Intelligent Occupancy Sensing for Heat Control In Academic Offices

Upson Hall Renovation Project

A CUSD Re-Innovations Team

Jennifer Boyd, Roshun Alur, Margot Haas, Nick Teo,
Jimmy Zhu, Priam Mukundan, Victor Delgado

May 16, 2014

Abstract

This project examines the potential energy and cost savings that can be provided by implementing intelligent occupancy sensing to control the heating system for academic (single professor) offices. Space conditioning and ventilation make up 53% of energy use in commercial buildings, so there is good reason to pursue methods that will reduce this demand. While occupancy sensing is primarily used for lighting control, it presents an additional challenge when applied to heating systems. To keep occupants comfortable, the room must be heated before the occupant arrives to ensure the temperature has reached the desired level by the time the occupant enters. In order to address this issue, our occupancy sensing system contains an intelligent component that determines not only where the occupant is, but also where they will be in the future. Our multi-sensor intelligent system is comprised of a radio frequency receiver and transmitter, passive infrarer (PIR) motion sensor, Outlook calendar interpretation, and a K-Nearest Neightbors machine learning algorithm to not only determine when the professor is in his or her office, but when they are likely to arrive. In order to determine how much earlier than arrival we would need to predict, we performed physical experiments on an office in Upson Hall and used ANSYS FLUENT to model the recovery time of a room numerically in order to evaluate different heat recovery equipment. We found that if our system was implemented in a building such as Cornell University's Upson Hall, for example, this system could potentially save 176,525 kWh of heat or $18,852 per year. The predictive machine learning algorithm provides 1% error in predicting the occupancy of the professor, 80% of which was generated in the first week of data processing, making the error after the first week of prediction only 0.2%. Though this project specifically focuses on single-professor offices in Upson Hall, it has the potential to be utilized in nearly any building on a campus with single professor offices, or be easily modified to suit the needs of other types of spaces such as dorm rooms or classrooms.

Table of Contents

1 Introduction **3**

2 Proof of Concept **3**

3 Design Process **7**

4 Technical **11**
4.1 Galileo . . . 11
4.1.1 Getting Started . . . 11
4.1.2 Communication with Linux . . . 11
4.1.3 Moving Forward . . . 11
4.2 Server . . . 11
4.2.1 Database . . . 11
4.2.2 Apache Tomcat Server . . . 12
4.3 Receiver/Transmitter . . . 12
4.3.1 Choosing the Technology . . . 12
4.3.2 Hardware . . . 13
4.3.3 Software . . . 14
4.3.4 Configuration . . . 14
4.3.5 Moving Forward . . . 14
4.4 Motion Sensing . . . 14
4.4.1 Hardware . . . 15
4.4.2 Software . . . 15
4.4.3 Moving Forward . . . 15
4.5 Schedule Interpretation . . . 15
4.6 Prediction . . . 16
4.7 Edge Cases and Code Framework . . . 16
4.8 Temperature Simulation . . . 17
4.8.1 Moving Forward . . . 20
4.9 Mechanical Analysis . . . 20
4.9.1 FLUENT Analysis . . . 23

5 Results **28**

6 Conclusion **28**

7 Appendix **30**
7.1 Appendix A . . . 30
7.2 Appendix B . . . 30
7.3 Appendix C . . . 32
7.4 Appendix D . . . 35
7.5 Appendix E . . . 38
7.6 Appendix F . . . 39
7.7 Appendix G . . . 42

1 Introduction

Space conditioning and ventilation makes up 53% of energy expended in commercial buildings [1]. However, the technology for heating, ventilation, and air-conditioning (HVAC) systems have changed very little over the past 50 years. These systems have not seen major improvements in efficiency or significant advancements in technology, so it is important to look at other ways in which we can save energy through HVAC control. In order to create significant reductions in the energy usage of an HVAC system, we must consider reducing our demand instead.

At this time, motion sensors are widely used for control of lighting systems in both residential and commercial buildings. However, it is much more rare to encounter a motion sensor being used for the purpose of controlling building heating, ventilation, and air-conditioning systems. This is due to the nature of heating and air-conditioning in general. That is, there is a time delay involved in providing heat to or removing it from a space while the air and thermal mass in the room slowly warms or cools. On the contrary, lighting systems can instantaneously provide light to a room based on occupancy, making basic motion sensors more effective with lighting systems instead of HVAC systems. Similarly, the energy savings from a light being switched off are clear and definitive. When the light is off, no energy is being expended in the form of electricity. Heating and cooling systems, however, must maintain a "setback" temperature, or a minimum temperature that the room cannot go below in order to prevent pipes from bursting. As a result, determining the energy saving potential from conserving the use of the heating system is not as straightforward.

In order to account for the fact that an HVAC system requires time to bring a space to a comfortable temperature, we must include a component that is predictive. To ensure that the occupant is comfortable upon entering the room, it is necessary to predict the occupancy of the room to heat or cool the space in advance of their entry. This creates a unique challenge in which electrical, computer, and mechanical systems work together to intelligently detect occupants of a space.

Upson Hall at Cornell University in Ithaca, New York was built in 1956 for the purpose of housing the Sibley School of Mechanical Engineering. Upson Hall is a 160,000 square-foot building that holds classrooms, professors' offices, laboratory spaces, and lounge or study spaces for students [2]. Since then, very little has been done to Upson Hall's facilities, with the exception of an addition built in the 1970's that added a 4th and 5th floor. Until now, levels three through five contained the Department of Computer Science. Due to the recent building of Gates' Hall for the purpose of education in Computer Science, the 3rd through 5th floors of Upson Hall are now unoccupied. The Board of Trustees of Cornell University granted permission for renovation of Upson Hall in September 2013, with construction to begin in 2015 for a redesign of the entire building [3].

This project focuses specifically on the potential for intelligent occupancy sensing in Cornell University's Upson Hall. We are looking only at professors' office spaces, as this allows for a simplification in that the room in question is relatively small and straightforward in its shape (there are no staircases or lofts, etc.) and the occupant is typically only one person. This allows us to predict the schedule of only one individual as opposed to multiple, which is not only increasingly complicated, but will likely result in a reduction of energy savings. Professors often have busy schedules that include teaching classes, performing work in their laboratory spaces (often separate from their offices) and holding office hours. Thus, the amount of time a professor spends in their office per day is potentially quite low. However, it is also likely that a professor follows a schedule, such that on certain days of the week, a professor's schedule is likely to repeat or follow a pattern. This makes professors the perfect candidate for intelligent occupancy sensing for HVAC control. For the purposes of this study, due to the fact that the heating load in Ithaca, N.Y. is much higher than its cooling load and that there is currently no central air conditioning system in Upson Hall, we will focus solely on the heating demand for the offices.

2 Proof of Concept

In order to establish the significance of the study, understand its impact on the energy usage of a building like Upson Hall, and quantify its monetary savings, we considered three cases: how the building operates currently ("Case 1"), how the building could potentially operate after its renovation without the use of our system ("Case 2"), and how the building would operate with our system of intelligent occupancy sensing in place ("Case 3"). Since the building temperature may not go below 55 degrees Fahrenheit to prevent pipes from bursting, we will consider 55 degrees Fahrenheit as our "setback" temperature, and 70 degrees as a reasonable "comfortable" indoor temperature.

The system currently operates on a user-controlled pneumatic thermostat. That is, there is no building-wide set of controls that determines building or zone temperatures. Every room in Upson Hall has one or more thermostats dedicated to controlling the temperature of the space. Thus, every professor has the potential to control his or her office. We assume for Case 1 that the office is set to 70 degrees Fahrenheit by a professor, and that the thermostat is not changed after this setpoint in order to maintain a comfortable level in their office all the time. We felt it was unlikely for a professor to think to turn down the temperature at any time, especially considering the potential for discomfort upon their return.

The second case assumes that the building will be controlled by a central control system that will dictate building "on" and building "setback" mode. This method is more common for modern buildings and allows for energy savings at late hours of the night when the building is nearly completely unoccupied. For the purposes of this analysis, we assume that a building "on" setting would be between 7:00 AM and 7:00 PM, during which time the building will be maintained at 70 degrees Fahrenheit, and when it is not between these hours, the building is reduced to the setback temperature of 55 degrees Fahrenheit.

The third case is the use of intelligent occupancy sensing through the system we would be providing. In this framework, we assume the professor is in their office 5 hours of the day. In order to get this number, we experimented with a basic motion sensor in a professor's office for a week. From this data we were able to create a histogram of the frequency, at a given hour, that the professor was in his or her office over the course of the week, as seen in Figure (1).

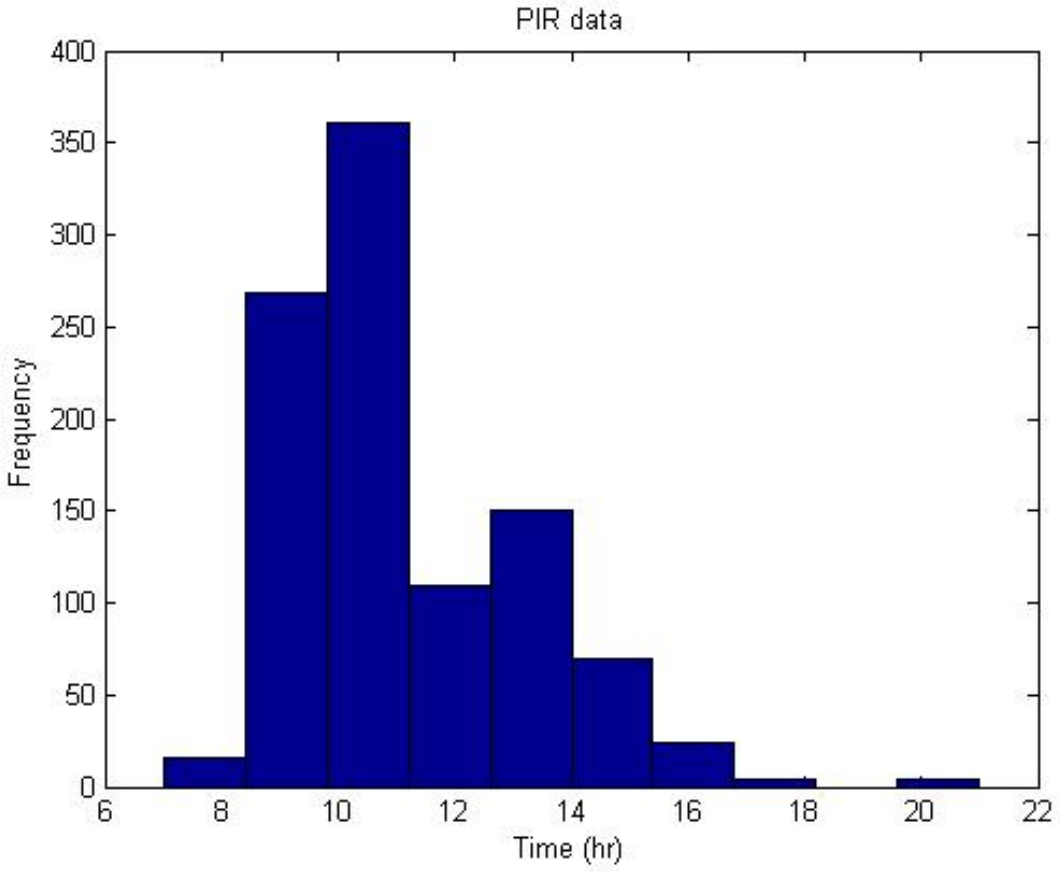


Figure 1: Histogram of occupancy data over the course of a week.

This histogram indicates that the most common hours of the day that the professor was in his/her office was between 9:00 AM and 2:00 PM, a total of 5 hours. It is important to note that the professor whose occupancy this reflects was not teaching a course during the time this data was taken.

To calculate the energy savings, we first must determine the amount of energy that the heating system would have to provide to maintain a comfortable temperature in the room. For the purposes of the analysis, we are only accounting for transmission losses, governed by the equation:

$$\dot{Q} = UA\Delta T \tag{1}$$

Where $\dot{Q}$ is the heat loss in W, U is the U-Value of the glass in W/m^2K, A is the area of the window in m^2, and ΔT is the difference in temperature between the outdoor air and the room in K. It is expected that after the renovation of the building a high R-Value insulator for the walls will be used, and we can neglect losses associated with the walls not connected to outdoor spaces. Similarly, the air handling units for the building were assumed to be providing air at a temperature of at least 70 degrees, so ventilation losses were not considered. Thus, we are only considering the transmission losses equation as it relates to windows.

The U-factor for the glass was chosen based on the fact that windows are typically R-2 (R=1/U) in English Units, making the SI unit for U-factor equal to 2.837. In order to convert from English units to SI units, we can simply multiply the U value by a conversion factor of 5.674 [4].

The area of the glass was based on an office in Upson Hall where we measured the window area to be 8.892 m^2. Thus, the U*A portion of the equation stays constant, herein referred to as 'window coefficient', which allows us to re-write the equation as:

$$\dot{Q} = 25.23\Delta T \tag{2}$$

The temperature portion depends on the case we use. In order to evaluate the different cases, we had to determine the amount of heating degree hours in Ithaca over the course of a year. A heating degree hour is the difference between the outdoor air temperature and a given setpoint temperature multiplied by the duration that the temperature is below this setpoint. For example, if the setpoint temperature we are interested in is 70 degrees Fahrenheit and the outdoor air temperature from 10:00 AM to 11:00 AM is 45 degrees Fahrenheit, the heating degree hours for that hour are (70-45)*1hr = 25 degF*hours. We use heating degree hours because we are only interested in the time that the temperature is below the setpoint, which is when we will theoretically need to heat the room. We obtained the hourly weather data over the course of the year 2013 for Ithaca, New York from the Northeast Regional Climate Center (NRCC), located in the Department of Earth and Atmospheric Sciences at Cornell University. This gave us an hourly temperature for every single hour of the year 2013, which was used to calculate necessary heating degree hours for each of the three cases.

For Case 1, the building's current operation, this calculation is fairly straightforward. Because we assume that the professor sets the temperature in his or her office to 70 degrees Fahrenheit and does not change it, we can simply take the total number of degree hours below 70 degrees F for the entire year. When the indoor setpoint temperature is 70 degrees F, we must utilize the heating system to maintain this indoor temperature for outdoor air temperatures below 70 degrees F. Thus, since we had hourly weather data, we simply scanned the data for a temperature that was less than 70 degrees, and summed the temperature differentials (each increment is for 1 hour). As a result, the total number of degree-hours (in degrees Celsius) is 114,610 degC*hours. (Note: For the purposes of calculations, we used metric units. For the discussion we use English units, as they are more familiar to the intended audience).

For Case 2, the building would be given an "on" time and an "off" time, during which the building temperature would be set to 70 degrees F and 55 degrees F, respectively. We assume for our project that the building would be "on" during the hours of 7:00 AM and 7:00 PM. Thus, we again used the hourly data provided by the NRCC to determine the times between these hours when the outdoor temperature was below 70 degrees F. For the building "off" time, we assume the building would be set to be 55 degrees F, so we utilized the same formula but with a setpoint temperature of 55 degrees instead of 70 degrees, and only for the hours between 7:00 PM to 7:00 AM. The result indicated that the building "on" degree hours were 45,348 degreesC*hours, and the builing "off" degree hours were 37,017 degrees C*hours.

For Case 3, the building occpancy was based on the actual data taken from a professor, which indicated that the hours of interest were between 9:00 AM and 2:00 PM. While this method is not perfectly accurate, it is a good estimation of the amount of heating that will be required on a professor's schedule with our method of intelligent heat control. The heating degree hours between the hours of 9:00 AM and 2:00 PM are 16,150 degreesC*hours and a setback heating degree hour of 52,496 degreesC*hours.

By calculating the required heating demand for each case, we can subsequently plug these numbers into the Equation (2) to calculate the heating required in kWh for each of the three cases:

$$\text{Case 1: } Q = 2,891.38 \text{ kWh} \tag{3}$$

$$\text{Case 2: } Q = 1,144.04\text{kWh} + 933.86, \text{ kWh} = 2,078 \text{ kWh} \tag{4}$$

$$\text{Case 3: } Q = 407.43\text{kWh} + 1,324.37, \text{ kWh} = 1,732 \text{ kWh} \tag{5}$$

Evidently, by reducing the demand for heating in an office of Upson we can reach some significant energy savings. If we scale these energy numbers to the entire building we can recognize even more savings. A count of the number of offices in Upson Hall indicates that there are approximately 151 single-person offices on all floors (basement through 5th floor). Thus, when we multiply these energy savings and scale it to every office in the building, we recognize potential building-wide energy usage of:

$$\text{Case 1: } Q = 436,597.96 \text{ kWh} \tag{6}$$

$$\text{Case 2: } Q = 313,763.11 \text{ kWh} \tag{7}$$

$$\text{Case 3: } Q = 261,501.64 \text{ kWh} \tag{8}$$

In order to estimate cost savings, we must determine the amount of steam Upson Hall uses for heat. The system currently operates such that the steam from the Cornell Combined Heat and Power Plant enters the building and is used to heat hot water that travels through the radiators. In order to quantify the cost associated with the heating demand, we determined the average yearly steam usage for Upson Hall and the price associated with that steam use. This data is readily available on the Cornell Facilities website. The average yearly steam use for Upson Hall is 9,515 klb of steam, which is an average of Fiscal Years 2010 through 2012 as well as Fiscal years 1998 through 2001 [5]. If we breakdown the cost Cornell paid for steam over every month of the years form 2010 to 2012, we find that the average cost per thousand pounds of steam is approximately $27.20.

Upson hall has a net building area of 141,645 square feet [2], and each office is approximately 169 square feet, based on a measurement of Upson Hall Room 338, a standard office for a single professor. After counting the number of offices in Upson Hall (Basement through 5th floor), we arrive at a total of 151 offices, which theoretically make up 25,519 square feet. This calculation assumes that all of the offices are the same size, which is not perfectly accurate but is a good approximation. Thus, the offices in Upson Hall make up 18% of the total area of the building. Therefore we can assume that Upson's offices account for approximately 18% of the steam use in the building.

For Case 1, which is the current building operation demand, we simply multiplied the average yearly steam usage by 18%, and multiplied by $27.20 per klb of steam, making the total cost associated with Case 1 equal to $46,627.28. In order to determine the monetary cost of steam for the centralized controls and our intelligent occupancy system, we simply scaled the building's steam usage by the ratio of the energy usage in kWh to the energy used in Case 1. As a result, we found that the cost associated with heating the offices in Upson Hall using a centralized system (Case 2) is $33,508.91 and the cost associated with heating the offices using our intelligent occupancy sensing design (Case 3) is $27,774.99.

Thus, there are significant energy savings to be had by implementing our intelligent occupancy sensing system. If we compare the current system to the utilization of our controls (Case 1 versus Case 3), we see that our system would save approximately 175,096 kWh per year, which translates to savings of $18,700 annually. If we compare the use of centralized controls to our system (Case 2 versus Case 3), we see energy savings of 52,261 kWh per year, which translates to $5,581 annually.

However, there is a caveat to this analysis. We are assuming that after Upson is rennovated that the building will still maintain the same number of office spaces as it did before the renovation. With the Computer Science department moving out of Upson Hall and into newly built Gates Hall, it is likely that Upson will only seek to retain the offices dedicated to professors of Mechanical and Aerospace Engineering (MAE). If that is the case, our analysis is scaled down significantly. The number of MAE offices is 43, compared to the total number of offices which is 151. If we only consider 43 offices in our analysis, we find that the Case 1 versus Case 3 savings are 49,862 kWh and $5,325 per year, and the Case 2 versus Case 3 savings are 14,882 kWh and $1,632.84 per year. It will be up to Cornell facilities and faculty to decide how many offices are retained after the rennovation, as well as if $1,589 dollars result is signficant enough to implement our system. Table 1 shows the results of the energy and cost savings analysis broken down by case and number of offices.

Table 1: Potential Energy and Cost Savings

	All Offices	MAE Offices Only
Intelligent Vs. Current	175,096 kWh $18,700	49,862 kWh $5,325
Intelligent Vs. Central	52,261 kWh $5,581	14,882 kWh $1,589

Nonetheless, these savings could be utilized throughout Cornell's academic buildings, generating even larger payback. The amount of savings is directly proportional to the number of academic offices, so the more buildings on campus that utilize this system the more beneficial it will become to Cornell.

3 Design Process

The design process was a key element in the development of our project. We started by identifying the key elements to our problem and discussing a detailed problem statement.

The problem we are trying to solve is intelligent occupancy sensing for controlling the heat in a professor's office. We started by making our project as specific as possible. We decided not to discuss other types of spaces or other types of occupants and focus solely on a professor's office space, as this was the most simple and straightforward case.

Our problem is unique in that in order to determine occupancy information that is valuable for heat control, we need to include elements to the system that allow for knowing where the professor is (or is not) at a given time, and using that information to determine when they will be in their office before they get there. This can be accomplished in various ways that we discussed. An important aspect of our project is to consider all three of the cases described previously, which is how our system will operate relative to the current case - heating set to 70 degrees and never changed, and the potential future case, which is centralized controls that turn the heat on at 7 AM and off at 7 PM on a daily basis. Thus, we kept in mind improving upon both of these cases with our solution, the intelligent occupancy sensing model.

We started with the idea that professors do not operate on the typical 8 AM to 5 PM schedule that many other business people work. Professors may come in early some days, and on other days come in right before their class at 10 AM. Similarly, we wanted to account for days that professors left for conferences, weekends, and sick days. These situations are times when we expect the professor not to be in their office, but with the building running on a centralized control system, the building would be set to "on" thus wasting energy heating their office. In order to account for this, we discussed a method of wireless communication between the professor and their office to be used to determine when a professor was within a certain radius of the space, which made it clear that the professor was coming in at some point that day. That is, they were not taking a sick day, at a conference, late to work, etc. We'll call this information "vicinity determination."

The next concept we discussed was that if a professor is in their office, we must always have them comfortable. That is, our system should always be able to be overridden by the professor if they are present in their office. No matter what, if a professor is in the office, the office must be at a comfortable temperature. We'll call this information "occupancy override."

Finally, in order to make occupancy sensing useful for heating systems, we must preemptively provide heat to the office before the professor enters. Thus, we must find some method of prediction to determine when the professor will be in his or her office, or find a way to know when the professor will be in their office based on definitive schedule information. We'll call this information "prediction and scheduling."

Vicinity Determination

Vicinity determination is one of the highest level types of determination in our project. It is the most broad but also able to save energy on its own. We considered a few cases for how to determine if a professor was in the vicinity of their office, which depended on a wide range of technology and personal parameters.

One of our first ideas was to utilize technology that was already created for tracking objects using a small keyfob or square. We initially assumed that these technologies used GPS to track the objects' location. There are many products similar to these, such as Tile, which give a small keyfob or square to

an object or person, allowing for digital tracking [6]. This idea was primarily conceptual, as we knew that technologies such as these existed. Upon further investigation into companies that had already created these technologies such as Ninja Blocks[7], Tile[6], and Hipkey[8], we found that many operated over BlueTooth 4.0. This is very useful for short range location determination such as within a building, but does not work well over long ranges or outdoors. Surprisingly, we did not find many of such technologies that used GPS instead of Bluetooth in order to track occupants, so we primarily ruled out using a pre-created technology to track the professor.

Next we discussed the potential of giving the professor the power to tell their office when they are on their way to work. We considered the idea of a smartphone application that allows professors to send an "I'm on my way" signal to their office to indicate that they will be coming in to work that day. This concept was rejected due to the fact that we must rely on the professor to remember to "turn on" their office at the start of the day. Not only does this require "work" on the end of the professor, we also run into the issue that the professor would then subsequently need to "turn off" their office when they left for the day. While this is in theory quite simple, we are relying on the professor's mindfulness for the success of the project. While the professor likely has incentive to turn their office "on" to maintain their personal comfort, they are much less likely to remember or be willing to turn their office "off" at the end of the day. If we could rely on professors to turn down their heat at night, they might already be doing this with the current system that is in place, but they do not. By relying on the professor to remember and be willing to expend the effort to open the app and turn down the office's heat is an imperfect system and relies on the professor's habits, a variable we did not want to dilute the results and potential energy savings. Similalry, this method is intrusive to the professor's life, and we assumed that he or she would not enjoy having to remember to do these changes. Finally, there is the issue that some professors do not have smartphones, and our ability to implement the project would rely on them owning smartphones, which not all do. If we were able to develop this app, in order to make it the most useful to those with smartphones we would have to develop it on both iPhone and Android, which is not a trivial thing to do. Thus, we did not consider making a system that relied on smartphone technology or required the professor to be responsible for the room's heat.

Finally, we considered using a small GPS system over radio frequency technology to determine the location of the professor and their distance from their office. This would mean that the professor would need to carry a small GPS module with them at all times, which, since we found a small GPS solution on sparkfun.com with the Venus GPS module and antenna would not be a major concern. However, the concept of tracking the professor's location via GPS for the purposes of our project was cumbersome. We simply needed to know when a professor was within a certain distance range of their office so that we know that they are coming to campus that day. Thus, it was unnecessary to create code that interprets GPS data to give a distance between the office and the professor. The advantage to using a GPS module is that we could have changed the range at which we wanted to activate the algorithm. That is, we could set any range using the GPS module to be the setpoint for when the vicinity is true. This is helpful as the mechanical model develops, as it would allow us to take into account recovery times of individual offices.

After discussing these methods we realized that the GPS was not necessary to determine if a professor is in range. We could simply use a the radio frequency receiver and transmitter that would have sent the GPS signal, and instead use them to determine when they are within range of one another. Our system would have one RF module sending information, and when the other RF module picks up the data, and send its back, we know the professor has entered the range of his or her office. We simply must choose a reciever and transmitter that have a long enough range. Thus, the vicinity determination method we chose was a simple receiver and transmitter that simply determines when the professor is in range, and simply returns "true" to our system, instead of a specific location.

Occupancy Override

The concept of occupancy override was of utmost importance to our project design. We discussed the need for the override option because no matter what the outcome of any of the predictive or other methods, we must always ensure that when the professor is in their office the heating system will be on to maximize occupant comfort. Similarly, having true occupancy information about the room allows us to more easily determine the effectiveness of our system and predict future occupancy. We are able to use basic occupancy information to predict the future occupancy by comparing the current day's occupancy to previous days' occupancy. Finally, having true occupancy information allows us to create a graph of our results that show when the professor was in his or her office and by simulating a temperature model, we can determine if the occupant was or was not comfortable while they were in their office. Thus, there

were various reasons why an occupancy override method was necessary to include in our system: for occupant comfort, prediction, and modelling our final results.

Prediction and Scheduling

Other research teams have used machine learning algorithms such as K-Nearest Neighbors in order to compare previous weeks of occupancy data with the current week's occupancy data as a method of prediction [9]. We found this to be a very straightforward algorithm that would suit the needs of our project the best. While we saw the benefits of using a machine learning algorithm to predict occupancy, we needed to find an aspect of the project to improve the predicted portion even further. We expected that by improving our ability to know where a professor is at any given time, we will be able to better understand when they will be in their office next. We found that many professors utilize their Outlook calendars extensively for keeping track of meetings and appointments, according to Professor Schneider. Similarly, many of the team members have encountered professors who use their calendar in the signature of their e-mails in order to better schedule with students, thus making their calendar likely very accurate. We realize, however, that there will be many caviats that come with using a professor's Outlook calendar. The professor may or may not be putting in the correct location for the meeting, if at all; the professor may be using different abbreviations for locations on their Outlook calendar such as "UPS213", "Upson 213" "Room 213 Upson" or even simply, "My office" or "office." Due to these shortcomings and the fact that we cannot expect the professor's entire day to be mapped out in Outlook with locations, we must include our occupancy sensing portion. Also, there is the ethical issue of checking a professor's calendar. We expect them to be forthcoming with this information, and understand that we are using the calendar purely for scientific, energy saving purposes and will not be looking at the calendar specifically. Nonetheless, we felt that utilizing the information provided by an Outlook calendar would increase the reliability of our results and give us a better method of predicting the professor's schedule than simple occupancy comparisons using K-Nearest Neighbors machine learning algorithm. We felt that using this method helped to set our project apart from other intelligent occupancy sensing reports.

Temperature Modeling

An important component of our design process was to determine how long it would take for a room to go from the setback temperature of 55 degrees to the comfortable temperature of 70 degrees. In order to accelerate the design decisions necessary for the electrical portion, we assumed a recovery time of 15 minutes which is not necessarily very accurate. While this is likely accurate if we are considering an empty room with only air, it is not very reasonable when we consider the impact of thermal mass on the system. Due to the fact that we needed to make this assumption early on in order to implement our design process of choosing a method for prediction and the range of our vicinity determination, we made assumptions based on initial experiments and calculations. In the future, we recommend to perform mechanical analysis well in advance of developing the electrical system in order to create the best system for the needs of the occupant and ensure prediction and temperature modeling is realistic from the beginning. We decided that as a final deliverable for our system, we would model the temperature as a function of time along with the occupancy and the temperature output signal to determine the effectiveness of our model. We designed the system such that our final product would be an overlay in which the output signal is either a 0 or 1 indicating whether the temperature should be set to 55 degrees or 70 degrees Fahrenheit, and the motion sensor data would also be graphed as a 0 or 1 indicating whether the occupant was in the room. Finally, we would use a temperature simulation based on the mechanical analysis results to show how the temperature varies over time and identify periods where the temperature was and was not making the occupant comfortable as well as saving energy.

Future Improvements

Real time tracking information - We discussed the potential for real time tracking using RFID tags or magnets that utilize the earth's magnetic field in order to determine where a professor is in a building at any given time. Using this data, we could potentially utilize the likelihood that based on where a professor is at one moment and compare that to any previous moments, we can predict where the professor is going. This type of all-in-one system would be utilizing these mechanisms for all three of our separate technologies used for vicinity determination, true occupancy, and prediction. While we decided not to go down this route as it seemed to be all-or-nothing as well as quite difficult to implement, this is an interesting route for the future. By being able to track the occupants throughout a building (or even the entire campus!) we can make more accurate predictions about the likelihood of being in their office.

Wearables- Wearables are an up-and-coming technology that we would like to utilize. Perhaps our project could be better suited to the needs of the professor if we incorporated a wearable piece that told the Professor how much energy he or she was saving, simply by utilizing our system. This type of device could double as a method of real-time tracking, as mentioned above. Wearable technology is quickly becoming a buzzword in the technology industry and it would be very interesting to pursue this further.

4 Technical

4.1 Galileo

The Intel Galileo is an exciting new Arduino-compatible microcontroller released by Intel that features the x86 architecture. The board contains a full Linux distro on it and is able to run Arduino sketches to make use of the Arduino I/O pin headers. The ability to access a Linux machine in addition to the ease of programming of Arduino sketches give flexibility that is needed in our system.

4.1.1 Getting Started

To get started with the Galileo, read the getting started guide in addition to the plethora of guides and documents that are provided by Intel to assist in developing and using their platform. Be sure to have a micro SD card that is greater than 4GB, but smaller than 32GB. This will be used to hold a more complete version of Linux that adds SSH support among other features like saving sketches so that the the Galileo can be plugged in to power it up and run the saved sketch. A micro USB cable is also necessary to be able to upload sketches onto the Galileo. There is software that needs to be downloaded to use with the Galileo: http://arduino.cc/en/Main/Software. This is a special version of the Arduino IDE needed to be able to write and upload sketches for the Galileo.

4.1.2 Communication with Linux

Communicating with the Linux portion was a little harder than expected. We were expecting to be able to connect to the Galileo through USB and then access the Linux portion through serial. However, this was not the case and instead we were able to connect the Galileo through ethernet to the computer and then connect to it through that connection.

4.1.3 Moving Forward

The Galileo has a lot of potential that has not been tapped in this project. Much of the code that we have written for the board has been solely Arduino sketch code. There may be ways to integrate the Linux portion of the Galileo to add more power and flexibility to our system. This would allow us to create more robust code and be able to interract with the environment better.

The Galileo also has an ability to connect to Wifi. We did not use this because that would require a wireless card to get it to work, but that may be another path that could be explored. This would make it easier to connect to the internet almost anywhere on campus because it may be hard to find areas with ethernet ports.

4.2 Server

The server provides a central location to aggregate all data to. Given that in a full implementation multiple Galileos would be used, we needed to design an easy to access server that would allow for quick transmission of relevant data to and from the Galileo. Also, the server would us to off-load any intense processing off of the Galileo.

Please note that we initially attempted to build a server based off an individual's laptop, but Cornell's network caused issues in opening and receiving data in ports. To resolve this issue, we went for a cloud based approach and built a website using Amazon's Cloud Tools to allow for unrestricted galileo access everywhere.

4.2.1 Database

The back-end of the setup relies on a PostgreSQL database. We decided to use PostgreSQL over another type of database primarily due to it's ability to have trigger functions. Trigger functions allow us to have the database handle some of the controlling logic behind the server without requiring the server to perform multiple queries on the database to set certain states, thus it helped to lower the response time of requests as the controlling logic of the room state runs faster with these trigger functions. Primarily

the trigger function looks for any change in the flags attached to a given room. If the database row for that room has any of its flags updated the trigger function will fire. Once triggered the function will set the state of the thermostat to on or off depending on what flags are currently set on that room.

Postgres also offers a wonderful IDE for connecting to and monitoring databases called PGAdminIII. This IDE helped tremendously in debugging and determining the status of the database. There is also much documentation in how to use the IDE to connect to and modify the database.

4.2.2 Apache Tomcat Server

The server is setup to run on Amazon Web Services using a lightweight Apache Tomcat webserver written in Python using the web framework called Flask. Originally the server was created using Flask; however, it was switched over to Apache Tomcat to better handle requests when under load. It uses simple GET requests to send and receive data from the server, which in turn communicates with the back-end database. The server is accessed via the following pages:

- \- Index Test page for connection
- \insertPIR?isinroom=true,false&roomid=integer - insert data to pir, should be substituted for value
- \insertxBee?isinrange=true,false&roomid=integer - insert data to xBee, should be substituted for value
- \sendGalileo?roomid=integer - returns the desired status of the thermostat for the roomid. Returns 1 or 0 to the webpage

In addition to the main thread, the server runs two additional threads, one to handle pulling calendar data and one to update and check room states. The calendar thread looks at every link to a calendar it has stored in the database and then pulls down the data from each calendar. It then extracts the location, start time and end time of each event. If it detects that the event is in Upson it then will proceed to insert this data into the database.

The room thread works by continuously monitoring the data in the database and then performing certain operations based on this data. The operations are as follows:

- Look for all rooms that have an event now or in the next 20 minutes. For all such rooms found set a flag on the room signalling that an event is starting
- Check for any room that has not had PIR data in the last 30 minutes and has last PIR data for it as false. For all these rooms set a flag on the room indicating that the pir data is now false.
- Check for any room that has had no PIR data within the past 35 minutes for a room that has been marked to have an event occuring. For all such rooms set the event flag ont he room to false
- Use the machine learning technique described in 4.6 to set a flag on each room if it predicts the professor will be in the room

4.3 Receiver/Transmitter

The system needs a way to keep track of the professor on campus so that we would know whether to look at PIR data and Outlook data to heat the room. This would involve knowing when a professor is within around a mile radius of Upson Hall.

4.3.1 Choosing the Technology

For this portion of the project, we had to decide between the RF receiver/transmitter, a GPS dongle, and using smartphones.

GPS was ruled out because if the dongle knows where the professor is, it needs a way to communicate with the server so that the system knows that the professor is on campus. This would require either a

wireless internet connection or some way to communicate with the Galileo and it would be both expensive and difficult to interface the GPS dongle with the Galileo. There is also the problem of privacy if GPS is used because the system will know exactly where the professor is at all times, which would be unsettling for most people.

The smart phone was also ruled out for a number of reasons. First of all, using GPS on a phone drains a lot of battery. Another reason is that not all professors have smartphones. There is also still a privacy issue with using GPS on the professors' phones. Using a smartphone would circumvent the problem of communicating with the server because smartphones would have internet connectivity and can send data to the server, however, due to the three reasons listed, smartphones could not be used.

The last candidate, RF receiver/transmitters, was promising because there were some that were advertised as having very long range (good for the large campus). They also are not as invasive as GPS because they only state whether a professor is within a certain range of the central transmitter. There is no specific location tracking where the professor is at all times. The team settled on the XBee made by Digi as the RF receiver/transmitter to be used.

4.3.2 Hardware

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 was to have a coordinator XBee attached to the Galileo. The Galileo will continually send serial data to the XBee which will broadcast the data to all the XBees in the network. The other XBees in the network will be connected in loopback and send the data straight back to the XBee on the Galileo. From this information, the Galileo will know which XBees are in range and which are not.

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.

Portable Unit A portable unit was built using an XBee and a smaller breadboard. This was done using using the XBee breakout board and a CR2450 battery to deliver 3V to the XBee. The XBee was connected such that any message that it receives would immediately be sent back out to the coordinator XBee. This involves shorting Dout and Din such that every message received will be put into the output buffer. This would signal to the coordinator that it is in range. There is also an LED on it to signal when the device has received communication from the Galileo.

One of the troubles with using the portable Unit was that the battery may not supply the XBee with enough power for long enough. This could be remedied by using multiple batteries or AA or AAA batteries.

Shield The shield was another hurdle that we had to overcome. According to a Sparkfun comment on the page, the Galileo has a 1 KΩ series resistor attached to the serial TX line. This causes problems with the shield because when the Galileo tries to send a low signal to the shield, the shield detects a 2.5V signal, which is not low enough for the shield to sense low. This was fixed by replacing resistor R6 in the XBee Shield Schematic (see Appendix D) with a 10 KΩ resistor.

4.3.3 Software

The Arduino code was written for the Galileo and is attached in the appendix. The code broadcasts a message to the XBees and then when it gets a response, it will know that the XBee is in range. Once the XBee is in range, the Galileo will send a message to the server saying that the XBee is in range, it will continually check in with the XBee to see if it is still in range for five minutes. If the XBee does not respond for five minutes, the Galileo will send a signal to the server stating that the XBee is now out of range.

4.3.4 Configuration

The XBee needs to be configured to get it to work. This is done using the X-CTU software. The PAN ID needs to be set to be the same on both XBees and the coordinator XBee needs to be set to broadcast the message, and the mobile XBee needs to be set to send messages only to the coordinator. Look at Tunnels Up's video series on Youtube explaining how to use the XBee to determine what configuration details need to be changed. To save battery, sleep cycle is used on the mobile XBee. This is so that the XBee will sleep and then wake up for a short interval to check if there is a message for it. Once it wakes up and there is a message for it, it will send back the message to signal that it is in range now. If it wakes up and there is no message, it will go back to sleep after a certain interval and wake up again after another cycle. If the XBee does not work at first, make sure that the voltage being supplied to it is sufficient for it.

4.3.5 Moving Forward

As of now, the Galileo is set to work only with one XBee because it cannot differentiate between XBees. This is solely for a proof of concept. To make the system work for multiple XBees, the coordinator must be set to API mode and broadcast a message to all the other XBees in the network. The XBees in range will then send the message back to the XBee on the Galileo which will send a "frame" to the Galileo which will contain information about the source address. Using this source address, the Galileo will be able to tell exactly which professor is in range of the building and use that information to determine which PIR sensors and Outlook calendars to keep track of. These, of course, are only if the group chooses to continue with the XBee receiver/transmitter pair.

XBees are rather expensive. Each one of the 900MHz XBees costs around $60. That is almost as much as a Galileo and each professor would need to carry around one of these. XBees are designed to have high throughput and low latency, neither of which are very important to our purposes. These both increase the cost of the device. There may be cheaper ways to keep track of whether professors are on campus.

XBees also do not have as much range as we need. Ideally, to effectively determine when to start looking at PIR data or Outlook data, the system needs to know when the professor is within a mile so that there is enough time to determine if the heat in the room should turn on in preparation for the professor's arrival. This is not accomplished with either of the XBee pairs that we have tested and there may be other alternatives.

Some ideas of alternative ways to determine if a professor is on campus are a device that connects to a campus WiFi network and sends a signal up to the server, smartphone apps that use the GPS on the phone to determine where the professor is, alternative RF transmitters that are cheaper. There may be other more cost effective ways to keep track of professors.

4.4 Motion Sensing

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.

4.4.1 Hardware

The PIR sensor has three pins; a Vcc pin (5 Volts), a Ground pin, and a signal pin that outputs high or low. 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.

4.4.2 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 can be seen in Appendix C, 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).

4.4.3 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.

4.5 Schedule Interpretation

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.

Therefore, it would be beneficial to know whether or not the professor was expected to be in his/her office or out of it 15 minutes before a given time. 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 E. 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. It might at first seem strange for a professor to be willing to share all of his calendar information with someone, but professors already provide the information of whether they are in their office or not to their students so that students know when to come in to ask them questions. Then, after the schedule has been shared, it will create a dynamic link (a .ics file) that will be continually polled by a python script running on the server, meaning that the group members would not be looking at the calendar or sharing it with anyone else. The reason it is shared with a Gmail is because if it is shared with another Outlook account, the schedule will just be added to a new calendar of the Outlook account of the person it was shared with. This is a problem since Outlook has a lot of barriers to access for privacy and security reasons, unlike google.

4.6 Prediction

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.

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.

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). For the specific code used for simulation see Appendix E. 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 2.

4.7 Edge Cases and Code Framework

The code running on the server will take in the data from the receiver/transmitter, motion sensor and Outlook calendar and determine whether the office should start heating up by sending a 1 or 0 to the Galileo board. If the receiver and transmitter are out of range of each other, the server will output a 0 to the Galileo board. If it is not in range then the server would look at the data from the motion sensor. If the motion sensor tells the server that the professor is in his/her office, the server will output a 1. Otherwise, it would look at the Outlook calendar. If the professor has an event in the next 15 minutes in his/her office, it will output a 1. However, if 30 minutes later the professor is not in his/her office (did not show up for the scheduled event), it will output a 0. If the professor has an event within the next 15 minutes in a location that is not his/her office, it will output a 0. If the professor has no event scheduled within the next 15 minutes with a location, then the predictive algorithm mentioned in the

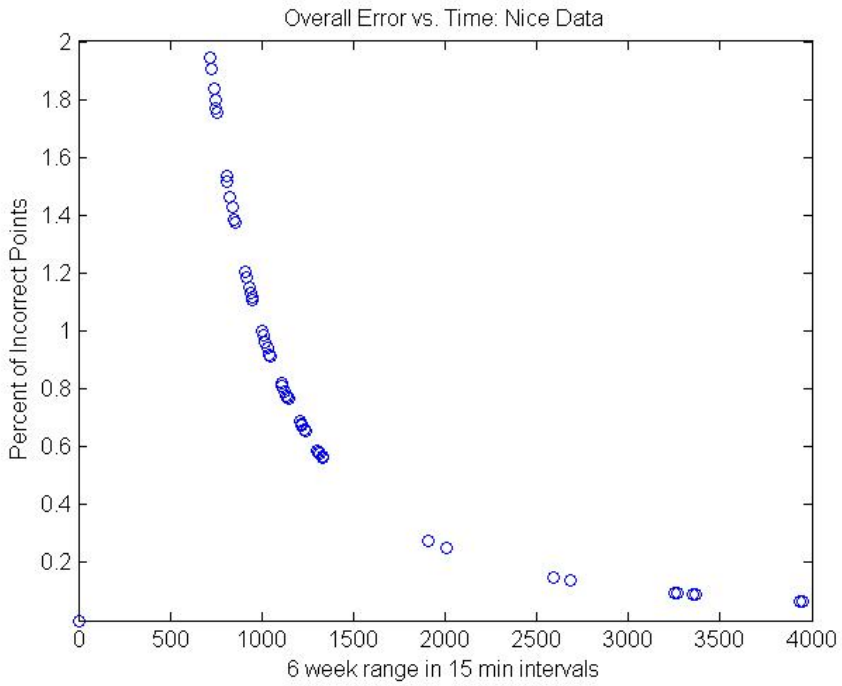


Figure 2: Error Points of Data Exhibiting a Typically Nice Weekly Trend

above section is used. For the overall code on the server see Appendix B.

There are many different cases that could occur that would affect the performance of the occupancy prediction algorithm. These cases will affect how accurate the predictive portion of the algorithm is since the other sections take in a definitive 1 or 0 while the predictive portion takes in past data and converts it to 1's and 0's. If the professor has a weekly trend to when he/she is in his/her office, the predictive algorithm will be very accurate (as shown above with the case of the sample data). The error was at 1% with a schedule that is close to a perfect weekly trend, with 80% of that error coming from the first week of prediction (so second week of initialization). This means that predictive part should always output a 1 until it has collected two weeks' worth of data. This means that for a 4 week period of prediction (after the first two weeks of collecting), the predictive portion would only be off for an average of 2.88 minutes a day (error of 0.2%). This allows the server to not have to store data that has been there for more than 6 weeks. If the professor has a random schedule, the error would be at 16%. However, as explained in section 4.6, professors follow a weekly schedule for most of the semester, so it is unlikely that it would be random. Also, the predictive portion is only one part of the overall code, so the overall error of the whole system is even less.

4.8 Temperature Simulation

The goal of the temperature simulation was to create an emulated plant model of an office. We wanted to be able to model the temperature of this theoretical office in real time given a heating signal (a 0/1 boolean signal).

The first step in designing our temperature simulation was to obtain heating data from offices in Upson. An office was allowed to cool via an open window until it reached a steady temperature. The window was then closed and the room was heated back to the "regular" temperature denoted by the thermostat setting (70 degrees Fahrenheit). Time points were taken of this process and after plotting them, the data cleared appeared to be exponential (see figure 3). A curve fit was then applied to the data in order to obtain a time dependent model.

After obtaining the heating and cooling functions that described these respective behaviors of the

office, the next step was to utilize them so that given the current temperature and heating signal, the temperature after a time step could be found. In order to do this, the inverse function of the heating and cooling models needed to be found. However, the model provided by MATLAB's curve fitting function was a two term exponential function ($a * e^{-bx} + c * e^{-dx}$) which has no symbolic function inverse. The curves were then re-approximated by hand in the form of a single term exponential summed with a constant ($a * e^{-bx} + c$). The goals were to preserve the "steady state values" (when $t \to \infty$) as well as the approximate rise time. The steady state values were nominally "known" to be 55 and 70 degrees Fahrenheit at low and high steady states (respectively). From the experimental data they were seen to be approximately 56 and 69, so for the model these values would be taken as the highs and lows. The time point data can be seen in figure 3.

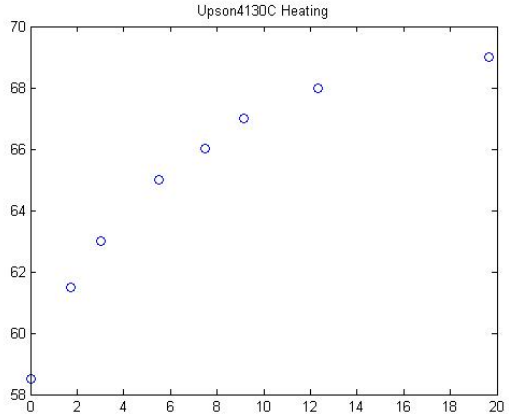


Figure 3: The timepoints taken from Upson4130C

After the heating and cooling curves were obtained, the mechanism for the simulation was designed. The only given knowledge was that the "start" temperature of the simulation each day would be the settled "cool" temperature. Since the input heating signal could change at any time, we would need to be able to switch between the cooling and heating curves every time the signal was sampled. A conditional statement was used in the model to determine which curve to use. From there however, the next problem was that in order to determine the next temperature, the location on the current curve was needed (both temperature AND time). To find the current time on the curve, the inverse of the function was used. The time step (taken as 1 second) was then added to the current time, and supplied to the function to obtain the next temperature. The general algorithm can be summed up as follows:

Algorithm 1 PseudoCode Summary of The Heating Model

Let $H(t)$ denote the heating curve
Let $C(t)$ denote the cooling curve
$T = 56°F$
$t_{step} = 1s$
while running **do**
  $HeatSig = UDP\_Receive$
  if $HeatSig == 1$ **then**
    $time = H^{-1}(T) + t_{step}$
    $T = H(time)$
  else
    $time = C^{-1}(T) + t_{step}$
    $T = C(time)$
  end if
  Store $(time, T)$ and $(time, HeatSig)$
end while

The model that was placed on the Raspberry Pi can be seen in figure 4. The heating signal and

temperature time histories are both stored in output files while a UDP-send block was used to send a packet to the Galileo as acknowledgement of receiving the input signal. The memory block was used to avoid causality issues with the temperature calculation. Without it, the feedback loop has no definite "starting point" and the output does not occur after the input.

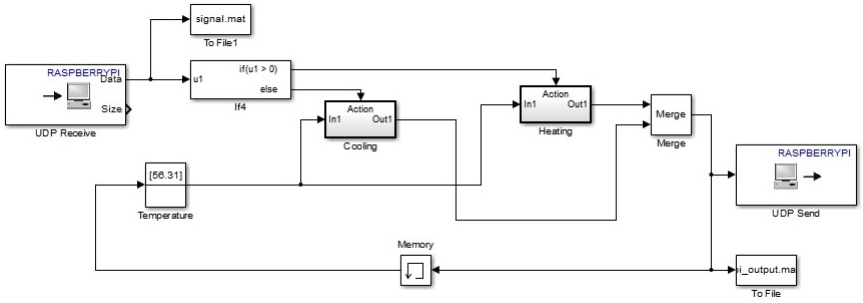


Figure 4: The Simple Raspberry Pi Simulink Model

After the simple model was finished, work focused on augmenting the model to include the effects of the thermal mass of objects in the room. Based on correspondence with Ed Bosco, it was taken (awk wording) that the time to heat the room from a cold state would increase to 60 minutes if the room had been cold for 3 hours or more. Though the actual thermal mass calculations are extremely complicated in practice, implementing such calculations in a generalized model of an office is impractical.

The key features that would be used in augmenting our original model would be the time spent near the high/low temperatures (where the objects would gather the most thermal inertia). In order to relate this to the speed at which a room heated or cooled a scale factor was introduced to the time constant of the exponential curves modeling both the heating and cooling of the room (in the simple model this could be taken as having a value of 1). As mentioned previously, the original curves were implemented in the form of $T(t) = A + B * e^{Ct}$, or in time constant form: $T(t) = A + B * e^{-t/\lambda_0}$ where $C = -1/\lambda_0$. The scale factor was introduced so that $\lambda = \lambda_0/scale$. This way a change in the scale factor would directly affect the thermal behavior of the system.

The next step was determining how our scale factor would be calculated from the time spent at high and low steady states. Since we know that the scale factor cannot grow or decrease without bound then a logistic function is ideal for replicating this behavior. The minimum bound for the scale factor is 1, as without no accumulated thermal inertia the heating and cooling curves behave as measured. The cap was estimated to be a factor of 3 (based on correspondence with Ed Bosco) and the peak rate of change of the scale factor was approximated to occur at approximately 1.5 hours.

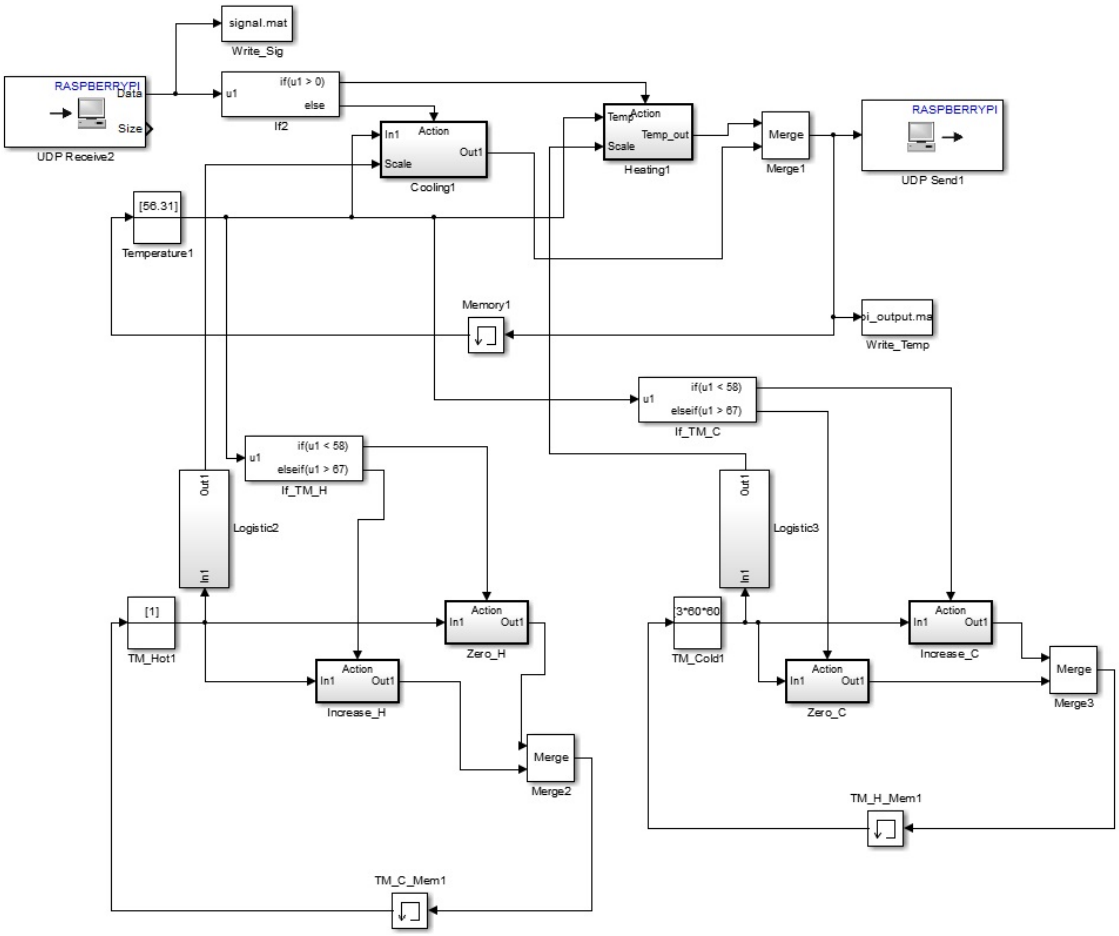


Figure 5: The Raspberry Pi Simulink Model with Thermal Mass Scaling Conditionals

4.8.1 Moving Forward

Improving the accuracy of the temperature computed by the model is the primary need. Due to the nature of the smart heating system, the heating signal to the Raspberry Pi can change at a moment's notice, so the control logic mandated by this adds a layer of complexity to the heating calculations. As our mechanical analysis (utilizing FLUENT, hand calculations and other techniques) advance, so will the Simulink model. Additionally, other methods of communicating with the Intel Galileo must be explored, such as using a directly wired serial connection.

4.9 Mechanical Analysis

The mechanical analysis began with an experimental test in which we opened a classroom on the third floor (Room 338) of Upson Hall and waited for the air temperature to reach 55 degrees Fahrenheit. We took measurements of the room and the radiator (See Appendix G Figure 21) and noted that the outdoor air temperature was 39 degrees Fahrenheit. The initial temperature of the room was just above 70 degrees to begin and because the office was vacant, the room had been in this state for multiple days, if not months. Thus, we can assume that the thermal mass in the room was also completely uniform at a temperature of about 70 degrees. We opened the window to the room and waited for our thermometer (stationed at the center of the room, suspended from a chair) to reach below 55 degrees. Once this occurred, it was safe to assume that the air temperature was, overall, no higher than 55 degrees. Then we closed the windows and waited for the thermometer to hit 55 degrees exactly, at which point we began our timer. This experiment yielded a recovery time of approximately 12 minutes. Thus, with the thermal mass already warm, the room can recover as quickly as 12 minutes with a radiator solution.

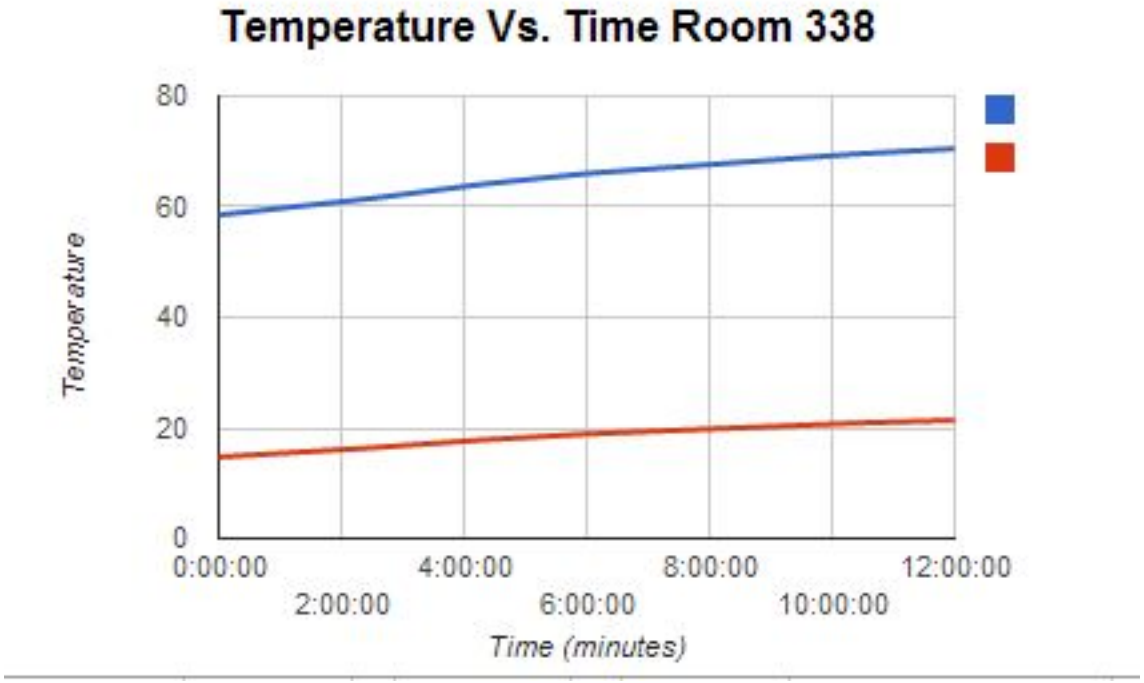


Figure 6: Real Room Temperature Distribution from Experiment

Because we do not have the authority to leave the windows open and allow the thermal mass to completely cool before repeating this analysis, we must utilize simulations and hand calculations to determine room recovery time when the thermal mass is acting as a heat sink instead of a heat source.

Similarly, because Upson hall currently only has radiators, we must use simulation to determine the effect of a convector unit on the recovery time in the space.

The following calculations and simulations (Section 4.11.1) increase in complexity as we continue to make our mathematical model match a realistic case.

Simplified All-Air

In this model, we examine a room full of pure air that changes in temperature as a lumped system from 55 degrees Fahrenheit (12.11 degrees Celsius, 285 degrees Kelvin) to 70 degrees Fahrenheit (21.77 degrees Celsius, 294.26 degrees Kelvin). The governing equation of this type of heat transfer is

$$Q = mC_p\Delta T_{room} \tag{9}$$

Where Q is the heat in Joules (J), m is the mass of the air in kg, C_p is the heat capacity of air in J/kg*K, and ΔT_{room} is the change in temperature of the air in the room in K. Since the volume (V) of the room (m^3) and the density (ρ) of air is known (kg/m^3), we can substitute the multiplication of these quanities for m as follows:

$$Q = \rho VC_p\Delta T_{room} \tag{10}$$

and calculate the heat required in Watts to raise the room from 55 degrees Fahrenheit to 70 degrees Fahrenheit directly. However, we are looking for the amount of time it takes to raise the room temperature. Thus, we must include the factors that contribute to the room's temperature, such as the loss of heat through the windows and the gain of heat from the radiator or other mechanical equipment. These values are rates of heat exchange, measured in Watts or Joules/sec. This means we can rewrite our equation as:

$$\dot{Q} = \rho VC_p\frac{\Delta T_{room}}{\Delta t} = \dot{Q}_{gains} - \dot{Q}_{losses} \tag{11}$$

Thus, by using the heat values in Watts on the right hand side of the equation, we enable ourselves to solve for Δt or the recovery time of the room:

$$\Delta t = \frac{\rho V C_p \Delta T_{room}}{\dot{Q}_{gains} - \dot{Q}_{losses}} \tag{12}$$

The heat gains and losses were determined as follows:

The heat gains were found using a product data sheet for a hydronic radiator by Sterling (Full data sheet can be foundin Appendix G Figure 22 through 24). In this product data sheet, the values for heat transfer were determined by selecting a model and determining the average hot water temperature at which the building operates.

Cover	Tube Size	Catalog Designation	Fin Size (inches)	Fin Per Ft.	Fin Thickness	Enclosure Depth & Height	Tiers & Centers	** Mtg.* Height	Steam 215° Factor 1.00	Average Hot Water Temperature °F					
										200°	190°	180°	170°	160°	150°
										Factor					
										0.86	0.78	0.69	0.61	0.53	0.45
SB-14	1 1/4" I.P.S.	ST-144	4 1/4 SQ.	40	.032" STL	14B	1	18	1500	1290	1170	1040	920	800	680
	2" I.P.S.	ST-243	4 1/4 SQ.	32	.032" STL	14B	1	18	1340	1150	1040	930	810	710	600
	3/4" CU.	STC-3/4 435	4 1/4 x 3 5/8	50	.020" AL	14B	1	18	1840	1580	1440	1270	1120	980	830
	1" CU.	STC-435	4 1/4 x 3 5/8	50	.020" AL	14B	1	18	1930	1660	1510	1330	1180	1020	870
	1 1/4" CU.	STC-1435	4 1/4 x 3 5/8	50	.020" AL	14B	1	18	1860	1600	1450	1280	1130	990	840
SB-20	1 1/4" I.P.S.	ST-144	4 1/4 SQ.	40	.032" STL	20B	1	24	1600	1375	1250	1100	975	850	720
	2" I.P.S.	ST-243	4 1/4 SQ.	32	.032" STL	20B	1	24	1390	1195	1085	960	845	735	625
	3/4" CU.	STC-3/4 435	4 1/4 x 3 5/8	50	.020" AL	20B	1	24	2090	1800	1630	1440	1270	1110	940
	1" CU.	STC-435	4 1/4 x 3 5/8	50	.020" AL	20B	1	24	2180	1870	1700	1500	1330	1160	980
	1 1/4" CU.	STC-1435	4 1/4 x 3 5/8	50	.020" AL	20B	1	24	2130	1830	1660	1470	1300	1130	960
SB-20	1 1/4" I.P.S.	ST-144	4 1/4 SQ.	40	.032" STL	20B	2	24	2250	1935	1755	1550	1370	1190	1010
	2" I.P.S.	ST-243	4 1/4 SQ.	32	.032" STL	20B	2	24	2000	1720	1560	1380	1220	1060	900
	3/4" CU.	STC-3/4 435	4 1/4 x 3 5/8	50	.020" AL	20B	2	24	2820	2430	2200	1950	1720	1490	1270
	1" CU.	STC-435	4 1/4 x 3 5/8	50	.020" AL	20B	2	24	2640	2270	2060	1820	1610	1400	1190
	1 1/4" CU.	STC-1435	4 1/4 x 3 5/8	50	.020" AL	20B	2	24	2510	2160	1960	1730	1530	1330	1130

• The ratings above include factors shown below for recommended mounting height. Two tier ratings are based on 6" spacing between elements. Ratings are in BTU per hour lineal foot of active length. Active length is catalog ordering length less 5" on steel, less 4" for copper. • Water ratings applicable to water flow rates of three or more feet per second have been determined by applying factors to steam ratings. • Steel elements are painted black. • Copper, unpainted. If the unit is to be installed at a different height than that recommended, the rating must be adjusted as follows:

Figure 7: Excerpt of Product Data Sheet for Sterling Radiator

The selection of the model was fairly arbitrary, because their heat transfer rates were all on the same order of magnitude and the purpose of this calculation is for preliminary results. However, the 1" CL tube size was chosen because that is the size of the pipes in Upson Hall, and the pipes carry hot water, not low pressure steam throughout the building. According to Matt Steel, the facilities coordinator of Upson Hall, hot water enters and leaves the units at a maximum of 180 degrees Fahrenheit and a minimum of 140 degrees Fahrenheit, based on outdoor air temperature. The units are provided in English units due to the fact that the data sheet was for consumers in the United States. Thus it was important to consider the issue of converting from English to metric units for our calculations. The values provided in the table are in units of BTU/hr*linear feet of the radiator, and must be multiplied by the radiator length (15.5 ft) before converting from BTU/hr to Watts at a conversion factor of 0.293 Watts per BTU/hr. After this conversion, we find that the unit provides 6,036 Watts to the room.

For the heat losses, we only considered heat transferred through the windows. This was due to the advice of Ed Bosco, PE who is leading the project as the head of the MEP firm performing the HVAC and electrical design for the upcoming Upson Hall renovation. He suggested that because the building is being renovated with brand new equipment and high R-value walls, we should only consider transmission losses through the windows as the main source of heat loss in the room. Similarly, despite the increased ventilation, preheat will be utilized from recycled ventilation air in the mechanical system such that losses due to ventilation will not be significant. The equation for transmission losses was described earlier in Equations (1) and (2). The ΔT used in these equations is the outdoor air temperature versus the indoor air temperature, which we will call ΔT_{window} so as not to be confused by the ΔT_{room} used in Equation 9 which represents the change in room temperature. Thus $\Delta T_{room} = (294.26K - 285.93K) = 8.33K$ and $\Delta T_{window} = (285.93K - 277.09) = 8.84K$. The chosen outdoor air temperature of 277.09 Kelvin came from the initial room experiment performed in Upson.

Thus if we plug in these numbers to Equation (12) and solve for our time to recover, we find Δt = 1.74 minutes.

Obviously, this result is unrealistic. We found the recovery time in our experiment to be about 12 times this value. Thus, the simplification to treat the air as one large body is not accurate enough. Next, we can take into account the thermal mass of the objects in the room.

Thermal mass takes in to account the amount of heat storage capacity of the objects in a system. That is, solid bodies can retain heat and reject it to the surrounding atmosphere at a slower rate than a fluid body such as air. Conceptually, this is similar to the way a capacitor stores charge and releases it later. This is the idea that it will take a longer amount of time for a desk to change temperature (whether it is increasing or decreasing) than it does for air. Thus, if the thermal mass of the objects in the room are at a temperature lower than the air temperature, they will act as heat sinks and delay the overall recovery time of the room, and if the thermal mass of the objects in the room is higher than the air, the objects will act as a heat source, and accelerate the time to recover the room. The initial experiment we performed utilized the latter condition. The thermal mass was already warm and contributing as heat sources to the room's recovery time.

Thermal Mass

Some hand calculations can be performed to determine the recovery time of a room including thermal mass. In order to determine the amount of energy it will require to raise the temperature of the thermal mass of an object in a room we can use the same formula as (9) with the values of $\rho\, C_p$ and V adjusted to the values for the mass. Using brick for the walls, and wood for the desk, chairs, table, and a bookshelf, we utilize the values found in the calculation page in Appendix G Figure 25. This gives us a total energy requirement of 84,809.84 kJ. Since this is also in energy, it contributes to the left hand side of the equation in Equation (11). Solving Equation (11) for Δt once again but including thermal mass, we get 355 minutes to recovery! This is a very long time, but it makes sense. The equation we are solving is determining the amount of time it takes for the full body of thermal mass to recover to 70 degrees Fahrenheit. This means that the entire volume of the walls and bodies must be at 70 degrees as well. This long time is not necessarily the time it takes for the room to feel comfortable again. We can assume that once the surfaces of these objects no longer radiate heat (or take in heat) as quickly as they did at their initial temperature and the air in the room is warm, we have reached convergence.

Thus we must now consider not just the amount of time it takes to raise the thermal mass and air to a comfortable temperature, but the coupled solution, in which the heat from the thermal mass contributes to the air temperature, and the air temperature governs the conduction in the thermal mass bodies, as this is the solution that will truly tell us the time it takes to make a person comfortable in the room. For the purpose of this analysis, we must utilize a more robust tool to solve the complex heat equation in 3-D. Thus, we must turn to ANSYS FLUENT as a method of generating this result.

4.9.1 FLUENT Analysis

Part A - Simulation Procedure

The actual FLUENT models performed will be saved to the Google Drive folder for this project so that another student can open them and see my settings. Instead of posting screenshots of every step, I will instead describe some of the challenges we faced using FLUENT and important notes for how to set up the geometry and mesh, and some tips for future modelling.

One issue with the FLUENT model was the fact that the 3-D analysis is complex and computationally expensive. We attempted to model the problem in 2-D, which, while we were able to replicate the results of the experiment, did not do a good job of taking into account the thermal mass in the room. Because we had objects distributed around the room, we had to utilize a 3-D model.

The geometry was created in Creo (PRO/E) software. The dimensions for the desk, chairs, and bookshelf all came from the staples Website listed on the thermal mass page in Appendix G Figure 25. Using these dimensions, individual table, chairs, bookshelf, and desk were assembled inside a "shell" of the actual room. The room was CADed based on the dimensions we measured in our original experiment, but with an additional 6 inches in every direction. This is so that after extruding the shape we can create a shell with the inside dimensions of the actual room, where we would be assembling the furniture. We needed to use a shell to give space for the objects to be assembled in the room. See Figure 8 for a cross sectional view of the geometry in Creo. It is important to note that the shell geometry tool or some other single type of part should be used to minimize the number of interfaces you need to deal with when

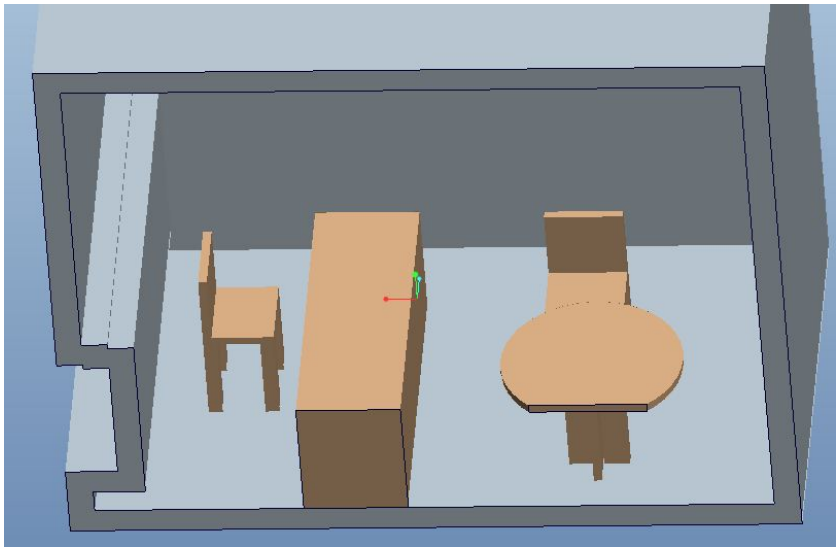


Figure 8: Geometry Cross-Section in Creo

the geometry is brought into FLUENT. A simple one-piece geometry for all of the walls is necessary or it will be too complex. You can always select individual wall faces and give them different conditions in the future. A key aspect of the geometry of the system happens after the geometry is brought into workbench. Once the geometry is linked to workbench, we must create the fluid body. In order to do this, create a new coordinate system, and choose the option to create it from a geometry selection, then choose the face of the wall that might represent a 2-D model - That is, one that shows the outline of the radiator well. Once you create this plane, create a new sketch on it, and trace the wall. Next extrude this body "To surface" and choose the wall on the other side of the room. Change from Add Material to Add Frozen, as this creates a new body instead of adding material to the existing ones. Name this body 'fluid' and make sure that you give it the condition of being a fluid in the properties section when you highlight it in the tree. We can now suppress (not just hide!) the room shell, as we no longer need it. Next we need to specify that the fluid is to take up the space between the objects. Click Create ¿¿ Boolean and make it such that the Boolean is subtracting material. The target body should be the fluid body, and the tool bodies are the objects in the room. Be sure to set retain tool bodies to yes, as we want to make sure the objects are still there. Finally, select the the fluid body and the objects in the room, right click in the tree, and make them all one "Part."

Next is the mesh. The mesh for the system was primarily made up of body meshing, because the geometry is 3-D. An important tip for meshing is to look at the topology of the mesh in 3-D, because a body sizing mesh may look strange on a 2-D face (in cross-sectional view for example) but when you look at the topology you will realize the true nature of the mesh. See Figure 30 in Appendix G as an example. I also used the MultiZone condition on the faces of the desk, table and bookshelf. For some reason, MultiZone did not want to work on the surface of the chair, so the chair was left with normal body sizing instead because the mesh still looked fairly reasonable. Another important aspect in the mesher is to create the proper named selections. Create a named selection for the walls you need to control for temperature by selecting the corresponding face on the fluid body. If multiple walls will have the same properties, it is recommended to call them one named selection, as this will save you time in FLUENT setup. Select the object BODIES (NOT FACES) and label them whatever works - ours are named "tablebody," "deskbody," and so on. Do not label the surfaces on these geometries - make sure you are using body selection. FLUENT will generate its own surfaces and we will need FLUENT to do this to create the right model. Finally, select the fluid body and name it fluid. Once again make sure it is recognized as a fluid and not as a solid.

For the FLUENT setup, we chose the surface to surface radiation model as it suits our needs the best. FLUENT writes the view factor file for us so we do not need to do any radiation view factor calculations of our own by hand. This method was used in the FLUENT tutorial for natural convection, and we felt it would also suit the needs of our project as well. An area for further research would be to investigate the other radiation models and their ability to solve our case. The boundary conditions at the wall were a major challenge for this project. Because we did not create named selections for the

walls and allowed FLUENT to do it for us, we have "shadow" walls generated. These shadow walls are an integral part of the FLUENT model. If the shadow walls do not appear in the boundary conditions list, then the model will not be right. By letting the walls and their corresponding shadow walls have their thermal properties chosen to be "Coupled," we allow the heat transfer to occur via convection on one side of the wall, and allow that heat to be transferred to the solid body via conduction through the inside surface of the wall. The boundary conditions that yielded accurate results were to allow the radiator to have a set temperature equal to the temperature of the hot water through the pipes which was adjusted to 355.37 degrees K. The window, which is right above the radiator in our model is also set to a fixed temperature boundary condition at 39 Fahrenheit, or 277.04 K, the temperature of the outside air when we performed the intial experiment. In order to monitor the temperature of the room air, we keep track of the volume-weighted average of the room's static temperature using the volume monitor tool. It is usually helpful to plot this curve over time to see how the room is reacting. It is also helpful to use a volume-weighted average of static temperature within the objects, which allow us to monitor the temperature of the thermal mass.

Some important aspects of the FLUENT model that would be good for future research and investigation include the FLUENT setting for double precision. I used single precision with 4 parallel processes for speed of calculation. Because we are working in a 3-D transient model, it would make the most sense to choose the least computationally expensive method at first to get worthwile results. In the future, I would recommend utilizing double precision and evaluating an comparison of the results to the results generated in this project. Similarly, the Solution Methods utilized for this project were highly simplified. For example, we typically solve Momentum and Energy equations using Second Order Upwind for the most accurate result, but because the first order solution was faster and more stable solution, I chose only to look at the First order Solution. Further investigation into this would make the analysis even more solid. Finally, refining the mesh and lowering the residuals (and subsequently raising the number of iterations per time step) will allow for more accurate results. I recommend that in the future, a more refined mesh is used to verify the results here, as well as increasing the iterations per time step to confirm convergence.

Part B - Simulation Results

The results of the FLUENT simulation can be broken down into 5 key simulations:

First, the 3-D FLUENT case which was modelled from our initial experiment. The significance of this case was to determine the corrent boundary and room conditions that simulated the real-life data we took. By ensuring this model and our experimental data matched, we could rely on similar models with controlled adjustments to produce accurate results. By running the model with 39 F outdoor air temperature at the window, thermal mass already at a warm temperature of 70 F, and 380 K temperature on the radiator, we got a convergence time from 55 degrees to 70 degrees Fahrenheit of 12 minutes. The plot of the temperature of the room versus time step can be seen in Figure 9. The fact that this result agreed with our experimental results was key to ensuring our other models were accurate.

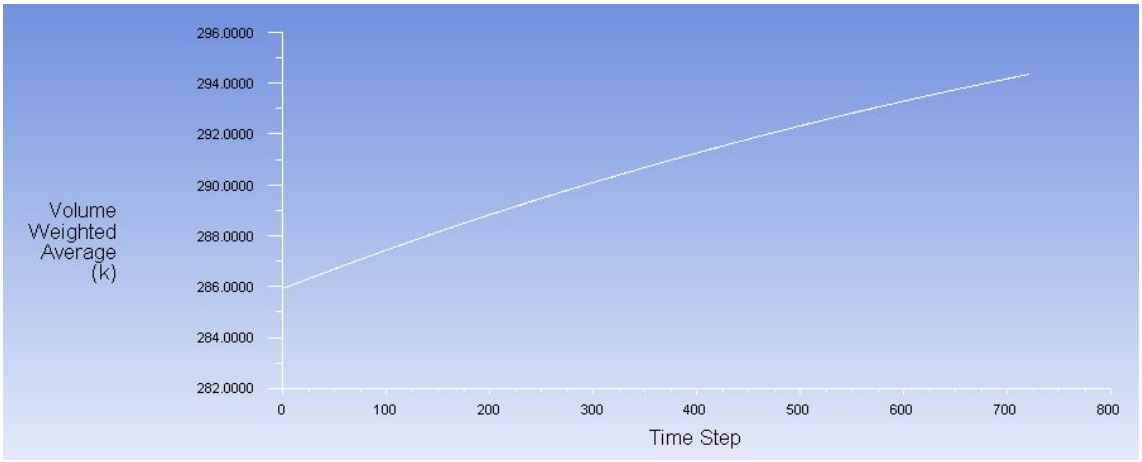


Figure 9: Temperature Vs. Time with Radiator and Warm Thermal Mass (Experimental Case)

The next important simulation was the 3-D FLUENT case in which the thermal mass begins cold instead of beginning warm like in the experimental case. In this case, the fact that the thermal mass stays

cold and lags behind the air temperature causes the air temperature to reach 70 degrees slower. The room is still being heated by the radiator at 380 K. We can see from these results that the temperature reaches 70 degrees in 40 minutes, significantly longer than the previous case:

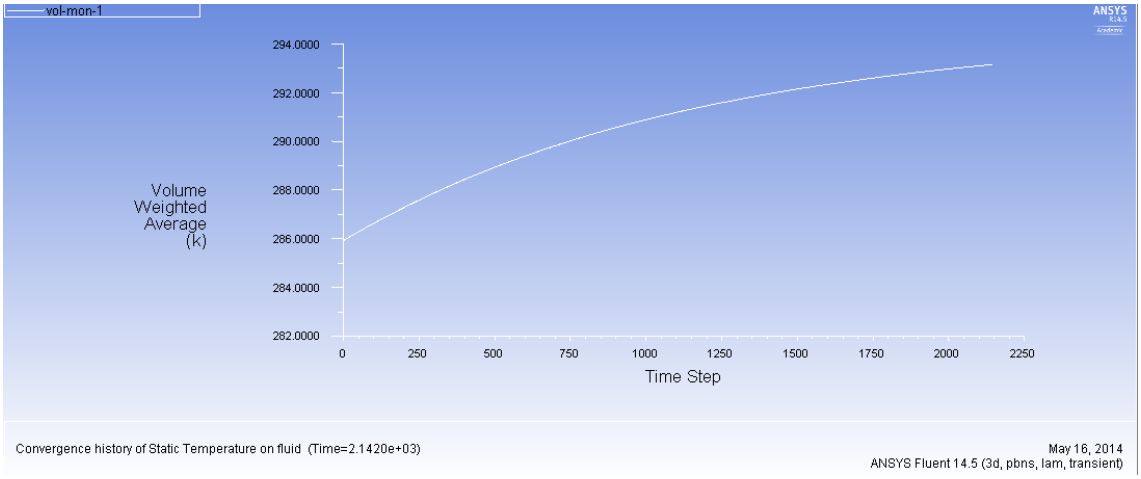


Figure 10: Temperature Vs. Time with Radiator and Cold Thermal Mass

The next set of results we were interested in is these same two cases where thermal mass begins hot and thermal mass begins cold but using a convection unit instead of a simple radiator. We want to understand the impact that a convection unit can have in improving the recovery time of both the air and the bodies in the room. Thus, we performed the same analyses with a convection unit that rejects air at a velocity of 0.1 m/s and 300 K (approximately 80 degrees F), and the same temperature as the radiator on its surfaces at 380K.

For the situation where the thermal mass begins cold with a convection unit, we found the recovery time to be approximately 11 minutes. The boundary condtions for this scenario included coupled walls for the surfaces, but an overall initial temperature of 285.93 K. The following Temperature versus Time graph in Figure (11) shows the distribution:

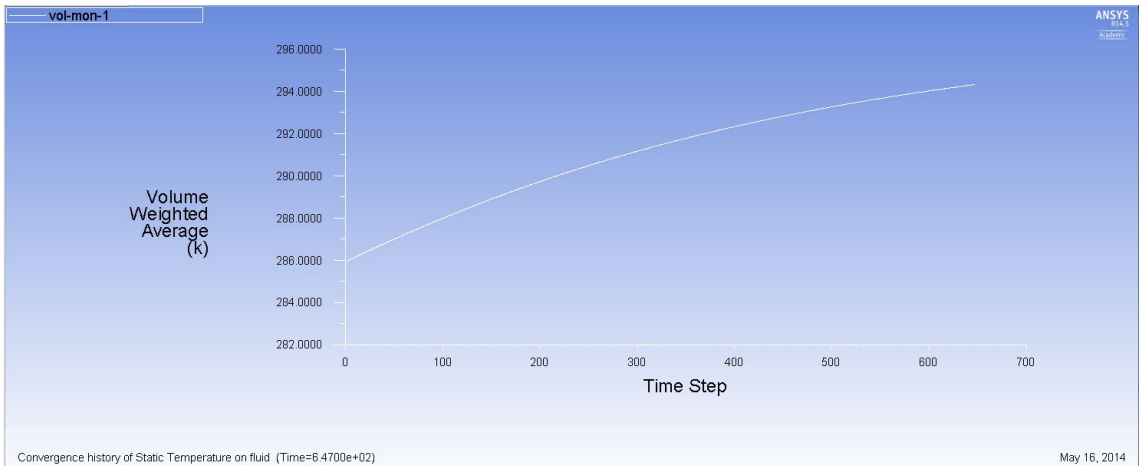


Figure 11: Temperature Vs. Time with Convection and Cold Thermal Mass

The case where the thermal mass began at a hot temperature changed from 55 degrees to 70 degrees Fahrenheit in 7.4 minutes. This quick change indicates that the convection unit can heat the room almost twice as fast as the radiator unit when the thermal mass begins hot. See Figure 12 for the time distributed temperature data.

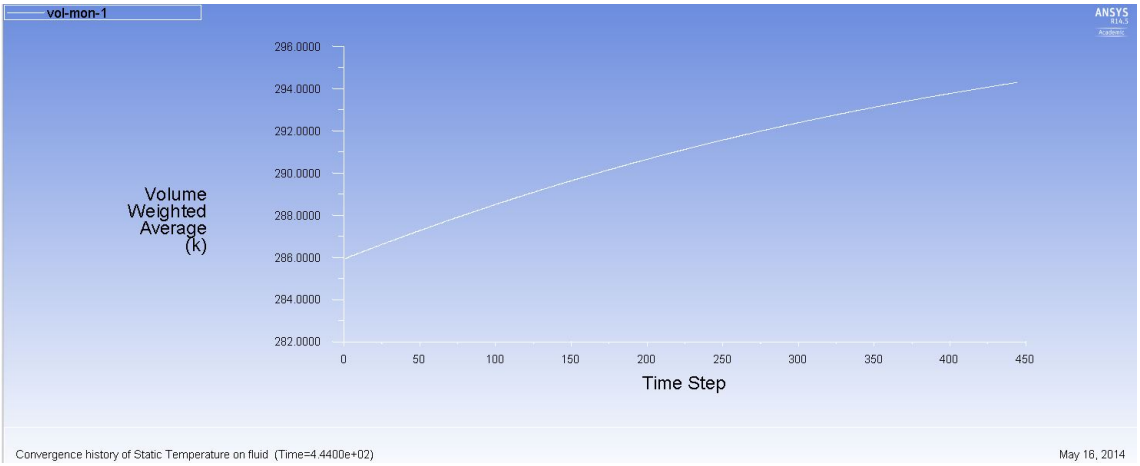


Figure 12: Temperature Vs. Time with Convection and Warm Thermal Mass

5 Results

The goal of the results of the project are to display graphically the results from our various sensors as well as the output from the code and temperature model. Essentially, the results of the project should be 3 graphs overlayed with one another. The first graph should be the square wave of occupancy data that was taken from the PIR sensor versus time. This is a simple 1 or 0 square wave output that indicates if the professor is there or not. The next graph should be the square wave of what the galileo would be telling the heat to do based on the sensor inputs and our code framework at the same time points as the PIR data. Finally, we would use the temperature model to overlay on top of these graphs, to show how the temperature model reacts to the signals from the galileo. An example of this output is shown in Figure 13:

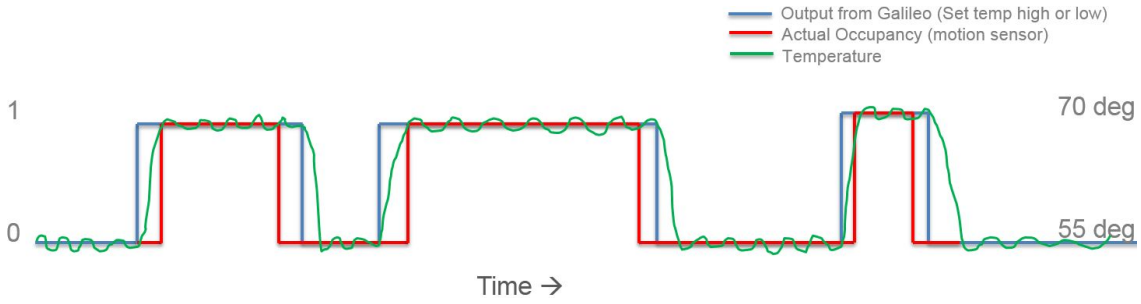


Figure 13: Model of Theoretical Results

The above model does not reflect the actual results of the project. The code framework and hardware has only been fully operational for 2 days at the time this report was written. While we have some output data from one day, it is evident that there are going to be errors, as we will not have any prediction occurring since we are still in the first week. More results will be obtained as time progresses, and we aim to complete a model of our actual results in the same way these results are displayed in the near future. This allows us to calculate actual energy savings over the course of multiple weeks, as well as track the occupant's comfort. That is, we can notice on the graph the incidents where the temperature had not yet reached 70 degrees when the occupant arrived, and track both savings as well as occupant comfort level for a final result.

The results of the FLUENT models indicate that a convection unit is far quicker at heating the room, and this quick recovery time can be taken advantage of in a system with intelligent sensing. In fact, if we know we can heat the room comfortably within 11 minutes when it began very cold, some professors may be willing to eliminate the predictive portion altogether and only use motion sensing. The professor would have to be willing to tolerate bout 11 minutes of an uncomfortable temperature in order to save additional energy, but for some this may be worth it. The utilization of convection units is highly recommended in conjuction with prediction to reduce the amount of occupant discomfort and the improve the response of the thermal system to changes in occupancy.

The results of the occupancy prediction show that the error is only 2% in the first week of prediction, and goes much lower to 0.2% for most other weeks.

Overall, the potential energy savings of our project depend on the number of offices in the building. Obviously, the more offices in a building, the more potential this project has for creating energy savings. When compared to the current building's system, our model could save 175,096 kWh of heating at an estimated cost savings of $18,700.

6 Conclusion

Overall, the project was a major success. Over the course of one semester this team was able to create a complex mechanical FLUENT model of a 3-D system, generate a simulated plant model for room temperature that takes into account the thermal mass of a system and change states quickly and efficiently, as well as develop a sensor and code framework that is able to intelligently sense and predict when to heat the space, all on a newly-developed microcontroller by Intel.

Moving forward, there are many directions to take this project. First, the project results should be developed further to determine actual energy savings from our prototype and system. In order to do

this, the interface with the server should be improved such that it is easy to pull data from the server and generate square wave data and drive the Raspberry Pi simulation. Because the full system was only up and running for 2 days, the data is not yet accurate. The system should be up and running for at least 3 weeks before pulling the data and using it to evaluate results. SImilarly, checking for bugs and issues with the result are necessary. Again, because of the short uptime of the system so far, we have not had the opportunity to find the shortcomings of our algorithm.

Another aspect to explore is the idea of making this a wearable technology. Our XBee transmitter prototype is already quite small. Thus, there is plenty of potential to make it even smaller and perhaps turn it into a wearable device. This goes hand in hand with the professor interface of the project. This should also be developed such that the professor can monitor their personal energy savings from utilizing the system. This will also add incentive to sacrifice small amounts of comfort in the name of saving energy. For example, if we allow the system's constraints to be smaller, that is, the professor would not mind being uncomfortable for under 5 minutes in their office, we can display how much more energy they would save in this instance. Creating a user interface of this nature would be very interesting to implement.

Finally, coming up with a way to create a mass-production scaled system would be helpful. Our current prototype is costly because we looked more toward developing a solid system than a wide-scale mass-production product. By looking for ways to save money by implementing the project on a large scale (like using cheap radio frequency receiver/transmitters instead of the developer-friendly XBee) would add an even more marketable dimension to the project.

7 Appendix

7.1 Appendix A

Galileo

7.2 Appendix B

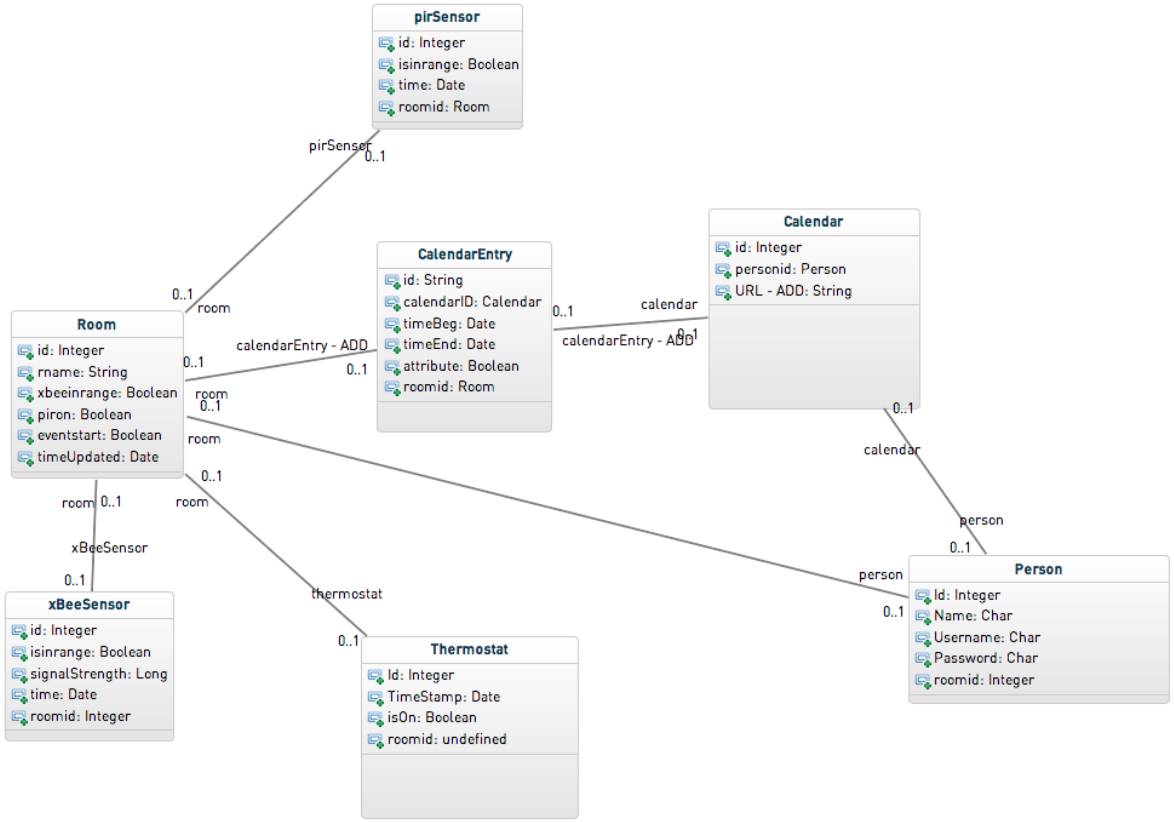


Figure 14: Database Schema

Instructions for starting the system on Amazon Web Services
Part A - Turn on the Database

1. Go to *https* : *//aws.amazon.com/* -¿ MyAccount-¿AWS Management Console
2. Login with username: upsontr@gmail.com and pwd: galileogalileo
3. Go to RDS
4. Select snapshots
5. Choose “maindb-final-snapshot”, or most current snapshot of db, and hit restore snapshot
6. For DB Instance Identifier enter “maindb”
7. Select db.t1.micro for instance type
8. Select No for Multi-AZ Deployment
9. Hit Launch DB Instance
10. DB will take a few minutes to start up
11. Once it’s started select the db and click modify

12. Set security group to debug-security

Part B - Turn on the Server

1. Go to *https* : *//aws.amazon.com/* -¿ MyAccount-¿AWS Management Console
2. Login with username: upsontr@gmail.com and pwd: galileogalileo
3. Go to EC2
4. Under Images on the left side bar, click AMIs. This brings up all saved Server Images.
5. Highlight MainServer and click launch (blue button at top)
6. Click next, and select the number of servers to launch.
7. Click next, and next again, and give it a name (this helps)
8. Click next, and select existing security group. Select Debug Security (IMPORTANT)
9. Click Review security, and launch the instance(s)
10. If you have multiple servers and would like to route requests to all of them (to increase load capacity) then you need to set up a load balancer (step 12). Otherwise go to step 11.
11. Go to the EC2 Home page. Click on the server you'd like to route requests to, and copy the public DNS in the information that is below. That is the website address of the server - simply copy paste the link into your browser to test the setup.
12. To set up a load balancer, simply navigate to load balancer on the left panel.
13. Once there click create Load balancer. Give it a name and click continue.
14. Change the ping path to "/" and click continue
15. Select an existing security group and click on debug security.
16. Add all the instances you want to be handled by the load balancer.
17. Review and Launch! It will take some ten minutes for all the instances to register and be available. Once they are, use the DNS name of the load balancer as the main website!

SSHing into the Server for Debugging

1. Get the public DNS of the server instance (not loadbalancer) you would like to access via terminal to upload or edit files.
2. Create a file with the following contents and save it to an accessible directory.
—BEGIN RSA PRIVATE KEY—
MIIEpAIBAAKCAQEAk/m9takjQpiAfuVtjoNzHsxjB9aGWGPxa/fGdGBd/VLRtsfPg3Fdf2LvpqZT
C/ZX7C3bxyuRipmjnQtpM7JPtR3XSHJP/xCUEOn0pf2y0q81NZF3KKJHgkX5+DAwrWhIq8R6FBhY
nKy9XoX+HXlg09417XvXuqN+/F9l8p2y9T31a36qFzm9kRxdEAo5g3v8sSjZpy97USYOlvBb70gL
A52aqlng8Kjcw4UV+769U7j1m8W6xzzofLBla3nQR6FJ+OIn1mah3panDlQn0KXXl2MXW4HJcFHW
xRg3raJrOrojVlkcc5X2ZOau2wZZFJ08gHXzKN4IKZviCERuWXJDXwIDAQABAoIBAQCE31xaIuZm
hknbqGgX8DEPFrhV8WvxMs02BedN2LqNHjDxz7mmbcVm/Y/0NKWduDZ6/TK8RYMK0ar4+aYdpZ+t
In6NrxfmERzjkzD7XDOWu30r/NLiydUsRWmQUrVjAKFAm8yC/tmS1gm1NtxX2wX3OTFzbiDolqcQ
Nxl0IebwDq8ZKENt8HSG8Yl3juZzSz66pTEL/sX8LLOpqU97qqBocGLJ8CWkbSLeMdrxThJ+jUqU
4fV0gpbb+DatRv2RXZuAdjLa0yVBHVuzzRl74lN9lsXSL8wZQFAD5umEcwn0IcJMmTc1tU05EnXn
6ODY4a52NmHQ24k63ncABUIdJ54BAoGBAOpifUG172Mfmn/RsYllJSsNhjzxUHWwm0AcrMqUKM+9
1tASAcF7wDItZKiWhntuj+Hc77liE074w9LROwxaRIsq6LpS/hDFnzfHNkTgv0YiHDIrWQ4Fc9wO
ALVjQO7DChTKFiPH6EgcO35yOPn6k9uowjkf3jROvd516pSLIy+ZAoGBAKGfPxdD+H1IRqFYaUMc
UDKj5YT4e2lSFoAlgCb7GG1xaggCeeHWjMIzwcwKslMaD0jGBSa+bZ86otn9yaVVybAOoM8F0Ycg
MaL84bEP77Bcu2VbovZmxrEOjX9RWyjGAoIqa1MW87BPNnggALNQUOX1eJlKTGqoKaoRmgP5j0W3
AoGBALugkiK7MOJJ11VBGP0i4ekgD3jTuJQuB2oASnZ0umLq1n6EG7G2jfndCrpGnWtL65RkV+cw
UsKiM6ic48cUr9A0Tk9xxn7IpJdzOsdSnuvWvHmQ595okQH0l66cwMshiP1xdN251wF60El9yIyc
giaSZsBx+WAoNKhT/rBg3Y7pAoGAMtG+5dLijIowY0VIe+ioMvQkFEaJj5lXJ94iyJ8FlEwoGfKl

XmS3B818b2shnU3BZpGVRxRzpbCrSj4prIooC+rdl5rtwj5WPTikwqcPJ/ZQmlNRD4dLjJrV1SXA
szZnzYQHES5TW6ncp7LwZXGorlYcIJ8jHhpYV3iHZB6RdUsCgYAPpCL3StBFMUnwFWQ5j6Bv1Z/D
MJ8J+DosgNQmbttnlHoBkzE4bR9AFuapF20O2Hw6pEWi/rZPsQ09kaf0PNdwGMWfroXNH5dz9r8m
fxBlTl72hOYdb+SDgV3AXg2NrgMWqq8PuJ62jnXv1skk76kUJyzpNhOwPaCE1xbobvFOCg==
—END RSA PRIVATE KEY—

3. Use the following command to ssh: ssh -i ¡PathToAboveFile¿ ubuntu@¡DNSofServer¿
4. Say yes if it asks to save the host. You should now be in the server!
5. Simply replace ssh with sftp for uploading and modifying files. Refer to sftp documentation to use!
6. These credentials can also be used for any FTP/SFTP software.

Configuring the System to fit Specific Configurations

1. The database needs to have certain rows added to configure it for use in a production system. First, determine the final configuration - namely how many rooms the system will be used in.
2. Connect to the database using PGAdminIII. (Download and install using http://www.pgadmin.org/download/)

   1 - Log on to the AWS Console using aforementioned instructions. Once there, Obtain the DNS of the database in the RDS menu.

   2 - Create a server in PGAdminIII using the DNS as the host, and the username as upsontr. No password. Port 5432.

   3 - Click OK and it should connect
3. Add a room using the following command, replacing roomname with the room's name:
   Insert into room values (nextval('roomseq'), RoomName, false, false, false, clock_timestamp(), false)
4. Also, for each person in the system, create a person using a similar insert command into the person table, an outlook calendar (with its url) into the calendar table. For each room, add a thermostat into the thermostat table. Note that each table has its own sequence with the name of the table and a seq added at the end. simply call nextval(tableseq) and it will handle primary keys.
5. Note that individual galileos are assigned roomids as their unique identifier

7.3 Appendix C

Arduino Code for PIR sensor Data Processing and Server Communication

```
#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();
  }
}
```

7.4 Appendix D

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

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

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial1.begin(9600);
  pinMode(13, OUTPUT);
```

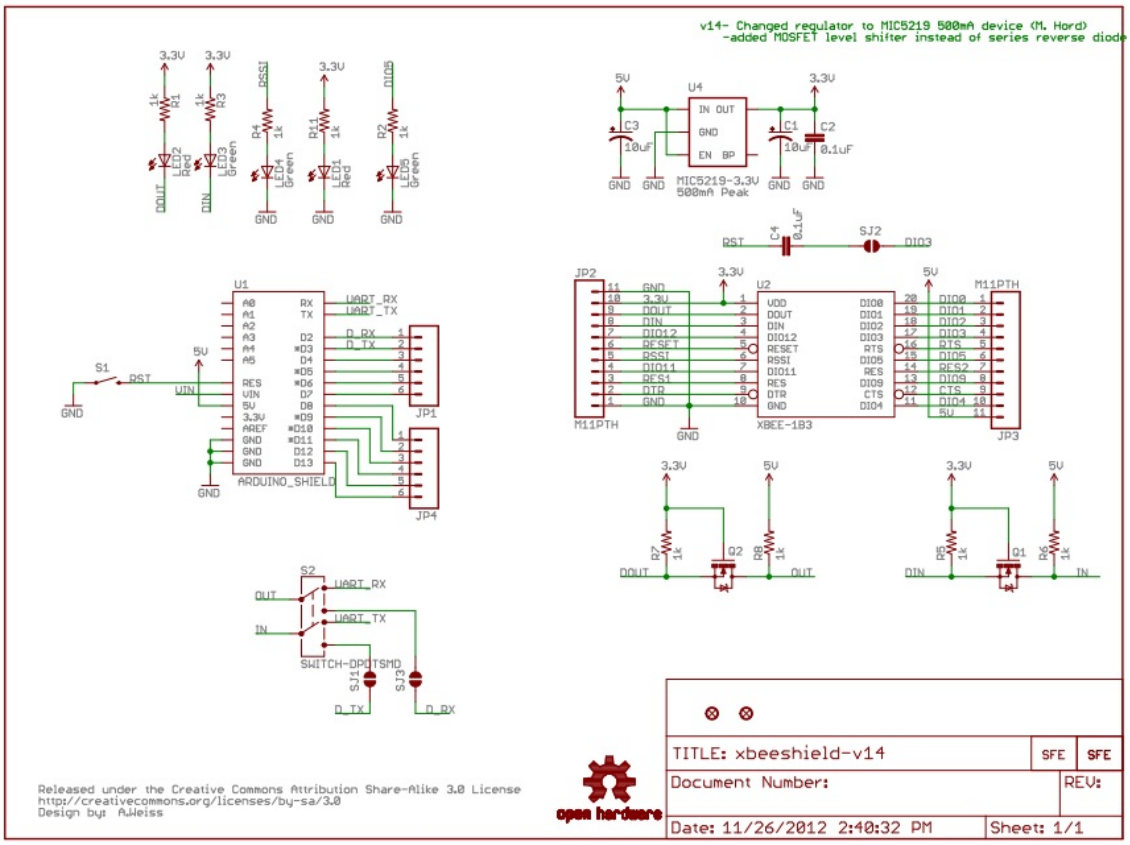


Figure 15: XBee Shield Schematic

```
  // 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() {
  // put your main code here, to run repeatedly:
  Serial1.write("Marco");
  if (Serial1.available() > 0) {
    while(Serial1.available() >0){
      Serial1.read();
    }
    Serial.println("XBee in range!");
    httpRequestXBee(); // Send the thing to the http
    for (int i = 0; i < 125; i++){
      Serial1.write("Marco"); // sends broadcast message to XBees in range
      if (Serial1.available() > 0) {
        while(Serial1.available() >0){
        Serial1.read();
        }
        Serial.println("XBee is still there");
        i = 0;
```

```
      }
      Serial.println(i);
      delay(2500); //wait 2.5 milisecond
    }
    Serial.println("XBee out of range!");
    httpRequestNoXBee();
  }
  delay(2500);
}

// Method for connecting to server and sending data corresponding to XBee in range
void httpRequestXBee() {
  // if there's a successful connection:
  if (client.connect(server, 80)) {
    // send the HTTP PUT request:
    client.println("GET /insertxBee?isinrange=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 XBee not in range
void httpRequestNoXBee() {
  // if there's a successful connection:
  if (client.connect(server, 80)) {
    // send the HTTP PUT request:
    client.println("GET /insertxBee?isinrange=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();
  }
}
```

7.5 Appendix E

Instructions for sharing Outlook calendar:

1. Go to the site: *http* : *//www.it.cornell.edu/services/owa*15/
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

```
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
```

Figure 16: MATLAB Code for Predictive Portion:

7.6 Appendix F

Raspberry Pi and Simulink Components

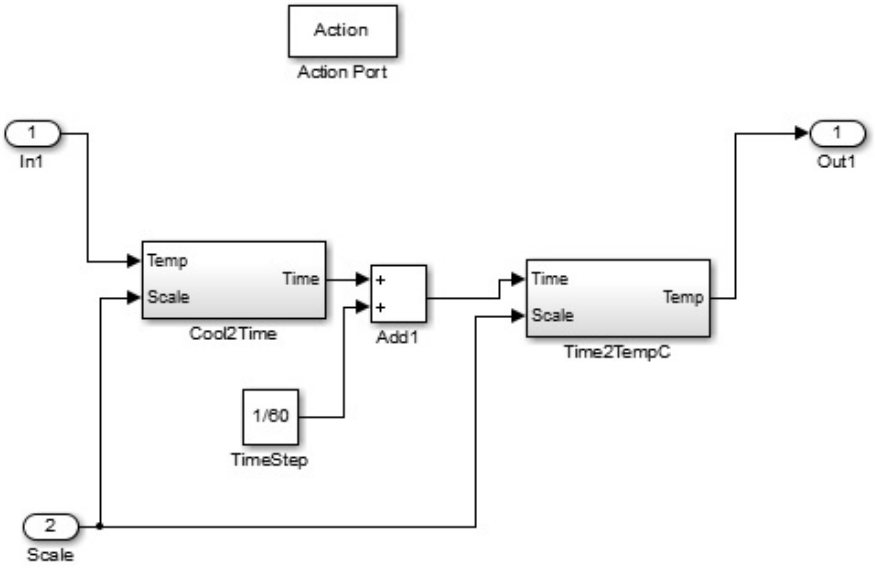


Figure 17: The Cooling Module of the Simulink Model, with a single second time step

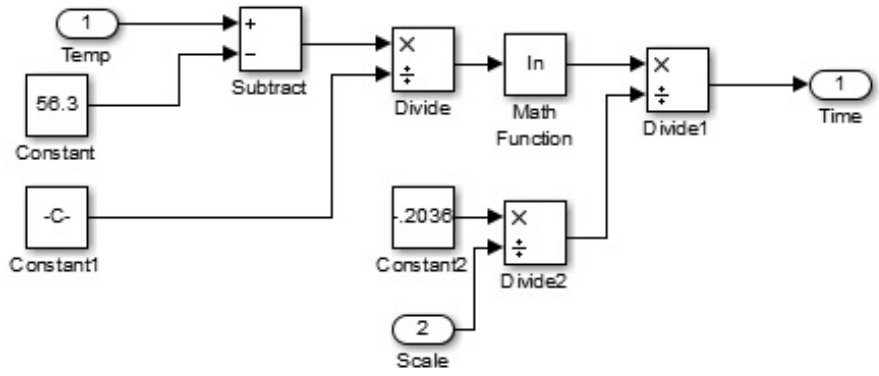


Figure 18: The Time Finder for the Cooling Curve

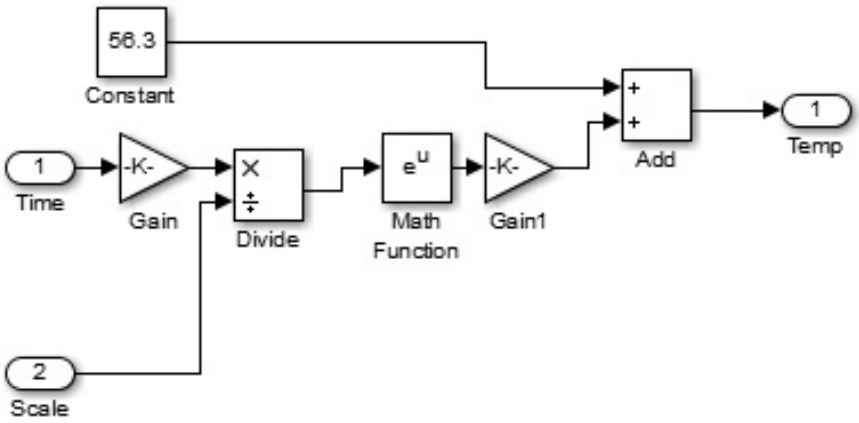


Figure 19: The Temperature Calculation

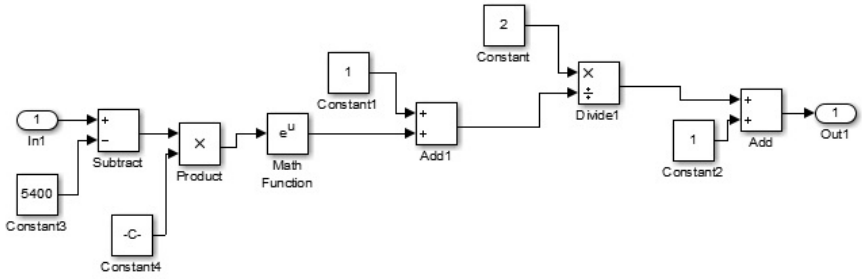


Figure 20: The Logistic Function used for Thermal Mass Scaling

7.7 Appendix G

rho	1.235		kg/m^3		
Cp	1006.5		J/kgK		
Room 1:					
Upson 338	English		SI		
Outdoor Air	39	F	3.89	C	
Room W	15.5	ft	4.72	m	
Room H	10.3	ft	3.14	m	
Room L	13	ft	3.96	m	
R-Value	8.00		1.41	m^	
Window W	15.5	ft	4.72	m	Area (%)
Window H	6.17	ft	1.88	m	59.87%
U-Value	1.1		6.24	W/	
Window on East Wall, Radiator on East Wall. Another smaller Radiator on west wall					
Radiator H	2.08	ft	0.64	m	East
Radiator W	15.5	ft	4.72	m	East
Radiator D	0.46	ft	0.14	m	East
Radiator H	2.08	ft	0.64	m	West
Radiator W	4.21	ft	1.28	m	West
Radiator D	0.77	ft	0.23	m	West

Figure 21: Measurements Taken During Experiment for Room 338

STERLING
HYDRONIC PRODUCTS

STERWYN

COMMERCIAL GRADE SLOPE TOP FINNED-TUBE ASSEMBLIES

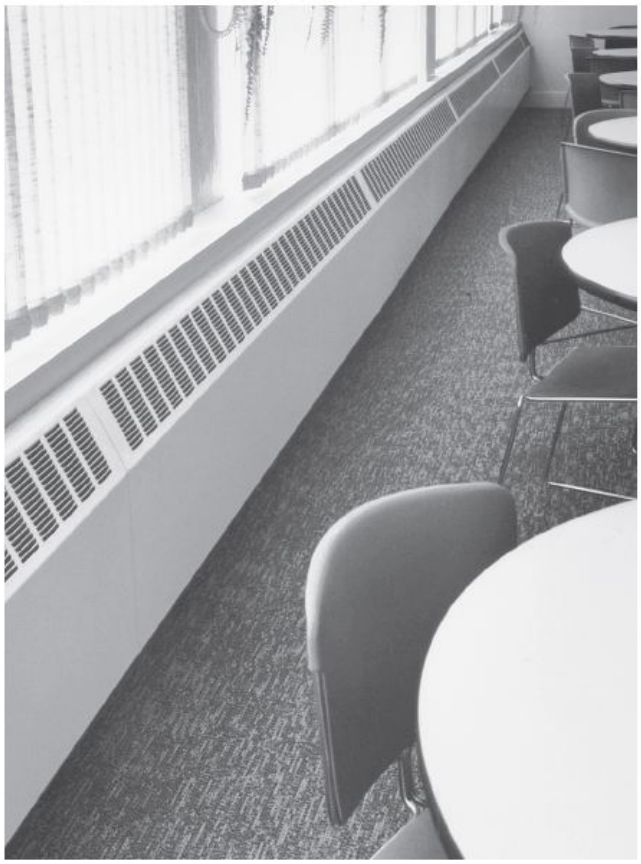
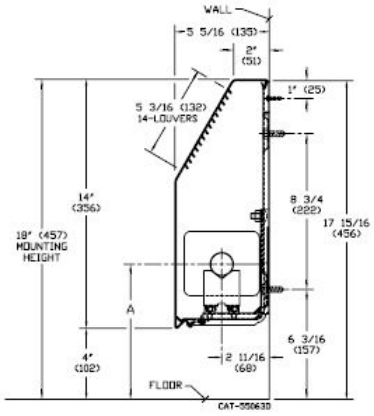
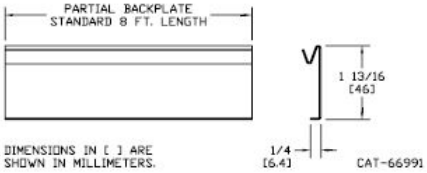


Figure 22: Sterling Product Data Sheet (Pg.1)

STERWYN DETAILS • DIMENSIONS



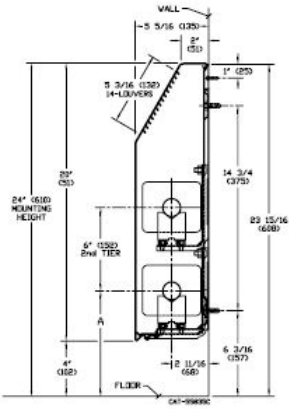
SB-14 – 16 GA.



DIMENSIONS IN [ ] ARE SHOWN IN MILLIMETERS.

JVB-S

ELEMENT TUBE SIZE	CRADLE	A MIN.	A MAX.
3/4" COPPER	2	7-3/8	8-3/4
1" COPPER	2	7-1/2	8-7/8
1-1/4" COPPER	2	7-5/8	9
1-1/4" STEEL	2	7-7/8	9-1/4
2" STEEL	1	7-5/8	9



SB-20 – 16 GA.

*Copper/Alum Elem — Stubs 2" Flared One End
*Steel Elem — Stubs 2½" W/Threaded ends (NPT)

GUARANTEED WORKING PRESSURES

2" IPS — 405 PSI AT TEMPERATURES UP TO 650°F.
1¼" IPS — 660 PSI AT TEMPERATURES UP TO 650°F.
1¼" CU — 194 PSI AT TEMPERATURES UP TO 300°F.
1" CU — 204 PSI AT TEMPERATURES UP TO 300°F.
¾" CU — 218 PSI AT TEMPERATURES UP TO 300°F.

- PARTIAL BACKPLATE AVAILABLE IN STANDARD 8' LENGTHS ONLY
- ELEMENTS FURNISHED IN THE FOLLOWING LENGTHS

ST-144, ST-243	3, 4, 5, 6, 8 ft. lengths only - Steel Elements
STC-1435	3, 4, 5, 6, 8 ft. lengths only - Copper/Aluminum Elements
STC-435	3, 4, 5, 6, 8 ft.
STC-¾ 435	3, 4, 5, 6, 8 ft.

- SLOPING TOP ENCLOSURE STANDARD GAUGE (PRIME FINISH) = 16 GAUGE
- ENCLOSURE FURNISHED IN LENGTHS OF 2' , 3' , 3' 6" , 4' , 5' , 6' , 7' , 8'

Figure 23: Sterling Product Data Sheet (Pg.2)

• RATINGS • ACCESSORIES

Cover	Tube Size	Catalog Designation	Fin Size (inches)	Fin Per Ft.	Fin Thickness	Enclosure Depth & Height	Tiers & Centers	** Mtg.* Height	Steam 215° Factor 1.00	Average Hot Water Temperature °F					
										200°	190°	180°	170°	160°	150°
										Factor					
										0.86	0.78	0.69	0.61	0.53	0.45
SB-14	1 1/4" I.P.S.	ST-144	4 1/4 SQ.	40	.032" STL	14B	1	18	1500	1290	1170	1040	920	800	680
	2" I.P.S.	ST-243	4 1/4 SQ.	32	.032" STL	14B	1	18	1340	1150	1040	930	810	710	600
	3/4" CU.	STC-3/4 435	4 1/4 x 3 5/8	50	.020" AL	14B	1	18	1840	1580	1440	1270	1120	980	830
	1" CU.	STC435	4 1/4 x 3 5/8	50	.020" AL	14B	1	18	1930	1660	1510	1330	1180	1020	870
	1 1/4" CU.	STC-1435	4 1/4 x 3 5/8	50	.020" AL	14B	1	18	1860	1600	1450	1280	1130	990	840
SB-20	1 1/4" I.P.S.	ST-144	4 1/4 SQ.	40	.032" STL	20B	1	24	1600	1375	1250	1100	975	850	720
	2" I.P.S.	ST-243	4 1/4 SQ.	32	.032" STL	20B	1	24	1390	1195	1085	960	845	735	625
	3/4" CU.	STC3/4 435	4 1/4 x 3 5/8	50	.020" AL	20B	1	24	2090	1800	1630	1440	1270	1110	940
	1" CU.	STC435	4 1/4 x 3 5/8	50	.020" AL	20B	1	24	2180	1870	1700	1500	1330	1160	980
	1 1/4" CU.	STC-1435	4 1/4 x 3 5/8	50	.020" AL	20B	1	24	2130	1830	1660	1470	1300	1130	960
SB-20	1 1/4" I.P.S.	ST-144	4 1/4 SQ.	40	.032" STL	20B	2	24	2250	1935	1755	1550	1370	1190	1010
	2" I.P.S.	ST-243	4 1/4 SQ.	32	.032" STL	20B	2	24	2000	1720	1560	1380	1220	1060	900
	3/4" CU.	STC-3/4 435	4 1/4 x 3 5/8	50	.020" AL	20B	2	24	2820	2430	2200	1950	1720	1490	1270
	1" CU.	STC435	4 1/4 x 3 5/8	50	.020" AL	20B	2	24	2640	2270	2060	1820	1610	1400	1190
	1 1/4" CU.	STC-1435	4 1/4 x 3 5/8	50	.020" AL	20B	2	24	2510	2160	1960	1730	1530	1330	1130

• The ratings above include factors shown below for recommended mounting height. Two tier ratings are based on 6" spacing between elements. Ratings are in BTU per hour lineal foot of active length. Active length is catalog ordering length less 5" on steel, less 4" for copper. • Water ratings applicable to water flow rates of three or more feet per second have been determined by applying factors to steam ratings. • Steel elements are painted black. • Copper, unpainted. If the unit is to be installed at a different height than that recommended, the rating must be adjusted as follows:

Bold, italicized units are rated.

RATING MULTIPLIED BY: FACTOR FROM TABLE B FOR THE ACTUAL MOUNTING HEIGHT* / FACTOR FROM TABLE B FOR THE RECOMMENDED MOUNTING HEIGHT

†NOTE: NPT THREADS FURNISHED ON STEEL ELEMENTS.
PLEASE USE DOMESTIC FITTINGS FOR PROPER INSTALLATION.
**MOUNTING HEIGHT IS 4" GREATER THAN ENCLOSURE HEIGHT

TABLE B

MOUNTING HGT. IN IN.	18 or less	19	20	21	22	23	24	25	26	27	28	29	30	32	34	36	38	40 or more
FACTOR 5¼" OFFSET	1.100	1.100	1.100	1.099	1.092	1.085	1.079	1.072	1.065	1.059	1.052	1.045	1.039	1.025	1.016	1.009	1.003	1.000

Enclosure Ordering Description "S" Sterwyn Sloping Top Enclosure, "B"-4¼" Fin. Size, 14" 20", Enclosure Heights.
Example: SB-14 is 4¼" Fin. Size, Enclosure 14" High.

ACCESSORIES

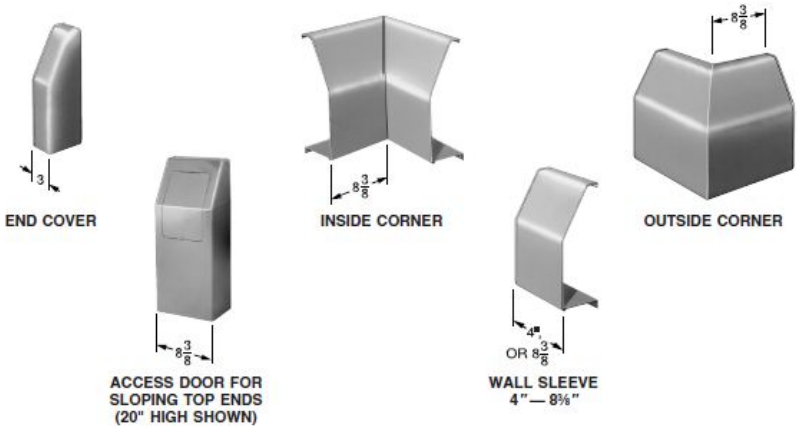


Figure 24: Sterling Product Data Sheet (Pg.3)

	#	Volume m^3	Density - kg/m^3	Cp - kJ/kgK	Thermal Mass - kJ/K	Energy Req (kJ)
			http://www.engineeri	http://www.engineeri		
Walls - Brick	3	1.90	1700	0.92	8,895.15	84,709.81
				http://environmentde		
Desk - Oak	1	0.27	750	2.5	501.44	
http://www.staples.c						
Chairs - Oak	3	0.04	750	2.5	202.73	
http://www.staples.c						
Bookshelf- Oak	1	0.23	750	2.5	436.57	
http://www.staples.c						
Table- Oak	1	0.07	750	2.5	129.68	
http://www.staples.c						
					10,165.58	
in > m						
0.0254						
F	**K**					
55	285.928					
70	294.261					

Figure 25: Thermal Mass Calculation Spreadsheet

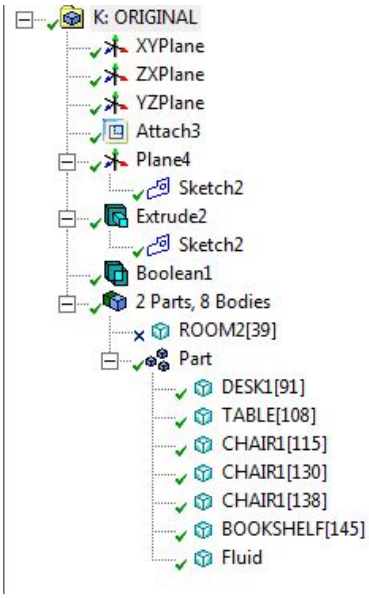


Figure 26: Geometry Tree in ANSYS Workbench

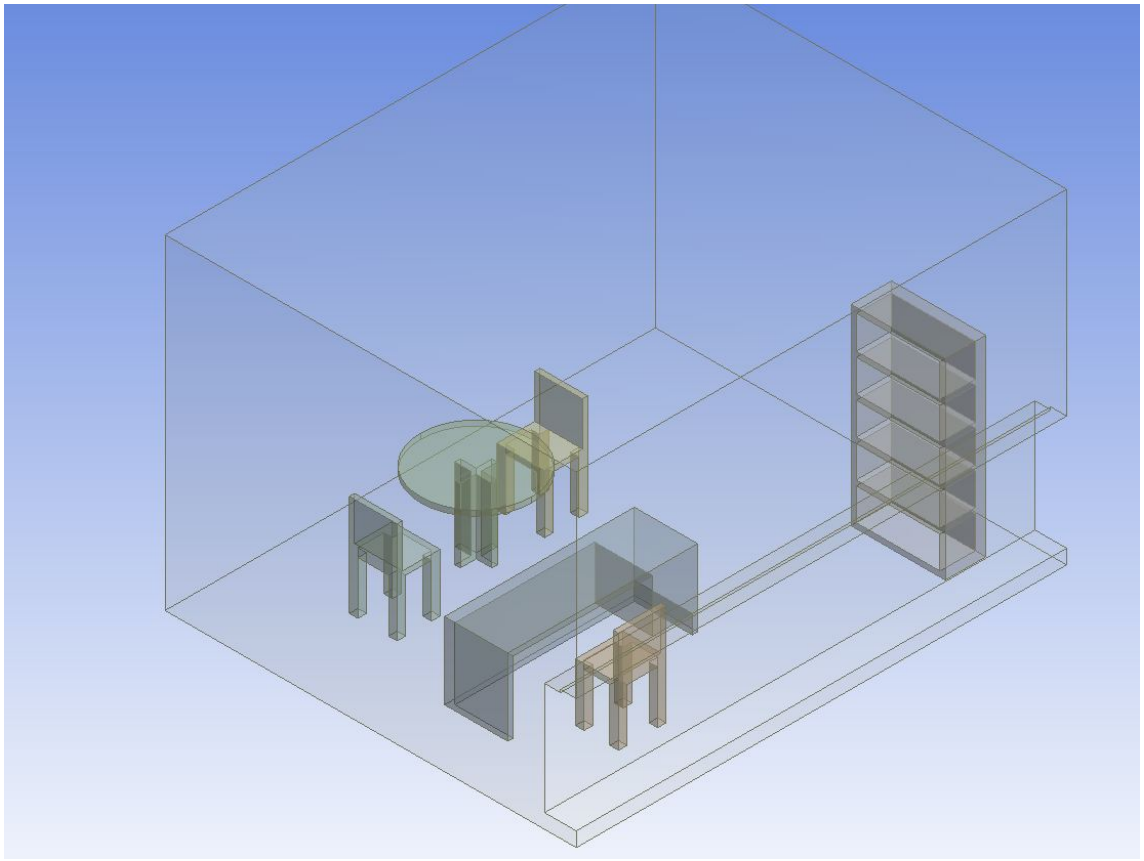


Figure 27: Actual Room Geometry in ANSYS Workbench

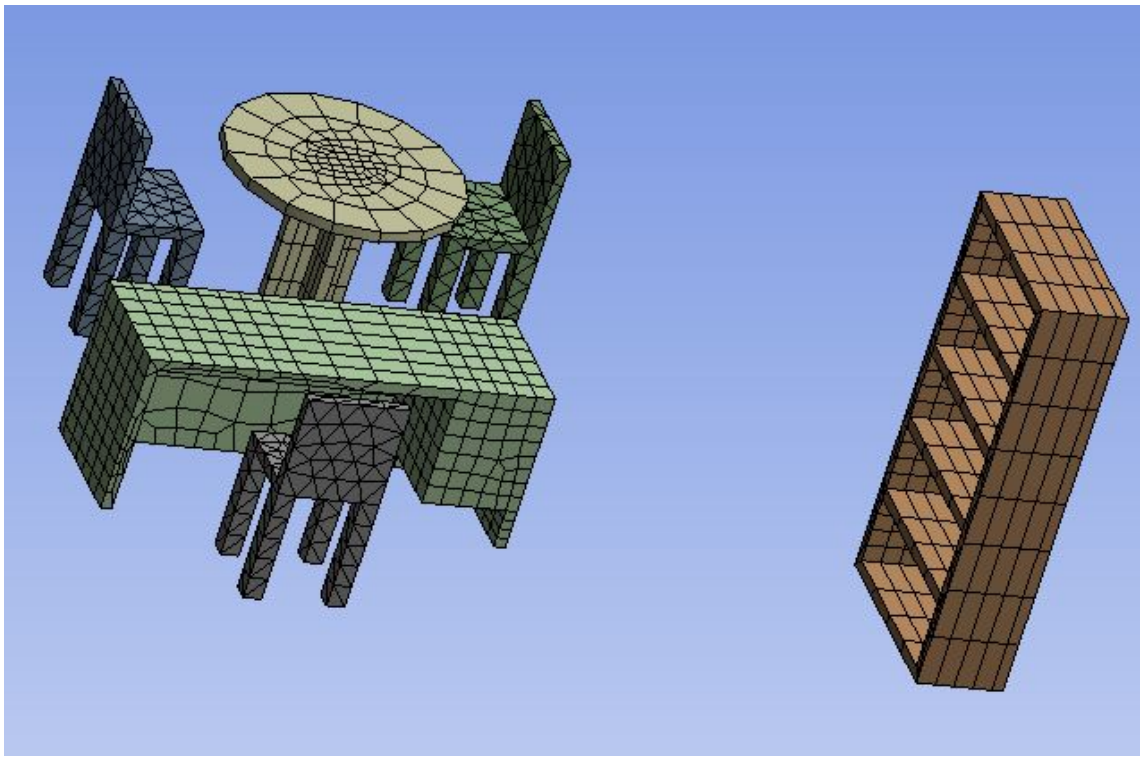


Figure 28: Mesh on Room Objects

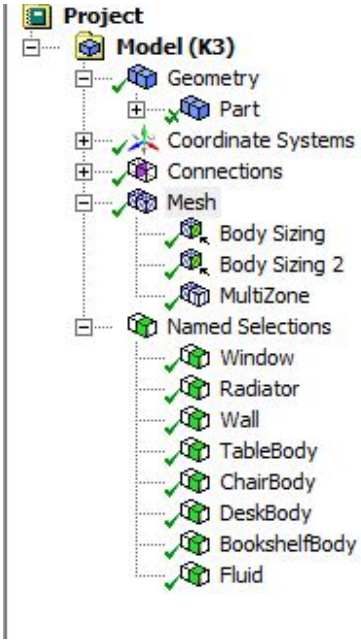


Figure 29: Mesh Tree

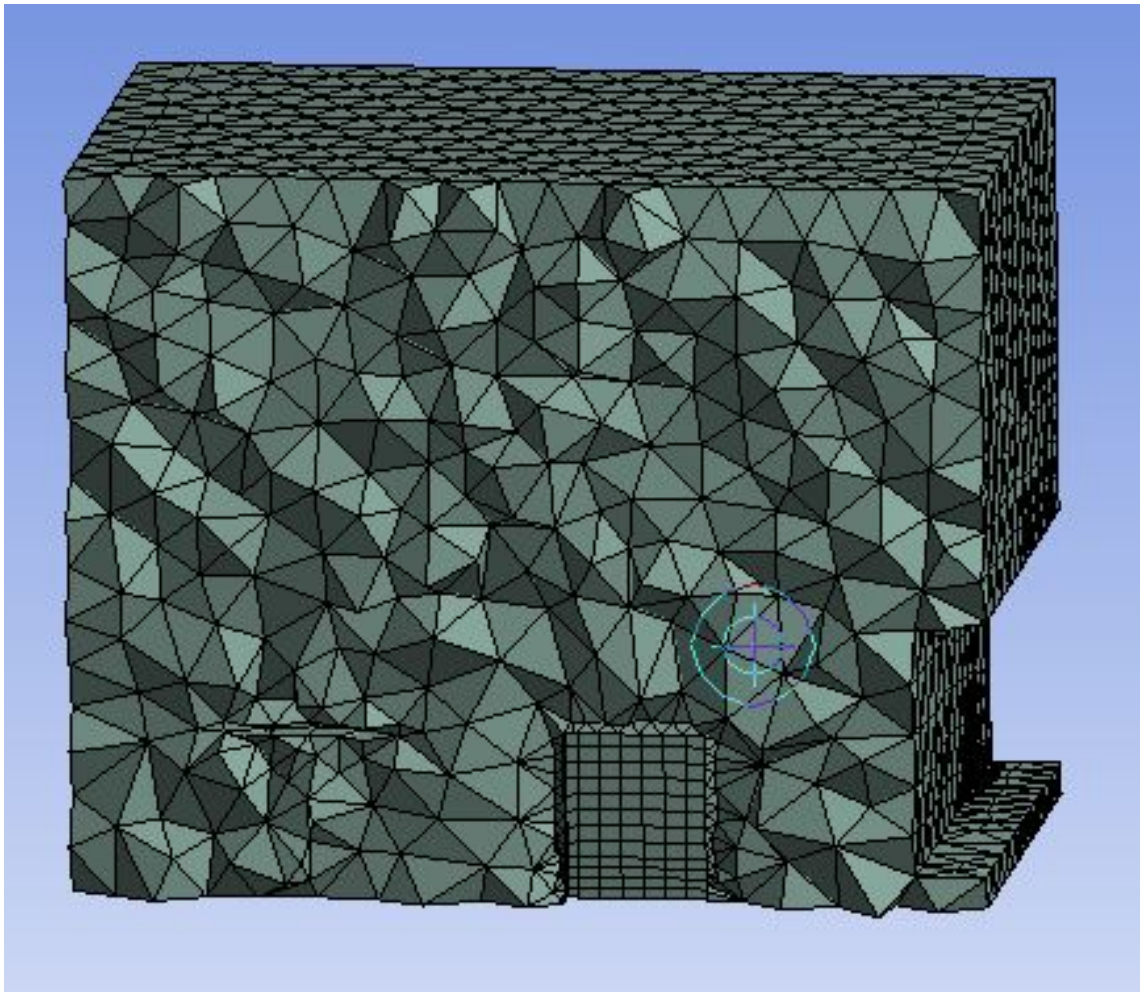


Figure 30: Volume Mesh over Fluid Body

References

[1] Hughel, Gregory. "Managers Need to Address Building-Wide Energy Use." *Facilitiesnet*. TradePress, Feb. 2009. Web. 16 May 2014

[2] Facilities. "2045-Upson Hall Facility Information." *2045-Upson Hall*. Cornell University, 2005. Web. 16 May 2014.

[3] Cornell Facilities. "Facilities Renovations." *Cornell Engineering: About.* Cornell University, Sept. 2013. Web. 16 May 2014.

[4] Eurostar Fenestration. "U-Values." *(* U-Values Defined and How to Calculate Them.) Eurostar Fenestration, 2009. Web. 16 May 2014.

[5] Facilities. "2045-Upson Hall : Utility Costs and Use." *2045-Upson Hall*. Cornell University, 2013. Web. 16 May 2014.

[6] Tile. "Tile, the World's Largest Lost and Found." *Tile.* Tile, Inc., 2014. Web. 16 May 2014.

[7] Ninja Blocks. "Ninja Sphere." *Ninja Blocks.* Ninja Blocks, 2014. Web. 16 May 2014.

[8] Hippih. "HipKey$^{\mathrm{TM}}$ALWAYS BY YOUR SIDE." *HipKey^{TM}.* Hippih, 2014. Web. 16 May 2014.

[9] Scott, James, A,J, B. Brush, John Krumm, Brian Meyers, Mike Hazas, Steve Hodges, and Nicolas Villar. "PreHeat: Controlling Home Heating Using Occupancy Prediction." *UbiComp* September (2011): n. pag. Web.