Household Environment Monitoring

Evan Orenstein
orensteine@wustl.edu

Scott Melenbrink
smelenbrink@wustl.edu

Chris McKiernan
christophermckiernan@wustl.edu

ABSTRACT

With smart homes becoming increasingly popular, it is important to ensure that the sensors that form the nervous system of the home are as accurate and sophisticated as the applications that they enable. To that end, we have designed, implemented, and tested a full sensor network in a home for the purposes of uncovering and solving the many problems associated with creating a high-quality sensor network.

1. INTRODUCTION

The goal of our project was to create a high-quality dataset that we could analyze as well as make available for use and viewing by other parties. This was accomplished by creating an indoor/outdoor wireless sensor network that was continuously monitoring several pieces of environmental data. The data collected by that network was sent via a Raspberry Pi TelosB basestation to a web server where various averages were calculated and the data was made available for use in other applications.

2. SENSORS

The sensor network was made up of TelosB motes and included five indoor sensors and one outdoor sensor. Section 2.1 will go into detail regarding the design and implementation of the software used to control the TelosBs. Section 2.2 will discuss the creation of a novel outdoor sensor case designed for a TelosB mote.

2.1 Sensor Software

Before a TinyOS program could be written and loaded onto the motes, a suitable environment was needed. The Raspberry Pi that was destined to be the basestation provided such an environment, but installing TinyOS was less simple than might be thought. The instructions on the TinyOS wiki were not up to date and frequently left out important programs, compilers, etc. that needed to be installed to support the TinyOS installation. Additionally, the information regarding the necessary environment variables was no long applicable and resulted in TinyOS compiler warnings when set. With persistence, though, and help from people who have had similar issues in the past, these problems were overcome and we were set with a TinyOS enabled Raspberry Pi.[10] [11]

With TinyOS installed, we were able to begin development of the sender and receiver TinyOS applications. The first application was a mote application designed to send data from the board to the TelosB plugged into the RPi. The design called for sending a custom packet every five minutes comprised of the ambient temperature, relative humidity, light level, and board temperature. The second was the basestation app running on the TelosB that acted as a receiver for the Java application running on the RPi itself. The basestation app interprets the packets from the motes using built-in TinyOS utilities and then sends the data by serial to the RPi using another TinyOS tool.

The first step in implementation was confirming that the radio on the mote was being accessed correctly, which was accomplished by loaded a preliminary version of the software onto a mote and then running the TinyOS java application net.tinyos.tools.Listen on the basestation mote. In this way, we confirmed that we were receiving hex data from the motes at the appropriate intervals.

Next, we needed to get valid data from that hex. This was a two step process, the first being able to decode the hex packets and get integers from the motes, the second being interpreting those integers as actual sensor values. Interpreting the hex as a packet was as simple as finding the TinyOS header that corresponds to each sensor on the TelosB board and integrating it into the application. In order to interpret the values from the TelosB board, the data sheet of each sensor had to be consulted and the appropriate algorithms found. Originally, we implemented the algorithms on the basestation TinyOS app, but soon learned that the TinyOS utility to send data over the serial bus only accepted 16 bit integers. Obviously, the data would need to be unpacked by the java application on the RPi. This ended up being better design anyway as we could then tune the algorithms without having to reinstall the application on the basestation every time.

2.2 Outdoor Sensors Case

A unique problem we faced with sensor deployment is protection against the elements when installed outdoors. However, simply enclosing the sensor in an airtight case would not suffice as closing in the sensor would inevitably cause issues with the accuracy of resultant data collection. We needed to ensure sensor data collected from the TelosB would be accurate while still protecting the mote.

2.2.1 Outdoor Sensor Case Design

To solve this problem we decided a 3D printed case was our best path forward due to its flexibility in design and ability to quickly produce a complex product. The idea behind our case was to allow airflow through the bottom of the case through channels running along the sides of the case which would enter into an internal chamber where the sensor resides. This will allow our temperature and humidity data to be accurate while protecting the sensor from weather such as rain. However, we also have a light sensor which must be exposed to external light to produce accurate readings. To solve this we added a window directly over the light sensor, using a clear piece of plastic sealed with super glue to allow light into the case and still protect the electronics. Our first iteration of the case was designed as depicted in Figure 1.

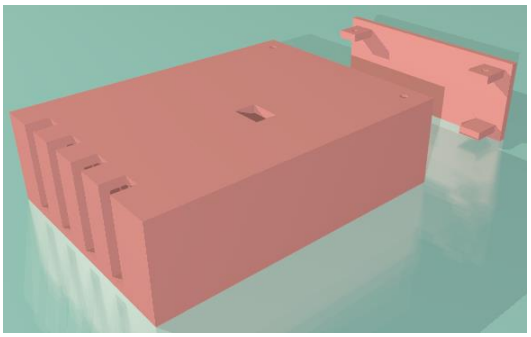

Figure 1. Original Sensor Case 3D Model

Originally we designed a 2 piece model which allowed the sensor to be lowered into the case from the top. However, our first attempt to print the model using a resin based 3D printer resulted in a case that had a sagging interior and structural inconsistencies. Upon further review of the model and collaboration with 3D printing open source forums, we found that our design was not ideal for 3D printing. To resolve this issue we reengineered our sensor case by splitting the model into a lid and base allowing us to still seal the case while creating a more printer friendly design. Our final design is depicted in Figure 2.

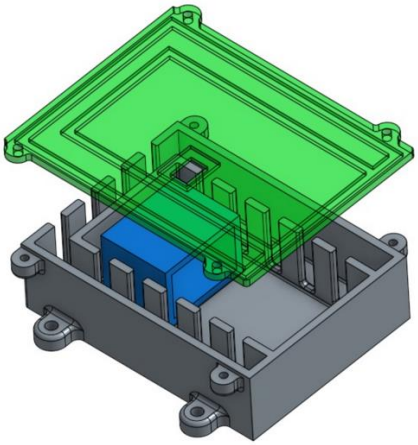

Figure 2. Sensor Case 3D Model

Another issue with our new design was being able to place the TelosB within the case, so we made the internal box holder removable so that we could easily access our sensor. Collaboration with the 3D printing community also revealed that using a resin plastic for our case was not optimal as it does not react well when in direct sun for extended periods of time. To resolve this we outsourced our 3D print to a 3D hub using HIPS plastic. Our final product resulted in a solid functional case that held up well to the outdoor environment. We deployed our case through two thunderstorms and one hail storm without any resultant issues with our electronics proving our concepts feasibility for environmental protection.

2.2.2 Outdoor Sensor Data Analysis

We were uncertain going into this experiment about the accuracy of data we would receive from our TelosB deployed outside within our sensor case. As will be covered in section 3, we had experienced issues with these sensors when in a stable indoor environment so naturally we were skeptical moving to a more complex external environment.

2.2.2.1 NWS Data Comparison

For our first round of viability tests we wanted to compare data taken from our sensor with National Weather Service data. Unfortunately the NWS takes its data in Lambert-St. Louis International Airport which is roughly 15 miles north of our sensor deployment site. Regardless of geographical discrepancies we were still interested in how well our data would align with the accurate data provided by the NWS. This data comparison is shown in Figure 3 and Figure 4 where the red line represents NWS control data and the blue line represents our outdoor sensor.

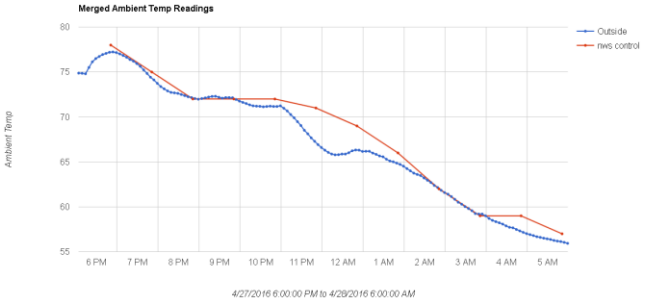

Figure 3. NWS Ambient Temp Readings Comparison

Figure 4. NWS Humidity Readings Comparison

Through our data comparison in Figure 3, we found that temperature readings were very accurate and the TelosB sensor performed well within the sensor case. We see that our outdoor sensor trended with the NWS data for a 12 hour period overnight starting at 6pm. Humidity however did not show the same level of accuracy in Figure 4. This may be explained due to the geographical differences in sensor locations for the NWS and our data collection. Humidity is much more reliant on location than temperature and therefore based on our NWS data we cannot make any claims to accuracy nor denounce the accuracy of our sensor. A follow on effort placing the outdoor sensor case within a reasonable distance from the NWS sensor or some other known good control would help prove the validity of our humidity sensor data.

2.2.2.2 Control Data Comparison

For our next experiment we sought to find another way to validate our case's ability to provide accurate sensor data. To do this we placed another TelosB without a case beside our outdoor TelosB in our case to analyze the effects of the case on our sensor readings. This comparison can be seen in Figure 5, 6, 7, and 8 where the red line representing the outdoor control sensor without a case and the blue line representing our sensor enclosed within the case.

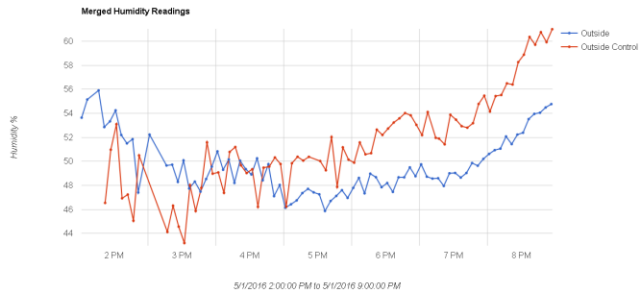

Figure 5. Control Humidity Readings Comparison

Humidity once again seems to give us the most trouble out of all of our sensor readings as shown in Figure 5. The outdoor control sensor data is very sporadic in the first half of the data set from 2pm to 5pm, though it seems to still be fairly comparable to the outdoor sensor data. However, from 5pm to 9pm we see a very common trend between the two sensors with the outdoor control consistently having 4% higher humidity than our outdoor sensor. This could be indicative of the sensor case consistently affecting the humidity readings possibly due to lack of airflow or the plastic itself altering the humidity within the case.

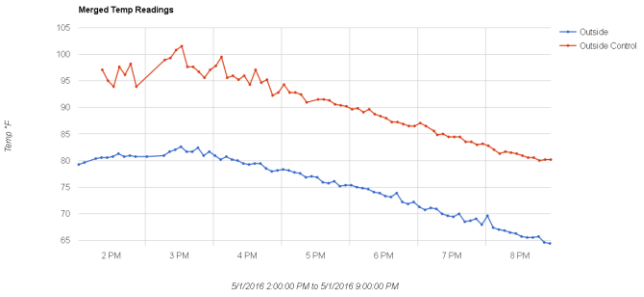

Figure 6. Control Board Temp Readings Comparison

Figure 6 shows the internal TelosB board temperature. As shown by the trends in data our sensor followed the same slow decrease in board temp as night fell by 9pm. Interestingly our outdoor sensor internal temp was nearly 10-15 degrees cooler than the outdoor control sensor. Most likely the outdoor control was hit with a large amount of light as shown in Figure 7 possibly causing the entire board temperature to rise while the outdoor sensor board was protected from the sun by the case allowing it to stay cooler throughout the day. This is very promising for our case design as electronics can be faulty and inconsistent at high temperatures.

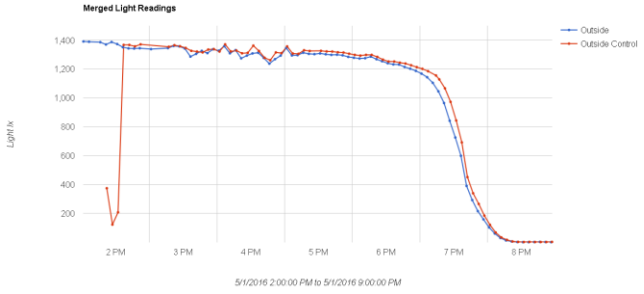

Figure 7. Control Light Readings Comparison

The light data collected between our outdoor control sensor and outdoor sensor was nearly identical for the entire data set as shown by the trending data in Figure 7. This proves that our light sensor window and plastic cover for the opening do not alter our light sensor readings, producing valid data.

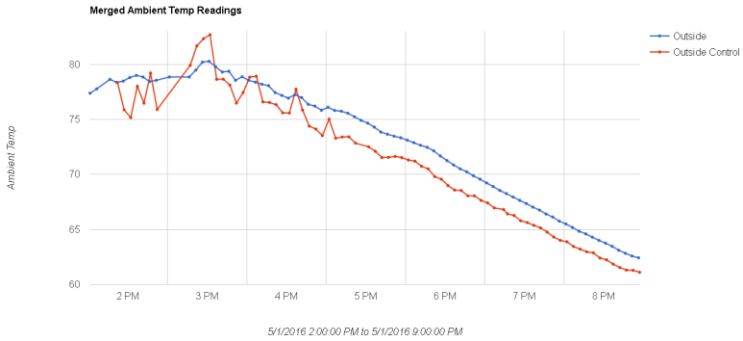

Figure 8. Control Ambient Temp Readings Comparison

The ambient outdoor temperature data taken between our two sensors also trended very well as denoted by Figure 8 with roughly a 2 degree average discrepancy. It is possible that the outdoor control sensor was exposed to wind causing the difference in temperature readings or that the sensor case itself was trapping the heat coming off of the sensor itself causing a greenhouse gas effect increasing the temperature slightly. Most likely the case is the cause of this difference due to retained heat however further experimentation would need to be done to confirm our hypothesis.

2.2.3 Outdoor Sensor Reliability Results

Overall the experimentation went very well with our outdoor sensor case. Based on our collected data we concluded that our case provides a viable solution to temperature and light data while protecting the sensor from the outdoor environment. Further experimentation would need to be done to identify discrepancies in humidity data and to analyze what can be done to our 3D model to increase the accuracy of our humidity readings.

3. ANALYSIS OF RESULTS

Over the course of the project, we were able to take hundreds of mote-hours of data. This section will be a discussion of the accuracy of the data, the challenges we faced in collecting it, and the solutions we found for those challenges.

3.1 Problems We Faced

During demo 2 we successfully demonstrated that we could handle data from three motes without issue. With that success, we added three more motes to the network, but immediately ran into issues. It seemed impossible to get all six motes sending valid packets at the same time. Sometimes one would die altogether, other times the data we would be getting would be inconsistent. Occasionally, a mote would run off sending data far more often than it should and sending out values much higher than possible (see figure 9). Finally, many of the motes we installed the TinyOS application on simply never seemed to send any packets at all.

With such a myriad of issues, it seemed likely there were many causes. The first attempt at a solution was to simply reinstall the TinyOS application. This rarely had an effect. The thought that maybe the software itself had a bug in it was briefly explored, but

it was discovered that the motes that began to die did so sporadically, so a memory leak seemed unlikely.

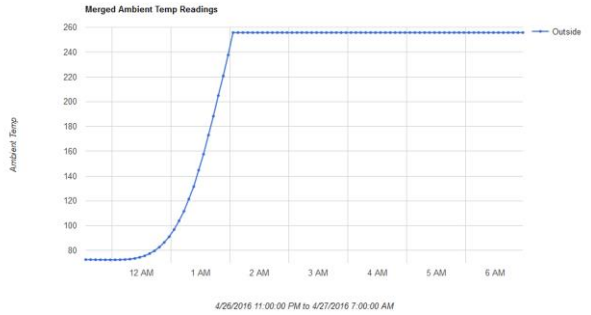

Figure 9. Runaway Mote

3.2 Solutions

The first problem to be solved was the issue of motes that seemed altogether dead. These were initially misleading because the LEDs that indicate a successful install of an application lit up exactly as they should, but no packets ever arrived at the basestation. The first thought was that there was a problem with the radio, so a test software load was written that blinked an LED each time an attempt was made to access the radio. When no LEDs were seen, a simple application that did nothing but turn on an LED was installed. This time, the application worked perfectly while still plugged into the basestation for loading, but as soon as it was unplugged, the LED would die. This test was done with brand new batteries whose voltage had been checked, so it was concluded that some motes were unable to draw power from the battery pack, but would otherwise work on USB power.

The second and more pernicious problem was that of inconsistent data, inconsistent timing, and motes dying. After a significant amount of testing, it was determined that these were all actually caused by the same thing, which is low voltage from the battery pack. Some motes using the same two AA batteries lit LEDs at different brightnesses, suggesting that even though the supply voltage was the same, the voltage powering the boards was not (LED brightness was an unscientific measure of voltage, but one we found effective). In many motes, though, it was simply a case of dead batteries. In hindsight, this problem is obvious and should have been discovered sooner, but with hardware problems causing very similar issues, the problem hid much longer than it otherwise may have.

In order to confirm that the batteries were in fact the cause of the bad packets, I found all of the motes that had died, removed the batteries, and plugged them into USB chargers. They all immediately began to send reasonable, if less than accurate, data. The consequences of long-term use of the USB chargers is discussed later on in this section.

The obvious next step was to discover why the batteries were being drained so quickly. Sending a packet every 5 minutes should have lasted weeks, but we were sometimes losing motes after 2-3 days. Again the answer is not as complicated as first thought. Some of the batteries used during the initial testing phase (when packets were often sent at once a second) had made their way into the final project motes and were causing confusion. The battery drained during the testing period was underestimated. Additionally, our testing showed that our estimate of several weeks of battery life was incorrect. The outside mote, which had been filled with brand new batteries, ran well for a little less than 200 hours. Reducing the frequency of the packets would obviously dramatically reduce these battery problems.

With all of the issues resolved, we began taking data in earnest. See figures 10 and 11 for samples of a typical day of data collected on all six motes.

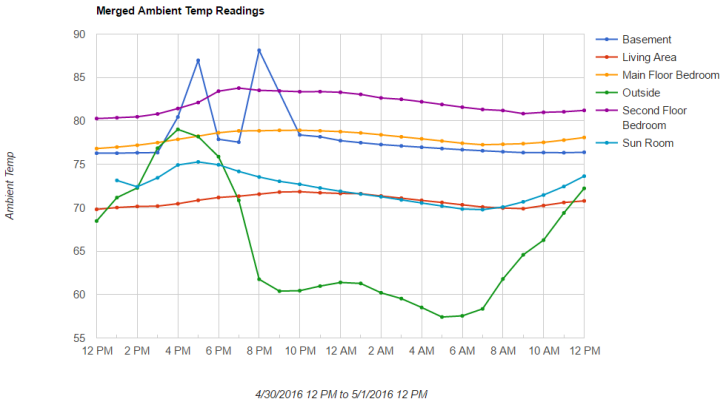

Figure 10. Hourly Averages of Ambient Temperature

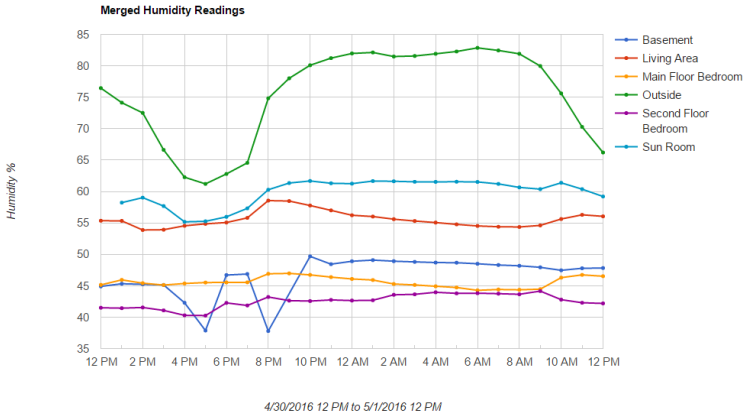

Figure 11. Hourly Averages of Relative Humidity

Note that the spikes in the basement data was due to the close proximity of the mote to a light source. When the light was on, temperatures went up and humidity went down.

Unfortunately, even with all of the problems understood, there was only so much that could be done. We had twelve motes and only needed six to function, but in the end were only to find three that would successfully send packets on battery power. One obviously went to the outdoor sensor, one went to the living area, and the last was in the sun room. The rest were powered using USB chargers. The first thing you'll notice is that all of the motes powered by USB are returning temperatures that are ~10-15% higher and humidities 10-15% lower than the rest. This is a result of the 5 volts put out by the charger rather than the 3 volts the mote was expecting from the batteries. The algorithms used to decode the integers from the motes had a lookup chart for voltage, and the values we used were for 3 volts.

Another interesting anomaly you'll notice is that from time to time, all of the motes on outlet power suddenly dip and come back

up (figure 12). It seems the household power grid has voltage hiccups from time to time. If for some reason you ever did want to use outlet power for a TelosB mote, you should note that a voltage regulator would be necessary. In the end, the USB chargers should stay what they originally were, a quick sanity check to see if the batteries were the problem.

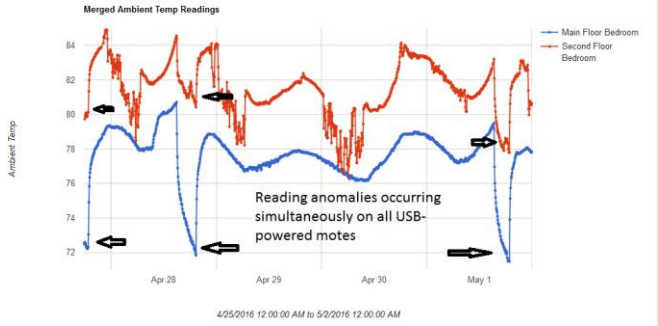

Figure 12. Anomalous readings due to house-wide voltage fluctuations

3.3 Comparison to a Control

A basic test on the data accuracy can be provided by noting how the temperature data aligns with the thermostat settings. During the time of data collection, the thermostat was set to turn on the heat at 68 degrees and the air conditioning at 71. In figure 13 you can see that to within a degree 71, the air conditioning is turning on at the correct time. (Note that spike other than the ones noted are likely due to factors other than the HVAC system, such as it getting late at night and the temperature is naturally dropping).

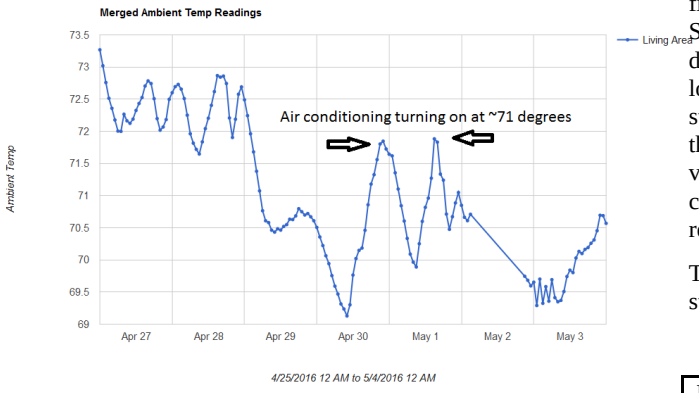

Figure 13. Living area data, annotated with thermostat information

4. DATA STORAGE AND DISPLAY

4.1 Back-end

Going in to this project we knew we needed a well developed back-end, so we could store and make sense of the data collected from the sensors. For the most part the back-end code was completely written from scratch to serve as a learning opportunity, but additionally to provide the flexibility that was deemed necessary for our intended purpose. After some initial thought it was decided to go with a LAMP based server. To that end a Dell OptiPlex 9020 which was on hand was converted to act as our server for this project.

The first step of the LAMP server was the Linux distribution. This ended up taking more time and effort than it should have due to variables outside our control. The first distribution that we attempted to use was the Mint Linux distro, however around the time we were shopping for a distro the Mint Linux site had been hacked, and had their links point to infected modified Mint Linux ISOs[12]. After some further failures with USB boot drives not working, we finally were able to get Ubuntu 15.10 up and running on the Dell.

Apache was for the most part a straight forward install with some minor tweaking. MySQL was designed to be the corner of our data-storage, though the live front-end currently uses the csv storage for its generation. We initially wanted to be able to store the data at various granularities at collection time to minimize the users wait for their dynamic front-end generation. This was not completed due to time-constraints, but could be put in place with minimal effort.

For PHP, it was decided to use the Symfony 3.0 framework to assist in the maintainability and adaptability of our back-end should it see life outside this project. The PHP controllers in the end were setup successfully, but could use some cleanup and minor improvements. There was a learning curve with using Symfont for the first week, and before work on the back-end could begin a basic understanding of the framework was needed.

SSH protocol was setup so we could remote in to the server for updates to the baseline and data. The development was primarily done remotely through SSH, but in hind sight it would have been a better idea to work on the files locally using an IDE and then GTP the work over the production server.

The first part of the back-end code was the PHP controllers. The first controller that was needed was the SensorCollectionController, responsible for collecting the sensor data via HTTP posts and putting that data in to the database and local csv files. The schema for the HTTP posts was 'sensorId' a string label of the sensor, 'timeStamp' a Unix time-stamp of when the data was collected, and then lastly the sensorTypes and their values. The sensorTypes which are currently supported in the controller are 'temp', 'ambientTemp', 'humidity', and 'light'. Light refers to the board temperature.

The MySQL database for which the SensorCollectionController stored its data used the following schema:

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto-increment
sensor_id	varchar(128)	NO	NO	NULL	
timestamp	timestamp(6)	NO	NO	0000-00-00 00:00:00.000000	
sensor_type	varchar(128)	NO	NO	NULL	
value	Float(20,10)	NO	NO	NULL	

The other controllers were responsible for the front-end generation.

4.2 Front-end

Our front-end is accessible by going to http://71.86.152.84, you will be prompted for a user/password. Which we can provide upon request. The three remaining PHP controllers are the IndexController, DataLogSelectionController, and the ChartSelectionController. The IndexController handles the various choices selected on the webpage listed above. Depending on those choices the data is sent to either the DataLogSelectionController which handles calling the custom python code to generate the data log page, or to the ChartSelectionController which calls custom python code to generate the data charts.
The data log front-end was created with the idea that you can select a singular date, and have that day's log information displayed in easily readable html tables. The generated log pages which are created by two modules that were developed for this project: 'create_data_log.py' which was used to parse the data and send it over to the modular 'html_helper' which created the log. The template page that the logs are built in to additionally have quick link functionality, and javascript toggle buttons to hide and show the groupings of data.

The ChartSelectionController sends the data passed to it from the IndexController to the chart generating python script 'create_sensor_charts.py'. A lot of effort was put in to create_sensor_charts.py and it is the most complete part of the back-end and front-end software package. The main purpose of the script is to dynamically create the charts and serve that data to the user upon request. The charts used in this project are created via Google Charts. The most intensive part of the script is the conversion of the raw data into json for the Google Chart API to match the front-end selections. The json data includes the sensor data and display options sent to the chart wrapper for Google Charts which is packaged according to the choices selected on the front-end.

Unlike the data logs, which has one choice a date, the charts have six choices; chart type, granularity, sensor types, merge on sensor type, sensor selection, and date range. The first option chart type is simply the chart you want the data displayed; line chart, area chart, stepped area chart, scatter chart, bar chart, and column chart.

The choice of granularity: all, hourly, daily, weekly, monthly, yearly, will cause the data passed in to the python script to go through averaging where applicable. The choice of granularity will also affect the x axis design, and labeling. Some options pair better with others, all for example is not as readable if used with column charts, as say a line chart.

Sensor Types is a list of sensors which is dynamically populated from the data and will let you select the sensor types to display in your charts, such as temperature or humidity. Deselecting one will exclude that sensor type from your charts, and will result in one less chart displayed on the generated page.

Selecting the merge on sensor type option will make a chart which will merge all data on to the sensor type, and the sensor ids will become series on the charts. This is useful for comparisons of the sensors measuring the same data type.

Sensor selection is similar to sensor type in that it is a dynamically generated from the data stored, and allows the user to hide or show sensor data from deselected and selected sensor ids.

Date range is a calendar widget that allows you to select a date range to show. Similar to the data log this calendar widget will not allow the selection of dates that don't have data, however you can select a range that goes over a day with no data, this will result in estimations of null data for certain chart types such as line.

Additionally on the front-end there is a database link, which will send you a login screen. Once logged in the user will have a GUI to the sensor database, where the user has read access.

The front-end has simple HTTP basic authentication which is something that could be improved upon depending on security concerns. Since we did not implement any sensor control options to the front-end we felt HTTP basic authentication to be secure enough for these purposes.

4.3 Missed Opportunities

As alluded to earlier there were a couple elements that while sought after were not able to be implemented in time for the final demo. A more complex database system would have helped minimize chart generation times. If one wanted to do a yearly average for example, right now the code will go through all the data points to generate those averages. If we could store or update that average in to the database when we store new data points we could greatly reduce the wait time of our chart generation. Adding this would not be that much more effort, but would require a little bit of design.

Outlier detection was something that we wanted to add, but did not end up doing so. Currently we have a very basic outlier detection system, but that is handled by the basestation, while it should be handled by our back-end.

Obviously, re-designing the mote software to increase battery life is also an opportunity that could be built upon.

Finally, we wanted to have the ability to act on the data programmatically, and give the user various control and configuration options for the basestation and sensors via the front-end.

5. REFERENCES

[1] Bowman, M., Debray, S. K., and Peterson, L. L. 1993. Reasoning about naming systems. *ACM Trans. Program. Lang. Syst.* 15, 5 (Nov. 1993), 795-825. DOI= http://doi.acm.org/10.1145/161468.16147.

[2] Ding, W. and Marchionini, G. 1997. *A Study on Video Browsing Strategies*. Technical Report. University of Maryland at College Park.

[3] Fröhlich, B. and Plate, J. 2000. The cubic mouse: a new device for three-dimensional input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (The Hague, The Netherlands, April 01 - 06, 2000). CHI '00. ACM, New York, NY, 526-531. DOI= http://doi.acm.org/10.1145/332040.332491.

[4] Tavel, P. 2007. *Modeling and Simulation Design*. AK Peters Ltd., Natick, MA.

[5] Sannella, M. J. 1994. *Constraint Satisfaction and Debugging for Interactive User Interfaces*. Doctoral Thesis. UMI Order

Number: UMI Order No. GAX95-09398., University of Washington.

[6] Forman, G. 2003. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.* 3 (Mar. 2003), 1289-1305.

[7] Brown, L. D., Hua, H., and Gao, C. 2003. A widget framework for augmented interaction in SCAPE. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology* (Vancouver, Canada, November 02 - 05, 2003). UIST '03. ACM, New York, NY, 1-10. DOI= http://doi.acm.org/10.1145/964696.964697.

[8] Yu, Y. T. and Lau, M. F. 2006. A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions. *J. Syst. Softw.* 79, 5 (May. 2006), 577-590. DOI= http://dx.doi.org/10.1016/j.jss.2005.05.030.

[9] Spector, A. Z. 1989. Achieving application requirements. In *Distributed Systems*, S. Mullender, Ed. ACM Press Frontier Series. ACM, New York, NY, 19-33. DOI= http://doi.acm.org/10.1145/90417.90738.

[10] Bhatia, Laksha. 2014. Installing tinyos on Raspberry Pi. (2014). Retrieved April 4, 2016 from https://lakshbhatia.wordpress.com/2014/04/22/installing-tinyos-on-raspberry-pi/

[11] Zhong, Xiaoyang. 2015. Install TinyOS on Raspberry Pi. (2014). Retrieved April 4, 2016 from https://zhongcs.wordpress.com/2015/02/15/install-tinyos-on-raspbian/

[12] "Beware Of Hacked Isos If You Downloaded Linux Mint On February 20Th!". *The Linux Mint Blog*. N.p., 2016. Retrieved May 8, 2016 from http://blog.linuxmint.com/?p=2994