MONITORING OF INDOOR RELATIVE HUMIDITY LEVELS IN RESIDENTIAL DWELLINGS: A SENSOR NETWORK APPLICATION

by

Lizabeth Lee

A report submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Dr. YangQuan Chen Major Professor	Dr. Wei Ren Committee Member

Dr. Anhong Zhou
Committee Member

UTAH STATE UNIVERSITY
Logan, Utah

2008

Abstract

Monitoring of Indoor Relative Humidity Levels in Residential Dwellings: A Sensor Network Application

by

Lizabeth Lee, Master of Science

Utah State University, 2008

Major Professor: Dr. YangQuan Chen
Department: Electrical and Computer Engineering

Indoor Air Quality is an increasing concern in the world today. The mere presence of people in a building or residence can significantly alter indoor air quality. Relative humidity over the range of normal indoor temperatures (66 - 80 degrees Farenheit) has been linked both directly and indirectly to various health and structural problems. The purpose of this project was to discover whether residential dwellings might benefit from an indoor humidification system. The project consisted of the deployment of three separate sensor networks consisting of 12 tmote sky modules manufactured by the Moteiv corporation, each equipped with a temperature and humidity sensor manufactured by Sensirion. Each tmote sky module continuously transmitted the raw data readings to a base station to be processed. The lifetime of each network was approximately four days of continuous data transmission. The results verified the hypothesis that relative humidity levels have a significant affect on the indoor environment and can be linked to the health and structural problems reported by the occupants of each monitored residence. Based on the project findings residential dwellings would benefit from an indoor humidification system, given the symptoms associated with relative humidity level problems exist.

(80 pages)

Acknowledgments

I would like to thank Utah State University, Department of Electrical and Computer Engineering, Center for Self-Organizing and Intelligent Systems (CSOIS) for providing the funding and hardware for this project. I would like to thank my committee, especially Dr. YangQuan Chen, acting director of CSOIS, who always believed in me and never let me give up on myself. I would also like to thank the participants of the project who disclosed health concerns relating to problems with Relative Humidity and allowed me to set up sensor networks in their residences. Special thanks goes to Daniel Scott Stuart for his collaboration on the C++ code used to partition the data files.

Lizabeth Lee

Contents

List of Figures

Chapter 1

Introduction

Indoor Air Quality is an increasing concern in the world today. In fact "the mere presence of people in a building or residence can significantly alter indoor air quality [1]." In a study evaluating student performance conducted in August 2003 by the United States Environmental Protection Agency (EPA) they concluded, "recent data suggests IAQ (Indoor Air Quality) may directly reduce a person's ability to perform specific mental tasks requiring concentration, calculation, or memory [2]." As the time spent indoors on average per person is on the rise [1], the need for a more accurate, properly maintained HVAC (Heating, Ventilation, and Air Conditioning) system is becoming increasingly necessary. This type of system currently exists in most businesses and schools, but is absent in most residential dwellings. This type of system generally does not include humidification control. The typical system in most residential dwellings utilizes one central thermostat regulating the heating/cooling needs of the entire home. Typically schools and businesses, depending on the size, maintain a small number of strategically placed thermostats through out the building. Unfortunately, in order to cut costs, the majority do not have a thermostat in every office or room where students or employees spend a major portion of their day. This can lead to physical discomfort of the occupants and actual health problems. Problems with health may result in increased absenteeism and/or a decrease in productivity. Low-cost, low-power, sensing devices, forming a wireless sensor network [3], is one alternative to the centrally located thermostat. These sensing devices equipped with the appropriate sensors, in this case temperature/humidity, transmit their raw data readings via radio waves to a central location to be processed. The system output can them be adjusted accordingly to maintain optimal comfort and ultimately better health.

1.1 Relative Humidity Defined

Humidity is the amount of water molecules in the air. In a normal environment air always holds humidity. However, the number of water molecules in the air can be as contrasting as black and white depending on the location. There is a maximum amount of water molecules that air can hold at any given temperature. The term used for this maximum amount is saturation water vapor pressure. When the number of water molecules in the air increases beyond that maximum level, condensation occurs in various forms, such as fog, mist, and/or precipitation. Relative humidity is a percentage of that maximum amount of humidity in the air at a given time and is temperature dependant. As the temperature increases or decreases so does the saturation water vapor pressure. This, in turn, causes the relative humidity to increase or decrease as a result of the direct correlation between the two [4]. Relative humidity plays an important role in how individuals perceive the comfort level and quality of the air in the indoor environment. In fact, "the human body is comfortable when relative humidity ranges between 20 and 60 percent," although as will be discussed, this range is not always conducive to optimal health [5]. The percentage of indoor relative humidity can also have a significant adverse effect on the structural soundness of buildings.

1.2 Effects of Indoor Relative Humidity

Relative humidity has been known to be the cause both directly and indirectly of various health and structural problems. In an article published in the Environmental Health Perspectives journal, Indirect Health Effects of Relative Humidity in Indoor Environments, it states "over the range of normal indoor temperatures (19 to 27 degrees Celsius) relative humidity has a direct effect on the body's physiological processes and an indirect effect on pathogenic organisms or chemicals [6]." The direct effects of relative humidity include illness and structural damage. The indirect effects involve an increase in the microscopic organisms suspended in the air we breathe. These organisms known as "Volatile Organic Compounds (VOC)" exist in the majority of substances used in every day life. A sample from the list of these substances includes building materials, furnishings, electronics, cleaning products, and perfumes [7]. VOC become suspended in air by the natural "out gassing" of materials. However, the rate of emissions and the concentration of microscopic air borne particles is increased by higher levels of relative humidity [8].

1.2.1 High Relative Humidity

Relative humidity that is too high may breed mold, rot, or pests, such as termites or cockroaches [9]. High relative humidity facilitates the growth of different varieties of mold. In fact, “all molds can potentially cause rashes, headaches, dizziness, nausea, allergic reactions including hay fever and asthma attacks [10].” The effects can be much worse in people with weakened immune systems, such as the every young and the elderly. The existence of mold is often detected by a musty [11] or mouldy [12] smell. High relative humidity (greater than 50 percent) can “produce enough condensation to stain ceilings and walls and cause flaking paint and peeling wallpaper [9].” The latter potentially increases the levels of VOC in the air. At high relative humidity levels microorganisms, such as fungi and bacteria, can survive on nonliving material including dust [13]. High relative humidities (above 70 percent) also “tend to favor the survival of viruses composed entirely of nucleic acids and proteins [6].” The most common groups of these viruses is the adeno viruses and the coxsackie viruses. The adeno viruses are a group of viruses that infect the membranes of the respiratory tract, the eyes, the intestines, and the urinary tract [14]. The coxsackie viruses are a group of viruses that cause a variety of infectious diseases, including a mild form of meningitis and hand-foot-and-mouth disease [15]. These two different types of viruses account for over 10 percent of the illnesses in children [14].

1.2.2 Low Relative Humidity

Low percentages of relative humidity also provides a good environment for the survival of different types of viruses and can make one more susceptible to them. For example, lipid containing viruses prefer low relative humidities [6]. These include rhinoviruses, influenza virus, and human rotavirus. The rhinoviruses consist of a group of viruses capable of causing common colds in adults and children [16]. The influenza virus and human rotavirus (a cause of gastroenteritis) exist on hard surfaces and thrive in low relative humidity [6]. In fact, for the viruses described above, the “viral inactivation rates increased sharply at relative humidities above 40 percent [6].” Accordingly, the “measles, influenza, herpesvirus varicellae, and rubella viruses survive longer during exposure to relative humidities below 50 percent [6].” Low relative humidity can compromise the “natural defense system of mucus in the nose and throat" through drying, which leads to a "more tolerant environment for germs" to enter the body [17]. To prevent the drying of the mucus membranes, "indoor relative humidity should be kept above 30 to 40 percent [6]." Low relative humidity has also been known to cause cracks in fingers [9]. One of the signs of indoor relative humidity being too low (less than 30 percent) is when movement about the home causes the production of static electricity [9]. In the case of structures, relative humidity that is too low may cause a house's drywall to crack and could also cause shrinking of wall paneling, wood flooring, and wood trim which opens joints [9].

1.3 Overview of Sensor Networks

Low-cost low-power sensors strategically placed through out the residence or office could detect problems with humidity and alert the occupant to potential problems. These sensors form a wireless sensor network. Wireless Sensor Networks are becoming increasingly popular in many fields including the field of micro-environmental monitoring of which this project is an application [18]. As a result of their easy deployment and small size, wireless sensors can be used in a variety of applications. These include habitat monitoring [19], both indoor and outdoor environmental observations [20], and structural detection above and under ground [21]. The TinyOS framework developed specifically for wireless sensor networks solves the problem of limited memory space inherent in the small sensors [22]. "Recent advances in miniaturization and low-cost, low-power design" has led to greater research in wireless sensor networks that can function virtually unattended for an extended period of time [23]. The low-cost features of sensors enables dense deployment of the modules. A greater number of sensors enhances the monitoring capabilities of the network. The dense deployment can also aid in sensor calibration. Many off the shelf sensors arrive precalibrated [4], but that does not always guarantee true readings when deployed. The blind calibration of the network can be performed by oversampling of signals of interest enabling the "recovery of unknown sensor gains [24]." However, this process is made much more difficult by the measurement noise inherent in the transmitted signal [25]. The low-power consumption of sensors has a direct correlation to the lifetime of the network. The greatest power drain is a result of data transmission to the base station. “In-network processing” may significantly reduce the power consumption of data transmission [26]. Selecting the optimal sensor for the sensing application and the optimal placement of that sensor may also aid in the reduction of energy consumption [27]. “Efficient filtering” of the measurement signal decreases the amount of noise in the signal and increases the accuracy of event detection in the network [28].

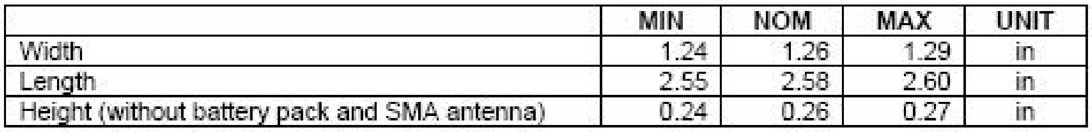
1.4 Project Definition and Scope

The purpose of this project is to demonstrate whether or not residential dwellings could benefit from an indoor humidification system to compliment the existing heating and/or cooling systems. If found necessary, the system could not only increase the individual perceived comfort level in the residence but may potentially reduce various health related problems especially in those with weakened immune systems. The project consisted of the deployment of three different sensor networks, one for each residence monitored. The lifetime of each network was dependant on the lifetime of the power source, 2 AA batteries. The data was collected during the months of December 2007 and January 2008. The winter months were chosen due to the generally higher relative humidity levels outside and the lower relative humidity levels inside the home. Each sensor network consisted of 12 tmote sky modules produced by the Moteiv Corporation [29]. Each module is equipped with an optional sensor manufactured by Sensirion which measures the temperature and humidity levels [30]. The features and operation of the tmote sky modules will be discussed in Chapter 2. Chapter 3 covers the operating system and programming language used to program each tmote sky module. Chapter 4 outlines the profiles for each monitored residence and the deployment details of the individual sensor networks. Chapter 5 contains the methods used for data manipulation and processing. The results of the processed data are also found in Chapter 5. The analysis of the results is contained in Chapter 6, as well as the limitations of the sensor networks and future work possibilities.

Chapter 2
Hardware Platform

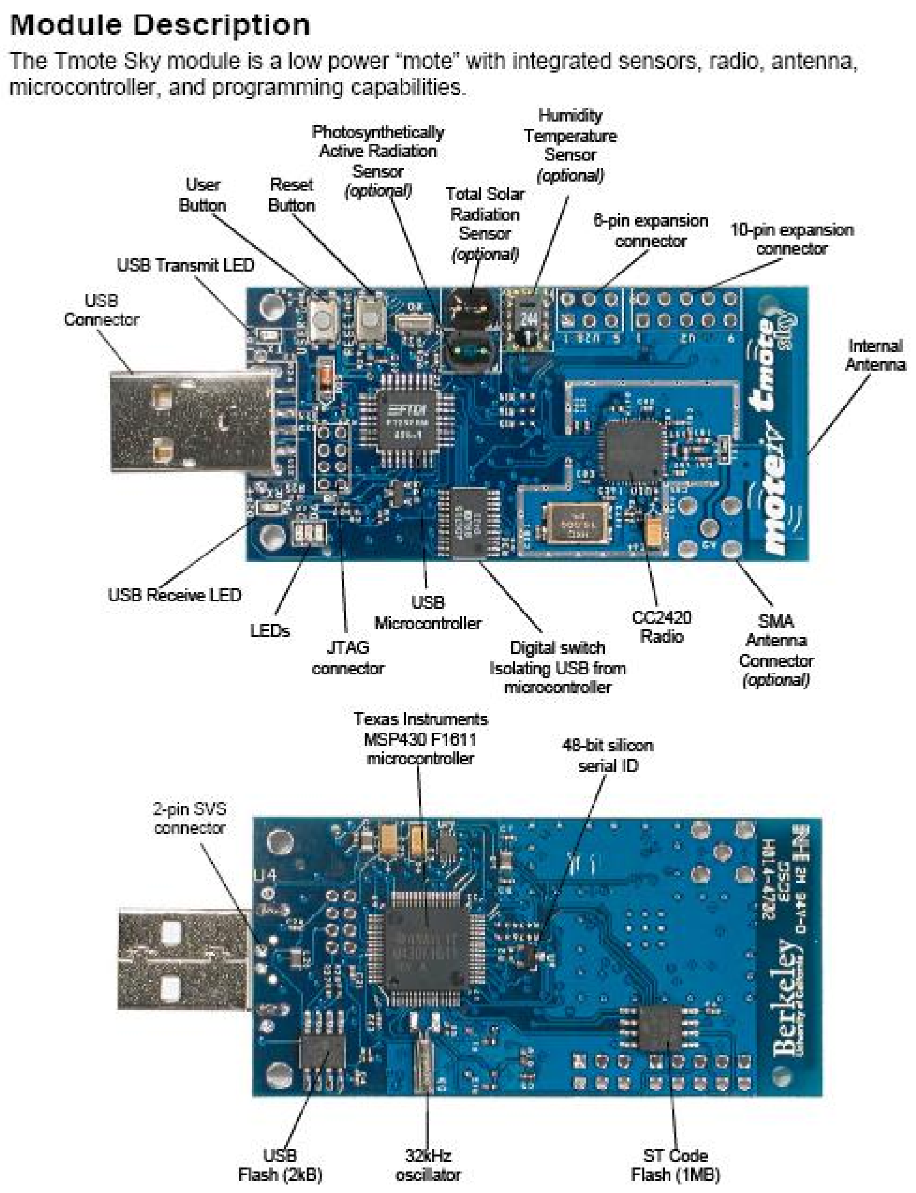
2.1 Tmote Sky

The tmote sky is the principle hardware component for this project [29]. It is a low-power wireless sensor module distributed by the Moteiv Corporation. The tmote sky module is compliant with IEEE 802.15.4 industry standards. It is also compliant with USB industry standards. This compliance facilitates the seamless interoperability with other devices. As a result of this interoperability, various optional sensors can be attached to the tmote sky module. These optional sensors are easily integrated into the operation of the tmote sky module. The names and locations of the various optional sensors attached to the tmote sky module used on this project are included in fig. 2.2. The optional sensor used to sense the variables of interest for this project is the attached temperature and humidity sensor manufactured by the Sensirion Corporation. The temperature and humidity sensor is discussed in the following section. Figure 2.2 illustrates the front and back views of the tmote sky module. The temperature and humidity sensor used in this project is located on the top of the front view. It is the optional sensor positioned third from the right. The dimensions of the tmote sky module are listed in fig. 2.1. As can be seen, the units for the measurements is inches. Note the small size of the tmote sky module.



	MIN	NOM	MAX	UNIT
Width	1.24	1.26	1.29	in
Length	2.55	2.58	2.60	in
Height (without battery pack and SMA antenna)	0.24	0.26	0.27	in

Fig. 2.1: Size in inches of tmote sky module.



Module Description

The Tmote Sky module is a low power "mote" with integrated sensors, radio, antenna, microcontroller, and programming capabilities.

User Button
Reset Button
Photosynthetically Active Radiation Sensor (optional)
Total Solar Radiation Sensor (optional)
Humidity Temperature Sensor (optional)
6-pin expansion connector
10-pin expansion connector
USB Transmit LED
USB Connector
Internal Antenna
USB Receive LED
LEDs
JTAG connector
USB Microcontroller
Digital switch Isolating USB from microcontroller
CC2420 Radio
SMA Antenna Connector (optional)
Texas Instruments MSP430 F1611 microcontroller
48-bit silicon serial ID
2-pin SVS connector
USB Flash (2kB)
32kHz oscillator
ST Code Flash (1MB)

Fig. 2.2: Front and back of the tmote sky module.

The tmote sky module operates on 2 AA batteries. The module operating range is 2.1 to 3.6 volts. Voltages exceeding 3.6 volts may cause damage to the microcontroller. When programming the tmote sky module, the voltage must be at least 2.7 volts. The programming is accomplished through the USB port of the host computer. When the module is connected to the host computer, no batteries are required and the tmote sky operates at approximately 3 volts. Once the programming is completed, the tmote sky module can function independent of the host computer. An exception to that is the designated “base” module. One of the tmote sky modules is designated as the base module during programming. This base module must remain connected to the host computer via the USB port to interface between the computer and the network. The base module receives the data from the other sensors in the network via radio waves and transmits it to the host computer through the USB port. The wireless transceiver of the tmote sky module transmits the data collected from the various optional sensors to the base module. The data is collected iteratively and stored in “packets” prior to transmission. Each iteration corresponds to one collected data sample from each of the participating sensors. The length and order of the packet is designated in the program. The data is transmitted at approximately 250kbps at a frequency of 2.4GHz, radio waves. The tmote sky module is equipped with a MSP430 microcontroller manufactured by Texas Instruments. Programming of the onboard microcontroller is performed via the USB port on the computer. The code is open source and is written in the nesC language [31]. The operating system was developed at UC Berkeley [22]. The nesC language and the TinyOS operating system are discussed in Chapter 3. Figure 2.3, located below, is the block diagram demonstrating the interconnectivity of the internal components inherent in the tmote sky module.

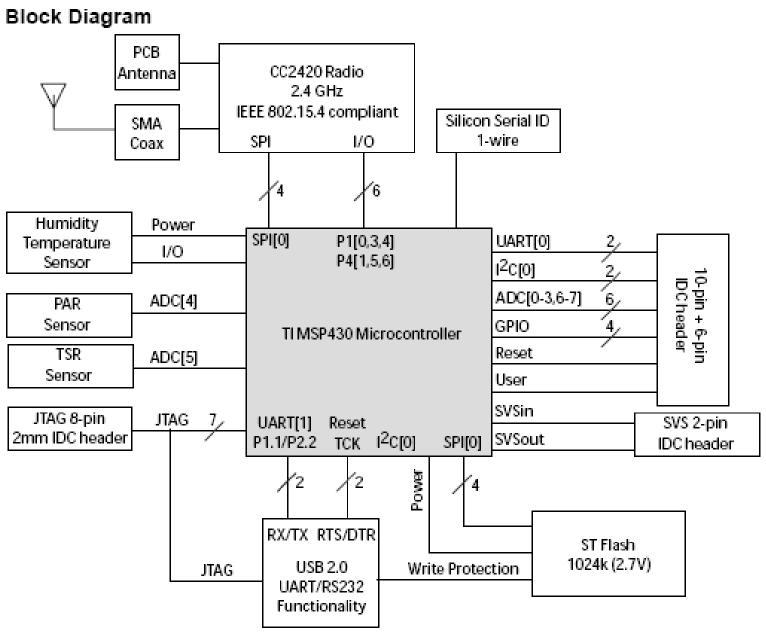


Fig. 2.3: Block diagram of the tmote sky module.

Next is the complete list of the key features of the tmote sky module [29].

- 250kbps 2.4GHz IEEE 802.15.4 Chipcon Wireless Transceiver
- Interoperability with other IEEE 802.15.4 devices
- 8MHz Texas Instruments MSP430 Microcontroller (10k RAM, 48k Flash)
- Integrated ADC, DAC, supply voltage supervisor, and DMA controller
- Integrated onboard antenna with 50m range indoors and 125m range outdoors
- Integrated humidity, temperature, and light sensors
- Ultra-low current consumption
- Fast wake-up from sleep (less than 6 microseconds)
- Hardware link-layer encryption and authentication
- Programming and data collection via USB
- 16-pin expansion support and optional SMA antenna connector
- TinyOS support: mesh networking and communication implementation
- Complies with FCC 15 and Industry Canada Regulations

2.2 Temperature/Humidity Sensor

The optional temperature and humidity sensor attached to the tmote sky is a multi-sensor module. This module is manufactured by Sensirion The Sensor Company. This multi-sensor is easily integrated into the external system of the tmote sky module by a 2-wire serial interface circuit. The onboard internal voltage regulator allows for a fast response time and an insensitivity to external disturbances [30]. The rapid response time integrates nicely into the data sample collection rate of the tmote sky microcontroller. This multi-sensor has low-power consumption and small size. Figure 2.4 demonstrates the size of this sensor. The sensor on the left is the surface mount version attached to the tmote sky module.

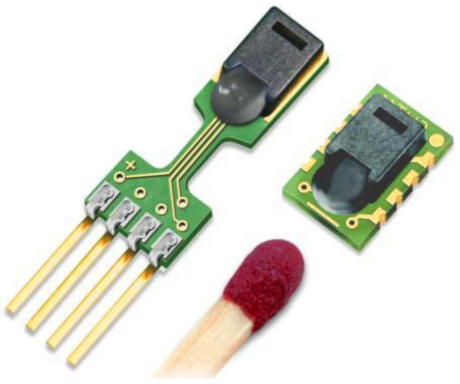


Fig. 2.4: Humidity and temperature sensor.

Included in this multi-sensor module is a band gap temperature sensor and a "capacitive polymer sensing element" to measure the relative humidity. Each multi-sensor is "individually calibrated in a precision humidity chamber [30]." The calibration coefficients are stored in the calibration memory which can be seen in fig. 2.5. These calibration coefficients are used internally during measurements to calibrate the measured data from the individual sensing components. Each sensing component outputs a digital signal. The multi-sensor is also equipped with a 14-bit analog-to-digital converter. This converter takes the digital output from each sensing component and converts it to an analog signal. This analog signal is then sent to the tmote sky module. Figure 2.5 illustrates the block diagram of the temperature and humidity sensor.

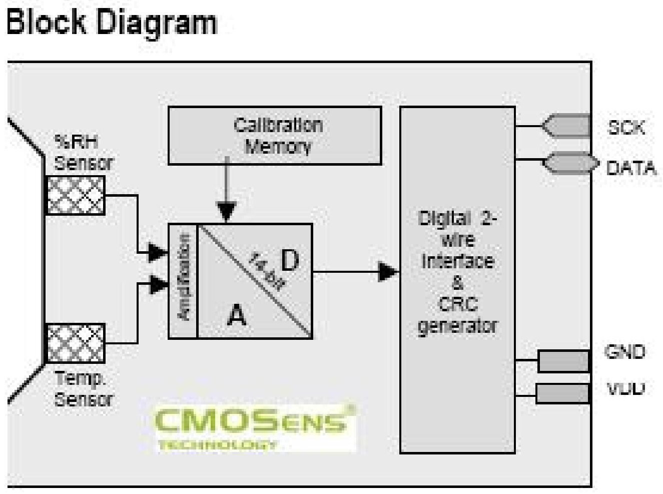


Fig. 2.5: Humidity and temperature sensor block diagram.

According to the calibration certificate, the calibration of the temperature and humidity sensor is valid from May 2007 to May 2009 [32]. The key features of this temperature and humidity sensor as found in the data sheet are listed below [30].

- Bandgap temperature sensor
- Capacitive polymer sensing for element for humidity
- 14-bit analog-to-digital converter
- Serial interface circuit
- Fully calibrated, digital output
- Excellent long-term stability
- No external components required
- Ultra low-power consumption
- Surface mountable or 4-pin fully interchangeable
- Small size
- Automatic power down

Chapter 3

Software Platform

3.1 TinyOS Operating System

The operating system framework used by the tmote sky module is TinyOS [22]. It is an embedded operating system and platform developed by UC Berkeley specifically for wireless sensor networks. Programming TinyOS is challenging because it is written in the nesC language, discussed below, as a group of cooperating tasks and processes [33]. TinyOS has a component based architecture [34]. The components included in this architecture are reusable. These components are connected by wiring specifications that are independent of component implementation. The "wiring" of components together is done by interfaces. Both components and interfaces are separate functions. Figure 3.1 is an example of component-based architecture. The blocks represent components while the wires represent the interfaces required to connect components together.

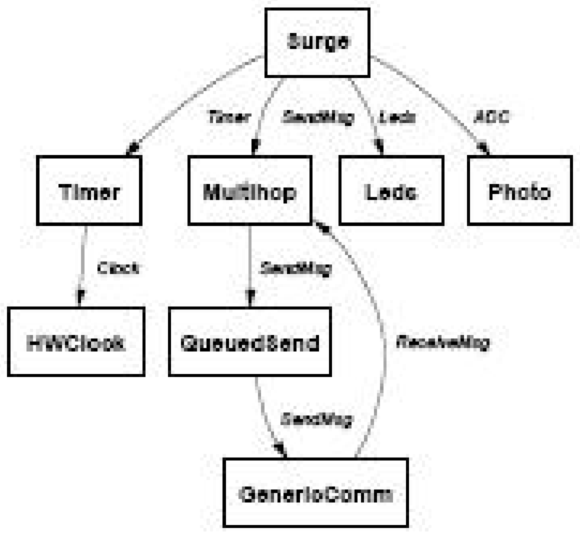


Fig. 3.1: Surge application.

TinyOS is designed for event-driven applications. The operating system core requires only 400 bytes of code [31]. Included in the 400 bytes is data memory. There are two concurrent processes in TinyOS, tasks and events. “Tasks run to completion and do not preempt each other [31].” Components can “post” tasks. The post operation immediately returns. Components can also use tasks when not restricted by timing requirements. Events also run to completion but can preempt the execution of another task or another event. Events signify the completion of an operation such as message reception or time passing. In TinyOS code runs either asynchronously in response to an interrupt or in a synchronously scheduled task. TinyOS is ultimately driven by events which represent hardware interrupts. The nesC language, explained below, “directly supports TinyOS’s event-based concurrency model [31].”

3.2 nesC Programming Language

The nesC language is an open source programming language for networked embedded systems [31]. One such system is a sensor network. The nesC programming language is an extension of the C programming language but has been optimized for the memory limitations of sensor networks [22]. It is a static language with no dynamic memory allocation and the “call-graph is fully known at compile time [31].” The nesC language provides three main contributions not found in C [31]. The first contribution is that nesC defines a model based on components. In nesC there are two types of components, modules and configurations. Modules provide application code which implement one or more interfaces. Configurations wire other components together. The nesC applications are built by writing and assembling components. The components used in the code for this project are reproduced below.

```
components Main
           , OscilloscopeTmoteSkyM as OscilloscopeM
           , TimerC
           , LedsC
           , HumidityC
           , OscopeC
           , GenericComm as Comm;
```

Each component provides and uses an interface. These interfaces are bidirectional to simplify event flow. The only way to access a component is through the interface. An interface generally models some service such as sending a message and is specified by an interface type. The interfaces used in the code for this project are reproduced below.

```
provides interface StdControl;
  uses {
    interface Timer;
    interface Leds;
    interface SplitControl as HumidityControl;
    interface ADC as Humidity;
    interface ADC as Temperature;
    interface Oscope as OHumidity;
    interface Oscope as OTemperature;
    interface ADCError as HumidityError;
    interface ADCError as TemperatureError;
  }
```

Interfaces contain commands and events which are basically functions. The interface implements the commands while the user implements the events. An example of a command function followed by an event function from the code used for this project is reproduced below.

```
command result_t StdControl.init() {
    call Leds.init();
    call Leds.set(0);
    state = HUMIDITY;
    call HumidityControl.init();
    return SUCCESS;
  }
```

```
event result_t HumidityControl.initDone() {
    return SUCCESS;
  }
```

The complete code used for this project is located in Appendix A. The second contribution of the nesC programming language is that the nesC compiler can detect most data races at compile time. This contribution allows applications to use concurrent behavior with limited resources. Data races "occur due to concurrent updates to shared state [31]." In order to prevent data races there are two things the compiler must do. The first is to understand the concurrency model and the second is to determine the target of every update. Synchronous code is "atomic" with respect to other synchronous code [31]. Atomic signifies that any shared state between the two sets of code will be updated atomically. This non-preemption allows for the avoidance of races between tasks. Concurrency between asynchronous code which involves events is controlled directly by interrupts. The third contribution is a reduced code size. This is a necessity for wireless sensor networks where each module has limited memory and requires a rapid respones. The code size is reduced by the nesC compiler. The compiler uses the "application call-graph to eliminate unreachable code and module boundary crossings [31]." This serves to inline small functions and allows for cross component optimization. These optimizations include constant propagation and common "subexpression elimination [31]." As a result of these optimizations the memory footprint is greatly reduced.

3.3 Cygwin

Cygwin is a collection of tools originally developed by Cygnus Solutions [35]. Cygwin provides header files and libraries that make it possible to compile Unix applications for use on computers running Microsoft Windows operating systems. Cygwin makes it possible to easily port many significant Unix programs without the need for extensive changes to the source code. The applications provided by the MOTEIV corporation including the Oscilloscope application used for this project are compiled using Cygwin. Cygwin is also used to program the tmote sky modules used for this project. Cygwin is utilized via the command prompt.

Chapter 4
Sensor Networks Deployment

4.1 Overview

The sensor networks were deployed in three different residences. In each case, 12 tmote sky modules were placed by the home owner in “places of interest” as outlined below. I chose these residential dwellings for various reasons. The primary reason was based on the complaints of the occupants and the construction methods of each residential dwelling. Two of the homes were manufactured homes which are notorious for problems with moisture accumulation. The third home was a conventionally built home sided with stucco. Homes sided with stucco have been reported to develop problems with mold beneath the stucco by several contractors in the location area. The secondary reason was based on performance testing of the sensor network properties as a whole and the individual sensor performance. One of the properties of interest was the transmitting range of the motes as related to the base station. Another property of interest involved the conditions affecting the network lifetime and the final property of interest was concerned with the overall robustness of a tmote sky based sensor network. These properties will be discussed in the Chapter 6. Each of the chosen dwellings possessed a conventional natural gas furnace but no home humidification system. I wish to concede that relative humidity in and of itself is not the sole contributor to the health and/or structural problems of these occupants and their dwellings. However as documented in the introduction, relative humidity levels could potentially be one of the contributing factors.

4.2 Network Profiles

4.2.1 Network 1

The first residence involved in the project was a manufactured home in a mobile home park. It was occupied by a single resident. The residence is a single level home with 1136 square feet of living space. This home is approximately twelve years old. One of the complaints of the occupant was that movement around the home frequently generated static electricity. One of the signs of indoor relative humidity being too low (less than 30 percent) is when movement about the home causes the production of static electricity [9]. Another problem reported by the resident was occasional cracks in the skin on their fingers. Low relative humidity has also been known to cause cracks in fingers [9]. There is visible mold behind the wall paper in the kitchen caused by a water leak that has since been repaired. As stated in the introduction "all molds can potentially cause rashes, headaches, dizziness, nausea, allergic reactions including hay fever, and asthma attacks" [10]. The resident experiences frequent headaches. There are visible cracks down the texture on the walls and some of the joints where the molding meets have separated. In the case of structures, relative humidity that is too low may cause a house's drywall to crack and could also cause shrinking of wall paneling, wood flooring, and wood trim which opens joints [9]. I chose this dwelling as one of the participants for the sensor network relative humidity monitoring project because the structural damage as stated above and the problems with static electricity are classic symptoms of a problem with relative low humidity levels.

4.2.2 Network 2

The second residence involved in the project was a conventional stick built home that is sided with stucco. This home was occupied by a family of four. As mentioned in the overview, homes sided with stucco have been reported by several contractors to develop problems with mold beneath the stucco in the location area. This is a two-story home with a finished basement. The home contains 3075 square feet of living space and is approximately 4 years old. Two of the occupants have been diagnosed with asthma and suffer from frequent head and body aches. The children in the home experience occasional rashes and skin irritations. High relative humidity facilitates the growth of different varieties of mold. In fact, "all molds can potentially cause rashes, headaches, dizziness, nausea, allergic reactions including hay fever, and asthma attacks [10]." One of the adult occupants is also bothered by "allergies" with symptoms of a "runny nose and scratchy throat" even during the winter months and wakes up coughing every night. Studies indicate a "positive association between allergic respiratory, nose and skin symptoms” with high relative humidity levels [36]. I chose this family as one of my participants in the sensor network relative humidity monitoring project because of their frequent health discomforts which have been linked to high levels of relative humidity.

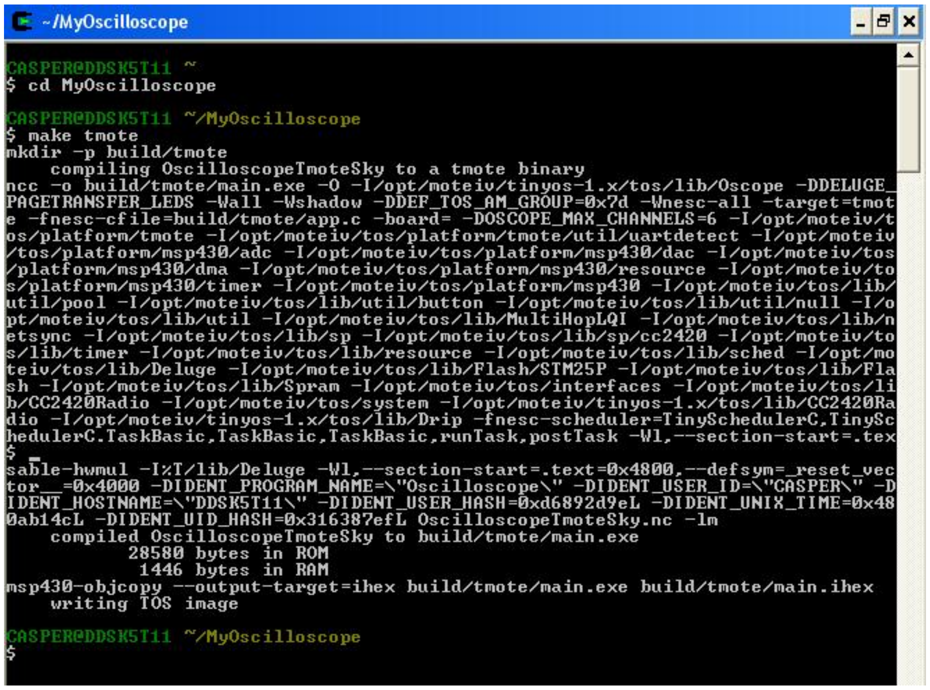
4.2.3 Network 3

The third and final residence involved in the project was another manufactured home in a mobile home park. This residence was also occupied by a single resident. This residence is a single level home with 1024 square feet of living space. This home is located adjacent to the home of the first participant. This home is equipped with an attic fan that runs continuously six hours on and six hours off. The purpose of this fan is to control the accumulation of moisture inside the home. The fan may have an impact on the indoor relative humidity levels of the residence. This occupant rarely experiences the heath symptoms related to the relative humidity levels high or low of the home. Static electricity is also reported to not be a problem. I chose this residence to be a type of control residence when compared to the first participant due to the location and similarities in the construction method.

4.3 Deployment

Each network consisted of 12 tmote sky modules. The individual modules were placed in various locations throughout each residence. The placement locations were chosen by the individual occupants based on their perceptions of possible problem areas. The bathrooms were excluded as high relative humidity levels in bathrooms are assumed to be the norm. Each module was programmed individually and assigned a number. The base module was designated and assigned number 3. The program code was a modified version of the Oscilloscope application code distributed by the Moteiv Corporation. The complete code is reproduced in Appendix A. The code was compiled using Cygwin from the command window. Figure 4.1 illustrates the Cygwin interface after the successful compilation of the Oscilloscope code.

I altered the code to transmit and display only the data of interest, namely the readings from the temperature and humidity sensor of each tmote sky module [30]. The code is open source and is written in the nesC language [31]. The raw data collected from each sensor was transmitted in



```
CASPER@DDSK5T11 ~
$ cd MyOscilloscope

CASPER@DDSK5T11 ~/MyOscilloscope
$ make tmote
mkdir -p build/tmote
    compiling OscilloscopeTmoteSky to a tmote binary
ncc -o build/tmote/main.exe -O -I/opt/moteiv/tinyos-1.x/tos/lib/Oscope -DDELUGE_
PAGETRANSFER_LEDS -Wall -Wshadow -DDEF_TOS_AM_GROUP=0x7d -Wnesc-all -target=tmot
e -fnesc-cfile=build/tmote/app.c -board= -DOSCOPE_MAX_CHANNELS=6 -I/opt/moteiv/t
os/platform/tmote -I/opt/moteiv/tos/platform/tmote/util/uartdetect -I/opt/moteiv
/tos/platform/msp430/adc -I/opt/moteiv/tos/platform/msp430/dac -I/opt/moteiv/tos
/platform/msp430/dma -I/opt/moteiv/tos/platform/msp430/resource -I/opt/moteiv/to
s/platform/msp430/timer -I/opt/moteiv/tos/platform/msp430 -I/opt/moteiv/tos/lib/
util/pool -I/opt/moteiv/tos/lib/util/button -I/opt/moteiv/tos/lib/util/null -I/o
pt/moteiv/tos/lib/util -I/opt/moteiv/tos/lib/MultiHopLQI -I/opt/moteiv/tos/lib/n
etsync -I/opt/moteiv/tos/lib/sp -I/opt/moteiv/tos/lib/sp/cc2420 -I/opt/moteiv/to
s/lib/timer -I/opt/moteiv/tos/lib/resource -I/opt/moteiv/tos/lib/sched -I/opt/mo
teiv/tos/lib/Deluge -I/opt/moteiv/tos/lib/Flash/STM25P -I/opt/moteiv/tos/lib/Fla
sh -I/opt/moteiv/tos/lib/Spram -I/opt/moteiv/tos/interfaces -I/opt/moteiv/tos/li
b/CC2420Radio -I/opt/moteiv/tos/system -I/opt/moteiv/tinyos-1.x/tos/lib/CC2420Ra
dio -I/opt/moteiv/tinyos-1.x/tos/lib/Drip -fnesc-scheduler=TinySchedulerC,TinySc
hedulerC.TaskBasic,TaskBasic,TaskBasic,runTask,postTask -Wl,--section-start=.tex
$ _
sable-hwmul -I%T/lib/Deluge -Wl,--section-start=.text=0x4800,--defsym=_reset_vec
tor__=0x4000 -DIDENT_PROGRAM_NAME=\"Oscilloscope\" -DIDENT_USER_ID=\"CASPER\" -D
IDENT_HOSTNAME=\"DDSK5T11\" -DIDENT_USER_HASH=0xd6892d9eL -DIDENT_UNIX_TIME=0x48
0ab14cL -DIDENT_UID_HASH=0x316387efL OscilloscopeTmoteSky.nc -lm
    compiled OscilloscopeTmoteSky to build/tmote/main.exe
            28580 bytes in ROM
             1446 bytes in RAM
msp430-objcopy --output-target=ihex build/tmote/main.exe build/tmote/main.ihex
    writing TOS image

CASPER@DDSK5T11 ~/MyOscilloscope
$
```

Fig. 4.1: Cygwin interface display.

real time via radio waves to the designated base module connected to the host computer via a USB port. The computer monitor was used to display the raw returning signals from each module using the modified version of the Oscilloscope application. This application saves all the transmitted data samples in two vertical columns. The first column is the sample time. The second column is the corresponding raw data reading taken at that time. The vertical columns are partitioned horizontally by the module number assigned during programming and the channel number over which the readings of interest were transmitted. The raw humidity readings were transmitted on channel zero and the raw temperature readings were transmitted on channel one. In order to facilitate the data processing, this vertical data set had to be separated into 24 individual files, one for each of the two channels corresponding to the 12 tmote sky modules. I chose the Oscilloscope application because of the inherent visualization tools for the real-time data collection. An example of the application collecting data in real time is illustrated in fig. 4.2.

Prior to the deployment of each sensor network, two new AA batteries were installed into the attached power source of each module. The data was collected until the majority of the modules ceased to transmit. The duration of the transmitting time was an indication of the lifetime of each

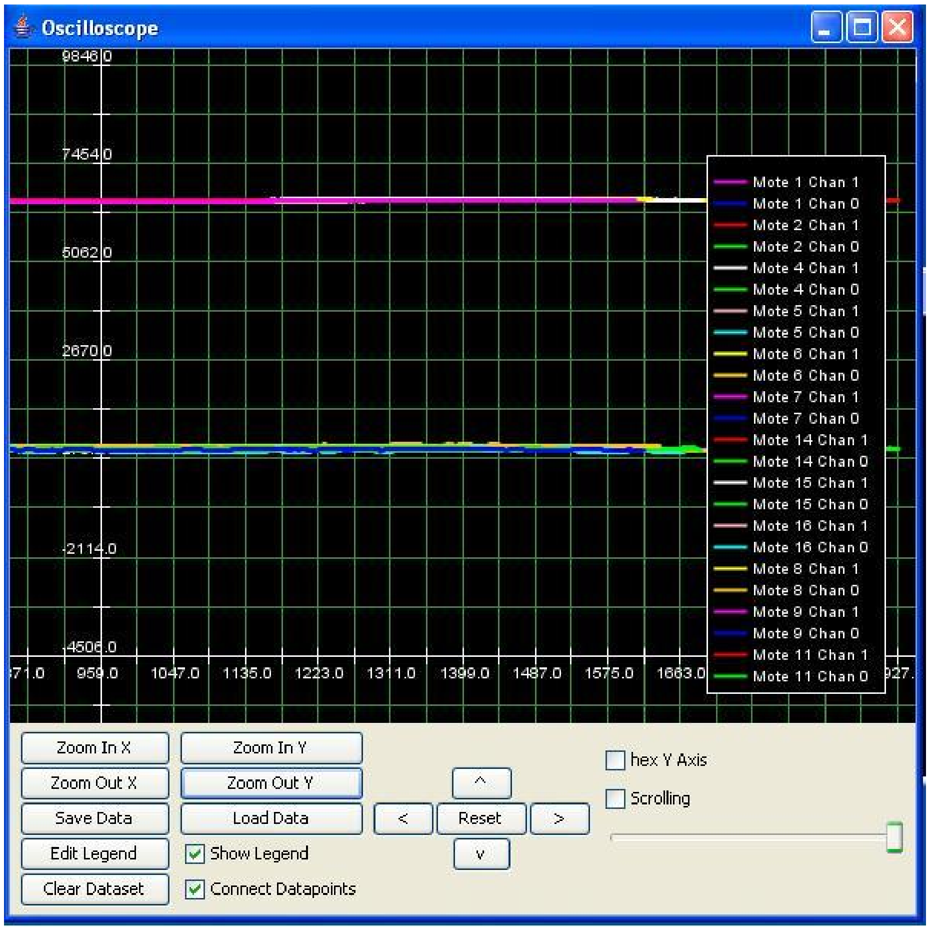


Fig. 4.2: Oscilloscope application display.

network when powered by two AA batteries. The data from the tmote sky modules was transmitted continuously over the lifetime of the network. Although continuous data transmission by the individual modules shorted the network lifetime, the continuous monitoring of the temperature and humidity levels of the residences was necessary for the scope of the project. The continuous monitoring of the transmitted data also made it possible to view the behavior of the individual tmote sky modules as the power level decreased.

4.3.1 Sensor Placement Locations Network 1

The first sensor network began transmitting on 14 December 2007. The lifetime of the network was approximately four days of continuous transmission at 250kbps. The base station for this network was located approximately 60 feet from the nearest mote with 40 feet of that distance being outdoors. The individual tmote sky modules were placed according to the following locations.

- Node 1 placed in northwest bedroom on top of chest of drawers
- Node 2 placed in kitchen located on east of home on north counter top
- Node 4 placed in southwest bedroom on wooden desk top
- Node 5 placed in dining area on top of chest on east wall
- Node 6 placed in living room located on west of home on top of coffee table
- Node 7 placed in kitchen located on east of home on west counter top
- Node 8 placed in northwest bedroom on top of bookcase
- Node 9 placed in kitchen on east wall on top of breadbox
- Node 11 placed in dining area inside of glass china hutch
- Node 14 placed in living room located on west of home on top of TV cabinet
- Node 15 placed in southeast bedroom on top of chest of drawers
- Node 16 placed in southwest bedroom on top of bookcase

4.3.2 Sensor Placement Locations Network 2

The second sensor network began transmitting on 24 December 2007. The lifetime of the network as a whole was approximately four days of continuous transmission at 250kbps. The base station for this network was located approximately 6 inches from the closest tmote sky module. The location of highest interest as far as the tmote sky module transmitting capabilities was node 11. This node was placed in a noninsulated cement storage room located in the basement with the foundation serving as three of the four walls. The individual tmote sky modules were placed according to the following locations.

- Node 1 placed in south facing kitchen above pantry shelf
- Node 2 placed in south facing entry way on phone desk
- Node 4 placed in living room on north wall fireplace mantel
- Node 5 placed in laundry room on hat shelf
- Node 6 placed in purple room on top of TV cabinet
- Node 7 placed in southeast bedroom on mirror dresser
- Node 8 placed in basement living room by TV
- Node 9 placed in basement computer room on desk
- Node 11 placed in basement cement cold storage room
- Node 14 placed in basement northwest bedroom on wardrobe
- Node 15 placed in basement northeast bedroom on desk
- Node 16 placed in basement storage room under stairs

4.3.3 Sensor Placement Locations Network 3

The third and final network began transmitting on 8 January 2008. The lifetime of the network was approximately four days of continuous transmission at 250kbps. The host computer connected to the base module was located in kitchen near inside wall. The base station was located approximately 8 feet from the nearest mote. The individual tmote sky modules were placed according to the following locations.

- Node 1 placed in northwest small bedroom on bottom shelf of bookcase by door
- Node 2 placed in northwest small bedroom on closet shelf by west wall
- Node 4 placed in north bathroom on top of cabinet near ceiling
- Node 5 placed in north bathroom in closet bottom next to floor
- Node 6 placed in living room in crate next to hallway
- Node 7 placed in living room on top of coat shelf by front door
- Node 8 placed in kitchen on west facing window sill
- Node 9 placed in northeast bedroom under table near inside wall
- Node 11 placed in northeast bedroom on closet shelf near north wall
- Node 14 placed in southeast bedroom on floor under plant near east wall
- Node 15 placed in southeast bedroom on top of headboard near south wall
- Node 16 placed in closet by skylight on top shelf

Chapter 5

Data Manipulation

5.1 Data Manipulation

The data was saved at varying time intervals, the longest being twelve hour blocks. This was done to discover the optimal length of transmitting time necessary to recover the majority of the sampled data using the Oscilloscope application code. The Oscilloscope application used to display the data being collected in realtime as seen in fig.4.2, was also used to save the data. The Oscilloscope application saves the transmitted data samples measurements into one long vertical file. This file is sectioned horizontally by module name and transmitting channel number with humidity being channel 0 and temperature being channel 1. The resulting file has a total of 24 subsections per data set per file. The data files ranged from approximately 3000kb to over 5000kb per sampled data set per channel depending on the transmitting interval between saving times. The data manipulation began with the separation of the data sets into 24 separate data set files, two for each tmote sky module. The code used to separate the data files was written by Daniel Scott Stuart and is located in Appendix B. The 24 separate raw data files per data set could then be imported into MATLAB for processing. The 24 separate files contained the temperature and humidity measurements from each of the individual tmote sky modules for that particular data set. The processing of the individual sensor readings was done using an m-file containing the equations supplied by the Oscilloscope application. These equations transformed the raw data samples into degrees Celsius and Fahrenheit which were used to calculate the corresponding percentages of relative humidity for each tmote sky module. The equations also take into account the non-linearities inherent in the sensors themselves [37]. The percentages of relative humidity from each tmote sky module were then used to create graphs. These graphs made possible the individual sampled data measurement comparison and averaging. The comparison graphs also aid in the evaluation of the accuracy of the data being transmitted from each tmote sky module and for calibration verification. The range of the data set graphs is used to indicate the relative humidity level of the residence for that particular time interval. The code used to import the raw data and convert it into the percentage of relative humidity was written by myself and is located in Appendix C.

5.2 Graphs of Collected Data

The data sets collected from the various participating residences are located below.

5.2.1 Residence 1

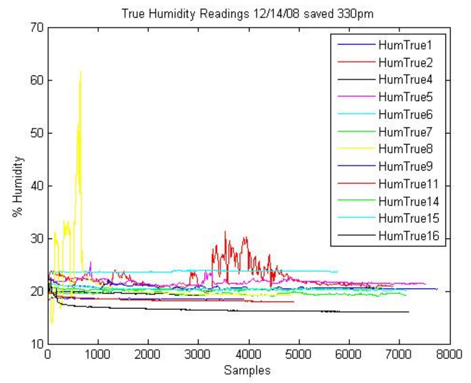
Figures 5.1, 5.2, and 5.3 illustrate the range of the percentage of relative humidity for the first residence. Each figure is labeled with the date and the time the data was saved. These measurements were taken over the lifetime of this sensor network. The variable of interest for this project is not the individual readings collected from each tmote sky module, but the range the collected measurements span.

5.2.2 Residence 2

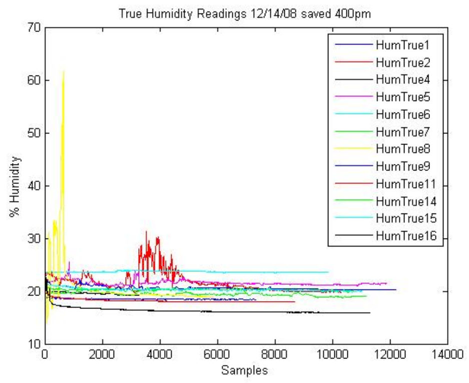
The graphs for the second residence are figs 5.4 through 5.7. They illustrate the data collected from the second residence over the lifetime of the sensor network. Each figure is labeled with the date and the time the data was saved. As can be seen, the percent relative humidity in this home is higher than in the other two. As a reminder, this home is sided with stucco whereas the other two are wood. The variable of interest for this project is not the individual readings collected from each tmote sky module, but the range the collected measurements span.

5.2.3 Residence 3

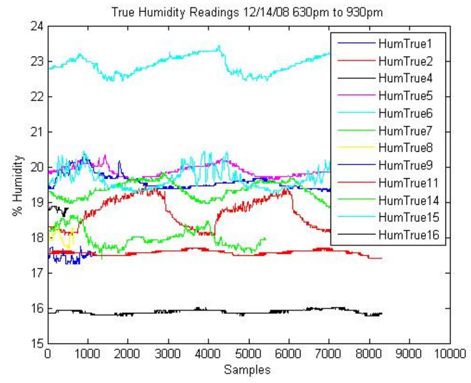
The graphs for the third residence are figs 5.8 through 5.12. The sets of graphs illustrating the range of the percentage of relative humidity for the third residence. Each figure is labeled with the date and the time the data was saved. These measurements were taken over the lifetime of this sensor network. The variable of interest for this project is not the individual readings collected from each tmote sky module, but the range the collected measurements span. The transients seen at the left of the first figure are a result of placement.



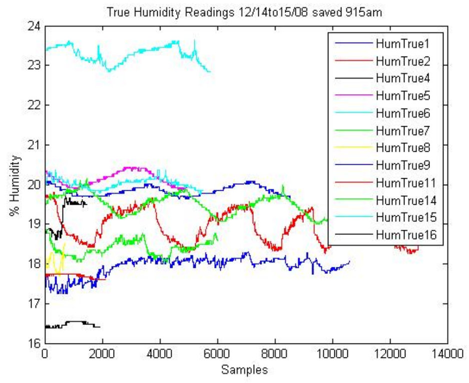
(a) Placement 12/14/07 at 3:30pm.



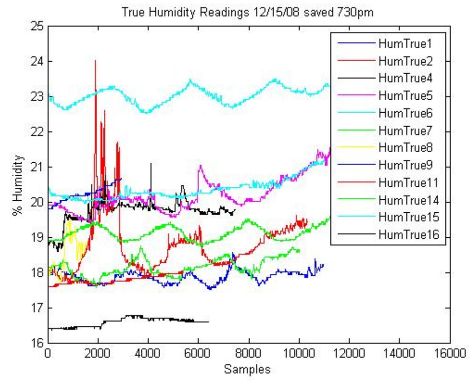
(b) Data Saved 12/14/07 at 4:00pm.



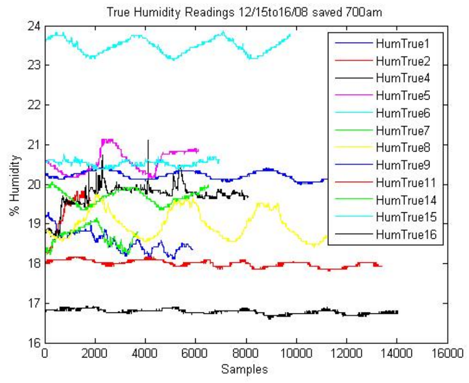
(c) Data Saved 12/14/07 at 9:30pm.



(d) Data Saved 12/14to15/07 at 9:15am.

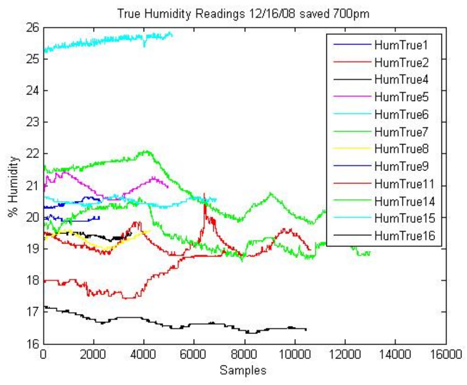


(e) Data Saved 12/15/07 at 7:30pm.

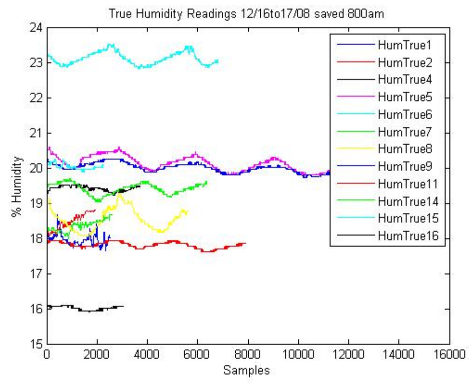


(f) Data Saved 12/15to16/07 at 7:00am.

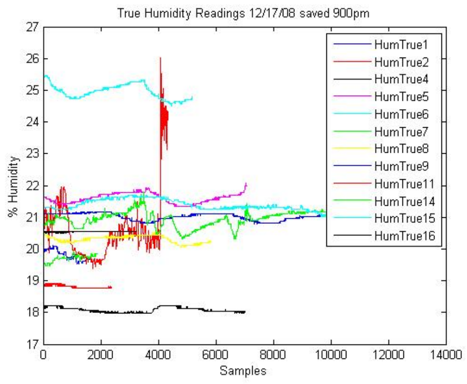
Fig. 5.1: Residence 1, 12/14/07 at 3:30pm to 12/16/07 at 7:00am.



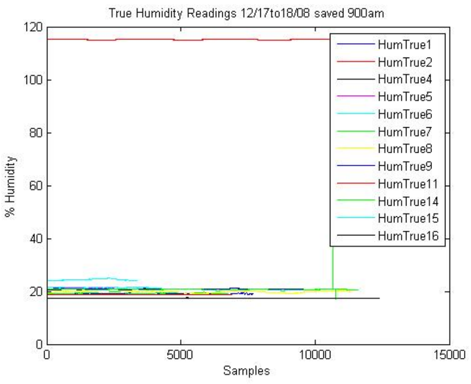
(a) Data Saved 12/16/07 at 7:00pm.



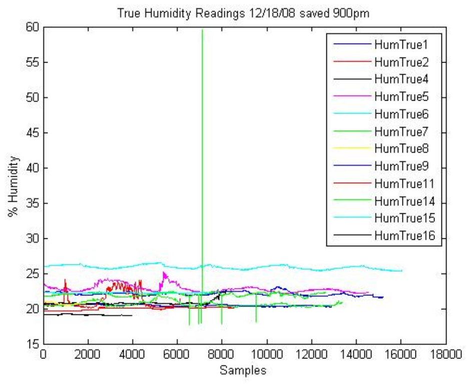
(b) Data Saved 12/16to17/07 at 8:00am.



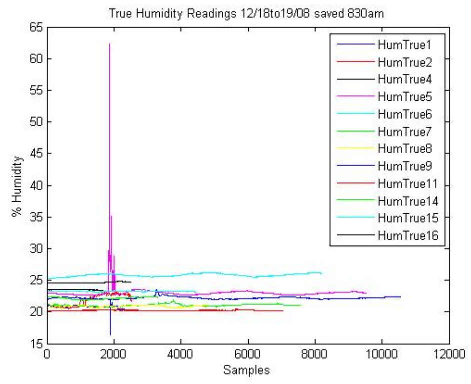
(c) Data Saved 12/17/07 at 9:00pm.



(d) Data Saved 12/17to18/07 at 9:00am.

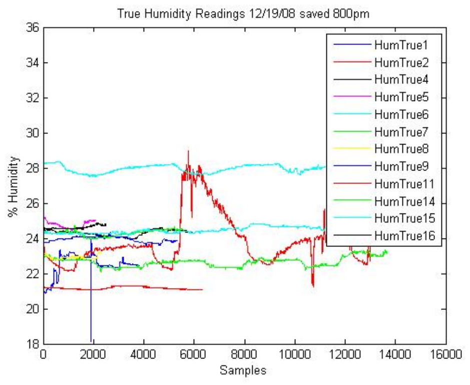


(e) Data Saved 12/18/07 at 9:00pm.

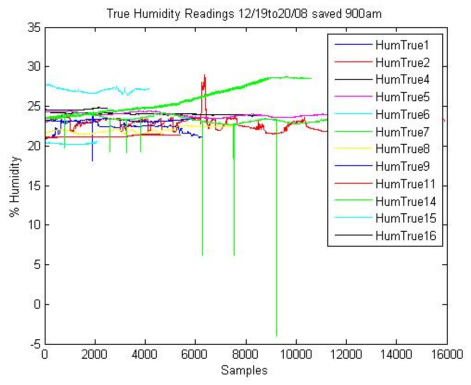


(f) Data Saved 12/18to19/07 at 8:30am.

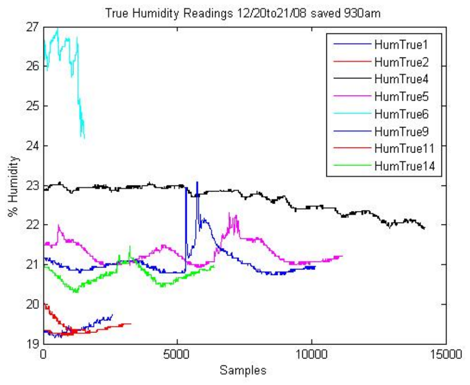
Fig. 5.2: Residence 1, 12/16/07 at 7:00pm to 12/19/07 at 8:30am.



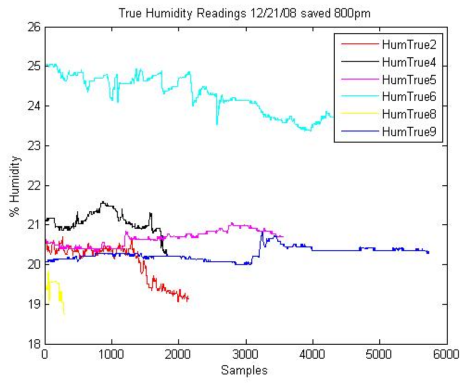
(a) Data Saved 12/19/07 at 8:00pm.



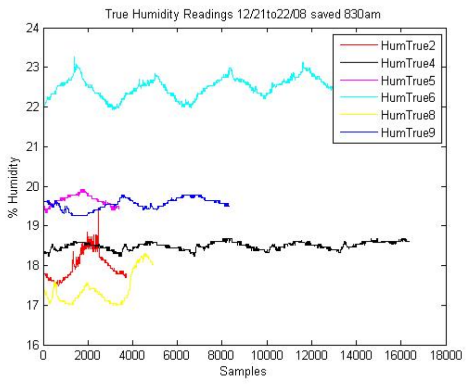
(b) Data Saved 12/19to20/07 at 9:00am.



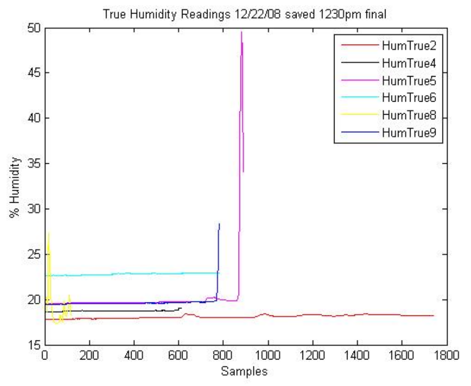
(c) Data Saved 12/20to21/07 at 9:30am.



(d) Data Saved 12/21/07 at 8:00pm.

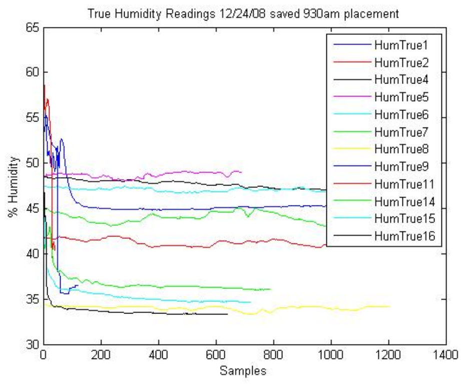


(e) Data Saved 12/21to22/07 at 8:30am.

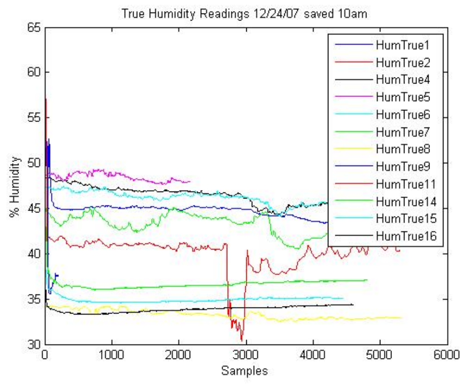


(f) Data Saved 12/22/07 at 12:30pm Final.

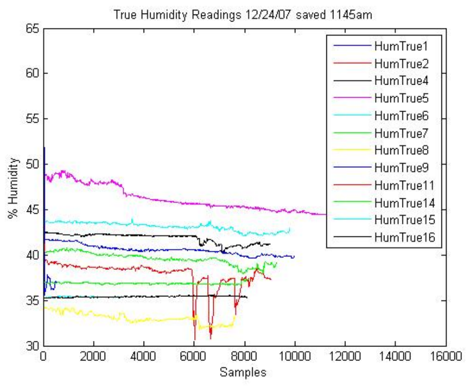
Fig. 5.3: Residence 1, 12/19/07 at 8:00pm to 12/22/07 at 12:30pm.



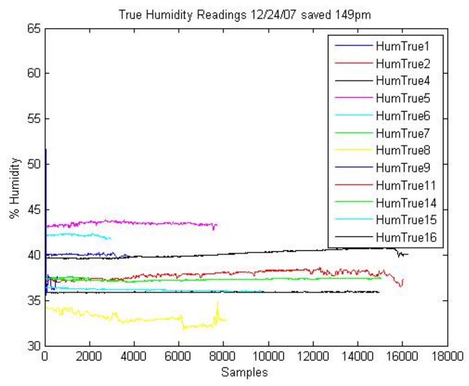
(a) Placement 12/24/07 at 9:30am.



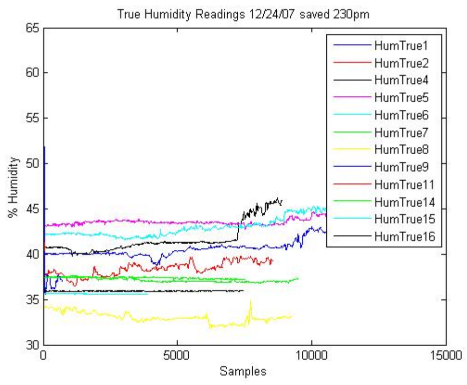
(b) Data Saved 12/24/07 at 10:00am.



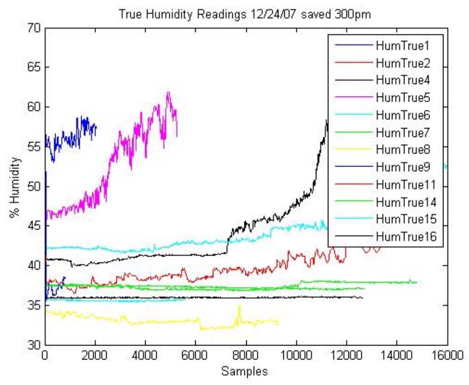
(c) Data Saved 12/24/07 at 11:45am.



(d) Data Saved 12/24/07 at 1:49pm.

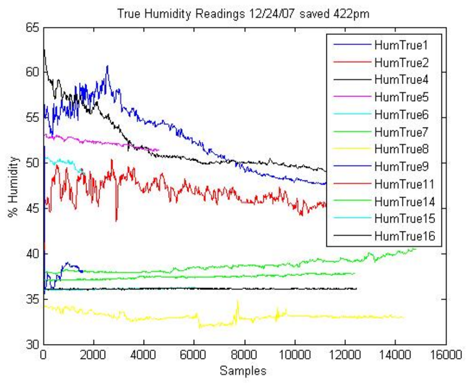


(e) Data Saved 12/24/07 at 2:30pm.

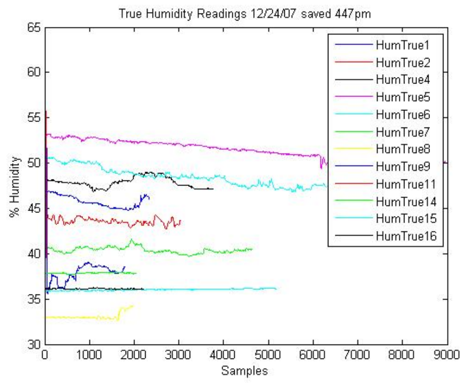


(f) Data Saved 12/24/07 at 3:00pm.

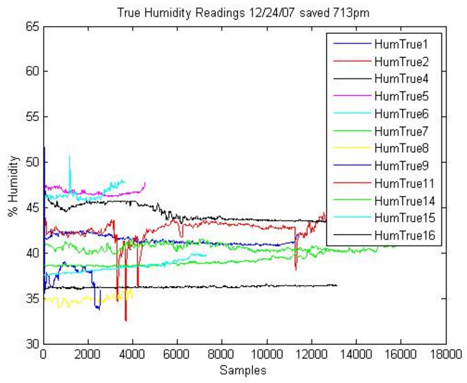
Fig. 5.4: Residence 2, 12/24/07 at 9:30am to 12/24/07 at 3:00pm.



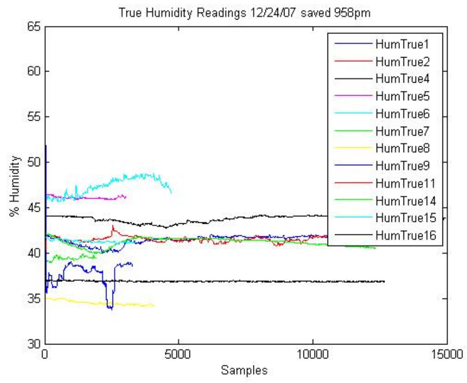
(a) Data Saved 12/24/07 at 4:22pm.



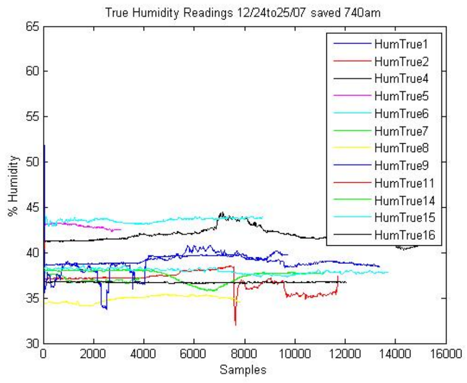
(b) Data Saved 12/24/07 at 4:47pm.



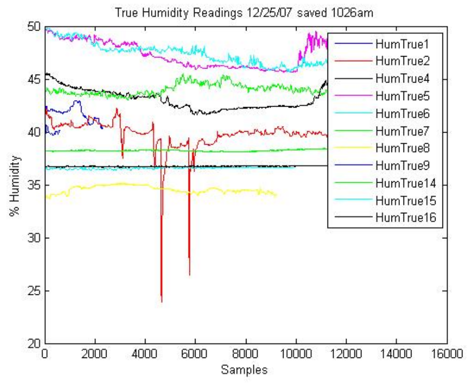
(c) Data Saved 12/24/07 at 7:13pm.



(d) Data Saved 12/24/07 at 9:58pm.

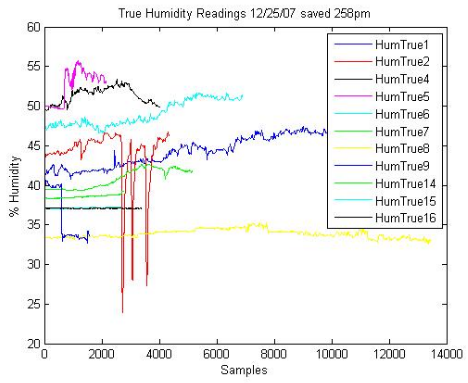


(e) Data Saved 12/24to25/07 at 7:40am.

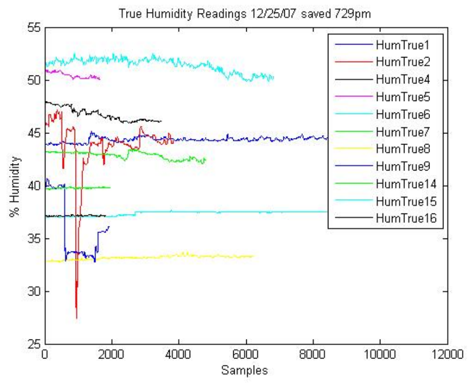


(f) Data Saved 12/25/07 at 10:26am.

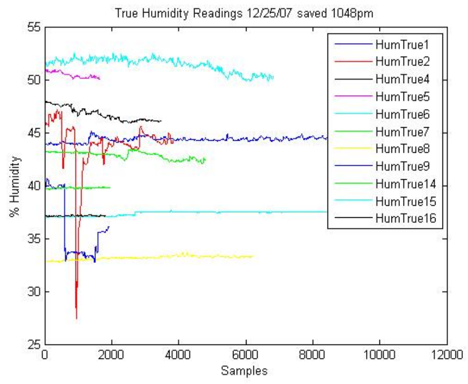
Fig. 5.5: Residence 2, 12/24/07 at 4:22pm to 12/25/07 at 10:26am.



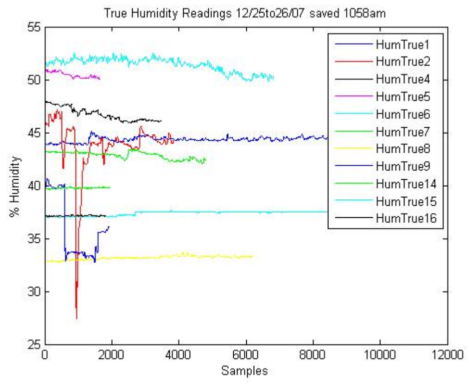
(a) Data Saved 12/25/07 at 2:58pm.



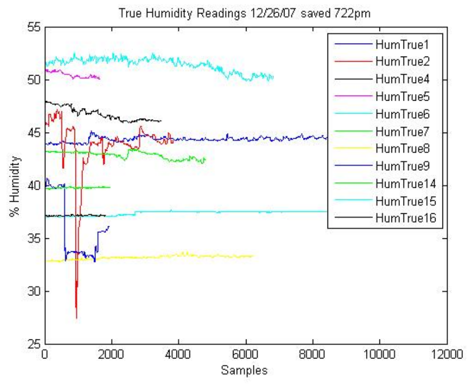
(b) Data Saved 12/25/07 at 7:29pm.



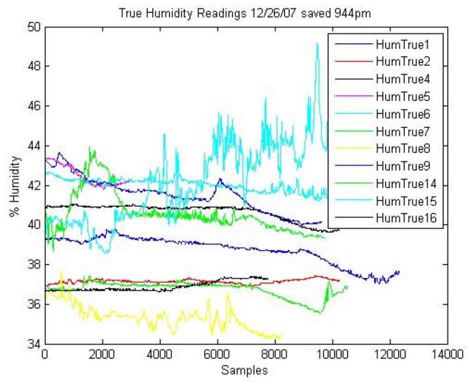
(c) Data Saved 12/25/07 at 10:48pm.



(d) Data Saved 12/25to26/07 at 10:58am.

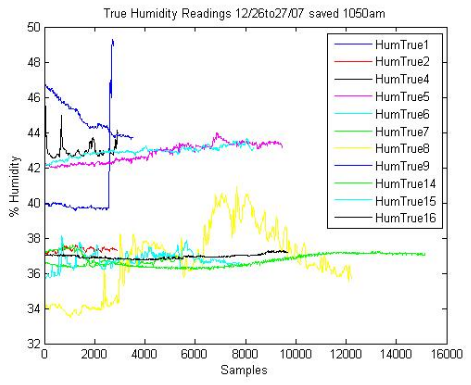


(e) Data Saved 12/26/07 at 7:22pm.

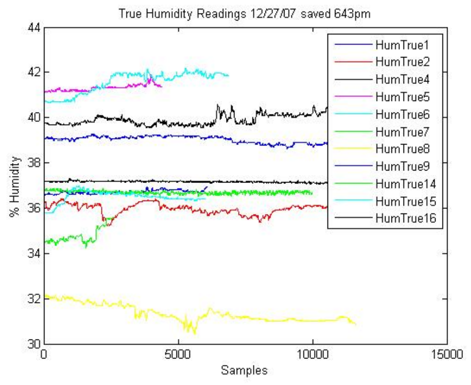


(f) Data Saved 12/26/07 at 9:44pm.

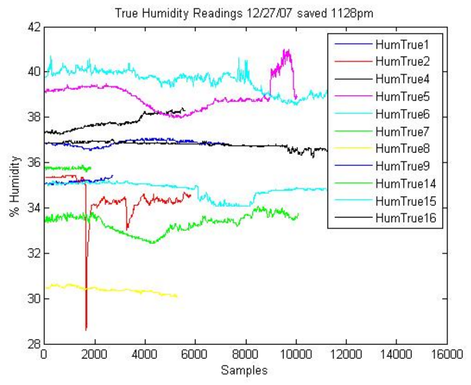
Fig. 5.6: Residence 2, 12/25/07 at 2:58pm to 12/26/07 at 9:44pm.



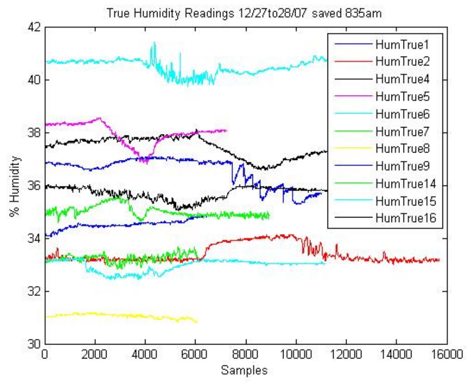
(a) Data Saved 12/26to27/07 at 10:50am.



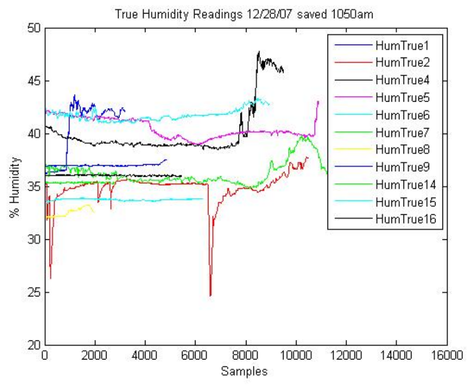
(b) Data Saved 12/27/07 at 6:43pm.



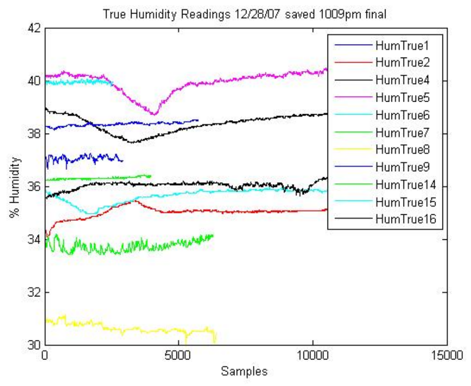
(c) Data Saved 12/27/07 at 11:28pm.



(d) Data Saved 12/27to28/07 at 8:35am.

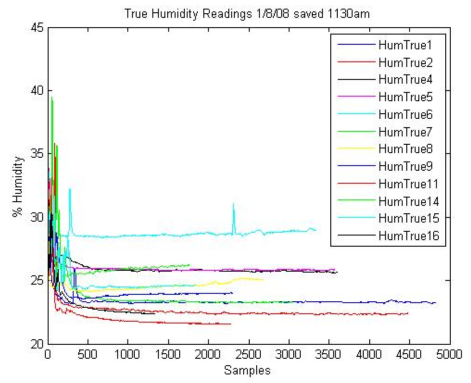


(e) Data Saved 12/28/07 at 10:50am.

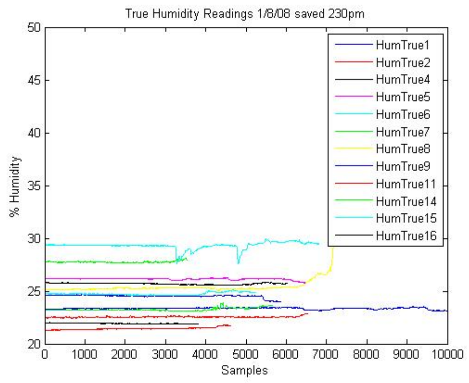


(f) Data Saved 12/28/07 at 10:09pm Final.

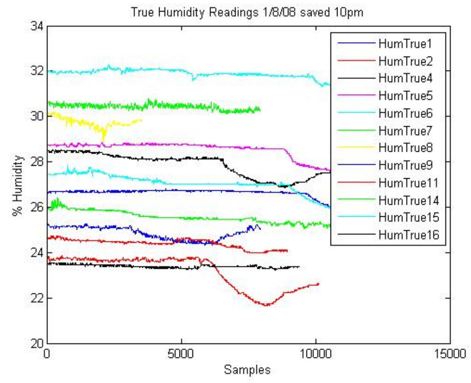
Fig. 5.7: Residence 2, 12/27/07 at 10:58am to 12/28/07 at 10:09pm.



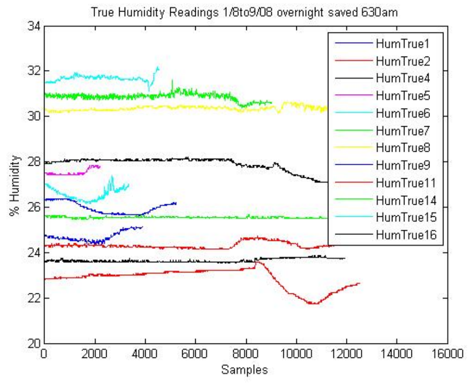
(a) Placement 1/8/08 at 11:30am.



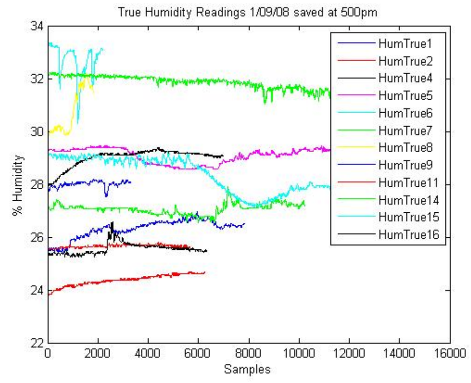
(b) Data Saved 1/8/08 at 2:30pm.



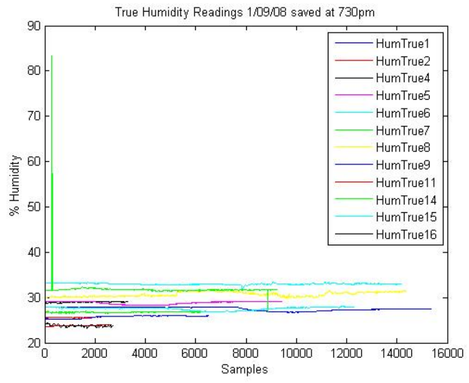
(c) Data Saved 1/8/08 at 10:00pm.



(d) Data Saved 1/8to9/08 at 6:30am.

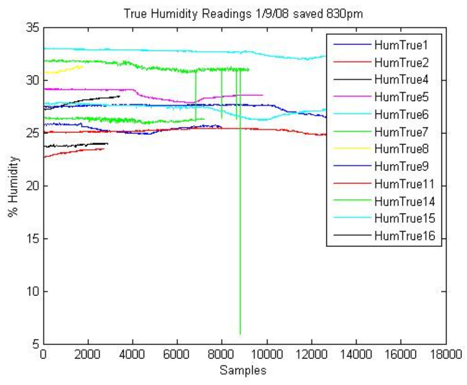


(e) Data Saved 1/9/08 at 5:00pm.

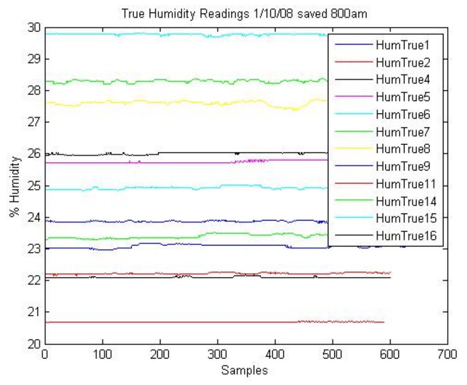


(f) Data Saved 1/9/08 at 7:30pm.

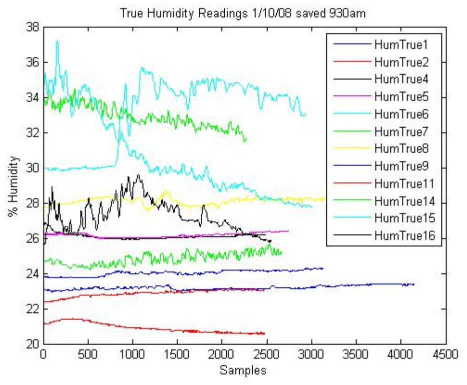
Fig. 5.8: Residence 3, 1/08/08 at 11:30am to 1/09/07 at 7:30pm.



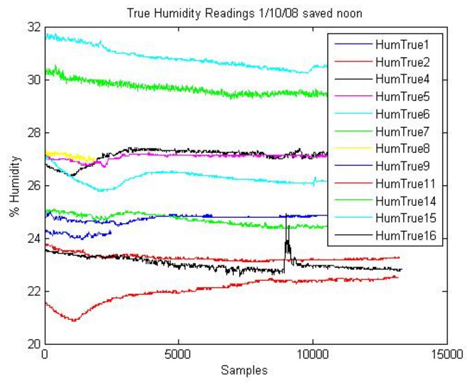
(a) Data Saved 1/9/08 at 8:30pm.



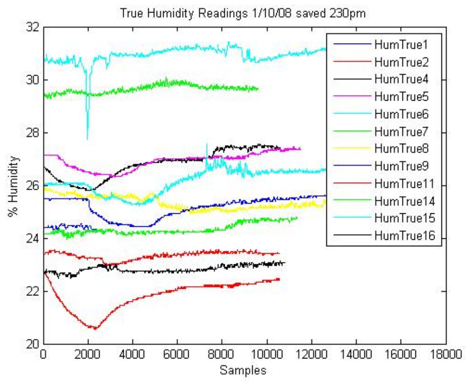
(b) Data Saved 1/10/08 at 8:00am.



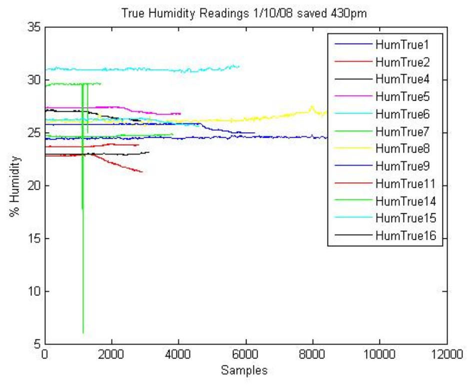
(c) Data Saved 1/10/08 at 9:30am.



(d) Data Saved 1/10/08 at 12:00pm.

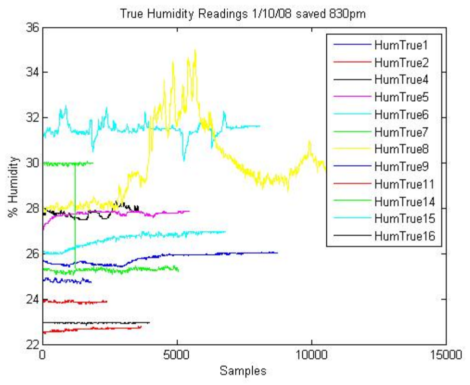


(e) Data Saved 1/10/08 at 2:30pm.

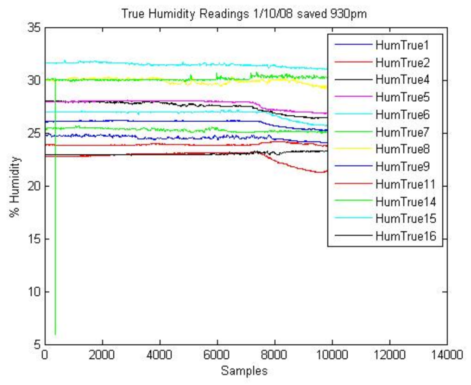


(f) Data Saved 1/10/08 at 4:30pm.

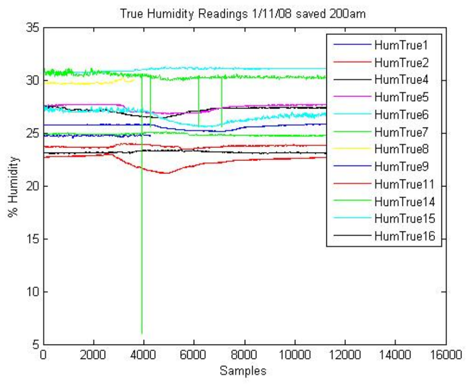
Fig. 5.9: Residence 3, 1/09/08 at 8:30pm to 1/10/08 at 4:30pm.



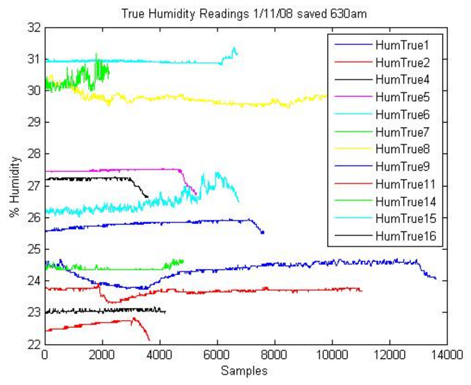
(a) Data Saved 1/10/08 at 8:30pm.



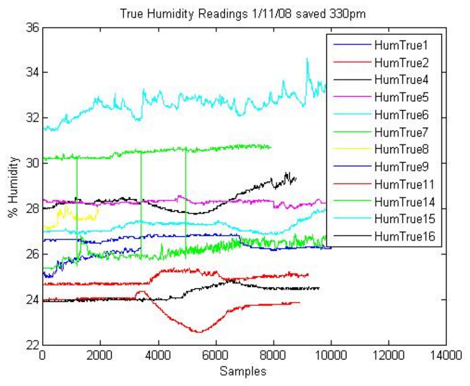
(b) Data Saved 1/10/08 at 9:30pm.



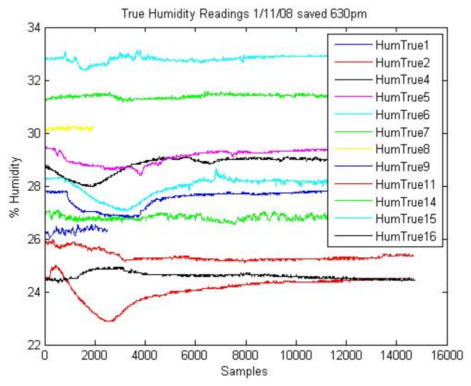
(c) Data Saved 1/11/08 at 2:00am.



(d) Data Saved 1/11/08 at 6:30am.

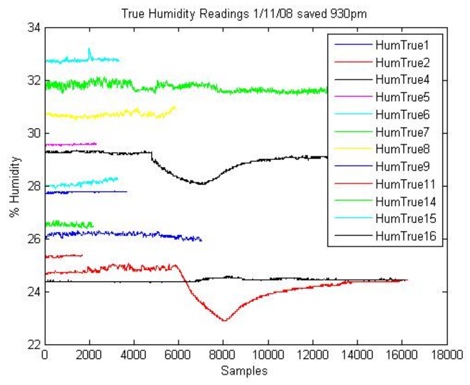


(e) Data Saved 1/11/08 at 3:30pm.

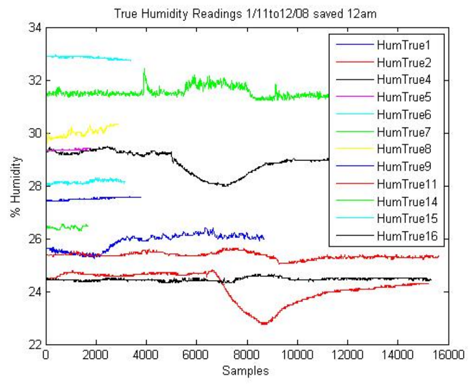


(f) Data Saved 1/11/08 at 6:30pm.

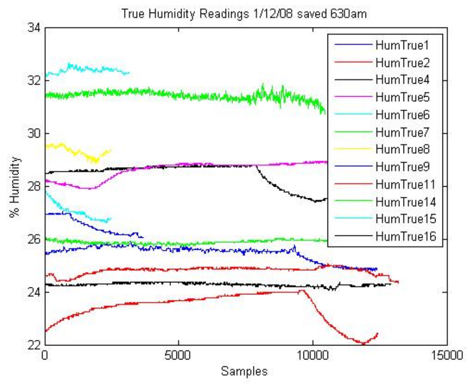
Fig. 5.10: Residence 3, 1/10/08 at 8:30pm to 1/11/08 at 6:30pm.



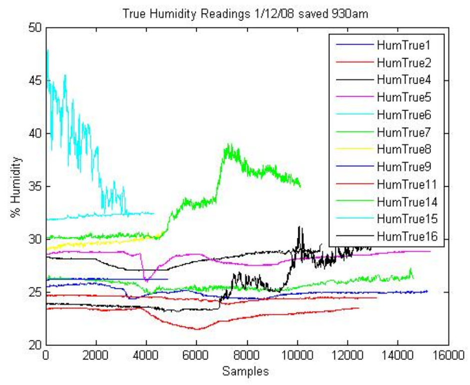
(a) Data Saved 1/11/08 at 9:30pm.



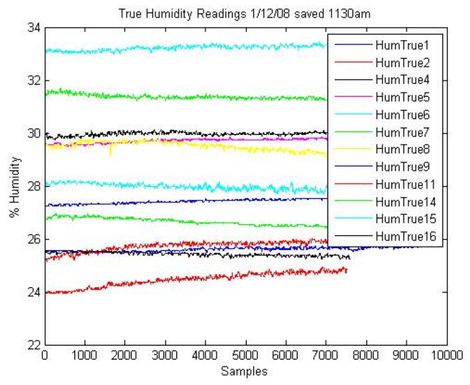
(b) Data Saved 1/12/08 at 12:00am.



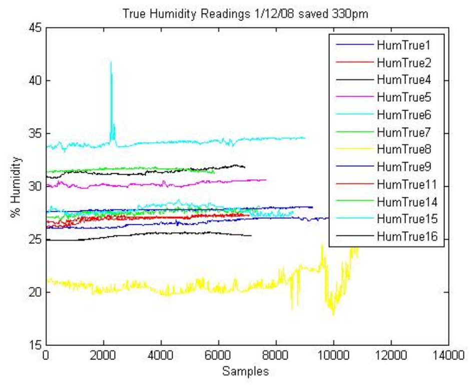
(c) Data Saved 1/12/08 at 6:30am.



(d) Data Saved 1/12/08 at 9:30am.

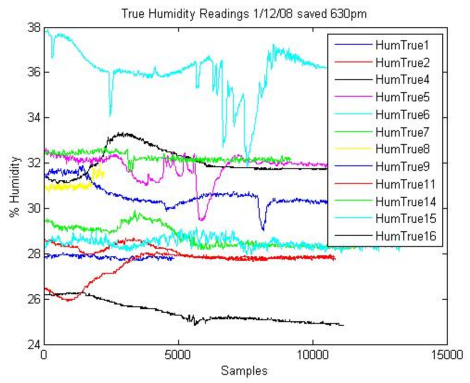


(e) Data Saved 1/12/08 at 11:30am.

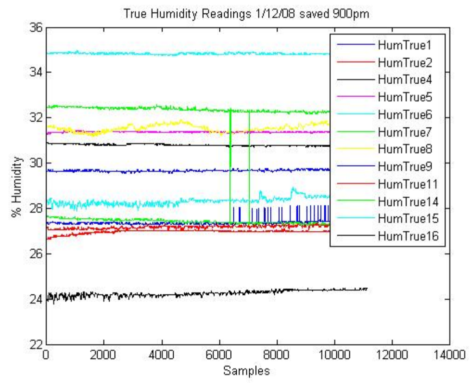


(f) Data Saved 1/12/08 at 3:30pm.

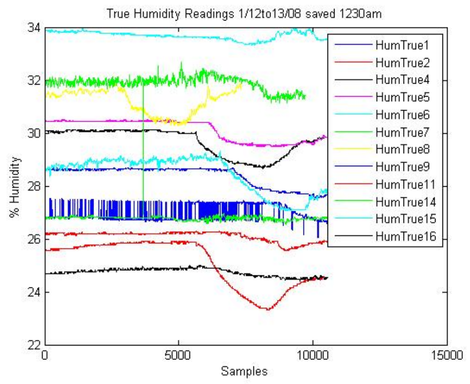
Fig. 5.11: Residence 3, 1/11/08 at 9:30pm to 1/12/08 at 3:30pm.



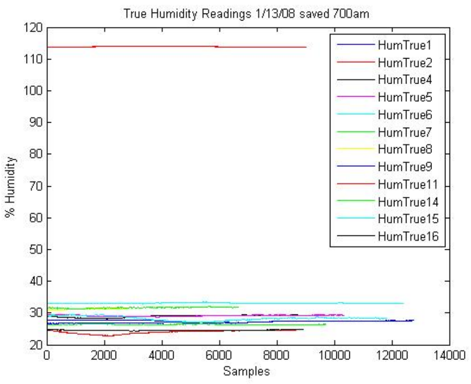
(a) Data Saved 1/12/08 at 6:30pm.



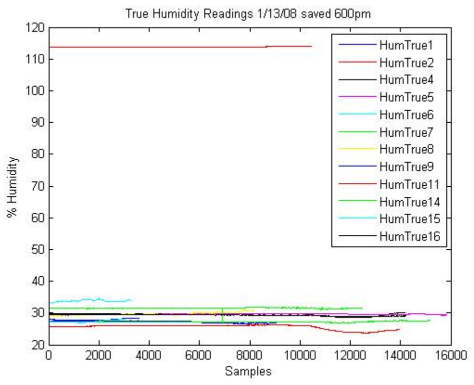
(b) Data Saved 1/12/08 at 9:00pm.



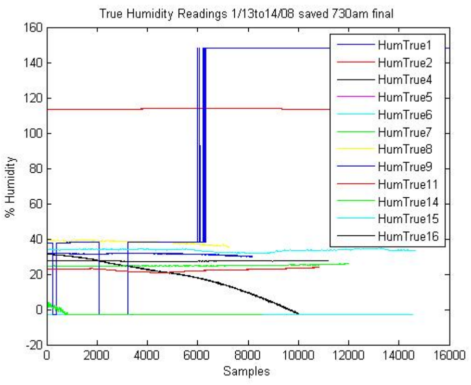
(c) Data Saved 1/13/08 at 12:30am.



(d) Data Saved 1/13/08 at 7:00am.



(e) Data Saved 1/13/08 at 6:00pm.



(f) Data Saved 1/13to14/08 at 7:30am Final.

Fig. 5.12: Residence 3, 1/12/08 at 6:30pm to 1/14/08 at 7:30am.

Chapter 6

Results and Conclusions

6.1 Fault Detection

I will first address fault detection and performance of each tmote sky module as it affects the sensor network as a whole. Fault detection is an important aspect of network monitoring [38]. Fault detection can be performed at the base station when one of the tmote sky modules is no longer transmitting data. Fault detection can also be performed within the network itself by the individual tmote sky modules. This is accomplished by implementing a multi-hop component into the tmote sky programming code [39]. TinyOS contains a multi-hop component [31]. Multi-hop allows the tmote sky module outside the transmitting range of the onboard antenna to transmit its data to another tmote sky module in the network. This second tmote sky module can then not only transmit its own data but also pass on the data received from the first module. The tmote sky module accomplishes this by first locating its nearest neighbors. It then chooses the closest neighbor along the signal path to the base station through which to route its transmission. This multi-hop technique contributes a self-healing attribute to the sensor network. If one of the tmote sky modules ceases to transmit, the remaining tmote sky modules will reroute their transmissions through a different, still functioning neighbor in the network. The antenna range of the tmote sky module is within 50m range indoors and 125m range outdoors [29]. For this project all the tmote sky modules placement locations in respect to the base station for each network were well within the transmitting range. As a result, the multi-hop component was not necessary for the sensor networks covered by this project.

6.2 Event Detection

Next I will address event detection pertaining to changes in the percentage of relative humidity levels. As mentioned in Chapter 5, the variable of interest for this project is not the individual readings collected from each tmote sky module, but the range the collected measurements span. However, the individual measurements taken by the sensors attached to the tmote sky modules dictate the data range. Therefore after data processing, an algorithm for monitoring the data range is found below.

- Implement a sorting algorithm to separate the high and low collective measurements for the interval of interest
- Compare these two data sets with the optimal range values
- Alert the network creator of a potential problem including the transmitting module numbers whose recorded data is deemed outside the optimal range

This algorithm would need to take into account individual spikes in the data measurements and noise in the transmitted signal. For this project, these spikes can be seen in the graphs towards the end of the network lifetimes. These spikes are a result of low power input to the tmote sky module caused by failing batteries. The noise in the transmitted signal can also be seen in the graphs as small deviations from the line of graphed data. Movement of the tmote sky module after placement increases the noise in the transmitted signal resulting in large transients in the graphed data. These large transients can be seen in the graphs for each network as a result of module placement. They can also be seen during transmission after an individual tmote sky module has been nudged.

6.3 Project Findings

As discussed in Chapter 1, the optimal range for relative humidity in a building to maintain both physical and structural health is between 20 to 40 percent. The average lifetime of each sensor network was approximately four days of constant data transmission. The continuous monitoring of the transmitted data also made it possible to view the behavior of the individual tmote sky modules as the power level decreased.

6.3.1 Residence 1

Figure 5.1 (a), (b), and (e) demonstrate the large transients in the signal caused by module placement and later movement discussed previously. Figure 5.2 (c), (e), and (f) as well as fig. 5.3 (a), (b), and (f) demonstrate the spikes in the signal caused by failing batteries. As can be seen from the graphs, the average range of percent relative humidity for this residence is approximately 18 to 21 percent over the network lifetime. The graphs of the measured data for the first participating residence confirm the hypothesis that the overall percentage of relative humidity in this residence during the network lifetime was below the optimal range for both physical and structural health. This hypothesis was based on the symptoms reported by the participant located in Chapter 4 Network 1 profile. The data collected from the residence of the first participant was saved in approximately 12 hour intervals. As will be discussed in the conclusion, this was not the optimal interval for maximum data sample retention using the Oscilloscope application.

6.3.2 Residence 2

Figures 5.4 through 5.7 contain the graphs of the measured data for the lifetime of the second network. The spikes in the signal signifying low power to the tmote sky module caused by failing batteries can be seen in red in figs 5.6 (f) and 5.7. The red signal according to the legend is from the data collected by tmote sky module number 2. Were this a network of longer deployment, these spikes would indicate the batteries in that module needed to be changed. The transients due to placement can be observed in fig. 5.4 (a) and (b). However, the transients due to module disturbances are for the most part non existent. This indicates that the tmote sky modules were virtually undisturbed during the network lifetime. As can be seen from the graphs, the average range of percent relative humidity for this residence is approximately 35 to 50 percent over the network lifetime. The graphs of the measured data for the second participating residence confirm the hypothesis that the overall percentage of relative humidity in this residence during the network lifetime was above the optimal range for both physical and structural health. This hypothesis was based on the symptoms reported by the participant located in Chapter 4 Network 2 profile. The data was saved at random time intervals in the range of 15 minutes to 10 hours. The length of the intervals compared with the number of data samples was explored to find the optimal interval for maximum sample retention. This optimal interval is discussed in the Conclusions.

6.3.3 Residence 3

Figures 5.8 through 5.12 contain the graphs of the measured data for the lifetime of the third network. The spikes in the signal signifying low power to the tmote sky module caused by failing batteries can be seen in fig. 5.12 (b), (c), and (f). However, the green signal of tmote sky module number 7 is also generating spikes at odd intervals throughout the network lifetime. These spikes are due to the tmote sky module placement location near the primary entrance to the residence. The event detection algorithm would need to be able to address that problem. As can be seen from the graphs, the average range of percent relative humidity for this residence is approximately 25 to 35 percent over the network lifetime. The graphs of the measured data for the third participating residence confirm the hypothesis that the overall percentage of relative humidity in this residence during the network lifetime falls in optimal range for both physical and structural health. This hypothesis was based on the symptoms reported by the participant located in Chapter 4 Network 3 profile. The data was saved at approximately four hour intervals. This interval length represent the optimal length for maximum data sample retention using the Oscilloscope application.

6.4 Conclusions

The first conclusion drawn from this sensor network project was that the Oscilloscope application was not the optimal programming code to modify and use for running this sensor network. The optimal time interval for saving the data using the Oscilloscope application code was found to be no greater than four hours. This interval length represents the optimal length for maximum data sample retention. A more efficient application programming code would write the data samples directly to a file as they were collected to minimize data loss. There were also many inconsistencies in the number of saved data samples. Apparently, the Oscilloscope application truncates the transmitted data samples after reaching a maximum number of collected samples. This maximum number is additive and not on a per transmitting module basis. The number of transmitted data samples saved for channel 0, humidity, and channel 1, temperature coming from the same tmote sky module was rarely equal even though they should have been collected iteratively and transmitted in the same packet. There was also a nonuniformity in the saved data samples that did not correlate to the module placement distance away from the base. Generally, the further from the base station the location of the transmitting module is, the fewer number of data samples are received at the base due to interference. However with this application for example, the number of data samples saved for a certain interval from a tmote sky module would be comparatively low. While the following interval resulted in the number of data samples from the same tmote sky module would be comparatively high. The second conclusion is that battery powered sensor networks are not cost effective. This holds for networks requiring the continuous transmission of data. Data transmission is the largest power consuming operation of the tmote sky module. The average lifetime of each of the networks in the project was four days. Although this interval is feasible for certain applications, generally speaking four days is not a sufficient amount of time for a monitoring application to accurately reflect the characteristics of interest in a monitored environment. Another significant factor affecting the lifetime of a battery powered sensor network is the placement location temperature. Colder temperatures significantly reduce the lifetime of the batteries.

6.5 Suggestions for Future Work

Even though the hypothesis held true for each of the three sensor networks involved in the study, four days is not a sufficient amount of time for environment monitoring. A future network would need to be deployed with a lifetime of at least a year to monitor the indoor relative humidity levels over every season. A more dense deployment of sensing modules in the area of interest would also increase the accuracy level of the overall measurement by averaging the adjacent module readings. Research is being done for more efficient power sources for sensing modules since a battery powered sensor network of a large deployment interval is not cost effective.‘

References

[1] B. O. Brooks and W. F. Davis, *Understanding Indoor Air Quality*. Boca Raton, FL: CRC Press, Inc, 1992.

[2] United States Environmental Protection Agency, "Indoor Air Quality and Student Performance," Aug. 2004.

[3] B. Krishnamachari, *Networking Wireless Sensors*. New York City, NY: Cambridge University Press, 2005.

[4] Sensirion, "Application Note: Introduction to Relative Humidity," [http://www.sensirion.com/humidity], 2007.

[5] Minnesota BLUE FLAME GAS Association, "Humidity and the Indoor Environment," [http://www.blueflame.org/datasheets/humidity.html], 2004.

[6] A. V. Arundel, E. M. Sterling, J. H. Biggin, and T. D. Sterling, "Indirect health effects of relative humidity in indoor environments," *Environmental Health Perspectives*, vol. 65, pp. 351–361, 1986.

[7] D. L. Hansen, *Indoor Air Quality Issues*. New York City, NY: Taylor and Francis, 1999.

[8] D. Shusterman and M. A. Murphy, "Nasal hyperreactivity in allergic and non-specific rhinitis: a potential risk factor for non-specific building-related illness," *Indoor Air*, vol. 17, pp. 328–333, 2007.

[9] T. A. Press, "Wrong humidity turns your house into a hassle," *USA TODAY*, 2004.

[10] B. Loecher, "The surprising truth about mold," *Prevention*, vol. 56, 2004.

[11] S. Maxwell, "Control moisture to cut odour," *The Toronto Star*, 2007.

[12] Y. Sun, J. Sundell, and Y. Zhang, "Validity of building characteristics and dorm dampness obtained in a self-administrated questionnaire," *ScienceDirect, Science of the Total Environment*, pp. 276–282, 2007.

[13] H. J. Choa, J. Schwartz, D. K. Milton, and H. A. Burge, "Populations and determinants of airborne fungi in large office buildings," *Environmental Health Perspectives*, vol. 110, pp. 777–782, 2002.

[14] Nemours Foundation reviewed by Joel Klein MD, "Adenovirus," [http://www.kidshealth.org/parent/infections/lung/adenovirus.html], 2006.

[15] Pediatric Health Care Brokton, "Coxsackie Virus or Hand Foot and Mouth Disease," [http://www.pediactrichealthcarebrokton.com/coxsackievirus.asp], 2003.

[16] Encyclopedia Britannica Online, "Rhinovirus," [http://www.britannica.com/eb/article-9063437], 2008.

[17] E. Hewitt, “How to avoid the airplane cold,” [http://www.msnbc.msn.com/id/21155643], 2007.

[18] X. Cao, J. Chen, Y. Zhang, and Y. Sun, “Development of an integrated wireless sensor network micro-environment monitoring system,” *ScienceDirect*, 2008.

[19] A. Mainwaring, J. Polastre, R. Szewczyk, D. Cutler, and J. Anderson, “Wireless Sensor Networks for Habitat Monitoring,” [http://www.csd.uwo.ca/courses/CS843b/Papers/mainwaring02.pdf], 2002.

[20] e2v technologies inc, “Infrared Sensor Application Note 1 A Background to Gas Sensing by Non-Dispersive Infrared (NDIR),” [http://www.e2v.com], 2007.

[21] M. Li and Y. Liu, “Underground Structure Monitoring with Wireless Sensor Networks,” [http://faculty.cs.tamu.edu/ajiang/UndergroundStructureMonitoring.pdf], 2007.

[22] Developed by UC Berkeley, “TinyOS,” [http://en.wikipedia.org/wiki/TinyOS], 2008.

[23] L. Girod, V. Bychkovskiy, J. Elson, and D. Estrin, “Locating tiny sensors in time and space: A case study,” [http://lecs.cs.ucla.edu/Publications/papers/iccd-2002.pdf], 2002.

[24] L. Balzano and R. Kowak, “Blind Calibration of Sensor Networks,” [http://nesl.ee.ucla.edu/fw/documents/conference/2007/ipsn160-balzanonowak.pdf], 2007.

[25] L. K. Balzano, “Addressing fault and calibration in wireless sensor networks,” Master’s thesis, University of California, Los Angeles, 2007.

[26] M. Rabbat and R. Nowak, “Distributed Optimization in Sensor Networks,” [http://www.ece.wisc.edu/ nowak/dosn.pdf].

[27] Z. Song, *Optimal Observation Problems Involving Wireless Sensor Networks*. Ph.D. dissertation, Utah State University, 2007.

[28] D. J. Abadi, S. Madden, and W. Linder, “REED: Robust, Efficient Filtering and Event Detection in Sensor Networks,” [http://people.csail.mit.edu/wolfgang/papers/VLDB05.pdf], 2005.

[29] Moteiv Corporation, “tmote sky Low-Power Wireless Sensor Module,” [http://www.moteiv.com], 2006.

[30] Sensirion, “SHT1x/SHT7x Humidity and Temperature Sensor,” [http://www.sensirion.com], 2007.

[31] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Cutler, “The nesC Language: A Holistic Approach to Networked Embedded Systems,” [http://nescc.sourceforge.net].

[32] Sensirion, “Calibration Certificate Humidity SHTxx E,” [http://www.sensirion.com/humidity], 2007.

[33] P. Levis, “TinyOS Programming,” [http://csl.stanford.edu/ pal/pubs/tinyos-programming.pdf], 2006.

[34] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, “System Architecture Directions for Networked Sensors,” [http://www.stanford.edu/class/cs344a/papers/tos.pdf], 2000.

[35] Developed by Cygnus Solutions, “Cygwin,” [http://cygwin.com/docs.html], 2008.

[36] K. W. Tham, M. S. Zuraimi, D. Koh, F. T. Chew, and P. L. Ooi, “Associations between home dampness and presence of molds with asthma and allergic symptoms among young children in the tropics,” *Pediatric Allergy and Immunology*, vol. 18, pp. 418–424, 2007.

[37] Sensirion, “Application Note: Nonlinearity Compensation,” [http://www.sensirion.com/humidity], 2007.

[38] R. E. V. Dyck, “Detection performance in self-organized wireless sensor networks,” *IEEE International Symposium on Information Theory*, p. 13, 2002.

[39] R. Draves, J. Padhye, and B. Zill, “Routing in Multi-Hop Radio, Multi-Hop Wireless Mesh Networks,” [http://research.microsoft.com/mesh/papers/multiradio.pdf].

Appendices

Appendix A

Modified Oscilloscope and TOSBase Application Code

A.1 Copy Right Notice

```
* copies can be found at http://www.moteiv.com/MOTEIV-LICENSE.txt
* and by emailing info@moteiv.com.
```

A.2 OscilloscopeTmoteSky.nc

```
@author Joe Polastre, Moteiv Corporation <info@moteiv.com>
modified Lizabeth Lee 2007 configuration OscilloscopeTmoteSky { }
implementation { components Main
           , OscilloscopeTmoteSkyM as OscilloscopeM
           , TimerC
           , LedsC
           , HumidityC
           , OscopeC
           , GenericComm as Comm;
components DelugeC;
Main.StdControl -> TimerC;
Main.StdControl -> Comm;
Main.StdControl -> OscopeC;
Main.StdControl -> OscilloscopeM;
OscilloscopeM.Timer -> TimerC.Timer[unique("Timer")];
OscilloscopeM.Leds -> LedsC;
OscilloscopeM.HumidityControl -> HumidityC;
OscilloscopeM.Humidity -> HumidityC.Humidity;
OscilloscopeM.Temperature -> HumidityC.Temperature;
OscilloscopeM.HumidityError -> HumidityC.HumidityError;
OscilloscopeM.TemperatureError -> HumidityC.TemperatureError;
OscilloscopeM.OHumidity -> OscopeC.Oscope[0];
OscilloscopeM.OTemperature -> OscopeC.Oscope[1];}
```

A.3 OscilloscopeTmoteSkyM.nc

```
* @author Joe Polastre, Moteiv Corporation <info@moteiv.com>
modified Lizabeth Lee 2007
module OscilloscopeTmoteSkyM {
provides interface StdControl;
uses {
    interface Timer;
    interface Leds;
    interface SplitControl as HumidityControl;
    interface ADC as Humidity;
    interface ADC as Temperature;
    interface Oscope as OHumidity;
    interface Oscope as OTemperature;
    interface ADCError as HumidityError;
    interface ADCError as TemperatureError; }
} implementation {
  enum {
    OSCOPE_DELAY = 10,
};
  enum {
    HUMIDITY,
    TEMPERATURE,
};
norace uint16_t humidity, temperature;
norace int state;
// Used to initialize this component.
command result_t StdControl.init() {
    call Leds.init();
    call Leds.set(0);
    state = HUMIDITY;
//turn on the sensors so that they can be read.
call HumidityControl.init();
    return SUCCESS;
}
event result_t HumidityControl.initDone() {
    return SUCCESS;
}
/**
* Starts the SensorControl component.
* @return Always returns SUCCESS.
*/
command result_t StdControl.start() {
    call HumidityControl.start();
    return SUCCESS;
}
event result_t HumidityControl.startDone() {
    call HumidityError.enable();
    call TemperatureError.enable();
    call Timer.start( TIMER_ONE_SHOT, 250 );
    return SUCCESS;
}
/**
* Stops the SensorControl component.
* @return Always returns SUCCESS.
*/
command result_t StdControl.stop() {
    call HumidityControl.stop();
    call Timer.stop();
    return SUCCESS;
}
event result_t HumidityControl.stopDone() {
    call HumidityError.disable();
    call TemperatureError.disable();
    return SUCCESS;
}
event result_t Timer.fired() {
    // set a timeout in case a task post fails (rare)
    call Timer.start(TIMER_ONE_SHOT, 100);
    switch(state) {
    case HUMIDITY:
      call Humidity.getData();
      break;
    case TEMPERATURE:
      call Temperature.getData();
      break;
    default:
      call Timer.start(TIMER_ONE_SHOT, 10);
   }
    return SUCCESS;
}
task void putHumidity() {
    call OHumidity.put(humidity);
    call Leds.yellowOn();
    call Timer.start(TIMER_ONE_SHOT, OSCOPE_DELAY);
```

```
}
task void putTemperature() {
    call OTemperature.put(temperature);
    call Leds.set(0);
    call Timer.start(TIMER_ONE_SHOT, OSCOPE_DELAY);
}
async event result_t Humidity.dataReady(uint16_t data) {
    humidity = data;
    post putHumidity();
    state = TEMPERATURE;
    return SUCCESS;
}
event result_t HumidityError.error(uint8_t token) {
    humidity = 0;
    post putHumidity();
    state = TEMPERATURE;
    return SUCCESS;
}
async event result_t Temperature.dataReady(uint16_t data) {
    temperature = data;
    post putTemperature();
    state = HUMIDITY;
    return SUCCESS;
}
event result_t TemperatureError.error(uint8_t token) {
    temperature = 0;
    post putTemperature();
    state = HUMIDITY;
    return SUCCESS;
   }
}
```

The following two sections of code are reproduced without modification. The code is responsible for data transmission via radio waves and is utilized by the MOTEIV Corporation for its applications. The Oscilloscope application reproduced above transmits its data by accessing this code.

A.4 TOSBase.nc

```
* Copyright (c) 2002-2003 Intel Corporation
* All rights reserved. *
* This file is distributed under the terms in the attached INTEL-
* LICENSE file. If you do not find these files, copies can be
* found by writing to Intel Research Berkeley, 2150 Shattuck Ave.
* Suite 1300, Berkeley, CA, 94704. Attention: Intel License
* Inquiry. @author Phil Buonadonna * @author Gilman Tolle */
configuration TOSBase {}
implementation {
components Main, TOSBaseM, RadioCRCPacket as Comm, FramerM, UART,
LedsC;
Main.StdControl -> TOSBaseM;
TOSBaseM.UARTControl -> FramerM;
TOSBaseM.UARTSend -> FramerM;
TOSBaseM.UARTReceive -> FramerM;
TOSBaseM.UARTTokenReceive -> FramerM;
TOSBaseM.RadioControl -> Comm;
TOSBaseM.RadioSend -> Comm;
TOSBaseM.RadioReceive -> Comm;
TOSBaseM.Leds -> LedsC;
FramerM.ByteControl -> UART;
FramerM.ByteComm -> UART;}
```

A.5 TOSBaseM.nc

```
* Copyright (c) 2002-2003 Intel Corporation
* All rights reserved. *
* This file is distributed under the terms in the attached INTEL-
* LICENSE file. If you do not find these files, copies can be
* found by writing to Intel Research Berkeley, 2150 Shattuck Ave.
* Suite 1300, Berkeley, CA,94704. Attention: Intel License
* Inquiry. *//* @author Phil Buonadonna * @author Gilman Tolle
* Revision: $Id: TOSBaseM.nc 804 2006-05-13 20:26:40Z polastre
* TOSBaseM bridges packets between serial channel and the radio.
* Messages moving from serial to radio will be tagged with the
* group ID compiled into the TOSBase, and messages moving from
* radio toserial will be filtered by that same group id. */
ifndef TOSBASE-BLINK-ON-DROP define TOSBASE-BLINK-ON-DROP endif
module TOSBaseM {
  provides interface StdControl;
  uses {
    interface StdControl as UARTControl;
    interface BareSendMsg as UARTSend;
    interface ReceiveMsg as UARTReceive;
    interface TokenReceiveMsg as UARTTokenReceive;
    interface StdControl as RadioControl;
    interface BareSendMsg as RadioSend;
    interface ReceiveMsg as RadioReceive;
    interface Leds;
  }
}
implementation {
  enum {
    UART-QUEUE-LEN = 12,
    RADIO-QUEUE-LEN = 12,
};
TOS-Msg    uartQueueBufs[UART-QUEUE-LEN];
uint8-t    uartIn, uartOut, uartCount;
bool       uartBusy;
TOS-Msg    radioQueueBufs[RADIO-QUEUE-LEN];
uint8-t    radioIn, radioOut, radioCount;
bool       radioBusy;
task void UARTSendTask();
task void RadioSendTask();
void failBlink();
void dropBlink();
void processUartPacket(TOS-MsgPtr Msg, bool wantsAck,
uint8-t Token);
command result-t StdControl.init() {
    result-t ok1, ok2, ok3;
    uartIn = uartOut = uartCount = 0;
    uartBusy = FALSE;
    radioIn = radioOut = radioCount = 0;
    radioBusy = FALSE;
    ok1 = call UARTControl.init();
    ok2 = call RadioControl.init();
    ok3 = call Leds.init();
    dbg(DBG-BOOT, "TOSBase initialized-n");
    return rcombine3(ok1, ok2, ok3);
}
command result-t StdControl.start() {
    result-t ok1, ok2;
    ok1 = call UARTControl.start();
    ok2 = call RadioControl.start();
    return rcombine(ok1, ok2);
}
command result-t StdControl.stop() {
    result-t ok1, ok2;
    ok1 = call UARTControl.stop();
    ok2 = call RadioControl.stop();
    return rcombine(ok1, ok2);
}
event TOS-MsgPtr RadioReceive.receive(TOS-MsgPtr Msg) {
    dbg(DBG-USR1, "TOSBase received radio packet.-n");
    if ((!Msg->crc) || (Msg->group != TOS-AM-GROUP))
      return Msg;
    if (uartCount < UART-QUEUE-LEN) {
      memcpy(uartQueueBufs[uartIn], Msg, sizeof(TOS-Msg));
      uartCount++;
      if( ++uartIn >= UART-QUEUE-LEN ) uartIn = 0;
      if (!uartBusy) {
    if (post UARTSendTask()) {
      uartBusy = TRUE;
    }
      }
    } else {
      dropBlink();
    }
    return Msg;
}
task void UARTSendTask() {
    dbg (DBG-USR1, "TOSBase forwarding Radio packet to UART-n");
    if (uartCount == 0) {
      uartBusy = FALSE;
    } else {
      if (call UARTSend.send(uartQueueBufs[uartOut]) == SUCCESS)
    {call Leds.greenToggle();
      } else {
    failBlink();
    post UARTSendTask();
      }
    }
}
event result-t UARTSend.sendDone(TOS-MsgPtr msg,result-t success)
     {if (!success) {
      failBlink();
    } else {
      uartCount--;
      if( ++uartOut >= UART-QUEUE-LEN ) uartOut = 0;
    }
    post UARTSendTask();
    return SUCCESS;
}
```

```
event TOS-MsgPtr UARTReceive.receive(TOS-MsgPtr Msg) {
    processUartPacket(Msg, FALSE, 0);
    return Msg;
} event TOS-MsgPtr UARTTokenReceive.receive(TOS-MsgPtr Msg,
  uint8-t Token) {
    processUartPacket(Msg, TRUE, Token);
    return Msg;
} void processUartPacket(TOS-MsgPtr Msg, bool wantsAck,
  uint8-t Token) {
    bool reflectToken = FALSE;
    dbg(DBG-USR1, "TOSBase received UART token packet.-n");
    if (radioCount < RADIO-QUEUE-LEN) {
      reflectToken = TRUE;
      memcpy(radioQueueBufs[radioIn], Msg, sizeof(TOS-Msg));
      radioCount++;
      if( ++radioIn >= RADIO-QUEUE-LEN ) radioIn = 0;
      if (!radioBusy) {
    if (post RadioSendTask()) {
      radioBusy = TRUE;
    }
      }
    } else {
      dropBlink();
    }
    if (wantsAck  reflectToken) {
      call UARTTokenReceive.ReflectToken(Token);
    }
}
```

```
task void RadioSendTask() {
    dbg(DBG-USR1, "TOSBase forwarding UART packet to Radio-n");
    if (radioCount == 0) {
      radioBusy = FALSE;
    } else {
      radioQueueBufs[radioOut].group = TOS-AM-GROUP;
      if (call RadioSend.send(radioQueueBufs[radioOut])==SUCCESS)
     {call Leds.redToggle();
      } else {
    failBlink();
    post RadioSendTask();
      }
    }
}
event result-t RadioSend.sendDone(TOS-MsgPtr msg,result-t success)
     {if (!success) {
      failBlink();
    } else {
      radioCount--;
      if( ++radioOut >= RADIO-QUEUE-LEN ) radioOut = 0;
    }
    post RadioSendTask();
    return SUCCESS;
}
void dropBlink() {
ifdef TOSBASE-BLINK-ON-DROP
    call Leds.yellowToggle();
endif
}
void failBlink() {
ifdef TOSBASE-BLINK-ON-FAIL
    call Leds.yellowToggle();
endif
  }
}
```

Appendix B

C++ Code for Data File Separation

```
#include <iostream> #include <fstream> #include <string> using
namespace std;

int main () {
//Dan Stuart,2-19-08,define string to hold line of values
  string line;
//Dan Stuart,2-19-08,holds lines of garbage we just read in.
  string garbage;
  char buffer [33]; //Dan Stuart,2-19-08,Hold number for file
  int cnt = 0;
  int flag = 0;
//Dan Stuart,2-19-08,Allocate space for 24 data files
  ofstream DataCollection[24];

//Dan Stuart,2-19-08,Open 24 data files
//Check to see if they opened!
  for(int x=0;x<24;x++)
  {
    string FileNumber = itoa(x,buffer,10);
    string fileName = "DataOutput"+FileNumber+".txt";
    DataCollection[x].open(fileName.c_str(), ios::out);
    if (!DataCollection[x].is_open())
    {
        printf("Your DataFile %d did not open!!!\n",x);
    }
  }

//Dan Stuart,2-19-08, open data file that contains data
  ifstream myDataFile("twentytwo8to12am");
  if (!myDataFile.is_open())
  {
//Dan Stuart,2-19-08,check to see if the file opened else quit
    cout << "Your Data File did not open!!!\n";
    return 0;
  }
//Dan Stuart, 2-19-08, read in 4 lines of garbage from file
  getline(myDataFile,garbage);
  getline(myDataFile,garbage);
  getline(myDataFile,garbage);
//Dan Stuart, 2-19-08, Write data file to current file of 24
  DataCollection[cnt] << garbage << endl;
  getline(myDataFile,garbage);



  while (!myDataFile.eof() )
    {
      line = "";
      string linetag = "# BEGIN";
      //print mote/ch# at bebinning of data
      DataCollection[cnt] << garbage << endl;
      while((strncmp(line.c_str(),linetag.c_str(),7)!= 0)
              && (flag !=1))
        { //Dan Stuart, 2-19-08,Read in a line from DataFile
             getline (myDataFile,line);
     //Dan Stuart,2-19-08, Write data file to current file of 24
             DataCollection[cnt] << line << endl;
     //Dan Stuart, 2-19-08, Write file to console for debugging
             cout << line << endl;
             if(myDataFile.eof()) flag = 1;
        }
    //Dan Stuart, 2-19-08, read in a line for garbage
      getline(myDataFile,garbage);
    //Dan Stuart, 2-19-08,Write data file to current file of 24
      DataCollection[cnt] << garbage << endl;
    //Dan Stuart,2-19-08,Close current data output file of 24
      DataCollection[cnt].close();
    //Dan Stuart,2-19-08,Increment to a new file for writing
      cnt++;
    }
    myDataFile.close();//Dan Stuart,2-19-08,Close the Data File
  return 0;
}
```

Appendix C
MATLAB Code for Raw Data Conversion

Written by Lizabeth Lee.

Contents

```
%take raw temp data and convert it to degrees farenheit
%the x axis is samples so far
clear all
close all  %closes all open figures
```

Import Raw Data

```
fid = fopen('mote1_temp_ch1.txt', 'r');
mote1_temp_ch1 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);

fid = fopen('mote2_temp_ch1.txt', 'r');
mote2_temp_ch1 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);
```

```
fid = fopen('mote4_temp_ch1.txt', 'r');
mote4_temp_ch1 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);

fid = fopen('mote5_temp_ch1.txt', 'r');
mote5_temp_ch1 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);

fid = fopen('mote6_temp_ch1.txt', 'r');
mote6_temp_ch1 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);

fid = fopen('mote7_temp_ch1.txt', 'r');
mote7_temp_ch1 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);

fid = fopen('mote8_temp_ch1.txt', 'r');
mote8_temp_ch1 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);

fid = fopen('mote9_temp_ch1.txt', 'r');
mote9_temp_ch1 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);

fid = fopen('mote11_temp_ch1.txt', 'r');
mote11_temp_ch1 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);
```

```
fid = fopen('mote14_temp_ch1.txt', 'r');
mote14_temp_ch1 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);

fid = fopen('mote15_temp_ch1.txt', 'r');
mote15_temp_ch1 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);

fid = fopen('mote16_temp_ch1.txt', 'r');
mote16_temp_ch1 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);

fid = fopen('mote1_hum_ch0.txt', 'r');
mote1_hum_ch0 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);

fid = fopen('mote2_hum_ch0.txt', 'r');
mote2_hum_ch0 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);

fid = fopen('mote4_hum_ch0.txt', 'r');
mote4_hum_ch0 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);

fid = fopen('mote5_hum_ch0.txt', 'r');
mote5_hum_ch0 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);
```

```
fid = fopen('mote6_hum_ch0.txt', 'r');
mote6_hum_ch0 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);

fid = fopen('mote7_hum_ch0.txt', 'r');
mote7_hum_ch0 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);

fid = fopen('mote8_hum_ch0.txt', 'r');
mote8_hum_ch0 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);

fid = fopen('mote9_hum_ch0.txt', 'r');
mote9_hum_ch0 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);

fid = fopen('mote11_hum_ch0.txt', 'r');
mote11_hum_ch0 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);

fid = fopen('mote14_hum_ch0.txt', 'r');
mote14_hum_ch0 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);

fid = fopen('mote15_hum_ch0.txt', 'r');
mote15_hum_ch0 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);
```

```
fid = fopen('mote16_hum_ch0.txt', 'r');
mote16_hum_ch0 = textscan(fid, '%f %f','headerlines', 2);
fclose(fid);
```

Assign Variables

```
rawT1 = cell2mat(mote1_temp_ch1(:,2));
rawT2 = cell2mat(mote2_temp_ch1(:,2));
rawT4 = cell2mat(mote4_temp_ch1(:,2));
rawT5 = cell2mat(mote5_temp_ch1(:,2));
rawT6 = cell2mat(mote6_temp_ch1(:,2));
rawT7 = cell2mat(mote7_temp_ch1(:,2));
rawT8 = cell2mat(mote8_temp_ch1(:,2));
rawT9 = cell2mat(mote9_temp_ch1(:,2));
rawT11 = cell2mat(mote11_temp_ch1(:,2));
rawT14 = cell2mat(mote14_temp_ch1(:,2));
rawT15 = cell2mat(mote15_temp_ch1(:,2));
rawT16 = cell2mat(mote16_temp_ch1(:,2));

rawH1 = cell2mat(mote1_hum_ch0(:,2));
rawH2 = cell2mat(mote2_hum_ch0(:,2));
rawH4 = cell2mat(mote4_hum_ch0(:,2));
rawH5 = cell2mat(mote5_hum_ch0(:,2));
rawH6 = cell2mat(mote6_hum_ch0(:,2));
rawH7 = cell2mat(mote7_hum_ch0(:,2));
rawH8 = cell2mat(mote8_hum_ch0(:,2));
rawH9 = cell2mat(mote9_hum_ch0(:,2));
rawH11 = cell2mat(mote11_hum_ch0(:,2));
```

```
rawH14 = cell2mat(mote14_hum_ch0(:,2));
rawH15 = cell2mat(mote15_hum_ch0(:,2));
rawH16 = cell2mat(mote16_hum_ch0(:,2));
```

Corrections for Data Sample #Mismatch

```
rawH1 = rawH1(1:6500,1);
rawH2 = rawH2(1:1872,1);
rawH4 = rawH4(1:3292,1);
%rawH5 = rawH5(1:12694,1);
rawH6 = rawH6(1:14182,1);
rawT7 = rawT7(1:6912,1);
rawH8 = rawH8(1:14356,1);
rawH9 = rawH9(1:15362,1);
rawT11 = rawT11(1:2632,1);
rawH14 = rawH14(1:9248,1);
rawH15 = rawH15(1:12292,1);
rawH16 = rawH16(1:2702,1);
```

Calculate Temperature From Raw Data

```
%calculate temp in degrees
%Fahrenheit and Celsius
tempC1 = -39.60 + 0.01.*rawT1;
tempF1 = (9/5).*tempC1 + 32;

tempC2 = -39.60 + 0.01.*rawT2;
tempF2 = (9/5).*tempC2 + 32;

tempC4 = -39.60 + 0.01.*rawT4;
```

```
tempF4 = (9/5).*tempC4 + 32;

tempC5 = -39.60 + 0.01.*rawT5;
tempF5 = (9/5).*tempC5 + 32;

tempC6 = -39.60 + 0.01.*rawT6;
tempF6 = (9/5).*tempC6 + 32;

tempC7 = -39.60 + 0.01.*rawT7;
tempF7 = (9/5).*tempC7 + 32;

tempC8 = -39.60 + 0.01.*rawT8;
tempF8 = (9/5).*tempC8 + 32;

tempC9 = -39.60 + 0.01.*rawT9;
tempF9 = (9/5).*tempC9 + 32;

tempC11 = -39.60 + 0.01.*rawT11;
tempF11 = (9/5).*tempC11 + 32;

tempC14 = -39.60 + 0.01.*rawT14;
tempF14 = (9/5).*tempC14 + 32;

tempC15 = -39.60 + 0.01.*rawT15;
tempF15 = (9/5).*tempC15 + 32;

tempC16 = -39.60 + 0.01.*rawT16;
tempF16 = (9/5).*tempC16 + 32;
```

Calculate True Humidity (Temperature Compensated)

```
Hum1 = -4 + 0.0405.*rawH1 + (-2.8*10^(-6).*(rawH1.*rawH1));
HumTrue1 = (tempC1 - 25).*(0.01 + 0.00008.*rawH1) + Hum1;


Hum2 = -4 + 0.0405.*rawH2 + (-2.8*10^(-6).*(rawH2.*rawH2));
HumTrue2 = (tempC2 - 25).*(0.01 + 0.00008.*rawH2) + Hum2;


Hum4 = -4 + 0.0405.*rawH4 + (-2.8*10^(-6).*(rawH4.*rawH4));
HumTrue4 = (tempC4 - 25).*(0.01 + 0.00008.*rawH4) + Hum4;


Hum5 = -4 + 0.0405.*rawH5 + (-2.8*10^(-6).*(rawH5.*rawH5));
HumTrue5 = (tempC5 - 25).*(0.01 + 0.00008.*rawH5) + Hum5;


Hum6 = -4 + 0.0405.*rawH6 + (-2.8*10^(-6).*(rawH6.*rawH6));
HumTrue6 = (tempC6 - 25).*(0.01 + 0.00008.*rawH6) + Hum6;


Hum7 = -4 + 0.0405.*rawH7 + (-2.8*10^(-6).*(rawH7.*rawH7));
HumTrue7 = (tempC7 - 25).*(0.01 + 0.00008.*rawH7) + Hum7;


Hum8 = -4 + 0.0405.*rawH8 + (-2.8*10^(-6).*(rawH8.*rawH8));
HumTrue8 = (tempC8 - 25).*(0.01 + 0.00008.*rawH8) + Hum8;


Hum9 = -4 + 0.0405.*rawH9 + (-2.8*10^(-6).*(rawH9.*rawH9));
HumTrue9 = (tempC9 - 25).*(0.01 + 0.00008.*rawH9) + Hum9;


Hum11 = -4 + 0.0405.*rawH11 + (-2.8*10^(-6).*(rawH11.*rawH11));
HumTrue11 = (tempC11 - 25).*(0.01 + 0.00008.*rawH11) + Hum11;
```

```
Hum14 = -4 + 0.0405.*rawH14 + (-2.8*10^(-6).*(rawH14.*rawH14));
HumTrue14 = (tempC14 - 25).*(0.01 + 0.00008.*rawH14) + Hum14;

Hum15 = -4 + 0.0405.*rawH15 + (-2.8*10^(-6).*(rawH15.*rawH15));
HumTrue15 = (tempC15 - 25).*(0.01 + 0.00008.*rawH15) + Hum15;

Hum16 = -4 + 0.0405.*rawH16 + (-2.8*10^(-6).*(rawH16.*rawH16));
HumTrue16 = (tempC16 - 25).*(0.01 + 0.00008.*rawH16) + Hum16;
```

Plot Data

```
figure(1) plot(HumTrue1)
hold on
plot(HumTrue2,’r’)
plot(HumTrue4,’k’)
plot(HumTrue5,’m’)
plot(HumTrue6,’c’)
plot(HumTrue7,’g’)
plot(HumTrue8,’y’)
plot(HumTrue9,’b’)
plot(HumTrue11,’r’)
plot(HumTrue14,’g’)
plot(HumTrue15,’c’)
plot(HumTrue16,’k’)
xlabel(’Samples’)
ylabel(’% Humidity’)
title(’True Humidity Readings 1/8to9/08 overnight saved 630am’)
legend(’HumTrue1’,’HumTrue2’,’HumTrue4’,’HumTrue5’,’HumTrue6’,
’HumTrue7’,’HumTrue8’,’HumTrue9’,’HumTrue11’,’HumTrue14’,
'HumTrue15','HumTrue16')
```