UPSON HALL

ITHACA CAMPUS INITIATIVES

Advitya Khanna, Adam Gleisner,
Justin Cray, Kristian Langholm, Faadhil Moheed

Date: 05/19/2015

INTELLIGENT OCCUPANCY SENSING FOR THE UPSON HALL HVAC SYSTEM

1. EXECUTIVE SUMMARY

2. INTRODUCTION

a. System Overview

i. Intel Galileo

ii. Prototype System

3. SubSystems

a. OCCUPANCY SENSING

i. XBee RF Communication

1. Summary
2. Semester Overview
3. How it works- XBee Wifi?
4. Challenges- XBee Wifi
5. Current Plan- XBee Transceivers
6. How it works- XBee Mesh Network
7. Benefits
8. Challenges & Next Steps

ii. PIR Motion Sensor

1. Hardware
2. Software
3. Moving Forward

iii. Outlook Schedule Integration

1. System
2. Privacy Concerns

iv. K-Nearest Neighbors Algorithm

1. Algorithm
2. Implementation

v. Indoor Occupancy Sensing

b. USER INTERFACE

i. Timeline of Events

ii. User Interface System Flow

iii. Challenges Encountered

iv. Accomplishments & Future Development

4. RECOMMENDATIONS

5. CONCLUSION

6. GLOSSARY

7. APPENDIX

EXECUTIVE SUMMARY

The Upson Hall team focused mainly on improving upon and implementing last semester's prototype design for a smart HVAC system for single-occupant professor office spaces within Upson Hall. In that prototype, we were able to predict with relatively good accuracy when a professor would be in their office ahead of time using a combination of RF transceivers, motion sensors, calendar interpretation, and machine learning such that we could heat their office to a desired temperature before their arrival. This system allows for a building to achieve significant efficiency improvements; often in the colder months of the year, we see professors leaving their heating systems on all the time, resulting in a large waste of energy and money.

By including new and improved sensors for enhanced occupancy sensing along with an intuitive user interface, we attempted to improve upon this existing design. As this project is still in its early phases, we found it valuable to revisit our initial project goal this semester and reiterate what exactly makes a good heating system. It was especially important to do so because our team was almost entirely comprised of new members; only one member of the team returned from last semester.

INTRODUCTION

Last semester's research focused on setting the baseline for this project, incorporating XBee Wifi transmitters to track professors on campus, passive infrared (PIR) motion detectors to detect whether a professor is within their office, Microsoft Outlook calendar interpretation, an intuitive email interface and K- Nearest Neighbors machine learning. This baseline system communicated with a cloud server, which processed the sensor data, performed machine learning, and ultimately performed the communication with the HVAC system.

Our main task was implementing the baseline system and improving upon each subsystem of that project, while adding additional functionality to establish communication between the system and the professor. We spent a lot of our time getting each new subsystem functional as independent units. In doing so, we've set up the system to integrate all of these technologies, while leaving possibility for future improvements by keeping in mind the scalability of the design decisions made. A brief introduction of the entire HVAC system follows:

System Overview

1. Intel Galileo

The central hub in this smart heating system is the Intel Galileo, an Arduino-compatible microcontroller that uses the x86 instruction-set architecture (Figure 1). The board contains a full Linux distribution (when a >8GB microSD card is plugged in) and is able to run Arduino sketches to make use of the Arduino I/O pin headers. The ability to access Linux in addition to the ease of programming of Arduino sketches gives flexibility that is vital to the success of this system. The Galileo is allowed us to run a simple Python Script that has been vital for the implementation of the User Interface. We were able to use common Arduino-compatible hardware to get the project started with ease, and for the most part were able to simply modify example sketches to get the functionality we desired from these hardware subcomponents.

Figure 1: The Intel Galileo Microcontroller.

2. Prototype System

The prototype system contains the Intel Galileo as a hub, connected to Internet via its built-in Ethernet port. In addition, there is a passive infrared (PIR) sensor connected to one of the digital inputs, which allows for the Galileo to detect motion in the professor's office space. It communicates with an RF transceiver subsystem containing XBee unit, where it receives both packets and address information for the unit. In addition, the Galileo receives email requests for system overrides from professors over the Internet, in the case that a professor decided that they want to bypass all prediction and simply turn on the system before their arrival at the office. It processes all of these sensors and data and communicates with a server in the cloud, which combines this information with Microsoft Outlook Calendar information for the professor as well as a K- Nearest Neighbors machine-learning algorithm. Based on all of this information, the server relays a message back to the Intel Galileo with an on/off signal corresponding to whether the heating system should be turned on or off.

OCCUPANCY SENSING

In order to implement an intelligent HVAC system in Upson Hall, we need to be able to determine when a professor will be in his or her office. Specifically, the system needs to be able to turn itself on about 12 minutes before a professor arrives in order to ensure that that professor is comfortable when they get to their office (based on data obtained with the Heating system in Upson). In order to do so, our system has four main components: XBees for location tracking, a PIR sensor for occupancy sensing, Outlook schedule integration, and a predictive algorithm for interpreting data. Each of these components work together to ensure that the professor has the maximum level of comfort.

XBee RF Communication Device

XBee Devices - Summary

The XBee is a coin-sized module used to transmit data wirelessly. It is manufactured by Digi International. We received an XBee pair, XBee shield, and XBee Explorer made by Sparkfun for an Arduino board. The XBee was chosen because it was advertised to have a mile range, is low powered, and does not require much, if any, configuration.

The idea is to use the XBee devices for two main functions - to transmit and to receive. In terms of the receiver, we will have a coordinator XBee attached to the Galileo in Upson Hall. The Galileo will continually send signals to the other devices in the network in order to determine if there are any professors in range. If the coordinator receivers a signal back acknowledging that there is a professor on campus, it will gather the required data (i.e location, identification number), process the information accordingly, and update the HVAC system in Upson Hall. In terms of the transmitters, the devices that make up the network and the XBee's on the professors themselves would transmit the required information to the central HUB in Upon to be processed.

Initially, we acquired the XBee Pro 802.15.4. These were advertised as having a mile line of sight range and 100m range indoor/urban. The problem that we ran into was that the XBee would be housed inside Upson, so the indoor range applied. This is a problem because the

range is not long enough so that we would be able to determine when the professor is in a mile range of Upson.

A possible solution to this problem was to get the XBee Pro 900 XSC S3B. This particular XBee has a 28 mile line of sight range and indoor/urban range of up to 610m. However, upon testing the device we were not able to get the full range. The range that we got was less than a quarter of a mile. This may have been because the XBee was underpowered or because it was ill advertised.

Semester Overview

This semester, we spent a majority of our time figuring out which course of action would be best in order to achieve a way to transmit information from an XBee device, to the central HUB in Upson Hall. Initially, we tried using Cornell's existing wifi infrastructure as our foundation for creating a network, however after much deliberation and research, we decided to pursue implementing our own network of XBee devices around campus in order to transmit the information we needed.

How it works - Wifi Network

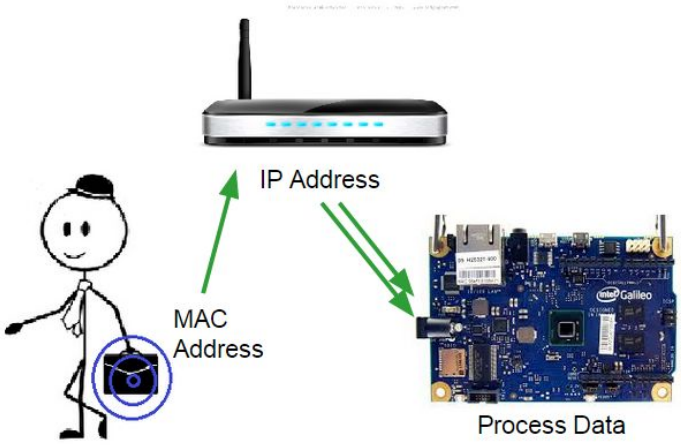

Figure 2: XBee Wifi System Flow

The diagram above describes the process in a very simply way. Each professor would be carrying a XBee device. This device has a unique tracking number, its Mac Address, that will

allow us to identify which professor is holding which device. This tracking number is then sent through the nearest router on campus based on the professor's location, and these two pieces of information - Mac Address, and router IP Address - are both forwarded through the Wifi network the the central HUB in Upson Hall that will process this information and change the HVAC system accordingly.

Implementing a network in this way would provide us with a few benefits:

1) We would not need to physically create a network - our single XBee device placed on each professor would connect to the existing wifi network, and the information would be sent through wifi to Upson Hall.
2) Cost would be much lower - we would only need to pay for the XBee devices we give to professors.
3) HVAC control could be easily interfaced with other buildings on campus - the wifi network will remain constant and available everywhere on campus, thus it would be very easy to implement the same system for other buildings.

However, after speaking with Cornell IT and researching the topic, we realized that we would face many challenges in terms of implementing this method.

Challenges

1) Security - security as a whole in terms of our project would be very hard to ensure and maintain. For our network to work in this way, we would need the IP addresses of the routers on campus in order to determine where each professor is located in relation to Upson Hall, and to receive this information, we would need to be cleared by Cornell's IT Department. For example, however, if our system were to be compromised, people would have somewhat unrestricted access to the rest of Cornell's network, which would be very bad..
2) Privacy - in order to use the wifi network, we would be using an XBee wifi specific unit to implement and determine the location of professors on campus. Since there are numerous routers around campus, we would essentially be able to determine the

location of a professor very accurately - which may concern a lot of people as they would not necessarily want to be tracked around campus.

3) Non - routable IP addresses - Based on Cornell's existing network, our XBee devices were given non-routable Ip addresses, meaning that we would not be able to send any information through the network as our devices were deemed non-routable to begin with. To resolve this we would need to talk with Cornell IT in order to give our devices routable addresses so we would be able to use them, which is somewhat of a hassle on their end.
4) Dynamic vs Static IP Addresses - Assuming our XBee device could be routed through the network, Cornell's router/wifi system is very dynamic. This means that certain IP addresses are allocated for a large portion of devices around campus, and once a device loses their connection with the network, these Ip addresses are then procured by new devices that are connecting to the network. In terms of our needs, once our XBee wifi unit on the professor connects with the network, sends it's information, and disconnects, the IP address sent to the router would have changed to another device or be unconnected. In effect, we would not be able to tell which XBee device sent what information as the addresses are always changing.

Current Plan

Based on the numerous challenges we faced, we decided to implement our own network of XBee devices around campus that will allow us to send data from the XBee unit on the professor to the main controller in Upson. After researching the topic, we realized that this is actually how XBee devices should be used, and that it is very easy to configure these devices through Digi International's Software XCTU, in order to make a very robust and dynamic network.

How it works - XBee Mesh Network

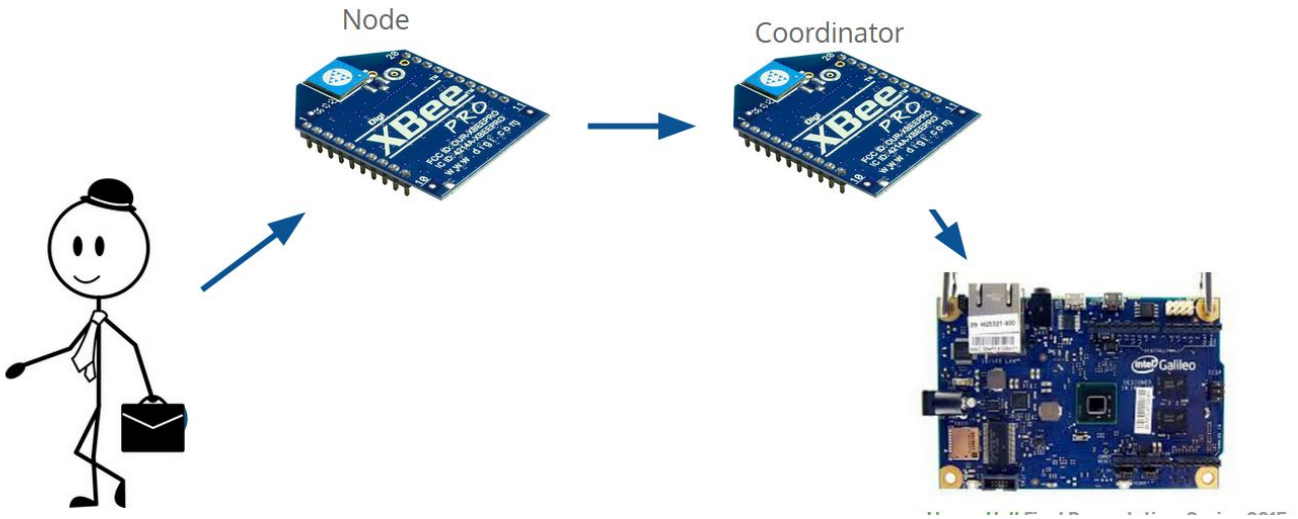

Figure 3: XBee Mesh Network System Flow

Instead of the data from the XBee device on the proffessor be sent through the wifi network the Upson Hall, the information is instead transmitted through a series of nodes on our network to the central coordinator in Upson, which will process that information.

Here is the mesh network in terms of a map:

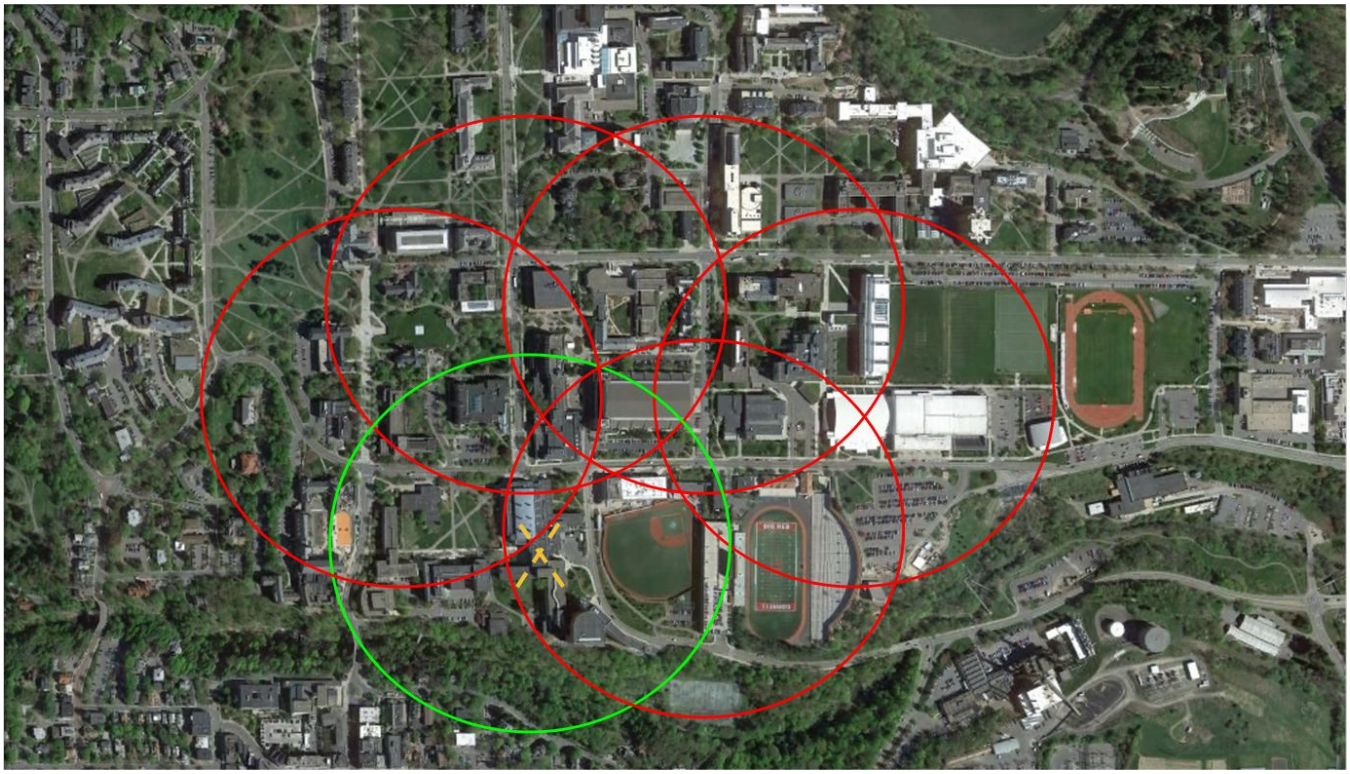

Figure 4: Map of Cornell Inlaid with Placement of XBee Nodes

This is a map of Cornell, with Upson Hall marked with a yellow X. Each red circle represents a node on our network - in this case a XBee device, and the green circle represents the central XBee attached to an Intel Galileo in Upson Hall. Once a professor walks in range of any of the nodes, that XBee device would transmit information to the respective node, which would send that information through the network of red nodes until it reaches the central controller, where the information would be processed. We decided to use a mesh network for two main reasons:

1) Each node knows where the central HUB is located, and thus will transmit information through the nodes until the central controller is reached.
2) Every node is dynamic. No node is simply just passing information to the central HUB, each node is actively looking for - in this case a professor - to enter its sphere of influence and then act accordingly. This is exactly what we want as the any professor could be in any location throughout campus, and thus does not necessarily appear through the same node on consecutive days.

Benefits

1) Surpass Cornell IT restrictions - since we are not going through the existing wifi infrastructure, we do not need to deal with any challenges we would have otherwise faced if we decided to use the wifi network (look above for explanation of challenges).
2) Increased security - since our network is not attached to anything - i.e Cornell's wifi infrastructure - we do not need to worry as much about security since our network is not accessible from outside sources - people do not know the specific XBee devices we are using, or even how they relate to professors or the campus itself.
3) Increased privacy - by using our network, professors would experience a drastic increase in their privacy in terms of locations around campus. Rather than have the XBee wifi unit transmit, with a somewhat great precision, the location of a professor, through the mesh network we would only be aware of a certain range based on which node the professor is in. Think of it like a Venn diagram, if the professor is within the sphere of influence of two nodes, we would receive a transmission from both nodes

consecutively, and thus, we would know that the professor is located somewhere in the area that these two nodes consecutively encompass.

Challenges and Next Steps

1) Cost - the cost would be drastically increased as we now not only need to purchase XBee's for each professor, but not we need to purchase around 6 - 10 more in order to create the network itself.
2) Need to contact facilities - to create the mesh network, we would need to place the XBee nodes in buildings throughout campus, and make sure each has an adequate power source. In order to achieve this, we would need to contact facilities and receive the required permits to implement our network, as we are not allowed to simply walk into a building and place devices at will.

All the information we gather and transmit is a somewhat automated process, however we also have a predictive algorithm already in place that will allow our system to learn based on the changing schedules of professors, and adapt accordingly. This algorithm will be showcased and described later in the report.

Some of the next steps for the project are that we need to set up the mesh network across the Cornell campus as depicted in the Cornell map above. This will probably require the cooperation of Cornell Facilities and we need to make the Xbee Transmitter Key chain carried by the professor more portable and less clunky as it is at the moment.

PIR Sensor

In order to record motion data in an office space, we used a combination of the Intel Galileo microcontroller, a passive infrared (PIR) sensor manufactured by RadioShack, and our offsite server. The PIR sensor works by detecting changes in infrared light within a 29.5 foot radius spanning 120 degrees. If any change is detected, the sensor outputs a high (1) signal; for every other time, the sensor's output stays low (0). This signal is fed to the Galileo, which then transmits the occupancy true/false data to the offsite server for processing.

1. Hardware

The PIR sensor has three pins; a Vcc pin (5 Volts), a Ground pin, and a signal pin that outputs high or low, as shown in Figure 2. All of these pins are connected to corresponding pins on the Intel Galileo microcontroller via wires and a breadboard, creating a compact and self-contained system. For the purposes of testing, we have placed the Galileo and breadboard containing the PIR sensor on a professor's desk near the location where they would keep the computer; this allows us to detect subtle motion, such as typing or navigating via trackpad or mouse, with high precision.

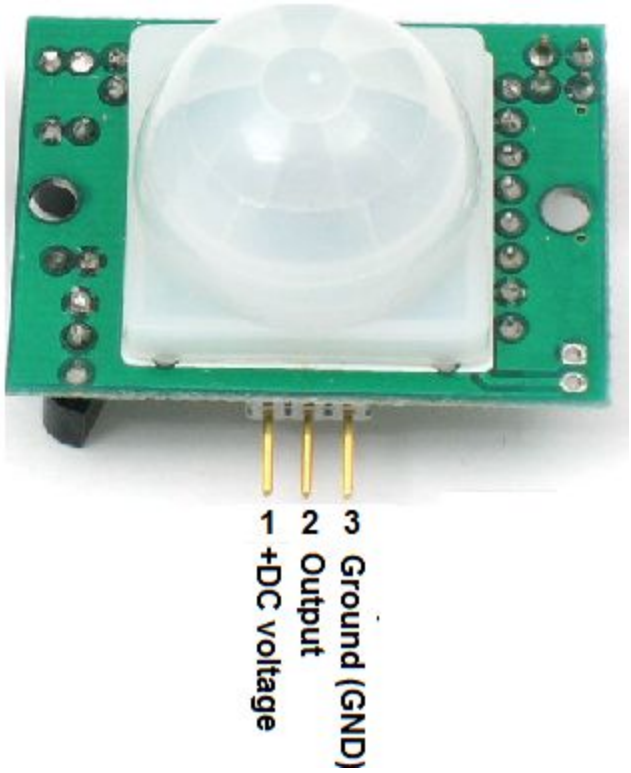

Figure 5: An example of PIR sensor

2. Software

In order to process the PIR sensor's output, we have written an Arduino sketch that runs perpetually on an Intel Galileo. This sketch, which is shown in Appendix I, begins by starting

the Galileo's Ethernet drivers and establishing an Internet connection, which is used for communication with the offsite server. Next, the sketch calibrates the PIR sensor for 30 seconds, a recommendation by the manufacturer for proper operation of the sensor. Finally, the sensor waits for data to come in from the PIR sensor. Motion sensor data comes in continuously, which requires that the Galileo does some processing before communicating with the server. It does this via the following:

If the incoming data is a high (1) signal, visualize this data by turning on a green LED on the Galileo, Communicate a motion-true signal to the server. Do not resume server communication until a low signal is received (this gets rid of repeated data).

If the incoming data is a low (0) signal, wait 5 seconds and check to see if, during those 5 seconds, a high (1) signal comes in. If not, we know that motion has truly ended and we can interpret the signal as a true false. Visualize this data by turning off the green LED on the Galileo. Communicate a motion-false signal to the server. Do not resume server communication until a high signal is received (this gets rid of repeated data).

3. Moving Forward

Currently, each Galileo only communicates with one PIR sensor, as the test system is built for only one single-occupant office. For further expansion and scaling of the project, we would need to communicate with multiple PIR sensors for different office spaces and have one central Galileo differentiate between the data and send data points to the server with corresponding room information. The Intel Galileo contains 12 digital input pins, which theoretically means that it can communicate with 12 separate PIR sensors at once. However, extensive testing would be necessary to ensure that the Galileo processes all of the data without any loss.

Another possible use of multiple PIR sensors would be for rooms larger than single-professor office spaces, such as classrooms or lecture halls. By simply adding an Intel Galileo and a number of PIR sensors corresponding to the size of the room, motion data can be recorded

very easily. Given the relatively low cost of PIR sensors (\$10 per sensor), implementing motion sensing in any room is quite inexpensive.

Outlook Schedule Integration

After taking in the current receiver/transmitter data and the motion sensing data, all you know is whether or not the professor could make it to his/her office before it finishes heating up and whether or not he is currently in his office. However, just knowing the professor is within a mile of his/her office does not tell you whether or not he/she will soon be in his/her office; it only tells you that he/she is near his/her office. The professor could be teaching a class nearby or out to lunch. Also, it is important to know whether the professor would be expected to arrive in his/her office since by the time the professor is in the room the office should have already spent 15 minutes heating up.

1. System

In order to do this, it would require you to know that professor's schedule, which is stored in an Outlook calendar. This would mean that the professor would need to share the full details of his/her schedule with a member of the group creating the predictive heating system. The instructions on how a professor could share an Outlook calendar with that group member are shown in Appendix II. After this schedule is shared, a .ics file is created that is continuously polled by a python script running on the server. The reason full details need to be shared is because the group would need to know the locations of the event the professor has planned. If the professor has an event scheduled in his/her office, the office should start heating up 15 minutes before the professor plans to be there. If the professor has an event in a room other than his/her office, the system knows that it should not expect the professor to enter his/her office, so there is no further prediction needed.

2. Privacy Concerns

We recognize that professors would not want students being able to know the full details of their schedules. However, with our system, that is not the case. Once the schedule is created as a .ics file, the location and time data is read entirely behind the scenes. The students will

have no other access to it, only the server. If a professor decides that they still don't want their schedule to be shared with our system, they can opt out of the Outlook schedule integration. Between the other components of our system, we should still have a good general idea of when the professor will be in his/her office. The main benefit of having a schedule is being able to know about exceptions to the professor's normal, which our predictive algorithm might not handle quite as well.

K-Nearest Neighbors Algorithm

However, professors are in their offices for more than just specific calendar events. The professor could be working in his/her office (i.e. grading papers, researching). This means that knowing if their calendars say they will be in their offices is not enough. This means the system will have to have a predictive portion to it. In other words, you would need to know if, based on previous occupancy data, the professor is expected to be in his/her office 15 minutes before a given time. This would require a specific predictive algorithm. K-nearest neighbors proved to be effective on the sample data below, which shows the error of a data set with a strong weekly trend.

1. Algorithm

K-nearest neighbors is an algorithm that will try to figure out information of a specific point based on knowing information about the "neighbor" of that point. A neighbor is a point that is considered similar to the point you are trying to figure out information about. For occupancy prediction, the points represent a specific date/time and the information you are trying to obtain through prediction is whether the professor is expected to be in his/her office during that time. The neighbors in this case are dates/times considered similar to the one in question. Since professors teach on a weekly schedule, the neighbors would be the same day of the week at approximately the same time at the weeks before it. Also, most professors will spend at least 30 minutes in their offices before leaving it, and normally they spend a good hour or more in it. This means the neighbors would also be the times around that one in the weeks before, looking more at the ones after it since it needs to predict to start heating up 15

minutes before the professor is expected in his/her office. The time range that proved to be effective is 15 minutes before the current time to an hour after the current time.

2. Implementation

The way K-nearest neighbors was specifically implemented involved calculating weighted sums of the values of the neighbors and then seeing if that number was large enough to mean the professor is expected to be in his/her office. Remember that a 1 represents the professor being in his/her office and 0 represents being out of it. This means the weighted sum will be a sum of the neighbors (1 or 0) divided by a weight. The weight represents how far away the neighbor is from the current time, so the value is divided by how many weeks away from the current time the neighbor is and how far away time-wise it is (in range of 15 minutes before to an hour after). This algorithm was implemented on a Raspberry Pi microcontroller by the team in Spring 2014. For the specific code used for simulation see Appendix III. For data experiencing a pretty good weekly trend, the error was only 0.2% (2.88 minutes a day) after 2-3 weeks of predicting, as can be seen in Figure 3.

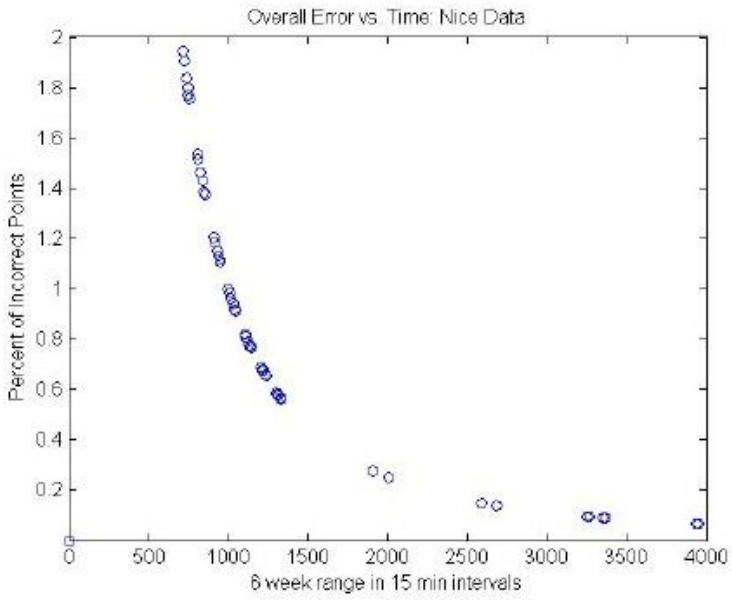

Figure 6: Overall Error vs. Time data for the K-Nearest Neighbors algorithm.

3. Moving Forward

As shown in Figure 5, after only a few weeks of gathering data, the algorithm already had excellent predictive capabilities. However, the system the team in 2014 used to implement this algorithm was different from the system we will be using. In order to get a better estimate of the error, we will need to run this algorithm on our fully implemented system, which is a goal for next semester.

Indoor Occupancy Sensing

Continuing from last semester, we had a plan to more accurately determine the location of professors within Upson Hall using a barometric sensor for the Xbee devices that the professors would carry on them during the course of the day. We abandoned this subsection of our project for two main reasons. First, we discovered that the Xbee devices we ordered were not compatible with third party sensors such as the barometric sensor. Finding a fix for this was not worth the time, as we discovered that the barometers would give an highly variable and inaccurate reading on which floor the professor was on, which was the second problem. We opted to drop this section of the project in favor for more time focused on outdoor occupancy sensing as well as a better UI.

USER INTERFACE

With the predictive and scheduling algorithms already in place, one of the main objectives of our project for the course of the semester was building a reliable and scalable user interface to integrate with the existing predictive algorithms. The interface would serve as a platform for a simple, yet powerful two-way communication network between a professor and the Galileo; that is, the professor could provide the Galileo instructions that would be able to override the predictive algorithms and in return, the Galileo would process the requests and transmit data delineating the professor's usage on a bi-monthly basis. In this way, the communication network established via the user interface gives the professor the option to have more or less control over the HVAC system designed. Machine Learning and Predictive algorithms may be able to predict normal behavior 99 percent of the time; however, these algorithms do not have the power to predict deviations from normal behavior—they could not predict whether a professor will be sick on a given day and not show up to his office (deeming the switching on of the HVAC useless) or perhaps the professor will be arriving early to his office to finish pending work. In any case, the interface serves as a platform for the professor to prompt the HVAC system to perform an instruction as desired and work in conjunction with the predictive algorithms to allow the professor to be both comfortable with the temperature in his office space and reduce the amount of electricity/gasoline consumption to heat the office spaces in Upson Hall.

As it can be imagined, there are multiple user interfaces that we could have employed, especially with the advent of smartphone applications and web interfaces. However, we determined that a hybrid of an email/text-message communication network between the professor and the Galileo provides the most scalable and reliable option for the scope of our project. This is primarily due to the fact that not all professors have access to smartphones or are comfortable with using web- based applications. Hence, it makes sense to utilize the Cornell email system that is already set up for the professors in order to allow them to communicate with our HVAC system, to allow for them to have more control over their heating supply than was permitted with just the predictive and scheduling algorithms. This is not to say that a smartphone application is a bad idea for the project; in fact, we are in the

process of building a smart phone application as future addition to our system. However,the first step was to set up a scalable and reliable simple communication network, which the email/text-message communication network provides us with.

Having provided an overview of some of the objectives of our project and the design decisions made, the rest of this section will focus on providing a detailed description of the exact mechanics of how the email/text communication network was set up. It will include a brief timeline of the progress that includes the progress that we made on our project over the semester, followed by a detailed explanation of the setup of our email/text communication between the professor and the HVAC system using the "Linux Bigger Image" in the Intel Galileos. Additionally, this portion will delineate some of the challenges that we faced in bringing the communication network together, and finally, some oversight on possible future developments that would make our HVAC system even more robust and scalable.

1. Timeline of Events

With the majority of our group unfamiliar with the Intel Galileo and the Arduino IDE and programming in C, the first portion of the semester was dedicated to getting familiar with the IDE and programming the Galileo. Hence, for about fifteen days, we focused on programming basic sketches onto the Galileo and compiling a report to document some of the tricks & tips that we learnt in programming the Galileo, including links to various open source development project and libraries that may be useful in the context of our project. As a side note, we found a large chunk of our code to set up the user interface on GitHub and were able to utilize it into our project.

Over the next month, the focus was on software development (and research) in order to improve our system by attempting to establish a user interface between the professor and the HVAC system. At this point in the semester, we determined that email communication would serve as the most viable option in the context of our project. Hence, we focused on developing and finding a simple way to incorporate the desired interface. At this point in our project, we had some difficulties in connecting the Galileo to the Internet via its Ethernet port

as well as the computer itself. A detailed description of this challenge will be described in the proceeding sections.

During the start of the semester, we were able to successfully compile and program a script that received and displayed the contents of inbox emails in Python. However, we were not able to run and receive the output of this Python script on the Galileo Board. At this point, our focus turned to the interaction of the the Galileo C code and the Python script.

Over the course of the next few weeks we came up with a solution to the Python/C code incompatibility issue through the use of the "Bigger Linux Image", a way to run a rudimentary Linux PC on the Galileo board. This allowed us to code a way for the C code to run the Python script, store the output in a text file, and parse through it for instructions.

2. User Interface System Flow

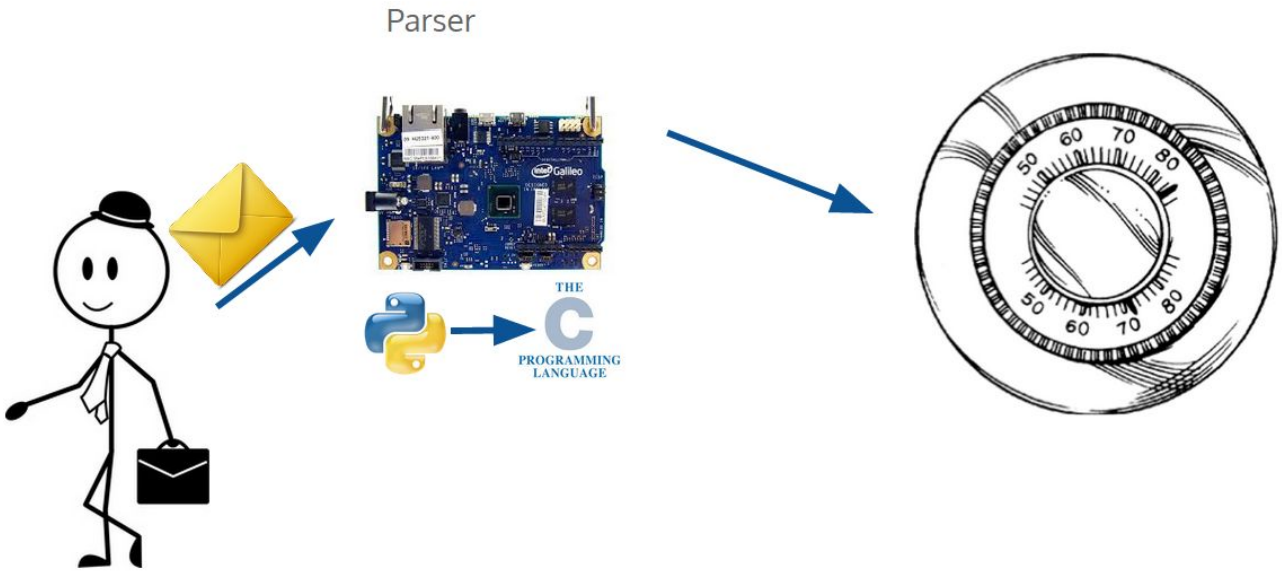

Figure 7: UI System Flow

As the diagram and description above depicts, the Galileo is able to access the a Gmail Inbox feed through the use of POP, which allows emails to be read locally through a third party application, such as a Python script. The way that the Galileo performs this function is by utilizing the Intel Galileo's ability to access a Linux machine, while utilizing the ease of programming of Arudino sketches. Essentially, the Galileo calls a "system()" command that takes in a Linux command as its input and is able to run Linux right out of the box. In the context of our project, we make a system call that prompts the Galileo to run a Linux command that retrieves the output of the Python code, store it as a text file, and parse through it. As a result, the Galileo's ability to run Linux commands in the manner described above makes the Galileo indispensable to our project (that is, we could not use a simple, less powerful Arduino microcontroller instead).

Having described the Galileo's ability to detect the inflow of requests provided by the professor to setup the user- interface, it is important to note that this interface is perhaps the most reliable and compatible user- interface given the resources available to the professors. That is, this user- interface only assumes that the professor has an email account (which in this context is a safe assumption as every student/staff of Cornell University has a Cmail address). As described earlier, it cannot be assumed that all professors have purchased smartphones,

because our HVAC system is designed to be scalable to larger audiences and we do not want to limit the scope of project by restricting it to professors with only smart phones. Another key selling point for this Email user- interface is that all of the major cell phone networks in the USA allow users to send and receive text messages as email. Hence, the professors do not really even have to open their email accounts, in order to send in a request as they could just simply text the dummy address CUSDemail@gmail.com, which would in turn receive the text message as an email and would be able to parse through the contents of the message, as necessary.

We offer an Android Application to the Emailing/ Texting UI that we have in place. The Android Application offers a more intuitive user experience to any professor that has a smart-phone. In fact, the Email UI serves as the backbone for the mobile application, as it utilizes the professor's desired command (on/off) on the off will be translated into an email and be sent over to the dummy address CUSDemail@gmail.com. This means that the Galileo parses the contents of the message in the same manner, as it had with the Simple Email UI. In the next semester, we hope to add an iOS mobile application, as well. Here is a depiction of the system flow of the Android Application:

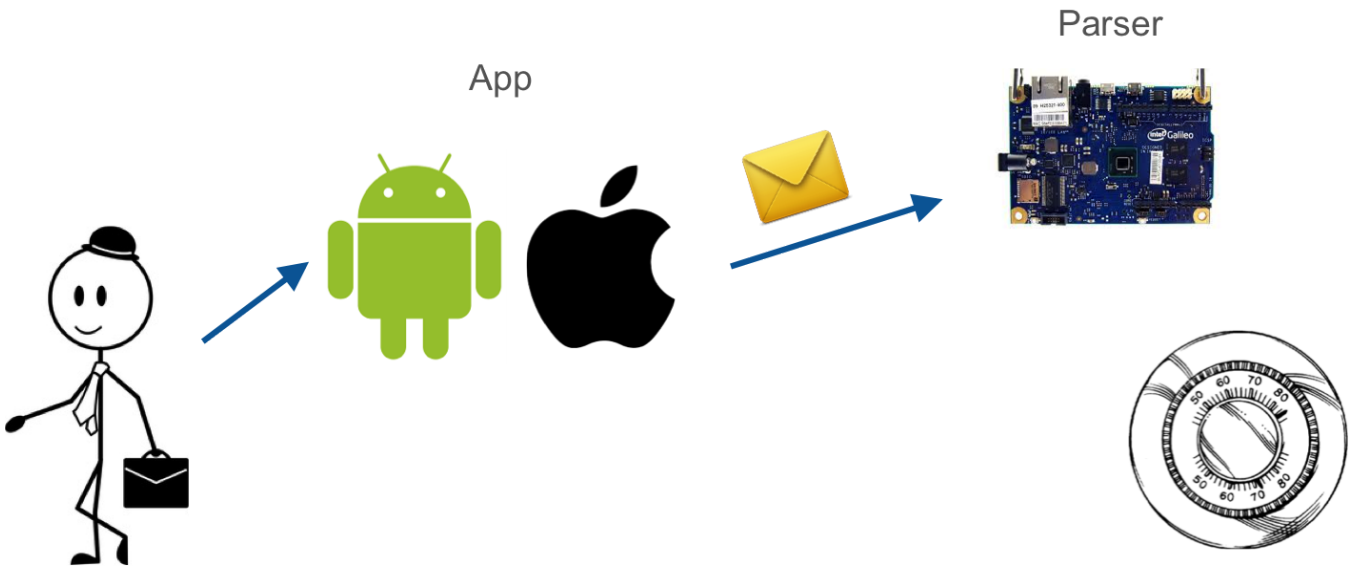

Figure 8: Mobile App System Flow

As a final remark, we would like to point out that the professors may believe that their rights to privacy is being encroached by the designed predictive algorithms and on- campus tracking. However, by offering the email user- interface and the mobile application, we can allow these professors to still utilize our HVAC system without tracking them through their connections to Wi-Fi routers on campus. That is, even if the professor leaves the office without turning off the heat (and we are not tracking them), he can choose to later remember to switch it off sending in an email/ text to the dummy email account. Although, this would mean that some of the responsibility is on the professor to manually prompt the Galileo to switch off the system. However, this does not mean that they could not participate/ utilize some of the services that our HVAC system employs.

3. Challenges Encountered

As described in the timeline section, we discovered that the Galileo was having not being able to connect to the Internet and read from the SD Card that hosted a Python Script for email parsing. The SD card image could not be seen on the Galileo. Hence, we had troubles with running it from the Galileo. This seemed to fail on all of the three Galileo's in our possession, which was particularly strange as this was working for the previous semester for the previous semesters. Initially, we focused on narrowing down on what could be causing the error: hardware issues, firmware update and Cornell's Internet connection compatibility with the Galileo. As a result, we made posts on the Intel Communities forum for the Galileo and contacted Intel's helpline to resolve the issue. However, none of this was able to resolve the issue. Fortunately, about a month back, we were able to find a post on the Intel Communities forum that required the Galileo to be programmed with the SD card in a certain order. The guide can be found here: https://learn.sparkfun.com/tutorials/galileo-getting-started-guide.

4. Accomplishments and Future Developments

As mentioned earlier, one of the focuses for this semester has been to design a concept to allow for a robust means of communication between the professor and the HVAC system. With the concept in place, we focused on implementing our design ideas and decisions that we had made earlier in the semester. Currently, we are able to access the dummy email addresses inbox via a Python Script that is run on the Intel Galileo

With that said, we are able to parse through specific messages with the code that we have to detect for key-words (as described in the previous section). This means that most components of the UI have been put in place. However, we do need to add an integrate our system with the iOS mobile platform as the mobile app development has only been for android devices. The plan is to complete this early next semester so that we can focus our attention on actually integrating our HVAC system with the Thermostats and the proposed HVAC in Upson Hall.

As an alternative to the Email UI, we are still considering Twilio, which is a Cloud Communication Service that handles that allow software programmatically make and receive phone calls and send and receive text messages. The service comes along with a nominal cost of $ 0.0075 per text message received. We could use this service to ease the burden on the Galileo, which will constantly have to poll for new emails that it receives in the dummy account. With this service, an external application could be interrupted to indicate the reception of a new message and that information could be relayed onto the Galileo to adequately perform the desired instruction. We are still a bit unsure of some of the details of the service and as a result, we plan to look for more details about the service and make a decision early next semester to complete the UI component of the HVAC system.

RECOMMENDATIONS

In moving forward with this system concept, we propose a few overarching recommendations aside from those mentioned in each subsystem description. First, we believe that the system can be run without the use of an external server component; that is to say, the Intel Galileo microcontrollers are more than capable of processing the data. If this were to be pursued, it would make sense to use a single Galileo unit as a central processing hub, with auxiliary Galileo units in each office collecting data for the sensors corresponding to each professor. In doing so, we recommend scaling the system to two or three offices to ensure that the Galileo is truly powerful enough to perform all of the data manipulation necessary to keep the system running. If it proves to be too much for the Galileo to handle, we recommend setting up a different low-powered computer as a central hub for the system, perhaps a Raspberry Pi or similar. This is to ensure that the system's overall costs are kept down.

We additionally recommend that the keychain units be soldered together; this will greatly reduce their size and make them seem more like a self-contained wearable device. Because the wearables industry is so popular in this day and age, it makes sense to embrace this design style. If possible, it would be nice to have a display of some sort on the keychain unit; this will allow the system to keep track of energy saving metrics in a simple self-contained manner. If this proves to be infeasible, we have developed an iPhone/Android app be developed that allows a professor to communicate with his/her office and receive energy saving metrics. Having an integrated web platform that keeps a more detailed record of the professor's electricity consumption and savings will be a great way to promote our project, as well as provide the professors with some useful information.

It is, also, recommended that the team next semester should set up a meeting with Joel Bender from Cornell's Building Automation and Controls Systems Integration Team to receive his input on our design and components of our entire system. Up until now, I feel that we have not aligned our project with the needs of Upson Hall as we were attempting to produce working solution. Now, I believe that we are close to have a working HVAC system; hence, it will be important to meet with the Cornell Facilities to be able to tailor our HVAC system with the Renovation Plan for Upson Hall. Additionally, a meeting with him may be useful as he had mentioned that he could help us secure Floor Maps for various buildings for our convenience.

This, in particular, would help us in the strategic placement of XBee nodes of our mesh network.

Please note that all guides and documentation can be found on the google drive that has been shared with all of the members of the group. Also, the code can be found on github @ https://github.com/Advitya/Upson-Hall-Team/tree/master.

CONCLUSION

Overall, this semester proved very valuable for the success of this project moving forward. The extensive research and development done this semester has certainly laid the groundwork down for this project to go in a very clear direction in the future. Given that most of the team will be staying on to continue work next semester and are freshman, the future is bright for this project, as long as integrating the subsystems once again does not prove to be too much of a hurdle. In the course of one semester, we were able to obtain and implement a new XBee RF transceiver system, as well as write code to integrate email parsing into this system that helped us develop both an Android Application as well as a simple Email User Interface. Of course, these components haven't been combined to fit in with the Thermostat hardware; that will likely be the work for next semester. Additionally, we hope that we will be able to coordinate with Joel Bender from Building Automation and Controls Systems Integration to get his input on our plans and help obtain Blueprints of campus buildings so as to strategically place nodes in our XBee mesh network.

WORKS CITED (if applicable)

Figure 1,2,3,4,5,6 – Intel Galileo photo, taken from Intel website and other images obtained from Google Images

APPENDIX I - Arduino Code for PIR Sensor

```
#include <SPI.h>
#include <Ethernet.h>

// the media access control (ethernet hardware) address for the Galileo:
byte mac[] = { 0x98, 0x4F, 0xEE, 0x00, 0x47, 0x91 };
//static IP address for the Galileo:
byte ip[] = { 192, 168, 1, 49 };

// Web address for Cloud Server
char server[] = "mainloadbalancer-400162624.us-west-2.elb.amazonaws.com";

// Initialize the Ethernet client library
// with the IP address and port of the server
// that you want to connect to (port 80 is default for HTTP):
EthernetClient client;
String s;

////////////////////////////
//needed for PIR sensor//////
////////////////////////////
//VARS
//the time we give the sensor to calibrate (10-60 secs according to the datasheet)
int calibrationTime = 30;
```

```
//the time when the sensor outputs a low impulse
long unsigned int lowIn;

//the amount of milliseconds the sensor has to be low
//before we assume all motion has stopped
long unsigned int mypause = 5000;

boolean lockLow = true;
boolean takeLowTime;

int pirPin = 2;    //the digital pin connected to the PIR sensor's output
int ledPin = 13;   //the digital pin connected to the Galileo's built-in LED

void setup() {
   Serial.begin(115200);  //115200 baud rate for Serial output
   ////////////////
   // PIR Setup Code
   pinMode(pirPin, INPUT);
   pinMode(ledPin, OUTPUT);
   digitalWrite(pirPin, LOW);
   //give the sensor some time to calibrate
   Serial.print("calibrating sensor ");
   for(int i = 0; i < calibrationTime; i++){
     Serial.print(".");
     delay(1000);
   }
```

```
    Serial.println(" done");
    Serial.println("SENSOR ACTIVE");
    delay(50);

    /// End of PIR Setup Code
    ////////////////

    // Start Ethernet Connection
    Serial.println("Attempting to start Ethernet");
    if (Ethernet.begin(mac) == 0) {
        Serial.println("Failed to configure Ethernet using DHCP");
        Serial.println("Attempting to configure Ethernet using Static IP");
        Ethernet.begin(mac, ip);
    }
    Serial.print("Your IP address: ");
    Serial.println(Ethernet.localIP());
}

void loop() {
  // Motion is detected
  if(digitalRead(pirPin) == HIGH){
    digitalWrite(ledPin, HIGH);   //the led visualizes the sensors output pin state
    if(lockLow){
      //makes sure we wait for a transition to LOW before any further output is made:
      lockLow = false;
```

```
      Serial.println("---");
      Serial.println("motion detected ");
      httpRequestMotion();
    }
    delay(50);
    takeLowTime = true;
    }

  // Motion has ended
  if(digitalRead(pirPin) == LOW){
    digitalWrite(ledPin, LOW);  //the led visualizes the sensors output pin state

    if(takeLowTime){
      lowIn = millis();     //save the time of the transition from high to low
      takeLowTime = false;  //make sure this is only done at the start of a low phase
    }

    if(!lockLow && millis() - lowIn > mypause){
      lockLow = true;
      Serial.println("motion ended ");      //output
      httpRequestNoMotion();
      delay(50);
    }
  }
}
```

```
// Method for connecting to server and sending data corresponding to motion detected
void httpRequestMotion() {
  // if there's a successful connection:
  if (client.connect(server, 80)) {
    // send the HTTP GET request:
    client.println("GET /insertPIR?isinroom=true&roomid=2 HTTP/1.0");
    //client.println("Host: www.arduino.cc");
    //client.println("User-Agent: arduino-ethernet");
    client.println("Connection: close");
    client.println();
    s = "";
    if (client.available()) {
      while (client.read() != -1) {
        char c = client.read();
        s += c;
      }
      if (s.substring(s.length()-9,s.length()-1) == "ucs nisr") {
        Serial.println("success in insert!");
      }
      client.stop();
    }
    else {
      Serial.println("Client not available :(");
    }
  }
  else {
```

```
    // if you couldn't make a connection:
    Serial.println("connection failed");
    Serial.println("disconnecting.");
    client.stop();
  }
}

// Method for connecting to server and sending data corresponding to motion ending
void httpRequestNoMotion() {
  // if there's a successful connection:
  if (client.connect(server, 80)) {
    // send the HTTP GET request:
    client.println("GET /insertPIR?isinroom=false&roomid=2 HTTP/1.0");
    //client.println("Host: www.arduino.cc");
    //client.println("User-Agent: arduino-ethernet");
    client.println("Connection: close");
    client.println();
    s = "";
    if (client.available()) {
      while (client.read() != -1) {
        char c = client.read();
        s += c;
      }
      if (s.substring(s.length()-9,s.length()-1) == "ucs nisr") {
        Serial.println("success in insert!");
      }
```

```
      client.stop();
    }
    else {
      Serial.println("Client not available :(");
    }
  }
  else {
    // if you couldn't make a connection:
    Serial.println("connection failed");
    Serial.println("disconnecting.");
    client.stop();
  }
}
```

APPENDIX II - Instructions for Sharing Outlook Calendar

1. Go to the site: http://www.it.cornell.edu/services/owa15/
2. Press the link to go to the web version of Cornell's outlook accounts
3. Login to your account
4. Click calendar in the top right corner (go to your calendar)
5. Click share in the top right corner
6. Type in the Gmail you want to share it with where it says share
7. Select full details for access (it will appear to the right of the email address in a drop down menu)
8. Press send

This will send an email with a dynamic link to a .ics file that is used to extract data from the Outlook Calendar

APPENDIX III - K-Nearest Neighbors Code

```
function [ predicted ] = predict( measured )
 [mr, mc] = size(measured);
 predicted = zeros(mr, 1);
for n = 1:1:mr/6
    predicted(1:mr/6, 1) = measured(1:mr/6, 1);
end

 sum = 0;

 vt = 1.6;
 %one day => 97 (each 15min)
 %one week => 679
for a = 2:1:6
    for b = 1:1:7
        for c = 1:1:97
            ind = (a-1)*679 + (b-1)*97 + c + 1;
            if a == 2
                for p = -1:1:4
                    sum = sum + measured(ind-679+p)/(p+2);
                end
            else
                for t = 1:1:a-1
                    for p = -1:1:4
                        sum = sum + measured(ind-679*t+p)/abs(t^2*(p+2));
                    end
                end
            end
            if sum < vt
                predicted(ind, 1) = 0;
            else
                predicted(ind, 1) = 1;
            end
            sum = 0;
        end
    end
end

end
```