Wireless Automobile Detection, License Plate Processing, and Data Availability Network

Kevin Emery, Santiago Gonzalez, Brandon Rodriguez, Taylor Sallee
Undergraduates, EECS Department, Colorado School of Mines

May 1, 2014

Abstract

This paper describes a parking lot monitoring system to be used at the Colorado School of Mines (CSM). The system is designed to help campus members enjoy a better parking experience by spending less time trying to find open parking lots. The project was a semester project for the computer science graduate course *Wireless Sensor Networks*, taught by Dr. Tracy Camp at CSM. The work described in this paper is an extension of a project called Smartlots started by the campus computing group ACMx, which uses a wireless sensor network to track the number of cars in specific parking lots. The system extends the ACMx project by adding components to the sensor network that record the license plate numbers of cars as they enter and exit lots, and by adding an administrator interface to the already existing Smartlots website. The information gathered by the sensor network is displayed on a website that lets users survey the state of the lots before arriving on campus, allowing them to make informed parking decisions. The website is also the result of collaboration between the authors of this paper and the ACMx group. The paper provides some background information, describes the parking lot monitoring system and its components, discusses the details of our implementation, and provides some initial results from testing the system and its various components.

1 Introduction

The parking lots at the Colorado School of Mines (CSM) can be a source of frustration for students and faculty members. Because space on campus is limited, the lots are often unable to accommodate everyone who needs to use them. Students need to choose a lot before they go to class; however, knowing which lots have free spaces usually requires driving through the lots looking for a space. The extra time spent searching parking lots is frustrating, and can make students or faculty late for class. If campus members could discover which lots had available spaces prior to arrival, much time and frustration could be saved. This project involved the design and creation of a parking lot monitoring system that will be the first step towards a better parking experience on the CSM campus. The system keeps track of both the number of vehicles within a parking lot and the license plate numbers of those vehicles. The information is made available online to system administrators and CSM campus members. There are two main parts to this project:

1. Parking lot capacity monitoring and
2. License plate detection and monitoring.

The idea behind the first part is that a student can get on their smartphone, tablet, or laptop before heading to class, see a nicely-formatted list/map of the lots on campus, and decide where to park based on how full each lot is. Administrators can monitor the information and make sure it is correct, updating the web application as needed with new statistics, lists, and maps as more lots are added to the system. Before beginning this project, the Association for Computing Machinery (ACMx) student organization at CSM had already started working on a system for monitoring lot car counts and capacities (see section 2). We collaborated with the ACMx group, integrating our new systems with theirs to design and test our parking lot monitoring system.

The second part of the project is motivated by a desire to test the feasibility of using the small Raspberry Pi Linux computer as a platform for capturing and processing images

of license plates. Because the Raspberry Pi is inexpensive ($25-$35 depending on the model), it could be useful in a wide variety of applications that require accurate license plate recognition. Currently, our web application simply lists the license plate numbers for site administrators to view, but we envision that this monitoring system may eventually be used by the campus parking department to detect when vehicles have entered lots without parking passes. The system itself consists of several subsystems, which are described in detail in section 3.

2 Related Work

The ACMx group at CSM has designed a system named Smartlots to track the number of vehicles in a parking lot. Their system is a wireless sensor network consisting of sensor nodes and a central base station. The sensor nodes are based on the Arduino Fio microcontroller platform equipped with triaxial magnetometers and XBee Pro radios. The base station is a Raspberry Pi connected to the internet via Ethernet. The information collected by the sensor nodes is transmitted to the base station and forwarded to a server running a small web application. The Smartlots system is described in [1] and [2]. [1] describes an implementation of a system to detect automobiles entering and exiting a parking lot using the Arduino Fio and the Raspberry Pi. This system uses the IEEE 802.15.4 communications standard to communicate automobile detection data from the sensor nodes to the base station. [2] describes the ongoing status of the ACMx project. Before we started our project, the ACMx team had created a website that would display the status of the CTLM upper and lower lots once the monitoring system had been deployed. [2] is updated periodically with new information regarding project progress. The project described in this paper augments and extends the ACMx project while collaborating with Roy Stillwell (ACMx president and coauthor of [1]).

K. V. S. Hari [3] researched the use of magnetometers for detecting and classifying automotive vehicles as part of the Centre for Development of Advanced Computing at

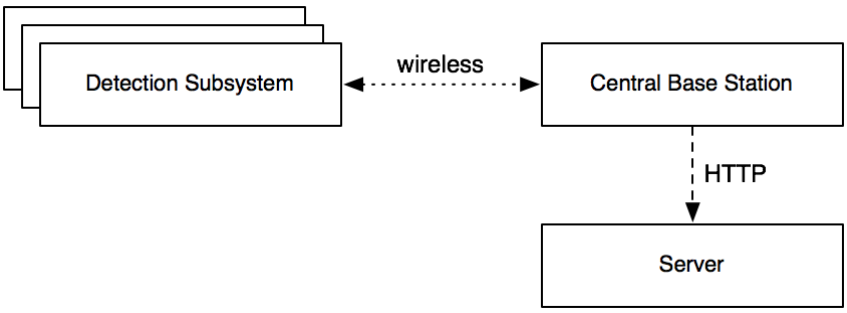

Figure 1: Overall System Architecture

Thiruvananthapuram, India. Hari conducted this project in the hopes that a similar system could eventually be used in traffic lights to improve the flow of vehicles. Automobiles are detected using a threshold system similar to that used in [1], and are classified using various features in the vehicle's magnetic signature with one or more sensor axes using clustered dipole models.

3 Project Overview and Methods

The project described in this paper is an extension of the ACMx Smartlots project. Our system consists of several components, each of which is either an extension of a previously developed Smartlots system or an entirely new subsystem that has been integrated into the existing system. Figure 1 shows the overall architecture of our system. The system relies on three main components:

1. Detection Subsystem,
2. Central Base Station, and
3. Server.

The detection subsystem consists of a small number of sensor nodes, placed strategically at parking lot entrances. Each sensor node consists of an Arduino Fio equipped with

a triaxial magnetometer that is used to detect passing vehicles. Before beginning this project, most of the work to determine the hardware configuration and software for the Arduino Fios had already been done by the ACMx group in [1]. Each sensing node is also equipped with a Raspberry Pi single-board computer and a 5 megapixel camera, which takes photographs of the license plates of cars as they pass by. When a car passes a sensor node, the magnetometer detects the vehicle, and the Arduino Fio sends a hardware interrupt to the Raspberry Pi, which kicks off a Python script so that the camera can take a picture of the vehicle's license plate. These images are then run through an OpenCV Python program to reduce the size of the image, through downscaling, cropping, and compression algorithms.

Another Raspberry Pi operates as a central base station that receives automobile detection and status data from every sensor node, via transmission from the XBee radios attached to the sensor nodes. The base station Raspberry Pi will be centrally located in a nearby building with internet connectivity. The base station utilizes its network connection to forward the data to the ACMx server.

The ACMx server is a Linode-hosted machine running Ubuntu 13.10. The web interface that displays the data we collect is served using Nginx. The Nginx server hosts a PHP web application with a RESTful HTTP API that both receives the data from the base station and routes it to the correct part of the application (database, image processor, etc.). This interface is the method of communication between the server and the base station. When data is received from the base station, the application processes it, stores it in a MySQL database, and displays it appropriately on a user-facing website. Part of the website is for administrators only, and provides direct access to license plate data and other system data that is not available to the general public.

The following sections describe the architecture and function of each of the aforementioned subsystems in detail. The detection subsystem is broken up into two parts: Automobile Detection (section 3.1) and Image Acquisition and Processing (section 3.2).

The glue between the sensors and the server is the Central Base Station (section 3.3). The server subsystem is also broken into two parts: License Plate Recognition (section 3.4) and Web Application (section 3.5).

3.1 Automobile Detection

The automobile detection subsystem provides a means for detecting entry and exit of vehicles from a parking lot. This subsystem is designed to be placed on the side of the road next to each parking lot entrance and exit. Automobiles are detected using a magnetometer which perceives the induced change in the local magnetic field as the metallic structure of the vehicle passes by, as described in [1]. The automobile detection subsystem is based around the commercially available Arduino Fio 16-bit platform which utilizes the ubiquitous Atmel ATMEGA328p microcontroller. When an automobile is detected by the Arduino Fio, a packet is sent through the onboard XBee radio to the central base station (described in section 3.3). The packet contains the direction of the passing vehicle, the processor's temperature, the battery voltage, and a portion of the magnetometer data stream. At the time of detection, a hardware interrupt is sent to the image acquisition Raspberry Pi, to notify it that a picture should be taken by its onboard camera. An easily accessible button is also wired onto the breadboard to allow a human to trigger the onboard camera for testing purposes.

The hardware used for automobile detection is contained in the same enclosure as the plate image acquisition hardware for simplicity. Both components share a unified power supply based on a high-capacity, 6000 mAh lithium polymer battery, which is charged by a relatively small photovoltaic panel as seen in Figure 2. A 5V boost converter is required to interface with the image acquisition Raspberry Pi since the lithium polymer batteries we use only provide 3.7V.

The use of a magnetometer ensures that automobiles and other roadway vehicles such as motorcycles are detected, while pedestrians and particulate deposits, such as snow, are

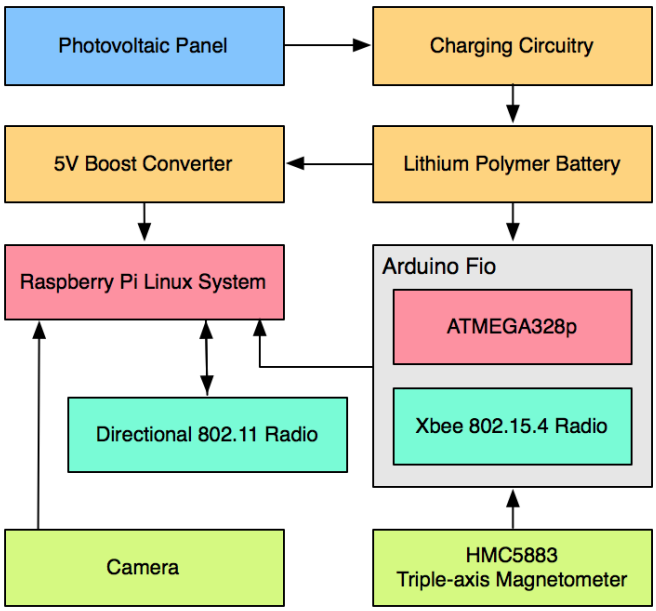

Figure 2: Automobile Detection Subsystem Architecture

ignored. As mentioned in [1], the automobile detection subsystem uses the commercially available HMC5883 triaxial magnetometer as it provides an adequate balance of sensitivity and low power consumption.

3.2 Image Acquisition and Processing

Once the Acquisition Raspberry Pi receives the signal from the vehicle detection Arduino Fio, it begins the process of obtaining a picture of the vehicle's license plate. This procedure is separated into two separate sections: image acquisition and image processing

3.2.1 Image Acquisition

While the automobile detection subsystem is responsible for correctly identifying when a vehicle enters the lot, the image acquisition subsystem is responsible for determining when to capture a picture of that car in order to optimize the quality of the license plate image.

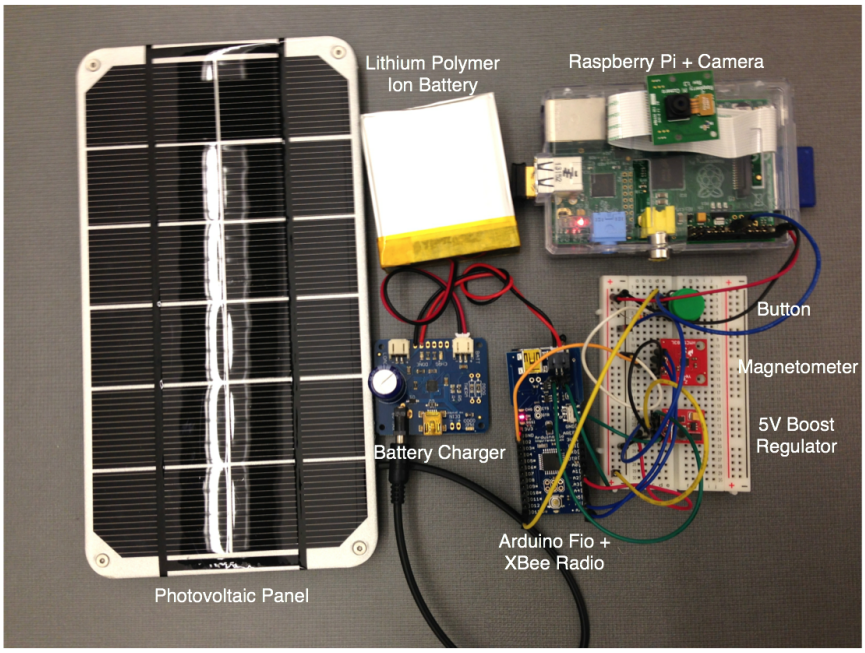

Figure 3: Automobile Detection Node Hardware

The timing difference between when the Raspberry Pi receives the signal from the Arduino Fio and when the image is captured is set to 500 milliseconds so as to allow the car to completely pass the Arduino Fio and enter the capture area. Only one picture is taken of each car so that capture time and power consumption on the Raspberry Pi are limited. This frees up more resources to be allocated towards image processing.

3.2.2 Image Processing

Once an image is captured, the Raspberry Pi creates a subprocess that takes care of manipulating the captured image. The image is processed using the OpenCV (Open Source Computer Vision Library) API [4]. This API provides many different algorithms that can be used for image detection and shape recognition. It was chosen for both its ease of use and widespread cross-compatibility; it has interfaces for development in C, C++, Python, Java, and MATLAB, and supports all major operating systems (including distributions of Linux such as Raspbian).

When processing the image, two factors are taken into consideration:

1. File Size and
2. License Plate Quality.

The file size is reduced as much as possible in order to expedite the file transfer process from the Acquisition Raspberry Pi. The second factor, license plate quality, is considered because the image must be readable by the Optical Character Recognition (OCR) software used on the server. Several different methods of processing the images using OpenCV were attempted, including but not limited to:

1. Contour Mapping,
2. Image Cropping,
3. Image Resizing and Compression, and

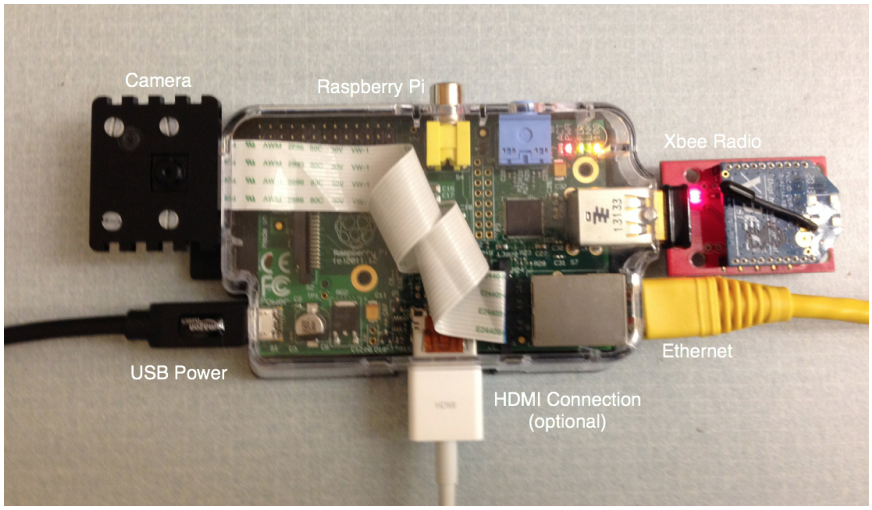

Figure 4: Central Base Station Hardware

4. Color Manipulation.

Two datasets were analyzed to see how well the license plate recognition software worked on various types of images (section 4.2). These two datasets, *cropped images* and *compressed images* had different methods of processing applied. The*cropped images* were naively cropped to eliminate space at the top and bottom of the picture. These images were converted to grayscale to reduce file size. The *compressed images*had various amounts of compression applied to them, as well as different color filters.

Once the OpenCV software has processed the image, it is sent from the image acquisition Raspberry Pi to the ACMx server via a Wi-Fi connection. As mentioned in section 5.4, we originally intended to send the compressed images to the base station through the same XBee radio that sends the magnetometer sensor's data, but we discovered that attempting to do this opened a can of worms that was too complicated to solve given the time constraints placed upon us.

3.3 Central Base Station

The central base station is the point of contact between the sensor network and the server. The data gathered by the Arduino Fio sensors are routed to the base station via XBee radios. The base station, which is itself a Raspberry Pi (see Figure 4, forwards the data to the ACMx server using HTTP requests. Forwarding data is the main function of the base station. The base station, along with all the other Raspberry Pis in our system, runs the Raspbian operating system [5], a lightweight version of Linux. Raspbian is optimized for the Raspberry Pi, so writing Python scripts that take care of forwarding the data via HTTP to the server was simple. The more difficult task was writing code to allow the base station Raspberry Pi to interface with the XBee radio, and listen for packets coming from the XBee attached to the Arduino Fio. The Python code that runs on the base station Raspberry Pi is listed in Appendix 2. The base station is also equipped to receive images via the XBee radio, and forward this data to the server. However, as previously mentioned, the time constraints of the project did not allow us to finish the sensor-side code that would need to run on the image acquisition Raspberry Pis to make image transmission via XBee possible. Therefore, when testing, we simply connected the image acquisition Raspberry Pis to a Wi-Fi internet connection, and POSTed directly to the server, using the RESTful web services already in place.

3.4 License Plate Recognition

Once an image has been captured by an Acquisition Raspberry Pi, it is transmitted to the server via the REST API provided on the server. The code that processes the image once it is received on the server serves several purposes. First, it authenticates the sender by checking for a username and password. Next, it timestamps the image and saves it to a directory on the server not directly accessible from the web. Lastly, the script calls a function to invoke two different automatic license plate recognition (ALPR) programs and stores the resulting character extractions to a MySQL database.

3.4.1 ALPR Literature Survey

Many license plate extraction algorithms have been developed, implemented, and critiqued in the academic community. For use in our project, we needed to find an implemented ALPR algorithm capable of running in a Linux environment (on the server) or a cloud service that would allow for reliable character recognition from license images.

Much of the research done on ALPR technology is less concerned with implementation and more interested in algorithmic advances. License plate recognition relies heavily on manipulation of graphics files and computer vision - each fields of study in their own right. Despite the lack of detailed literature on implemented programs, we were still able to find papers that briefly mentioned two operational systems: OpenALPR and JavaANPR. Note that ALPR and ANPR are synonymous (some packages use the term 'number plate recognition' in place of 'license plate recognition').

A. OpenALPR

OpenALPR is a relatively new open source ALPR system that is built on top of OpenCV (a computer vision framework), and the Tesseract OCR engine. Initial commits to the source code were made in December, 2013 [6]. Using such a new program can be more risky than going with an established leader in the market, but we found that OpenALPR offered just the type of functionality we were looking for [7].

B. JavaANPR

JavaANPR is an older program (published in 2007) that was created as part of a Bachelor's thesis at the Brno University of Technology in the Czech Republic [8]. Stemming from an academic environment, JavaANPR has much more detailed documentation on the program and the algorithms implemented in the program than any other option surveyed (both academic and industrial). On the same token, development on the program seems to have stopped after the thesis was submitted, so the software has not seen updates in eight years. No software is perfect, and knowing that we would be using a program that is no longer being maintained is just one of the drawbacks we discovered about JavaANPR. The

other main disadvantage of JavaANPR is that it was developed in Europe for European license plates. The source code would have to be correctly adapted in order to recognize North American plates, and the OCR engine would need to be trained for fonts used in America.

C. General ALPR Algorithm

The two programs do share the same general algorithm for extracting license plates. First, the plate is identified using some means to isolate rectangles in the image (texture, color, boundaries, etc.). Once the bounding box containing the plate is found, segmentation begins to isolate each individual character on the license plate. Finally, these image representations of characters are mapped to their most likely plain text counterparts.

In general, ALPR systems only work in the specific geographic region for which they have been 'trained.' For example, North American plates are a different size than European plates. Detection algorithms to identify a license plate in America would not work on an English car without modifications to the ALPR's configuration. Similarly, different states and countries use different fonts on their plates. In order to achieve maximum accuracy, you would need to collect hundreds of samples of the fonts and teach the program what those letters look like [9].

3.4.2 ALPR Industry Survey

Many ALPR systems exist today for commercial and government purposes. Uses can range from cross-referencing address information with the DMV to bill users of toll roads to monitoring roadways for stolen cars or even measuring car speeds. Our team was able to contact two manufacturers of ALPR systems to learn more about their use in industry: Q-Free ASA and The 3M Company.

A. Q-Free ASA

One commercial provider of ALPR software is Q-Free ASA, which is based in the Netherlands. Q-Free is a top search result when searching the web for ALPR software.

Their ALPR product, named Intrada, offers a cloud-based solution that allows users to send the image of the license plate to their servers, and get back the plain text representation of the characters. One unique feature Intrada includes that we did not find in any other package surveyed is the ability to identify the state of origin of the license plate. Intrada does this by analyzing minor differences in fonts used across state borders [10].

Note that this feature is not the same as training OpenALPR or another system with a new font. In OpenALPR, new trained data helps increase accuracy in detection. As a user, you can ask OpenALPR to examine the probability of a "template match" to a specific font, but the software does not intelligently determine which font family it most closely matches. Intrada offers the simplest solution in terms of startup costs because there is no local software to configure on our machine.

B. The 3M Company

E-470 uses high speed cameras to capture and extract license plate data in order to bill users of their toll road in the Denver Metro area. We reached out to the E-470 Public Highway Authority to inquire more about their deployment of sensors and ALPR software. Representatives informed us that their toll booths used an ALPR system developed by The 3M Company. Even though E-470 is using an ALPR which automatically detects the characters of a license plate, every image must be reviewed by a human operator before the driver is billed [11]. In our project, we don't want a human operator to have to confirm each plate - we want to trust the ALPR to do its job.

After communicating with E-470, we contacted The 3M Company with regards to their ALPR software. A representative reached out and confirmed that they do offer an API to their program, allowing their technology to be built into other projects. Because we are creating a parking lot monitor system, and not just a license plate recognition system, an API is crucial so we can integrate the ALPR into our workflow [12]. Unfortunately, our request for an educational use license for their ALPR was denied, as after reviewing our team's proposal 3M was worried we might try to commercialize our network. We were

unable to learn more about their algorithm or API after that point.

3.4.3 Selection of ALPR Programs

Our literature and industry surveys left us with three viable options for license plate recognition: OpenALPR, JavaANPR, and Q-Free's Intrada ALPR. Of those three, only OpenALPR was ready to be integrated into our network. Intrada ALPR is a commercial service, so our ability to use it was dependent on Q-Free granting an educational use license to us for the semester. Lastly, JavaANPR would need to be modified to detect and recognize North American plates and fonts. JavaANPR also lacks additional information that both OpenALPR and Intrada ALPR provide, such as execution time and confidence range.

A main goal of this project is to demonstrate a proof of concept network that is capable of monitoring parking lots and capturing license plate data. To stay aligned with this goal, and because of our unfamiliarity with graphical algorithms, we decided not to pursue using JavaANPR as a software solution for our network.

After reaching out to Q-Free with regards to using their Intrada service for the semester, we were referred to their Global Account Manager OEM Partners representative Marco Sinnema. Dr. Sinnema, on behalf of Q-Free, graciously granted our request to use their product free of charge for the spring 2014 semester.

Instead of choosing between two ALPR systems, we decided to integrate both platforms into our network to allow a comparison between the two tools, and offer better data integrity by checking if results from both systems (for the same photo) matched.

3.4.4 Test Data Collection

In order to evaluate the ALPR platforms, we needed to collect test data. Both platforms were already trained to run North American license plates out-of-the-box, so the stage was set to process images for a comparison in performance. With our final deployment in

mind, we also wanted to collect images from a variety of different angles and distances, as the resulting evaluation could help determine the best camera placement for a real-world deployment. To accurately test the ALPR programs for our project, the test images needed to be taken with the same camera that would be used during a deployment.

We loaded a special testing script written in Python onto a Raspberry Pi to accomplish this task. The script was programmed to wait for a hardware interrupt before taking an image. We wired a button via a breadboard to provide the hardware interrupt. Our team brought this Pi, camera, and button out to a parking lot on campus and positioned it in a location that would be a likely candidate spot for a future deployment.

Once powered on, we had one teammate drive their car in and out of the parking lot multiple times. While he was driving, two other team members operated the camera and breadboard hardware to take images. Using the same license plate for this portion of our test data acquisition was intentional to lower the number of uncontrolled variables in gathering data.

Besides acquiring many images of the same license plate, we also collected several images of nearby parked cars, and of cars that entered or exited the parking lot during that time period. And because our system is designed to be deployed on a college campus, we also collected images from different states, as the population of nonresidents is much greater here than in neighboring small towns.

Overall, we were able to collect 52 images of seven different plates that contained recognizable numbers. Each image from the Raspberry Pi camera was about 3 megabytes in size in its uncompressed form. The Python script also specified that the camera should save in a JPEG format.

The data set, which we will now refer to as the *unaltered image data set*, is the basis for all other data sets in this paper. That is, all other data sets are derived from these original, uncompressed images.

3.4.5 ALPR-Project Integration

Both OpenALPR and Intrada ALPR had to be integrated into our server-side code so they would run automatically without the need for human intervention. Because OpenALPR it is not a cloud service, it needed to be installed locally on the ACMx server. The Intrada ALPR was much simpler because we were only interacting with the API.

A. OpenALPR

As stated in Section 3.4.1, OpenALPR is built on top of OpenCV, a computer vision library, and Tesseract OCR, an OCR engine. Tesseract OCR had additional dependencies that needed to be installed before it would compile.

We used Ubuntu's `apt-get` command to install almost all of Tesseract's dependency packages. Tesseract makes use of a recent version of an image processing software called Leptonica. Because the version used in Tesseract superseded the versions available in Ubuntu's repository, we had to download the source files for Leptonica and compile it from scratch.

After installing all the dependencies, we were finally able to compile Tesseract from its source files. Additionally, OpenCV came bundled with all of its dependencies, so we were able to build the OpenCV library rather effortlessly.

Once OpenCV and Tesseract were operational, compiling OpenALPR was a breeze. The entire process of installing all the dependencies and compiling everything (correctly) took a full working day. After installation, OpenALPR was able to start recognizing license plates immediately because it comes bundled with training data for both North American and European plates. The result of an OpenALPR output string contains *a*) the extracted license plate characters, *b*) the confidence in the characters; and *c*) the execution time.

B. Intrada ALPR

Intrada ALPR uses the SOAP protocol for exchanging information [13]. To create SOAP requests, we used the `SoapClient` class available in PHP to create, send, and receive the output of our request.

Intrada requires a username and password to process images. The SOAP API allows for the sending of the username and password as part of each request object for authentication. Then, Intrada runs their proprietary algorithm on its own servers before returning a result string back to our web application. The result string contains colon delimited fields that carry *a*) the extracted license plate characters, *b*) the confidence in the extraction, *c*) the state of origin of the license plate, *d*) the confidence in the state of origin, *e*) coordinates of the boundary box of the detected license plate, and *f*) the execution time.

C. MySQL Interface

The resulting plain text representations of the license plate are stored in MySQL database tables. There exists one table for OpenALPR recognitions, and one table for Intrada ALPR recognitions. Each table has the same structure: it stores the license plate and a reference to the 'images' table where the image filepath is stored. The Intrada ALPR table also contains a column for the recognized state. These fields are then used by the web application to be displayed to administrator users.

3.4.6 ALPR Evaluation Environment

We evaluated our two programs (OpenALPR and Intrada ALPR) by having each service process three different data sets: the *unaltered images*, the *cropped images*, and the *compressed images*. OpenALPR ran on the ACMx server, while Intrada ALPR ran as a cloud service, so its environment was outside of our control. Although each service offered methods to enqueue an entire directory of images and process them in bulk, we opted to evaluate them individually, because our system would be getting images from the sensors individually in a real-world deployment.

3.5 Web Application

Prior to the start of our project, the ACMx group created a web application designed to display the data collected from the sensors in their Smartlots parking lot monitoring

system. Since the current site was essentially just a shell with no data, one main goal of this project was to improve the website. Now that we have accomplished that goal, the new web application consists of four main parts:

1. Front-facing, publicly-accessible Smartlots website,
2. Back-end HTTP interface,
3. MySQL database, and
4. Administrator website.

The Smartlots website makes use of the magnetometer sensor data; it provides maps showing the number of cars in each parking lot, maintains lists of parking lots, and lists usage statistics about each lot. The HTTP interface provides a way for data to flow in and out of the application. The MySQL database provides permanent storage of the data collected by the sensor network. The administrator website provides tools for system administrators. These tools are described in detail below.

Before we started this project, the ACMx group had already created the publicly-accessible Smartlots webpage. Therefore, we chose to focus on creating the administrator side of the site, and improving the back-end of the entire site to make it more secure and easier to extend in the future. The homepage of the Smartlots site is shown in Figure 5. You can visit this site at `http://acmxlabs.org/smartlots/`

The web application subsystem encompasses several components, each of which helps to accomplish our goal of providing campus members with a better parking experience. The interfaces for both general users and administrators make the data we collect useful and easy to access. The web application itself is written with PHP on the back end and HTML5/CSS3/JavaScript on the front end. The data from the sensors is stored in a MySQL database, which is described in detail below. The following subsections describe the various pieces of the site that we completed during this project.

Figure 5: The Smartlots Homepage

3.5.1 HTTP Interface

The base station Raspberry Pi needs some way to communicate its data with the web application. The simplest way to meet this need is to provide an HTTP interface on the server. We use HTTP to handle requests from the front end of the web application, so it made sense to also allow the base station to communicate with our server through HTTP. We designed our backend services to be as RESTful [14] as possible. We used the PHP library Slim [15] to create our REST API. For each table in the database, our API includes code to create, read, update, and delete the items, via HTTP POST, GET, PUT, and DELETE requests, respectively. For example, when a car is detected by the magnetometer sensor, the data is received by the base station Raspberry Pi, which then sends an HTTP POST request to http://acmxlabs.org/smartlots/fioData. Upon receiving this request, the server runs the appropriate PHP code to add new entries to the database tables that store the magnetometer and car count data. Furthermore, when the "View Licenses" web page on the administrator website needs to display a table with all the license plate data, it sends an HTTP GET request to http://acmxlabs.org/smartlots/licenses.

Our API uses built-in PHP security to protect certain routes from being accessed by unauthorized users. To POST data to any of the routes, an administrator username and password is required. The routes that service the publicly-accessible Smartlots site are not authenticated, as we want anyone to be able to access that part of the site.

3.5.2 Database

The server stores all application data in a MySQL database. This database stores information about the sensor network, along with the data collected from the sensors. The database schema is shown in Figure 6. The sensors table stores information about the individual sensors, and includes fields for car count and previous car count, so that the web application knows how many cars have passed by each sensor. The lots table maintains a list of lot names, and the sensormapping table is a join table that maps each sensor

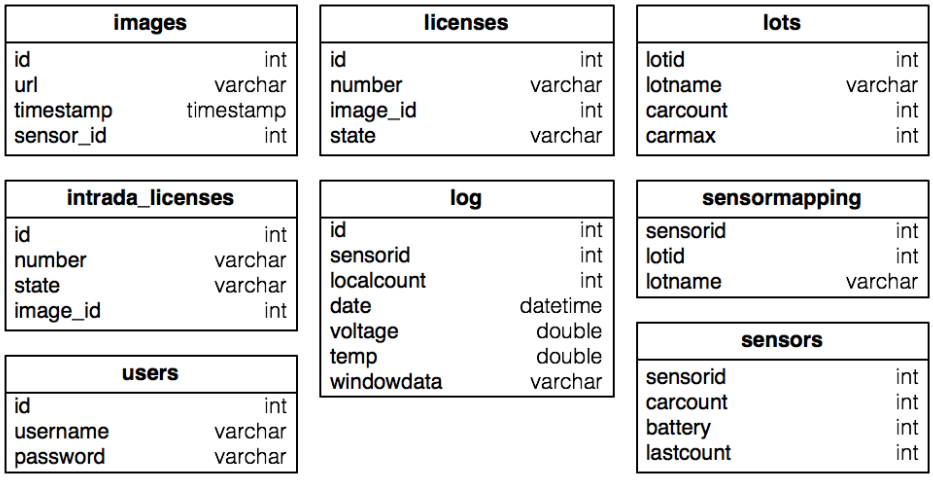

Figure 6: MySQL Database Schema

to the lot where it is deployed. The lots table also keeps the current car count for each parking lot, which is used by the front page of the Smartlots homepage. The images table maintains a list of image urls, and keeps track of which sensor the image came from. This table is updated whenever a new image is received from the base station Raspberry Pi. The licenses and intrada_licenses tables are also updated whenever a new image is received. These tables have an identical format, but store different data depending on the results returned upon running the two license plate recognition programs. The log table keeps a running list of all magnetometer and count data received from the sensor nodes.

3.5.3 Administrator Website

Much of the work done on the ACMx server during this project involved developing and deploying the administrator website. The site serves as an easy-to-use interface that provides tools for administrators (our group members and ACMx group members). These tools allow administrators to view the data collected by the magnetometer sensors and the image-acquisition Raspberry Pis, view health data about the sensors, and manage user

accounts. Each component of the administrator website is described below.

A. User Account Management

Administrators are able to create, delete, and view other administrator accounts. Maintaining user accounts is important for this system because our API requires user authentication for certain HTTP routes, as described above. We have created a dedicated user that is allowed to access all the back-end routes; this user is the one that is used by the base station Raspberry Pi to POST its magnetometer and image data to the server. Other administrator accounts are used to gain access to the administrator site, and to provide authentication when testing specific routes.

B. License Plate Viewer

As mentioned in section 3.5, the license plate images are translated by the ALPR software from an image into a string of plain text, and stored in the MySQL database on the ACMx server. One of the main functions of the web interface is to display this data to administrators. The page that displays the license plate data is a simple interface that shows every license plate number we have collected, along with the time it was collected, the parking lot it came from, and the original image that the license number came from. The data is divided into two tables: one for the data returned from the OpenALPR software, and one for the data returned by the Intrada software. The entries are sorted by most-recent first. For this project, we were only interested in creating a proof-of-concept type interface that shows we have the capability to display the collected plate data in a useful format; in the future it could be useful to cross-reference our data with the campus parking department's database to check if a car has entered a lot without a permit. Figure 7 shows a screenshot of the license plate viewer page in the administrator interface.

C. Sensor Data Viewer

Another feature of the administrator webpage is the page that allows administrators to view sensor data. This page maintains a table showing each sensor, its current and last count, its battery level, and the lot where it is deployed. Although this is a simple page,

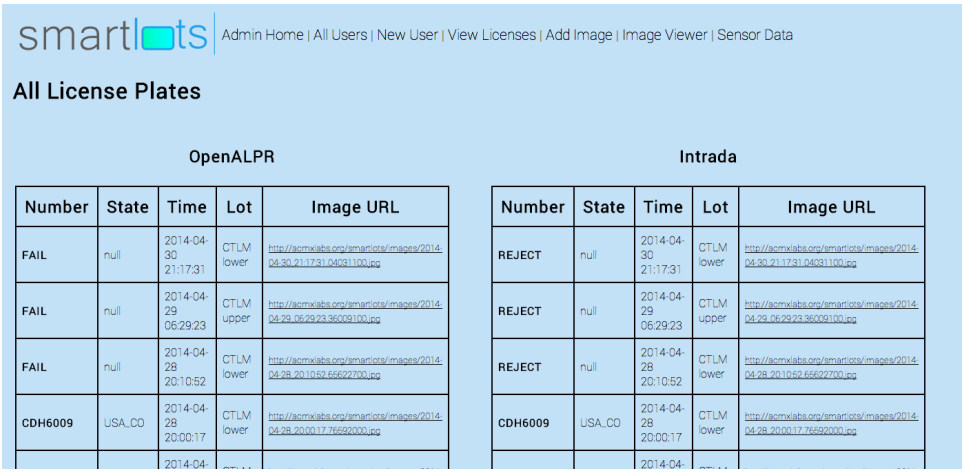

Figure 7: The License Plate Viewer Page

it provides useful information to the system administrators. Future additions to this page will include a way to add and remove sensors from the database.

4 Results

4.1 Image Processing Results

The original goal in image processing was to crop the image down from a large image to a small, manageable image of just the license plate. This software does exist, and is used in many applications, such as [16] and [11]. However, this software is very hard to implement in practice. Most applications use contour mapping to detect the license plate, but this relies on having an image where the license plate is one of the prominent features in the image. When contour mapping methods were applied to the images in this project, most commonly the contours detected pebbles on the ground in the concrete or lines in the parking lot instead of license plates. While successful in other applications, it did not prove to be a good method in this instance.

With contour mapping out of the question, the chosen method for image processing

was converting the image to grayscale, resizing it to reduce the file size, and then naively cropping the top and bottom of the picture to eliminate the ground below the car and the sky above. While not 100% effective, it proved to be the best solution in testing.

4.2 ALPR Results

Three data sets were run through each ALPR system. In each case, Intrada ALPR was able to recognize at least twice as many plates as OpenALPR.

Data Set	Correct Recognition	Total Images	Percent Correct
Unaltered images	20	52	38.46 %
Cropped images	4	52	7.69 %
Compressed images	4	27	14.81 %

Table 1: OpenALPR Results

Data Set	Correct Recognition	Total Images	Percent Correct
Unaltered images	34	52	65.39 %
Cropped images	12	52	23.08 %
Compressed images	8	27	29.63 %

Table 2: Intrada ALPR Results

As seen in Table 1 and Table 2, Intrada ALPR outperformed OpenALPR in each case in terms of accuracy. In terms of speed, OpenALPR always outperformed Intrada ALPR by at least a factor of three. In some cases, OpenALPR was better by a factor of 110.

These results were calculated by determining whether or not the ALPR software correctly identified *any* license plate in the photo. It is the case that, for certain out of state plates or nonstandard plates (e.g. Disabled Veteran), the ALPR programs will not identify the primary, or largest, license plate in the photo. However, both programs were successful in recognizing a much smaller, but standard, license plate. Figure 8 shows an example image where a New Mexico plate is ignored in favor of a smaller plate in the background.

Figure 8: The foreground license plate goes unrecognized in favor of a smaller plate in the background

4.3 REST API and Route Security Results

We performed a few manual tests to evaluate the performance of our REST API. These included attempting to access the routes provided in the API through different interfaces. The API proved to work successfully when accessed through a web browser (i.e. all GET requests returned the correct resource when accessed in the URL bar of a browser. The GET routes also all proved to be accessible through curl, and via JavaScript/AJAX XMLHttpRequests. Our base station Python code was also able to successfully send all necessary requests to the API. The front-end general user interface and admin website also makes use of the API, and performs as expected. All in all, using Slim was a good choice for creating the REST API, and we have yet to see it fail when the requests are correctly formed. The security implemented was also very successful. We tried accessing all the secure routes through the various interfaces previously described (browser, curl, PHP, AJAX), and none of them worked without providing a username and password. The all worked successfully when a valid administrator username and password was provided.

4.4 Overall System Performance

To test that our various subsystems performed as they were supposed to, we formulated a few manual tests to make sure everything was connected. We made sure the various pieces were in place before testing the whole system, and it seems that a few pieces work more consistently than others. As described above, when a request is sent to the server, the REST API we developed is very robust, and always processes the request in the expected way, unless the request itself is not in the correct format. This means that images and sensor data that are sent to the server are consistently placed into the correct database tables, and are immediately available for access via the administrator interface and the front-end Smartlots website. The two ALPR software packages also consistently perform their jobs in the expected way. When an image arrives at the server, it is passed to both OpenALPR and the Intrada ALPR. Both programs always return some results, although,

as mentioned above, they may not be correct. If an image fails to be recognized by either software package, a RECJECT or FAIL string is returned, and our database and web application reflects that.

The sensor nodes are fairly finicky right now, and do not always work as expected. Because both the Raspberry Pi and the Arduino Fio share a common power source, the Pis sometimes shut down unexpectedly. The magnetometers do not always pick up metal objects passing over them, and therefore we do not always get data when we should. Because we did not have time to implement sending the images through the XBees, our sensor nodes always have to be in range of a Wi-Fi connection, or plugged into an Ethernet port. This would be unrealistic for a real-world deployment. All in all, our sensor nodes need some work before they will be ready to deploy. The rest of the system above the sensor nodes works consistently, and we can confidently say that once the issues with the sensors are worked out, we will be able to consistently send both magnetometer and image data from the sensors to the base station, and have everything be updated on the website in near-real time.

5 Future Research

There is substantial future research that can be performed to add features and improve the performance of our system. This area of research appears very promising, as similar systems may be applied and adapted to many different settings.

5.1 ALPR Training

The accuracy rate for our ALPR software is not as high as we had originally hoped it to be. OpenALPR offers a separate program, 'train-ocr' to help build definition files for certain geographic regions. Because it initially came seeded with training data for the entire continent, the recognition of Colorado plates may be improved if we create a training file for only Colorado.

Intrada ALPR cannot be configured by the end user (our team). We can, however, send sample images from our deployment to Q-Free, where our files can be used to train our account for better recognition.

Because we ended up using two ALPR systems instead of one, it would be beneficial to utilize both to provide a confidence level for our project. Having validation from both ALPRs would likely mean the characters that were extracted were correct. Extractions that were not correct could be flagged in the database for human intervention.

A proposed long-term goal for this project was to integrate license recognition with CSM's parking database to identify users who do not have valid parking permits. With the work that has been completed, we are now at a place where we could begin coding an interface to pass license data to Parking and Facilities.

5.2 Validating and Adjusting Lot Capacity Data

We expect that our first deployment of this system may not be 100% accurate at detecting all vehicles that enter and exit lots. Therefore, we would like to create a system to periodically collect overhead images of the parking lots from a camera mounted on the base station Raspberry Pi. These images would serve as a visual representation of how many cars were actually in the parking lot at the time the image was taken. If this feature is implemented, administrators will be able to periodically compare the overhead images to the collected data from the sensor network to make sure the numbers being computed by the system accurately represent the car counts of the lots. A page would be added to the web interface to allow administrators to pull up the image for any lot with an overhead camera and see the numbers for that lot right next to the image. If any discrepancies are found, the administrator would be able to overwrite the count in the database to reflect reality.

Such an interface would be useful in helping to calibrate the system. For example, if we find that some sensors consistently report less cars in the lot than the image shows,

these sensors can be fine-tuned to make sure fewer cars slip through unnoticed. Another improvement to work in this area could include an automated version of this reconciliation process, where a computer program could compare the overhead images to the sensed data and make any necessary corrections to the car count in the database.

5.3 Real-world Deployment

Due to time-constraints, we were unable to test our system in a real-world environment. We had initially planned to deploy a working version of our system in two parking lots on the CSM campus. Even though we never deployed our system, we were able to get everything connected together in the lab as described in section 2, so that data collected by the sensors flowed all the way up through the system and was displayed in the web application. Our plans for a future deployment are as follows:

The ACMx group, along with members from our group, plans to deploy a version of our system at the CTLM Upper and Lower parking lots at CSM. These lots were chosen because of their accessible location, unobstructed visibility from the CTLM building, and frequent use. Both parking lots have one entryway, each of which will be equipped with a sensor node. The junction between the two lots is also of interest, because we want to know when a car moves from the upper lot to the lower lot, and vice versa. The locations of the entrances and junction are shown in Figure 9. The locations for the three sensor nodes benefit from having a clear view of the sky for increased photovoltaic efficiency.

The central base station, as described in section 3.3, will be placed in an upper window of the CTLM building, allowing it to take a clear overhead picture of at least one of the two parking lots. The first deployment may only have one overhead camera to start off with, although ideally we would have two - one for each lot. Several potential locations exist that allow for a clear line-of-sight to all sensor nodes; one potential location is shown in Figure 9. The location of the base station must put it in range of the XBees mounted on the sensor nodes, to allow data to be transferred from the detection subsystem to the

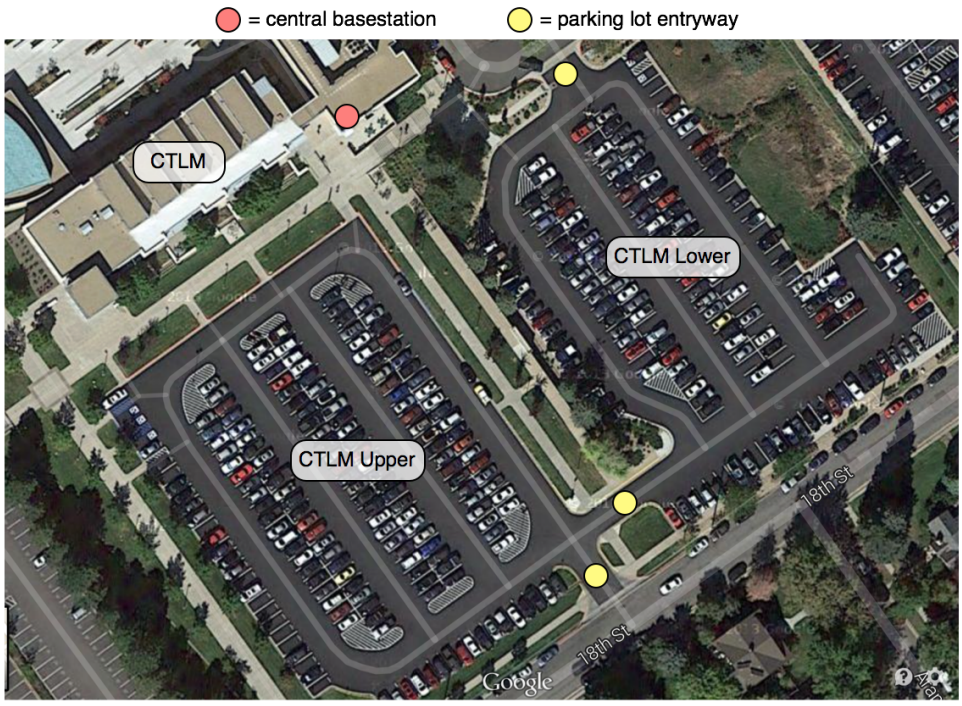

Figure 9: Deployment Location

base station. The locations being considered must also provide internet access to the base station Raspberry Pi through an Ethernet port.

5.4 Xbee Image Transmission

Currently, our system relies on a Wi-Fi network for transmitting license plate images from the Acquisition Raspberry Pi. Future systems would benefit from having such images funneled and transmitted through the Xbee radio, because transmission range would be significantly improved, and eliminates the need for the sensor node to be in range of a Wi-Fi network in order to function. We envision that this would be implemented by sending a bitstream of the image from the Raspberry Pi to the Arduino Fio through a serial link that would then be packetized by the Arduino Fio, and forwarded to the base station via the Xbee radio. However, such a feature would have to be carefully thought out so as to not obstruct the Arduino Fio's automobile detection operations and not congest the Xbee data link. One concern with transmitting images over the XBee comes from the fact that we did not implement any collision avoidance algorithms, and transmitting the images over the XBees would greatly increase the chance of collisions when multiple sensor nodes are sending data to the base station at the same time.

5.5 Image Acquisition and Processing

The current solution for imaging is solely based on the limited set of test images we were able to acquire, and is likely not robust enough for a full deployment. Once a successful deployment has been achieved, the image acquisition time needs to be fine-tuned so the license plate is always captured in the image. If the naive cropping strategy for image processing is maintained, then it can be expanded upon to eliminate pieces on the left and right of the car in order to further reduce file size.

While contour mapping proved to be unsuccessful when processing the full images, further work should be done, once image acquisition has been fine-tuned, to crop the

image and then apply contour mapping to the cropped image. The method failed originally because the license plate was not featured prominently in the image, and cropping the image more effectively could help the contours of the license plate better stand out.

6 Summary

This project extends the work done by ACMx, with the goal of creating the first comprehensive parking management system for the Colorado School of Mines. The system's expansions cover interfacing with the existing automobile detection network described in [1], acquiring images of license plates, processing said images, and presenting the resulting data in a web application.

Image acquisition is done by Raspberry Pis stationed at parking lot entrances. These images are sent to a central base station Raspberry Pi, which then uploads the data to the server via our RESTful API. Once on the server, additional image processing is done to retrieve the characters composing the license plate. This data is then stored in a database for use in the web application.

Acquiring images of license plates includes listening for an interrupt from the automobile detection sensor, temporarily storing the image, pre-processing the image on the Raspberry Pi. By completing some processing of the image on the Raspberry Pi, we reduce the size of data we need to transmit, which saves battery power while also reducing the transmission time, thereby improving system latency and throughput.

The web application serves to show the state of the parking lots, and provides an interface that allows administrators to view license plate data and sensor data, and manage user accounts.

Once deployed, the Smartlots system, augmented with our additional license plate extraction subsystem and improvements to the web application, will offer the Mines campus an accessible way to manage campus parking lots. Faculty and students will be able to use this to find open parking and save time. Future applications could include interfacing

with Parking Services to locate illegally parked vehicles or to gather data regarding campus parking needs.

References

[1] R. Stillwell, A. Wilson "Magnetometer Parking Sensor," *EGGN 383 Final Project, Colorado School of Mines*. December 12, 2013.

[2] R. Stillwell. (2014). *Parking Sensor Wiki* [Online]. Available: `http://github.com/ColoradoSchoolOfMines/parking_sensor/wiki`

[3] K. V. S. Hari. (2011). *Study of Magnetometer Signals for Vehicle Detection and Classification* [Online]. Available: `http://www.intranse.in/its1/sites/default/files/D2-S2-03_Study%20of%20Magnetometer%20Signals%20for%20Vehicle.pdf`

[4] *OpenCV* [Online] Available: `http://opencv.org/`

[5] *Raspbian* [Online] Available: `http://www.raspbian.org/`

[6] OpenALPR. (2014). *Train OCR* [Online]. Available: `https://github.com/openalpr/train-ocr`

[7] OpenALPR. (2014). *OCR Technology* [Online]. Available: `http://www.q-free.com/product/ocr-technology/`

[8] Ondrej Martinsky. (2007). *Automatic Number Plate Recognition System* [Online]. Available: `http://javaanpr.sourceforge.net/`

[9] C. Nikolaos, I. Anagnostopoulos, V. Loumos and E. Kayafas, "A License Plate-Recognition Algorithm for Intelligent Transportation System Applications," *IEEE Trans. Intelligent Transportation Systems* vol. 7, no. 3, Sept. 2006.

[10] Q-Free ASA. (2014). *OCR Technology* [Online]. Available: `http://www.q-free.com/product/ocr-technology/`

[11] E-470 Public Highway Authority. (2014). *How E-470 Works* [Online]. Available: `https://expresstoll.com`

[12] The 3M Company. (2014). *ALPR Technology* [Online]. Available: `http://solutions.3m.com`

[13] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte and D. Winer. (2014). *Simple Object Access Protocol (SOAP) 1.1* [Online]. Available: `http://www.w3.org`

[14] *Representational State Transfer* [Online] Available: `http://en.wikipedia.org/wiki/Representational_state_transfer`

[15] *Slim* [Online] Avaliable: `http://www.slimframework.com/`

[16] S. Du, M. Ibrahim, M. Shehata and W. Badawy, "Automatic License Plate Recognition (ALPR): A State-of-the-Art Review," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 23, no. 2, Feb. 2013.

Appendix 1 — Intra-team Work Distribution

This work described in this project was completed by six people. The four authors of this paper contributed the bulk of the work, and two ACMx group members also helped us, mainly by getting us up to speed on the state of the ACMx project when we started, and continuing to be our points of contact to the ACMx group. The work was broken down as follows:

WSN Group Members

Brandon Rodriguez worked with the license plate recognition software. All of the items he worked on are described in section 3.4 and section 4.2. Brandon did a lot of work researching ALPR software, and made the final decision to use OpenALPR and the Intrada ALPR software for our project. Basically, when reading this paper, realize that if something pertains to license plate recognition, Brandon was responsible for getting it all working. Brandon also worked with Taylor to write some of the server-side PHP code that gets the images from the HTTP requests and runs them through the ALPR software.

Taylor Sallee wrote all code for the web application, excluding the front-end "Smartlots" part of the site that was already in place before the start of this project. Roy Stilwell from the ACMx group wrote the code for that part of the site. Taylor's contribution was the entire administrator interface, the RESTful API that services the website and the Raspberry Pis when they make POST requests, the user authentication and PHP security added to all sensitive routes created, the database tables for all data that our group was adding (images, licenses, intrada_licenses, users), and the PHP code for viewing, creating, updating, and deleting data in these tables. He also collaborated with Andrew Koelling to write the base-station Python code that listens for data from the Arduino Fios and Raspberry Pis, and sends that data to the server via HTTP requests. Some of this code was in place before the project, but Taylor and Andrew refactored almost all of it, and added in the parts to handle the image data, and the part that differentiates the Arduino Fio data from the Raspberry Pi (image) data. Taylor also worked with Santiago and Kevin on the sensor Raspberry Pis, making sure that everything was installed correctly, and testing the top-to-bottom functionality of the system (that is, make sure data is getting from the sensors all the way to the web application). Taylor's work is described in sections 3.3, 3.5, 4.3, and 4.5.

Kevin Emery worked with the image processing on the image acquisition Raspberry Pis. He wrote Python scripts that make use of the OpenCV library to process images, which mainly includes cropping and compressing to sizes that are reasonable for transmis-

sion over the XBee radio. He explored contour mapping as one possibility for reducing image size while still maintaining a readable license plate, but found the current OpenCV implementation to be ineffective given the provided images. His work is detailed in sections 3.2 and 4.1

Santiago Gonzalez worked primarily with the hardware needed to get this system working. He built the sensor nodes that include both the Arduino Fio and the Raspberry Pi. He deployed the code written by Roy Stilwell to the Arduino Fios, modifying slightly to suit our purposes. He also did most of the work deploying code to the image Acquisition Raspberry Pis, and he was the one that wrote the code that makes the Pis take an image when the magnetometer detects a car. His work is described in section 3.1.

ACMx Group Members

Roy Stilwell is the head of the ACMx group. Although he wasn't available to help us very much during the course of this project, much of what he and the other ACMx group members did laid the groundwork for what we accomplished in our project. The code that is listed below that runs on the Arduino Fios was written almost entirely by Roy. Roy also designed the Smartlots webpage, and wrote some of the code that is used on the base station Raspberry Pi.

Drew Koelling was a huge help to our group during this project. He met with us several times during our hackathon, even though he did not have as much stake in the success of this project, given that his grade did not depend on what we got done. He primarily helped us understand the base station code, and the existing web interface, and provided Taylor with the information necessary to create a useful REST API. He also helped rewrite some of that base station code. He was overall a great point of contact between our group and the ACMx group, and he was very helpful and good-natured the whole time. We want to explicitly thank him for all the help he gave us in this project.

Appendix 2 — Code Listings

Arduino Fio Sensing Firmware

```
/*
Arduino, HMC5883, magnetometer, XBee code
 by: Roy Stillwell, Andrew Wilson
 Colorado School of Mines
 created on: 10.30.13
 license: This work is licensed under a Creative Commons Attribution
    license.
 updated: 3.19.14 Roy Stillwell
 updated: 4.25.14 Santiago Gonzalez

 Arduino code example for interfacing with the HMC5883
 by: Jordan McConnell
 SparkFun Electronics
 created on: 6/30/11
 license: OSHW 1.0, http://freedomdefined.org/OSHW

 Analog input 4 I2C SDA
 Analog input 5 I2C SCL

 EEPROM code adapted from Tomorrow Lab
 Developer: Ted Ullricis_measuringh <ted@tomorrow-lab.com>
 http://tomorrow-lab.com


 ******************
 HOW IT WORKS:
 ******************

 An array of 'baseline' or nominal values -essentially when the sensor does
     NOT have a vehicle over it- is gathered initially.
 This baseline can be 'sized' to allow for fine tuning using the '
    baselineSize' variable in the 'Variables you can change' section.
 After the initial buffer is filled, its average is calculated.
 A sensor value is then read and compared to the average and a threshold.
    If it is outside this threshold, a counter is started.
 Once The counter is larger than the window size ( a very good indication
    of a car), we have detected a vehicle!
 We then grab the last three values in the window and check to see if it is
     greater or smaller than the baseline average.
 This gives us a direction of the vehicle.

 The code <will be> designed to re-calibrate the 'baseline' every 10
    minutes (basically in the event a sensor is hit wiht a solar storm or
    something).
```

```
 Currently this only uses the Y axis data (as that may be all we need)
 Using datasets taken at the CTLM exits and entrances, it correctly
     calculates entrances and exits.

 */

#include <Wire.h> //I2C Arduino Library, required for interface
    communication with HMC5883 Device
#include "stats.h"  //a custom library for doing Average and Standard
    deviation calculations.
#include <cmath>  //Standard cmath library
#include <string>
using namespace std;
#include <avr/wdt.h>
#include <avr/sleep.h>
#include <avr/interrupt.h>



//----------Variables you can change--------------------
double recalibrateTime = 600000; //time in seconds. 600000 = 10 min -- used
     to auto-recalibrate sensor, if time is greater than 10 minutes,
    recalibrate
#define baselineSize 100  //Size of baseline to use for baseline values
#define windowSize 30  //Size of 'window' to use for previous values of the
     sensor to be considered in calculating a positive detection.
#define windowConsidered 3 //Consider the first n numbers in window to
    determine direction
double carThreshold = 15.0; //Used to sort out car 'hits'. Anything above
    or below this 'threshold' is counted as a hit
double pingTime = 60; //(in seconds) Used to make sure sensor is still
    alive after specified time
//------------------------------------------------------


const int battPin = A0;
const int ledPin = 13;
const int interruptPin = 12;
const int XBeeSleep = 2;                // Connect to XBee DTR for
    hibernation mode
const int waitPeriod = 1;               // Number of 8 second cycles before
    waking
// up XBee and sending data (8*8 = 64 seconds)
// Variable Definition

//double Vcc = 0.0;
double Temp = 0.0;
double batt = 0.0;

double pingCounter;    //Used to count cycles so system can 'ping'
```

```
    basestation every 30 seconds
int localCarCount=0;
long startTime ;                    // start time for the algorithm watch
long elapsedTime ;                  // elapsed time for the algorithm
bool didWeCalculateAveDevAlready, saidit = false;
double lastCalibrateTime, stdDev;
double baselineAvg = 0;
double calibrationPercentDone;
int x, y, z;
int windowTotal = 0, count = 0, detectorCount = 0;
int baseline[baselineSize];
int windowy[windowSize];
int windowx[windowSize];
int windowz[windowSize];
String yomama_data;


#define address 0x1E //0011110b, I2C 7bit address of HMC5883

String getWindowData(int window[]) {
  String data = "";
 for (int i = 0; i < windowSize; i++ ) {
   data+= String(window[i]) + "␣";
 }
 return data;
}


// See: http://code.google.com/p/tinkerit/wiki/SecretVoltmeter
//double readVcc() {
//  signed long resultVcc;
//  double resultVccFloat;
//  // Read 1.1V reference against AVcc
//  ADMUX = _BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
//  delay(10);                            // Wait for Vref to settle
//  ADCSRA |= _BV(ADSC);                  // Convert
//  while (bit_is_set(ADCSRA,ADSC));
//  resultVcc = ADCL;
//  resultVcc |= ADCH<<8;
//  resultVcc = 1126400L / resultVcc;     // Back-calculate AVcc in mV
//  resultVccFloat = (double) resultVcc / 1000.0; // Convert to Float
//  return resultVccFloat;
//}

// See: http://code.google.com/p/tinkerit/wiki/SecretThermometer
double readTemp() {
  signed long resultTemp;
  double resultTempFloat;
  // Read temperature sensor against 1.1V reference
  ADMUX = _BV(REFS1) | _BV(REFS0) | _BV(MUX3);
```

```
  delay(10);                              // Wait for Vref to settle
  ADCSRA |= _BV(ADSC);                    // Convert
  while (bit_is_set(ADCSRA,ADSC));
  resultTemp = ADCL;
  resultTemp |= ADCH<<8;
  resultTempFloat = (double) resultTemp * 0.9338 - 312.7; // Apply
      calibration correction (Roy S)
  resultTempFloat = resultTempFloat * 1.8 + 32.0;  // Convert to F
  return resultTempFloat;
}

double readBatt() {
  batt = analogRead(battPin) * .00324 * 2 ;
  return batt;
}

void configureSensor() {
  Wire.begin(); //Initalize I2C interface in Arduino

  //Put the HMC5883 IC into the correct operating mode
  Wire.beginTransmission(address); //open communication with HMC5883
  Wire.write(0x02); //select mode register
  Wire.write(0x00); //continuous measurement mode
  Wire.endTransmission();

  //Crank the speed up to 75Hz read speeds (default is 15Hz)
  Wire.beginTransmission(address);
  Wire.write((byte) 0x00);
  Wire.write((byte) 0x18); //this jumps it to 75Hz
  Wire.endTransmission();
  delay(5);

  //Tell the HMC5883 where to begin reading data
  Wire.beginTransmission(address);
  Wire.write(0x03); //select register 3, X MSB register
  Wire.endTransmission();

  //Read data from each axis, 2 registers per axis (helps initialize
      readings)
  Wire.requestFrom(address, 6);
  if(6<=Wire.available()){
    x = Wire.read()<<8; //X msb
    x |= Wire.read(); //X lsb
    z = Wire.read()<<8; //Z msb
    z |= Wire.read(); //Z lsb
    y = Wire.read()<<8; //Y msb
    y |= Wire.read(); //Y lsb
  }
  //Pause
  delay(1000);
```

```
}

void setup(){
  //Initialize Serial speed
  Serial.begin(57600);
  configureSensor();

  pinMode(interruptPin, OUTPUT);
  digitalWrite(interruptPin, HIGH);

  Serial.print("You␣have␣10␣seconds␣to␣place␣sensor");
  delay(10000);


}

//The main loop to repeat indefinitely
void loop(){

  //Tell the HMC5883 where to begin reading data
  Wire.beginTransmission(address);
  Wire.write(0x03); //select register 3, X MSB register
  Wire.endTransmission();


  //Read data from each axis, 2 registers per axis
  Wire.requestFrom(address, 6);
  if(6<=Wire.available()){
    x = Wire.read()<<8; //X msb
    x |= Wire.read(); //X lsb
    z = Wire.read()<<8; //Z msb
    z |= Wire.read(); //Z lsb
    y = Wire.read()<<8; //Y msb
    y |= Wire.read(); //Y lsb
  }

  //Get temp and voltage data
  //Vcc = (double) readVcc();
  Temp = (double) readTemp();
  batt = (double) readBatt();


  //build the baseline array to be used for average calculation
  if (count < baselineSize) {
    baseline[count] = y;
    if (count == 0) {
      Serial.println("Calibrating...");
    }
      //Echo out to serial how far we are in calibrating.
```

```
      //    double calibrationPercentDone =  baselineSize;
      //    calibrationPercentDone = count / calibrationPercentDone * 100;
      //    Serial.print("Hang on, Calibrating ");
      //    Serial.print(calibrationPercentDone);
      //    Serial.print("% done. ");
      //    Serial.print(x);
      //    Serial.print(" ");
      //    Serial.print(y);
      //    Serial.print(" ");
      //    Serial.println(z);

      delay(40);  //Pause for a bunch of cycles so the values are collected
          over a few seconds.
      count++; //increment counter to fill buffer of size 'baselineSize'

      if (count==baselineSize) {
        Serial.println("Done. Ready to take data.");
      }
    }



  //Figure out baseline values
  if (count == baselineSize && didWeCalculateAveDevAlready == false) {  //
      The baseline array is full, so we can begin collecting data!
    baselineAvg = Average(baseline); //get average

    didWeCalculateAveDevAlready = true;
    startTime = millis();
  }

//  elapsedTime = millis() - startTime;
//  //TODO: write recalibration code
//  if (elapsedTime > recalibrateTime) {
//    //look to recalibrate by building a new temp array
//
//    //check standard deviation of new array against current standard
    deviation
//    //if within limits, use the new calibration data
//    Serial.println("Uh hang on a minute, recalibrating.");
//    startTime = millis();
//    count = 0;
//  }


  bool something = false;  //used to flag whether or not something is over
      the sensor

  //This algorithm calculates whether 'something' was found, and then to be
       sure, starts a detectorCount to verify
```

```
  //there are enough hits to constitute an actual car passing, and not some
      random flux in the Earth's field.
  if ( didWeCalculateAveDevAlready == true && (y >= (baselineAvg +
      carThreshold) || y <= (baselineAvg - carThreshold))) {
//     Serial.print("[something here ");
//     Serial.print(y);
//      Serial.println("]");
    something = true;  //something is probably out there!

    delay(14); //The HMC is set to run at 75Hz, this delays just that
        amount!

    //When 'something' may be out there, there are three scenarios.
    //1) The detector count goes up if there are less 'something' hits than
         the window size,
    //2) The detector count hits the windows size -what we qualify as a
        true car over the sensor!- and as such reports it,
    //3) There wasn't a new event, and the detector count is decremented.
        Once it hits zero, the car is thought to have passed the sensor
    // Effectively, while a car is over the sensor, detectorCount stops
        counting, and sort of cruises at the max window size.  As the car
        passes
    //and sensor values drop to the nominal baseline, it starts clearing
        the detectorCount buffer and readies for a new car.

    // 1) Something may be in the works, we need to count the 'something'
        events to be sure and store it in the window
    // We will use the detectorCount as the indexer
    if (detectorCount <= windowSize) {
      windowy[detectorCount] = y;  //We add the current value to the window
          , for analysis later
      windowx[detectorCount] = x;
      windowz[detectorCount] = z;
    }
    detectorCount++;

    if (detectorCount >= windowSize && saidit==false) { //here we check to
        see if we have detected enough events
      //We did detect a car! Now to see direction
      double windowAve;

      for (int i=0; i < windowConsidered; i++) { //using the first few
          values of the window (the orignal direction of the Magfield)
        //we find the direction.
        windowAve += windowy[i]; //add up all the values
      }
      windowAve = windowAve / windowConsidered;  //generate the average
          value to be used

        //If the first value is less than the average, we have a car
```

```
      heading in! So transmit!
if (windowAve < baselineAvg  ) {
  localCarCount++;

  String windowData = getWindowData(windowy);
  Serial.print("<");
  Serial.print(localCarCount); //output car count
  Serial.print("␣");
  //Serial.print(Vcc); //output battery voltage (in theory)
  //Serial.print(" ");
  Serial.print(batt);
  Serial.print("␣");
  Serial.print(Temp); //output device temp (in theory)
  Serial.print("␣");
  Serial.print(windowData);
  Serial.println(">");

  //TODO Calibrate temp
  //TODO verify vcc output is correct NOTE, it wasn't. went to new '
      batt' calculation


  digitalWrite(interruptPin, LOW); // send interrupt to Raspberry Pi
  delay(500);
  digitalWrite(interruptPin, HIGH);
  pingCounter = 0; // Device has communicated with the base station,
      so reset counter for communication
}
else {  //Otherwise the car was heading out!
  localCarCount--;

  //The data packet will automatically be split into 'frames' by the
      xbee
  //so we will 'encpauslate' our data. ex: < data > .
  String windowData = getWindowData(windowy);
  Serial.print("<");
  Serial.print(localCarCount); //output car count
  Serial.print("␣");
  //Serial.print(Vcc); //output battery voltage (in theory)
  //Serial.print(" ");
  Serial.print(batt);
  Serial.print("␣");
  Serial.print(Temp); //output device temp (in theory)
  Serial.print("␣");
  Serial.print(windowData);
  Serial.println(">");

  //TODO Calibrate temp
  //TODO verify vcc output is correct
```

```
      //Serial.print ("Data: ");
      digitalWrite(interruptPin, LOW); // send interrupt to Raspberry Pi
      delay(500);
      digitalWrite(interruptPin, HIGH);
      pingCounter = 0; // Device has communicated with the base station,
          so reset counter for communication
    }
    //For debugging, we echo the window average and baseline average to
        makes sure the algorithm is working
    //        Serial.print("windowAve: ");
    //        Serial.print(windowAve);
    //        Serial.print(" baselineAvg: ");
    //        Serial.println (baselineAvg);
    saidit = true; //mark that we have said there is a car already, so we
        don't repeat it
  }

}
else {
      //3) An event was not detected, so the detectorCount begins
          decrementing its values
          if (detectorCount > 0) detectorCount--;
          //Once it zero, the algorithm is essentially ready for a new
              car.
          if (detectorCount == 0) saidit = false;

          //This code does is used for diagnostics.  It essentially sends
               a 'heartbeat' to the base station
          if (pingCounter >= (pingTime*200)) { // is about 1 second with
              this program code
            //To be sure everything is working (mostly for diagnostics,
                ping every 30 seconds).
               //The data packet will automatically be split into '
                   frames' by the xbee

      //so we will 'encpauslate' our data. ex: < data > .
      String windowData = getWindowData(windowy);
      Serial.print("<");
      Serial.print(localCarCount); //output car count
      Serial.print("␣");
      //Serial.print(Vcc); //output battery voltage (in theory)
      //Serial.print(" ");
      Serial.print(batt);
      Serial.print("␣");
      Serial.print(Temp); //output device temp (in theory)
      Serial.print("␣");
      Serial.print(windowData);
      Serial.println(">");
            pingCounter = 0;
            //delay(40);
```

```
        }
        pingCounter++;
  }

}
```

Image Acquisition

```
#!/usr/bin/env python

from time import sleep
import os
import RPi.GPIO as GPIO
import subprocess
import datetime

GPIO.setmode(GPIO.BCM)
GPIO.setup(24, GPIO.IN)

count = 0
up = False
down = False
command = ""
filename = ""
index = 0
camera_pause = "500"

def takepic(imageName):
  print("picture")
  command = "sudo␣raspistill␣-o␣" + imageName + "␣-q␣100␣-t␣" +
      camera_pause
  print(command)
  os.system(command)

while(True):
  if(up==True):
    if(GPIO.input(24)==False):
      now = datetime.datetime.now()
      timeString = now.strftime("%Y-%m-%d_%H:%M:%S")
      filename = "photo-"+timeString+".jpg"
      takepic(filename)
      subprocess.call(['./processImage.sh'])
  up = GPIO.input(24)
  count = count+1
  sleep(.1)
print "done"
```

Image Processing

```
import sys;
from cv import *;
import getopt;

# Resizes the image
def resizeImage(rootdir, filename, prefixString, resizeFactor):

    # Loads the original image
    original_img = LoadImage(rootdir + filename)

    # Creates a destination image
    img = CreateImage(GetSize(original_img), 8, 1)

    # Convert the image to gray_scale
    CvtColor(original_img, img, CV_RGB2GRAY);
    size = GetSize(img)

    # Set up the resized image size
    newImage = CreateImage(( size[0] / resizeFactor, size[1] / resizeFactor
        ), img.depth, img.nChannels)

    # Resize the image
    Resize(img, newImage)
    size = GetSize(newImage)

    # Crop the image
    newImage = newImage[size[1]/3:4*(size[1]/5), 0:]

    # Write the image to files
    SaveImage(prefixString + filename, newImage)


# Gets the appropriate command line arguments
def getCommandLineArguments():
    filename = ""
    prefixString = ""
    try:
        opts, args = getopt.getopt(sys.argv[1:], "n:p:", ["name=", "prefix=
            "])
    except getopt.GetoptError:
        sys.exit(1)
    for opt, arg in opts:
        if opt in ("-n", "-name"):
            filename = arg
        elif opt in ("-p", "-prefix"):
            prefixString = arg

    return filename, prefixString
```

```
rootDirectory = ""
resizeFactor = 3

filename, prefixString = getCommandLineArguments()

resizeImage(rootDirectory, filename, prefixString, resizeFactor)
```

Image Sending

```
# Python script for the pi to post to website after getting an image

import sys
import requests

# This file will post an image to the ACMx server. It uses the 'requests'
# module for Multipart-Encoding of the image file. See
# http://docs.python-requests.org/ for documentation. Prints server
   response
# to the console.
#
# Args:
#               sensorID: The ID of the sensor who recorded the image.
#               image: A file handle that was opened for binary reading.
#                                   e.g. f = open('img.png', 'rb')
#
#       Returns:
#               (nothing)
#
imagePath = sys.argv[1]
sensorID = 2

print imagePath

image = open(imagePath, 'rb')

url = "http://wsn:raspberryp1@acmxlabs.org/smartlots/pidata"

# Construct the file payload.
fPayload = {"image": image}

# Construct normal form variables payload.
dPayload = {
        "id": sensorID
}
# Create a requests object to handle all the urllib2 stuff.
r = requests.post(url, data=dPayload, files=fPayload)

# Print out the server's response.
print r.text
```

Base Station Code to Receive Fio and Pi Data from XBee

```
#! /usr/bin/python

"""
receive_samples_async.py

By Paul Malmsten, 2010
pmalmsten@gmail.com

This file reads from the serial port and asynchronously processes the data
received from a remote XBee.
"""

from xbee import XBee
import time
import serial
import update

PORT = '/dev/tty.usbserial-A602TT7M'
BAUD_RATE = 57600

# A dictionary (map) that stores the Fio packets, with each packet
# in the map growing until the full data is received
FIO_DATA_BUFFER = {}

# A dictionary (map) that stores the Pi packets, with each packet
# in the map growing until the full image is received
PI_DATA_BUFFER = {}

# A dictionary that stores the lengths of each image sent by the Pis
PI_STREAM_LENGTHS = {}

# Open serial port
ser = serial.Serial(PORT, BAUD_RATE)

# Takes a data stream from a Fio and sends it to the server for storage
def parseFioDataAndSend(sensorId, dataStream):
        # Remove the < and > from of datastream
        dataStream = dataStream.replace("<", "")
        dataStream = dataStream.replace(">", "")

        valuesplit = dataStream.split()
        carcount = valuesplit[0]
        voltage = valuesplit[1]
        temperature = valuesplit[2]
        window = valuesplit[3:]
        update.postFioData(sensorId, carcount, voltage, temperature, window
            )
```

```
# Takes a data stream from a Pi and sends it to the server for storage
def parsePiDataAndSend(sensorId, dataStream):
        f = open("kitten.jpeg", "rb")
        update.postPiData(sensorId, f)

"""
The following function 'message_received(dataPacket)' is described below:
Since XBee's will split the data along frames, we only want to send data to
     website once we have all the data
To solve this, each data stream will end with a specific char, in this case
     it is '>'.
data streams will be

        header_data < localcount voltage temperature window_data
            window_data window_data ... window_data >

Occasionally debug statements will be sent. all debugs will be surronded by
     square brackets  [ ]
Debug statements will be only a single xbee frame, and no data statments
    will be in the same window as a debug statment
therefore all debug statments will be ignored can simply have the entire
    stream thrown out
debug statments will be

        header_data [ DEBUG_STATEMENT ... DEBUG_STATEMENT ]

Because the entire data stream will not exist in a single XBee frame, when
    the id is extracted from the header,
the stream will be added to a map. If there is already an id in the map, it
     will append the current frame into the data
already in the dictonary
When the terminating character is found (the > char) it will append the
    current frame to the one in the map and
then send the data in the map. The id will be removed from the map when the
     data is sent to site.

The name of the map is FIO_DATA_BUFFER and exists in global scope.
"""

def message_received(dataPacket):
        print "Got␣a␣data␣packet"

        print dataPacket
        try :

                # Get the actual data payload we want to look at
                sensorPayload = dataPacket["rf_data"]

                if hackyMethodToIdentifyData(sensorPayload) == "pi":
                        processPiMessage(dataPacket)
```

```
                elif hackyMethodToIdentifyData(sensorPayload) == "fio":
                        processFioMessage(dataPacket)

        except Exception as e:
                # The last steps will fail for messages such as on
                    calibration, we need to catch this
                pass

def processPiMessage(dataPacket):

        sensorId = getSensorId(dataPacket)

        # Get the actual data payload we want to look at
        sensorPayload = dataPacket["rf_data"]

        # Already received at least one packet for this sensor, so we will
        # add this data to the existing value in the map
        if sensorId in PI_DATA_BUFFER:

                # If the stream is still smaller than the specified length
                if len(PI_DATA_BUFFER[sensorId]) < PI_STREAM_LENGTHS[
                    sensorId]:

                        # Append this payload to the end of the stream
                        PI_DATA_BUFFER[sensorId] += sensorPayLoad

                        # Check to see the new length. If it's >= the
                            specified length,
                        # we know we have reached the end of the image
                            stream
                        if len(PI_DATA_BUFFER[sensorId]) ==
                            PI_STREAM_LENGTHS[sensorId]:
                                parsePiDataAndSend(sensorId, PI_DATA_BUFFER
                                    [sensorId])
                                del PI_DATA_BUFFER[sensorId]

                        if len(PI_DATA_BUFFER[sensorId]) >
                            PI_STREAM_LENGTHS[sensorId]:
                                print("Error: Sensor " + sensorId + " sent
                                    the wrong number of image packets")
                                exit()

                else:
                        print("Error: Sensor " + sensorId + " sent the
                            wrong number of image packets")
                        exit()

        else:
                streamLength = int(sensorPayload[0:sensorPayload.find(" ")
```

```
                ])
            firstPacket = sensorPayload[sensorPayload.find("␣") + 1:]

            PI_DATA_BUFFER[sensorId] = firstPacket
            PI_STREAM_LENGTHS[sensorId] = streamLength


def processFioMessage(dataPacket):

        sensorId = getSensorId(dataPacket)

        # Get the actual data payload we want to look at
        sensorPayload = dataPacket["rf_data"]

        # Remove dangerous characters
        sensorPayload = sensorPayload.replace("\r", "").replace("\n", "")

        # Already received at least one packet for this sensor, so we will
        # add this data to the existing value in the map
        if sensorId in FIO_DATA_BUFFER:

                # If this payload contains a '[', it means that this
                    statement
                # is all debug info, so we ignore it
                if sensorPayload.find("[") != -1:
                        None

                # Check that this payload is not the last packet in the
                    stream
                elif sensorPayload.find(">") == -1:
                        FIO_DATA_BUFFER[sensorId] += sensorPayload

                # If it is the last packet in the stream
                else:
                        # Append the last payload
                        FIO_DATA_BUFFER[sensorId] += sensorPayload
                        # Send the entire stream to the server for storage
                        parseFioDataAndSend(sensorId, FIO_DATA_BUFFER[
                            sensorId])
                        # Delete the data that was just sent to save space
                        del FIO_DATA_BUFFER[sensorId]

        # This is the first packet from this sensor, so we create a new
        # entry in the map, with the sensorId as the key, payload as the
            value
        else:
                # If this payload contains a '[', it means that this
                    statement
                # is all debug info, so we ignore it
                if sensorPayload.find("[") != -1:
```

```
                        None
                else:
                        FIO_DATA_BUFFER[sensorId] = sensorPayload

def getSensorId(dataPacket):
        # Grab hex string representing sensor id, it's given as a raw
            binary number
        sensorIdHex = dataPacket["source_addr"]

        # Get the actual sensor id, and make sure that there is data to
            grab
        sensorId = ord(sensorIdHex[0]) * 256 + ord(sensorIdHex[1])

        return sensorId

def hackyMethodToIdentifyData(sensorPayload):

        print("Currently␣using␣hacky␣method.␣Please␣update␣as␣soon␣as␣
            possible.␣Let's␣not␣be␣lazy␣here.␣Seriously␣though.␣FIX␣IT!")

        isFio = True
        for char in sensorPayload.replace("<", ""):
                if not (ord(char) == ord('␣') or ord(char) == ord('.') or
                    ord(char) == ord('>') or (ord(char) in range(ord('0'),
                    ord('9') + 1))):
                        isFio = False
                        break

        if isFio:
                return "fio"
        else:
                return "pi"

# testDataStream = "<200 4.7 50 10.1 10.2 10.3 10.4 10.5 10.6 10.7 10.8
    10.9>"
# parseFioDataAndSend(7, testDataStream)

# testPiStream = "102400 as;ofijw9r8uapw9erjas jdfzsdjf jawli,,.
    fhxfglq8uw3498ysf#R$**9hdfsaehrksdf;sfawr68569#$("
# parsePiDataAndSend(3, testPiStream)

# Create API object, which spawns a new thread
xbee = XBee(ser, callback=message_received)

print "Listening␣for␣input␣on␣serial␣port␣" + PORT

# Do other stuff in the main thread
while True:
   try:
       time.sleep(.1)
```

```
    except KeyboardInterrupt:
        break

# halt() must be called before closing the serial
# port in order to ensure proper thread shutdown
xbee.halt()
ser.close()
```

Slim/PHP code to create RESTful API

```
<?php

        error_reporting(E_ALL);
        ini_set('display_errors', '1');

        require 'vendor/autoload.php';
        require 'admin/validate_credentials.php';

        require 'backend_php/fios.php';
        require 'backend_php/images.php';
        require 'backend_php/licenses.php';
        require 'backend_php/pis.php';
        require 'backend_php/sensors.php';
        require 'backend_php/users.php';

        # Create the server/app
        $app = new \Slim\Slim();

        date_default_timezone_set('America/Denver');

        # Define all the routes our application will accept below

        ################ FRONT END (WEBPAGE) ROUTES ###################
        ###############################################################

        # return 'home.php' when someone requests the root of our site
        $app->get('/', function() {
                readfile('home.html');
        });

        # return 'list.php' when someone requests /list
        $app->get('/list', function() {
                readfile('list.html');
        });

        # return 'stats.php' when someone requests /stats
        $app->get('/stats', function() {
                readfile('stats.html');
        });
```

```
# return 'home.php' when someone requests /map
$app->get('/map', function() {
        readfile('home.html');
});

######################### USER ROUTES ##########################
################################################################

$app->get('/users', function() {
        authorize();
        extract_all_users();
});

$app->get('/users/:id', function($id) {
        authorize();
        extract_user($id);
});

$app->post('/users', function() use ($app) {
        authorize();
        create_user($_POST);
        $app->redirect('/smartlots/admin/view_users.php');
});

$app->delete('/users/:id', function($id) use ($app) {
        authorize();
        delete_user($id);
});

#################### IMAGE ROUTES ##############################
################################################################

$app->get('/images', function() {
        authorize();
        extract_all_images();
});

# Return the image with the specified date/timestamp
$app->get('/images/:date', function($date) {
        // DON'T REQUIRE AUTHORIZATION! INTRADA NEEDS TO BE ABLE TO
             ACCESS IT W/O AUTHORIZATION
        header('Content-Type: image/jpg');
        imagejpeg(imagecreatefromjpeg("/var/license_plates/images/
            $date"));
});

// $app->post('/images', function() {
//      authorize();
//      create_image($_POST);
// });
```

```
// $app->delete('/images/:id', function($id) {
//      authorize();
//      delete_image($id);
// });

############### LICENSE PLATE NUMBER ROUTES ####################
################################################################

$app->get('/licenses', function() {
        authorize();
        extract_all_licenses();
});

$app->get('/licenses/:id', function($id) {
        authorize();
        extract_license($id);
});

// $app->post('/licenses', function() {
//      authorize();
//      create_license($_POST);
// });

// $app->delete('/licenses/:id', function($id) {
//      authorize();
//      delete_license($id);
// });

############### INTRADA PLATE NUMBER ROUTES ####################
################################################################

$app->get('/intrada_licenses', function() {
        authorize();
        extract_all_intrada_licenses();
});

$app->get('/intrada_licenses/:id', function($id) {
        authorize();
        extract_intrada_license($id);
});

// $app->post('/intrada_licenses', function() {
//      authorize();
//      create_intrada_license($_POST);
// });

// $app->delete('/intrada_licenses/:id', function($id) {
//      authorize();
//      delete_intrada_license($id);
```

```
        // });

        ###################### SENSOR DATA ROUTES ######################
        ################################################################

        $app->get('/sensors', function() {
                authorize();
                extract_all_sensors();
        });

        $app->get('/sensors/:id', function($id) {
                authorize();
                extract_sensor($id);
        });

        // $app->post('/sensors', function() {
        //      authorize();
        //      create_sensor($_POST);
        // });

        // $app->delete('/sensors/:id', function($id) {
        //      authorize();
        //      delete_sensor($id);
        // });

        ############ BACK END (PARKING SYSTEM) ROUTES ################
        ################################################################

        # Adds data from the parking lot fio sensors to the database
        $app->post('/fiodata', function() {
                authorize();
                process_fio_data($_POST);
        });

        # Adds data from the parking lot pi sensors to the database
        $app->post('/pidata', function() use ($app) {
                authorize();
                process_pi_data($_POST, $_FILES);
                $app->redirect('/smartlots/admin/view_licenses.php');
        });

        $app->run();

?>
```

ALPR Invocation Function

```
function get_license_number($image_filepath, $image_url) {
        $license_number = "FAIL";
        $intrada_license_number = "FAIL";
```

```
$state = NULL;

// Start the OpenALPR engine on the ACMx server.
// Sample result string (plate is LTM378):
// - LTM378 confident: 92.4582 template_match: 0
$unfiltered_result = exec("/var/license_plates/openalpr/src/alpr␣-r
    ␣/var/license_plates/openalpr/runtime_data␣-n␣1␣$image_filepath"
    );

if ($unfiltered_result != '') {
        $exploded_result = explode('␣', $unfiltered_result);
        $license_number = $exploded_result[5];
}

// Create a SOAP request for Intrada ALPR cloud service.
$intrada_client = new SoapClient('http://intrada2.cloudapp.net/
    IntradaService.asmx?WSDL',
        array('soap_version' => SOAP_1_2));
$intrada_request = $intrada_client->IntradaRecognizeDirectPassage(
        array(
                'project' => 'brodrigu.demo',
                'key' => '8053',
                'images' => array($image_url),
                'metadata' => array()
        )
);

// Sample result string (plate is LTM378):
// RESULT:{hash}:{plate},{confidence},{state},{confidence},{
    coordinates of license plate in image}:{execution time}
// RESULT:592E1EE8D1DE427BB6E0042F34745523:LTM378,750,USA_CO
    ,750,(309,156),(503,159),(309,206),(503,207):1594
$unfiltered_result = $intrada_request->
    IntradaRecognizeDirectPassageResult;
if (!is_soap_fault($unfiltered_result)) {
        $exploded_result = explode(':', $unfiltered_result);
        $unfiltered_result = $exploded_result[2];
        if ($unfiltered_result == '[reject]' || $unfiltered_result
            == 'Could␣not␣download␣file') {
                $unfiltered_result = 'REJECT';
        }
        $exploded_result = explode(',', $unfiltered_result);
        $intrada_license_number = $exploded_result[0];
        if ($unfiltered_result != 'REJECT') {
                $state = $exploded_result[2];
        }
}

// $license_number will be FAIL if OpenALPR didn't find a number,
    or the plate/partial.
```

```
        // $intrada_license_number will be FAIL if Intrada had an error,
        // [reject] if it could not find a number, or the plate/partial.
        $license_info = array(
                0 => $license_number,
                1 => $intrada_license_number,
                2 => $state
        );

        return $license_info;
}
```