Smart Lot

by

Landon Anderton, Alex Freshman, Kameron Sheffield, and Sunny Trinh

Contents

1 Abstract

This document contains the proposal for a 2010 computer engineering senior project at the University of Utah. The project being proposed is an automated parking lot assistant that directs traffic to the closest available parking spots. This project contains both an original hardware and software component. The automated parking lot will be demonstrated on the final day of classes Fall Semester 2010.

2 Introduction

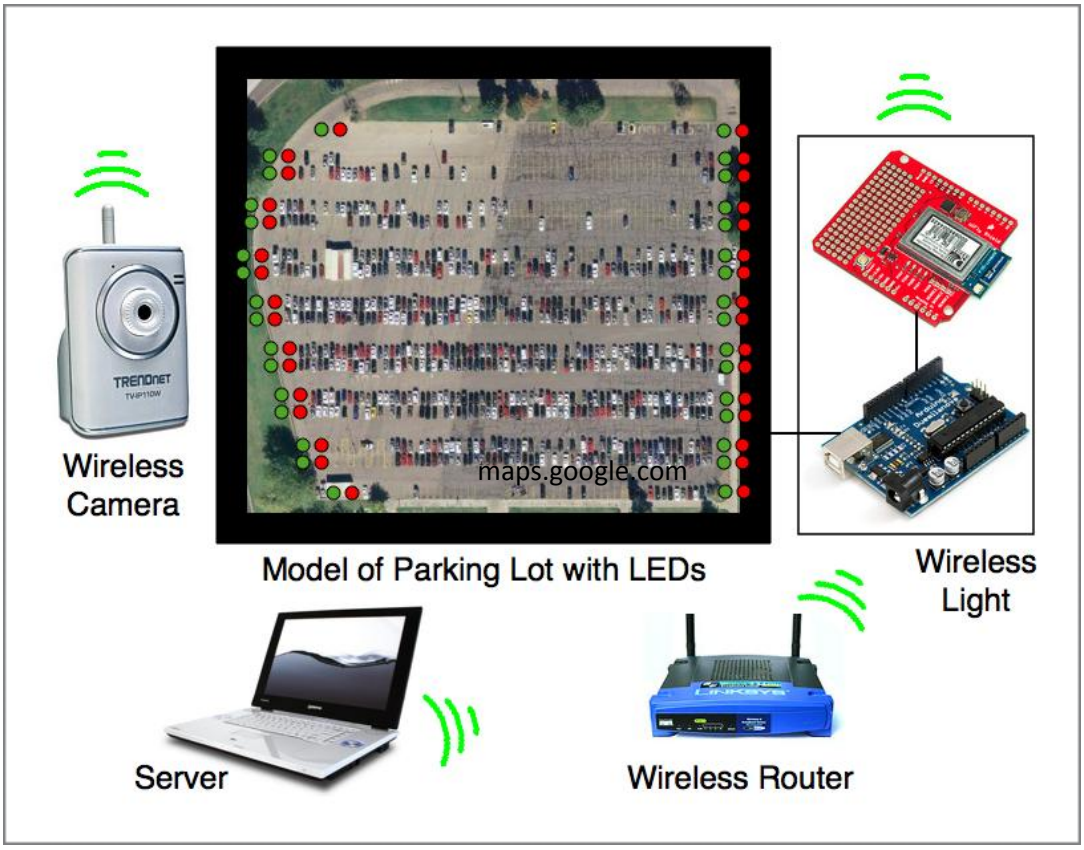
The idea stems from a common obsession to get the closest parking spot. The motivation for the project is to solve a problem many commuters face when aimlessly circling a parking lot. Our solution is to visually show vacant parking spots with traffic lights at the edges of parking rows. If a given row contains a vacant spot, the light will show green; likewise if a given row is entirely full, the light will show red. When entering a parking lot, the driver will no longer need to circle the lot to find the closest spot, he will simply need to look at the lights and identify the closest row with a green light.

The next obvious question is how to detect the vacant spaces from the occupied spaces? One easy solution would be to install some sort of sensor in every parking spot. Any number of sensors could be used, like induction sensors found at intersections or laser sensors. The problem with installing sensors in every parking space is that it would be impractical in most situations. The cost to install such a system in an already existing parking lot would be too great, not to mention the cost of each individual sensor. Our solution is to install only a handful of cameras, each with the ability to monitor the occupancy of many parking spaces. In a real world application the system will consist of several cameras mounted high enough to obtain a bird's eye view of a parking lot. In most applications, the cameras will be mounted to light poles that are commonly found in most parking lots. Each camera will send its images to a central server that will process the image and determine the occupancy of each space. The server will in turn command the traffic lights to change between green and red.

Our team proposes to demonstrate the functionality of this automated parking lot system in two ways. The first will be a live real-world demonstration. Using at least one wireless Internet camera we can monitor any parking lot, near or far away. The camera will need to be kept indoors and out of the elements, so we will locate a suitable parking lot adjacent to a tall building. Whatever spaces the camera is able to capture, are the ones we will monitor and demonstrate. This same camera will be placed early in development to allow our system to be tested against many different situations. This allows for our second method of demonstration, to replay a few interesting recordings, such as night, large crowds, and possibly snow. The traffic lights will not be installed for the demonstration, but rather shown on sight using small LED lights.

2.1 System Overview

Below is a diagram depicting the overall system. Subsequent sections describe each aspect of the system as a whole.



www.sparkfun.com

2.1.1 Wireless Camera

The system requirements for the camera are that it can wirelessly update the server with new images. In a real world application the camera would need to withstand the outdoor elements. Rather than choosing an outdoor surveillance camera, we chose a practical indoor camera that will allow for easy testing and development of our system. The selected camera is the TV-IP110W IP camera from Trendnet. This camera uses a CMOS image sensor to achieve a maximum resolution of 640 x 480 at 30 frames a second. This camera is ideal because it comes ready for wireless communication. The TV-IP110W runs an embedded version of Linux which is in turn running a web server on the device. The on board web server allows the end user to log onto the device by typing in its IP address in a web browser. Once logged in the user may set the camera to do whatever they choose via its web configuration page. Further, this camera provides many different ways to access its recorded images. The user can connect to the camera using its web interface, a direct socket connection, or it can be set up to upload images on a specified interval to a FTP server. Yet another attractive feature of this camera is the open source firmware. The firmware is written in C and compiled on GCC, which makes modification easy. The TV-IP110W will provide us the maximum flexibility we need to develop the system.

2.1.2 Server

The system is a client-server model, where the clients are the wireless camera and the wireless light. The camera sends still images to the server to be processed and analyzed for vacant parking spots. Data is sent both to and from the wireless light. The light receives data containing which lights to turn red or green, as well as, sends acknowledgement messages back to the server. This will be described in detail in later sections. The server for the demonstration will be a laptop running the server code written for this project.

2.1.3 Wireless Router

A wireless router will be used to route the appropriate data to the appropriate recipient. In a real world application it will need to be placed in a fashion to cover the range of the parking lot. For the demonstration, however, it will be in close proximity to the server and wireless light.

2.1.4 Model of Parking Lot with LEDs

A small scale model of the parking lot chosen for surveillance will be made for a clear demonstration. This will likely be a piece of cardboard outlining the parking spaces and rows with LED lights at the edges of each monitored row. This small scale model will be used to demonstrate the functionality of the wireless light.

2.1.5 Wireless Light

The wireless light is the original hardware component for this project and will be discussed in later sections. In a real world application the wireless light would control a small handful of the traffic lights placed at the edges of parking rows. This would allow for installing the system in an existing parking lot without laying data lines to each traffic light. But for the demonstration, the wireless light will control the switching of all the LEDs on the model parking lot.

3 Project Tasks

3.1 Original Hardware Component

The switching of the traffic lights will be controlled wirelessly over 802.11b/g using a microcontroller. Below are images of a wireless 802.11b/g card and an Arduino micro-controller.

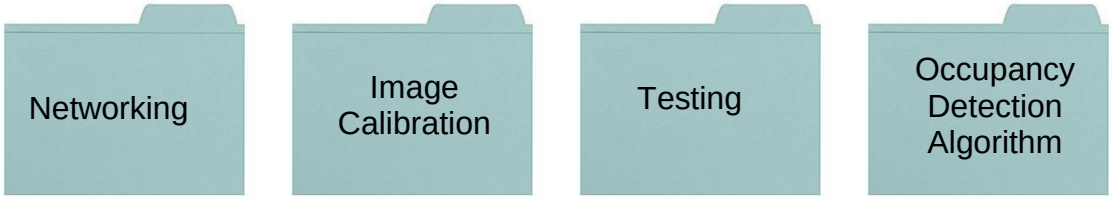
www.sparkfun.com

Data will be sent and received to and from the wireless light through the 802.11b/g WiFly Shield and processed by the Arduino micro-controller. The frequency at which data is received and sent will vary depending on how many vacant parking spots currently exist in the lot. When the lot is virtually empty, such as late night, the frequency can be slowed down to increments of 20 minutes, but when the lot is full or close to full, frequency can be sped up to increments of 5 seconds. This will allow the microcontroller to sleep during slow traffic times.

The wireless light will receive data packets from the server containing which parking lot regions are occupied, as well as, when to expect the next packet. The Wireless Light will send acknowledgement messages back to sever to verify the data was received. The incoming data packets will be processed and the Arduino will switch the traffic lights appropriately. During down time, when the Wireless Light is not expecting data, it will sleep. This will save on power consumption. When the time comes to receive another packet, it will wake up process the data and then go back to sleep.

3.2 Original Software Component

The original software component for this project is the server software as well as the occupancy detection algorithm. The server software and the occupancy detection algorithm will be done in C# using .Net framework. The software will be broken up into four code sections depicted below. Details of each code section follow in subsequent headings.



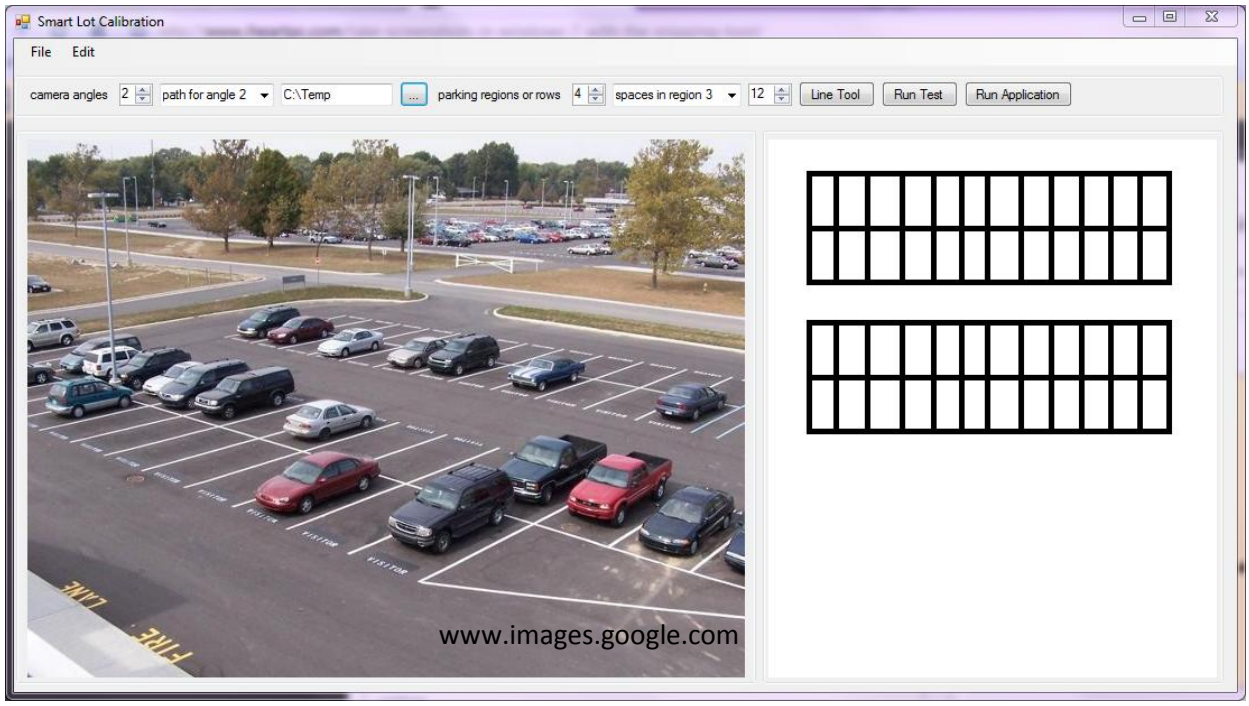
3.2.1 Networking

The networking code section will have the following work load:

- Determining the frequency at which to read in new images and send packets to the wireless light.
- Reading in new images at the given frequency and passing them along to the test GUI.
- Building and sending the packets to the wireless light.
- Receiving acknowledgment packets from the wireless light.

3.2.2 Image Calibration

The Image Calibration for test images and varying camera angles will be done with a GUI. The GUI will allow a user to draw lines on a parking lot image to denote parking spot regions. After the user is done calibrating an image the GUI will write a text file containing the coordinate data for the camera angle. An example of the calibration GIU is shown below.

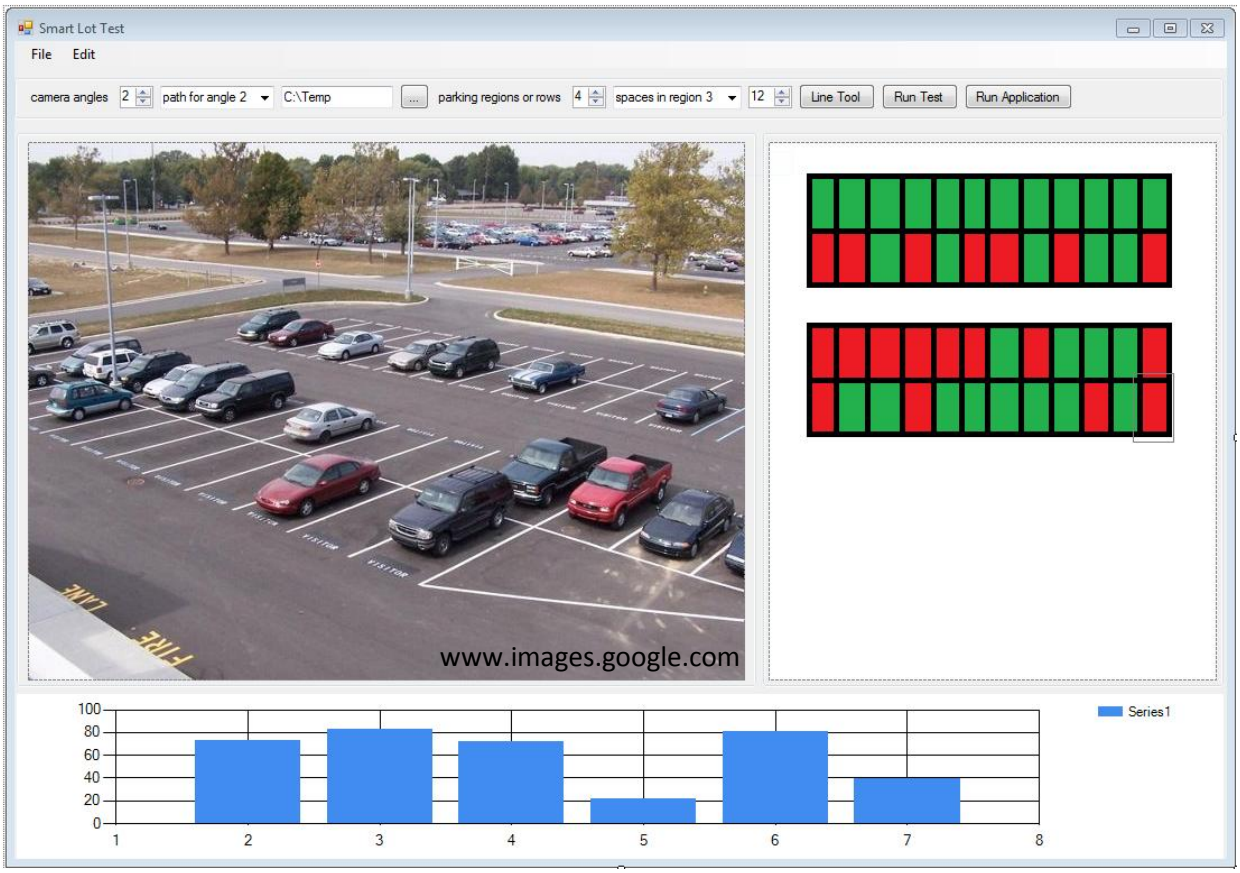


The calibration GUI will allow for the following functionality:

- An option for opening/starting a new parking lot.
- A field for entering how many different camera angles exist in the lot.
- A save as option.
- A field for entering the number of regions in a given camera angle.
- A field for entering the number of spots in each region in each camera angle.
- A field for entering the path to where the different camera angle images are being saved.
- A line drawing tool that has different colors for each region in a camera angle.
- An option for running a test on the newly created or re-opened camera angle.

3.2.3 Testing

The Testing of the occupancy detection algorithm will also be done with a GUI. The GUI will visually show the results of the detection algorithm. It will show red squares for parking spots found to be full and green squares for parking spots found to be empty. The GUI will also allow the user to scroll between different cameras and see the results. An example of the test GUI is shown below.



The test GUI will allow for the following functionality:

- Shows the image with red or green squares on each parking spot.
- By clicking on the square the user can see the histogram plot.

3.2.4 Occupancy Detection Algorithm

We plan to develop the occupancy detection algorithm in C# using the AForge.Net framework. The AForge.Net framework offers a large variety image processing tools. Our plan is to experiment with these tools until we find something reliable. Our hope is that we can detect empty spots using histograms. There exist two kinds of histograms that might prove useful when detecting vehicles in an image.

The first is a histogram that shows tonal differences in an image. It is a graph of the number of pixels for each tonal value. This could prove to work given that the surfaces of cars reflect light. They have dark tonal values, like shadows and tires, as well as very bright tonal values where the car reflects the light. Using the method of tonal differences, the algorithm will perform a tonal histogram on each individual parking spot. If there exists a large number of tonal differences versus a very flat area, then the algorithm will report occupied. Something to be determined about tonal histograms is whether or not they will function at night, when there won't be much reflection from the cars.

Another alternative to tonal histograms is color histograms. A color histogram is a graph of the number of pixels for each color value. This could also prove to work given that most vehicles have a large variety of colors from end to end, like bumpers, hoods, windshield, objects inside the vehicle, tires and headlights. Again the algorithm will perform a color histogram on each parking space, and if there exists a large variety of colors versus an area that is mostly grey, the algorithm will report occupied. With this algorithm it will also need to be determined whether or not it will function at night, as well as, whether or not it will function on grey colored cars.

Yet another algorithm provided by the AForge.Net framework, is an edge detection filter. Performing this algorithm on an entire row of cars, rather than individual parking spots, will show where each car begins and ends. Then empty spots can be detected by measuring the distance between cars. This algorithm may run into problems with parking rows containing a lot of shadows, and areas where the ground is dirty or snowy.

All of the algorithms have pros and cons. It will need to be determined which one, or even a combination of several, will work best for our system. Experimenting with different algorithms, different conditions, and different camera angles will be the bulk of this project.

4 Interface Specification

4.1 Hardware Interfaces

4.1.1 Wireless Light to Server

The Wireless Light will communicate to the server using the WiFly module which will provide 802.11g to the Arduino. Using the WiFly shield is easy, first the user must set up the device to communicate with the Arduino via an onboard SC16IS750 SPI-UART bridge by setting the register values on the WiFly to their appropriate settings. Next the WiFly will automatically start looking for networks to join. At this point the user can specify a network as well as a password for the WiFly to join, again by setting some internal settings. By now the device should have appeared as a node on the local network. The last step in the process will be to set the WiFly shield to listen for incoming connections. The server will then initiate a direct socket connection with the WiFly module by specifying its corresponding IP address and then sending it a packet regarding what light should be on and how long it should wait before reactivating itself to receive another update packet from the server. The packet will consist of a short followed by an integer. Each bit in the short will represent a particular light; a value of 0 will be used for red and 1 for green. The integer will specify an amount of time to wait in seconds, before the light should wake back up and wait for updates.

4.1.2 WiFly Shield to Arduinio

The WiFly Shield connects directly to the Arduino. Two 6 pin headers and two 8 pin headers need to be soldered to the WiFly Shield before connecting it to the Arduino.

4.1.3 Arduino to Parking Lot Model LEDs

The Arduino will process the packets delivered to it from the WiFly and turn on the necessary LEDs. Two LEDs will be tied to the same signal, since they are mirrors of each other on both sides of the parking lot (See system overview diagram). The LEDs will be powered externally to simulate higher voltage lights that would be necessary in a real world environment.

4.1.4 Wireless Camera to Server

First, because there will be no direct access to the camera a FTP server will have to be setup on a public web server so that the camera can upload its new data there. The second step will be to set the camera up to upload its data to the FTP on a specified interval, or when motion is detected. Finally the server software that computes occupancy detection will have to log into the FTP server and download the latest data for processing. The camera will be set up to upload an image with a file name that will identify the camera as well as time and date the image was taken. The upload interval may need to be increased or decreased depending on the number of remaining empty sites in the parking lot. Meanwhile the server software will be periodically monitoring the image folder for new images, when it discovers that a new image has arrived it will flag the new image as an image that needs to be processed and start the process. The server will also periodically clean up its cache of images from the cameras. When the folder size is larger than a specified threshold the server will delete as many files as necessary until the folder is half of the previously mentioned threshold.

4.2 Software Interfaces

4.2.1 Calibration GUI to Occupancy Detection

The calibration GUI will create a text file containing the coordinates of the individual parking spots. The occupancy detection algorithm will read the calibration text file and use the coordinates as endpoints to perform a histogram. An example text file is shown below. The file begins with a unique identifier, which is used to distinguish between lots. It is followed by a description of the lot, how many regions are in the lot and how many spaces are in each region. Finally the lines used for calibration of each spot, where each location is delimited by a curly brace, and each line is delimited by parenthesis. Each line is specified as endpoints using ordered pairs x:y separated by hyphens.

```
calibration.txt

lotId: 0x0842;

numRegoins: x; // the number of regions in the lot e.g. rows

numSpaces_x: y; // the number of spaces per region

{(65:105-78:137), (78:137-122:139), (122:139-145:102)}
```

{(65:105-78:137), (78:137-122:139), (122:139-145:102)}

{(65:105-78:137), (78:137-122:139), (122:139-145:102)}

4.2.2 Occupancy Detection to Test GUI

The open spaces will be output to the test GUI via a map data structure that will use region identifiers for the keys and a list of open spaces as the value. Then the test gui will simply iterate through the map and update its output appropriately.

5 Testing and Integration Strategy

Testing of the occupancy detection algorithm can begin once the testing GUI is completed. To start off, this will be done using a number of selected test images, a few of which are shown below. The benefit of this strategy is that the occupancy detection algorithm can be tested in parallel with the development of the rest of the software architecture. Once the rest of the application is complete the two units can be combined rather easily and the algorithm can begin work on live images from the camera. This will provide the maximum amount of time to work on the occupancy detection algorithm. The goal is to have a very robust system that works in most conditions snow, rain, during the day, and at night.



www.images.google.com

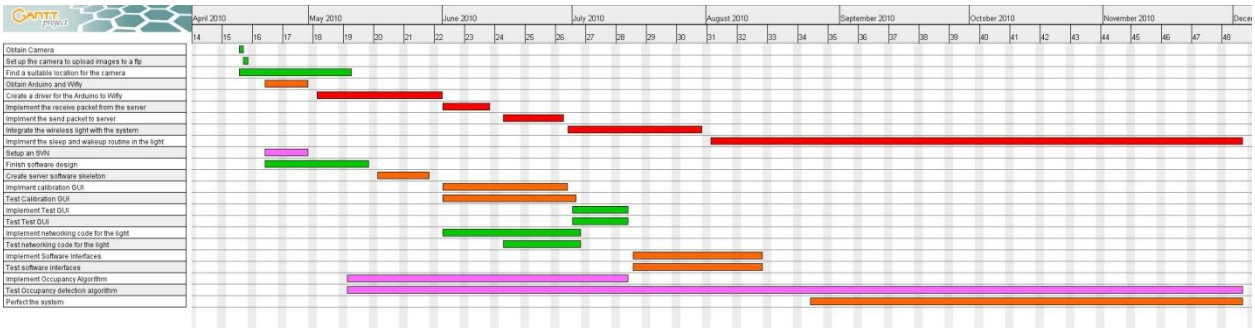
6 Group Communication Plan

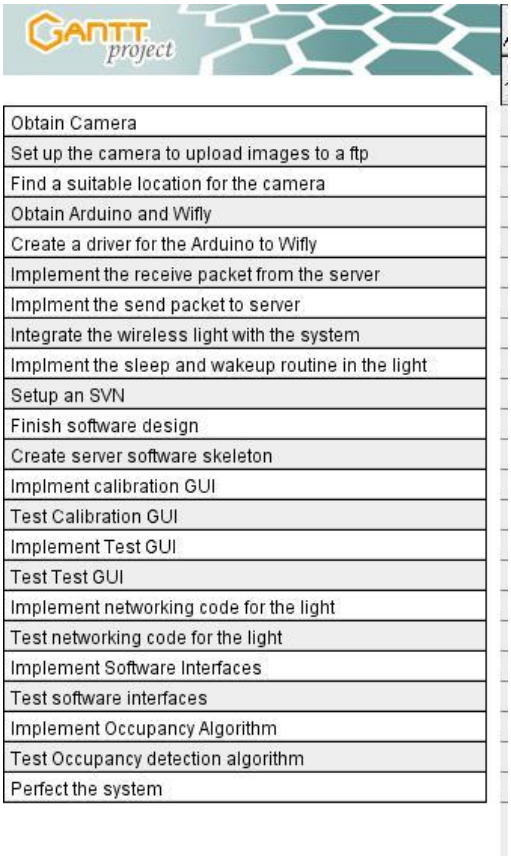
The communication plan is subject to change as the project progresses, but for the time being our plan is as follows. Our team shall meet in person once a week in the conference room where class is held on Wednesdays one hour before class begins. Outside of our weekly meetings, we have a Google Group set up, so we can keep in touch and share ideas.

7 Schedule and Milestones

The schedule for the project is shown below, as well as, an enlarged image of the tasks. Each team member is assigned a task to oversee, which is depicted by the various colors used for the tasks. Although each team member has a specific task, it is planned that every team member will help out on each piece of the project to a certain degree. Alex will head up the hardware aspect of the project. Kameron will be in charge of getting the camera working in a remote location, as well as the design of the overall software architecture, and implementing all network

functionality. Landon will oversee the development of the overall software skeleton as well as implementation of the calibration GUI. Sunny will take the lead on creating the occupancy detection algorithm.

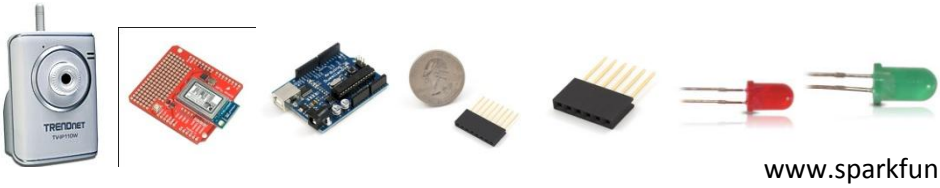




8 Risk Assessment

The only current risk is the functionality of the occupancy detection algorithm. Getting to work properly at different angles, different times of day, and different weather conditions will likely be the bulk of the work for this project.

9 Bill of Materials



www.sparkfun.com

Product	Vendor	Quantity	Total Price
TRENDnet - Wireless Internet Camera	newegg.com	1	$70
WiFly Shield 802.11b/g	sparkfun.com	1	$90
Arduino Main Board	sparkfun.com	1	$30
Arduino Stackable Header - 8 pin	sparkfun.com	2	$1
Arduino Stackable Header - 6 pin	sparkfun.com	2	$1
LEDs - Red	RadioShack	32	~$5
LEDs - Green	RadioShack	32	~$5
Total			$202

10 Conclusion

Smart Lot is an excellent senior project. The project proposes both an original hardware and software component, which allows us to show off our computer engineering knowledge and skills. Not only does it provide a challenging learning experience for those involved, but we believe it will improve the quality of everyday life for students commuting to school. Smart Lot is both something we can be proud of as a team and a legacy we can leave behind.

11 References

Support page for the camera:
http://www.trendnet.com/downloads/list_subcategory.asp?TYPE_ID=32&SUBTYPE_ID=1172

Information about the WiFly shield:
http://www.sparkfun.com/commerce/product_info.php?products_id=9367

Information about the Arduino main board:
http://www.sparkfun.com/commerce/product_info.php?products_id=666

WiFly Shield tutorial: http://www.sparkfun.com/commerce/tutorial_info.php?tutorials_id=158

Information about the Aforge.Net framework: http://aforgenet.com/