroller, and a camera system. The system was able to collect, synchronize, and store accurate location and light level data through a microcontroller. This version was a success and proved this type of data collection could be done in a standalone system.

Through this project we were able to prove that a mobile sensor system could be used to monitor the street light levels of a city, and that it could be done cheaper, but just as accurately as a static networked sensor system.

Literature Cited

Anguelov, Dragomir, et al. "Google street view: Capturing the world at street level." *Computer* 43.6 (2010): 32-38.

Brinkschulte, U., Krakowski, C., Kreuzinger, J., & Ungerer, T. (1999). A multithreaded Java microcontroller for thread-oriented real-time event-handling. In *Parallel Architectures and Compilation Techniques, 1999. Proceedings. 1999 International Conference on* (pp. 34-39). IEEE.

European Standard. (2003). *Road lighting – Part 3: Calculation of performance* (EN 13201-3:2003).

Gao, H., Barbier, G., & Goolsby, R. (2011). Harnessing the crowdsourcing power of social media for disaster relief. *Intelligent Systems, IEEE, 26*(3), 10-14.

Hamsa, A. A. K., Miura, M., Sakurai, O., & Seki, S. (2009). Analysis of Streetlight Illuminance in Residential Areas in Kuala Lumpur. *Journal of Asian Architecture and Building Engineering, 8*(2), 547-554.

Huang, X. M., Ma, J., & Leblanc, L. E. (2004, April). Wireless sensor network for streetlight monitoring and control. In *Proceedings of SPIE* (Vol. 5440, pp. 313-321).

Jing, C., Shu, D., & Gu, D. (2007, May). Design of streetlight monitoring and control system based on wireless sensor networks. In *Industrial Electronics and Applications, 2007. ICIEA 2007. 2nd IEEE Conference on* (pp. 57-62). IEEE.

Kaempchen, N., & Dietmayer, K. (2003, October). Data synchronization strategies for multi-sensor fusion. In *Proceedings of the IEEE Conference on Intelligent Transportation Systems* (pp. 1-9).

Krakiwsky, E. J., Harris, C. B., & Wong, R. V. (1988, November). A Kalman filter for integrating dead reckoning, map matching and GPS positioning. In*Position Location and Navigation Symposium, 1988. Record. Navigation into the 21st Century. IEEE PLANS'88., IEEE* (pp. 39-46). IEEE.

Kreuzinger, J., Brinkschulte, U., Pfeffer, M., Uhrig, S., & Ungerer, T. (2003). Real-time event-handling and scheduling on a multithreaded Java microcontroller. *Microprocessors and Microsystems, 27*(1), 19-31.

Phan, L. N. (2012). *Automated rapid thermal imaging systems technology*(Doctoral dissertation, Massachusetts Institute of Technology).

Siegel, J. E. (2011). *Design, development, and validation of a remotely reconfigurable vehicle telemetry system for consumer and government applications* (Bachelor's thesis, Massachusetts Institute of Technology).