Dense, Low-Power Environmental Monitoring for Smart Energy Profiling

by

Ashley K. Turza

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Bachelor of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2010

Author . . . .
Department of Mechanical Engineering
May 10, 2010

Certified by . . . .
Joseph Paradiso
Professor of Media Arts and Sciences
MIT Media Lab
Thesis Supervisor

Accepted by . . . .
John H. Lienhard V
Collins Professor of Mechanical Engineering
Chairman, Undergraduate Thesis Committee

Dense, Low-Power Environmental Monitoring for Smart Energy Profiling

by

Ashley K. Turza

Submitted to the Department of Mechanical Engineering
on May 10, 2010, in partial fulfillment of the
requirements for the degree of
Bachelor of Science

Abstract

Recent architectural trends have included exploring open space and the extensive use of glass as building material. While the details of these large, light-exposed, open-air environments can be modeled as thermal fluid systems in CFD simulations, the use of dense sensor networks can provide real-time monitoring of a building's airflow and thermal management systems without the need for computationally-intensive theoretical models, and can use this data to inform and advance these models. Sensor networks can provide an accurate picture of the actual conditions of a building and how those conditions can change over time, due to deterioration or external influences. The information gathered from such networks will be critical in determining the energy efficiency of a building.

To do this, a sensor network made of two types of sensors, temperature-humidity and airflow, was deployed in the large, glass-enclosed atrium of the recently-completed MIT Media Lab Extension (E14) in late March 2010. Their performance was calibrated, monitored, and the preliminary results analyzed in conjunction with the external weather conditions in the Boston metropolitan area. The results show that while the use of the sensors in monitoring temperature and humidity is successful, the airflow sensors currently require a different solution to solve both the need for low-power consumption and resolution, range, and stability in its measurements.

Thesis Supervisor: Joseph Paradiso
Title: Professor of Media Arts and Sciences
MIT Media Lab

Acknowledgments

This project, although it has my name on it, would not have been possible without some very specific people, and for that, they have my endless gratitude.

First, to Professor Joe Paradiso, my thesis advisor, for offering me this fantastic opportunity, his assistance, and support throughout this last semester. The man makes things look easy, and is amazingly easy to talk to when things just don't work the way you want them to.

To Nan-wei Gong, for assistance in countless ways and who tag-teamed this entire project with me. To Mark Feldmeier, for making the comfort control node sensors work, and letting me pick his brain on for calibration advice. To the rest of the Responsive Enviroments group at the MIT Media Lab, for being such a fantastic group of people to work with.

To the Schnedier Electric, specifically Gary Ware, Brett Leida, and Phil London, for working with us on this project and offering me the opportunity to do it in the first place.

To Ray Speth and Alessondra Springmann, for assistance when MATLAB or LyX made me want to slam my head through a wall.

To Mark Mascaro, TB Shardl, Jessy McQuaw and the rest of the Little Tetazoo Historic District who forced me to finishing this project, offered much-appreciated distractions when I needed it, and agreed to punch me if I ever started uselessly panicking. And to Mark especially for playing editor and introducing me to Asian Kung-Fu Generation, which I listened to exclusively for days while writing this.

Finally, to my parents and younger brother, who, through the years and late-night phone calls of desperate frustration, never stopped believing in and supporting me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Energy simulations and computational fluid dynamics (CFD) models are useful prior to construction to aid in determining heating, ventilating, and air conditioning (HVAC) constraints as well as predictions of the thermal air flows within buildings. [7] However, these are theoretical models; they are less useful for determining thermal and airflow considerations in buildings already constructed. While they can be used in such situations, they lack the dynamic ability to couple with systems designed for thermal comfort. It is of more use for energy considerations to be able to know when an area of a building is above or below certain threshold values for temperature in reality than it is to know when they theoretically might be outside the acceptable range. Additionally, the ability to sense these thermal and airflow changes is useful in situations where altering the HVAC system is impossible, due to the fact that the building is already constructed or considering the age of the building in general. Finally, the combination of dense sensing with a CFD or other airflow framework can lead to significantly better predictions and improve the accuracy of existing models.

In situations such as these, an alternate method for climate monitoring within buildings is needed. Most HVAC systems have some method of temperature sensing, but expanding this capacity existing in older versions is difficult and costly. However, that does not mean that these older systems cannot benefit from a ubiquitous monitoring system in order to become more energy-efficient; there simply needs to be an alternate solution. Rather than replacing the HVAC system, the aim of this project was to develop a low-powered solution that can be deployed at any time as a retrofit.

The project was developed with Schneider Electric and deployed in the MIT Media Lab Expansion (E14), which opened in March 2010. The goal was to measure the air quality and thermal ranges of a new, modern building. The sensors developed were deployed in the atrium of E14, shown in Figure 1.1. The area is a large, open space, four stories high, with windows composing the majority of the south-east wall.

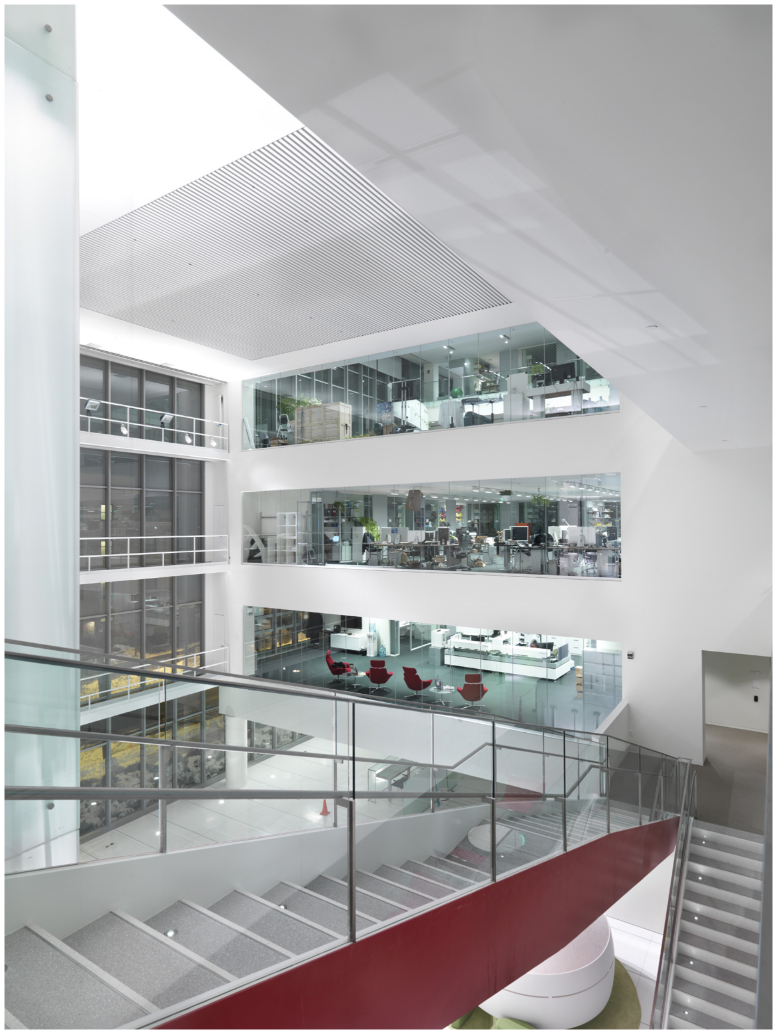


Figure 1.1: Interior of the MIT Media Lab Expansion (E14), as seen from the 3rd floor.

The architecture of the building is such that the atrium has the space needed for installing a dense sensor network. Additionally, the atrium presents the ability to study the energy-efficiency problems associated with open spaces of this type by observing the thermal airflow and air quality in the area.

Chapter 2

Sensor Network

Many types of individual sensors have been used in monitoring the environment, and all are specialized in different ways. There are a myriad of types of sensors that can measure thermal activity, air flow, air speed, and air quality alone. The development of low-power, low cost sensors coupled with embedded processing and RF networks have given rise to the use of sensor networks, wherein sensor nodes can measure different phenomena in their immediate vicinity, and transmit the data back to a central database. In using a sensor network, the sensor nodes cooperate to provide a complete picture of the monitored phenomenon in the area under study, as each node contains an on-board processor for transmitting only the required and partially-processed data. [1]

Two types of sensors were developed by Schneider Electric for use specifically in this project: three-axis airflow sensors that also measured temperature and humidity, and temperature and humidity-only (TH) sensors.

2.1 Development

Power was the limiting factor in designing the sensors. The requirement that the devices be low-power limited the types of sensors that could be used on-board. Further limiting was the requirement that the power source be light-weight, limiting the source to small batteries. These batteries were to last as long as possible, as changing the batteries would require redeploying the entire sensor net. The batteries also had be enough to provide power to the processor as well as to the parts of the sensor that measured velocity.

There are several methods for measuring airflow velocity averaged over a small region. Some of the more common methods that could be considered for measuring airflow are devices such as propeller or turbine meters, pitot-static tubes, and hot-wire anemometers. However, all of these methods have significant disadvantages for this particular application. Rotating sensors are rather large and are unidirectional. They also require a small active power draw to make an accurate reading, as they work by counting the number of rotations in a particular period. Although deep processor sleep between rotary pulses and interrupts can be explored, power must be supplied by the battery during this period, and the time scale must be great enough to count enough rotations to be accurate; unlike temperature sensors, this cannot be done on the order of milliseconds. Hot-wire anemometers suffer from an even worse power problem. Although highly recommended for non-liquid fluid flow, as they are sensitive enough for measuring small variations in the flow and work at low velocities, they require a relatively large amount of power draw to work. The wire must be heated up to a specific temperature, which would place a significant drain on the battery life, and cannot be quickly duty-cycled because of heat capacity and cooling rates.

Unlike the other options, the pitot-static tube works on low power, as it simply measures the pressure differential at either end of the tube. However, like the propellers, it is strictly unidirectional. Worse, it is not suitable for low-velocity measurements in gases because the pressure differences developed are difficult to detect [6].

As all these options are less than ideal, Schneider decided to develop their own sensor for measuring airflow velocities. The design used strain gages attached to thin brass strips. These brass strips are placed between two balsa wood “wings”, one set of wings per each strip, for each of the three axises. As Bernoulli’s equation (Eq. 2.3) can be used to determine a pressure head, the equation can be used to determine possible relation between the pressure readings from the strain gauge and air velocity. This assumes that the pressure orthogonal to the axis in question is dominant to first order, and that the effects of lift on the axis parallel to the air stream are negligible.

The relationship between stress and strain is Hooke’s Law, shown in Eq.2.1.

$$\sigma = E\varepsilon \tag{2.1}$$

Stress is also equal to the pressure

$$P = \frac{F}{A} = \sigma$$

which means

$$P = E\varepsilon \tag{2.2}$$

Bernoulli’s equation can be manipulated to show a relation between velocity and pressure in Eq 2.3, where ß is a constant.

$$\frac{v^2}{2} + gz + \frac{P}{\rho} = \beta \tag{2.3}$$

$$v = \sqrt{2(\beta - gz - \frac{P}{\rho})} \tag{2.4}$$

Combining Equations 2.2 and 2.4 gives us the following

$$v = \sqrt{2(\beta - gz - \frac{E\varepsilon}{\rho})} \tag{2.5}$$

which provides the relationship between velocity and the strain on the brass strips.

In this case, the constant is unknown. Therefore, calibration will be needed to determine that value empirically in order to satisfy the equation. Rather than solve for the constant directly, it was easier to determine a relationship between the strain gauge readings and measured air velocity. The methodology and results of this are discussed in Chapter 3.2.

Although the use of strain gauges to determine the pressure depends on the circuit and sensitivity, and although this method might not be ideal, it does solve the low-power requirement. Other airflow sensors, such as those flowmeters that are propeller-based, require a much higher power draw than strain gauges, which would in turn severely limit the life of the batteries.

The development of the TH sensors was more straightforward, using low-power temperature and humidity sensing circuit components on a specially-designed PCB. Both the airflow and TH sensors would send their collected data to a wall router over wireless. The wall router would then send the data to the database server over wireless.

2.2 Construction

The airflow sensors underwent two major revisions. The second version was developed after discovering problems with the gain and thermal drift in the original. Testing of the overall network continued with the first version while the second was under development. The revision was to the circuit board; the mechanical hardware of the device remained the same.

The wings of the airflow sensors seen in Fig 2.1 were constructed out of balsa wood.

Balsa has two major characteristics that make it an excellent choice for the sensor. The material's low density and high stiffness are required for the wings that catch the air flows that activate the strain gauges. Additionally, as it is typically used by hobbyists, it is remarkably easy to work with. Using a laser cutter, the semi-circular wing patterns were cut out of the wood.

Two small, steel hooks were attached via a thin hot glue layer in grooves cut into the wood. These hooks attached to a vertical brass rod for the x- and y-axes. Two horizontal brass rods were mounted on the circuit board in the center of the sensor to provide attachment points for the z-axis wings. The two semi-circles formed by each wing were attached together via thin brass strips, on which were mounted the strain gauges.

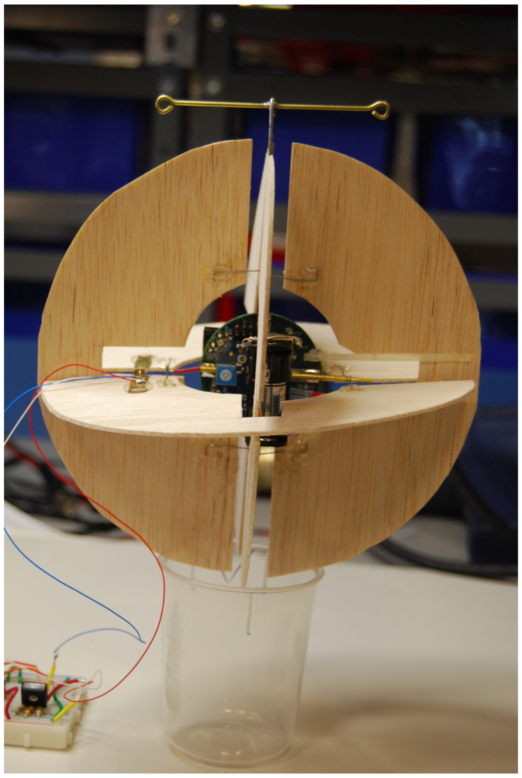


Figure 2.1: Airflow sensor.

The construction for the TH sensors was much simpler, as shown in Fig 2.2. The only requirements were that the box that held the circuit board and batteries provided easy access for resetting the device, that the device had the ability to accurately measure the necessary phenomenon, and that suitable mounting points were provided. Small plastic boxes were dremelled in order to fit the circuit boards; holes were drilled to provide the necessary access and mounting points.

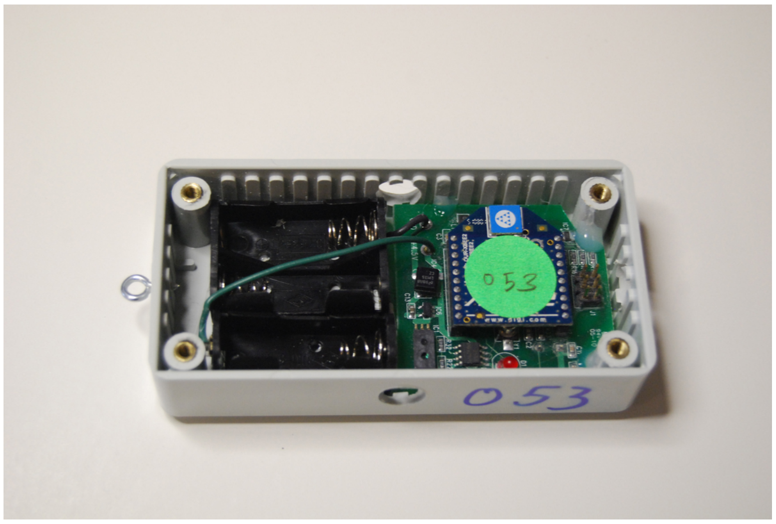


Figure 2.2: Temperature-humidity sensor, open.

One of the main flaws of the airflow sensor design was a lack of robustness. The sensor was designed to be assembled by hand instead of prefabricated, in order to keep costs down. The drawback is that the sensors proved to be finicky and difficult to calibrate, which will be discussed in Section 3.2. The TH sensors, in contrast, were very robust, mostly due to a high degree of integration and a lack of moving parts.

2.3 Deployment

For the first test, 8 airflow sensors and 57 TH sensors were deployed in the atrium of E14, in 4 layers corresponding to 4 floors of the building. Fig 2.3 shows the configuration on the 4th floor.

The data from the operational sensors was sent wirelessly to wall routers scattered around the atrium. The inclusion of wall routers into the sensor network solved many

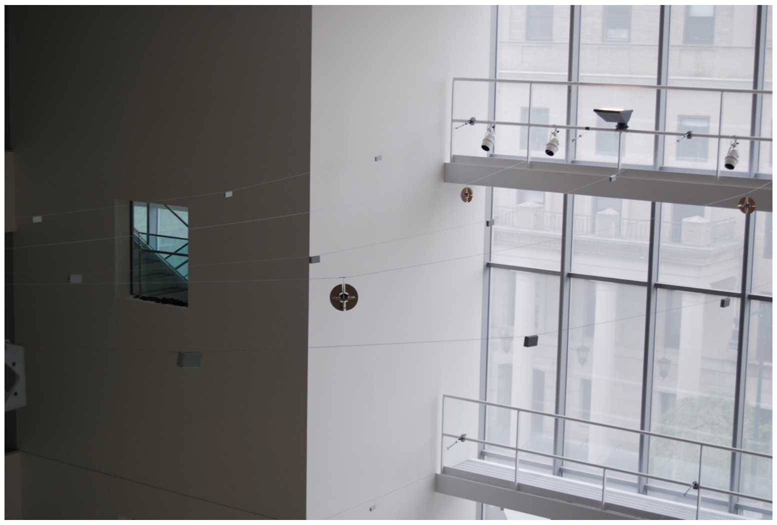


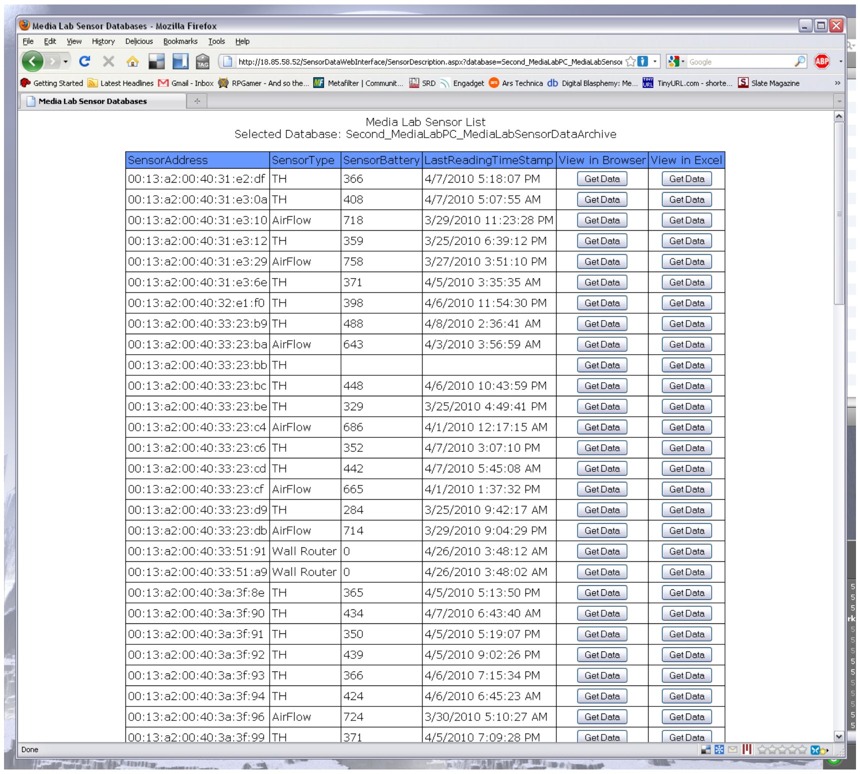
Figure 2.3: Sensors deployed in the atrium of E14.

of the power problems. The power required to transmit data wirelessly is a function of the distance between the nodes. From previous studies, measurements indicate that the power cost for transmission in wireless networks is at least two orders of magnitude higher than the costs for computation. [3]

Traditionally, low power requires the trade-off of reduced quality of service (QoS)[1]. Low power consumption was a requirement in order to extend battery life; the introduction of wall routers solved the power problem for transmitting data without sacrificing QoS. A sensor node transmits on its programmed duty cycle to the nearest wall router, and multiple nodes transmit to the same router. The router has a functionally unlimited power source, as it is connected to the wall socket. Therefore, the wall routers have the necessary power required to transmit the data from each node to a computer running an SQL database and server on the 5th floor of E14.

A series of Python scripts allowed access to the database in two ways: on-site monitoring and a website that allows the user to access the database on an individual node basis off-site. The web interface for the database is accessible at http://18.85.58.52/SensorDataWebInterface/ , as shown in Fig. 2.4.

From the on-site monitoring system, the user could change various data collection



Media Lab Sensor List
Selected Database: Second_MediaLabPC_MediaLabSensorDataArchive

SensorAddress	SensorType	SensorBattery	LastReadingTimeStamp	View in Browser	View in Excel
00:13:a2:00:40:31:e2:df	TH	366	4/7/2010 5:18:07 PM	Get Data	Get Data
00:13:a2:00:40:31:e3:0a	TH	408	4/7/2010 5:07:55 AM	Get Data	Get Data
00:13:a2:00:40:31:e3:10	AirFlow	718	3/29/2010 11:23:28 PM	Get Data	Get Data
00:13:a2:00:40:31:e3:12	TH	359	3/25/2010 6:39:12 PM	Get Data	Get Data
00:13:a2:00:40:31:e3:29	AirFlow	758	3/27/2010 3:51:10 PM	Get Data	Get Data
00:13:a2:00:40:31:e3:6e	TH	371	4/5/2010 3:35:35 AM	Get Data	Get Data
00:13:a2:00:40:32:e1:f0	TH	398	4/6/2010 11:54:30 PM	Get Data	Get Data
00:13:a2:00:40:33:23:b9	TH	488	4/8/2010 2:36:41 AM	Get Data	Get Data
00:13:a2:00:40:33:23:ba	AirFlow	643	4/3/2010 3:56:59 AM	Get Data	Get Data
00:13:a2:00:40:33:23:bb	TH			Get Data	Get Data
00:13:a2:00:40:33:23:bc	TH	448	4/6/2010 10:43:59 PM	Get Data	Get Data
00:13:a2:00:40:33:23:be	TH	329	3/25/2010 4:49:41 PM	Get Data	Get Data
00:13:a2:00:40:33:23:c4	AirFlow	686	4/1/2010 12:17:15 AM	Get Data	Get Data
00:13:a2:00:40:33:23:c6	TH	352	4/7/2010 3:07:10 PM	Get Data	Get Data
00:13:a2:00:40:33:23:cd	TH	442	4/7/2010 5:45:08 AM	Get Data	Get Data
00:13:a2:00:40:33:23:cf	AirFlow	665	4/1/2010 1:37:32 PM	Get Data	Get Data
00:13:a2:00:40:33:23:d9	TH	284	3/25/2010 9:42:17 AM	Get Data	Get Data
00:13:a2:00:40:33:23:db	AirFlow	714	3/29/2010 9:04:29 PM	Get Data	Get Data
00:13:a2:00:40:33:51:91	Wall Router	0	4/26/2010 3:48:12 AM	Get Data	Get Data
00:13:a2:00:40:33:51:a9	Wall Router	0	4/26/2010 3:48:02 AM	Get Data	Get Data
00:13:a2:00:40:3a:3f:8e	TH	365	4/5/2010 5:13:50 PM	Get Data	Get Data
00:13:a2:00:40:3a:3f:90	TH	434	4/7/2010 6:43:40 AM	Get Data	Get Data
00:13:a2:00:40:3a:3f:91	TH	350	4/5/2010 5:19:07 PM	Get Data	Get Data
00:13:a2:00:40:3a:3f:92	TH	439	4/5/2010 9:02:26 PM	Get Data	Get Data
00:13:a2:00:40:3a:3f:93	TH	366	4/6/2010 7:15:34 PM	Get Data	Get Data
00:13:a2:00:40:3a:3f:94	TH	424	4/6/2010 6:45:23 AM	Get Data	Get Data
00:13:a2:00:40:3a:3f:96	AirFlow	724	3/30/2010 5:10:27 AM	Get Data	Get Data
00:13:a2:00:40:3a:3f:99	TH	371	4/5/2010 7:09:28 PM	Get Data	Get Data

Figure 2.4: Sensor database web interface.

parameters for each node. For example, the user could send a command to a particular node that would change the transmission cycle between 30 - 180 seconds. Since data transmission is the largest source of power drain for the sensor, changing the transmission cycles was the easiest way to alter the lifetime of each sensor after the network was deployed.

Chapter 3

Sensor Testing

Before deployment, the sensors, especially the airflow sensors, needed to be tested and calibrated in order to establish a baseline for the readings gathered. Calibration of the TH sensors was simple and could be accomplished via software. The airflow sensors, in contrast, needed specialized testbed hardware to be developed. One of the key pieces developed for this process was a portable, low-cost wind tunnel that would allow for complete control over the air velocity vector in the airflow sensor calibration.

3.1 Test Structure Design

A basic wind tunnel was constructed in order to test and calibrate the airflow sensors. The wind tunnel was designed so that the sensor could hang approximately in the middle of the tunnel and measure the velocity of the air moving past it, as shown in Fig. 3.1. This constraint led to some particular design decisions.

A wind tunnel was necessary in order to create a controllable testing environment for the airflow sensors. A wind tunnel constricts the airflow to a one direction vector; furthermore, the wind tunnel restricts the airflow to strictly laminar flow, thus removing the effects of turbulence on air speed. This is necessary to make the velocity measurements more accurate. In order to force the air from the inlet source, a fan, into laminar flow, it was necessary to use a flow straightener. The flow straightener is designed to smooth out any turbulent and transitional flows, thus ensuring operation in the laminar regime. Figure 3.1 shows the basic design of such a system.

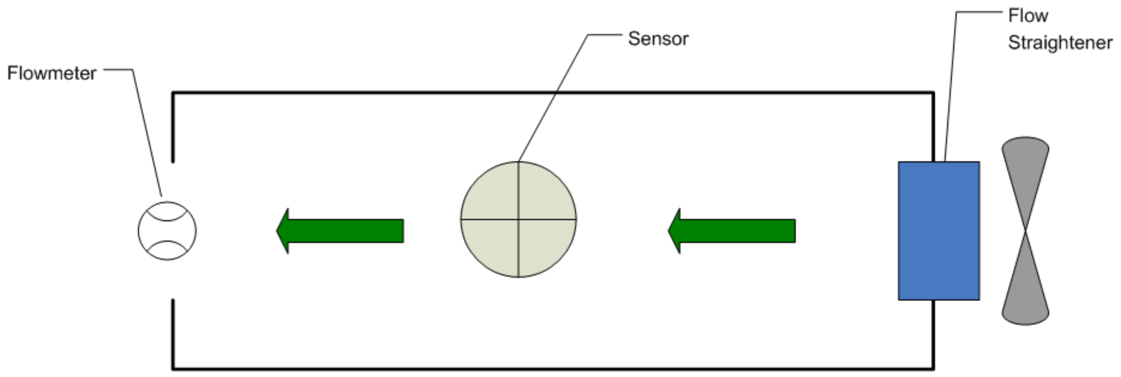


Figure 3.1: Schematic of the wind tunnel testing apparatus.

The outlet is as large as the inlet in order to avoid creating a nozzle, which would cause the velocity inside the tunnel to change in ways that could not be easily measured. Ideally, the inlet velocity is equal to the outlet velocity, so that the velocity seen by the sensors inside the tunnel is known. This was confirmed by measuring air velocity at the inlet and outlet of the constructed tunnel.

To construct the wind tunnel, a 10.5in-by-10.5in-by-38in cardboard box was obtained. A hole of diameter 7.5in was cut on two opposite faces. A honeycomb matrix of drinking straws was constructed to act as the flow straightener . The straws were cut to a length of 5in and glued together, providing the same function as a commercial flow straightener for a fraction of the cost. The honeycomb was then placed inserted into the inlet hole and duct taped to remain in place, as shown in Figure 3.2. Figure 3.3 shows the completed wind tunnel.

A small, propeller-based, electronic flowmeter was also attached to the box, centered on the outlet. At the inlet was placed a small, three-speed desk fan, which was plugged into a variable transformer. The variable transformer allowed for further voltage control between the fan and the 120V wall socket, which in turn gave for more

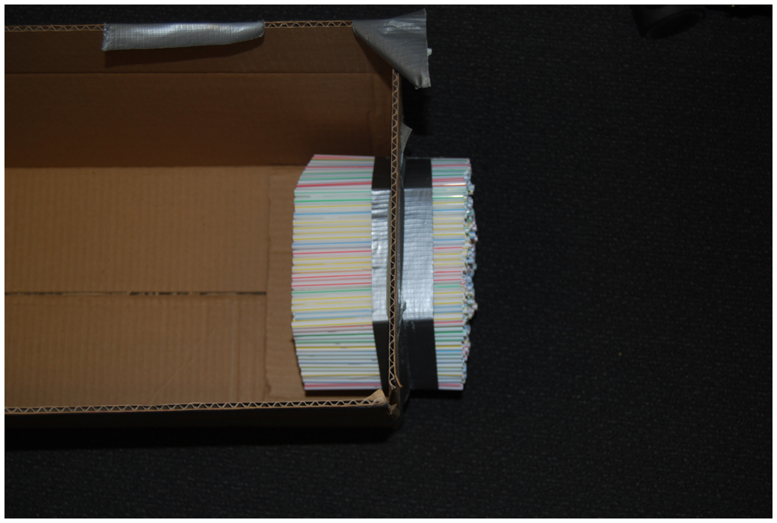


Figure 3.2: Flow straightener honeycomb for wind tunnel.

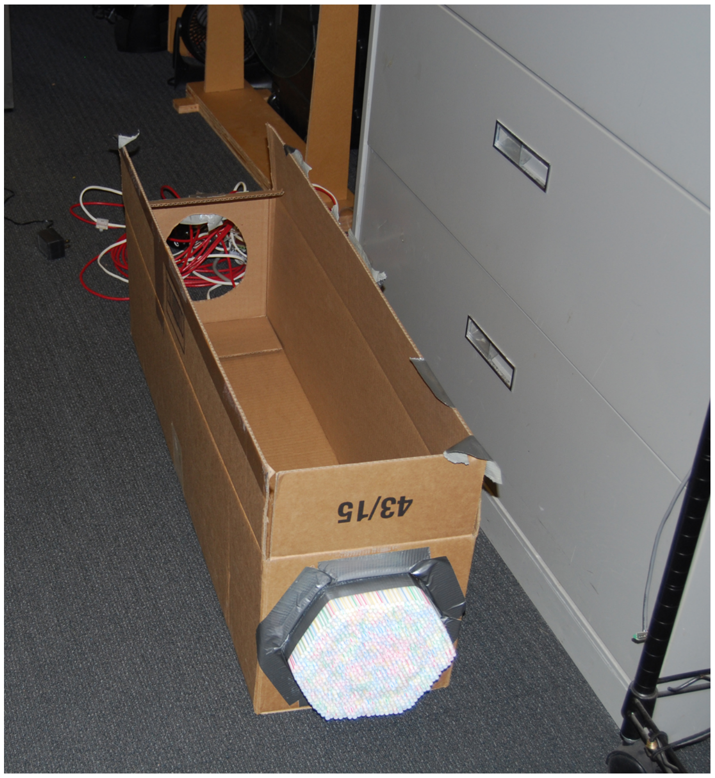


Figure 3.3: Constructed wind tunnel.

variance in possible fan speeds beyond the fan's designated three.

The sensor was placed inside the finished apparatus, roughly centered in the airflow stream. The box was then taped shut during the actual testing to seal the tunnel.

3.2 Calibration Test Methodology

Our airflow sensor calibration was designed to find a correlation between wind velocity and voltage. Calibration was accomplished using the methodology developed by Feldmeier in the Responsive Environments group of the MIT Media Lab. The sensors were calibrated as described in the work [2], and used the same calibrated propeller flowmeter as a baseline measurement, although in this case the sensor data was calibrated to voltage instead of RPM, as the sensors in question use strain gauges instead of propeller blades. Using the wind tunnel designed in Section 3.1, two of the three axes of the airflow sensor were measured against the Extech 45170 Environmental Meter (flowmeter).

The sensor was placed in the center of the wind tunnel with the desired axis orthogonal to the direction of the airflow. Wires were attached to the outputs of the strain gauges being measured before the tunnel was sealed. These wires were then connected to an improved copy of the bridge amplifier test circuit on-board the device. We replaced the differential amplifier with a low-drift instrumentation amplifier and added additional low-pass filtering and gain. From this circuit, an Arduino Duemilanove board was connected. The Arduino board allows a program to monitor the serial output of the ADC and transmit it to a connected computer via USB. The ADC values from the Arduino board would change based on the physical effects on the strain gauge.

The test circuit was centered at a voltage of 2.5V on the input to the op-amps in the circuit. From this, there is a relationship between the ADC values and voltage as

Eq.3.1 demonstrates.

$$V = (\frac{ADC}{1023}) \times 5V \tag{3.1}$$

Two axes were tested using this method: the x-axis and the z-axis. The wind tunnel was kept horizontal for the x-axis calibration and mounted vertically for the z-axis test. The sensor would settle during a 5 minute period after activating its 2.5V via power supply. Using the variable transformer attached to the fan, the effects on the ADC of air speeds between 0 and 3 m/s were measured. The settling period was only for the 0 m/s measurement; all tests took place immediately after each other, with no significant break between different velocity settings.

Once this step was completed, the sensor was placed in a plastic box with a TH sensor and a glass of hot water. While still attached to the Arduino board, the sensor sat in the temperature- and humidity-controlled environment for 20 hours. The TH sensor would monitor the temperature and humidity conditions, while the Ardunio board would monitor the drift on the sensor's z-axis while in stagnant air. From the data gathered, the effects of drift, humidity, and temperature on the airflow sensor could be determined.

Chapter 4

Evaluation

Two versions of the airflow sensors were evaluated in separate sensor and sensor network tests. The first test used the initial version of the airflow sensors; the second explored detailed calibration of only the airflow sensors with the revised circuitry. Both evaluations are discussed separately in this chapter.

4.1 Version 1 Sensor Network Evaluation

For purposes of evaluation, Version 1 of the sensor network only measured temperature and humidity in the E14 atrium. As previously mentioned, each node is identifiable by its MAC address. The last three characters of this address serve as a unique identifier for each node, and serve as the name of the node.

The evaluation looked at all 53 nodes, 7 airflow and 46 TH sensors, that took data March 27 - March 29, 2010. This was the longest time period available in which all nodes were sending data; the duration of the battery life for each node was independent and some lasted much longer than others, for reasons that are still being investigated. Other nodes otherwise stopped sending data due to internal errors that could not be corrected until after the test duration. The list of the 53 nodes and their locations are listed in Table 4.1. These locations are in a 500 unit x 500 unit x 5 story

(1 unit = 0.064 meter) grid, representing the area of the atrium up to the ceiling of the 4th floor. The origin is located in Figure 4.1, in the top-left corner of the atrium, as seen looking from the 4th floor balcony.

Table 4.1: List of tested nodes in the E14 atrium, March 25, 2010 - April 2, 2010. x and y positions represent locations of the nodes in units (1 unit = 0.064m) relative to the left wall-window corner; z positions represent story, with 10cm displacement from cable droop

Node	X	Y	Z		Node	X	Y	Z		Node	X	Y	Z	
’cc3’	80	400	2		’ce5’	80	400	3		’cd2’	80	400	4	
’cc0’	80	320	2		’cf1’	80	320	3		’cbb’	80	240	4	
’cbd’	80	240	2		’ce2’	80	240	3		’cad’	80	320	4	
’caa’	80	160	2		’cd3’	80	160	3		’074’	80	160	4	
’ccf’	80	80	2		’1f0’	240	400	3		’99d’	80	80	4	
’30a’	240	400	2		’f94’	240	320	3		’cbc’	240	400	4	
’f92’	240	240	2		’f9b’	240	160	3		’fce’	240	320	4	
’f8e’	240	160	2		’fc6’	240	80	3		’cb8’	240	240	4	
’fd0’	240	80	2		’fbf’	400	400	3		’cf2’	240	160	4	
’f93’	400	400	2		’f91’	400	320	3		’cde’	240	80	4	
’fbc’	400	320	2		’fc8’	400	160	3		’cb9’	400	400	4	
’cfd’	400	240	2		’2df’	400	80	3		’fb5’	400	320	4	
’caf’	400	160	2		’3bc’	80	450	3		’fb4’	400	240	4	
’f9e’	400	80	2		’a8c’	240	450	3		’f90’	400	160	4	
’fdf’	250	50	2		’996’	400	450	3		’f99’	400	80	4	
’cd7’	240	450	2		’cd4’	80	80	3		’fe2’	240	30	4	
’cf8’	260	450	2		’312’	400	240	3		’3c6’	400	30	4	
’d04’	240	320	2		’fab’	240	240	3						

Outdoor weather data was obtained via the Weather Underground METAR archive [4, 5]. From this data, it is possible to compare and draw connections, if any, between the exterior conditions of the building and the indoor conditions as detected by the sensor network.

The data from March 29, 2010 provide a visible connection between outdoor and indoor temperature and humidity conditions. Fig 4.3 shows node-measured temperature and humidity side-by-side at five intervals, separated by approximately six hours each. The data from this date clearly show both the temperature and humidity vary-

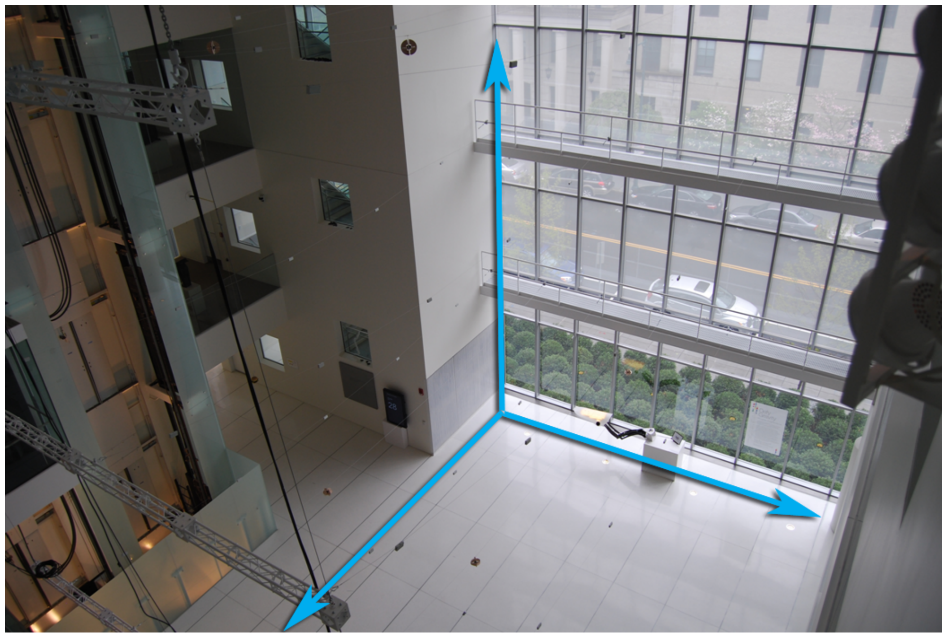


Figure 4.1: E14 atrium as a Cartesian coordinate system.

ing over the course of a 24-hour period. In this particular case, temperatures remained rather constant between 18 - 23ºC.

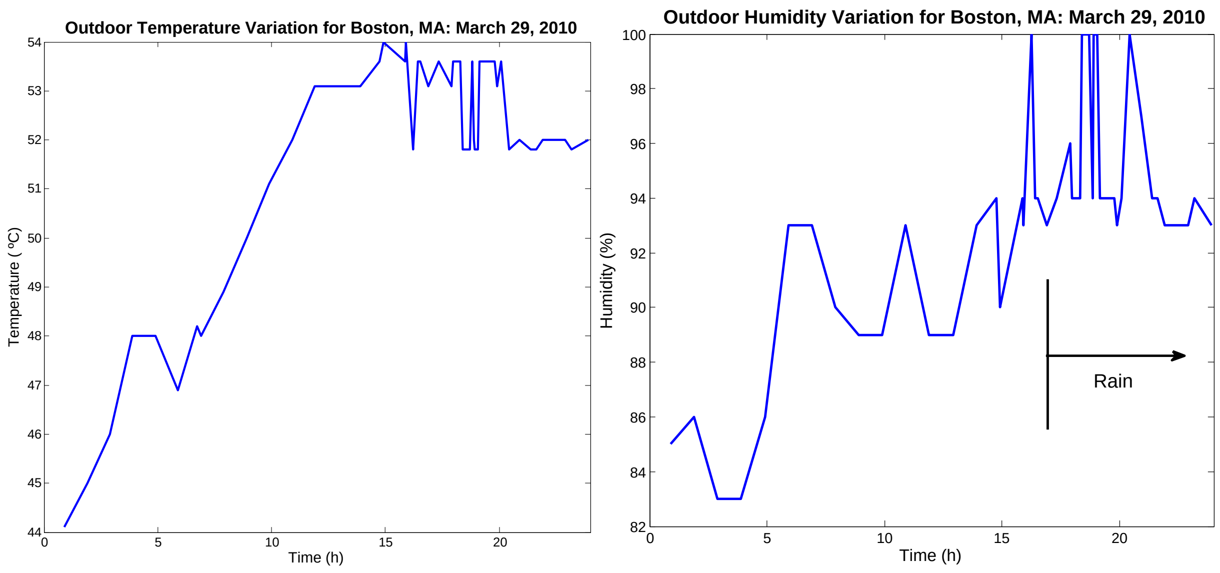


Figure 4.2: Exterior temperature and humidity for Boston, MA for March 29, 2010 (from Weather Underground METAR data).

Although this describes the interior conditions, this data is relatively meaningless without appropriate context to the outdoor conditions. According to the METAR

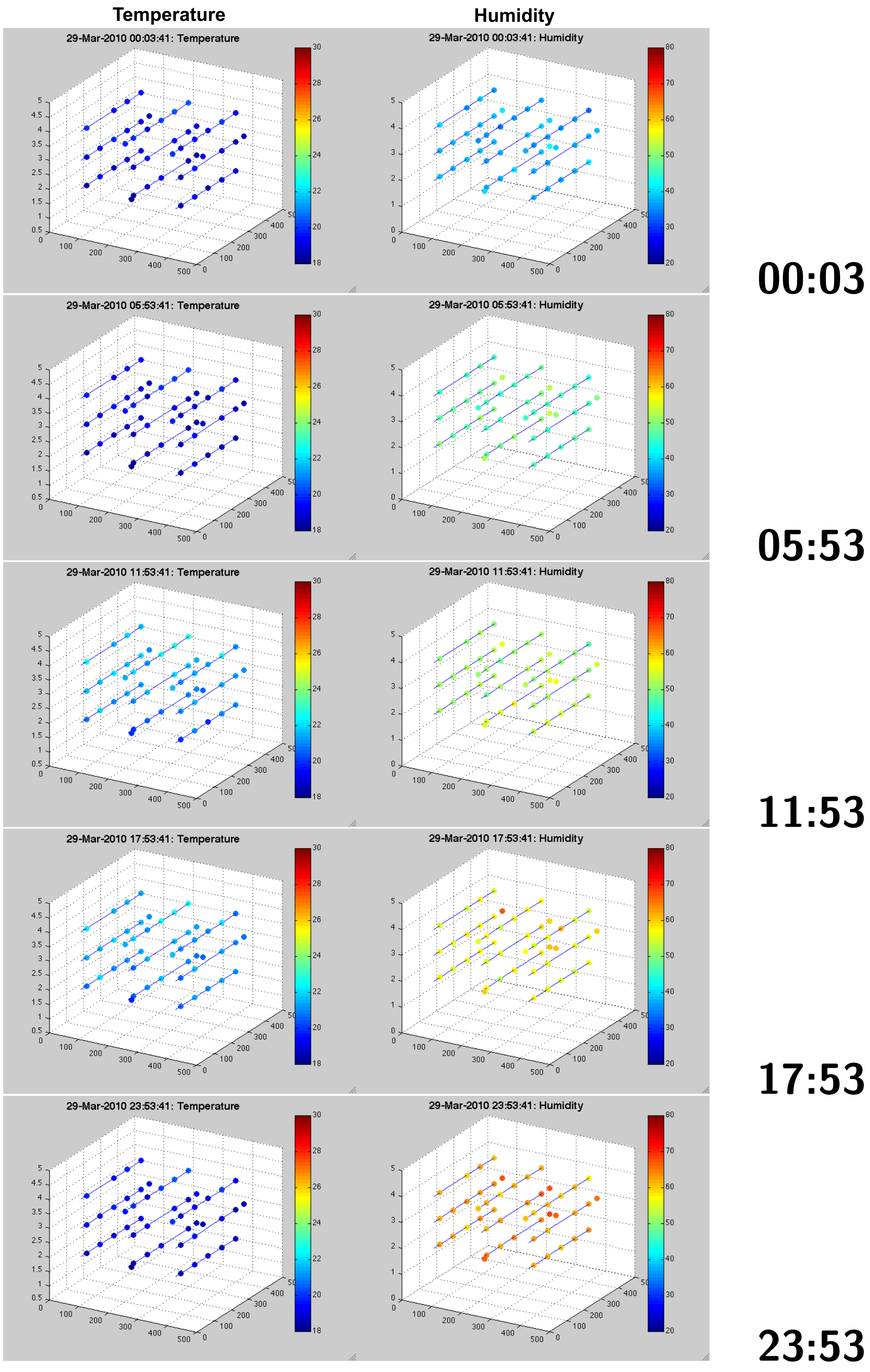


Figure 4.3: Temperature and humidity sensor data for March 29, 2010.

data, it rained all day March 29, 2010, with heavy rain starting at 17:54 and continuing into the night [5], as noted in Fig. 4.2.

Despite the 10ºC variation in temperature during the 24-hour period outside, the HVAC system indoors kept the atrium temperature relatively constant with only a 5ºC variation over the same period.

The data from METAR for March 28, 2010, seen in Fig 4.5, show a much lower humidity during the afternoon, but a greater variation in temperature [4]. Comparing the interior and exterior temperatures does in fact show a rise in temperature at the same time period between 8:00 - 17:00. In this case, however, the humidity remained relatively constant inside, despite the large drop outside. From this, it can be argued that the HVAC system does in fact attempt to keep the relative humidity inside the building independent of the exterior humidity, as expected.

Because the sensor network is arrayed in a grid pattern, the data collected can also be used to model the variations of temperature and humidity within the four levels of the atrium itself. Figure 4.6 takes the plane made by three sensor layers (Floors 2 - 4) at 12:23, and examines the temperature gradient between the nodes to calculate a contour map. From these maps, for Floors 2 and 3, higher temperatures are located away from the windows, while for Floor 4 the opposite is true. According to the building plans, there is an HVAC air outlet located right against the windows of the atrium on the 1st floor; it is possible that the 2nd and 3rd floors feel the effects of the air outlet strongly enough to mitigate the effects of sunlight through the windows, and that these effects have dissipated by the time the air gets to the 4th floor. Therefore, on the 4th floor, the thermal effects of the sun are dominant. This can also explain why the 4th floor has much cooler temperatures farther away from the windows. As the top of the 4th floor of the atrium is the ceiling of the atrium, HVAC vents are also placed, which can affect the temperatures measured.

To examine the effects over a longer period of time than a gradient contour map,

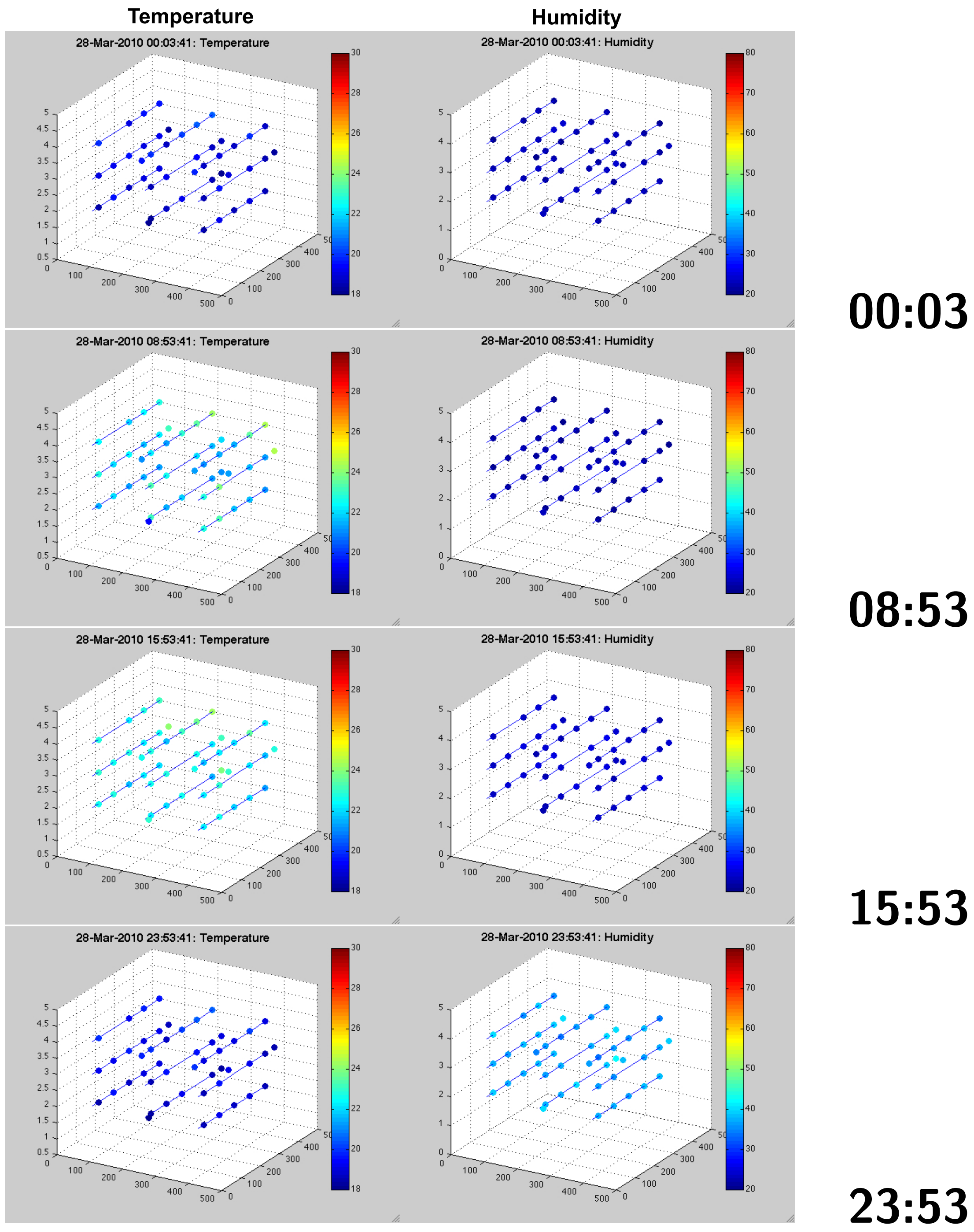


Figure 4.4: Temperature and humidity sensor data for March 28, 2010.

the data from four nodes for an 8 day period were chosen for particular examination here. Each node is located on a different level in the sensor net, as listed in Table 4.2, and the temperature and humidity data for each was plotted in Figures 4.7- 4.9.

While the variation between the maximum and minimum temperatures measured

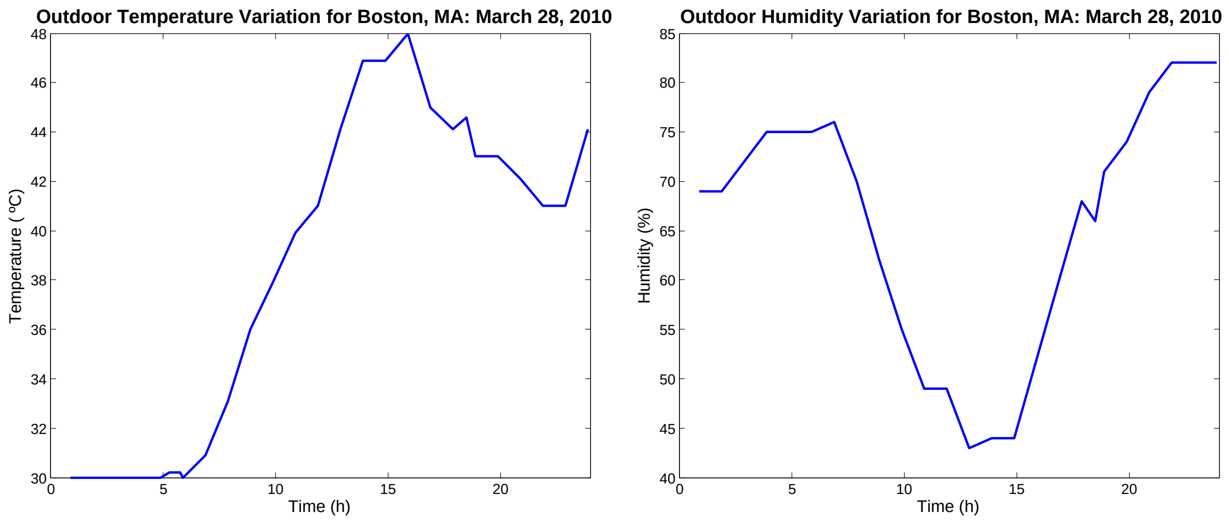


Figure 4.5: Exterior temperature and humidity for Boston, MA for March 28, 2010 (from Weather Underground METAR data).

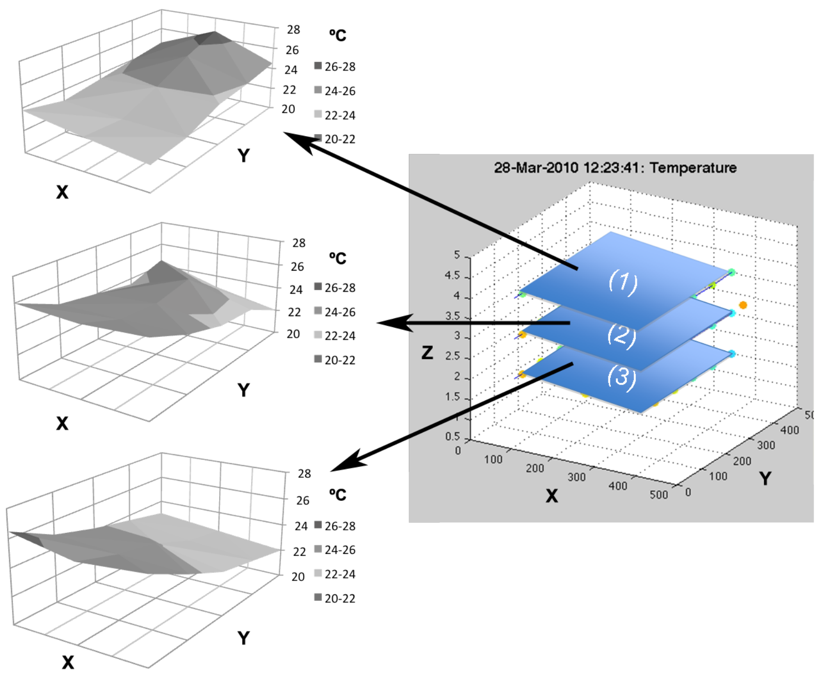


Figure 4.6: Gradient temperature for March 28, 2010 in E14 atrium, noon.

during this time period was 1ºC at most, the rising and falling of the indoor temperature on a day-to-day basis is clearly visible. Temperatures in this region show warmer temperatures at the lower levels, which fits with the gradient profile shown in Fig. 4.6, as these nodes are not located next to the window, but beyond the midpoint

Table 4.2: Examined Node Locations

Node	X	Y	Z
cbd	80	240	2
ce2	80	240	3
cbb	80	240	4

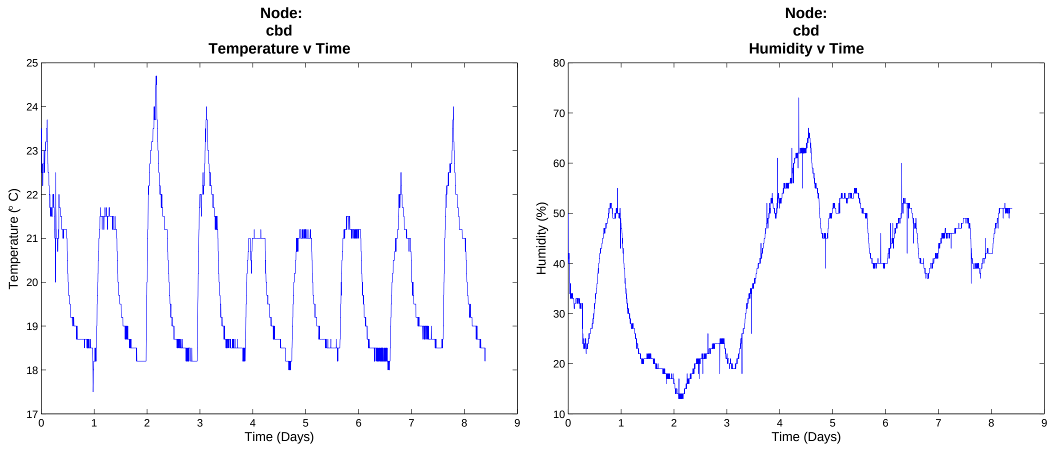


Figure 4.7: Node cbd (located on the 2nd floor near the elevators) temperature and humidity data from period between March 24, 2010 - April 3, 2010.

of the y-axis. The day-to-day data trends match the gradient contour.

The humidity analysis over the same nodes shows an almost uniform image. The maximum in all plots happens on March 29, 2010, which is the same day analyzed in depth earlier. There are some variations between the four nodes, but as a general rule, humidity does not seem to be as affected by location in the grid as temperature does. This hypothesis is further supported by the distribution for humidity in Figures 4.3 and 4.4; the humidity variation between the nodes is incredibly uniform with only a few outliers.

Also notable is that all the figures show a significant variation on Days 2 and 3: spiking for temperature and drops in humidity. The dates for this time period match up with a weekend, and HVAC systems generally are run differently on weekends. Without people there, the assumption is that the building can run warmer and that the humidity will be lower. This HVAC behavior is reflected in the data captured by the nodes.

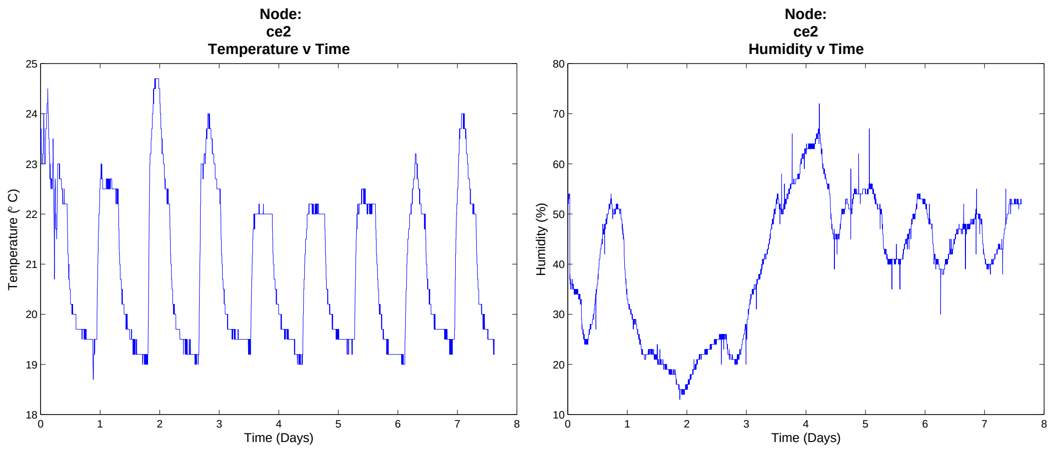


Figure 4.8: Node ce2 (located on the 3rd floor near the elevators) temperature and humidity data from period between March 24, 2010 - April 3, 2010.

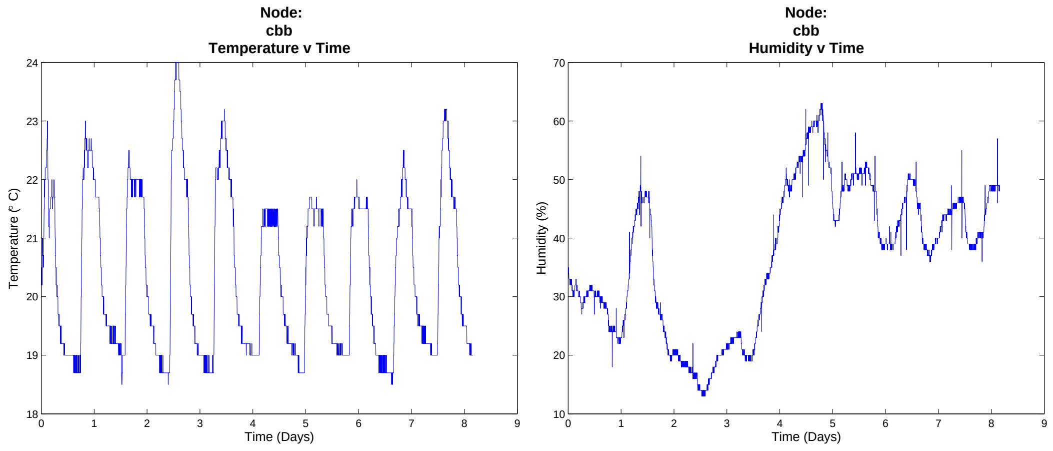


Figure 4.9: Node cbb (located on the 4st floor near the elevators) temperature and humidity data from period between March 24, 2010 - April 3, 2010.

4.2 Upgraded Airflow Sensor Evaluation

As previously mentioned, the original electrical design of the airflow sensor nodes proved to be problematic and could not produce consistent data. Two airflow sensors were refitted with the new circuit design and reevaluated. Both the x-axis and z-axis were tested. The results are shown in Figures 4.10 and 4.11. Tables 4.3 and 4.4 tabulate the $\frac{\chi^2}{DoF}$ results for 1 degree of freedom for each axis. The test for the x-axis was conducted inside the wind tunnel; due to design issues, the z-axis test was conducted without using the wind tunnel.

Table 4.3: χ^2 values for voltage and mean air speed values for x-axis with 1 degree of freedom

Air Speed (m/s)	Voltage (V)
0	0.0191
0.8000	0.0328
1.1000	0.0282
1.2733	0.0205
1.5133	0.0203
1.7067	0.0213
2.2333	0.0310
2.7200	0.0250

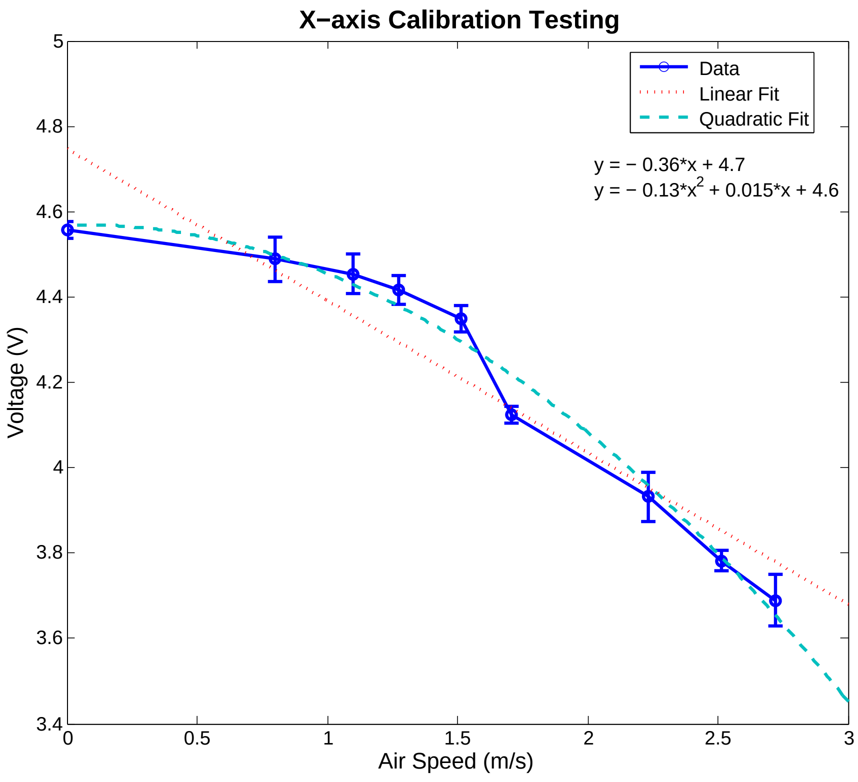


Figure 4.10: Calibration results of the airflow x-axis.

Eq. 2.5 implies a square root relationship. Taken at a small range of values, we assumed that a linear approximation would be sufficient to the first-order. However, the data does not resemble a linear relationship between air speed and voltage, although the results from the sensor are far too noisy and unstable to make that expressly clear. We tried a linear fit to unsatisfactory results, and found that the data more

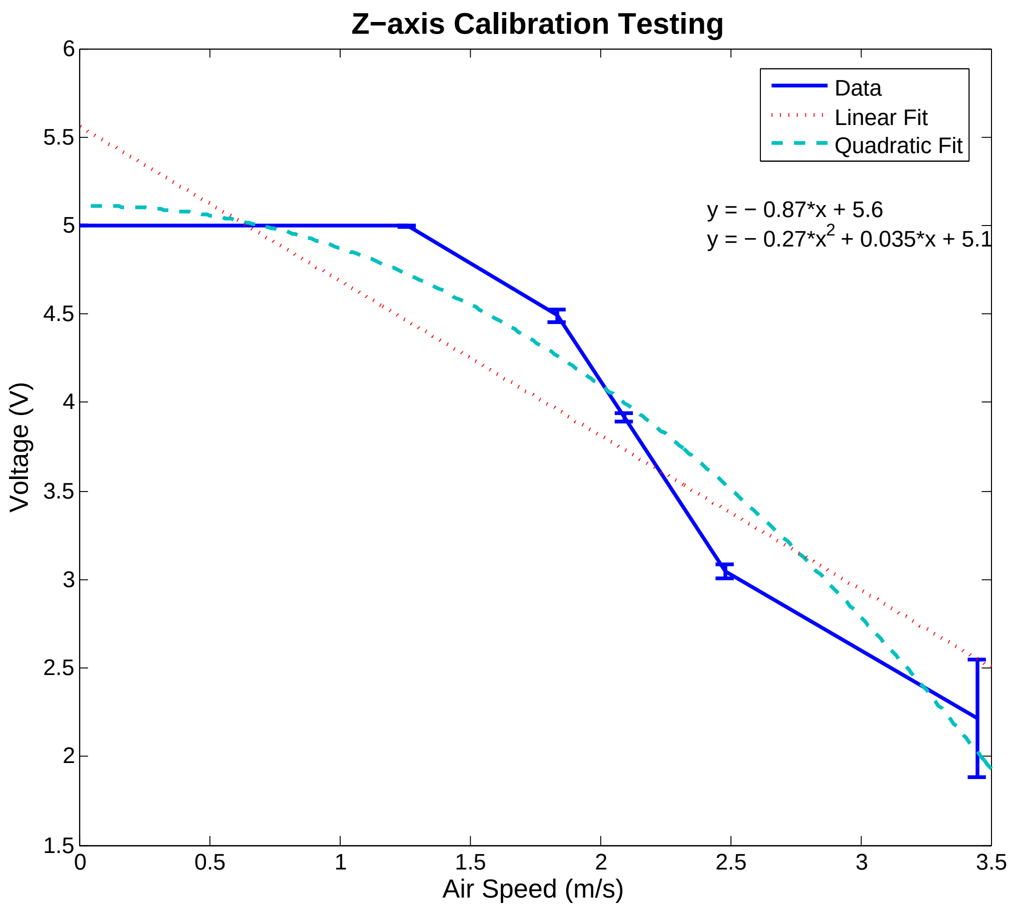


Figure 4.11: Calibration results of the airflow z-axis.

Table 4.4: χ^2 values for voltage and mean air speed values for z-axis with 1 degree of freedom

Air Speed (m/s)	Voltage (V)
0	0.0146
1.2600	0.0147
1.8333	0.0199
2.0933	0.0284
3.4467	0.0885

follows a quadratic curve. The reasons for this behavior are unknown and completely contradict the assumptions for mechanical behavior that were made in the design. The error bars from both tests are rather significant, and the distribution of voltages for each air speed overlap in some cases, meaning that it would be difficult to tell from voltages alone if the sensor were reading 1.5 m/s or 1.0 m/s, for example.

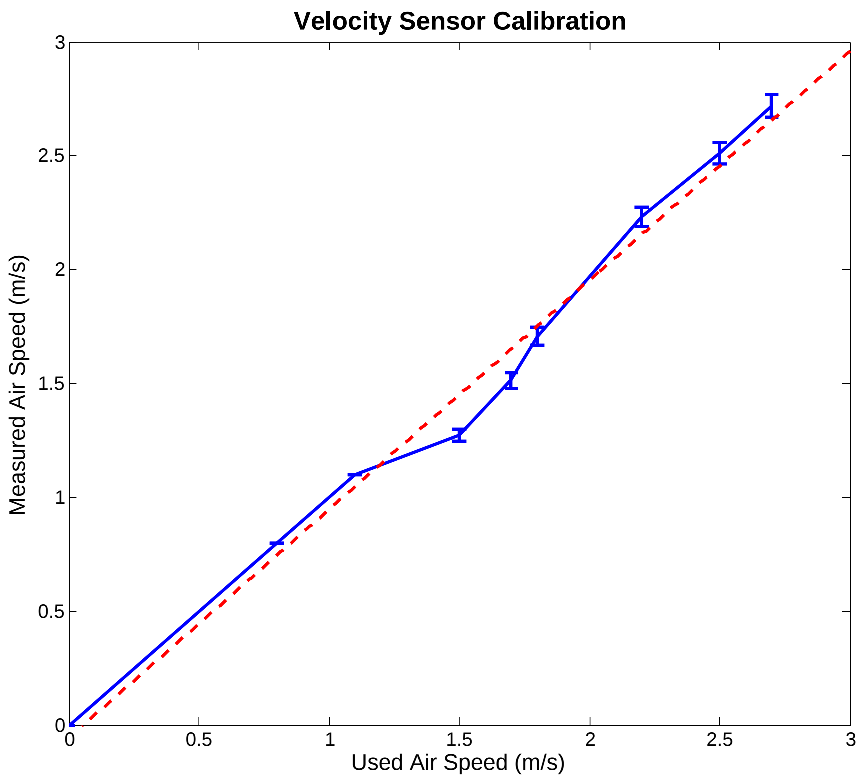


Figure 4.12: Extech Environmental Meter air velocity data.

In contrast, the stability of the Extech 45170 Environmental Meter used to measure the velocity of the airflow past the sensor was much higher. This flowmeter uses a propeller to measure the velocity of the air moving past it; Figure 4.12 shows the average air speed used in as the x-axis in Figures 4.10 and 4.11 against the raw measured air speed as read by the flowmeter. The latter contains all variations in velocity readings. As shown, the meter is acceptably stable, much moreso than the data received from the airflow sensor under test.

To further categorize the effects of drift, the sensor was, as previously mentioned, placed in a sealed box and left running for a 20-hour period. Due to the glass of initially steaming water in the box, the humidity remained nearly constant at 102%, which is a glitch in the device and thus not included, and the temperature varied within 2 ^{o}C, seen in Fig. 4.13. However, during this same period of time, the sensor did not remain constant; it did not even read the same result as 0 m/s reading in the

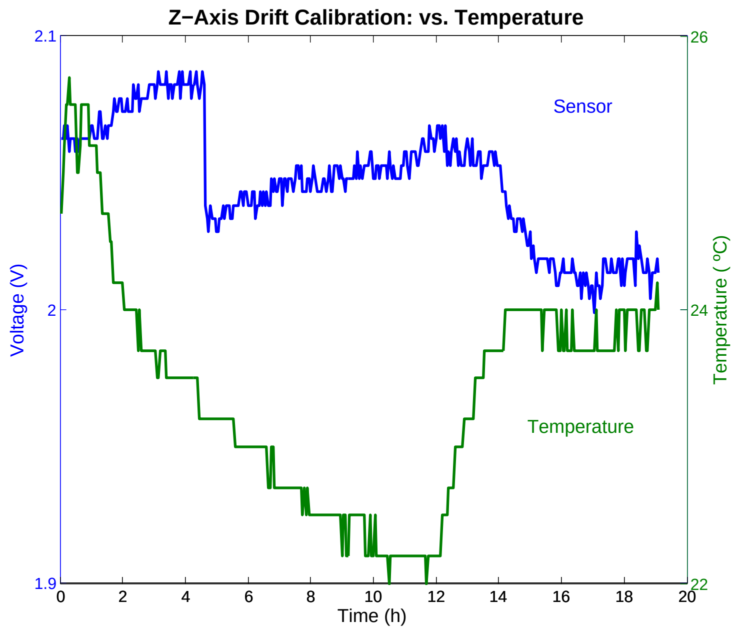


Figure 4.13: Drift calibration testing on z-axis, related to temperature.

wind tunnel testing indicated. Instead, in a box of stagnant air, the sensor interpreted the air speed to be at least 3 m/s, according to the calibration from previous test. Obviously, this is incorrect, but it is unclear if this is a result of the high humidity or not, as the humidity sensor did not give accurate data. It could also be due to mechanical disturbance as the sensor was very sensitive to handling.

Over the entire test, the sensor drifted 0.088 V, which corresponds to approximately 0.1 m/s according to the calibration of Fig. 4.11, although offset by 3.0 m/s according to the same calibration. Fig. 4.13 shows the drift following temperature, with the exception of a disturbance at Hour 4 of unknown nature. However, this glitch is characteristic of the mechanical instability that this sensor exhibited.

Chapter 5

Conclusion

5.1 Sensor Network

Overall, the temperature and humidity data from the sensor network paint an accurate and revealing picture of the immediate conditions within the sensor network. The data gathered are reasonable values for interior building temperatures, and the trends they portray follow expectations and geometry. Nodes close to the window but also near air outlets for the HVAC system have lower temperatures than nodes close to the window and far from air outlets. The sensor nodes can create temperature gradient contour maps that accurately depict the conditions in the net in real-time, which was the goal of this project. Figures 4.3 and 4.4 show that our spacial sampling was adequate for monitoring temperature and humidity in the E14 atrium.

Furthermore, the readings from these sensors are sufficient to draw conclusions about the performance of the HVAC system of the building in relation to the conditions outside the building. In the future, this data can be used to more accurately characterize the performance of active, installed HVAC systems in constructed buildings, which can be used to determine the energy consumption of the system and other useful information.

5.2 Airflow Sensor

From these results, there are fundamental problems with the design of the airflow sensors. They are relatively low-cost in terms of components and fulfill the low-power requirements; however, they are complicated to assemble and lack the robustness and stability needed for meaningful measurement. The theory behind them, of relating strain to Bernoulli's equation for air velocity, states that they should work. In practice, the strain gauges may not be the right sensor type. They cannot measure the low velocities characteristic of the types of airflow most commonly seen indoors. Furthermore, there is a significant amount of drift related to the sensors, meaning that it is incredibly difficult to calibrate even one axis. Balsa wood also proved to be too dense for low airflow measurements, and proved to be too dependent on the gravity vector.

5.3 Future Work

There are still two main problems with the airflow sensors: power and stability. The strain gauges solve the former, the propeller meets the latter. One option with the strain gauge sensor is to redesign the circuit to ensure greater stability and eliminate drift. However, this requires tuning the gain on the op-amps just right so as to filter out noise while retaining actual data; the trick is determining what constitutes "noise" at low frequencies. Additionally, the mechanism does not work as desired at the range of airflow velocities required.

In contrast, the Extech meter used in the calibration itself was much more accurate and stable. The propeller blade mechanism worked better than the strain gauges; however, while the airflow sensors allowed for measurement of three simultaneous axes, the flowmeter is strictly unidirectional. It can only measure parallel flow, which necessitates a triad of orthogonal devices. Additionally, the flowmeter used is heavier and uses much more power than the airflow sensors, although proper duty cycling and exploitation of interrupts or low-power timers could alleviate this. While the Extech meter had a minimum of 0.1 m/s, further work in the design of propeller-based meters can possibly give smaller measurements. Therefore, it is strongly recommended from these findings that a solution other than the strain gauge airflow sensors or propeller-based flowmeters, as they currently exist, be utilized in the development of low-power sensors for the purpose of measuring low air velocities. Instead, solutions might include developing a low-power adaptation of a propeller triad, a sensitive differential pressure pitot tube capable of measuring low velocities, or powering the sensors externally and using hot-wire anemometry.

Bibliography

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393 – 422, 2002.

[2] Mark Feldmeier. *Personalized Building Comfort Control.* PhD thesis, Massachusetts Instittute of Technology, Media Lab, September 2009.

[3] Seapahn Meguerdichian, Farinaz Koushanfar, Miodrag Potkonjak, and Mani B. Srivastava. Coverage problems in wireless ad-hoc sensor networks. *IEEE Infocom*, Vol 3:1380–1387, 2001.

[4] Weather Underground. Daily history. Retrieved electronically from web. http://www.wunderground.com/history/airport/KBOS/2010/3/28/DailyHistory.html.

[5] Weather Underground. Daily history. Retrieved electronically from web. http://www.wunderground.com/history/airport/KBOS/2010/3/29/DailyHistory.html.

[6] Frank M. White. *Fluid Mechanics, Sixth Edition.* McGraw Hill, sixth edition edition, 2008.

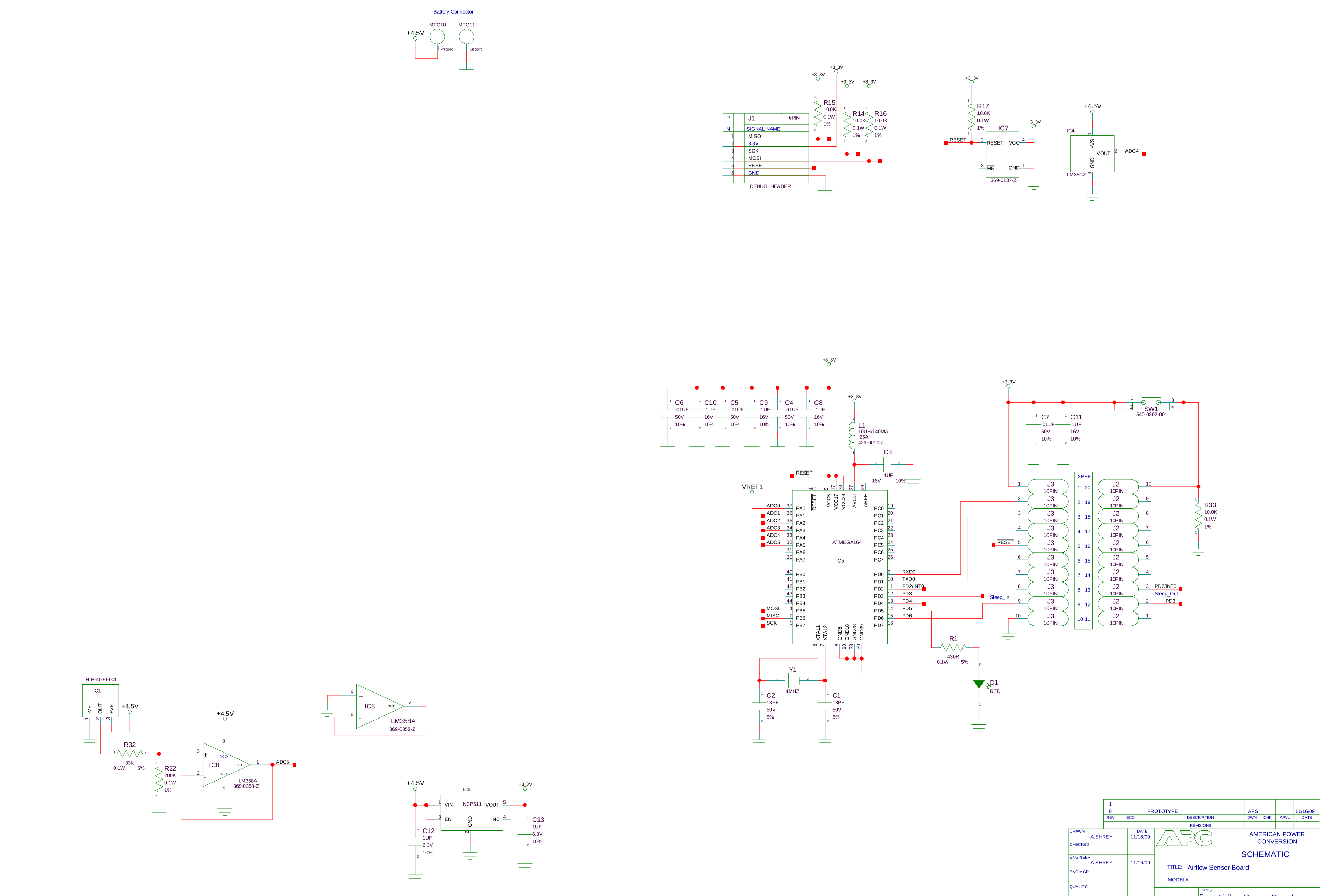
[7] Zhiqiang Zhai, Qingyan Chen, Philip Haves, and Joseph H. Klems. On approaches to couple energy simulation and computational fluid dynamics programs. *Building and Environment*, 37(8-9):857 – 864, 2002.

Appendix A

PCB Layout and Schematics

1. TH sensor layout
2. Airflow sensor layout, v1.0
3. Airflow sensor layout, v2.0

NOTICE OF PROPRIETARY PROPERTY: THIS DOCUMENT AND THE INFORMATIO N CONTAINED IN IT ARE THE PROPRIETARY PROPERTY OF AMERICAN POWER CONVERSION COMPANY. IT MAY NOT BE COPIED OR USE D IN ANY MANNER NOR MAY ANY OF THE INFORMATION IN OR UPON IT BE USED FOR ANY PURPOSE WITOUT THE EXPRESS WRITTEN CONSENT 6OF AN AUTHORIZED AGENT OF AMERICAN POWER CONVERSION COMPANY.



NOTE: ALL VALUES ARE FOR REFERENCE ONLY. SEE BOM FOR ACTUAL VALUES AND POPULATIONS.

REV	ECO	DESCRIPTION	DWN	CHK	APVL	DATE
1						
0		PROTOTYPE	APS			11/16/09

REVISIONS

		DATE
DRAWN	A.SHREY	11/16/09
CHECKED		
ENGINEER	A.SHREY	11/16/09
ENG.MGR.		
QUALITY:		
PRODUCTION		

APC
AMERICAN POWER CONVERSION

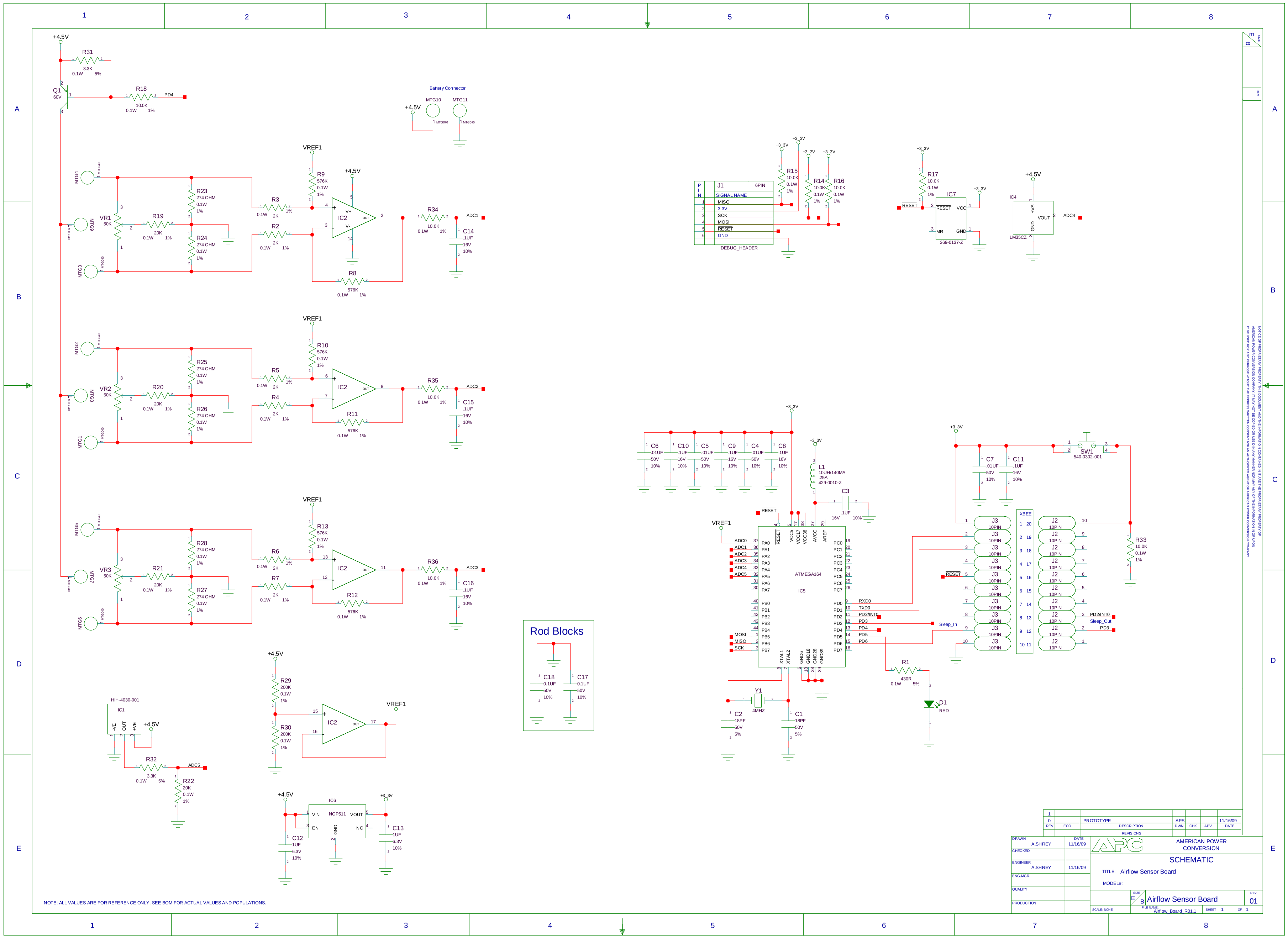
SCHEMATIC

TITLE: Airflow Sensor Board

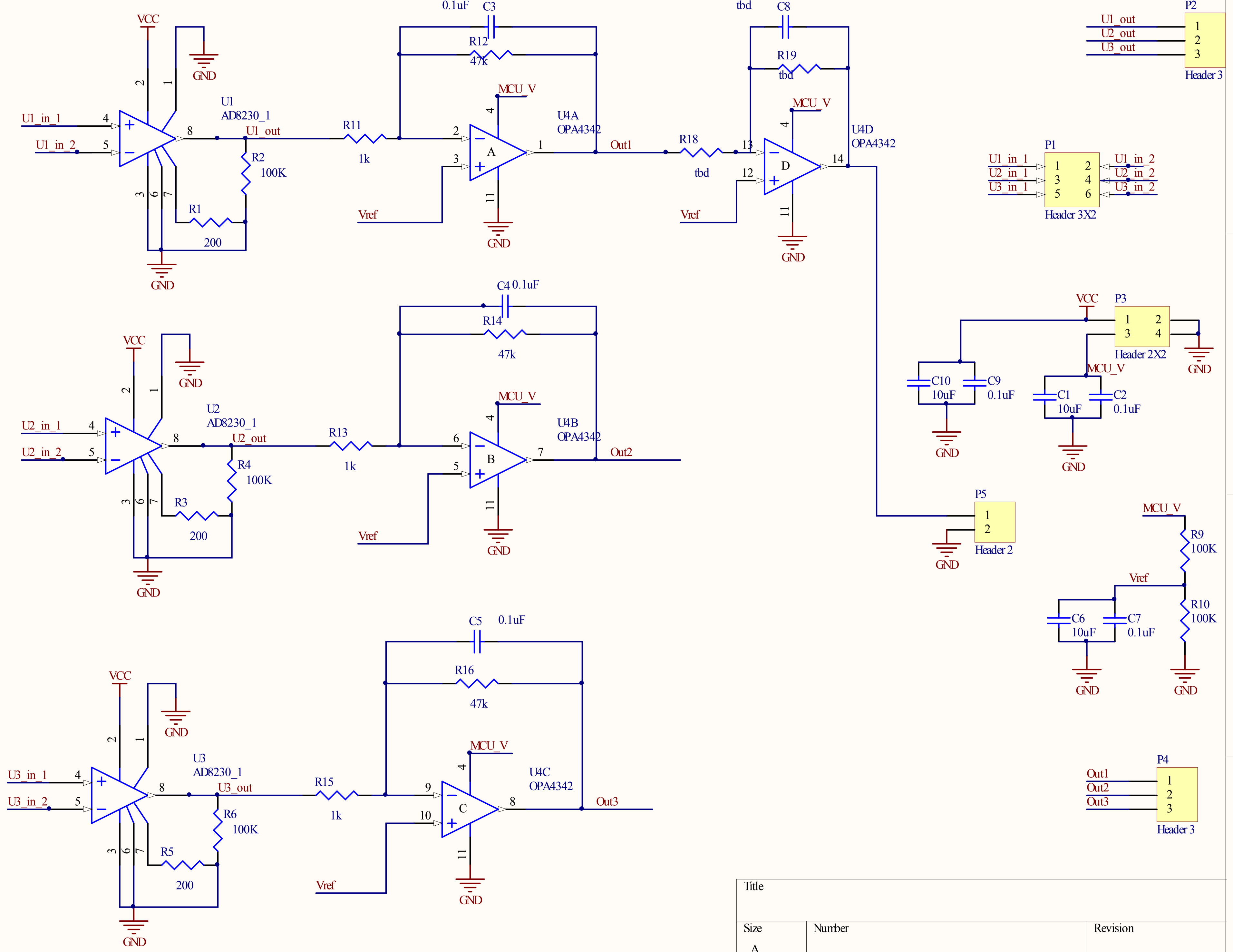
MODEL#:

SIZE E B Airflow Sensor Board REV 01

SCALE: NONE FILE NAME: Airflow_Board_R01.1 SHEET 1 OF 1



Battery Connector
MTG10
MTG11
+4.5V
+3_3V
R31 3.3K 0.1W 5%
Q1 60V
R18 10.0K 0.1W 1%
PD4
VREF1
R9 576K 0.1W 1%
R10 576K 0.1W 1%
R13 576K 0.1W 1%
R3 2K 0.1W 1%
R2 2K 0.1W 1%
R5 2K 0.1W 1%
R4 2K 0.1W 1%
R6 2K 0.1W 1%
R7 2K 0.1W 1%
R8 576K 0.1W 1%
R11 576K 0.1W 1%
R12 576K 0.1W 1%
R34 10.0K 0.1W 1%
R35 10.0K 0.1W 1%
R36 10.0K 0.1W 1%
C14 1UF 16V 10%
C15 1UF 16V 10%
C16 1UF 16V 10%
ADC1
ADC2
ADC3
ADC4
ADC5
IC2
V+
V-
OUT
R23 274 OHM 0.1W 1%
R24 274 OHM 0.1W 1%
R25 274 OHM 0.1W 1%
R26 274 OHM 0.1W 1%
R28 274 OHM 0.1W 1%
R27 274 OHM 0.1W 1%
R19 20K 0.1W 1%
R20 20K 0.1W 1%
R21 20K 0.1W 1%
VR1 50K
VR2 50K
VR3 50K
MTG1
MTG2
MTG3
MTG4
MTG5
MTG6
MTG7
MTG8
MTG9
MTG040
MTG070
HIH-4030-001
IC1
-VE
OUT
+VE
R32 3.3K 0.1W 5%
R22 20K 0.1W 1%
R29 200K 0.1W 1%
R30 200K 0.1W 1%
IC6
NCP511
VIN
EN
GND
VOUT
NC
C12 1UF 6.3V 10%
C13 1UF 6.3V 10%
Rod Blocks
C18 0.1UF 50V 10%
C17 0.1UF 50V 10%
J1
6PIN
PIN
SIGNAL NAME
1 MISO
2 3.3V
3 SCK
4 MOSI
5 RESET
6 GND
DEBUG_HEADER
R15 10.0K 0.1W 1%
R14 10.0K 0.1W 1%
R16 10.0K 0.1W 1%
R17 10.0K 0.1W 1%
IC7
RESET
VCC
MR
GND
369-0137-Z
IC4
+VS
VOUT
GND
LM35CZ
C6 .01UF 50V 10%
C10 .1UF 16V 10%
C5 .01UF 50V 10%
C9 .1UF 16V 10%
C4 .01UF 50V 10%
C8 .1UF 16V 10%
L1 10UH/140MA .25A 429-0010-Z
C3 .1UF 16V 10%
ATMEGA164
IC5
PA0 PA1 PA2 PA3 PA4 PA5 PA6 PA7
PB0 PB1 PB2 PB3 PB4 PB5 PB6 PB7
PC0 PC1 PC2 PC3 PC4 PC5 PC6 PC7
PD0 PD1 PD2 PD3 PD4 PD5 PD6 PD7
VCC5 VCC17 VCC38 AVCC AREF
XTAL1 XTAL2 GND6 GND18 GND28 GND39
ADC0
MOSI
MISO
SCK
RXD0
TXD0
PD2/INT0
PD3
PD5
PD6
Sleep_In
Sleep_Out
Y1 4MHZ
C2 18PF 50V 5%
C1 18PF 50V 5%
R1 430R 0.1W 5%
D1 RED
C7 .01UF 50V 10%
C11 .1UF 16V 10%
SW1 540-0302-001
J3 10PIN
J2 10PIN
XBEE
R33 10.0K 0.1W 1%
NOTE: ALL VALUES ARE FOR REFERENCE ONLY. SEE BOM FOR ACTUAL VALUES AND POPULATIONS.
NOTICE OF PROPRIETARY PROPERTY: THIS DOCUMENT AND THE INFORMATION CONTAINED IN IT ARE THE PROPRIETARY PROPERTY OF AMERICAN POWER CONVERSION COMPANY. IT MAY NOT BE COPIED OR USED IN ANY MANNER NOR MAY ANY OF THE INFORMATION IN OR UPON IT BE USED FOR ANY PURPOSE WITHOUT THE EXPRESS WRITTEN CONSENT OF AN AUTHORIZED AGENT OF AMERICAN POWER CONVERSION COMPANY.
REV 0 ECO PROTOTYPE APS 11/16/09
DESCRIPTION
REVISIONS
DWN CHK APVL DATE
DRAWN A.SHREY 11/16/09
CHECKED
ENGINEER A.SHREY 11/16/09
ENG MGR
QUALITY
PRODUCTION
APC
AMERICAN POWER CONVERSION
SCHEMATIC
TITLE: Airflow Sensor Board
MODEL#:
SIZE E B
Airflow Sensor Board
REV 01
SCALE: NONE
FILE NAME Airflow_Board_R01.1
SHEET 1 OF 1



Title		
Size A	Number	Revision
Date: 4/25/2010		Sheet of
File: F:\Documents and Settings\..\airflow.SchDoc		Drawn By:

Appendix B

MATLAB Evaluation Code

The contents of this appendix include all MATLAB functions written to evaluate the raw data from the SQL database set up to receive data from the sensor nodes. All code was written in MATLAB R2009a, Version 7.4.0.287.

The format of the data from the sensor nodes is

Node Date Temp %Humidity Batt u v w

in .txt files. The date is in the format dd-mm-yyyy HH:MM:SS, and u, v, and w refer to air speed measurements, if any, in the x, y, and z directions respectively. Not all sensors measure air speed.

```
function do_thesis


nuke_it = questdlg('Run What?', 'My god, it is full of data', ...
    'Individual Sensor Plots', 'Batch Sensor Plots', 'Sensor Net', ...
    'Sensor Net');


switch nuke_it
    case 'Individual Sensor Plots'
        sensor_plot;
        return
    case 'Sensor Net'
        sensor_net_fin;
        return
    case 'Batch Sensor Plots'
        batch_sensor_plot;
end
```

```
function sensor_plot

%dialog box to determine airflow sensor or temperature sensor data

sensor_choice = questdlg('Choose sensor type:', 'Sensor Type', ...
    'Airflow', 'TH', 'Cancel', 'Cancel');

switch sensor_choice
    case 'Airflow'
        airflow_sensor_plot;
        return
    case 'TH'
        TH_sensor_plot;
        return
    case 'Cancel'
        disp('End');
end
```

```
function airflow_sensor_plot

%import and load data from specified file
%GUI import

file_name = uigetfile('*.txt', 'Select the txt file data');

%file_name = input('Input name of file to import. ', 's');
%F = xlsread(file_name, 'B2:H2');

if file_name == 0
    disp('End');
    return
else
    F = importdata(file_name, '\t', 1);
end

d = F.data(:,1); %date
T = F.data(:,3);
H = F.data(:,4);
B = F.data(:,5);
%L = F.data(:,3);
X = F.data(:,6);
Y = F.data(:,7);
Z = F.data(:,8);
```

```
%user input for data specification

prompt_file_spec = {'Trial Date', 'Node'};
dlg_datenode_title = 'Input for Data Specification';
dn_num_lines = 1;
dn_defAns = {'null', 'null'};
date_node = inputdlg(prompt_file_spec, dlg_datenode_title, ...
    dn_num_lines, dn_defAns);

date = date_node{1};
node = date_node{2};

dirname = [date '_' node];

%Choose unit of time to be displayed on the plot.  Then input the time
%interval the sensor reported data back to the database, in seconds.

tunit_choice = questdlg('What unit of time?', 'Time Select', 'Day', ...
    'Hour', 'Min', 'Min');

switch tunit_choice
    case 'Hour'
        tu = 3600;
        tlabel = 'h';
    case 'Min'
        tu = 60;
        tlabel = 'min';
    otherwise
        tu = 3600*24;
        tlabel = 'd';
end



prompt_TI_spec = {'Time Interval in seconds?'};
dlg_TI_title = 'Time Interval';
TInum_lines = 1;
TIdefAns = {'30'};
TI_string = inputdlg(prompt_TI_spec, dlg_TI_title, TInum_lines, TIdefAns);

TI = str2double(TI_string{1});

n = 1:1:length(X);
ts = n.*TI;

t= ts./tu;

%Data came in reversed (first sample at bottom of stack).  Flipped for
%convience.
Xrev = flipud(X);
Yrev = flipud(Y);
Zrev = flipud(Z);
Trev = flipud(T);
Hrev = flipud(H);
Brev = flipud(B);
```

```
%Declare initial value.

baseX = Xrev(1);

baseY = Yrev(1);

baseZ = Zrev(1);


%Find delta

del_X = Xrev - baseX;

del_Y = Yrev - baseY;

del_Z = Zrev - baseZ;


%Correct axis flow for Voltage reference

Xfix = Xrev + B;

Yfix = Yrev + B;

Zfix = Zrev + B;


%Find corrected range; will be outputted in Command Window

rX = range(Xfix);

disp(['Range of corrected X is ', num2str(rX)]);


rY = range(Yfix);

disp(['Range of corrected Y is ', num2str(rY)]);


rZ = range(Zfix);

disp(['Range of corrected Z is ', num2str(rZ)]);


rT = range(T);
```

```
disp(['Range of Temperature is ', num2str(rT)]);

rH = range(H);
disp(['Range of %Humidity is ', num2str(rH)]);

rB = range(B);
disp(['Range of Battery is ', num2str(rB)]);

rHeader = {'Corrected X' 'Corrected Y' 'Corrected Z' 'Temp' '%Humid' ...
    'Batt'};
rData = {num2str(rX) num2str(rY) num2str(rZ) num2str(rT) num2str(rH)
    num2str(rB)};

range_data = [rHeader; rData];

%made new directory for files
mkdir(dirname);

%Plots saved in new directory as .pdf and .fig files

cd(['./' dirname]);

%Ranges saved in .txt file
xlswrite('ranges', range_data);

hx = plot(t, del_X);
title(['\bf', date, ' ', 'Node: ', node, ...
    ', Variance of raw X-axis v Time'], 'FontSize', 14);
xlabel(['Time (',tlabel,')'], 'FontSize', 12);
ylabel('\Delta X', 'FontSize', 12);
saveas(hx, 'x_axis', 'pdf');
saveas(hx, 'x_axis', 'fig');

hXfix = plot(t, Xfix);
title(['\bf', date, ' ', 'Node: ', node, ...
    ', Variance of X-axis v Time Corrected for Vref'], 'FontSize', 14);
xlabel(['Time (',tlabel,')'], 'FontSize', 12);
ylabel('X', 'FontSize', 12);
saveas(hXfix, 'x_axis_vref_fix', 'pdf');
saveas(hXfix, 'x_axis_vref_fix', 'fig');

hy = plot(t, del_Y);
title(['\bf', date, ' ', 'Node: ', node, ...
    ', Variance raw Y-axis v Time'], 'FontSize', 14);
xlabel(['Time (',tlabel,')'], 'FontSize', 12);
ylabel('\Delta Y', 'FontSize', 12);
saveas(hy, 'y_axis', 'pdf');
saveas(hy, 'y_axis', 'fig');

hYfix = plot(t, Yfix);
title(['\bf', date, ' ', 'Node: ', node, ...
    ', Variance of Y-axis v Time Corrected for Vref'], 'FontSize', 14);
xlabel(['Time (',tlabel,')'], 'FontSize', 12);
ylabel('Y', 'FontSize', 12);
```

```
saveas(hYfix, 'y_axis_vref_fix', 'pdf');
saveas(hYfix, 'y_axis_vref_fix', 'fig');

hz = plot(t, del_Z);
title(['\bf', date, ' ', 'Node: ', node, ...
    ', Variance raw Z-axis v Time'], 'FontSize', 14);
xlabel(['Time (',tlabel,')'], 'FontSize', 12);
ylabel('\Delta Z', 'FontSize', 12);
saveas(hz, 'z_axis', 'pdf');
saveas(hz, 'z_axis', 'fig');

hZfix = plot(t, Zfix);
title(['\bf', date, ' ', 'Node: ', node, ...
    ', Variance of Z-axis v Time Corrected for Vref'], 'FontSize', 14);
xlabel(['Time (',tlabel,')'], 'FontSize', 12);
ylabel('Z', 'FontSize', 12);
saveas(hZfix, 'z_axis_vref_fix', 'pdf');
saveas(hZfix, 'z_axis_vref_fix', 'fig');

hT = plot(t, Trev);
title(['\bf', date, ' ', 'Node: ', node, ...
    ', Temperature v Time'], 'FontSize', 14);
xlabel(['Time (',tlabel,')'], 'FontSize', 12);
ylabel(['Temperature (\circ C)', 'FontSize', 12]);
saveas(hT, 'temp', 'pdf');
saveas(hT, 'temp', 'fig');
```

```
hH = plot(t, Hrev);
title(['\bf', date, ' ', 'Node: ', node, ...
    ', Humidity v Time'], 'FontSize', 14);
xlabel(['Time (',tlabel,')'], 'FontSize', 12);
ylabel('Humidity (%)', 'FontSize', 12);
saveas(hH, 'humidity', 'pdf');
saveas(hH, 'humidity', 'fig');

hB = plot(t, Brev);
title(['\bf', date, ' ', 'Node: ', node, ...
    ', Battery v Time'], 'FontSize', 14);
xlabel(['Time (',tlabel,')'], 'FontSize', 12);
ylabel('Battery Charge', 'FontSize', 12);
saveas(hB, 'battery', 'pdf');
saveas(hB, 'battery', 'fig');
```

```
function TH_sensor_plot

%import and load data from specified file
%GUI import
%uiimport('-file');

file_name = uigetfile('*.txt', 'Select the txt file data');

%file_name = input('Input name of file to import. ', 's');

%F = xlsread(file_name, 'B2:H2');
if file_name == 0
    disp('End');
else
    F = importdata(file_name, '\t', 1);
end

T = F.data(:,1);
H = F.data(:,2);
B = F.data(:,7);

%user input for data specification

prompt_file_spec = {'Trial Date', 'Node'};
dlg_datenode_title = 'Input for Data Specification';
dn_num_lines = 1;
dn_defAns = {'null', 'null'};
```

```
date_node = inputdlg(prompt_file_spec, dlg_datenode_title, ...
    dn_num_lines, dn_defAns);

date = date_node{1};
node = date_node{2};

dirname = [date '_' node];

%declare time interval; time samples were gathered at intervals.
%Change as needed. (variable TI)

tunit_choice = questdlg('What unit of time?', 'Time Select', 'Hour', ...
    'Min', 'Sec', 'Min');

switch tunit_choice
    case 'Hour'
        tu = 3600;
        tlabel = 'h';
    case 'Min'
        tu = 60;
        tlabel = 'min';
    otherwise
        tu = 1;
        tlabel = 's';
end

%t_unit = input('What unit of time? (hour, min, sec) ', 's');
```

```
prompt_TI_spec = {'Time Interval in seconds?'};
dlg_TI_title = 'Time Interval';
TInum_lines = 1;
TIdefAns = {'30'};
TI_string = inputdlg(prompt_TI_spec, dlg_TI_title, TInum_lines, TIdefAns);

TI = str2num(TI_string{1});

n = 1:1:length(X);
ts = n.*TI;

t= ts./tu;

%Data came in reversed (first sample at bottom of stack).  Flipped for
%convience.

Trev = flipud(T);
Hrev = flipud(H);
Brev = flipud(B);

rT = range(T);
disp(['Range of Temperature is ', num2str(rT)]);

rH = range(H);
disp(['Range of %Humidity is ', num2str(rH)]);
```

```
rB = range(B);
disp(['Range of Battery is ', num2str(rB)]);

rHeader = {'Temp' '%Humid' 'Batt'};
rData = {num2str(rT) num2str(rH) num2str(rB)};

range_data = [rHeader; rData];

%made new directory for files
mkdir(dirname);

%Plots saved in new directory as .pdf and .fig files

cd(['./' dirname]);

%Ranges saved in .xls file
xlswrite('ranges', range_data);

hT = plot(t, Trev);
title(['\bf', date, ' ', 'Node: ', node, ', Temperature v Time'], ...
    'FontSize', 14);
xlabel(['Time (',tlabel,')'], 'FontSize', 12);
ylabel('Temperature (\circ C)', 'FontSize', 12);
saveas(hT, 'temp', 'pdf');
saveas(hT, 'temp', 'fig');

hH = plot(t, Hrev);
```

```
title(['\bf', date, ' ', 'Node: ', node, ', Humidity v Time'], ...
    'FontSize', 14);
xlabel(['Time (',tlabel,')', 'FontSize', 12]);
ylabel('Humidity (%)', 'FontSize', 12);
saveas(hH, 'humidity', 'pdf');
saveas(hH, 'humidity', 'fig');

hB = plot(t, Brev);
title(['\bf', date, ' ', 'Node: ', node, ', Battery v Time'], ...
    'FontSize', 14);
xlabel(['Time (',tlabel,')'], 'FontSize', 12);
ylabel('Battery Charge', 'FontSize', 12);
saveas(hB, 'battery', 'pdf');
saveas(hB, 'battery', 'fig');
```

```
function data = batch_sensor_plot

%Like the individual plot function (TH_sensor_plot), this function plots
%the temperature, humidity, and battery of the node.  Unlike
%TH_sensor_plot, this function is designed to take all node entries in a
%directory and batch process them, rather than do an individual plot.

dirname = uigetdir;

disp(dirname); %test point

pfs = {'Trial Date Start (form mmddyyyy)'};
dlg_dn_title = 'Input for Data Specification';
dn_num_lines = 1;
dn_defAns = {'null'};
test_date = inputdlg(pfs, dlg_dn_title, dn_num_lines, dn_defAns);

new_dir = [test_date{1} '_plots'];
mkdir(new_dir);

%Node Locations

% positions = [300 250 1; 350 260 1; 400 270 1; 10 510 2; 470 510 2;
%      0 280 2; 470 10 2; 80 80 2; 80 160 2; 80 240 2; 80 320 2; 80 400 2;
%      240 80 2; 240 160 2; 240 240 2; 240 320 2; 240 400 2; 400 80 2;
%      400 160 2; 400 240 2; 400 320 2; 400 400 2; 50 130 2; 50 260 2;
%      50 370 2; 160 140 2; 160 300 2; 320 140 2; 320 300 2; 80 80 3;
%     80 160 3; 80 240 3; 80 320 3; 80 400 3; 240 80 3; 240 160 3;
%     240 240 3; 240 320 3; 240 400 3; 400 80 3; 400 160 3; 400 240 3;
%     400 320 3; 400 400 3; 80 50 3; 240 50 3; 400 50 3; 180 30 3;
%     320 30 3; 0 480 4; 80 80 4; 80 160 4; 80 240 4; 80 320 4; 80 400 4;
%     240 80 4; 240 160 4; 240 240 4; 240 320 4; 240 400 4; 400 80 4;
%     400 160 4; 400 240 4; 400 320 4; 400 400 4; 240 50 4; 400 50 4;
%     160 140 4; 160 300 4; 320 140 4; 320 300 4; 240 350 5; 10 350 5;
%     100 480 5; 260 200 5];
%
% node_id = {'cbb', 'c32', '15b', 'c25', 'c58*', 'cd2', 'ca3*', 'cc3', ...
%     'cc0*', 'cbd', 'caa', 'ccf', '30a', 'd04', 'f92', 'f8e', 'fd0', ...
%     'f93', 'fbc', 'cfd', 'caf', 'f9e', 'fdf', 'cd7', 'cf8', '971', ...
%     '3cf', '3c4', '329', 'ce5', 'cf1', 'ce2', 'cd3', 'cd4', '10f', ...
%     'f94', 'fab', 'f9b', 'fc6', 'fbf', 'f91', '312', 'fc8', '2df', ...
%     '3bc', 'a8c', '996', '3cd', '3d9', '1a9', 'cd2', 'cad', 'cbb', ...
%     'x74', '99d', 'cbc', 'fce', 'cb8', 'cf2', 'cde', 'cb9', 'fb5', ...
%     'fb4', 'f90', 'f99', 'fe2', '3c6', '310', '972', 'f96', '3db', ...
%     '191', 'c41', '107', 'cc1'};

positions = [10 510 2; 470 510 2; 470 10 2; 80 80 2; 80 160 2; 80 240 2;
    80 320 2; 80 400 2; 240 80 2; 240 240 2; 240 320 2; 240 400 2;
    400 80 2; 400 160 2; 400 240 2; 400 320 2; 400 400 2; 50 130 2;
    50 260 2; 50 370 2; 80 80 3; 80 160 3; 80 240 3; 80 320 3; 240 80 3;
    240 160 3; 240 320 3; 240 400 3; 400 80 3; 400 160 3;400 320 3;
    400 400 3; 80 50 3; 240 50 3; 400 50 3; 180 30 3; 0 480 4; 80 80 4;
    300 250 1; 80 400 3];
```

```
node_id = {'c25', 'c58', 'ca3', 'cc3', 'cc0', 'cbd', 'caa', 'ccf', ...
    '30a', 'f92', 'f8e', 'fd0', 'f93', 'fbc', 'cfd', 'caf', 'f9e', ...
    'fdf', 'cd7', 'cf8', 'ce5', 'cf1', 'ce2', 'cd3', '1f0', 'f94', ...
    'f9b', 'fc6', 'fbf', 'f91', 'fc8', '2df', '3bc', 'a8c', '996', ...
    '3cd', '1a9', 'cd2', 'cbb', 'cd4'};

cd(new_dir);

files = dir(dirname);
j = 0;

%Do all things for each file before moving onto the next one.

for i=1:length(files)
    filename = [dirname '/' files(i).name];
    if length(filename) > 4 && strcmp(filename(end-3:end), '.txt')
        F = importdata(filename, '\t', 0);
        j = j+1;

        id = files(i).name([1,2,3]);
        node_index = find(strcmp(id, node_id));
        if length(node_index) == 0
            disp(id)
        end
        data(j).x = positions(node_index, 1);
        data(j).y = positions(node_index, 2);
        data(j).z = positions(node_index, 3);
```

```
d = datenum(flipud(F.textdata(:,2)), 'dd-mm-yyyy HH:MM:SS'); %date

data(j).node = flipud(F.textdata(:,1))
data(j).D = d+linspace(0, 10/86400, length(d))';
data(j).T = flipud(F.data(:,1)); %temp
data(j).H = flipud(F.data(:,2)); %humidity
data(j).B = flipud(F.data(:,3)); %battery


data(j).time = 1:1:length(data(j).D);

t = (data(j).time*3)/(60*24);

%determine range on data
rT = range(data(j).T);
rH = range(data(j).H);
rB = range(data(j).B);
rHeader = {'Temp' '%Humid' 'Batt'};
rData = {num2str(rT) num2str(rH) num2str(rB)};

range_data = [rHeader; rData];

nd = data(j).node(1);

%creates specific directory for the node data
newdir = sprintf('%s', nd{:});
```

```
mkdir(newdir);
cd(['./' newdir]);

xlswrite('ranges', range_data);

%plots
hT = plot(t, data(j).T);
title(['\bf Node:' nd 'Temperature v Time'], 'FontSize', 14);
xlabel('Time (Days)', 'FontSize', 12);
ylabel('Temperature (\circ C)', 'FontSize', 12);
saveas(hT, 'temp', 'pdf');
saveas(hT, 'temp', 'fig');

hH = plot(t, data(j).H);
title(['\bf Node: ' nd 'Humidity v Time'], 'FontSize', 14);
xlabel('Time (Days)', 'FontSize', 12);
ylabel('Humidity (%)', 'FontSize', 12);
saveas(hH, 'humidity', 'pdf');
saveas(hH, 'humidity', 'fig');

hB = plot(t, data(j).B);
title(['\bf Node: ' nd 'Battery v Time'], 'FontSize', 14);
xlabel('Time (Days)', 'FontSize', 12);
ylabel('Battery Charge', 'FontSize', 12);
saveas(hB, 'battery', 'pdf');
saveas(hB, 'battery', 'fig');
```

```
        %return to top-level before going onto the next node
        cd('..');


    end
end
```

```
function data = sensor_net_fin

%Take all the log files.  Interpret the filenames so that the correct node
%is noted.  Use structs to associate data with the correct node.
%associates data_number to node? Get variables from node log. Associate all
%data to its appropriate node.  Send data to another function to generate
%animated graph.

dirname = uigetdir;
disp(dirname); %test point

%Node Locations

%Complete Listing
% positions = [300 250 1; 350 260 1; 400 270 1; 10 510 2; 470 510 2;
%      0 280 2; 470 10 2; 80 80 2; 80 160 2; 80 240 2; 80 320 2; 80 400 2;
%      240 80 2; 240 160 2; 240 240 2; 240 320 2; 240 400 2; 400 80 2;
%      400 160 2; 400 240 2; 400 320 2; 400 400 2; 50 130 2; 50 260 2;
%      50 370 2; 160 140 2; 160 300 2; 320 140 2; 320 300 2; 80 80 3;
%      80 160 3; 80 240 3; 80 320 3; 80 400 3; 240 80 3; 240 160 3;
%      240 240 3; 240 320 3; 240 400 3; 400 80 3; 400 160 3; 400 240 3;
%      400 320 3; 400 400 3; 80 50 3; 240 50 3; 400 50 3; 180 30 3;
%      320 30 3; 0 480 4; 80 80 4; 80 160 4; 80 240 4; 80 320 4; 80 400 4;
%      240 80 4; 240 160 4; 240 240 4; 240 320 4; 240 400 4; 400 80 4;
%      400 160 4; 400 240 4; 400 320 4; 400 400 4; 240 50 4; 400 50 4;
%      160 140 4; 160 300 4; 320 140 4; 320 300 4; 240 350 5; 10 350 5;
%      100 480 5; 260 200 5];
```

```
%
% node_id = {'cbb', 'c32', '15b', 'c25', 'c58*', 'cd2', 'ca3*', 'cc3', ...
%     'cc0*', 'cbd', 'caa', 'ccf', '30a', 'd04', 'f92', 'f8e', 'fd0', ...
%     'f93', 'fbc', 'cfd', 'caf', 'f9e', 'fdf', 'cd7', 'cf8', '971', ...
%     '3cf', '3c4', '329', 'ce5', 'cf1', 'ce2', 'cd3', 'cd4', '10f', ...
%     'f94', 'fab', 'f9b', 'fc6', 'fbf', 'f91', '312', 'fc8', '2df', ...
%     '3bc', 'a8c', '996', '3cd', '3d9', '1a9', 'cd2', 'cad', 'cbb', ...
%     'x74', '99d', 'cbc', 'fce', 'cb8', 'cf2', 'cde', 'cb9', 'fb5', ...
%     'fb4', 'f90', 'f99', 'fe2', '3c6', '310', '972', 'f96', '3db', ...
%     '191', 'c41', '107', 'cc1'};

%Nodes Used

positions = [80 400 2; 80 320 2; 80 240 2;
    80 160 2; 80 80 2; 240 400 2; 240 240 2; 240 160 2; 240 80 2;
    400 400 2; 400 320 2; 400 240 2; 400 160 2; 400 80 2; 250 50 2;
    240 450 2; 260 450 2; 80 400 3; 80 320 3; 80 240 3; 80 160 3; 240 400 3;
    240 320 3; 240 160 3; 240 80 3; 400 400 3; 400 320 3;400 160 3;
    400 80 3; 80 450 3; 240 450 3; 400 450 3; 80 400 4;
    80 240 4; 80 80 3; 80 320 4; 80 160 4; 80 80 4; 240 400 4;
    240 320 4; 240 240 4; 240 160 4; 240 80 4; 400 400 4; 400 320 4;
    400 240 4; 400 160 4; 400 80 4; 240 30 4; 400 30 4; 400 240 3;
    240 240 3; 240 320 2];

node_id = {'cc3', 'cc0', 'cbd', 'caa', 'ccf', ...
    '30a', 'f92', 'f8e', 'fd0', 'f93', 'fbc', 'cfd', 'caf', 'f9e', ...
    'fdf', 'cd7', 'cf8', 'ce5', 'cf1', 'ce2', 'cd3', '1f0', 'f94', ...
    'f9b', 'fc6', 'fbf', 'f91', 'fc8', '2df', '3bc', 'a8c', '996', ...
    'cd2', 'cbb', 'cd4', 'cad', '074', '99d', 'cbc', 'fce', ...
    'cb8', 'cf2', 'cde', 'cb9', 'fb5', 'fb4', 'f90', 'f99', ...
    'fe2', '3c6', '312', 'fab', 'd04'};



files = dir(dirname);
j = 0;



for i=1:length(files)
    filename = [dirname '/' files(i).name];
    if length(filename) > 4 && strcmp(filename(end-3:end), '.txt')
        F = importdata(filename, '\t', 0);
        j = j+1;

        %check that all nodes listed above exist in the folder.  The names
        %of those that do not will be printed in the Command Window.

        id = files(i).name([1,2,3]);
        node_index = find(strcmp(id, node_id));
        if length(node_index) == 0
            disp(id)
        end
        data(j).x = positions(node_index, 1);
        data(j).y = positions(node_index, 2);
        data(j).z = positions(node_index, 3);
        d = datenum(flipud(F.textdata(:,2)), 'dd-mm-yyyy HH:MM:SS'); %date

        data(j).D = d+linspace(0, 10/86400, length(d))';
        data(j).T = flipud(F.data(:,1)); %temp
        data(j).H = flipud(F.data(:,2)); %humidity
        data(j).B = flipud(F.data(:,3)); %battery
        data(j).u = flipud(F.data(:,4));
        data(j).v = flipud(F.data(:,5));
        data(j).w = flipud(F.data(:,6));

    end
end

%Data start and end times are not uniform.  Find the largest common block
%of time between all files to analyze:

starttime = -inf;
endtime = inf;

for i = 1:length(data)
    starttime = max(data(i).D(1), starttime);
    endtime = min(data(i).D(end), endtime);
end

time = starttime:600/86400:endtime;
```

```
%Data for each time does not necessarily exist.  Therefore, data is
%interpolated in order to make the plot.

for i = 1:length(data)
    data(i).t = time';
    data(i).T = interp1(data(i).D, data(i).T, time)';
    data(i).H = interp1(data(i).D, data(i).H, time)';
    data(i).B = interp1(data(i).D, data(i).B, time)';
    data(i).u = interp1(data(i).D, data(i).u, time)';
    data(i).v = interp1(data(i).D, data(i).v, time)';
    data(i).w = interp1(data(i).D, data(i).w, time)';
end

what_data = questdlg('What type of data to analyze?', 'Data Selection', ...
    'Temperature', 'Humidity', 'Battery Life', 'Temperature');

switch what_data
    case 'Temperature'
        type = 'temp';
        das_blinkenlights(data, type);
        return
    case 'Humidity';
        type = 'humi';
        das_blinkenlights(data, type);
        return
    case 'Battery Life'
        type = 'batt';
        das_blinkenlights(data, type);


end
```

```
function das_blinkenlights(d, t)

%Takes in data generated by sensor_net and plots it on a 3D scatter-point
%graph.  Each node is represented as a point that changes color with time
%based on the readings from the node log files.

%Uses addframe for creating .avi files of data; frames are added as they
%are generated to an .avi file.  The file is completed once all time
%interval entries are finished.

%Time stamp format:
%title(['\bf' data(1).time(i) ': Temperature/Humidity/Battery']);

%Video files must be compressed externally, as this was written on a 64-bit
%system and 64-bit versions of the standard codecs for MATLAB do not exist.
%Use something like VirtualDub to compress video files.
%Suggested Codec: Cinepak

pfs = {'Trial Date Start (form mmddyyyy)'};
dlg_dn_title = 'Input for Data Specification';
dn_num_lines = 1;
dn_defAns = {'null'};
test_date = inputdlg(pfs, dlg_dn_title, dn_num_lines, dn_defAns);

new_dir = [test_date{1} '_animation'];
mkdir(new_dir);
cd(new_dir);
```

```
xline1 = ones(1,6)*80;

yline1 = 50:70:400;

zline1 = ones(1,6)*2;

xline2 = ones(1,6)*240;

yline2 = 50:70:400;

zline2 = ones(1,6)*2;

xline3 = ones(1,6)*400;

yline3 = 50:70:400;

zline3 = ones(1,6)*2;

xline4 = ones(1,6)*80;

yline4 = 50:70:400;

zline4 = ones(1,6)*3;

xline5 = ones(1,6)*240;

yline5 = 50:70:400;

zline5 = ones(1,6)*3;

xline6 = ones(1,6)*400;

yline6 = 50:70:400;

zline6 = ones(1,6)*3;

xline7 = ones(1,6)*80;

yline7 = 50:70:400;

zline7 = ones(1,6)*4;

xline8 = ones(1,6)*240;

yline8 = 50:70:400;

zline8 = ones(1,6)*4;

xline9 = ones(1,6)*400;
```

```
yline9 = 50:70:400;
zline9 = ones(1,6)*4;


if strcmp(t, 'temp')
    C = [d.T];
    figure('Name', 'Temperature');
    aviobj = avifile('temperature.avi', 'compression', 'none');
    for i = 1:length(d(1).T)
        scatter3([d.x], [d.y], [d.z], 50*ones(size(d)), C(i,:), 'filled');
        title(['\bf' datestr(d(1).t(i)), ': Temperature'], 'FontSize', 14);
        colorbar('location', 'EastOutside');
        axis([0 600 0 600 0 5 15 26])
        hold on;
        plot3(xline1, yline1, zline1);
        plot3(xline2, yline2, zline2);
        plot3(xline3, yline3, zline3);
        plot3(xline5, yline5, zline5);
        plot3(xline4, yline4, zline4);
        plot3(xline6, yline6, zline6);
        plot3(xline7, yline7, zline7);
        plot3(xline8, yline8, zline8);
        plot3(xline9, yline9, zline9);
        pause(0.1);
        frame = getframe(gcf);
        aviobj = addframe(aviobj, frame);
    end
else if strcmp(t, 'humi')
```

```
    C = [d.H];
    figure('Name', '% Humidity');
    aviobj = avifile('humidity.avi', 'compression', 'none');
    for i = 1:length(d(1).T)
        scatter3([d.x], [d.y], [d.z], 50*ones(size(d)), C(i,:), ...
            'filled');
        title(['\bf' datestr(d(1).t(i)), ': Humidity'], 'FontSize', 14);
        colorbar('location', 'EastOutside');
        axis([0 600 0 600 0 5 0 80])
        hold on;
        plot3(xline1, yline1, zline1);
        plot3(xline2, yline2, zline2);
        plot3(xline3, yline3, zline3);
        plot3(xline5, yline5, zline5);
        plot3(xline4, yline4, zline4);
        plot3(xline6, yline6, zline6);
        plot3(xline7, yline7, zline7);
        plot3(xline8, yline8, zline8);
        plot3(xline9, yline9, zline9);
        pause(0.1);
        frame = getframe(gcf);
        aviobj = addframe(aviobj, frame);
    end
else
    C = [d.B];
    figure('Name', 'Battery Depletion');
    aviobj = avifile('battery.avi', 'compression', 'none');
```

```
        for i = 1:length(d(1).T)

            scatter3([d.x], [d.y], [d.z], 50*ones(size(d)), C(i,:), ...

                'filled');

            title(['\bf' datestr(d(1).t(i)), ': Battery'], 'FontSize', 14);

            colorbar('location', 'EastOutside');

            axis([0 600 0 600 0 5 200 700])

            hold on;

            plot3(xline1, yline1, zline1);

            plot3(xline2, yline2, zline2);

            plot3(xline3, yline3, zline3);

            plot3(xline5, yline5, zline5);

            plot3(xline4, yline4, zline4);

            plot3(xline6, yline6, zline6);

            plot3(xline7, yline7, zline7);

            plot3(xline8, yline8, zline8);

            plot3(xline9, yline9, zline9);

            pause(0.1);

            frame = getframe(gcf);

            aviobj = addframe(aviobj, frame);

        end


    end

end


aviobj = close(aviobj);


cd('..');
```