Personalized Building Comfort Control

by

Mark Christopher Feldmeier

S.M., Massachusetts Institute of Technology (2003)
S.B., Massachusetts Institute of Technology (1996)

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2009

Author [illegible]
Program in Media Arts and Sciences
September 4, 2009

Certified by [illegible]
Joseph A. Paradiso
Associate Professor
Program in Media Arts and Sciences
Thesis Supervisor

Accepted by [illegible]
Deb K. Roy
Chair
Departmental Committee on Graduate Students

Personalized Building Comfort Control

by

Mark Christopher Feldmeier

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on September 4, 2009, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Media Arts and Sciences

Abstract

Creating an appropriate indoor climate is essential to worker productivity and personal happiness. It is also an area of large expenditure for building owners. And, with rising fuel costs, finding ways of reducing energy consumption is more important than ever. This idea is promoted further by the notion that most buildings are currently being run inefficiently, due to the non-adaptable nature of their control systems. Not just the occupants, but also the buildings themselves have ever changing needs, for which a single setpoint is inadequate.

This dissertation presents a novel air-conditioning control system, focused around the individual, which remedies these inefficiencies through the creation of personalized environments. To date, the measurement of thermal preference has been limited to either a complex set of sensors attempting to determine a Predicted Mean Vote (PMV) value, or to direct polling of the user. The former is far too cumbersome and expensive for practical application, and the latter places an undue burden on the user. To overcome these limitations, an extremely low power, light weight, wireless sensor is developed which can measure temperature, humidity, activity and light level directly on the user's body. These data are used to immediately infer user comfort level, and to control an HVAC system in an attempt to minimize both cost and thermal discomfort.

Experimental results are presented from a building under continual usage, modified with a wireless network with multiple sensing and actuating modalities. For four weeks, ten building occupants, in four offices and one common space, are thermally regulated via wrist-worn sensors controlling the local air-conditioning dampers and window operator motors. Comparisons are made to the previous four week period of standard air-conditioning control, showing an increase in comfort, while decreasing energy usage at the same time. The difficult problems of control adaptation, comfort determination, and user conflict resolution are addressed. Finally, the limitations of this format of control are discussed, along with the possible benefits and requirements of this proactive architecture.

Thesis Supervisor: Joseph A. Paradiso
Title: Associate Professor, Program in Media Arts and Sciences

Personalized Building Comfort Control

by

Mark Christopher Feldmeier

The following people served as readers for this thesis:

[illegible]

Thesis Reader____

William J. Mitchell
Professor
Program in Media Arts and Sciences

[illegible]

Thesis Reader____

Samuel Madden
Associate Professor
Department of Electrical Engineering and Computer Science

Acknowledgments

This thesis could not have been completed without the help and support of many people and organizations, and for each of these i am deeply grateful. The work was partially funded by an anonymous donor through the MITEI, and through a fellowship from the Martin Family Society of Fellows for Sustainability. M.I.T. Facilities, including Greg Tucker, Kevin Davis, and Alan Legere, were amazingly helpful in allowing the use of their building's HVAC system, and educating me in the details of its operation. Endless appreciation goes to Joe Paradiso for creating an incredible research environment, and whose belief and encouragement brought this project into reality. To Bill Mitchell and Sam Madden, my gracious readers, for their help and willingness to operate under deadline. To the entire Responsive Environments Group, whose complicity in acting as test subjects for my experiments allowed this work to be evaluated. Special thanks are due to Mat Laibowitz, Bo Morgan, Josh Lifton, Ari Benbasat, and Matt Malinowski, who lent their expertise at moments when it was most necessary. To Josh Lifton, Zoz, Tinsley Galyean, and James Patten, your providing, and sometimes withholding, of programming expertise, finally forced me to overcome my fears of abstraction. The knowledge offered to me by Rosalind Picard and Vikash Mansinghka made the pattern classification aspects of this work possible. Many thanks to the whole Galyean family and Nearlife, for allowing me to use their home as a testbed, and always making me feel at home. And, to the residents and alumni of EAsT camPUS and Seniour House, this work was completed because of you, and in spite of you, and for both i am thankful.

hope alls well with you

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Our urban landscape is dotted with thousands of buildings which remain fixed in our lives like the features of a mountainside. This long time scale of architectural evolution can create a comforting and stable environment in a world where the rest of our lives seem to be changing faster than we can adjust. Perhaps there are areas where a more flexible and responsive physical environment could aid our lives, and even protect us from the rapid change of the outside world. This reactive, or proactive, architecture could tend to its own needs in a more efficient and less obtrusive fashion than currently available. It could also meet the needs of its occupants in a way which is far more personalized than previously possible.

This problem of a static architectural landscape is particularly noticeable in the urban workplace. In our homes we have a certain level of freedom to modify and control our surroundings, allowing them to morph to fit our needs. By contrast, commercial spaces have locks on light switches and thermostats to keep the environmental control systems from being tampered with. Granted, this level of Draconian building control is required in most cases to keep the systems dynamically stable and improve efficiency, but this is not to be taken as the only method available to achieve these goals. Considering the eight hour work day, resulting in some one third of our lives spent in these spaces, the individual should have more control over the variables determining his or her comfort.

Not just the occupants, but also the commercial buildings themselves have ever-changing needs, for which a single setpoint is inadequate. With the average job turnover rate for American adults being a change once every six years [1], the type of work, and who is doing it, will vary with time for a given space. The arrangement of office furniture, and even the placement of walls and windows changes with time. And this is in addition to the usual seasonal waxing and waning of needs. The holiday rush in a department store is not the same environment as it is on some Tuesday evening in March. These changing needs are also seen in the domestic environment, where summer vacations can leave a home unoccupied for weeks on end.

To achieve this state of proactive architecture, buildings must become aware of their own needs, and the needs of their occupants. Fortunately, this task is becoming ever more attainable through the increasing availability of low-cost and low-power sensing. Wireless monitors of many kinds [2, 3, 4, 5, 6, 7, 8] can be purchased for home and office use, collecting continual streams of data for the home owner or building maintenance professional to use. These dense sensor networks are capable of creating their own ad-hoc communication connections and can be battery powered for months at a time.

The main issue with deploying these dense sensor networks is not a technological barrier, as the wireless communication, sensing, and actuation technologies have all been in existence for decades. Instead, the programming of such a large network of nodes becomes the complicating issue. The user of the building should not have to make all the direct mappings between inputs and outputs in the system. Furthermore, the user may not be capable of applying the fine level of control that is required to realize added efficiency bene rom the system. Therefore, an agile control scheme is required to handle the complex and constantly changing inputs and outputs.

This dissertation presents one solution to this mapping problem, and demonstrates the improvements that can be achieved through the application of dense sensor networks to architectural environments. Specific sensor hardware advancements will be shown, which make the task of installing and operating these systems easier. Adaptive pattern recognition and control algorithms will be presented, along with their efficacy in increasing the personal comfort of building occupants, and reducing building operational costs. By no means complete or optimal, this work gives insight into what can be realized through a proactive architecture.

1.2 Specific Instantiation

There are many aspects of human comfort that a building provides, but first and foremost is protection from the elements. Creating an appropriate indoor climate is essential to worker productivity [9] and personal happiness. It is also an area of large expenditure for building owners [10]. Over the past four years, energy consumption has become an increasingly important topic at both the national and international level. Within the United States alone, the average petroleum price has risen over 200 percent in the past four years [11]. This is a marked increase over the previous 21 year average, which had a peak deviation of 39 percent [11]. With these rising fuel costs, finding ways of reducing energy consumption is more important than ever.

The largest consumer of energy in the United States is buildings. Residential stock accounts for 21 percent of the total energy usage, and commercial stock accounts for 18 percent, combining to 39 percent [12]. Within buildings themselves, the largest energy sinks are the basic support systems, the lighting and heating, ventilation and air-conditioning (HVAC) systems. In residential applications, HVAC accounts for 26.1 percent and lighting 8.8 percent of the total energy usage [13], whereas in commercial applications, HVAC accounts for 53.4 percent and lighting 13.5 percent [14]. This makes building support systems, especially in the case of commercial buildings, a prime target for energy savings.

But, what can be done to reduce costs in these areas? Either more efficient lighting and ventilation technologies can be developed, or the existing technologies can be used more efficiently. Considering the long life span of buildings, and the fact that most commercial buildings are more than 15 years old [15], the latter proposition seems more cost effective, as merely adding a new control and sensing layer would be far less expensive than replacing a whole ventilation system. This idea is promoted further by the notion that most buildings are currently being run inefficiently due to the non-adaptable nature of their control system, and that savings of 35 percent are possible [8]. For these reasons, a novel air-conditioning control system, focused around the occupant, is developed for this thesis as a prime example of this proactive architecture.

The system has its foundations in the learning building concept put forth by Michael Mozer in 1992 [16]. The requirements placed upon the user will be minimal, merely a press of a button to indicate the direction of discomfort (too hot or too cold). The system will track general location within the building, and accommodate the needs of the person at that location. This has the added benefit of informing the building of where people are absent, allowing the system to be shut off when they are not around, effectively creating a scenario much like that of the light bulb in the refrigerator: from the user's perspective, it's always on. A larger unifying control structure will be informed by preassigned knowledge, e.g. which dampers and windows affect which rooms. Eventually, this could be done by the system as well, as the system could choose a time when users are not present, and apply a local step change in damper position, and locate which sensors respond. The building could eventually build a model of itself.

1.3 Contributions

Although the individual components of personalized climate control have been well documented, very little work has been done to put all the pieces together in a real-world experiment. This thesis seeks to fill that gap, with the addition of novel, on-body, comfort sensing hardware, and a reduced set of comfort indices. To date, the measurement of thermal preference has been limited to either a complex set of sensors attempting to determine a Predicted Mean Vote (PMV) value, or to direct polling of the user. The former is far too cumbersome and expensive for practical application, and the latter places an undue burden on the user. To overcome these limitations, an extremely low power, light weight, wireless sensor is developed which can measure temperature, humidity, activity and light level directly on the user's body. These data are used to immediately infer user comfort level, and to control an HVAC system in an attempt to minimize both cost and thermal discomfort.

Despite the commercial availability of distributed sensor networks for HVAC retrofits [8], the majority of comfort research is carried out in climate controlled chambers, or individual HVAC research rooms. In contrast, this thesis presents experimental results from a building under continual usage, modified with a custom wireless sensor network. For four weeks, ten building occupants, in four offices and one common space, were thermally regulated via wrist-worn sensors controlling the local air-conditioning dampers and window operating motors. Comparisons are made to the previous four week period of standard air-conditioning control, showing an increase in comfort, while decreasing energy usage at the same time. The difficult problems of control adaptation, comfort determination, and user conflict resolution are addressed. Finally, the limitations of this format of control are discussed, along with the possible benefits and requirements of this proactive architecture.

Chapter 2

Background

This thesis presents a complicated system, which takes its inspiration from many different aspects of the research field of comfort control. This system must necessarily be able to apply distributed environmental sensing, actuation, and control. It must also understand the factors involved in human thermal comfort, and successfully apply pattern recognition algorithms to detect an occupant's current state. Finally, the entire system needs to be adaptive with time to maintain high efficacy under changing circumstances. To cover these broad topics, a historical perspective on the current state of HVAC control is first given. Next, advances in HVAC control theory are shown to present the range of solutions to the multiple-input and multiple-output (MIMO) control problem that distributed sensor/actuator networks entail. This is followed by a discussion of personal comfort determination and actuation methodologies. Finally, previous attempts at the difficult task of creating a learning environment are shown as an influence for the work herein.

2.1 Historical Perspective

Distributed control of HVAC systems for greater efficiency and comfort is by no means a new topic. Although today we have more robust and accurate sensors, actuators, data communication lines, and computers, the main control schemes in use have not changed significantly since the early 1900s. A central unit still produces energy and distributes it to the rest of the building where it is locally controlled in a closed loop fashion. There are different ways of accomplishing this task (schedule control, fixed volume control, and VAV control), and even some that incorporate functions to minimize energy (Air-bypass control, reset control, setback control, economizer control, and CO_2 control), but the majority still don't take the entire plant state into consideration when making energy production and distribution schedules. This is further hampered by the lack of down-stream damper control to effect the fine grained level of distribution scheduling required to make this happen.

The main reasons for this lack of efficiency in HVAC systems is not technological, as the control theory has existed since the 1960s to deal with MIMO systems, and the sensing and actuation hardware has been around for much longer. The limitations are a function of economics. The cost of computation did not fall to a reasonable level until the mid 1980s, at which point an economic recession was beginning. No one was willing to invest in developing a new technology, especially when the demand for new architecture was dropping [17]. It wasn't until the mid 1990s, with the financial resurgence of the computer and telecommunication industries, that building investment would increase and the networking infrastructure would be inexpensive enough to implement such systems. Indeed, a number of research projects came out at this time showing how new control schemes could reduce energy costs and maximize human thermal comfort [18, 19, 20, 21, 22, 23]. These projects detail the implementation of a truly distributed control.

Until recently, however, the implementation of these systems still remained an academic pursuit. New building owners are extremely apprehensive about installing these sorts of infrastructure. Because of the long life expectancy of a building, it is very risky to invest in untested ideas which may not produce the energy savings claimed, or worse yet, may not be robust enough to last the decades required and instead need replacing in the near future. The cost of a new technology is generally higher to begin with, and with the installation cost of HVAC systems already being 16 percent of a building's total cost and 40 percent of its recurring cost [24], the decision to implement a new system is difficult to make. This is further compounded by the fact that building operating costs are far outweighed by personnel costs for commercial buildings, usually by a factor of 1:100 [10], so implementing something that might reduce worker efficiency by as little as one percent would already discount any energy savings gained. For this reason, we still saw papers in 2002 that describe internet-controlled HVAC systems as novel [25].

Two recent developments have the potential to end this trend of energy efficient technologies being overlooked for initial investment avoidance. The first is the increase in fuel prices, which has driven the recurring costs of energy inefficient systems higher (as energy represents 40 percent of all recurring costs of a building [10]). The second is the development of distributed sensor networks, which are just starting to leave the laboratory [8]. Their promise of implementing sensing at a reduced cost over current systems will eliminate the initial investment fears of building owners. They will also allow for the retrofitting of old buildings, which represent the largest percentage of building stock. Distributed sensor networks accomplish this by using extremely low cost wireless transmitters, which can establish mesh networks, rather than requiring wires to be run. The running of wires has been quoted as costing anywhere from ten percent to 90 percent of total HVAC installations [26]. With the dense sensing provided by these networks, more efficient control schemes can be implemented, and as result, there is renewed research in this area [27, 28, 29, 30, 31, 32].

2.2 Distributed Sensing and Control of Building Systems

It seems that almost every paper on either distributed sensor networks or distributed control theory lists HVAC systems as one of their main application areas. Gerard Guarracino et al. give an excellent overview of the work in these fields [33]. In order to reduce the scope of the background, references will mostly be made to research where an actual HVAC system was controlled with the algorithms described. This will provide grounding for the results discussed, and a better comparison point for the system which was built and tested in this body of work. Not surprisingly, the amount of real world experience is limited, mostly due to the cost and extremely long validation time required to thoroughly test a system (a full year is preferred to assess both heating and cooling).

Initial efforts into the MIMO control problem utilizing distributed control via sensor networks have been focused on fuzzy logic controllers [19, 20, 29]. Alcala shows very good results for a control scenario of a single room with two vents, a CO_2 sensor, and multiple temperature sensors, where energy savings of up to 30 percent are cited [29]. This is aided by the fact that not just the dampers, but also the energy production device is under control of the system. Despite the large savings, fuzzy logic controllers require pre-programming with expert knowledge of the system, and are not very tolerant of changes in system parameters with time, a critical component to system longevity.

These limitations of fuzzy logic control are addressed by genetic algorithms and neural networks, which evolve with time to limit the amount of pre-information required, and make the system robust to the ever-present devolution of the physical components. Kajl et al. show energy savings of 16 percent with the use of a genetic algorithm based controller for an air handling unit servicing 70 offices [30]. The system leveraged the pre-existing sensors and actuators, and did not seek to retrofit the building. Similarly, Curtiss et al. have shown power savings of 15 percent [23] with the addition of neural networks in the control loop. The results are from an ideal laboratory test environment with no human occupancy, and do not use wireless distributed sensing. In comparison to these rather complicated adaptive techniques, auto-tuning of a control system is a simpler way of achieving these gains, and can be applied to a variety of controllers. An example of this is demonstrated by Wang et al. [34], but unfortunately only time response performance values are given, and no efficiency data were collected.

Current research trends have focused on hybrid controls [32] and multi-agent controls [35]. In these systems, the control problem is broken down into small sections that are handled on a somewhat distributed basis. With hybrid controls, there are multiple levels of control in a hierarchical structure, with information flowing in both directions, but with each level maintaining its local status given current information. In the multi-agent systems, the individual nodes bid on shared resources to maximize global efficiency while minimizing local costs. Results for these systems seem to still be in the simulation phase for the area of HVAC controls, although they are being applied elsewhere.

One of the few examples of an HVAC system being controlled by an actual distributed sensor network is shown by Kintner-Meyer [28] for two office buildings. The first building was retrofitted with 30 battery powered, wireless temperature sensors. The sensor update rate was once every ten minutes, which is not fast enough for use in a continuous control loop, but did allow the building maintenance staff a better view of the current state of the system. This new view led to the identification of key problems in the HVAC system, and once these were addressed, reduced the total energy consumption by seven percent. The second building used 120 of these temperature sensors and allowed the use of set-back control, which was previously unavailable due to concerns over extreme temperature excursions. These savings ended up being approximately 6,000 $US annually for a 150,000 square foot building (estimated equivalent of four percent).

This trend of using sensor data to give insight into a plant's operational state is quite effective at solving the inefficiency problems. Most buildings undergo a balancing procedure when they are first commissioned. This entails setting dampers and flow controls within specified limits to maximize the effectiveness of the HVAC system. Unfortunately, this is usually the last time this balancing occurs. Slow degradation of the mechanisms, and dramatic shifts in usage of the building, lead to the initial balancing being completely inappropriate for its current situation, causing gross inefficiencies.

To combat this, the concept of 'continuous commissioning' was developed [36]. Companies such as Infometrics [37] constantly gather data from networked sensors (usually part of the original HVAC system) and analyze these data to look for anomalies or areas of improvement. A large portion of the process is automated, in a sense implementing the self aware building concept, but at a time lag, and with human involvement required to alter the various setpoints.

2.3 Personal Comfort Methodologies

Ultimately, the majority of the HVAC control work is focused on energy savings and temperature regulation, not human comfort. Although the control algorithms and adaptive strategies are directly applicable, the determination of personal comfort is not a solved problem. Multiple factors have been studied in their relationship to comfort, with the PMV [38] being the most common metric. The PMV attempts to average over large populations for the following factors: temperature, humidity, wind speed, thermal radiation, activity, and clothing. This is shown to work well in practice [39], but being a statistical quantity, does not fit all needs. A variety of other factors have been shown to influence comfort, including age [40], local climate and culture, and the availability of natural ventilation [41]. This led to the creation of the adaptive PMV, which begins to account for these variations as subsets given a certain operating point.

The major use of the PMV is to set boundaries on temperature, humidity, and wind speed to a comfortable level within a building. Distributed sensor networks have been employed in an attempt to assess comfort by measuring PMV values in real-time. Ivanov et al. developed a wearable sensor [42] for measuring PMV which contained a temperature sensor, humidity sensor, CO_2 sensor, and wireless transmitter. Calculated PMV values from the sensor are shown for a three day period, but no air-conditioning control is shown, nor is the data correlated to the user's actual comfort level. Tse and Chan report a set of six wired PMV sensors [43] which were deployed in an office setting and used to gather data for a simulation of PMV based control, but no actual closed loop tests were run. Federspiel and Asada demonstrated a full control loop operating on a single room with a single occupant based on PMV values [44], but these were directly polled from their user at regular intervals, creating a cumbersome interaction. This all points to a much more personalized view of comfort that is about local variables and a given person's perspective, not a fixed temperature point on a wall.

These previous works, with the exception of Federspiel, attempt to assess the PMV as a global variable: a fixed standard for all people. Megri et al. take a different approach [45]. Using PMV sensors similar to Chan's, they poll the user as Federspiel does, but instead use a support vector machine algorithm to determine the indices of the occupant's comfort, rather than looking up the comfort index from a PMV table. They show 99 percent accuracy at predicting comfort in this manner, which points to the possibilities of automatic recognition of comfort, on a person-by-person basis. The major downside of this work is that it involved large, tethered sensors which are required in three locations in close proximity to the user. This sort of infrastructure is prohibitively expensive and bulky for real-world deployment.

Previous work by the author (see Appendix A) shows that it is possible to predict comfort from a relatively smaller set of parameters in comparison to those in the PMV, without the need to correlate to a fixed index of any kind. The system is also lightweight and low power, making it a good candidate for office use. The system learns the meaning of various temperature and humidity points via reinforcement from the user, similar to Megri. The main advantage is the wearable nature of the sensors, obtaining direct skin temperature and humidity, wherever the person is located. 74 percent predictability is shown with a k-nearest neighbor (KNN) algorithm for a single sensor, and 82.5 percent predictability with two sensors, with marginal gains for increasing numbers of sensors.

This form of on-body sensing has a long history in comfort research, often referred to as Standard Effective Temperature (SET) [46] or the 'rational indices', as it attempts to take a physiological approach to understanding the conditions of comfort. Large environmental chambers are used to set particular conditions, and skin temperature and humidity readings are taken at multiple points on the body. Rate of moisture loss from the body can be measured by placing the subject on a scale. Radiative absorption and wind speed are also measured with local photometers and anemometers. Unfortunately, the methodologies entailed in deriving these indices are extremely cumbersome, and do not lend themselves to real time usage.

However, the problem is not only one of assessing an individual's personal comfort level. An effective control system must also be able to locate the person and effect that proximate temperature. As temperatures can vary greatly, even within the same room, this proves a difficult task. On-body sensing solves the first portion of this task, and a number of solutions have been derived to solve the second portion. These present some of the most commercially available solutions to personal comfort.

Forced air distribution systems, particularly underfloor methods, can be tapped at points along the run to allow air to circulate locally. Many companies make systems which implement this, usually under the name Task Ambient Conditioning (TAC). Johnson Controls made a particularly extravagant model called the Personal Environmental Module which also included acoustic and lighting controls [47]. These systems allow the user to adjust the air flow, and sometimes temperature, at a local vent. Not only is the availability of fresh air shown to give greater comfort, it is also more efficient [48] as air is only chilled where needed, and larger sections of the building can be allowed to drift out of normal comfort zones.

TAC systems, although efficient and effective, are rather expensive and difficult to install after initial construction. A commercial solution for the residential market deals with some of these issues by placing small inflatable bladders in local ductwork for localized control [7]. This relatively easy to install system is supplemented with wireless temperature sensors that act as thermostats for the room they are set in. They can be moved around the room to regulate where needed, but still are fixed per room and not small enough to be considered wearable. The programming interface is quite cumbersome, involving a series of menus for inputting temperatures and setback times for various rooms. However, it does herald a new era of retrofittable personalized building comfort control.

This concept of localized control brings up an even more important factor, the psychological parameters of comfort. It is found that given certain environmental conditions, a person will be more comfortable if he or she has set those conditions [49, 50]. Things such as operable windows and settable thermostats will increase comfort, even if they do little to change temperatures. This is both a positive and negative realization for the majority of personal comfort research, as it becomes difficult to distinguish actual comfort gains from placebo gains.

2.4 Building Behavioural Models

In the not too distant future, when dense distributed sensing and actuation has finally penetrated the market, a far more difficult question will need to be answered: how does the building become aware of itself and its occupants in a meaningful manner? The programming of these large systems will be crucial to their success, as each failure will be keenly perceived by their occupants. Learning from these failures will allow the buildings to better understand how to react to the conflicting needs of all involved. The following examples point to an answer to this question. They suggest that first and foremost, the system be robust, and provide basic functions regardless of state. They show that multiple forms of control can be leveraged to perform tasks particularly suited to their skills, each acting as an autonomous agent in a larger structure. And finally, they insist the system and occupant both be capable of learning from and adapting to one another.

Although the idea of a responsive architecture predates even "The Jetsons" [51], the work to create these environments began in earnest with the beginning of ubiquitous computing in the late 1980s [52]. At Xerox PARC, offices were equipped with radio-frequency identification (RFID) and light, temperature, and occupancy sensors which were allowed to turn off outlets, adjust HVAC systems, and control lighting [53]. Portable devices allowed users to edit preferences wherever they were in the building. The design philosophy was one of reliability, invisibility, and simple control layers on top of the pre-existing control, which would only react when needed. They could be completely removed and the system would still operate as normal. This philosophy is particularly relevant in an experimental setting, as long-term validation is not possible, and failures will occur often.

A current example of this sort of work tackles a very difficult question in shared office environments: whose comfort is being optimized? Lee et al. use temperature, humidity, light, and sound data to evaluate the comfort state of occupants [54]. In simulation, light levels and thermostat settings are modified to improve this comfort or save energy. The space is divided into public and private spaces, with users being tracked with RFID tags to localize their position. Comfort is registered with local lighting and temperature controls that the occupant can move, updating a data base which keeps track of preferences. This rule base changes in the public versus private spaces, with personal goals such as comfort weighted more heavily than others for the owner of a private space, and preference averaging in the public space. There are also privileges, so that occupants with higher social status will be have their comfort optimized over others.

These very programmatic responses are challenged by Mozer [16], who asserts that the building can learn from its occupants with only mild intrusion, essentially learning from the mistakes. His neural networked house would purposefully turn lights off in order to understand whether they were wanted on or not. He sensed the status of light switches, the thermostat, ambient indoor and outdoor light levels, ambient indoor and outdoor temperatures, human motion, door positions, and window positions. The system is also made aware of its energy consumption and current setpoints on all heating, water supply, and lighting appliances. In this manner, it can monitor the habits of the occupants and make predictive control decisions to minimize energy costs. Time of day and time of year are also taken into account, and passive solar gains are included in its model. Although this creates a longer learning curve than preassigned knowledge, it is capable of adapting over time without intentional user input. This also reduces the level of sensing required. The sensors are the environment, the normal inputs an occupant would use anyway.

This sort of adaption is evaluated by a number of researchers. Rutishauser [55] takes a unique approach of modeling the building as a multi-agent system, where each actuator is its own independent agent. These actuators use a fuzzy logic rule base that links them to various sensors. The rules are adaptable with time, with user input being seen as negative reinforcement: if the building is acting appropriately, no user action is required. The system incorporates automated blinds and lights, and it monitors light switches and blind switches, along with presence and light detectors. There are fixed rules at the lowest level which ensure that every light switch operates the light assigned to it, so even under extreme failure modes the basic comfort systems will still operate.

This is similar to the ideas put forth by Sterk [56], where the whole system is built up upon smaller pieces — an example of hybridized control. Smaller agents within the system deal with local problems while exchanging information and updating state through interactions with larger agents, which do the same from their interactions. This manages the complexity of the many sensor and actuator streams by breaking it down into repeatable tasks, yet allowing the individual repeatable tasks to modify based on new information. This also leaves the entire system capable of functioning if a small portion breaks. And perhaps most importantly, the local control can be simple enough for the occupant to understand what is occurring — to have a mental model of his or her environment.

This last point is critical, as it is not only important that the building understand what the user is doing, but also that the user understand what the building is doing. As Vastamäki et al. clearly describe in their analysis of thermostat usage [57], the fundamental efficiency of the building and the comfort of the occupants both suffer when the occupant does not understand the behaviour of the building. Users are shown to consistently turn thermostat dials in response to uncomfortable conditions, and to continue turning them until a comfortable state is reached. From the user perspective this seems completely reasonable: increase an action until the desired result is reached. From the building perspective, however, the time lag due to the thermal mass of a room is usually on the order of hours. Furthermore, most cooling systems are limited to excursions of one degree per hour according to the American Society of Heating, Air-conditioning, and Refrigeration Engineers (ASHRAE) standards. This means that over turning of the thermostat will drive the temperature too far in one direction, at which point it will be driven back again when the user is uncomfortable. This sort of oscillation wastes energy and creates a thermally unstable environment.

Vastamäki et al. propose a model of human behaviour with respect to wants, actions, and outcomes. The building must provide appropriate feedback, such that the user creates a mental model that shows the wants being eventually satisfied, so the user interprets the building's actions as signal and not noise. This sort of symbiotic relationship is also promoted by Kroner [58], who describes the two-way communication required for the creation of intelligent architecture. Peter Anders and Michael Phillips, with their ARCHOS project [59, 60], are testing these ideas with real hardware. Their system uses a relatively simple set of sensors (cameras, microphones, HVAC and lighting data, network data traffic, and elevator location) to infer the general state of the building/occupant unit. In their work, the line between the building and the person becomes blurred, and the line between the physical and virtual worlds becomes irrelevant. All of these collected data are streamed via the internet and FM radio to occupants and nonoccupants alike, altering their immediate sense of space. Although the majority of the applications are merely data sonifications or visualizations, the 'random lift button' gives a good example of a more tangible interface to the network. Just as it sounds, pressing the button within the elevator will take you to a random floor. Sometimes predictability and control are not what we want from our buildings.

Chapter 3

Hardware Design

The applications for distributed sensor networks are extremely varied, and they are also extremely demanding. Size, cost, and power consumption constraints constantly limit what can be done with individual sensor nodes. As a result, generic wireless sensing platforms have not left the research laboratories. Although they offer a standard platform by which various algorithms can be benchmarked, and usually allow for a wide range of applications, they often fail at doing any particular application effectively enough to allow for continued usage in a real-world deployment. This shortcoming of available sensor nodes necessitated the design of the system used herein.

3.1 System Overview

At the heart of this system is the building occupant. This is where the comfort information resides. To best assess the occupant's comfort level, a wearable sensor node was developed (herein referred to as the 'portable' node). This sensor node needs to be lightweight and small to remain almost unnoticed by its user. If it becomes too cumbersome to use, it will be left on the table and forgotten about. Ultimately, the user's comfort is being optimized. If the portable node is obtrusive, the user will not be comfortable even if the ambient temperature is perfect. Similarly, the portable node must also have a low power consumption, as constant battery recharges or large battery size will be a nuisance to the user.

This portable node senses the local temperature, humidity, light level, and inertial activity level of the user. It sends these data wirelessly, at one minute intervals, to the central network hub. The portable node also has three buttons on the side which allow the user to input current comfort state (one button each for hot, cold, and neutral). These buttons must be held down for at least two seconds to guarantee a successful transmission of data. This delay eliminates false transmissions due to jostling or bumping of the device.

These portable nodes are also given to the building itself, so it can be aware of its own temperature and not just that of the humans. The exterior of the building has two portable nodes (one east-facing, one west-facing) to gather local climate data. Each room has a different portable node, which only senses temperature and humidity, mounted near the conventional thermostat. In spaces where the thermostat could not be found, the portable node is mounted away from areas where humans would come in contact with it, but still in the general vicinity to gather proximate data.

The actuation of the various air sources (windows and air-conditioning dampers) is achieved via 'control' nodes. These nodes are tethered to a $24\,\mathrm{V_{DC}}$ power source, and have a motor which can open or close the associated mechanical element via wireless commands. They also monitor local wind speed, temperature, humidity, and light level. These data, along with the current state of the motor, are sent off wirelessly at one minute intervals to the central network hub. The motor can also be actuated locally with a set of pushbuttons, allowing the user direct control of the air source. The aim of the system is to enable the user to improve his or her climate, not confine the user to some externally controlled concept of comfort.

The backbone of the system is comprised of 'room' nodes. These nodes receive the data from the portable and control nodes, and act as network coordinators, ensuring that each proximate node is only talking to one device. Since each room has at least one room node, the locations of the portable nodes can be inferred from the received signal strength indicator (RSSI) of the RF device. They also assess the local temperature, humidity, light level, and activity level and send these data to the central network hub at thirty second intervals. Communication to the central network hub is accomplished via an on-board ethernet module. This ethernet module not only routes all sensor data back to the central network hub, but also transfers motor control commands from the central network hub to the room boards, where they are sent off wirelessly to the control nodes.

The sensor network is deployed on the third floor of the M.I.T. Media Laboratory building (E15). A map of the various sensor nodes and their locations can be seen in Figure 3.1.1. Four offices and one common space are outfitted, encompassing the workspace of 10 people. Two operable windows exist in this space, and they are equipped with control nodes and motorized openers. There are also seven variable-air-volume (VAV) dampers which have control nodes and motors.

This deployment provides a particularly challenging and relevant scenario. The usage of the space is mostly by graduate students who have ever-varying schedules, so an adaptive control scheme will be required, as compared to a typical office environment where 9AM to 5PM work is common. The mixed usage of space — offices versus common space, with frequent transitions between them — is a good test of the location abilities of the system. Finally, with multiple occupants per office, and multiple control points per office, the granularity of both temperature demands and temperature actuation allows for thorough evaluation of the system's agility at scheduling conflicting requirements.

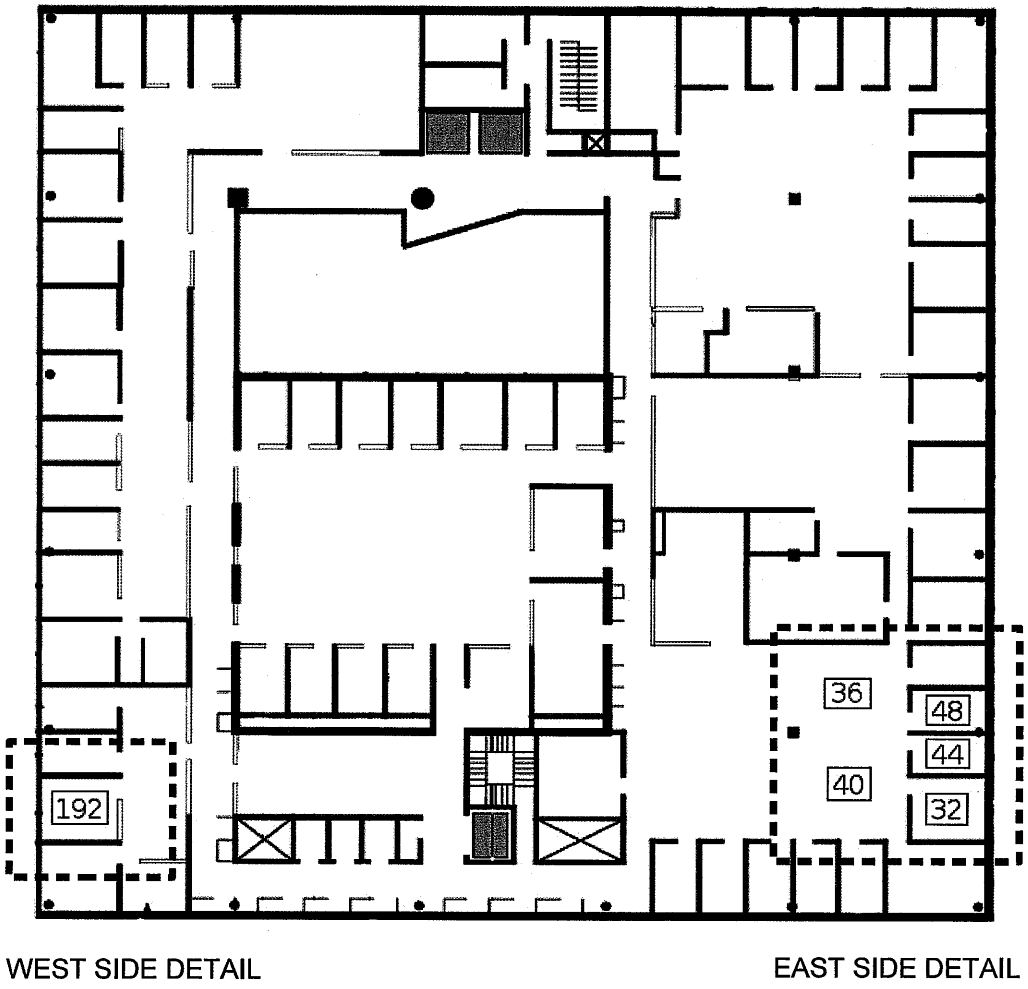


Figure 3.1.1: Floorplan of Sensor Deployment Areas

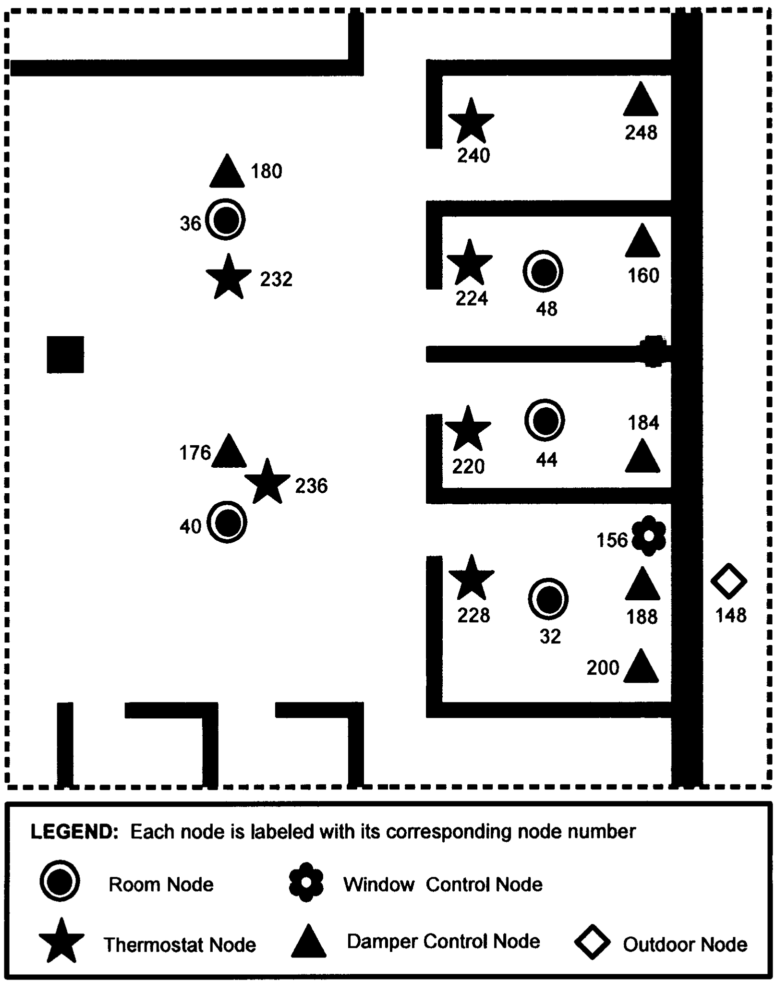


Figure 3.1.2: Detail of All Sensors Deployed on East Side of Building

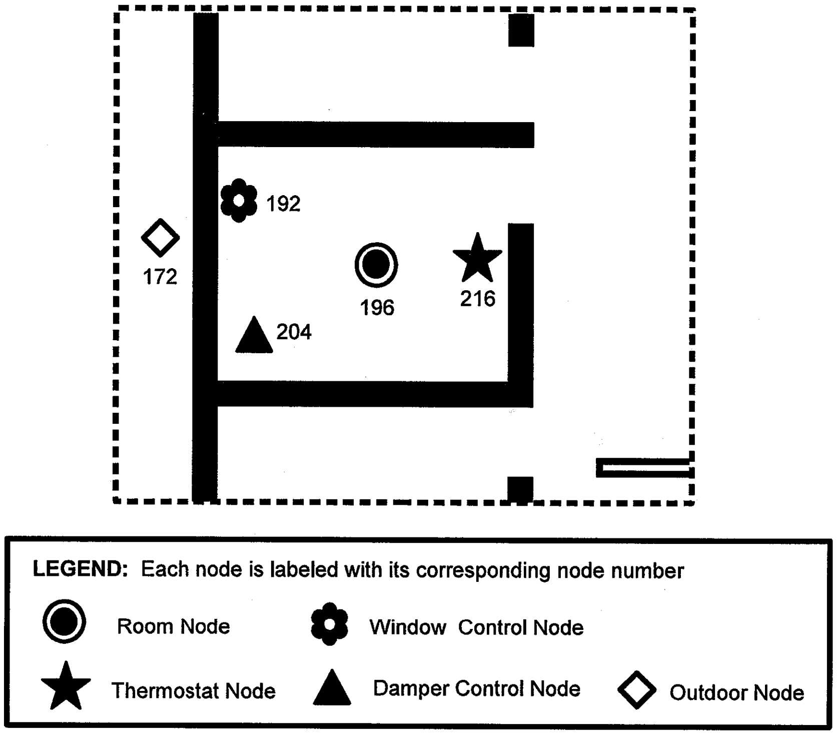


Figure 3.1.3: Detail of All Sensors Deployed on West Side of Building

3.2 Wireless Networking Protocol

The main topology for the network is a tree, with the central network hub acting as the trunk, room nodes acting as branches, and control and portable nodes acting as leaves. The full network topology is shown in Figure 3.2.1. All data is sent to the central network hub before being processed and distributed back to the relevant leaf nodes. This organization was chosen for its ease of implementation, as a stable network routing table could be established once for the duration of the work. It also reduces network overhead, as nodes do not spend much time establishing connections. A single RF channel is used, as collisions are low enough not to warrant the extra power demand on the portable nodes of keeping aware of the network's current channel. The RF unit, however, is capable of detecting open channels and switching frequencies.

The physical (PHY) layer of the wireless transceivers is based on the ZigBit module [61] by Meshnetics. It comes prepackaged with an Atmel ATmega1281 microcontroller [62], Atmel AT86RF230 2.4GHz radio [63], and dipole chip antenna. It automatically implements the critical aspects of the 802.15.4 medium access control (MAC) layer, including clear channel assessment, random exponential back-off, successful receipt acknowledge, and cyclic redundancy check (CRC) calculation and checking. Additionally, the radios do not respond to packets with a different personal area network (PAN) ID or destination ID (except in the case of broadcast packets which do not require receipt acknowledge).

Each node is hard-coded with an ID when deployed, so the locations of room nodes and users of portable nodes can be tracked. Upon wakeup, leaf nodes request a branch node to communicate with. The first branch node to respond becomes the destination ID for that leaf node until the leaf node is no longer in contact. At this point, the leaf node will retry the transmission process and request another branch node from the network upon failure. A leaf node can only communicate with its allocated branch node. In this manner, the low power leaf nodes do not spend an appreciable amount of time finding optimal connections, but rather stay joined to their primary branch node for as long as possible.

Each RF packet that is sent contains a preamble, the ID of the transmitting node, the ID

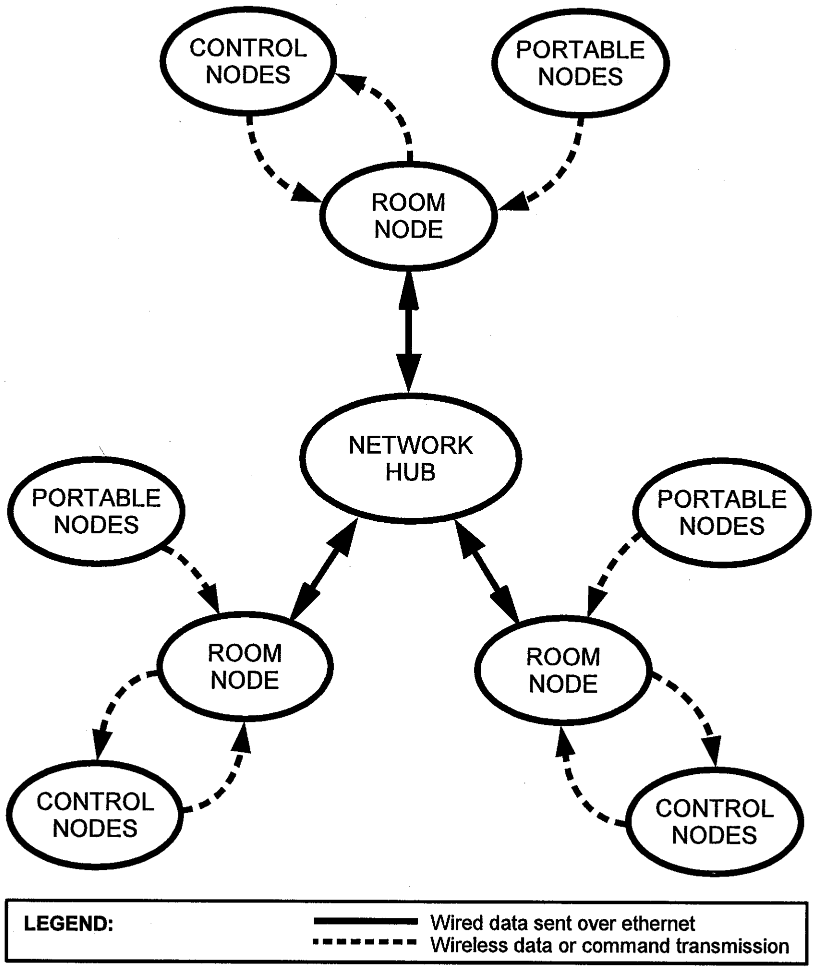


Figure 3.2.1: Network Topology

of the receiving node, the PAN ID, the packet type, the packet length, payload data, and a CRC. When a room node receives an RF packet, it strips the CRC and PAN ID, adds RSSI data, a header, and two footer bytes before sending it over Ethernet to the network hub. The network hub uses the header, packet length, and footer to determine valid packets, and uses the same format for transmissions to the room nodes. Currently, the only available commands the network hub can send over the network are motor control commands, and they are shown in Table 3.2.2.

Successful branch node acknowledgement packets are not sent over the network, but failed packets are sent over the network, so the system can be aware of whether a command was

Network Transmit Command	Packet [0x]
Open Motor Full	ff 06 08 DES 08 08 ff fe fd
Close Motor Full	ff 06 08 DES 08 06 ff fe fd
Open Motor n-Steps	ff 06 08 DES 08 08 n fe fd
Close Motor n-Steps	ff 06 08 DES 08 06 n fe fd
Stop Motor	ff 06 08 DES 06 0a ff fe fd

Network Receive Command	Packet [0x]
Data From Portable Node	ff 0d 02 SRC BTN HUM TEMP ACT1 LITE LQI RSSI DES fe fd
Data From Room Node	ff 07 03 LITE TEMP HUM ACT2 SRC fe fd
Data From Control Node	ff 0d 07 SRC MOTR HUM TEMP WIND LITE LQI RSSI DES fe fd
Join Request	ff 05 01 SRC LQI RSSI DES fe fd
Join Request Fail	ff 03 06 DES SRC fe fd
Location Beacon	ff 05 00 SRC LQI RSSI DES fe fd

SRC = Data Source Address (2 bytes, LSB first)
BTN = Button Press Information (1 byte)
HUM = Humidity Data (1 byte)
TEMP = Temperature Data (2 bytes, LSB first)
ACT1 = Inertial Activity Data (2 bytes, LSB first)
LITE = Light Level Data (2 bytes, LSB first)
LQI = Link Quality Indicator (1 byte)
RSSI = Received Signal Strength Indicator (1 byte)
DES = Data Destination Address (2 bytes, LSB first)
ACT2 = PIR Motion Data (1 byte)
MOTR = Motor State Information (1 byte)
WIND = Wind Speed Data (2 bytes, LSB first)

Figure 3.2.2: Network Transmit Packets

received or not. For example, if a motor command is sent, but the receiving node is down, that packet will get returned, and the system will be aware of the failure.

3.3 Portable Node

The portable nodes represent the main hardware contribution of this thesis. The requirements on these nodes are the most strict, subsequently leading to the development of novel sensing techniques. The main goals for these nodes are:

- Small in size and weight, so as to be unnoticed by the user (on the same order as other wearable devices such as wristwatches and key chain fobs).
- Long battery life to make the usage seamless (years if possible).
- Sense parameters relevant to human temperature comfort (temperature, humidity, light level, activity level).
- Transmit these data via a wireless link at a rate frequent enough to enable closed loop control with the data.
- Allow for user input of current comfort level.

The portable node internals are shown in Figure 3.3.1, and an assembled node is shown in Figure 3.3.2. It weighs 30 g, and measures 54 mm by 40 mm by 14 mm. The enclosure for the node is taken from another wearable electronic device, a keychain picture viewer, so its form is particularly suited to this application. The device comes with a clip, so it can be attached to clothing, keys, or merely placed in a pocket. They can be attached to lanyards and worn around the neck (as shown in Figure 3.3.3), or to wrist straps and worn like a watch (as shown in Figure 3.3.4).

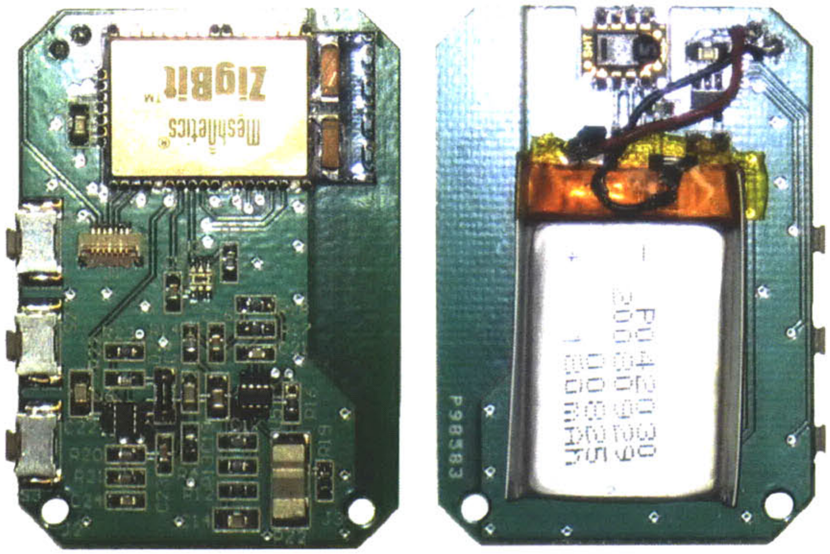


Figure 3.3.1: Portable Node Circuit Boards: Front and Back

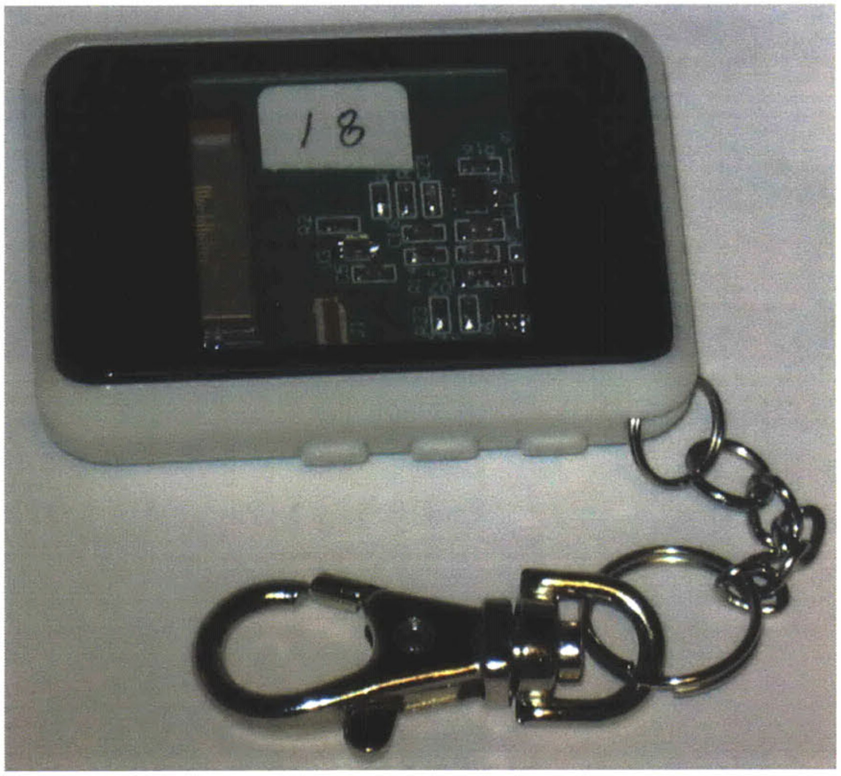


Figure 3.3.2: Portable Node on Keychain

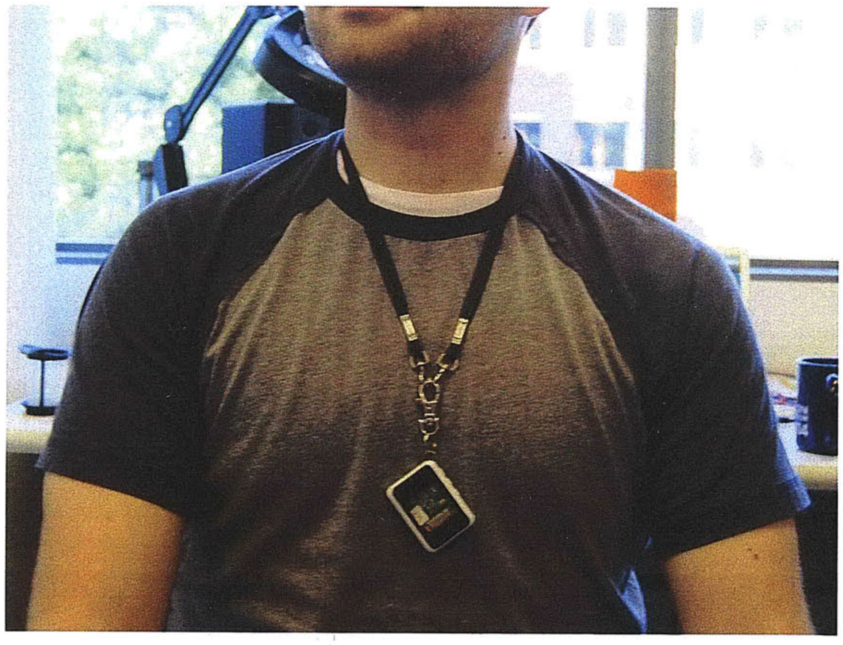


Figure 3.3.3: Portable Node Worn on Lanyard Around Neck

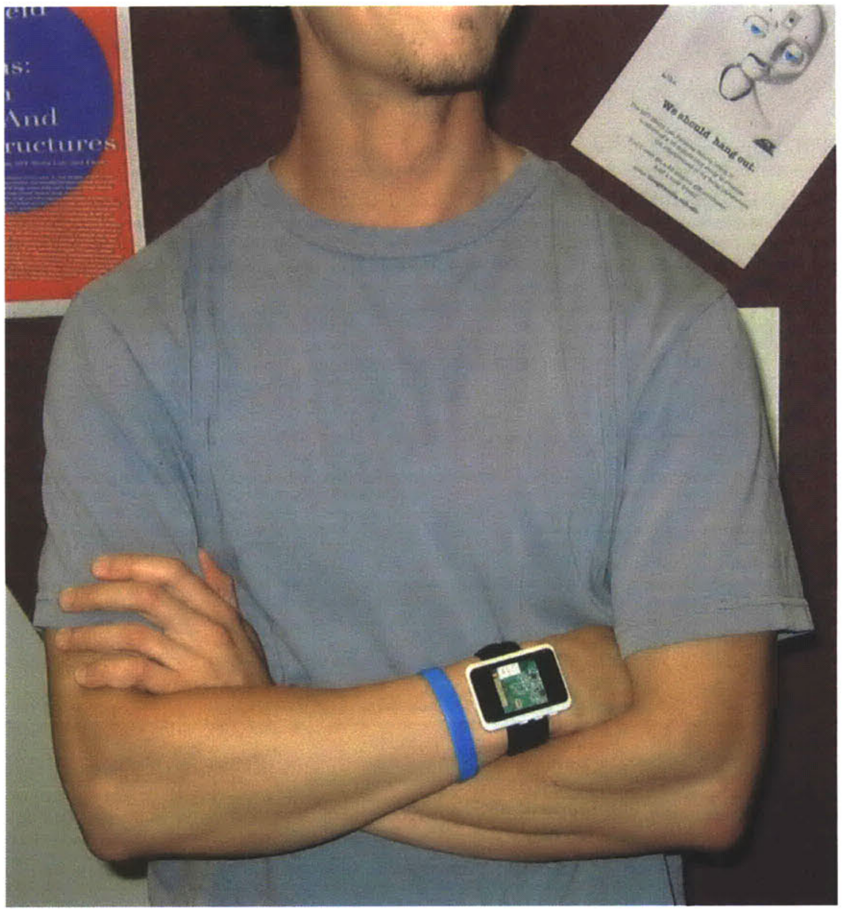


Figure 3.3.4: Portable Node Worn on Wrist

Portable nodes are also used to monitor room climate and outdoor environmental conditions. Indoors, they are mounted below the standard thermostat in the space, as shown in Figure 3.3.5. These nodes only have temperature and humidity sensing, as light and vibration information are not necessary. The outdoor nodes are modified to be protected from excessive moisture. The penetration in the case for the humidity sensor is covered with a long tube, and the remainder of the case is sealed with silicone, as can be seen in Figure 3.3.6. The top of the node is covered with a four stop, neutral density filter, to bring light levels down into the range of the light sensor. This has the unfortunate side effect of absorbing a large amount of thermal radiation, and heating up the unit, causing the temperature and humidity readings to no longer reflect the ambient conditions.

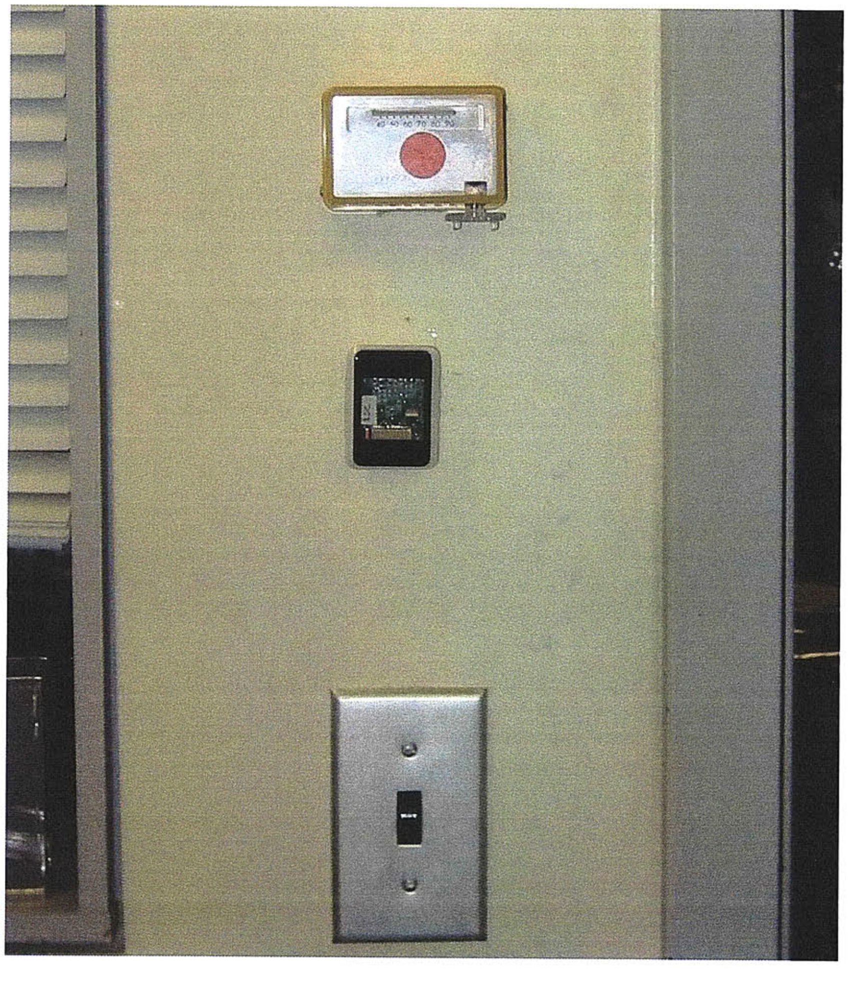


Figure 3.3.5: Portable Node Mounted Near Thermostat

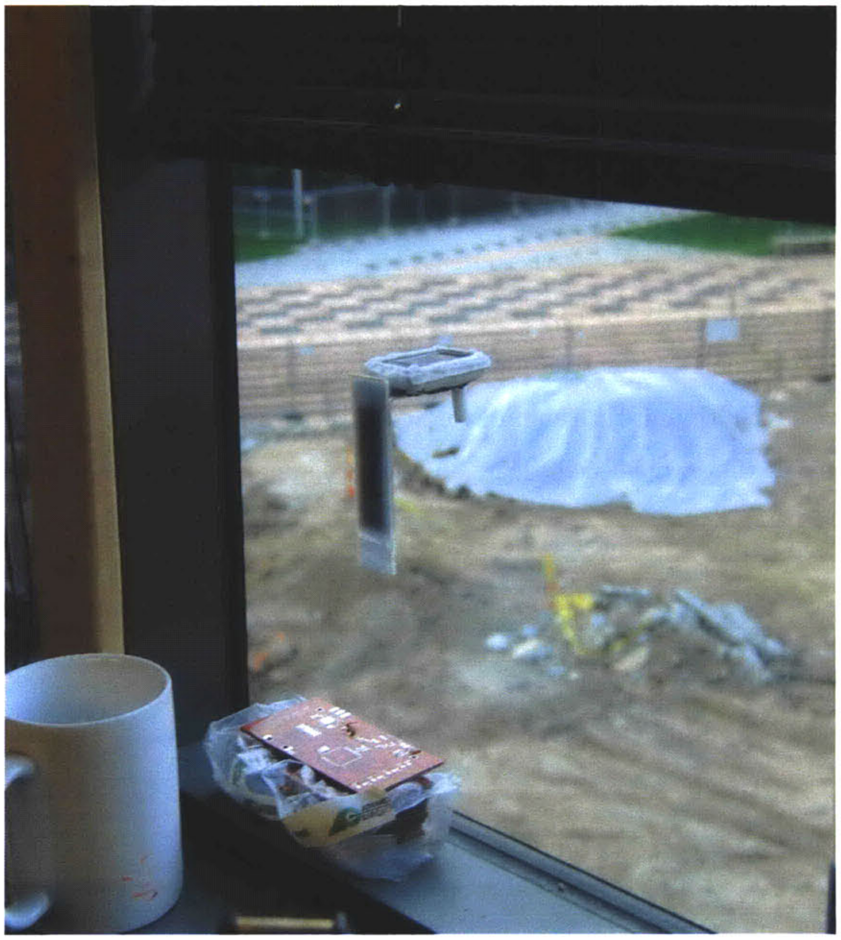


Figure 3.3.6: Portable Node Mounted on Exterior of Building

A data update rate of once per minute is chosen in order to minimize the RF unit power consumption time, but still allow for a responsive control system. Since the thermal and humidity time constants of the sensors are on the order of minutes, finer resolution data would not lead to substantial improvements. As will be shown in Table 3.3.1, the sensor sampling procedures and RF transmissions account for 20 percent of the total power consumption, so slight increases in sampling frequency could be attained without exorbitant sacrifices in battery life.

The battery for the device is a lightweight, rechargeable, 150 mAh, lithium polymer battery. This is the battery that came with the keychain picture viewers, and it has battery protection circuitry to ensure that the users can not be hurt by battery shorts. Although the battery is rechargeable, no recharging circuitry is placed on-board, as the expected battery life is two years.

The extended battery life of the portable node is achieved via extremely low power sensing elements, and minimization of processor and RF unit on-times. A full breakdown of these can be seen in Table 3.3.1. All of these items are powered by a 2.7 V low quiescent current voltage regulator, the TI TPS780270200 [64]. At 500 nA, it places minimal load on the battery, but has poor transient response, with excursions up to 150 mV when high power sections such as the RF unit turn on. All voltage sensitive operations, such as ADC readings, are performed at times when such excursions do not happen. The digital operations remain unaffected.

Device	on-time [per 60 s]	current	total [μAs]
Real Time Clock	60 s	6 μA	360
Voltage Regulator	60 s	.5 μA	30
Activity Sensor	60 s	2.8 μA	168
ADC	12 ms	700 μA	8.4
Comparator	10 ms	700 μA	7
RF Unit	4 ms	20 mA	80
Light Sensor	1 s	5 μA	5
SHT15	55 ms	600 μA	36
TOTAL	**11.5 μA average**		

Table 3.3.1: Power Consumption Breakdown for Portable Node

The 9.3 μA current during sleep mode is dominated by the real-time-clock (RTC) module on the Atmel ATmega1281 microcontroller unit. This utilizes a 32kHz crystal to keep track of time and wake up the unit every minute. By using this built in module, the lowest sleep state available is 'power-save' mode, which has a base current draw of 2.5 μA at 2.7 V Vcc and 25 °C. The remaining 3.5 μA come from the power required to vibrate the crystal. External RTC modules can run under 1 μA [65], which would both reduce the crystal drive signal and allow for 'power-down' sleep mode, which only consumes 0.2 μA at 2.7 V Vcc and 25 °C. This would more than halve the sleep current consumption, and double the battery life. This particular direction was not chosen for ease of implementation, as the two year battery life already exceeded the needs of the system.

The remaining 2.2 μA are a result of computation, sensing, and RF transmissions. Since the microcontroller is being run at 1 MHz from its internal RC oscillator, the power consumption is fairly low (1 mA), and the processor sleeps for most of the sensing time, making it quite negligible in comparison to the 4 ms the RF unit is on for at 20 mA. The next biggest power consumer is the SHT15 [66] temperature and humidity unit, which requires 55 ms at 600 μA to accomplish its task. The ADC and comparator units operate at 700 μA for 12 ms and 10 ms, respectively. The light sensor is awake for a full second due to its long start up time, and its current consumption varies with light level. On average it draws only 5 μA, though, which is small in comparison to the other sources. This gives 133 μAs of current draw for each wakeup cycle. These cycles come once a minute, averaging to 2.2 μA.

The temperature, humidity, and light level senors are all off-the-shelf components. The Sensiron SHT15 temperature and humidity sensor is run in 8-bit humidity and 12-bit temperature mode to save power. Higher resolution would allow for tighter feedback control loops around the room temperature, but the device is not accurate much past 12 bits, and this still allows for less than 0.1 °F resolution. The light sensor is the ISL29102 [67] by Intersil, which has an integrated human eye response filter and an analog compressor at the output, allowing for its reading to be taken by the ADC unit on the microcontroller. Without this compression, the six orders of magnitude of light levels could not be represented by the 12 bits of the ADC.

The final sensing modality is especially designed for this application. Traditionally, inertial activity level is measured with a MEMS accelerometer, but these tend to draw approximately 180uA at their lowest [68]. Considering that the activity sensor needs to run continuously so as not to miss any relevant motion, this would increase the device's current consumption by an order of magnitude — without even accounting for the processor time necessary to sample an ADC or send values over a digital bus. Luckily, this application does not require the full frequency range of MEMS accelerometers. Specifically, the 0 Hz attitude information is not needed to determine if the user is moving around a little or a lot. Instead, a vibration integrator is developed around a Murata PKGS [69] passive piezo shock sensor. Usually employed in hard disk drive and air bag deployment sensors, these small devices are very sensitive in all directions (1:2:4 x:y:z sensitivity ratios), mitigating the need for multiple axis integration for some applications.

The piezo ceramic is biased with high value (200 MΩ) resistors to keep the low frequency response of the sensor. Since the capacitance of the sensor is approximately 450 pF, this gives a lower cutoff of 3.5 Hz. The remainder of the gain stages pass 0.1 Hz to 10 Hz, with the second stage being a second order low pass filter to eliminate 60 Hz pickup. An active peak detector and integrator follow to give an output which is dependent on the total activity over the past period. A reset function is implemented by the microcontroller to drain the charging capacitor between samples.

The op-amps used in this circuit are the OPA2369 [70] from TI, which run at 700 nA per channel, and have very low crossover distortion. Since the passive components draw negligible power, this gives approximately 2.8 μA for the entire circuit. The low crossover distortion is helpful in reducing errors in the active peak detector. An active peak detector is required due to the low power supply rail of 2.7 V, causing losses due to a diode drop to become too large a portion of the full range.

To further keep power down, the integration charging currents are kept under 100nA. This poses a difficult problem as reverse leakage in standard rectifiers is usually on this order or greater. To solve this issue, the low leakage FDLL300A [71] from Fairchild is used. At 50 pA of reverse current, it exhibits low leakage for its relatively low forward voltage drop of 0.3 V.

The activity sensor is extremely sensitive to slight movement. However, as it has a one minute integration time, these movements must be maintained for the output to show significant change. The output values for various activities can be seen in Table 3.3.2. Although these values are representative, there is variation amongst circuits, as small leakage paths discharge the integration capacitor differently. To help prevent this, each board is thoroughly cleaned with flux remover before commissioning. Other factors such as light level and temperature contribute to variations over time. The rectifier is light sensitive, having greater reverse leakage under higher light levels, and the charging capacitor and piezo element are temperature sensitive. Fortunately the temperature and lighting conditions remain fairly consistent in an indoor environment, and these are both measured on-board, and can therefore be compensated for in firmware. Future revisions could reduce the light effects by placing a metal shield over the circuit, eliminating electromagnetic interference as well.

sensor usage (ten sample average)	mean value	standard deviation
on desk with no one at desk	47.9	4.6
on desk with person working at desk	39.8	6.2
worn around neck by person sitting still	46.7	6.0
worn around neck by person sitting and working	191	121
worn around neck by person walking	1015	19.8

Table 3.3.2: Representative Performance of Activity Sensor

It is interesting to note that extremely low vibration levels exhibit lower readings than no vibration levels. It is not clear exactly why this happens, but is hypothesized that a low level of excitation keeps the diode in the active rectifier forward biased more often, reducing the number of transitions made by the nonlinear circuit as it tries to maintain stability. These transitions can transfer current via the capacitance across the diode, as they happen very quickly. Ultimately, the difference is within a standard deviation of the samples, making it difficult to distinguish between the cases in practice, although some of the nodes are sensitive enough to perform this task.

3.4 Control Node

Since the system is intended to allow for retrofitting of existing buildings, one requirement of the control boards is that they accommodate many different actuators. To accomplish this, the LMD18200 PWM motor controller [72] is used. It can handle up to 3 A of continuous current, and will operate 12 V to 60 V motors. As $24\,V_{DC}$ is a common power supply in the control industry, this is deemed adequate for the situation. $24\,V_{AC}$ supply is the most common in the control industry, but AC voltage does not allow for easy reversibility of motors, so DC is used. There is also an over-current comparator attached to the LMD18200, so the microcontroller can automatically shut off the unit if a motor has hit the end of its travel. An assembled control node is shown in Figure 3.4.1.

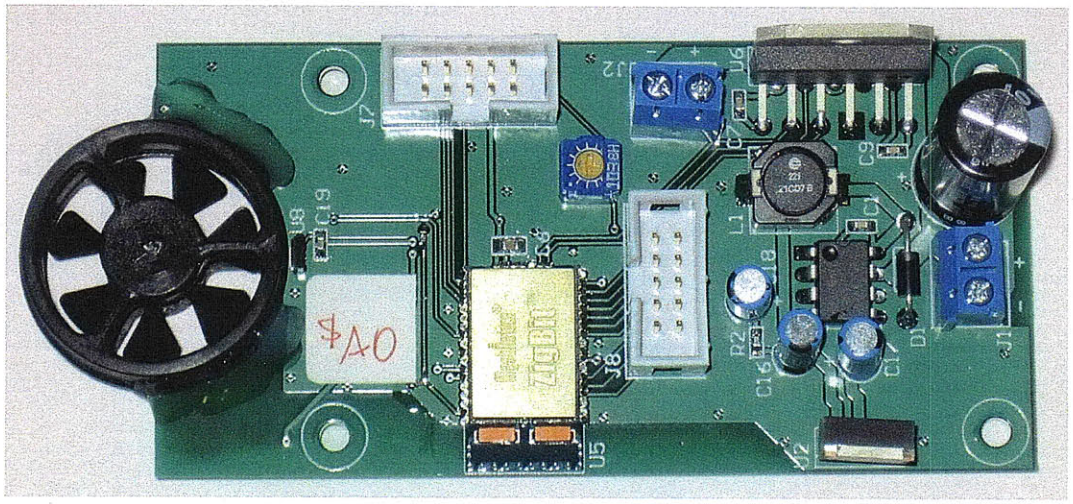


Figure 3.4.1: Control Node Circuit Board

The motors used depend upon the application. For operating the windows, a 24 V_{DC} motor from Wintrol [73] is used. It is designed for this application and comes with the appropriate mating connectors. The full window opening time is two minutes. The VAV box dampers are controlled with Belimo CM24-3-T [74] actuators. They have automatic over-current shutoff, and variable hard stops to set the end of travel. They also have a magnetic clutch, which allows them to be disengaged quite easily. This feature allows for quick switching between the normal control system and the experimental control system. The full damper opening time is 30 s. The mounting scheme for these motors can be seen in Figure 3.4.2 and Figure 3.4.3. The control node is mounted nearby in both scenarios to capture accurate wind speed, temperature, and humidity data.

The control nodes have the same light, temperature, and humidity sensing as the portable nodes. In comparison to the portable nodes, the control nodes are kept on continuously as they have a direct power source, eliminating the need to maintain any sort of battery life. The nodes also use the same ZigBit module for their communications and sensor sampling tasks. An external header is supplied which allows for a manual pushbutton control board

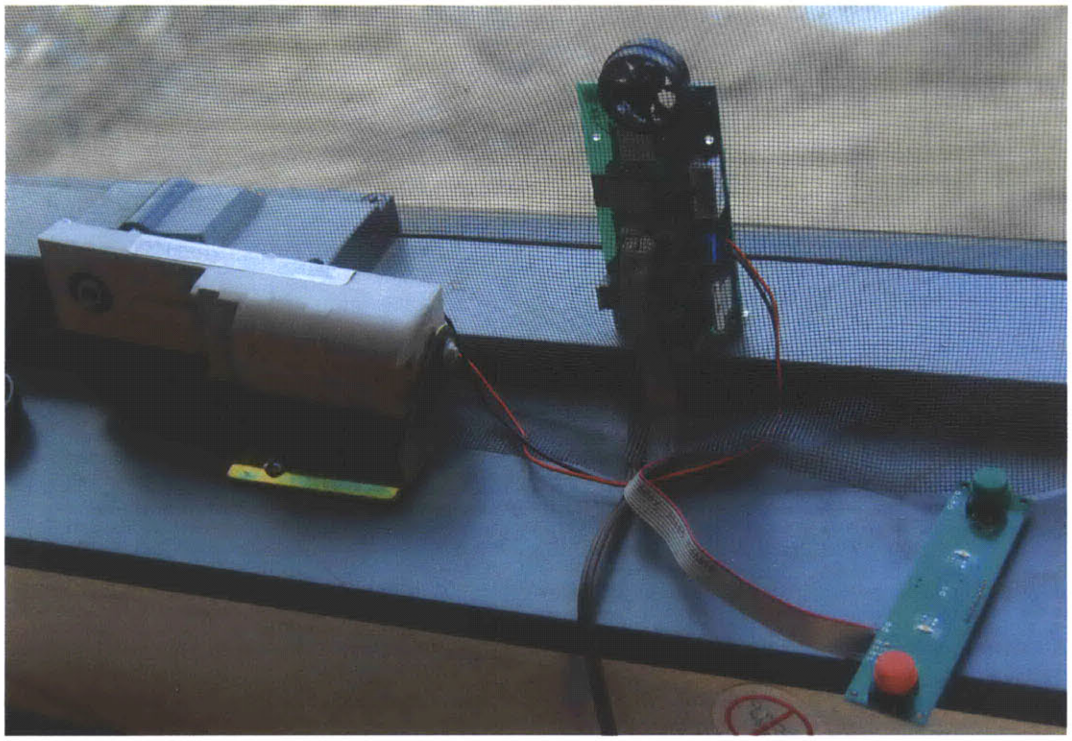


Figure 3.4.2: Control Board Mounted with Window Motor

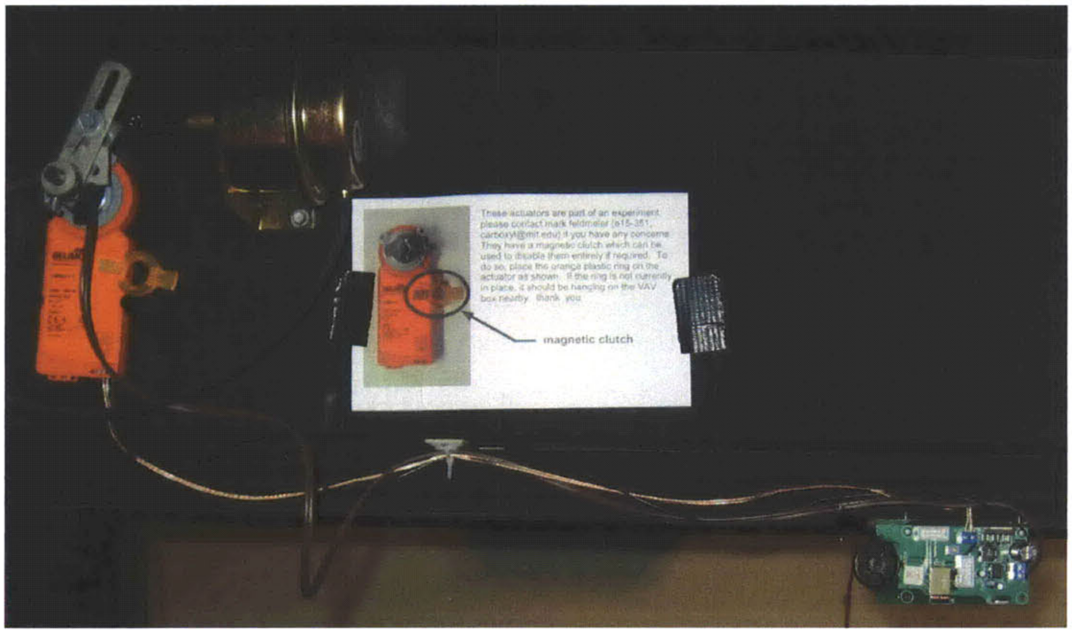

These actuators are part of an experiment please contact mark feldmeier (e15-351, carboxyl@mit.edu) if you have any concerns. They have a magnetic clutch which can be used to disable them entirely if required. To do so, place the orange plastic ring on the actuator as shown. If the ring is not currently in place, it should be hanging on the VAV box nearby. thank you.

Figure 3.4.3: Control Board Mounted with Damper Motor

to be attached, and also provides for future expansions of functionality.

The control node also has a wind speed sensor based upon a spinning impeller design. The impellers are replacement parts for the Kestrel 1000 wind speed meter [75]. They have a magnet on the shaft which is picked up by a hall effect sensor and sent through a filter and Schmitt trigger comparator to produce a square wave of the same frequency as the impeller's rotation. The impellers themselves are fairly linear, although they have a finite amount of friction, and will not sense below a certain level. These wind speed sensors are used to measure the chilled air flowing from the VAV boxes, and are the main method used to estimate energy consumption.

Initial calibration of the wind speed sensor is performed to determine the linearity of the design, and the results are shown in Figure 3.4.4. The sensor is compared to a hand-held meter of a similar design which claims to have an accuracy of $\pm 3\,\%$ of full scale and a range of $0.4\,\mathrm{m/s}$ to $30\,\mathrm{m/s}$. To determine the sensor's linearity in an actual application, a control node is mounted to a VAV box, and a large ductwork is created to collimate the air with limited resistance placed upon the air flow. The output of this ductwork is measured at a number of points with the hand held meter, and these values are averaged over the exposed surface area to give the total flow rate. The opening in the ductwork is made as large as possible while still giving a reliable reading on the hand held meter (usually around $0.4\,\mathrm{m/s}$). The test set-up can be seen in Figure 3.4.5, and the results are shown in Figure 3.4.6. In general, the constructed wind speed sensor does a good job of measuring the total air flow from the VAV boxes.

Ultimately, although the sensors themselves work well and have good linearity, they make a rattling noise at high wind speeds which is not tolerable to the building occupants. For this reason, a series of sheet metal shims are used to limit the air flow from the VAV boxes to a level where the sensor no longer rattles. This reduces the total range of the sensor, making the initial dead band problem worse as it now occupies a greater percentage of the range. Fortunately the VAV dampers spend the majority of their time in a high flow rate regime, so the data obtained from these sensors are still meaningful.

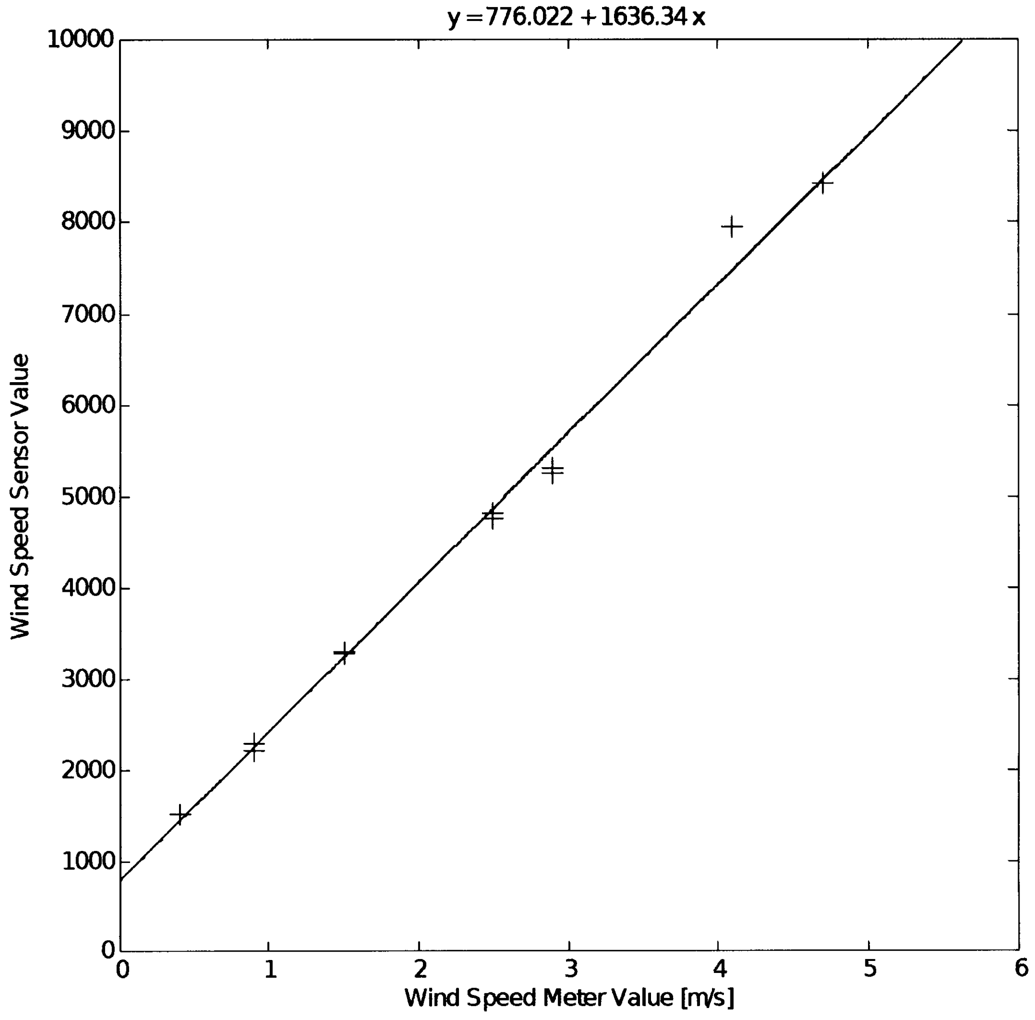


Figure 3.4.4: Initial Calibration of Wind Speed Sensor

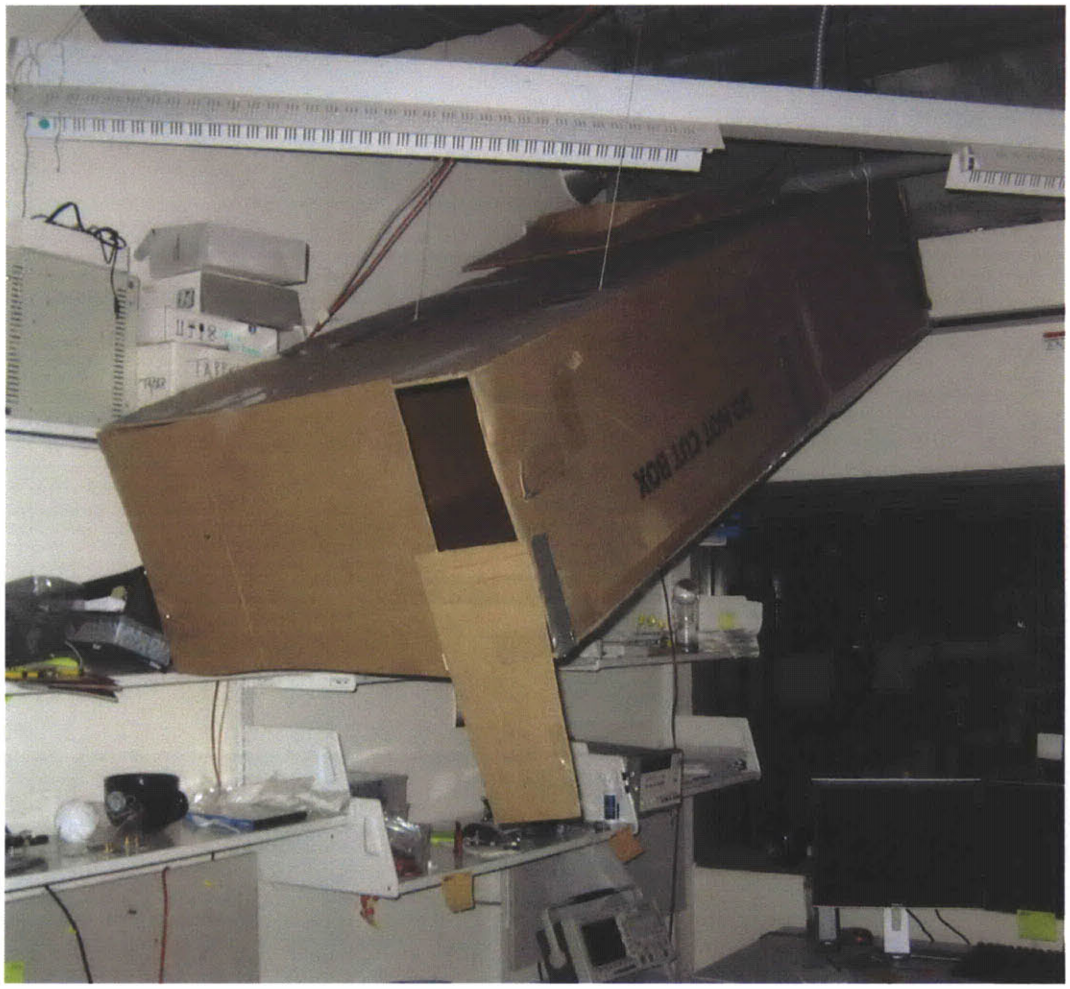


Figure 3.4.5: Collimator for In-situ Wind Speed Sensor Test

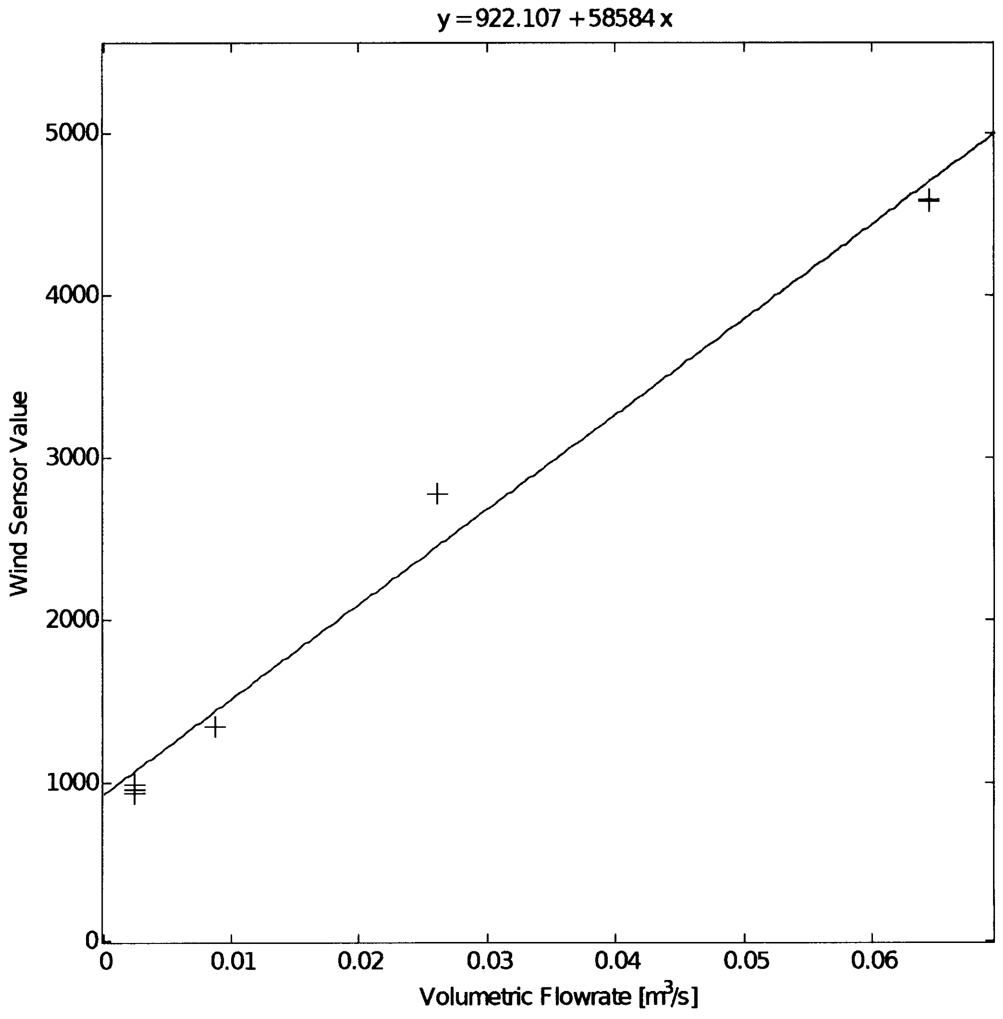


Figure 3.4.6: In-situ Calibration of Wind Speed Sensor

To compensate for this low flow rate regime, a second set of wind speed sensor calibration tests is performed. Each VAV damper is incremented by ten percent, through its entire range. At each level, the air flow is measured at multiple points with the hand held meter, and the wind speed sensor value is recorded. This gives a direct mapping of each sensor's associated flow rate, and the results can be seen in Figures 3.4.7 – 3.4.14. The majority of the sensors have a linear, or at least monotonically increasing, correlation between flow rate and wind speed sensor value. Unfortunately, two of the sensors show a decrease in sensor value for extremely high flow rates. This is hypothesized to be a result of the output scoop of the VAV box directing the air directly down, and away from the sensor, at high velocities. A example of these decreasing sensor values can be seen in Figure 3.4.15, with the horizontal axis showing the damper opening up by ten percent increments with time, and the vertical axis showing the varying wind sensor values. To accomodate for this problem, a non-linear mapping is used to extract flow rate data. As it is impossible to determine the exact flow rate above a certain threshold, the average flow rate of all values above this level is returned. Plots of the calibration lines used for all wind speed sensor mappings are shown along with the data.

This calibration process was repeated on three separate days, with test points A being the full damper incrementing process. Test points B represent samples from the previous day at three levels, lowest possible, mid-range, and highest possible. Test points C represent two samples from a month previous, one at the lowest level, one at the highest level. The lowest level recorded is shown with an asterisk on the charts, as this is the flow rate below which the friction in the wind vane sensor can not be overcome. Any flow rate below this level will be read as zero. The variations in maximum values are due to differences in the main system air flow, temperature, and humidity on the corresponding days.

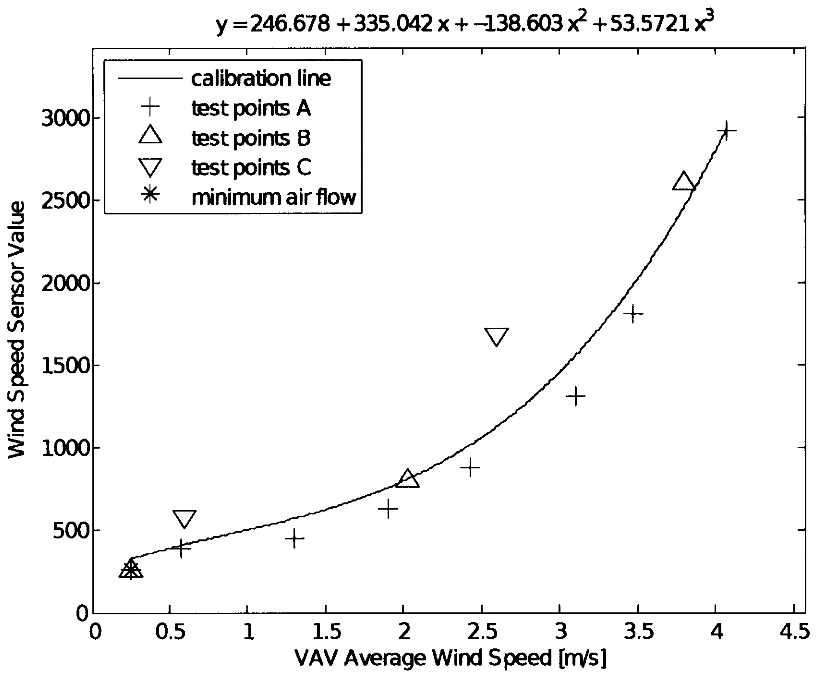


Figure 3.4.7: In-situ Calibration of Shimmed Wind Speed Sensor: Node 160

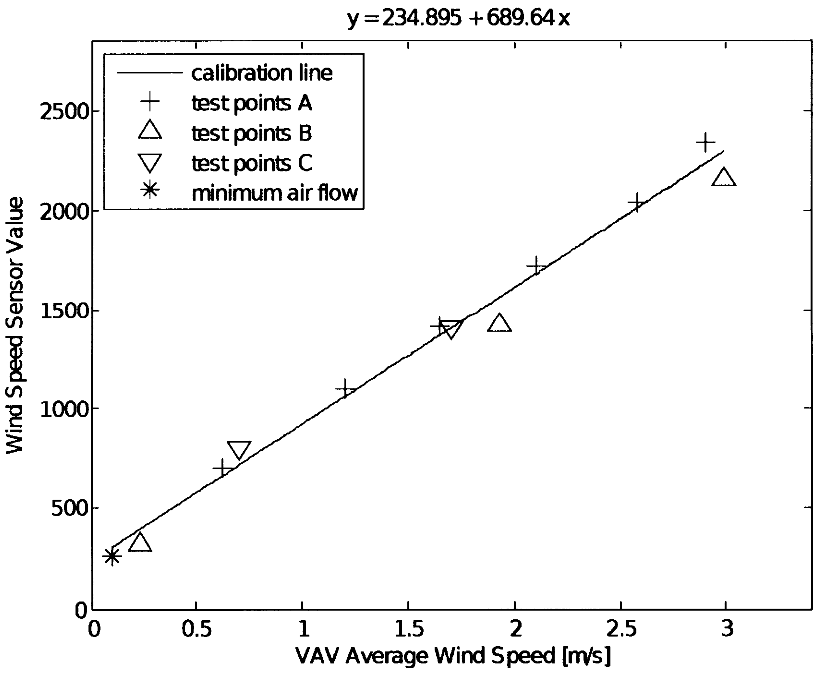


Figure 3.4.8: In-situ Calibration of Shimmed Wind Speed Sensor: Node 176

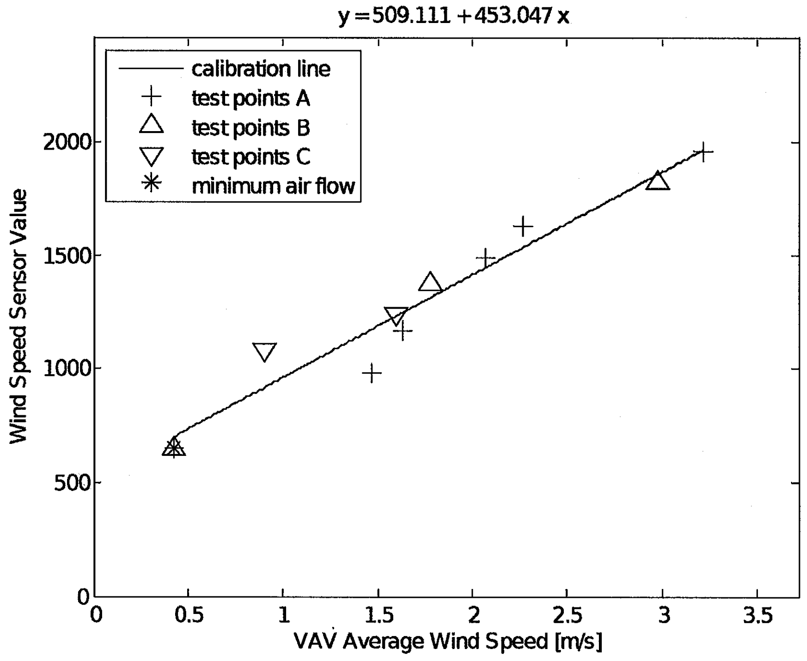


Figure 3.4.9: In-situ Calibration of Shimmed Wind Speed Sensor: Node 180

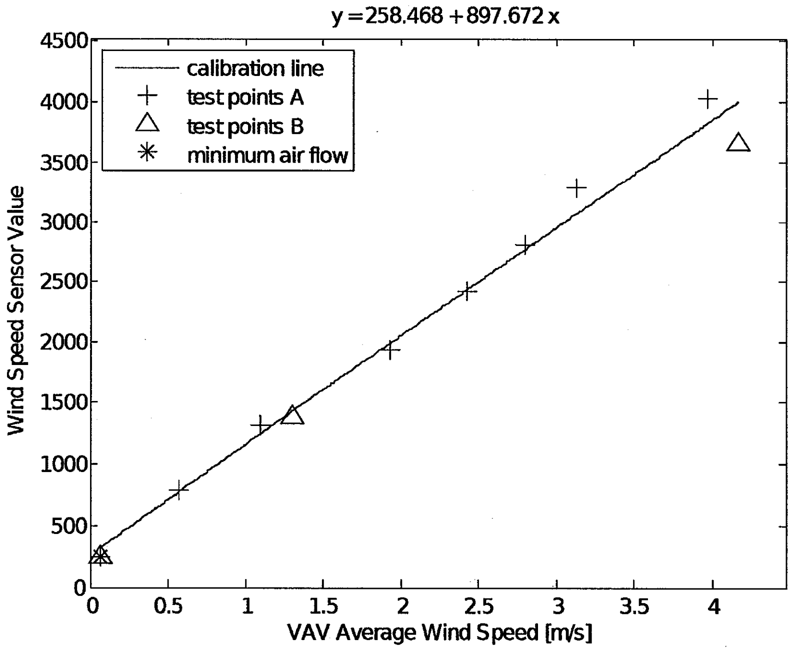


Figure 3.4.10: In-situ Calibration of Shimmed Wind Speed Sensor: Node 184

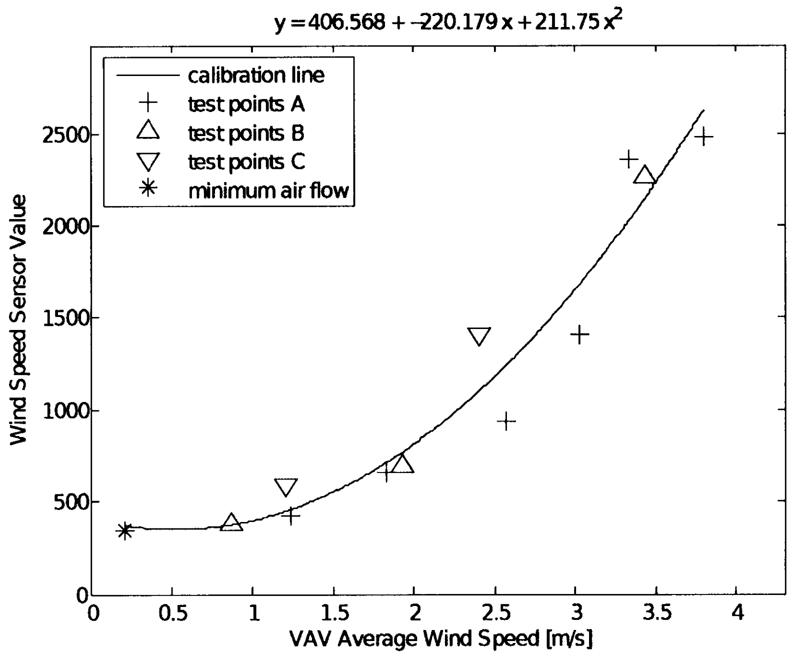


Figure 3.4.11: In-situ Calibration of Shimmed Wind Speed Sensor: Node 188

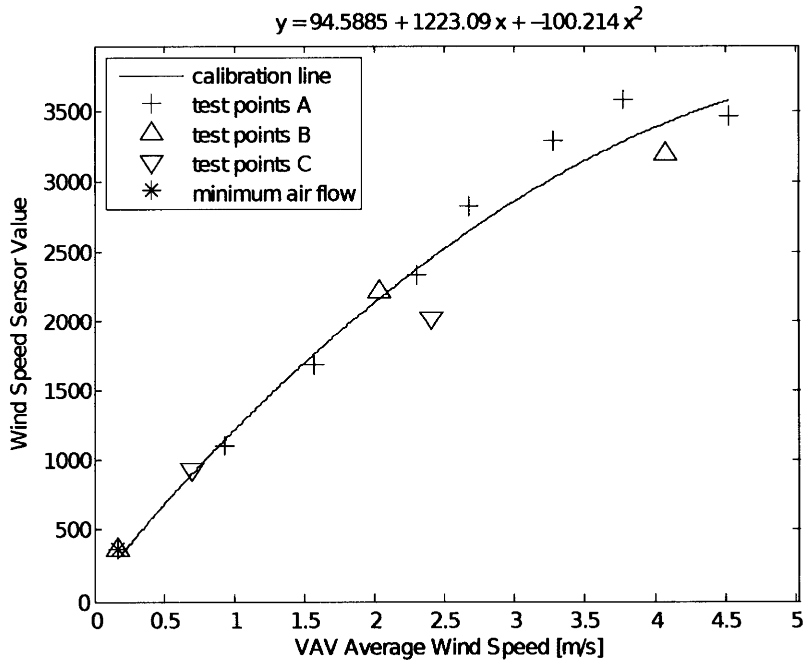


Figure 3.4.12: In-situ Calibration of Shimmed Wind Speed Sensor: Node 200

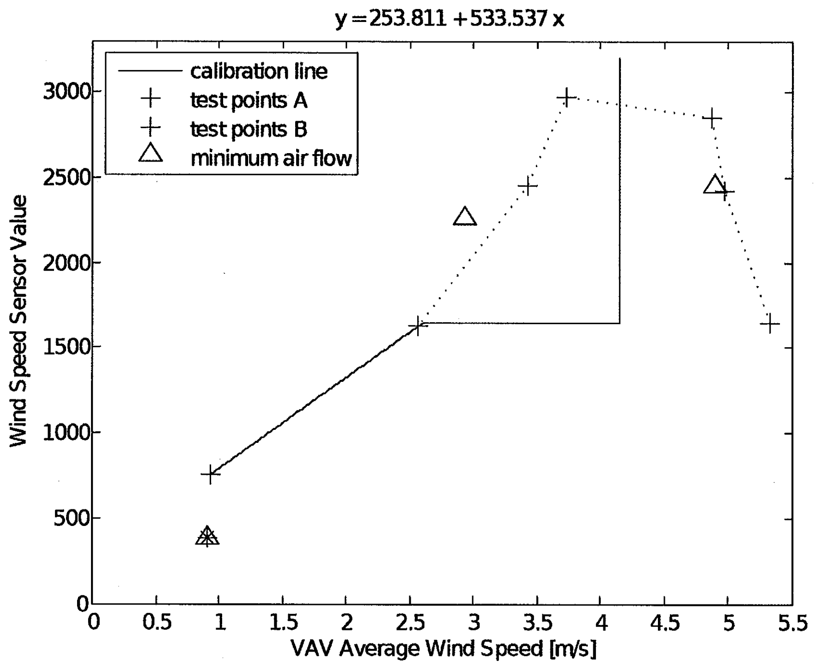


Figure 3.4.13: In-situ Calibration of Shimmed Wind Speed Sensor: Node 204

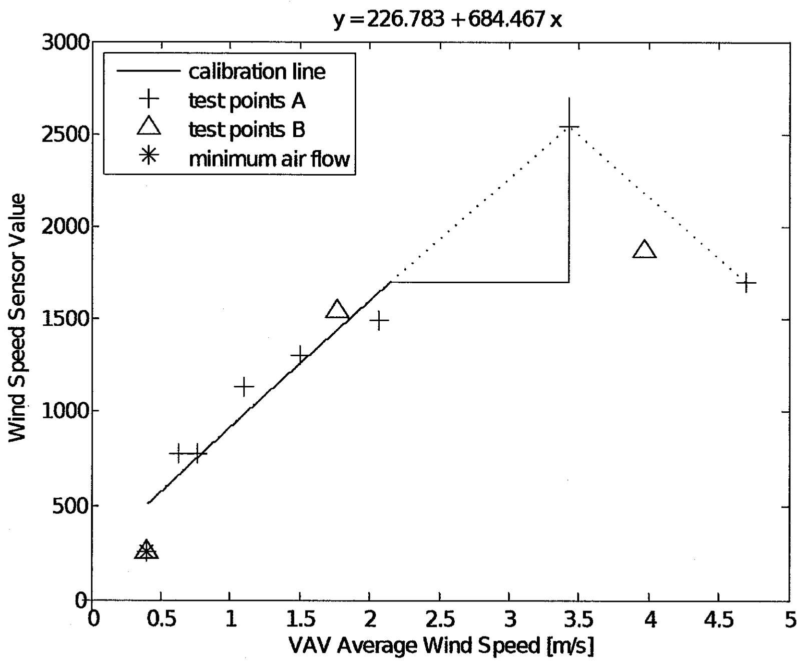


Figure 3.4.14: In-situ Calibration of Shimmed Wind Speed Sensor: Node 248

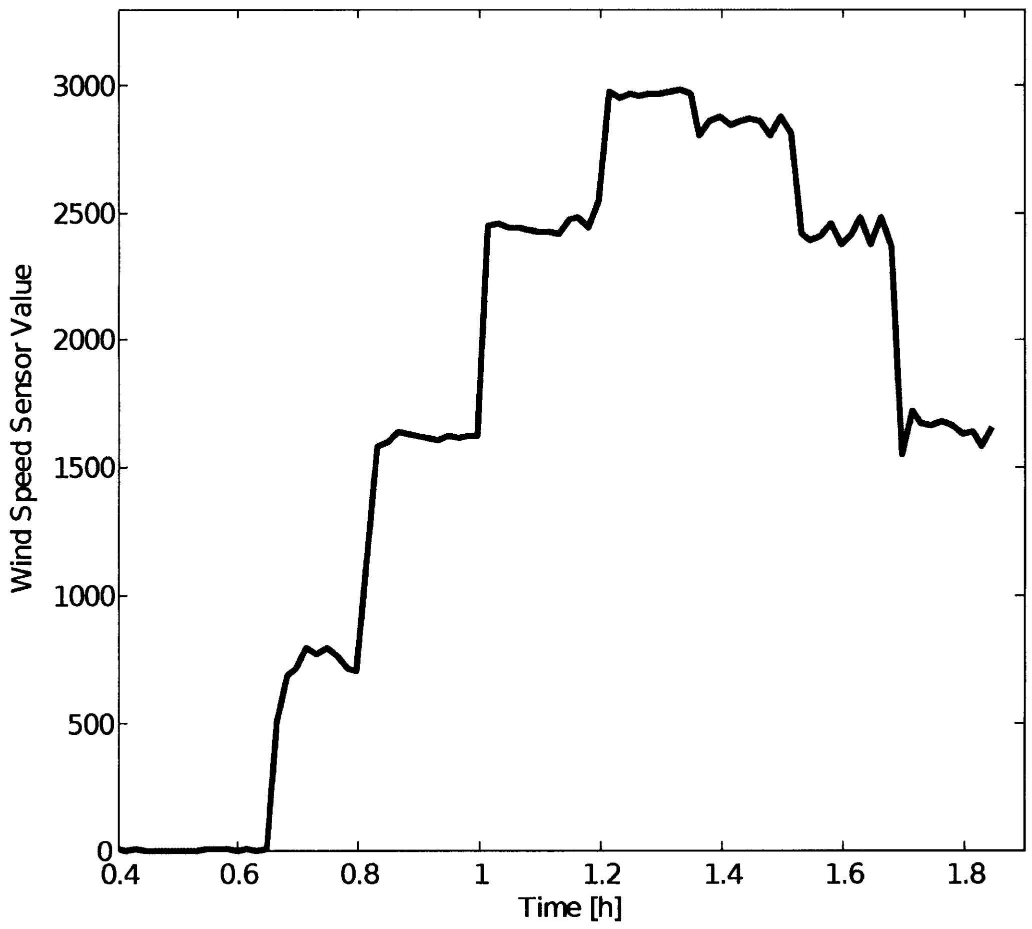


Figure 3.4.15: Example Non-linear Wind Speed Sensor: Node 204

It is not clear what the appropriate solution is for the wind speed sensing requirement. Most industrial applications measure the pressure drop across a Bernoulli plate. Although these are currently installed in buildings and could be connected to, this makes installation more difficult. A hot-wire anemometer would give excellent data, but is very costly and degrades with time due to particulate build-up. It is possible that with a large enough demand, the hot-wire solution could come down in price to make it viable for this application. It may also be the case that accurate wind speed is not required, in which case a simple IR beam-breaking wind vane would be the most economical choice of all.

3.5 Room Node

The room nodes are the simplest portion of the system, although arguably the most important. They collect and distribute all the RF packets, and keep track of the location of portable nodes in the network. They use the ZigBit module and the Lantronix Xport Direct [76] to accomplish these goals. The XPort is configured in UDP mode for ease of implementation. The full configuration settings are shown in Table 3.5.1.

Setting	Value
Baud Rate	38400
I/F	0x4c
Flow	0x00
Port No.	10001
Connect	0x0c
Datagram	0x01
Broadcast	N
Remote IP	X.X.X.X
Port	10001
Pack Ctrl	0x30
Send Char1	0xfe
Send Char2	0xfd

Table 3.5.1: XPort Configuration Settings for Room Node

The room nodes use a Panasonic AMN31112 PIR motion sensor [77] with a 5 m sensing radius. The output is digital, with a level change for every detected motion. These transitions are counted by the microcontroller over a 30 s sampling period, giving the average activity for that period. These data, along with temperature, humidity, and light levels, are sent out every 30 s over the Ethernet port. The temperature, humidity, and light sensors are identical to the ones used on the portable nodes, and they are kept on continuously as there is a wired power source. An assembled room node is shown in Figure 3.5.1.

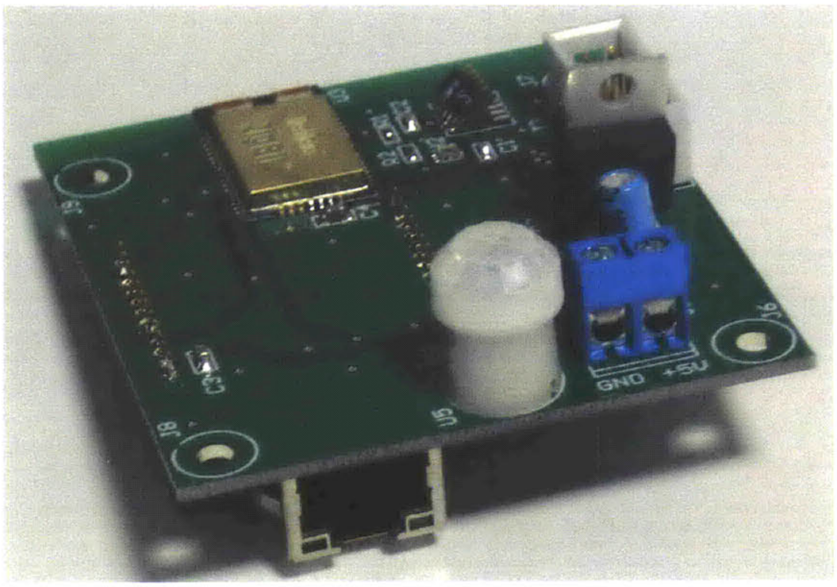


Figure 3.5.1: Room Node Circuit Board

The only difference in its sensing is that the SHT15 temperature sensor is mounted at an angle to the board (see upper right hand corner of Figure 3.5.1). This is done to help limit the thermal effects from the other warm components on the board. The XPort is the main current sink on the board, running at 200 mA when active, and 100 mA idle. since the board runs from 5 V, this gives 0.5 W to 1 W of power dissipation. The extra heat produced raises the sensor temperature significantly (~10 °F) and makes it sensitive to local air flow, as convective cooling now plays a role since the sensor temperature is no longer at the ambient temperature. The angle mounting only slightly improves this situation, but the sensor still gives useful results.

Chapter 4

Control System Design

The entire Personalized Comfort Control System was designed to operate in summer weather, although this limitation is merely a matter of convenience, as the tests were performed in the summer months. The tests included ten people with wearable sensors, six room nodes monitoring their positions and the activity levels in the rooms, six corresponding thermostat modules returning room temperature and humidity, two outdoor environmental sensors, seven independent damper controllers, and two motorized window operators. Although this collection of 33 nodes is extremely small in comparison to the usual building installation of hundreds of thermostats and VAV damper motors, it is a more difficult control scenario, as the usual one-to-one, or one-to-many mappings are no longer valid. There are multiple sensor inputs that may control any number of output devices, at any one time. To complicate matters even further, this mapping necessarily changes with time as people physically move throughout the network, altering the locations of their cooling needs.

To effectively handle the complex mapping requirements of this MIMO network, a hybridized control system is employed. In hybrid systems, individual modules exchange information and trade off responsibilities in an ad hoc but hierarchical fashion. This fits particularly well with the topology of sensor networks, as the control layer matches the physical instantiation. Ultimately, if it were desired, the entire control scheme herein could run in the network on the individual nodes (the location module being the only difficult one to implement). In addition to reducing wiring costs and the single point of failure that a central server represents, this could also increase battery life in the portable nodes by eliminating the need to transmit all data every minute for processing. This also creates a much more secure and private scenario for personal data contained within the portable nodes.

A full system model chart can be seen in Figure 4.0.1. The system consists of a Control Module, Location Module, Window Module, Outdoor Module, Thermostat Module, Portable Module, and Room Module. Each node in the network has a module associated with it, to keep track of its own local state and needs. The Control Modules receive setpoint commands from the Window, Thermostat, Room, and Outdoor Modules, and make decisions concerning how to appropriately control the associated damper and window motors to reach these setpoints. The Location Modules aggregate all the wireless transmissions from the portable nodes, and create a location table that the Room Modules can access to find out who is where. The Window Modules receive information from the window control nodes, and keep a log of when the window was last manually operated. This is accessed by the Outdoor Modules when determining whether to open or close the windows. The Window Modules also monitor the air speed coming in through the window, and close the windows if it becomes to windy. The Outdoor Modules receive outdoor temperature and humidity information, and poll the indoor Thermostat Modules to decide whether to open the windows and let in cool outside air. The Thermostat Modules track individual room air temperatures, and poll the Room Modules to determine whether or not they should be performing control actions. Based upon the Room Modules' states, the Thermostat Modules will either run normal control, setback control, or no control at all. For summer operation, the Thermostat Modules also determine if the room is too cold, at which point the Window Modules will not open the windows. The Portable Modules keep track of each individual user, and whether they are active and comfortable. The Room Modules poll the Portable Modules and Location Modules to find out if users are currently active, where they are located, and if they are comfortable. Based upon these findings, they either relinquish control to the Thermostat Modules, or work to minimize the discomfort in the rooms. Each of these modules is explained in depth in the following sections.

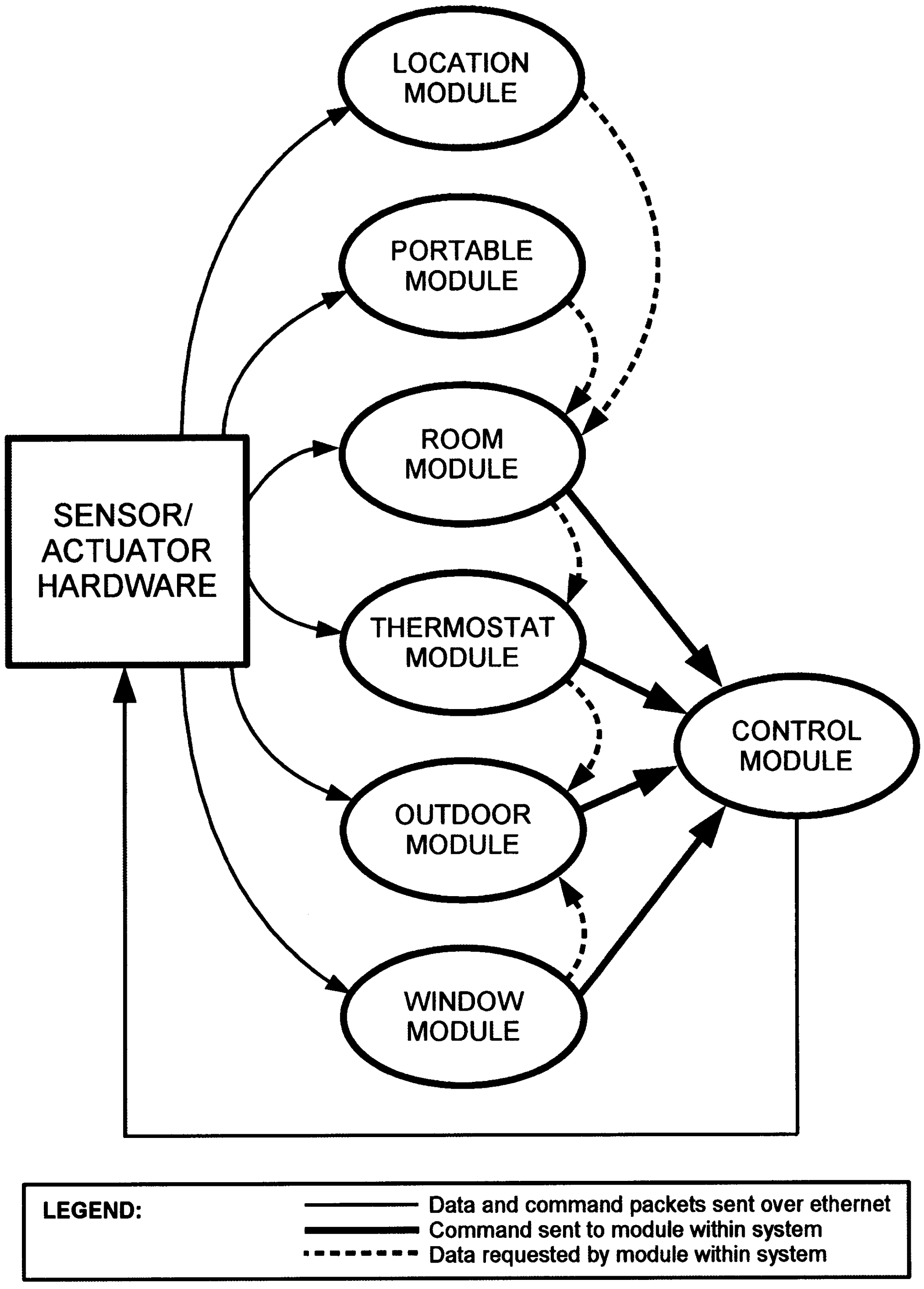

LOCATION MODULE
PORTABLE MODULE
ROOM MODULE
SENSOR/ ACTUATOR HARDWARE
THERMOSTAT MODULE
CONTROL MODULE
OUTDOOR MODULE
WINDOW MODULE
LEGEND:
Data and command packets sent over ethernet
Command sent to module within system
Data requested by module within system

Figure 4.0.1: Control System Model

The system also incorporates a back-up function, which is not related to the hybrid controller, but is critical to its proper functioning. After a short period of running, the control system establishes a complicated map of the state of the sensor network. This map continues to update for the entire duration of its operation. Some states evolve on a minute-by-minute basis, and others over hours or days. To maintain this state during system crashes, critical components are backed up to the hard drive whenever they are changed. Upon restart, these components are loaded before the program begins to run. Not all components are backed up, as some are modified too often, or are not important enough, to warrant the overhead of continual file writes. Currently, all comfort preference variables and expected occupancy variables are backed up.

4.1 Control Module

The Control Module represents one of the few hierarchical elements in the network, as it merely processes and passes incoming commands. The main purpose of the Control Module is to map the desired temperature control signals to the correct output devices, and maintain state between the various control hand-offs that occur throughout the day, as many devices control the same actuator. In this way, local control of the damper is maintained in a stable fashion, regardless of the incoming control signals. This also allows for a central repository for storing control variables specific to certain rooms. A flowchart for the Control Module is shown in Figure 4.1.1.

The Control Module implements a hybrid proportional and integral (PI)/dead-band controller. The PI controller is chosen for its simplicity and stability, limiting the number of unknown variables when analyzing the full control loop. Sufficient performance is obtained from the PI controller, such that a full proportional, integral, and derivative (PID) controller is not necessary. The predictive setback algorithm (see Section 4.7) further reduces the need for the fast system response time of derivative control, as the room temperature is set before it is occupied. A dead-band is placed on the issuing of control signals around the setpoint in order to reduce network traffic. Since all of these commands are required to be sent

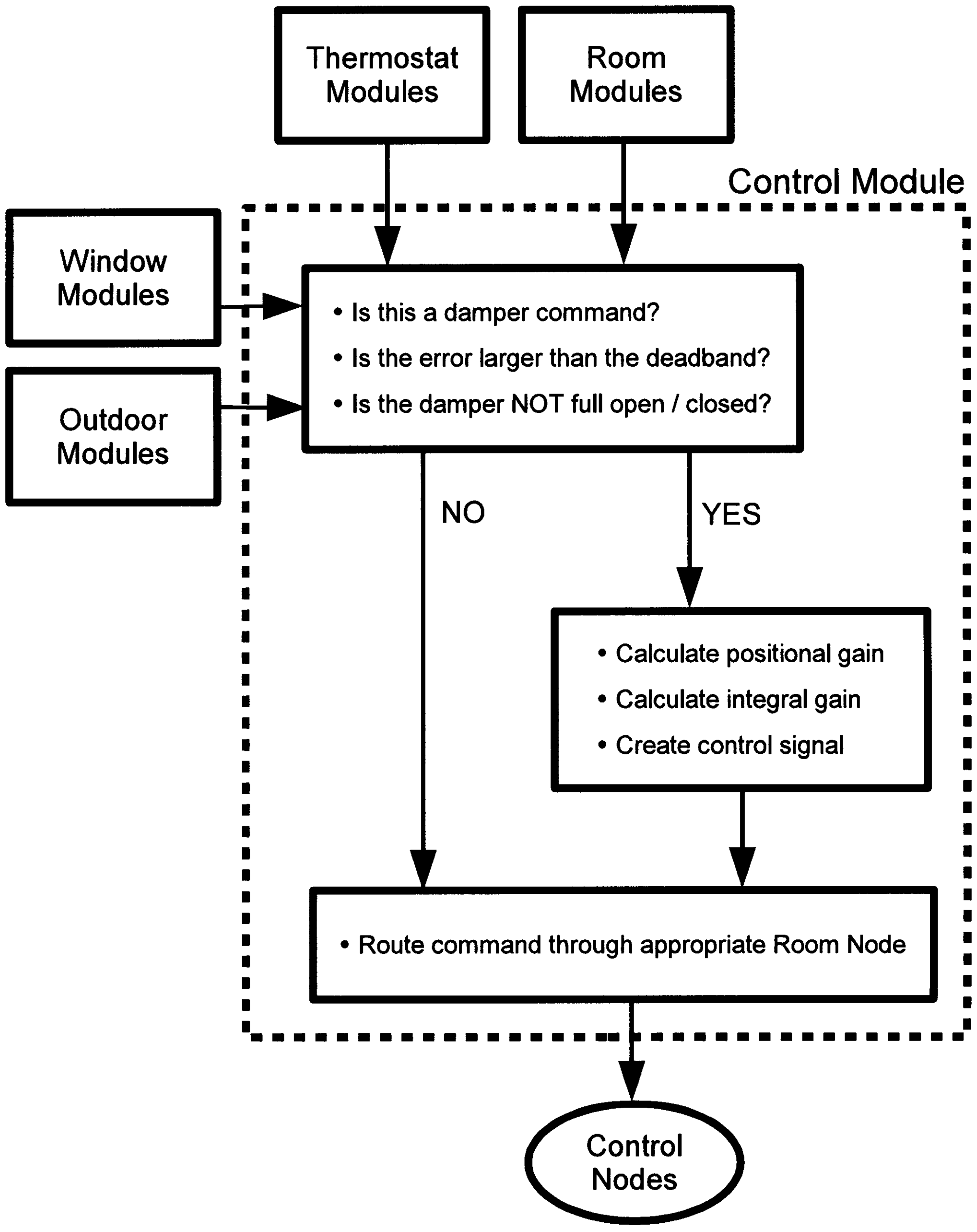

Thermostat Modules
Room Modules
Control Module
Window Modules
Outdoor Modules
• Is this a damper command?
• Is the error larger than the deadband?
• Is the damper NOT full open / closed?
NO
YES
• Calculate positional gain
• Calculate integral gain
• Create control signal
• Route command through appropriate Room Node
Control Nodes

Figure 4.1.1: Control Module Flowchart

over the wireless network, and the incoming control signal is digital with a finite resolution, in the absence of a dead-band a control signal would be sent out every minute, increasing the probability of collisions. This dead-band is chosen to be ± 2 LSB ($\pm 0.1\,^{\circ}$F) as a good balance between maintaining temperature accuracy and limiting excessive transmissions. In practice, this keeps temperature excursions between $\pm 0.2\,^{\circ}\mathrm{F}$.

The PI gains are set via a combination of manual tuning and the Ziegler-Nichols method [78]. The Ziegler-Nichols method involves increasing the feedback gain until resonance is reached, and then setting the gain parameters as a function of the critical resonance gain (K_c) and period of the critical resonance (τ_c). The proportional gain (K_p) is set to $0.5 \times K_c$, and the integral gain (K_i) is set to $1.2 \times K_p/\tau_c$. The Ziegler-Nichols method was not employed exclusively, as long evaluation times are required to check for stability or resonance. After two days of testing, it was determined that the response was well enough understood to make a reasonable estimate of the optimal PI gains.

The gain is increased by factors of two until instability is observed. The resonance can be seen in Figure 4.1.2, and is measured as having a period of approximately 0.2 hours at a positional gain of 32,000. This gives $K_p = 0.5 \times 32,000 = 16,000$ and $K_i = 1.2 \times 16,000/(0.2 \times 3,600\,s) = 26.7$. Application of these control gains still produces excessive overshoot and oscillation, as can be seen in the first half of Figure 4.1.3, so the gains are halved again, assuming that the actual critical resonance could be anywhere between the 16,000 gain test and the 32,000 gain test. This produces gains of $K_p = 8,000$ and $K_i = 13.3$, which can be seen to improve performance greatly, as shown in the second half of Figure 4.1.3. These gains are then manually adjusted to their final gains of $K_p = 8,000$ and $K_i = 10$ after more testing. An example of their performance can be seen in Figure 4.1.4, with a four degree step response in less than an hour. The $\pm 0.2\,^{\circ}\mathrm{F}$ dead-band oscillation is also visible in this figure. Since all of the offices are of similar size, and have similar cooling equipment, the same gains are used for all offices. The only exception to this is a double office, which has two VAV boxes that are operated in parallel at one half the controller gains.

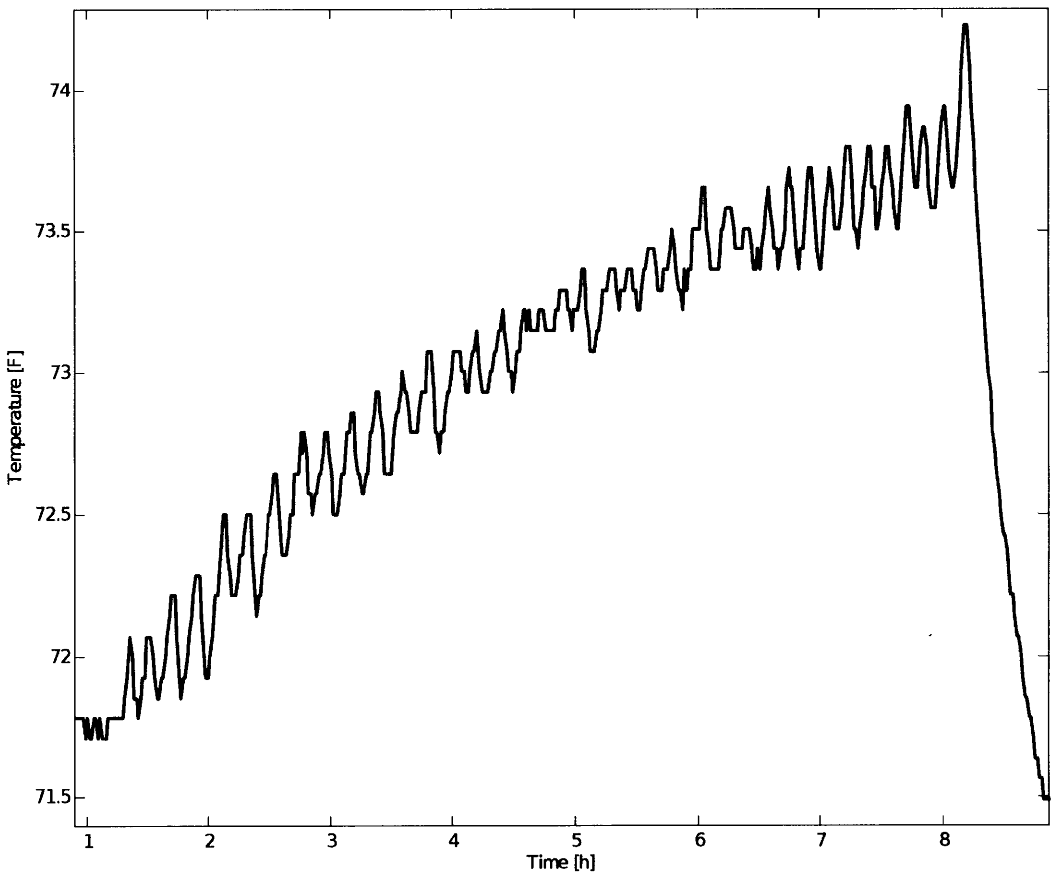


Figure 4.1.2: Control Oscillation for PI Gain Calculations at $K_p = 32,000$

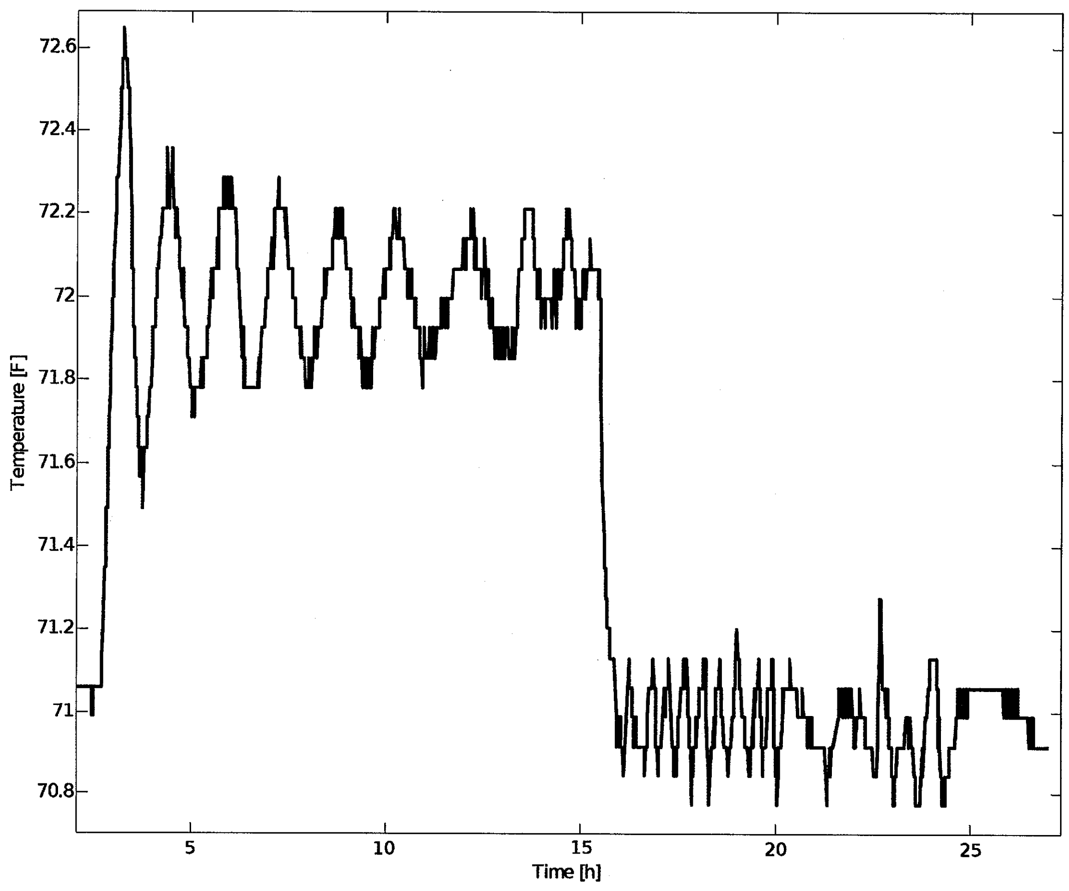


Figure 4.1.3: Overshoot: First Half is $K_p = 16,000$, Second Half is $K_p = 8,000$

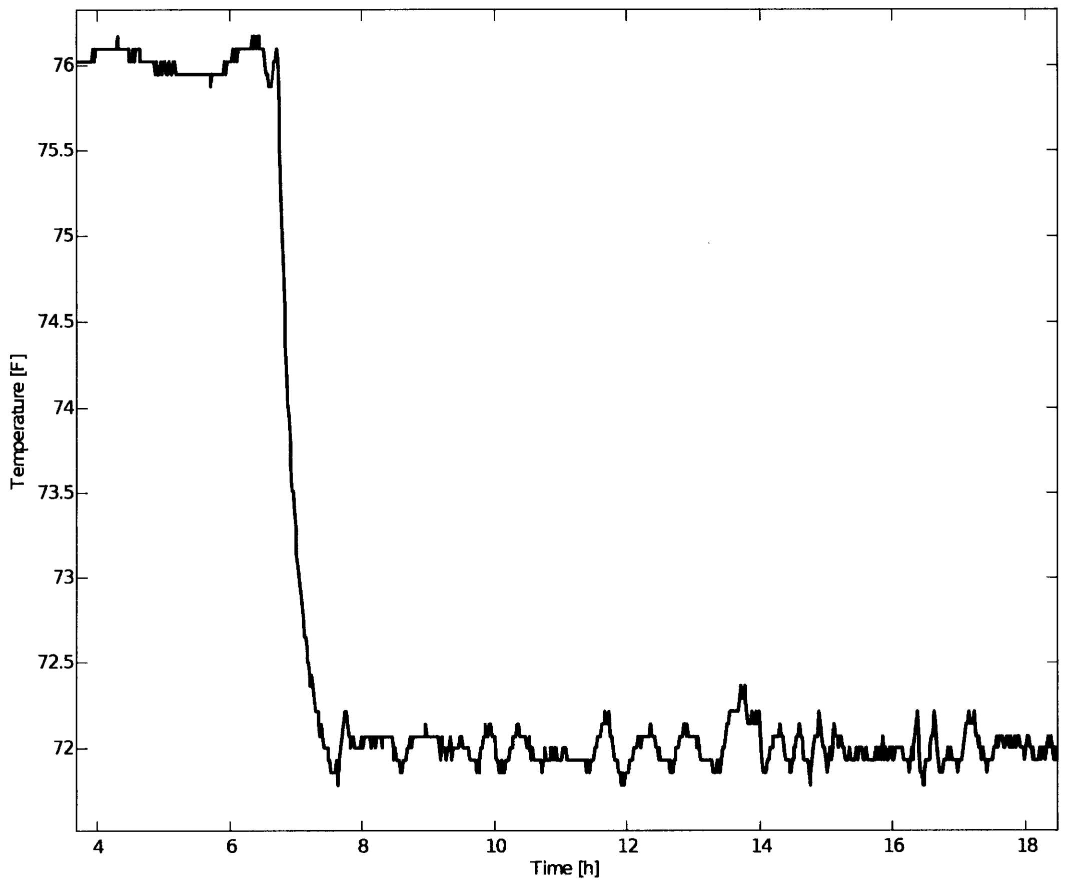
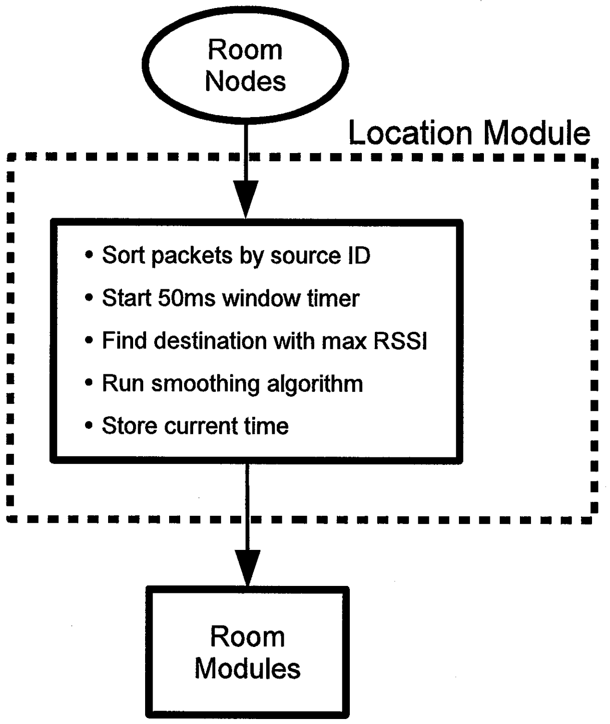


Figure 4.1.4: Example of Final Control Settings in Practice

4.2 Location Module

The Location Module (see Figure 4.2.1) is in charge of keeping track of where each node is located in the system. It does this by aggregating all of the location packets, which are received by each room node and passed along via Ethernet to the network hub. Since packets arrive in tight succession, it is important to window out those packets which do not belong to a particular broadcast transmission. To do this, the Location Module starts a timer when the first location packet arrives from a particular portable node. It then only accepts packets for the next 50 ms. Since location is based upon the strongest RSSI of these received packets, the actual location may not be reflected by the data, as multi-path can easily confound RSSI data. To accommodate this, the Location Module performs a smoothing algorithm on the incoming data, taking a majority vote on the past three location results. If no majority can be reached, it takes the currentresult. This allows for temporary excursions to be excluded, but relatively quick response to changes in location.



Room Nodes
Location Module
• Sort packets by source ID
• Start 50ms window timer
• Find destination with max RSSI
• Run smoothing algorithm
• Store current time
Room Modules

Figure 4.2.1: Location Module Flowchart

An example of the received RSSI for a portable node is shown in Figure 4.2.3. The data represents a time when there are no people in the building, and the node is neither being worn, nor is it moving. This node is placed in perhaps the most challenging location, directly in the center of the network of five receivers. It is in this location that it would most likely be picked up by neighboring nodes. A map of the transmitter and receiver locations can be seen in Figure 4.2.2. It is important to note that the distances on the map are not representative of the actual distances from the portable sensor to the room nodes, as the room nodes are mounted on the ceiling at a height of 3 m. The portable node is usually at a height of approximately 1 m, as the user spends most of his or her time seated. This makes the actual distance $D_{act} = \sqrt{2^2 + D_{map}^2}$, where D_{map} and D_{act} are in meters. A listing of the actual distances is shown in Table 4.2.1 for the tests done in this section.

It can be seen from Figure 4.2.3 that the RSSI values strongly correlate with distance between transmitter and receiver, although this relationship is neither fixed nor linear. Since the system does not need to know exact location, rather merely which room a person is in, this rough location based around maximum RSSI values works quite well. This is further aided by the fact that the office walls are constructed of steel studs and drywall, with metal whiteboards and metal shelving units being common on the walls. The doors to the offices are also metal. This creates a tight zone around each room node. A plot of the RSSI values for the same portable node during a time when it is being worn can be seen in Figure 4.2.4. The user is generally seated for this test, with the node hanging from the neck in front of the chest. The user's back is facing room 48, and the sensor is facing room 32. It can be seen that the RSSI values begin to fluctuate much more under these conditions, as both the body and sensor are moving. The difference between RSSI values decreases greatly, but the closest node still retains the strongest signal. The voting algorithm helps window out short term false readings, and its output for this same time period can be seen in Figure 4.2.5. The location algorithm reports two false readings over the hour of testing, giving a success rate of $58/60 = 96.7$ percent, as the portable module transmits once a minute.

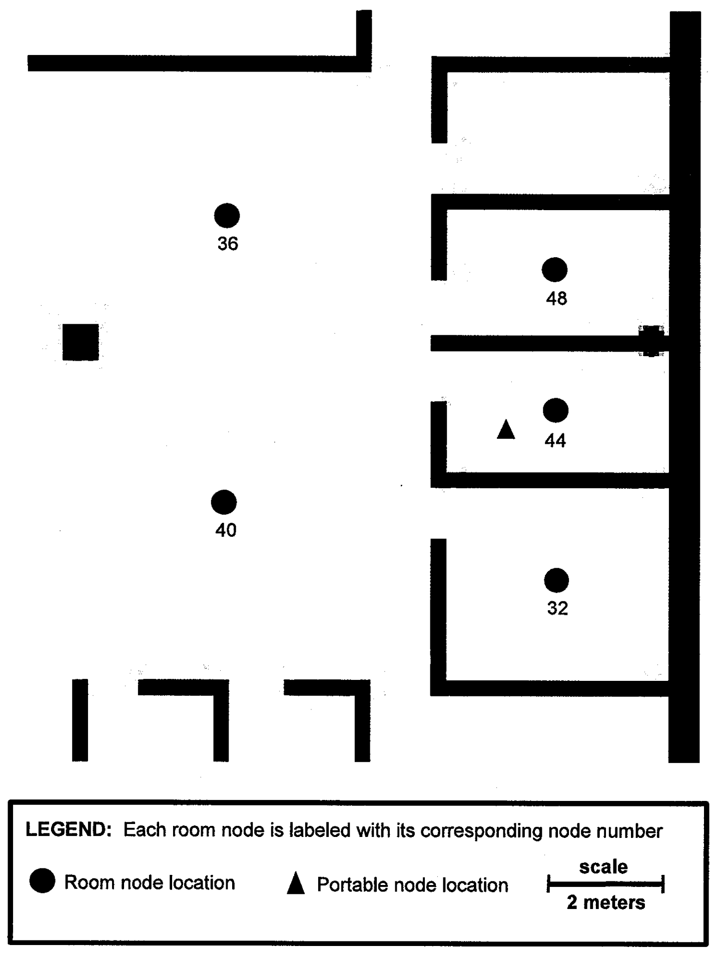

36
48
44
40
32
LEGEND: Each room node is labeled with its corresponding node number
Room node location
Portable node location
scale
2 meters

Figure 4.2.2: Map of Receive and Transmit Locations for RSSI Tests

Room Node	Distance [m]
32	3.40
36	6.32
40	5.39
44	2.23
48	3.61

Table 4.2.1: Distance from Portable Node to Room Nodes

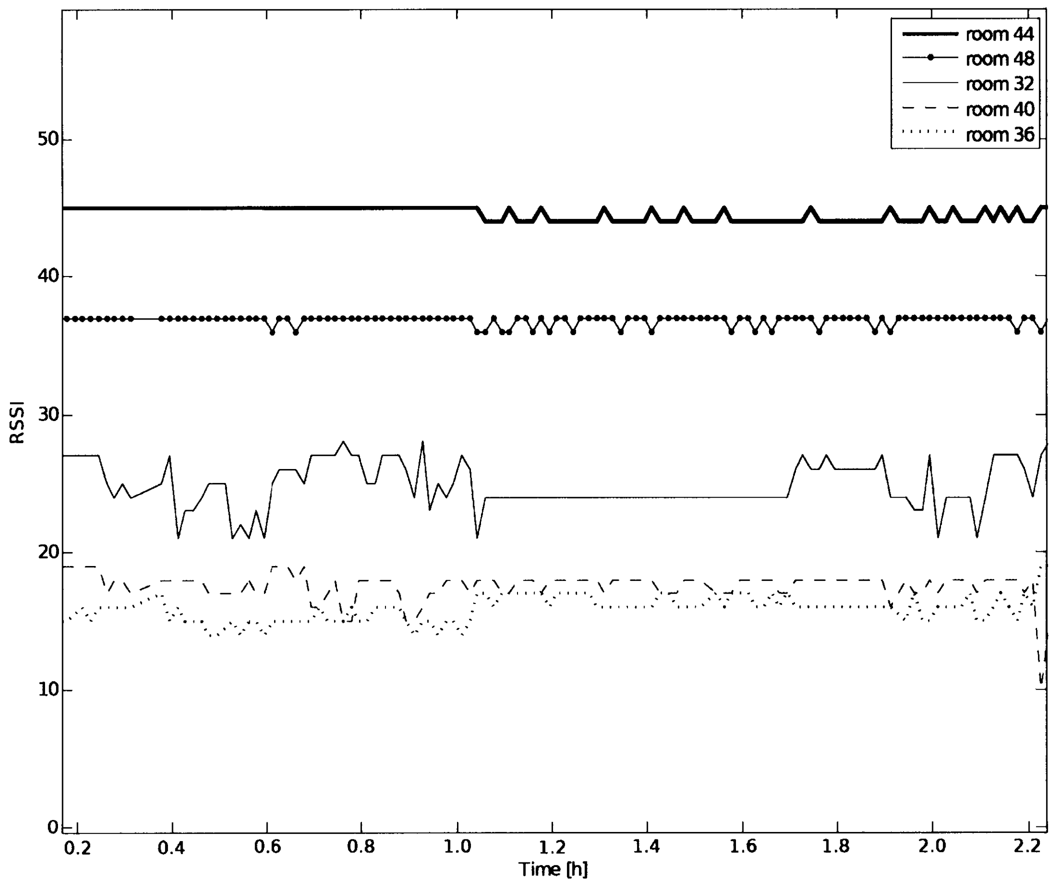


Figure 4.2.3: Example of RSSI Values for Unoccupied Period

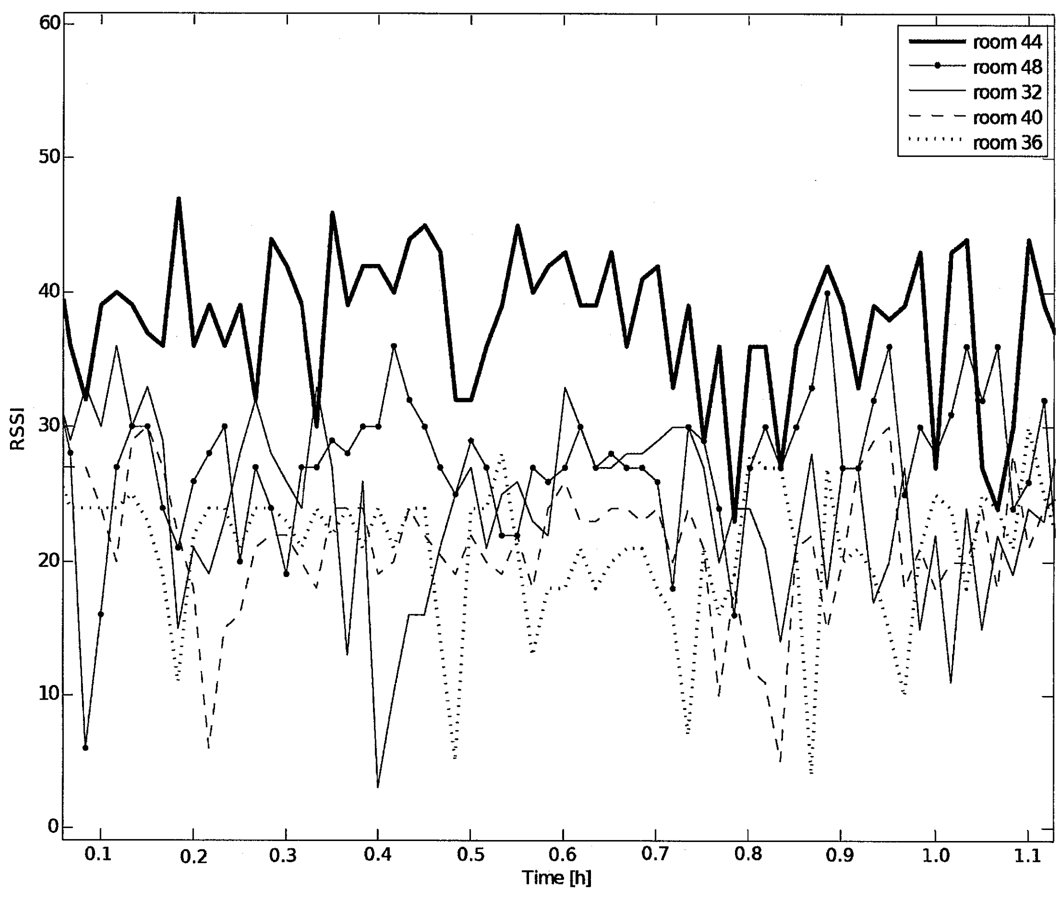


Figure 4.2.4: Example of RSSI Values While Sensor is Worn

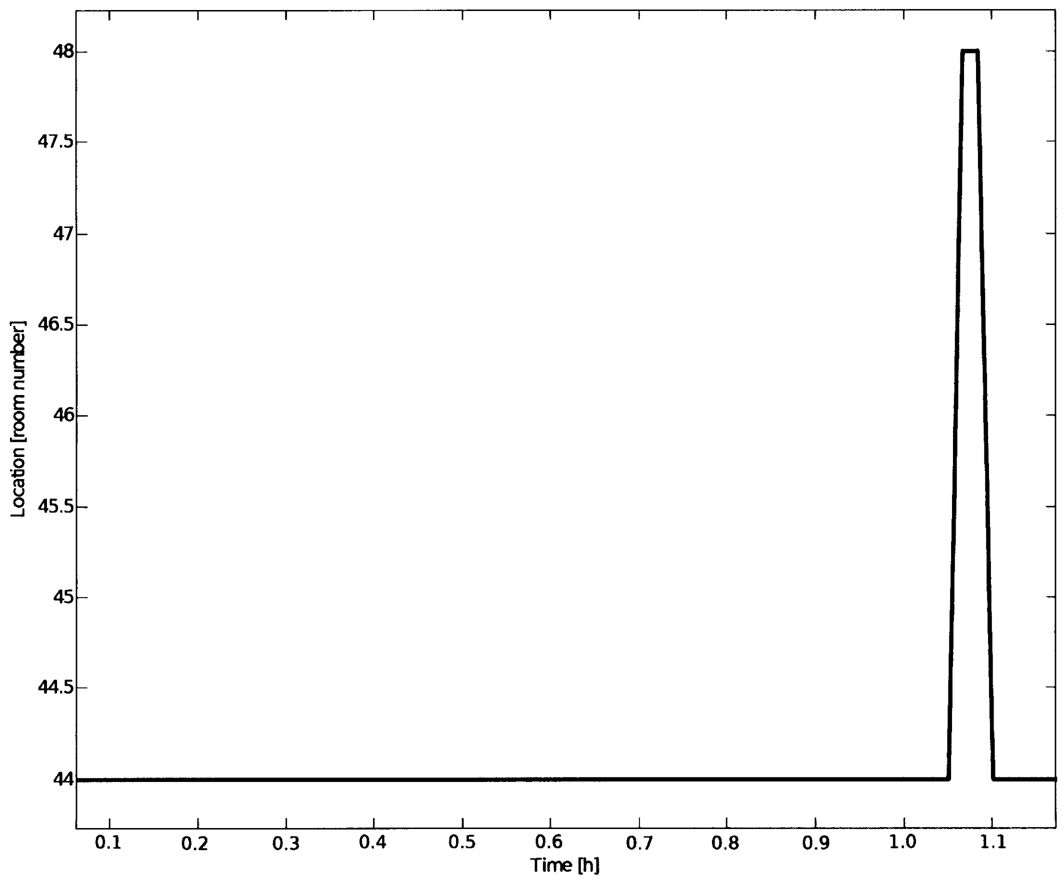


Figure 4.2.5: Location Algorithm Results for Period Shown in Figure 4.2.4

4.3 Window Module

The Window Module receives data from the two window controllers in the network. These window controllers monitor the light levels, temperature, humidity, wind speed, and motor state. The Window Module currently only takes into account the motor state and wind speed, and a flowchart of its actions can be seen in Figure 4.3.1. Via the motor state information, the Window Module can be notified if a user has pressed a button to manually open or close the window, if the window was last driven in the open or closed direction (either by a user or some other module in the network), or if the window was driven to its limits in either the open or closed direction. From these data, the Window Module creates a repository that can be accessed by other modules, letting them know if the window is fully shut or not, or if the window has been manually operated in the past three hours or not. The manual timeout of three hours is chosen to keep other modules in the network from over-riding user preferences, but still allowing the system to respond if a window is left open overnight.

The other function the Window Module performs is keeping the wind speed coming through the window below a certain level. If wind speed greater than this level is detected, the window is closed a small amount. This process repeats until the wind speed has dropped sufficiently. Again, this automated control of the window is not performed if the window was operated manually in the past three hours. Both the window control nodes and the damper control nodes have built in functions for driving their respective motors to maintain a particular flow rate through the wind speed sensor, but these are not used at this time to allow for increased predictability of the complete system.

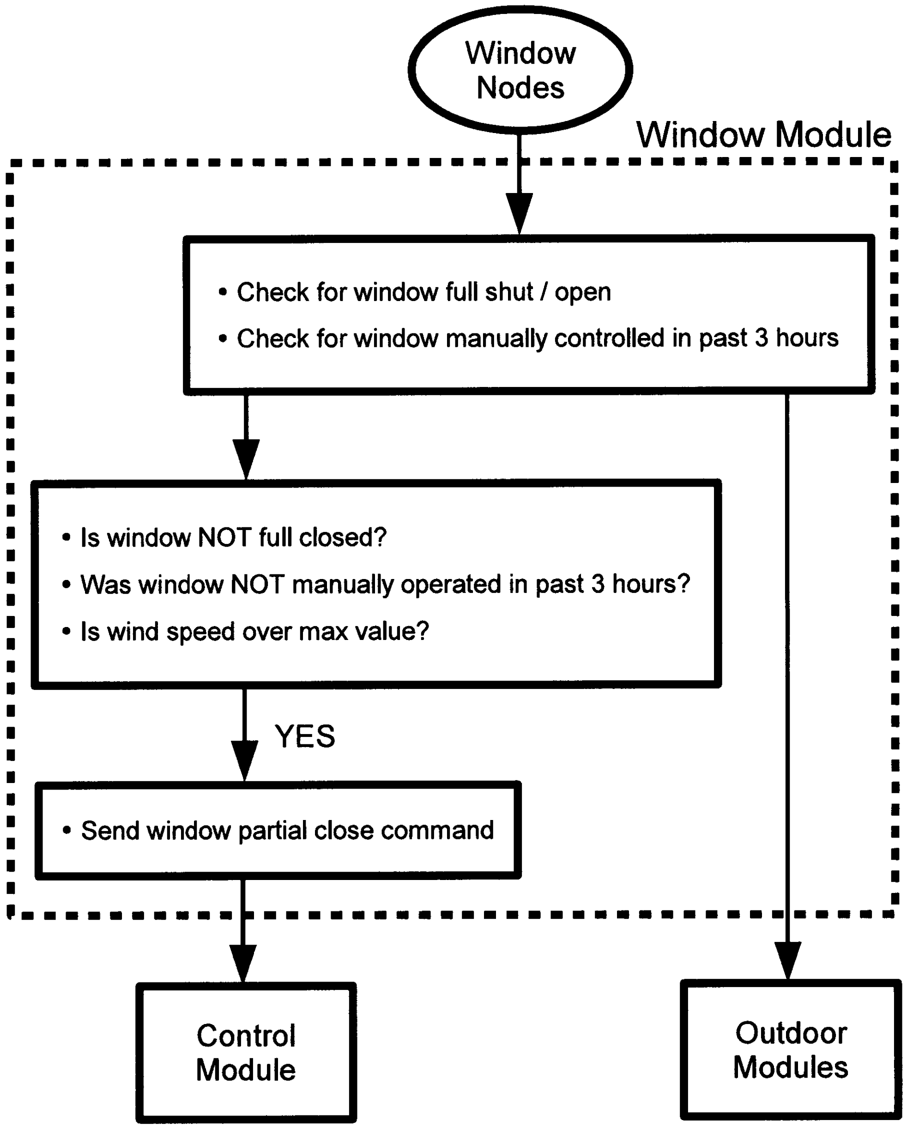

Window Nodes
Window Module
• Check for window full shut / open
• Check for window manually controlled in past 3 hours
• Is window NOT full closed?
• Was window NOT manually operated in past 3 hours?
• Is wind speed over max value?
YES
• Send window partial close command
Control Module
Outdoor Modules

Figure 4.3.1: Window Module Flowchart

4.4 Outdoor Module

The Outdoor Module is driven by the two outdoor environmental sensors, and a flowchart of its decisions can be seen in Figure 4.4.1. The outdoor temperature and humidity are stored, and the outdoor air enthalpy is calculated. The enthalpy of a gas is a measure of its total stored energy at a given pressure and temperature. Since the contributions due to pressure variations are negligible in this case, they are ignored, and the enthalpy can calculated based upon the temperature and moisture content of the air. To minimize computational load, an approximation is used based upon Equation 4.4.1, where H is the enthalpy in Btu/lb, T is the temperature in degrees Fahrenheit, and RH is the relative humidity in decimal form. This approximation gives one percent accuracy over the range of conditions experienced by the sensors [79], which is more than adequate considering it is only the difference in enthalpy that is of interest, not absolute enthalpy.

$$H = (0.007468 \times T^2 - 0.4344 \times T + 11.1769) \times RH + 0.2372 \times T + 0.1230 \qquad (4.4.1)$$

If the outside air has less energy, i.e. lower enthalpy, it is useful in cooling down a room. For this reason, the Outdoor Module will open the window if the outside enthalpy is one Btu/lb less than the indoor enthalpy. Although the main air-conditioning system has an economizer which brings in outside air under similar conditions, air brought in through the window does not require a fan to move it, thereby reducing the net energy to cool a room. The window closes if the outdoor and indoor enthalpies are equal, effectively putting hysteresis in the system and eliminating excessive window motion. The indoor enthalpy is queried by the Outdoor Module from the room Thermostat Module associated with the window, along with an indicator of room temperature. If the room is more than a degree below its setpoint, it is assumed that the VAV dampers are already closed, and that additional cooling is not necessary. All of these functions are performed after consulting the Window Module to ensure that the window isn't already open or closed, and that it hasn't been manually operated recently.

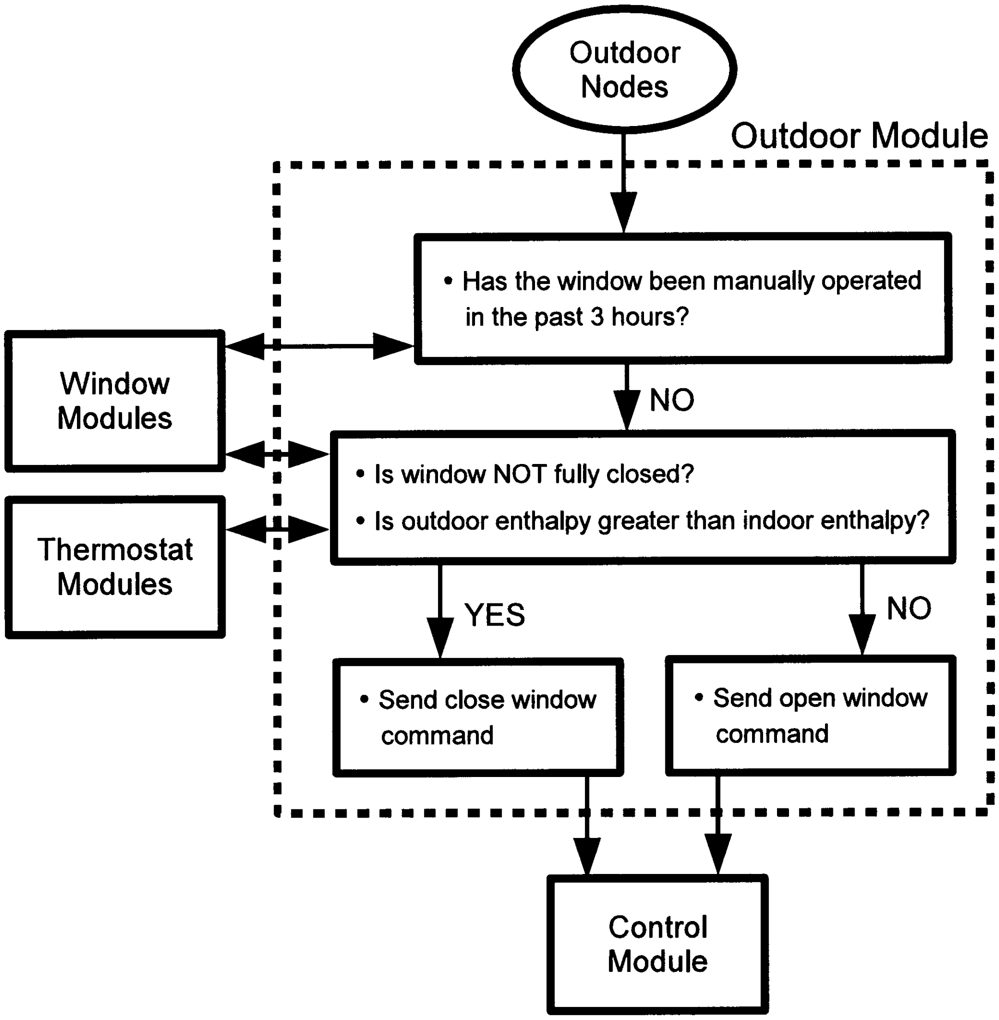

Outdoor Nodes
Outdoor Module
• Has the window been manually operated in the past 3 hours?
Window Modules
NO
• Is window NOT fully closed?
• Is outdoor enthalpy greater than indoor enthalpy?
Thermostat Modules
YES
NO
• Send close window command
• Send open window command
Control Module

Figure 4.4.1: Outdoor Module Flowchart

4.5 Thermostat Module

The Thermostat Module performs many of the room temperature control functions. It receives temperature and humidity information from sensors mounted in each room below the pre-existing thermostats. It calculates enthalpy according to equation 4.4.1, and determines if the room is too hot or too cold. Before making these assessments, it first consults the associated Room Module to check its state. If the room is occupied, but not by a person wearing a portable node, the Thermostat Module performs normal control, regulating the room temperature to a fixed setpoint, usually an average of the documented comfortable values of its occupants. If the room is unoccupied, it performs setback control, regulating the room to a fixed temperature usually six degrees Fahrenheit higher than Normal Mode. This setback temperature is limited to six degrees in order to allow for fast transitions back to comfortable temperatures when users arrive unexpectedly. In cases where the room is occupied by users with portable nodes, the Thermostat Module relinquishes control of the VAV dampers to the Room Module, which has more information as to who is present and what his or her comfort needs are. A flowchart for the Thermostat Module can be found in Figure 4.5.1.

The Thermostat Module has a number of variables that are accessible by other parts of the network. The enthalpy, temperature, and humidity are used widely throughout the system. The Thermostat Module also has a series of lower temperature limits it checks for, and sets a 'too cold' indicator if any of them occur. This function is necessary as there may be many different control operations happening at any one time, some of which might be conflicting. For example, both the VAV dampers and window motors can be operated to cool the room down, and they act independently, so a hard-stop is required to keep the system from driving itself too far in one direction. Depending upon the current room occupancy condition, different lower limits are used. The lowest limit is during setback mode, when occupancy comfort is not relevant and reducing energy consumption is most important. In these conditions, temperatures are allowed to drift down to 65 °F, helping to store energy in the thermal mass of the internal building. If the room is occupied, the lower limit is set to one degree Fahrenheit lower than the control temperature. In cases where the

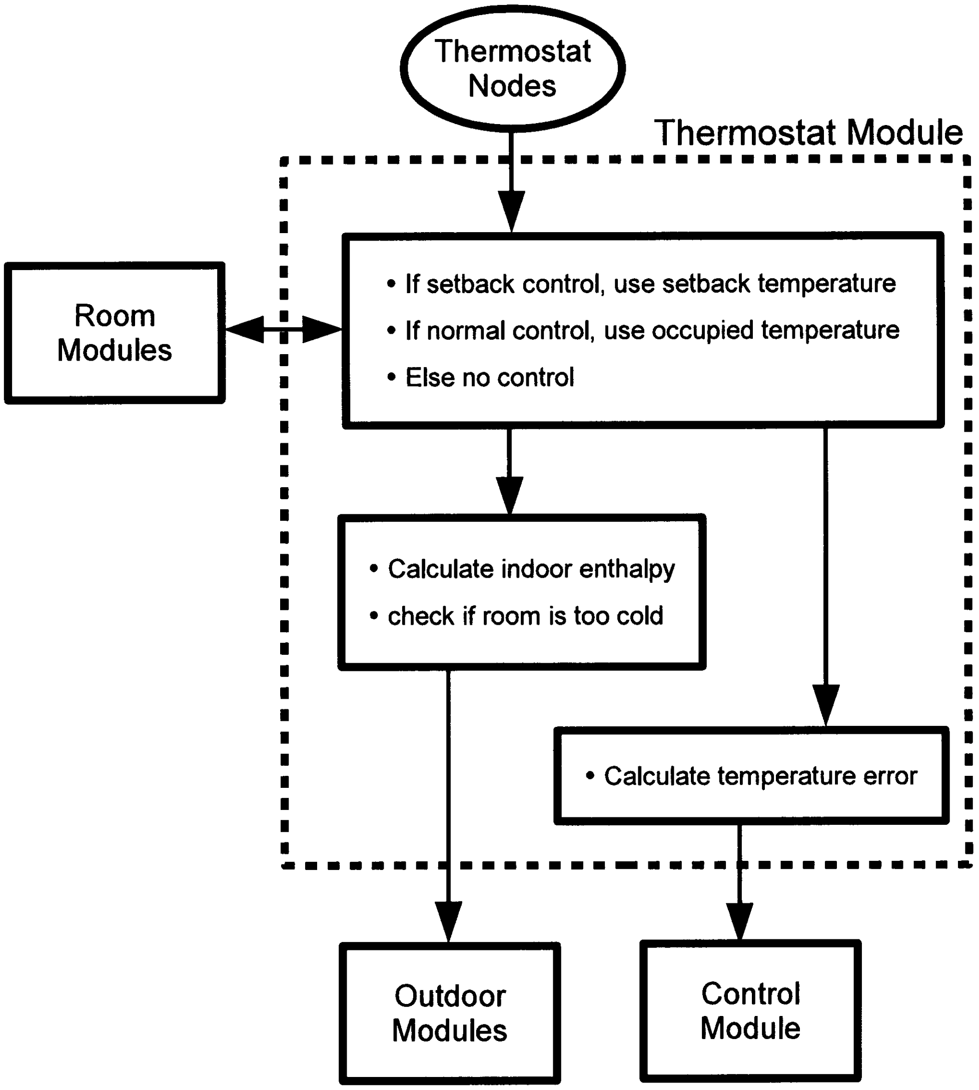

Thermostat Nodes
Thermostat Module
Room Modules
• If setback control, use setback temperature
• If normal control, use occupied temperature
• Else no control
• Calculate indoor enthalpy
• check if room is too cold
• Calculate temperature error
Outdoor Modules
Control Module

Figure 4.5.1: Thermostat Module Flowchart

room is occupied by users wearing portable nodes, this lower limit is kept at 70 °F. This is because the desired temperature is unknown by the Thermostat Module, and it is best to take the conservative approach of not cooling the room too much, but instead allowing the main system to control the temperature.

4.6 Portable Module

The Portable Module keeps track of each user in the system, and determines whether the user is active and comfortable. A flowchart of the Portable Module's actions can be seen in Figure 4.6.1. Activity determination is necessary, as the nodes are often left on the users' desks when they leave for the day. The activity determination is based upon activity and temperature information from the portable node. Data from the portable node light sensor was considered as a possible indicator of activity, but the variations in light levels due to sunlight, and the lack of predictability as to whether the unit is being worn under clothes, made it an inconsistent activity detection variable. Temperature is only mildly influential in the activity algorithm, as it is the main control parameter for the system. If it were to trigger a false positive, and then drive the air-conditioning system to either reinforce its state, or attempt to change its state when this is not possible, an unstable feedback loop could result. An example of such a situation would be a user leaving a portable node next to a computer which is warm. This would heat up the portable node and could indicate activity, causing the control system to believe a user is in the room. It would then try to cool down the node, which would be impossible given the sensor's proximity to a heat source. To eliminate such scenarios, the temperature alone can not indicate activity.

The main determinants of activity are windowed mean and variance of the piezoelectric motion sensor on the portable node. It is very important that the activity algorithm quickly detects when a user is no longer wearing a sensor. The sensor temperature will quickly drop after removal, and the system would react by shutting off the air-conditioning, to the displeasure of the remaining users in the room. For this reason, a window time of three samples is chosen, giving a maximum delay of three minutes to clear the buffer. As the average delay of a windowed sample is one half the window time, the average delay for this system is only 1.5 minutes. This also has the advantage of giving a fast start-up time, so users are added to the system as soon as they arrive. The only disadvantage is the increase in false positives, which, fortunately, are short lived due to the small window time.

The incoming data from the portable nodes are first separated by time since last arrival, as

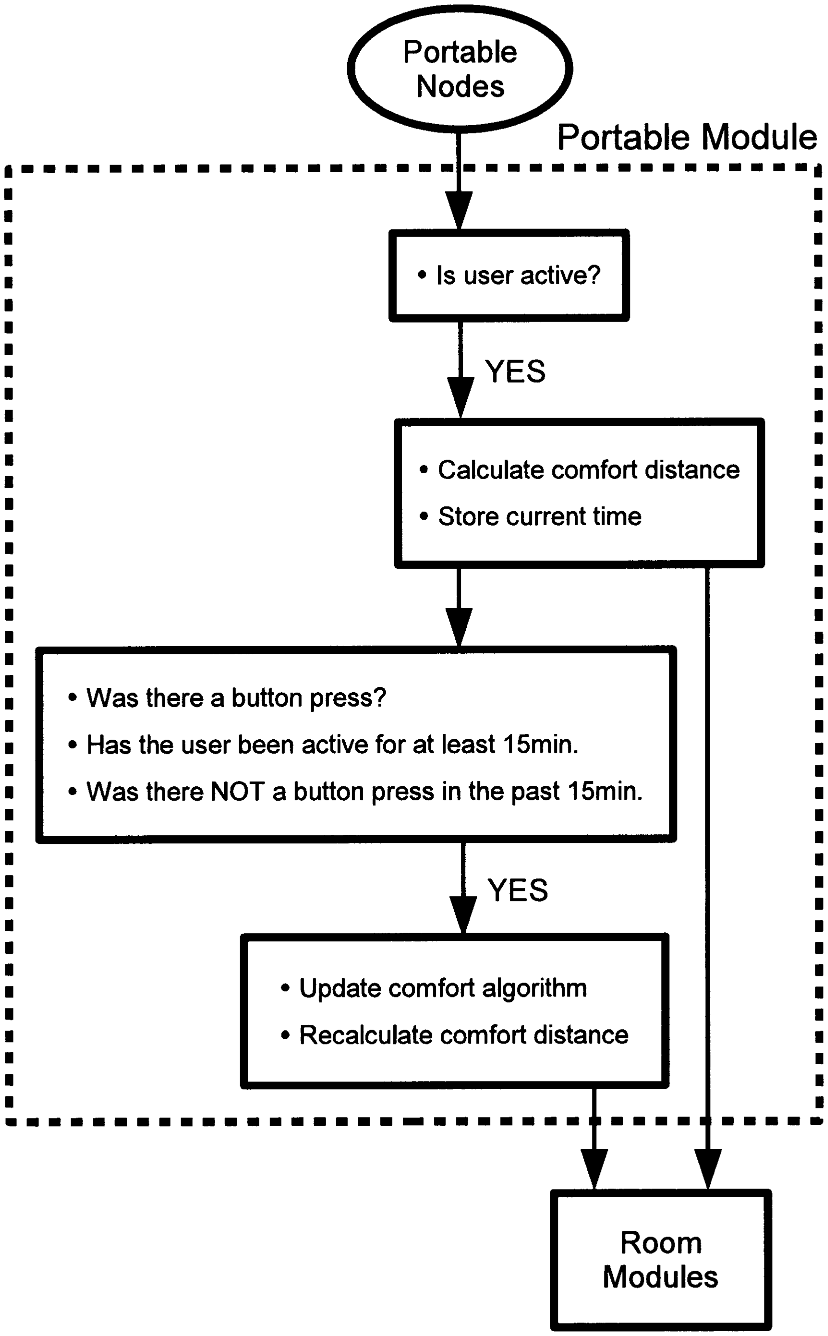

Portable Nodes
Portable Module
• Is user active?
YES
• Calculate comfort distance
• Store current time
• Was there a button press?
• Has the user been active for at least 15min.
• Was there NOT a button press in the past 15min.
YES
• Update comfort algorithm
• Recalculate comfort distance
Room Modules

Figure 4.6.1: Portable Module Flowchart

packets are sometimes sent twice due to the acknowledge transmission failing even though the data packet transmission was successful. Only packets which arrive more than 55 s since the last packet are accepted. Furthermore, for activity recognition, any button press packets are ignored, as these are out of sequence time-wise and result in incorrect activity data. Since the activity data is an accumulation since the last transmission, and button presses can occur at any time, the actual activity is not representative. Although the time since last packet arrival is known, and the actual activity level could be backed out, button presses occur so infrequently that this added complexity is unnecessary. Also, the high forces during the actual button press action make these data of little value.

After this windowing process, the Portable Module checks for activity start or continue conditions. The activity start conditions have much higher values than the activity continue conditions, to help reduce false positives under the assumption that once activity is detected, it is more probable that the user is still active rather than not. This keeps users in the system through periods of low activity, when the activity sensor is returning low values. Start conditions include windowed mean and variance above certain thresholds, and continue conditions include windowed mean and variance and temperature above certain thresholds. There is also a timeout on the continue conditions, excluding temperature. Temperature is excluded to eliminate thermal lock-up conditions. If no continue conditions are detected before the timeout period of six minutes, the system assumes the user is no longer active, and requires that a start condition be detected before re-establishing him or her in the system. The user is only flagged as active if a continue condition is reached, even if the timeout has not yet expired. The time of last activity is then logged for other modules to access in their decision processes. A time stamp is chosen as the pass variable instead of an activity flag, as a user could leave the system while active and leave the activity flag set, which could not be reset as no new user data would be arriving.

A representative plot of the activity algorithm running on received data can be seen in Figure 4.6.2. The windowed mean and variance are plotted alongside the actual activity and temperature data for a sensor worn, via a lanyard, around the neck. Some of the difficulties in detecting the active condition can be seen in Figure 4.6.3, which is a detail of

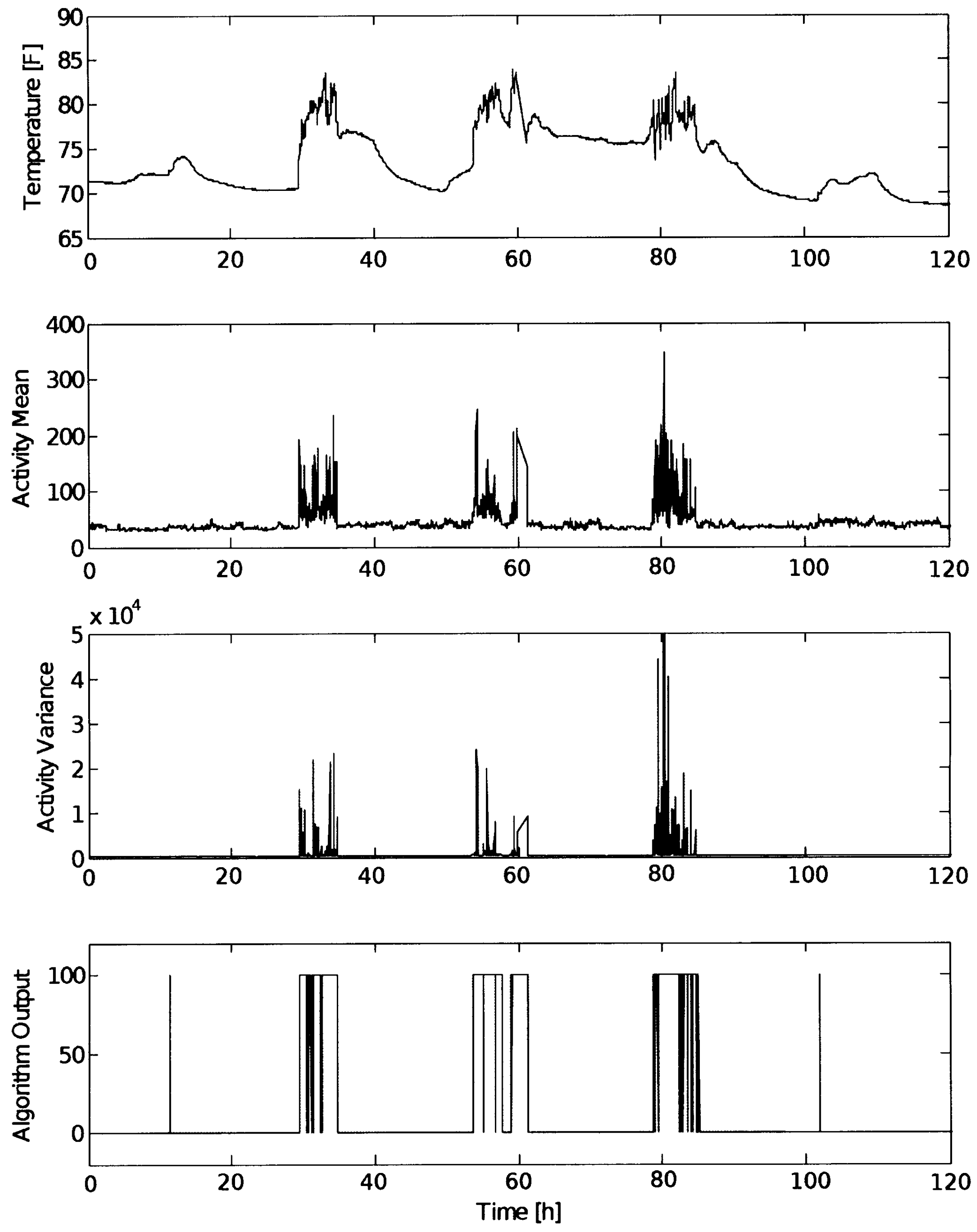


Figure 4.6.2: Representative Activity Algorithm Performance (sensor on lanyard around neck)

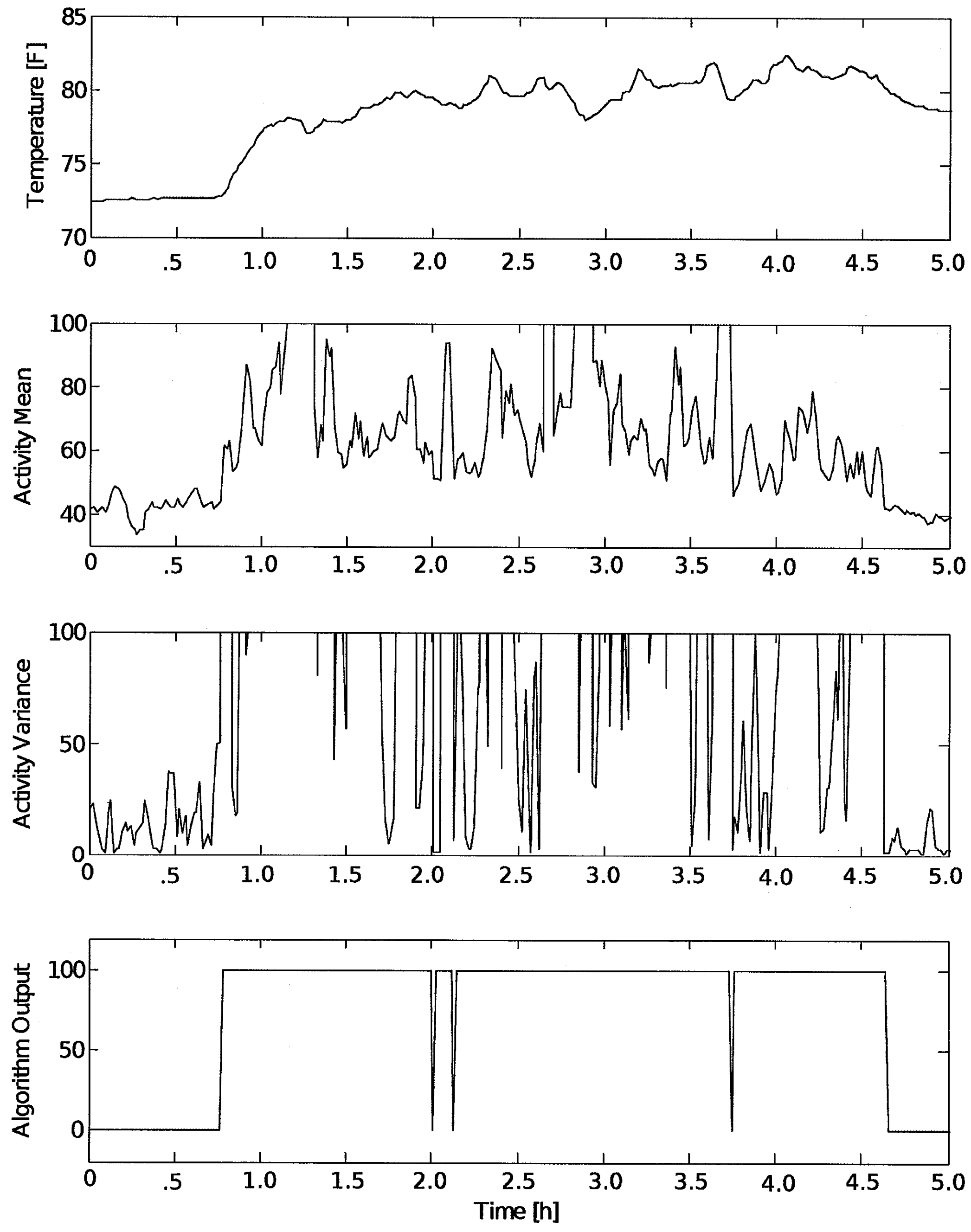


Figure 4.6.3: Representative Activity Algorithm Performance: Detail

a section of Figure 4.6.2. The vertical axes are expanded to show the low levels of many of the signals, as there are many very low activity states during normal usage by the users in the system. Many of the tasks the users in this study perform on a daily basis include reading and thinking, neither of which require much physical motion. As each portable node, and the user of the node, is slightly different, the thresholds of activity need to be different as well. These thresholds are picked manually for the users from visual evaluation of six weeks of user data. As the data is unlabeled, estimations are made of the actual active times, and the algorithms are modified to match these estimates as best as possible. A listing of the thresholds used can be seen in Table 4.6.1.

Portable Node	Mean Start	Variation Start	Mean Continue	Variation Continue	Temperature Continue
4	70	200	45	15	80
8	120	200	70	25	80
20	70	200	50	25	80
24	70	200	45	15	80
28	50	200	20	15	80
76	70	200	30	15	80
84	100	200	65	25	80
88	70	200	30	15	80
96	70	200	50	20	80
104	70	200	30	15	80

Table 4.6.1: Threshold Settings for Portable Node Activity Algorithms

After the activity is sensed, a timeout is placed on any use of the data for fifteen minutes. This is done to help alleviate erroneous control signals due to the temperature and humidity sensors not having acclimated. A plot of a portable node being taken on and off the body can be seen in Figure 4.6.4. The rise time varies, as the temperature it is trying to reach is also varying, as can be seen clearly at the end of the plot. The fall time, however, gives a clear indication of the thermal time constant, which is approximately 15 minutes to achieve 90 percent of final value. It would be more stable for the system to wait even longer, but this would also reduce the responsiveness of the system. If a user is not active for more than 20 minutes, the algorithm is restarted and the user must wait another 15 minutes for his or her data to become valid again. This 20 minute window time allows for short excursions out of the system and compensates for short term false negatives in the activity algorithm.

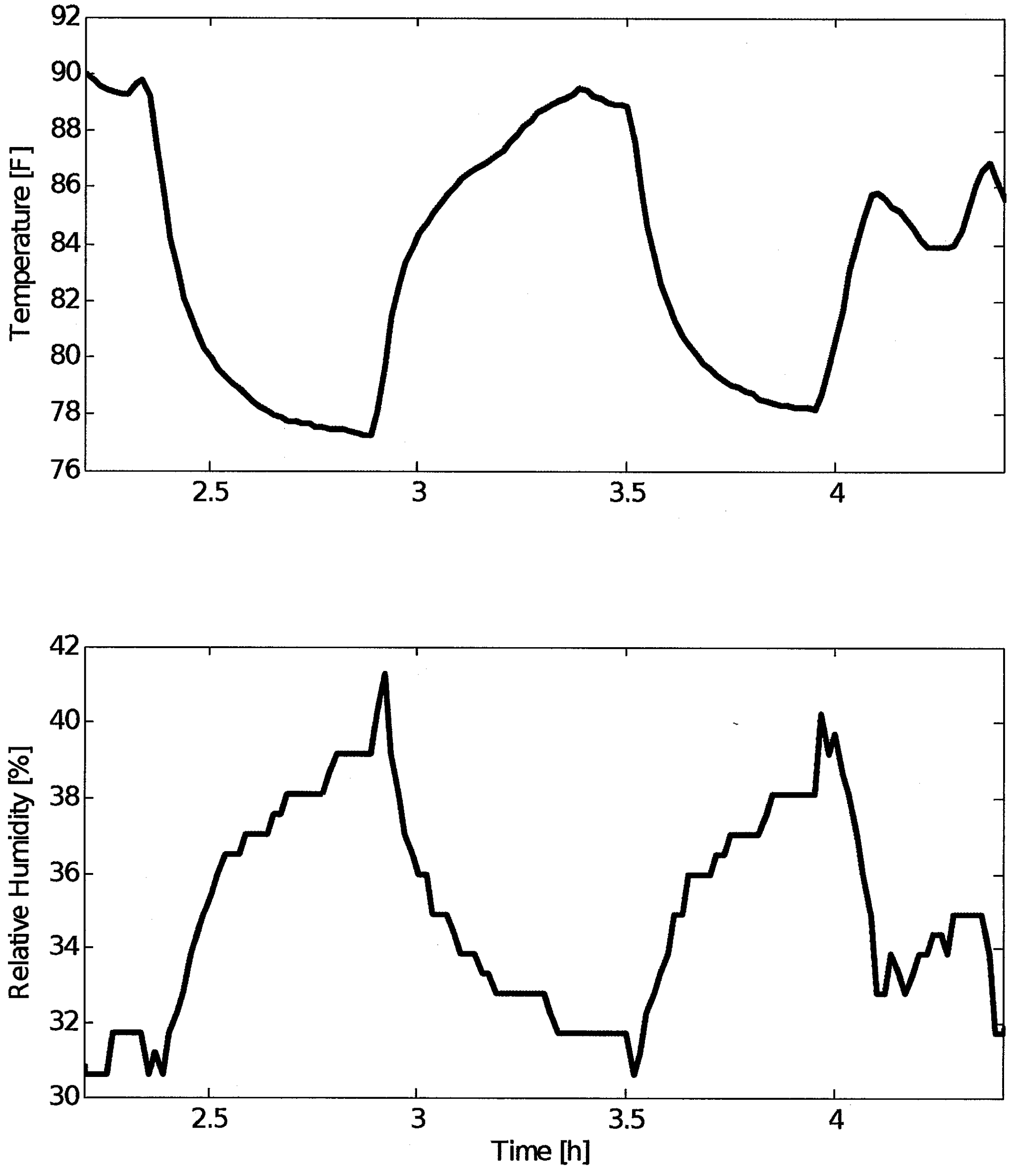


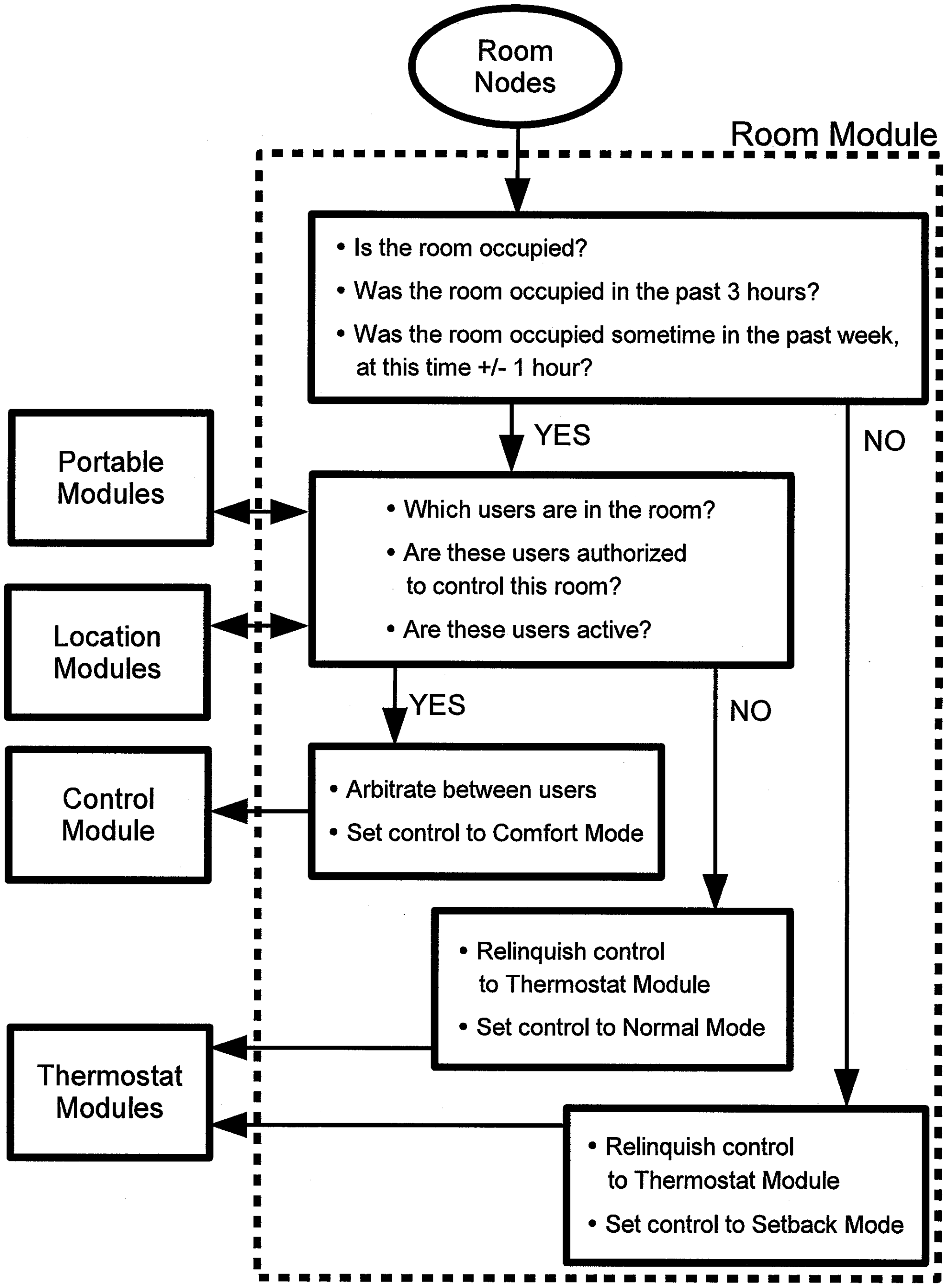
Figure 4.6.4: Temperature and Humidity Acclimation Times of Portable Node (node worn under shirt, and then taken off and placed on desk, twice)

If a button is pressed, these data are processed by the comfort algorithm, which stores the last nine button presses each of hot, cold, and neutral. For the same reasons employed by the activity algorithm, the comfort algorithm only allows button presses from users who have been active for at least 15 minutes. The comfort algorithm is updated and stored in a back-up file. As each subsequent data packet arrives, a comfort distance metric is calculated based upon this updated algorithm, and made available to the Room Module for temperature control calculations. As different comfort control algorithms were evaluated, they are explained in Chapter 5, where an in-depth analysis can be found.

4.7 Room Module

The Room Module is one of the most influential modules in the network, even though it only has one sensor data stream of its own that it uses: the passive infrared (PIR) motion sensor. In addition to its own sensor, the Room Module references many other modules in the network before making its decisions. It determines what format of control the room should be set to (normal, setback, or comfort), and performs the comfort control itself. The reason for this is that the room nodes are fixed in location, and always active, so they can be relied upon to maintain the system state. The room nodes also have 30 s data updates, making them more responsive than the rest of the nodes in the network, which have at least 60 s updates. A flowchart for the Room Module is shown in Figure 4.7.1.

The Room Module first processes its activity information. Although the PIR sensor returns a count which increases with the amount of motion in the room, all levels greater than zero are counted as a level of one. This is done to even out the algorithm results, as only the presence of activity is desired to be known, not the level of activity. The algorithm performs two windowed means on the incoming data, one over the past 70 samples, and the other over the past 12 samples. Two different averaging times are used to drive two different outputs. The first is a determinant of whether the room is occupied, which requires a fast response time. The second is a determinant of the first arrival time of users in a day, which requires high accuracy but response time is irrelevant. Both algorithms use a higher threshold for initiating an occupancy state than they use for maintaining an occupancy state. This limits false positives and excessive oscillation between states throughout the day.



Room Nodes
Room Module
• Is the room occupied?
• Was the room occupied in the past 3 hours?
• Was the room occupied sometime in the past week, at this time +/- 1 hour?
YES
NO
Portable Modules
Location Modules
• Which users are in the room?
• Are these users authorized to control this room?
• Are these users active?
YES
NO
Control Module
• Arbitrate between users
• Set control to Comfort Mode
• Relinquish control to Thermostat Module
• Set control to Normal Mode
Thermostat Modules
• Relinquish control to Thermostat Module
• Set control to Setback Mode

Figure 4.7.1: Room Module Flowchart

An example of the occupancy algorithm running on received data for a week can be seen in Figure 4.7.2. This gives insight into a number of the problems faced, and some of the benefits and shortcomings of the detection system. It can be seen that there are several short periods of occupancy, especially late in the evening when custodial staff enter for cleaning. The algorithm generally windows these out, although periods lasting more than four minutes are usually detected as an occupancy. There are also multiple gaps in occupancy throughout the day, which the algorithm bypasses with a three hour time-out, but this also gives longer periods of false positives after the user leaves for the day.

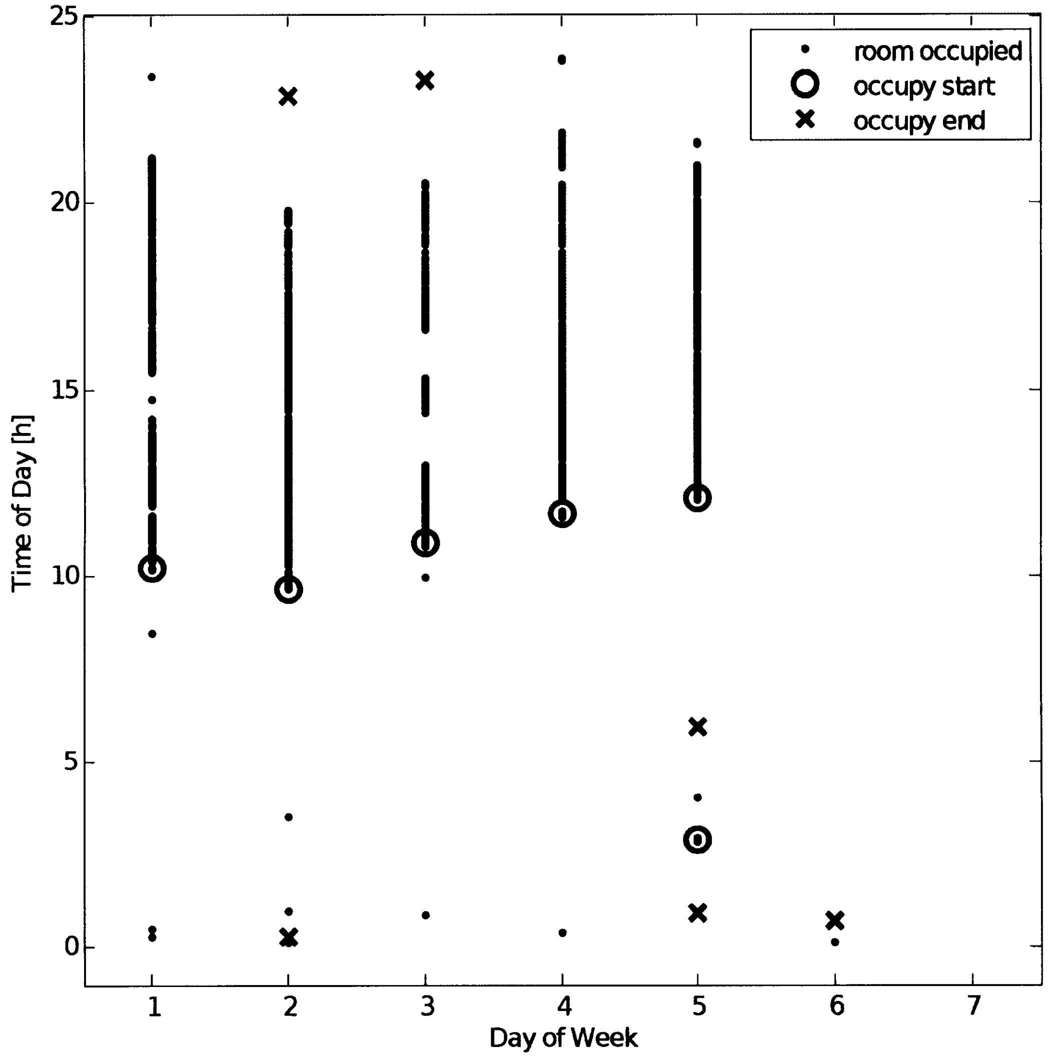


Figure 4.7.2: Example Occupancy and Algorithm Results for a Single Room

The main role of the occupancy algorithm is to determine what level of cooling the room needs. For this reason, a large amount of lag is placed in the transitions, in order to keep the air-conditioning system from cycling too heavily and wasting energy. Once occupancy is detected, the system maintains an occupied state for three hours. This was done to bridge large gaps in the day when users leave to go to lunch or meetings. These happen quite frequently, and a three hour window covers most scenarios. This does, however, place a high penalty on false positives, as it will cool a room excessively for three hours. The window averaging time of 12 samples, which equates to six minutes, attempts to strike a balance between fast activation and accuracy. This gives an average delay of three minutes for new occupant entries. Although there is a long time window for this global occupancy variable, the system also keeps track of immediate occupancy for use in determining what cooling actions are required.

If no occupancy is detected, the system checks if a user is expected to arrive soon, in order to ensure that the room is at an appropriate temperature when he or she enters. This is done by checking the arrival times stored in a buffer over the past week. If a user arrived any time in the past week within two hours of the current time, the system assumes he or she will most likely arrive again, and it prepares the room accordingly. The window time is much larger for this algorithm, with an average lag of 17.5 minutes, to ensure that any person who entered the room in the past week actually stayed long enough to make it worth cooling the room down for him or her. This algorithm also employs a three hour time-out, in order to bridge gaps in the day, and return only one entry point for the day. If a room is unoccupied for more than three hours, the next entry will be stored in a buffer for future reference.

Once the Room Module has calculated its occupancy state, it then performs control actions. These control actions are performed at one minute intervals, even though data is updated at 30 s intervals. This is done to synchronize with the other controllers which are limited to one minute updates. Doubling the control action time would be the equivalent of doubling all the PI control gains, causing system instability. If no occupancy is detected, and it is not within two hours of an expected entry, the Room Module sets the control state to setback and performs no actions. If occupancy is detected, or expected to occur in the next two hours, but there are no users wearing portable nodes in the room, the Room Module sets the state to normal and performs no control actions. Under both setback and normal states, the Thermostat Module controls the VAV dampers to regulate the room temperature. If the room has been occupied by a user with a portable node in the past three minutes, the Room Module checks the comfort level of all users present in the room and averages them to create a control scalar. It then turns off both normal and setback control, and acts upon the control scalar to minimize the average room discomfort. Once users have been detected there is a six minute timeout before either setback or normal mode can be entered, in order to maintain system state during periods of false location responses and short trips out of the room.

When determining the control scalar for a room, the Room Module only considers those occupants who are normally situated in the environment. The test setting incorporates four closed offices, and one large common space which is divided into two sections. For the four offices, only the occupants of those offices are allowed to control the comfort setting. In the large common space, all users are averaged together when determining the setpoint. Although room occupant information was pre-assigned in this case, it could be replaced by a weighting algorithm that determines how much time each user spends in a particular space. This would produce similar results as the pre-assigned knowledge, as office occupants would make up the highest percentage for their respective offices, but it would also increase the amount of control workers in the public space had over their area.

Chapter 5

Comfort Algorithm Design

In order for the entire Personalized Comfort Control System to function effectively, it must have some metric by which to judge the individual user's 'comfort distance'. The term 'comfort distance' is used here as a measure of how hot or cold a user is, or more precisely, how far away he or she is from being comfortable. The use of this metric places a number of difficult constraints on what types of algorithms can be used, but is required due to the fact that the output of the implemented algorithm must drive a control loop. For this reason, it must have a monotonic structure to avoid instabilities and local minima. These could be accounted for through a non-linear control structure, but this is avoided in this work in order to minimize the number of variables involved in determining system stability and performance.

Many standard pattern recognition techniques are inadequate for this control task as they seek to draw boundaries around similarly labeled data, giving accurate classification, but no distinction of levels within those classes. A simple Bayesian analysis could give a probability of comfort, but would require a much more accurate model than is currently available, given the limited set of on-body comfort indices used. Only temperature and humidity are measured on-body, whereas Fanger's model [38] requires clothing level, metabolic rate, and air flow. This problem is compounded by the limited labeling of the acquired data. In comparison to Fanger's seven point scale, users of this system only have three choices, hot, cold, or neutral. This means there is no way of knowing exactly how hot or cold they are at the instant a button is pressed. This metric must be inferred from the distribution of the received data.

Not only are the labeled data points ambiguous as to their level of discomfort, but they are also very sparse in their occurrence. The users are not required to press any buttons, and are only asked to do so if they feel uncomfortable, limiting the amount of labeled data points to an average of about one per person per day. Ideally, if the system were to function flawlessly, this number would be even lower, as the users would be comfortable more often. Another issue faced in this reduced data set is the lack of an even distribution of hot and cold labels. For some users, the room never went cold enough to make them feel cold, so only hot data points exist, giving no information by which to determine a lower limit on comfort.

All of these drawbacks in the data set place a final constraint on the comfort algorithm. It must be robust to insufficient, inaccurate, and indeterminate data. If one user has his or her sensor in a pocket, where the temperature is much warmer than its usual location, the entire system can not be allowed to drift too cold in order to compensate. There must be the inclusion of some pre-assigned knowledge which incorporates a rational view of comfort boundaries. This requirement will therefore favor more general approaches, which may not be as effective for each individual, but will reduce the problems associated with over-fitting of the data (e.g. the system attempting to cool a room to match an accidental button press). Finally, any algorithm developed must be able to run and update itself in real-time, to effect the comfort of the users when it is needed.

5.1 Selection of Indices

At any given instant, several features are available from which to determine comfort. Examples of these include on-body environmental conditions, room environmental conditions, outdoor environmental conditions, location, time of day, day of week, and local VAV air flow. Even without the creation of any subfeatures, the system already logs 24 different data indices each minute, for each user in the system. Given the extremely high dimensionality of the data set in comparison to the number of labeled data points (an average of one per person per day), the number of indices used to determine comfort must be kept to a minimum. Any algorithm trained on a small data set, with too many features, will classify those particular points well, but is unlikely to be representative of the function as a whole. A typical metric is to have a training set much, much larger than the feature set. This limits the scope for these data to two or three features.

From extensive evaluation of the first two months of data, little correlation is found between the time series data and hot or cold preferences. Although users were more likely to register discomfort early in the day, and earlier in the experiment, there was no distinction between hot or cold. Similarly, the outdoor environmental data gave mixed results, with the strongest correlation coming from a user who did not have a view of a window from his workspace. This is probably due to the weather being extremely consistent for the first two months of the study: cold and wet. It rained in Boston on all but two days in the month of June. The VAV damper information proved useful for determining some of the users' comfort, with them reporting either comfortable or cold conditions when it was on, and hot or neutral conditions with it off. However, since this is also the main method the system has for affecting the comfort of the users, it is not used as a control parameter in order to prevent oscillations which would occur with the system detecting a 'hot' state, initiating air flow, detecting a 'cold' state, turning off, and repeating this cycle indefinitely.

The main parameters that returned consistent results amongst all users, were the room and on-body environmental conditions. Scatter plots for the majority of users comparing temperature and humidity for the various comfort conditions can be seen in Figures 5.1.1 – 5.1.7. The wall mounted thermostat nodes provided the best overall clustering of classes, with the on-body portable nodes and ceiling mounted room nodes performing similarly. Linear combinations of these data were also evaluated, which showed mixed results, giving large improvements to some and greatly hindering others.

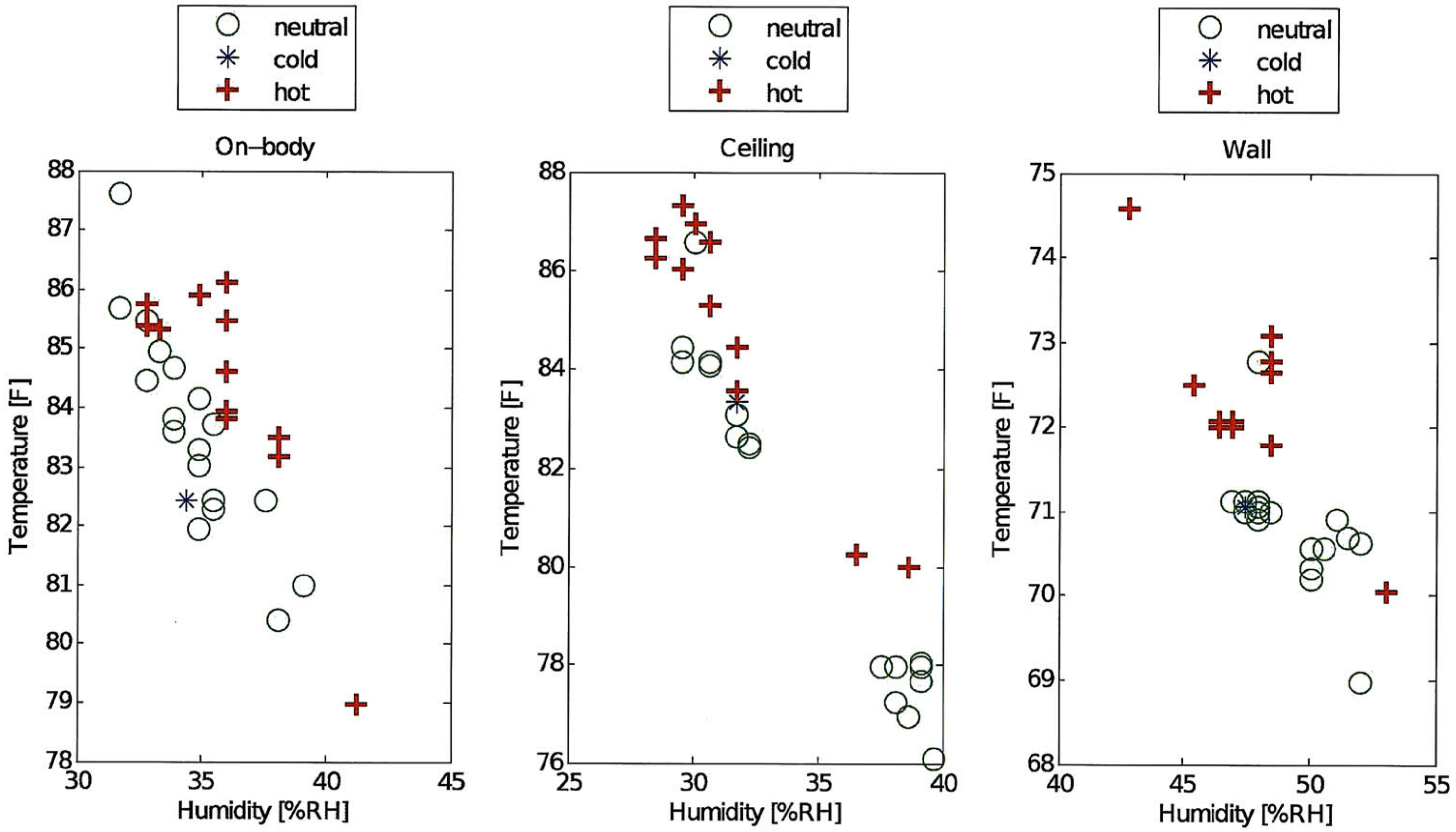


Figure 5.1.1: Clustering Comparison for Various Sensor Locations: Node 4

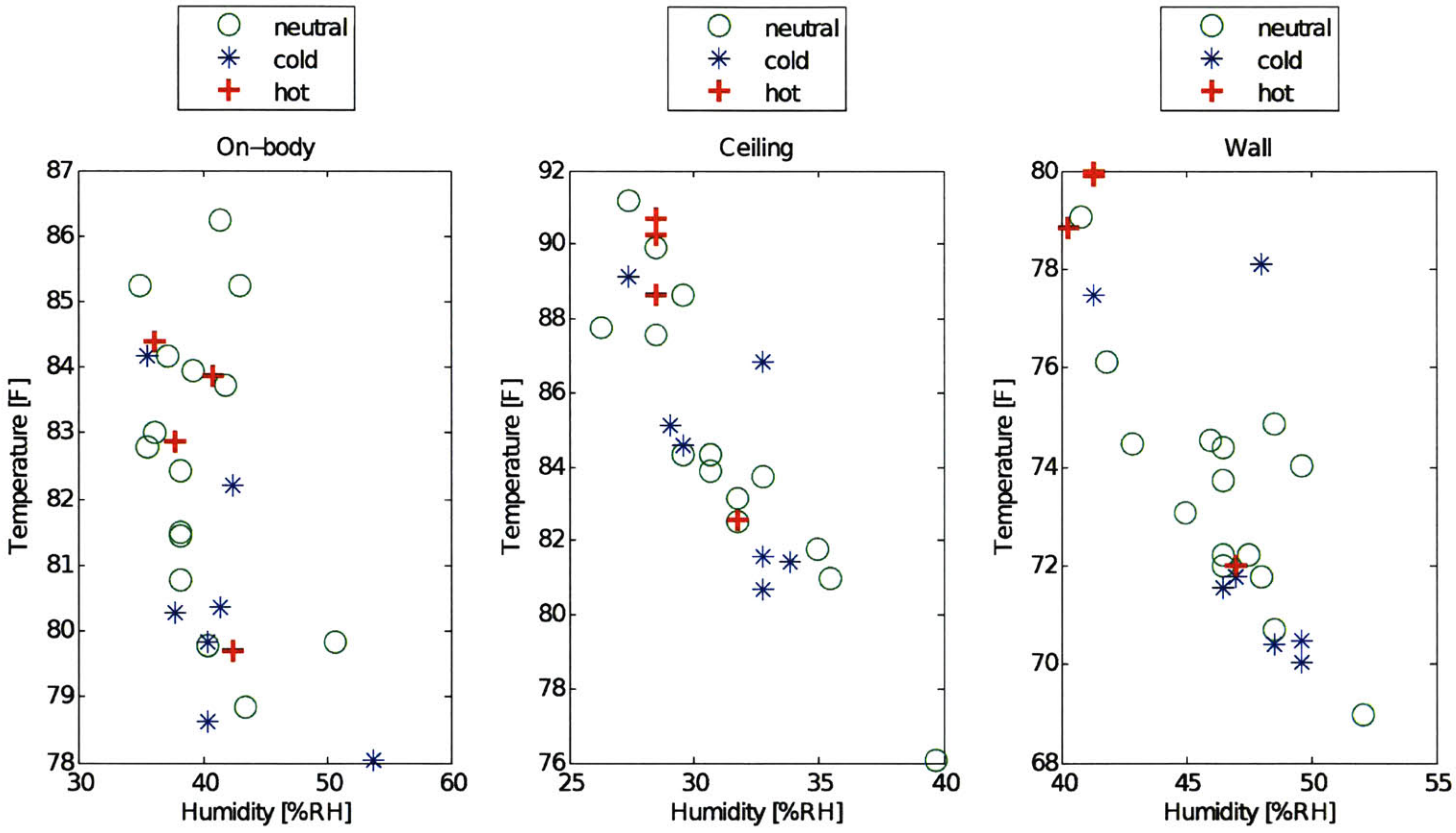


Figure 5.1.2: Clustering Comparison for Various Sensor Locations: Node 8

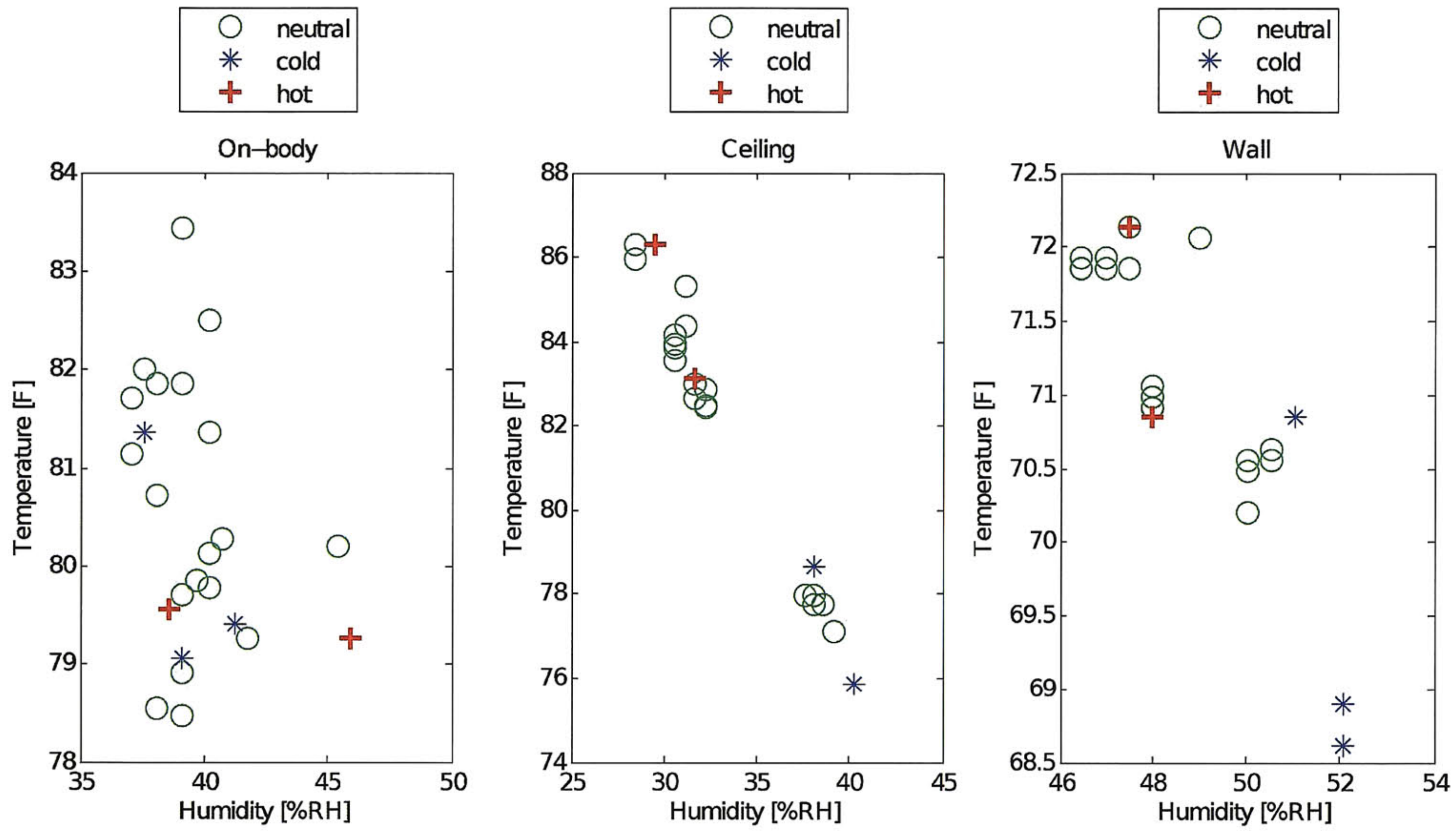


Figure 5.1.3: Clustering Comparison for Various Sensor Locations: Node 20

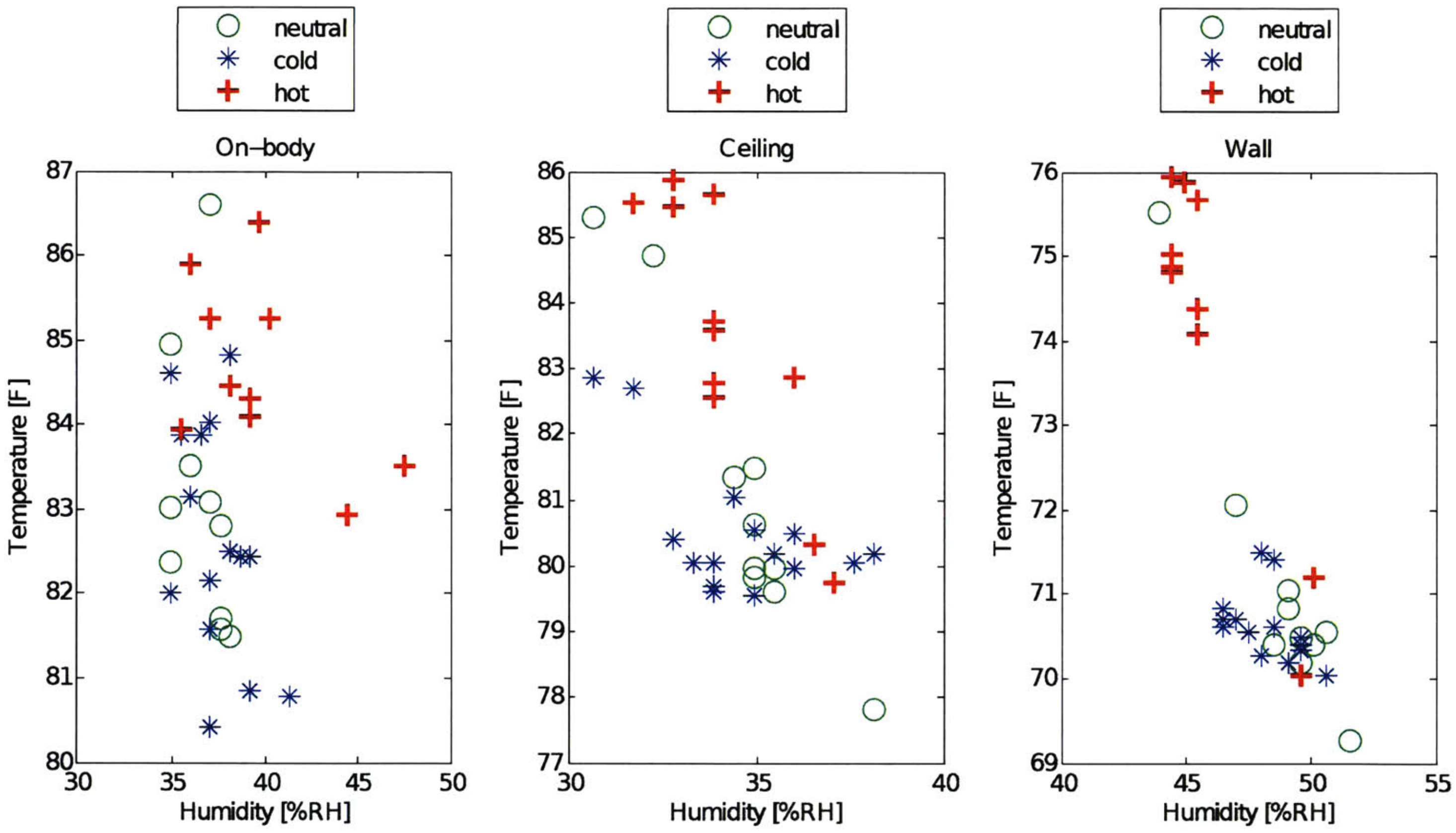


Figure 5.1.4: Clustering Comparison for Various Sensor Locations: Node 24

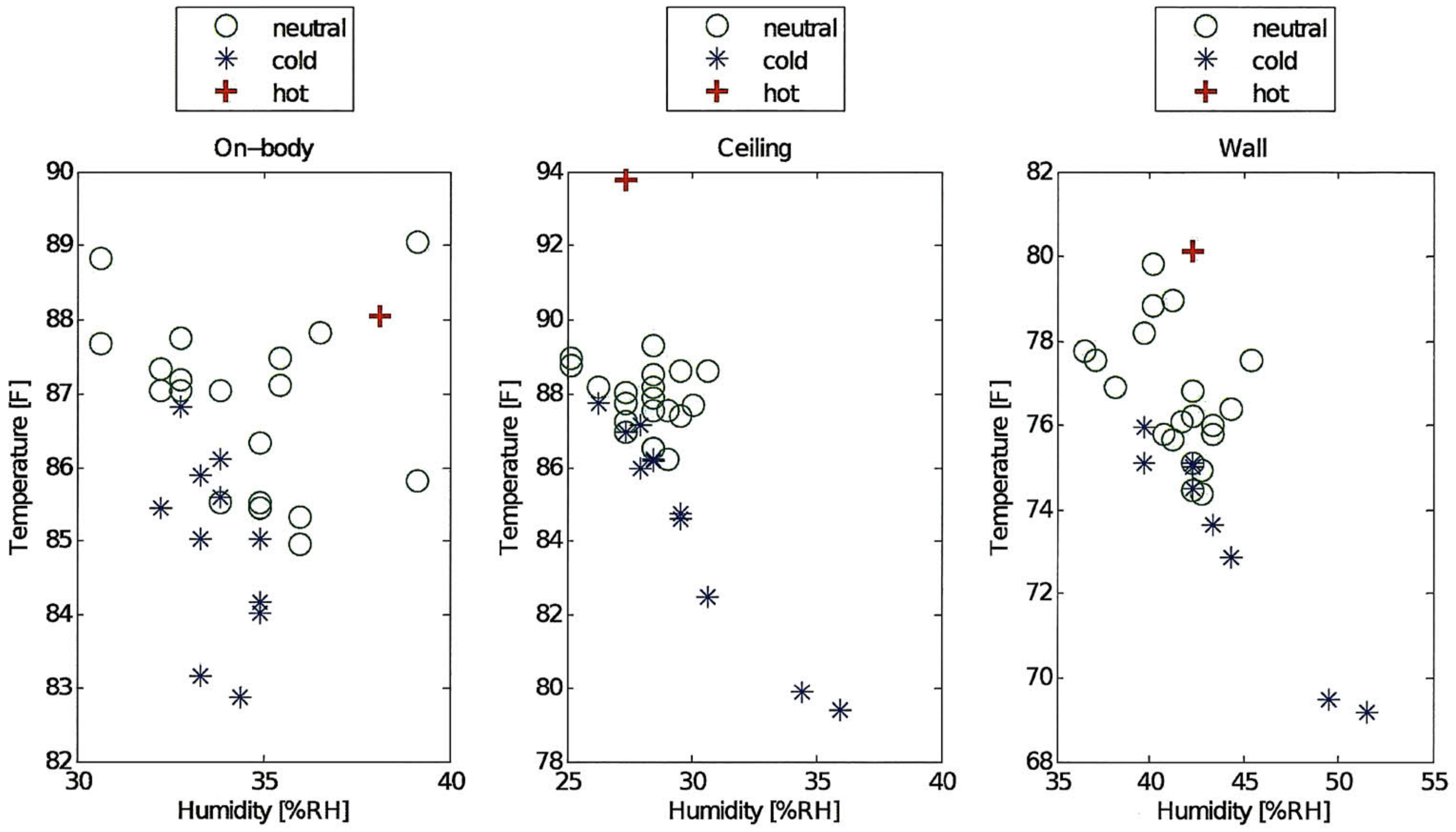


Figure 5.1.5: Clustering Comparison for Various Sensor Locations: Node 72

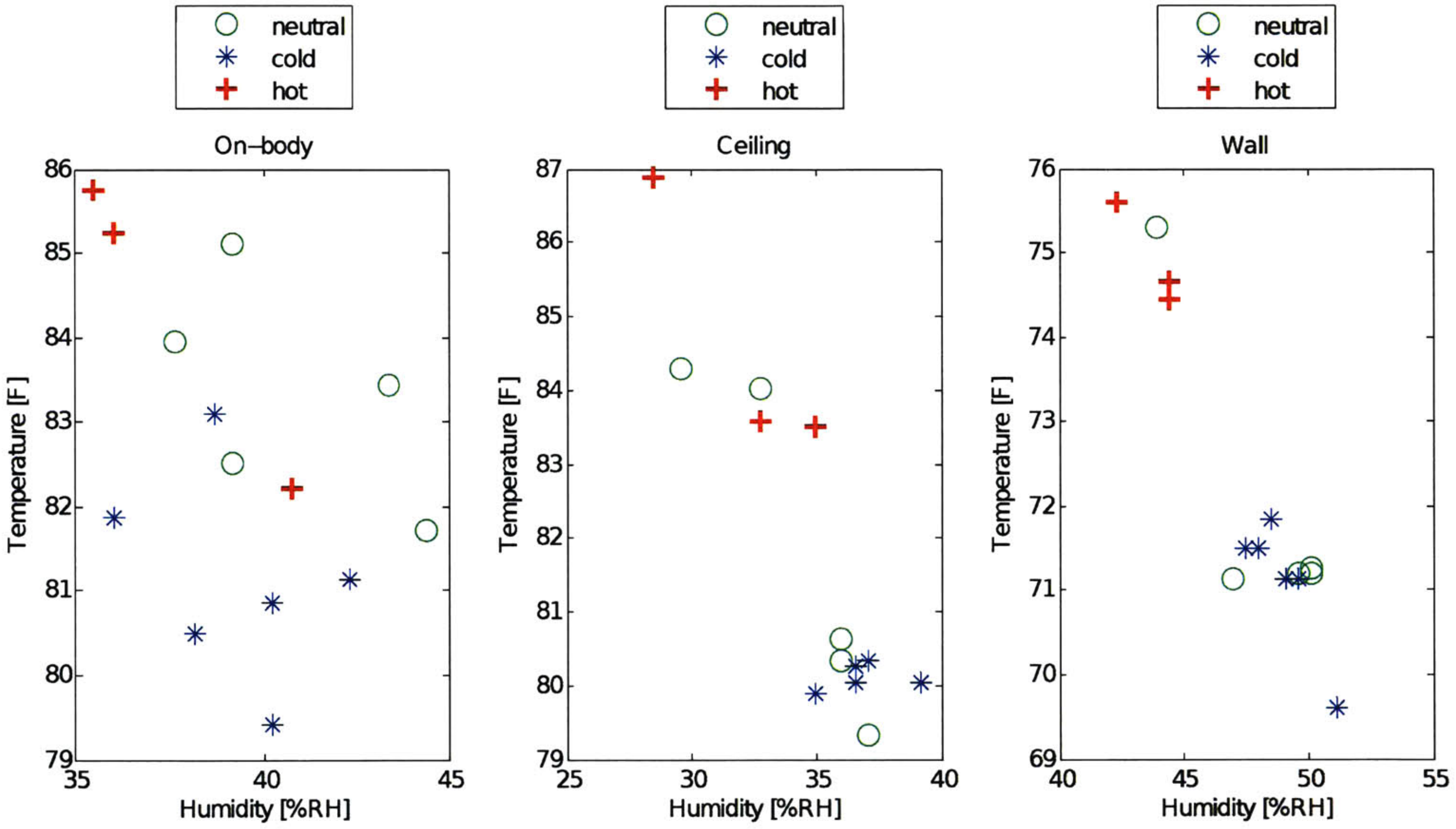


Figure 5.1.6: Clustering Comparison for Various Sensor Locations: Node 84

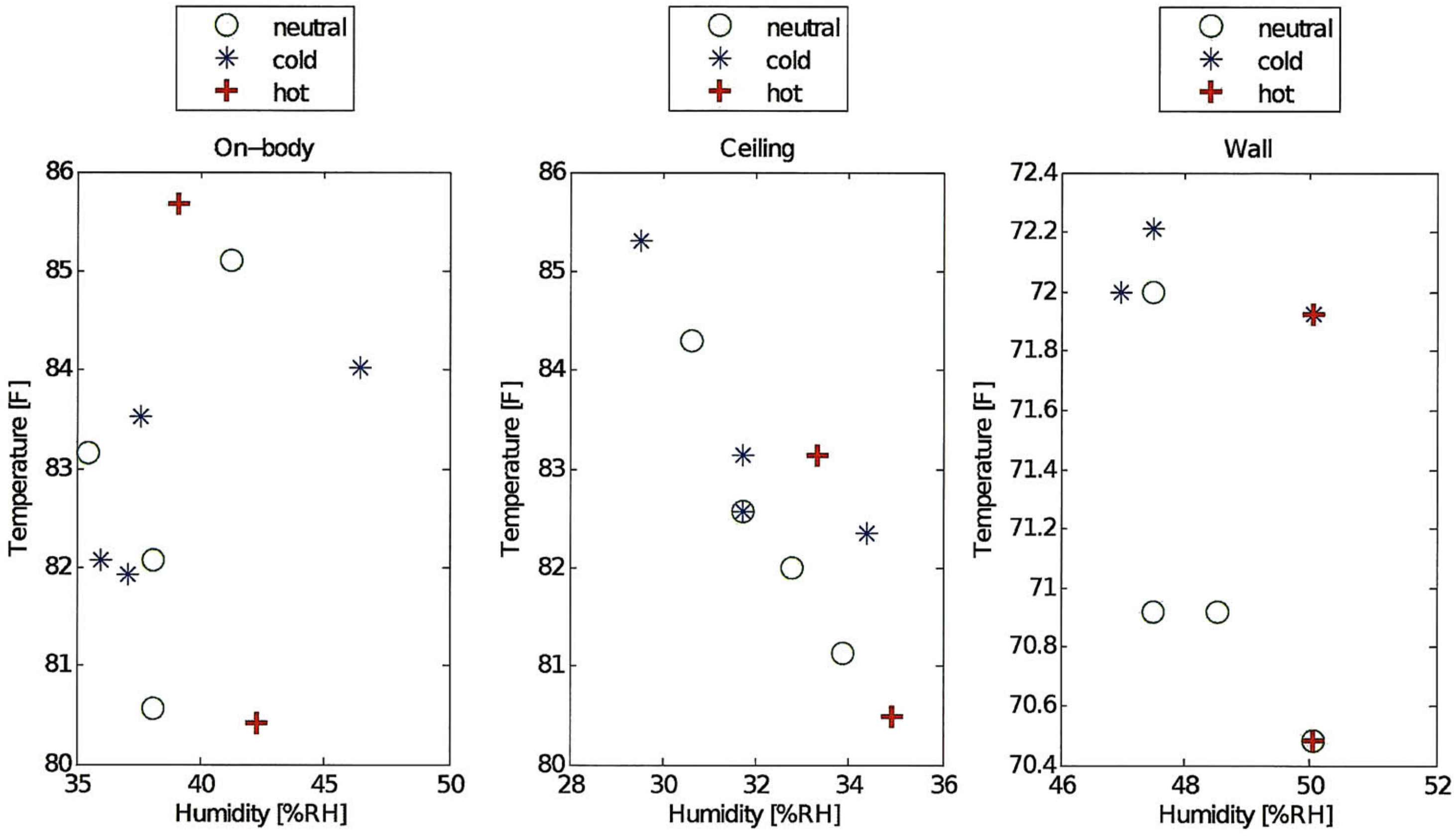


Figure 5.1.7: Clustering Comparison for Various Sensor Locations: Node 88

Ultimately, as one of the main objectives of this thesis is to evaluate the effectiveness of controlling an HVAC system via on-body sensing, it was decided that the clustering is adequate to exclusively use the portable node's temperature and humidity data, even though these are not the strongest classifiers. This will give a clear indication as to the limits of this format of comfort control. The on-body light and activity data are excluded from the comfort algorithm, as they give extremely poor separation of the classes. For the most part, the activity and light levels of the user are high, with large variance, whenever the node is being worn, giving very little difference from day to day, or hour to hour.

5.2 KNN Distance Metric

Based upon initial success with a KNN comfort algorithm (see Appendix A), a modified KNN Distance Metric was developed. Since the original algorithm merely returned a class label, a new method was needed to determine the level of discomfort. A KNN approach is well suited for distance measurement, as this is an intrinsic aspect of locating nearest neighbors. In this case, 'comfort distance' is measured as the distance away from a known labeled point, and an average of these comfort distances is used to help reduce the effects of outliers.

Although it is possible to accurately measure the Cartesian distance between any two data points, it is unclear exactly how this distance relates to comfort. For this reason, outside knowledge is required to help train the algorithm. It has been clearly shown from comfort research and popular experience that increasing temperature and humidity levels lead to increases in heated thermal discomfort. Fanger's PMV model gives explicit equations for these relationships, but this equation is also a function of many other variables which are not known in this case. To determine the appropriate weightings of temperature and humidity for this situation, the slopes of the 'comfort lines' are measured from the received data from the portable nodes. These comfort lines are the boundaries between the hot and cold labeled data points, essentially enforcing a comfort metric based upon the orthogonal distance to this line of comfort.

This particular method has two distinct advantages. First, it incorporates expert knowledge of the system, allowing the system to respond as expected, and constraining the output to apply negative feedback to the control system. It is important that the decreases in temperature always result in decreases in heated discomfort, as this is the only method by which the VAV damper motors can affect thermal sensation. Furthermore, this comfort line can be determined off-line by visual inspection, which can achieve closer fits due to the researcher's better understanding of the difference between representative data and outliers than the computer's. Secondly, the KNN approach can easily incorporate the neutral class into its determination, giving a better average of how the user might be feeling at a particular point. This also allows for morphing of the comfort space to match an individual's experience, as local clusters can pull away from the enforced distance metric, if reinforcement is high enough.

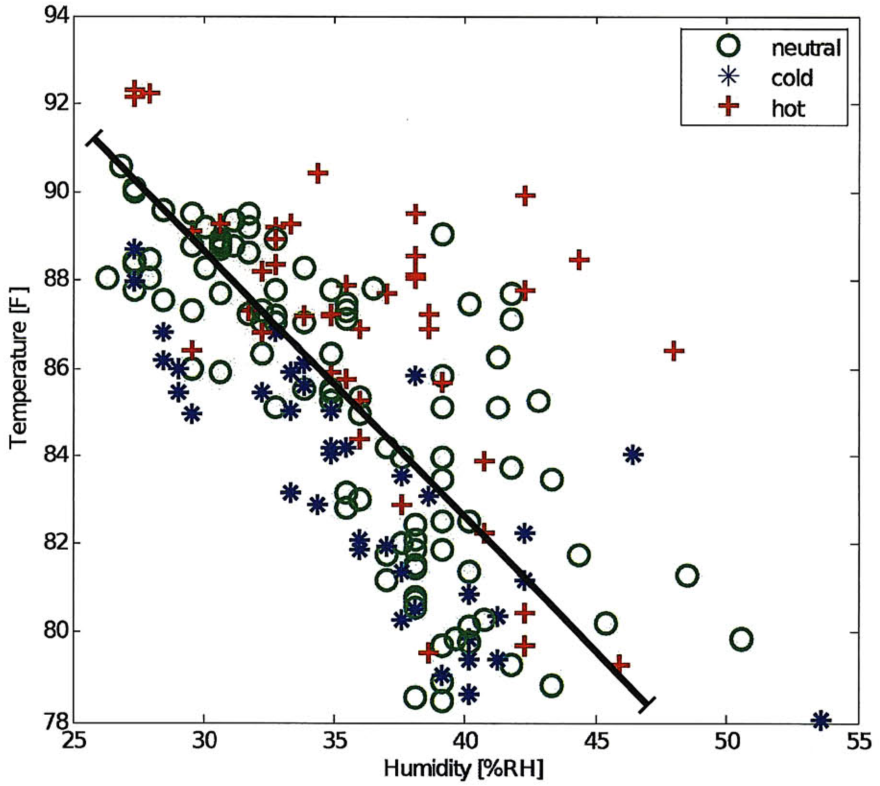


Figure 5.2.1: Comfort Boundary Superimposed on User Preferences: Nodes 8, 20, 72, 84, and 88

To determine the comfort line, an average of all comfort boundaries is used. A plot of this comfort boundary can be seen in Figure 5.2.1, superimposed upon the comfort labels from half of the user population. A linear boundary is assumed to limit the ambiguity of the weightings of temperature and humidity at a given point, by restricting it to a constant for the entire space. An average is also taken to make the system more representative of the preferences of the entire group. Individual comfort slopes could be used for each user, but these slopes are based on relatively sparse data, and it was decided that an average will suffer less from the problem of over-fitting. The average comfort line derived from these evaluations gives a ratio of temperature to humidity of approximately $-3/5$ (e.g. an increase in temperature of 3 °F could be balanced out by a decrease in relative humidity of 5%).

The form of the KNN distance algorithm is shown in Equation 5.2.1, where D_d is the level of thermal discomfort at point d, for n nearest neighbors. T is the temperature in degrees Fahrenheit, RH is the relative humidity in percent, and $G_T/G_{RH} = -3/5$ is the temperature to humidity ratio. C_i is the class label of the i^{th} neighbor, with -1 representing cold, 0 representing neutral, and $+1$ representing hot. The relative weighting of the enforced linear distance metric is set by G, with lower values of G allowing the algorithm to adapt to local data more readily, at the cost of greater overall coherence.

$$D_d = \frac{1}{n}\sum_{i=1}^{n}(\frac{T_d - T_i}{G_T} - \frac{RH_d - RH_i}{G_{RH}}) \times G + C_i \tag{5.2.1}$$

Representative results for the KNN Distance Metric can be seen in Figures 5.2.2 – 5.2.13, with the darkness of the red sections indicating the level of hot discomfort, and the darkness of the blue sections representing the level of cold discomfort. As there are very few samples in the data sets, each value of n was tried, and the best visual fit was selected. A range of G values is included to show the influence of the enforced linear boundary. It can be seen that a standard KNN would produce extremely non-linear control inputs, and that the linearization process is both necessary and effective. Unfortunately, despite the heavy linearization, there still exist islands of inconsistency in the results. This is even more problematic considering that the data sets included here represent the relatively more coherent ones. It is also unclear as to the benefit of including the neutral category. Although this helps in cases where the data sets are not well populated, there often exist clusters of neutrals near clusters of hots, which reduces the likelihood of that particular area being labeled as 'hot'. Although this ambiguity necessarily arises from the nature of the data, it would perhaps be a more conservative approach to consider any labeling of a point as hot to be indicative of conditions to be avoided.

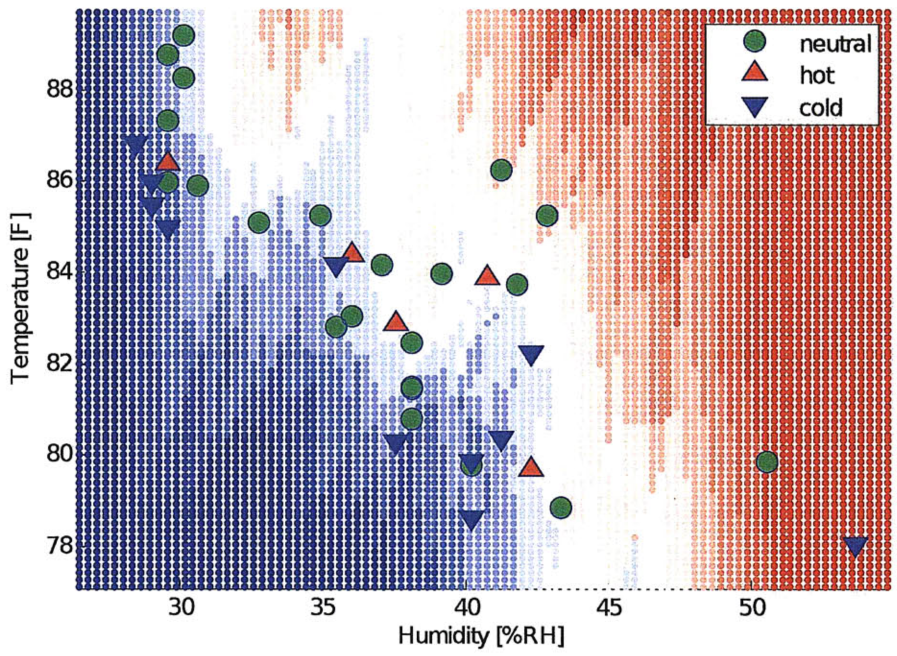


Figure 5.2.2: KNN Distance Metric, $n = 10$, $G = 0$: Node 8

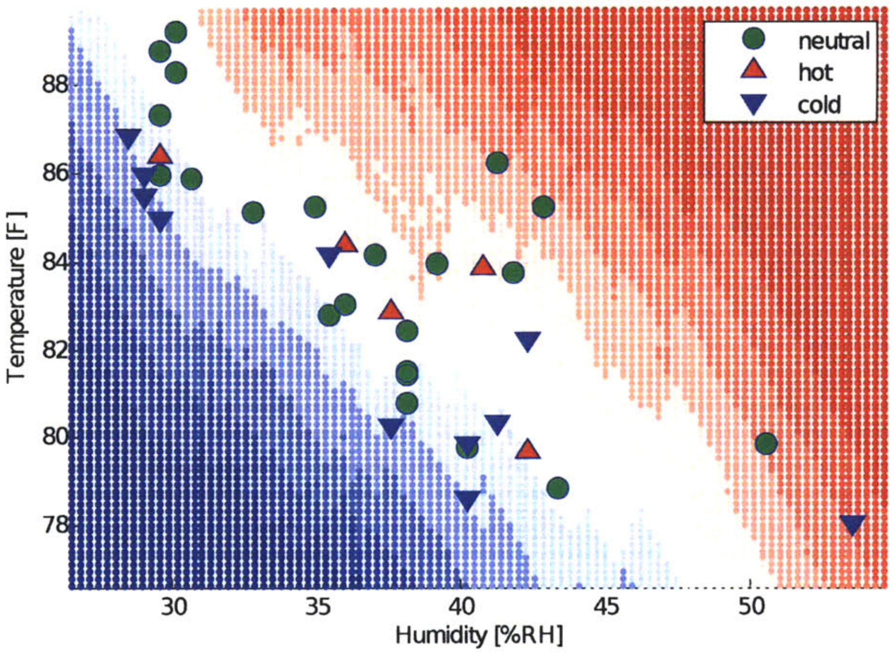


Figure 5.2.3: KNN Distance Metric, $n = 10$, $G = .5$: Node 8

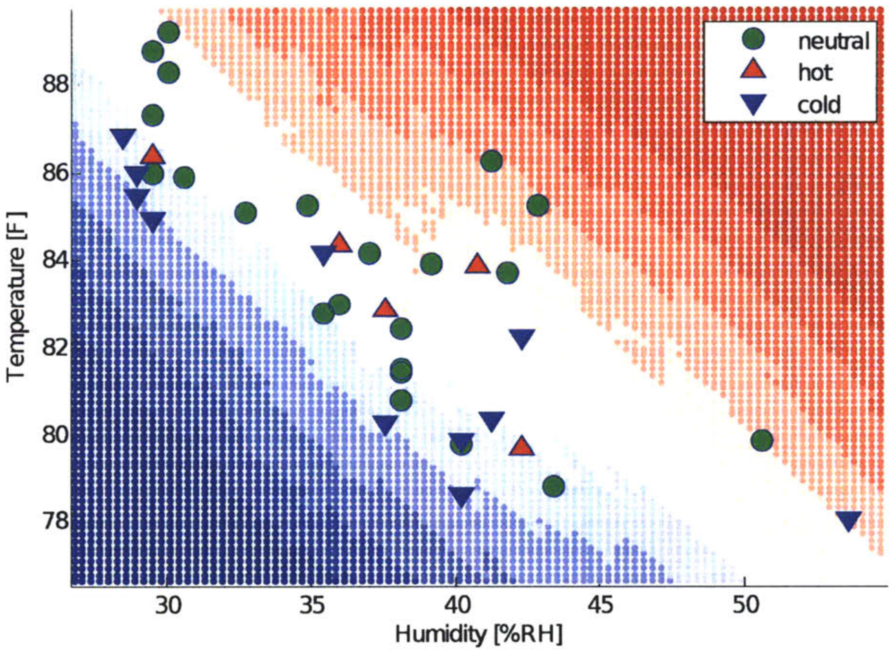


Figure 5.2.4: KNN Distance Metric, $n = 10$, $G = 1$: Node 8

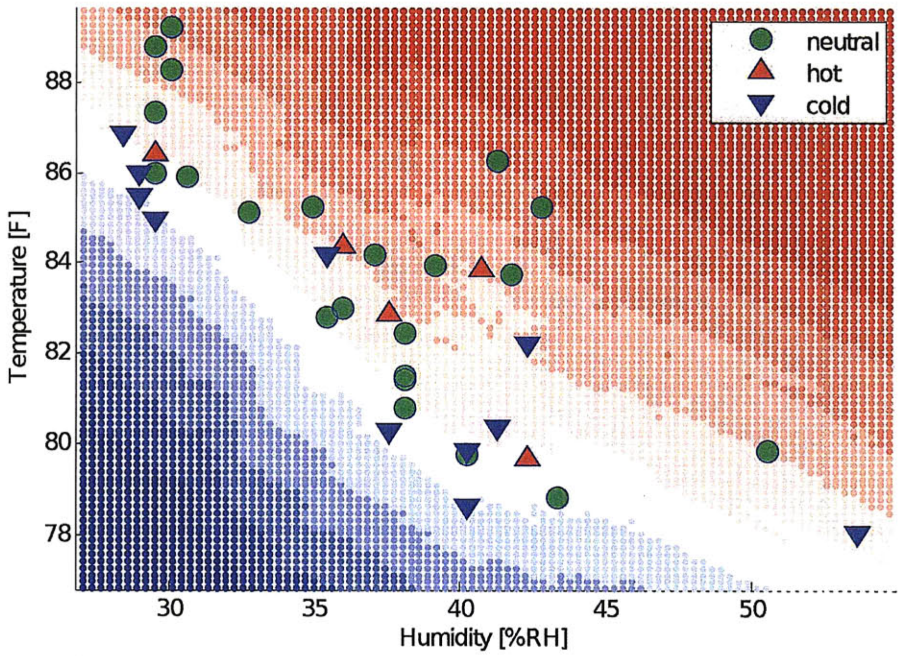


Figure 5.2.5: KNN Distance Metric, $n = 10$, $G = 2$: Node 8

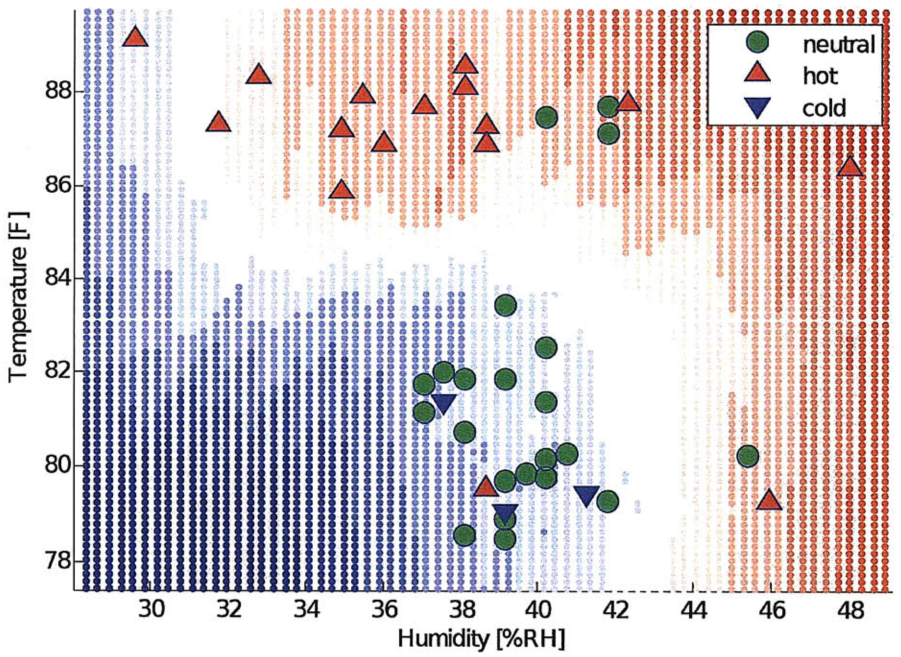


Figure 5.2.6: KNN Distance Metric, $n = 10$, $G = 0$: Node 20

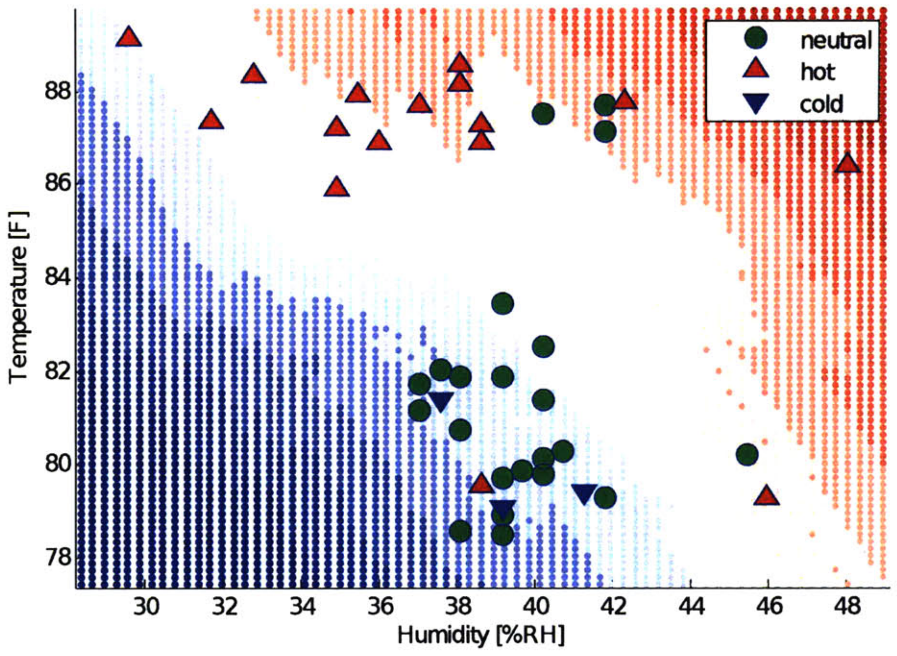


Figure 5.2.7: KNN Distance Metric, $n = 10$, $G = .5$: Node 20

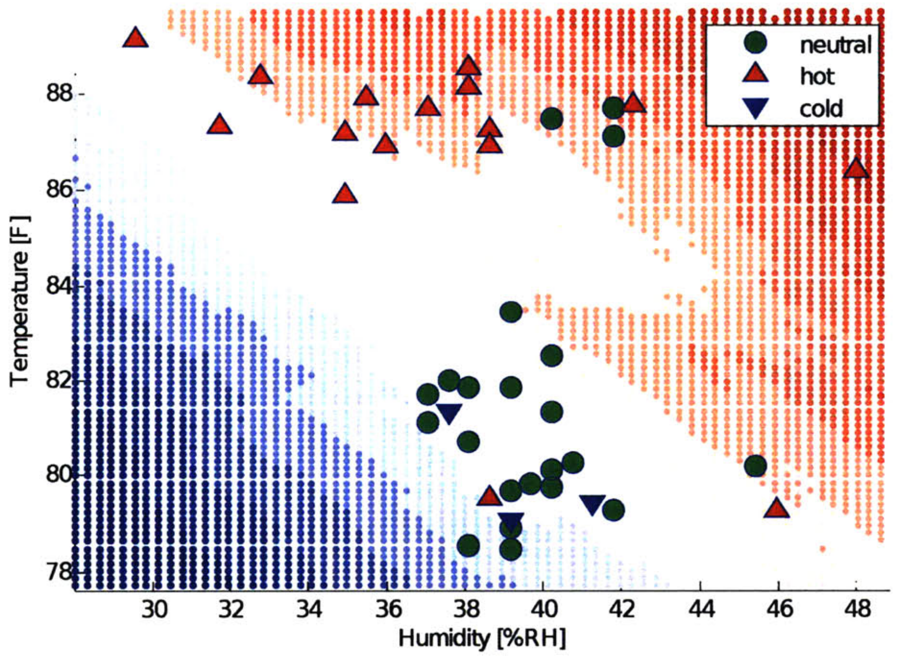


Figure 5.2.8: KNN Distance Metric, $n = 10$, $G = 1$: Node 20

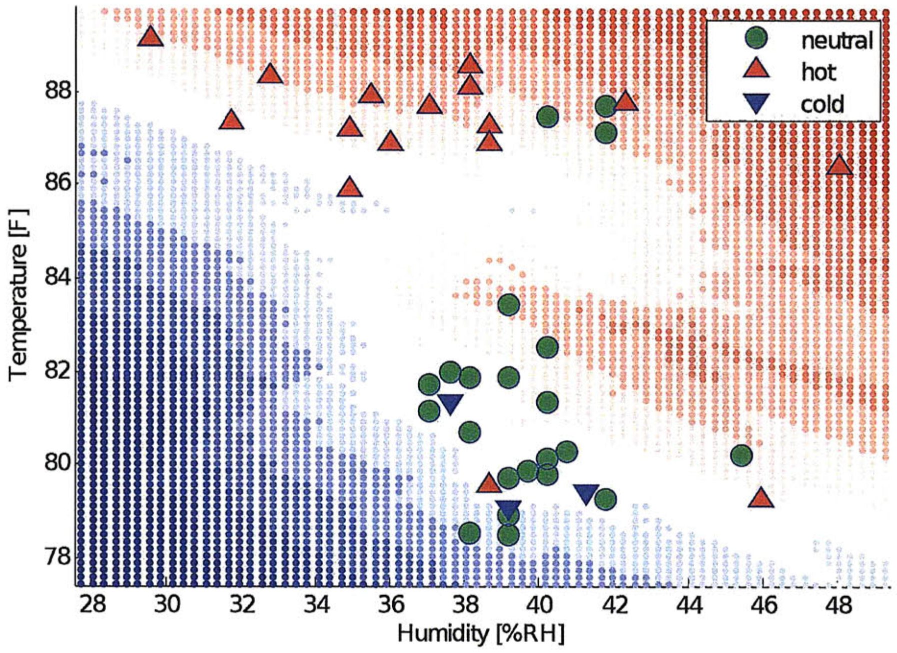


Figure 5.2.9: KNN Distance Metric, $n = 10$, $G = 2$: Node 20

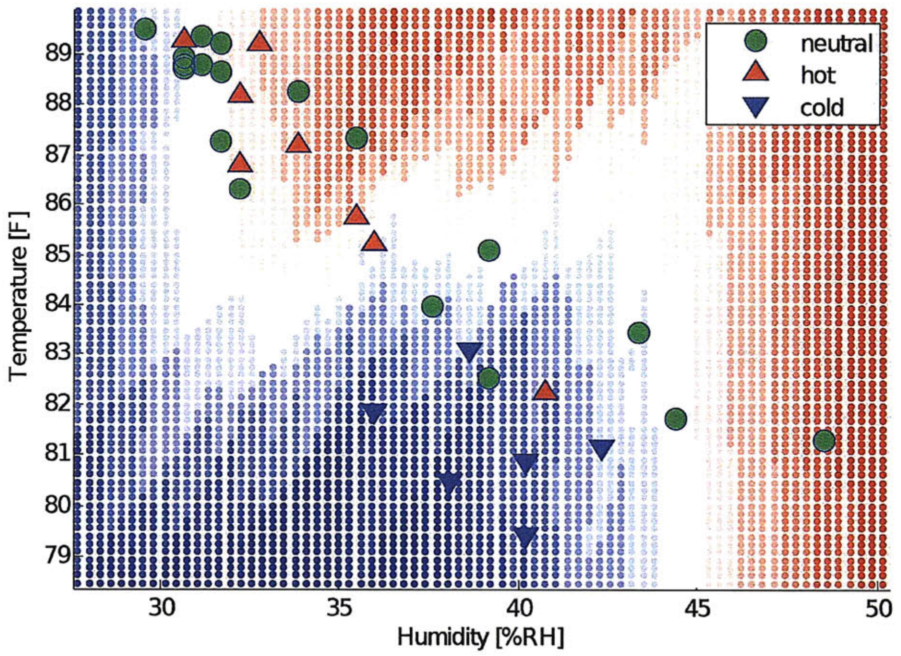


Figure 5.2.10: KNN Distance Metric, $n = 10$, $G = 0$: Node 84

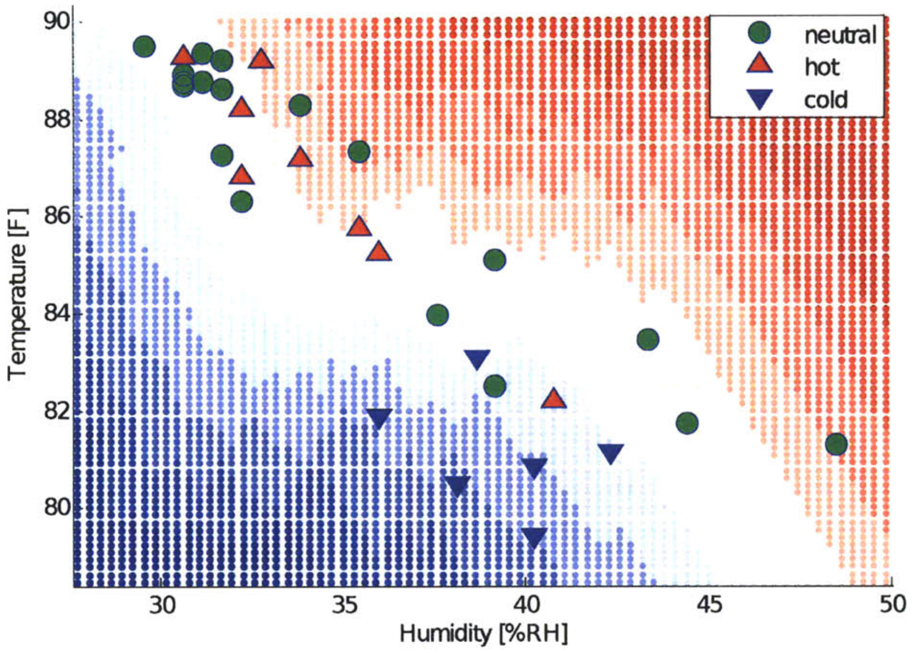


Figure 5.2.11: KNN Distance Metric, $n = 10$, $G = .5$: Node 84

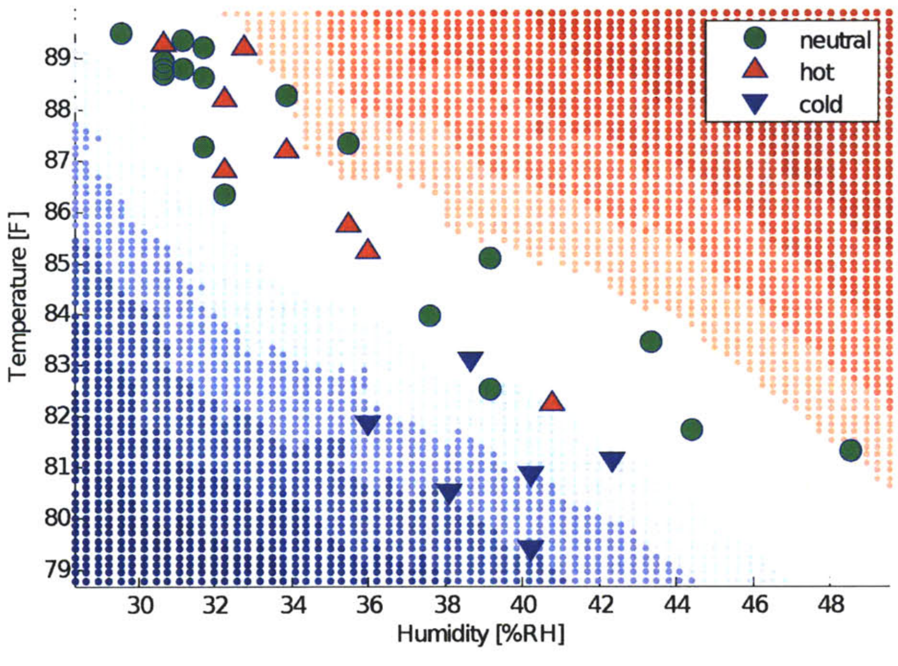


Figure 5.2.12: KNN Distance Metric, $n = 10$, $G = 1$: Node 84

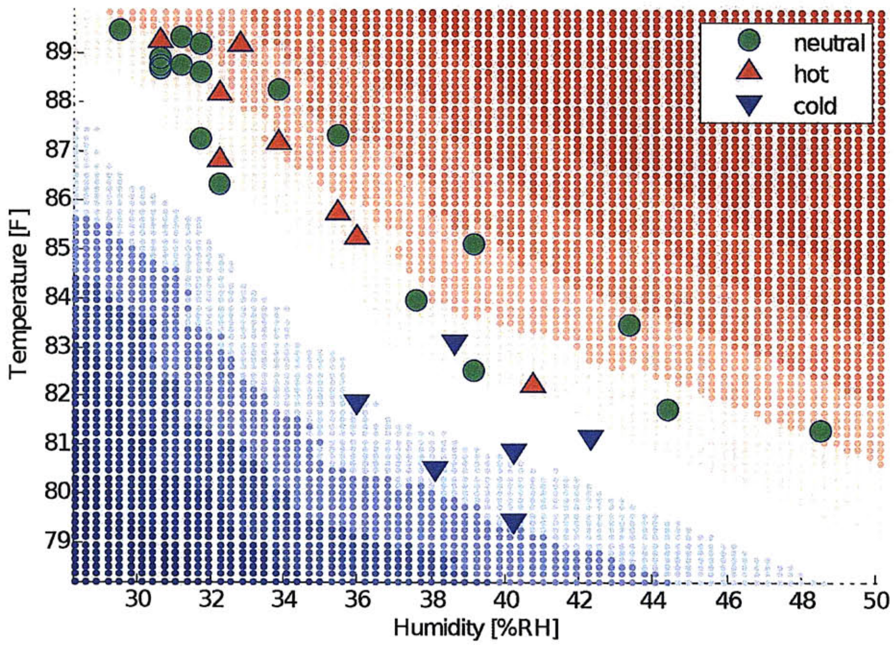


Figure 5.2.13: KNN Distance Metric, $n = 10$, $G = 2$: Node 84

5.3 Fisher Discriminant

Despite the promising results of the KNN Distance Metric, it was decided that they contain too many points of instability to reliably control the HVAC system for the month long trial. For any system to be effective, it must deliver predictable results for users, or they might respond in ways which counter both the goals of the system and themselves. Since the goal of the KNN Distance Metric is the desire to fit a linear boundary between labeled data and measure the distance from this boundary, an algorithm more suited to this task is employed: the Fisher Linear Discriminant.

The Fisher Discriminant seeks the most effective rotation matrix for the given data set, to produce a projection on a lower dimensional space with high class separation. It takes a statistical approach of finding the greatest between-class scatter for the lowest within-class scatter. For this case, it is a simple matter of reducing a two dimensional space to one, with the only difficulty being in choosing a decision boundary. Rather than the usual approach of using the intersection of sample distributions, the decision is based upon discriminating between points which represent the boundary conditions. In this case, the most representative training points are assumed to be those with the most extreme values (e.g. 'hot' data with the lowest temperature values), and a separating line is created at the mean of these data. The comfort distance can then be simply computed as the distance to this boundary.

In order to accommodate the updating and adaptation of the comfort algorithm, a limited set of data points is used in creating the decision boundary. Nine points each of hot and cold are used, which allows a complete update in two to three weeks (users press buttons on average once a day), which is enough time for users to adapt the system before the end of the experiment. In cases where nine data points are not available, as many as are present are used. If less than two data points exist, two points are selected which create a reasonable line in comparison to other users. Representative results of the Fisher Discriminant are shown in Figures 5.3.1 – 5.3.7. Two each of the hot and cold training points are used to select the decision boundary. One point each generally returned more favorable results, but two are used in order to limit problems associated with outliers.

As can be seen from Figures 5.3.1 – 5.3.7, the distance between the 'hot' and 'cold' labeled points varies greatly for the different users. Accordingly, the calculated comfort distance will also vary greatly between users. To normalize this reported comfort distance so the control system can effectively arbitrate between users, the mean distance of 'hot' and 'cold' points from the decision boundary is calculated. As new temperature and humidity data are collected by a user's portable node, the final comfort output is computed as the comfort distance divided by this mean distance. In this way, each user is equally uncomfortable for a given comfort value.

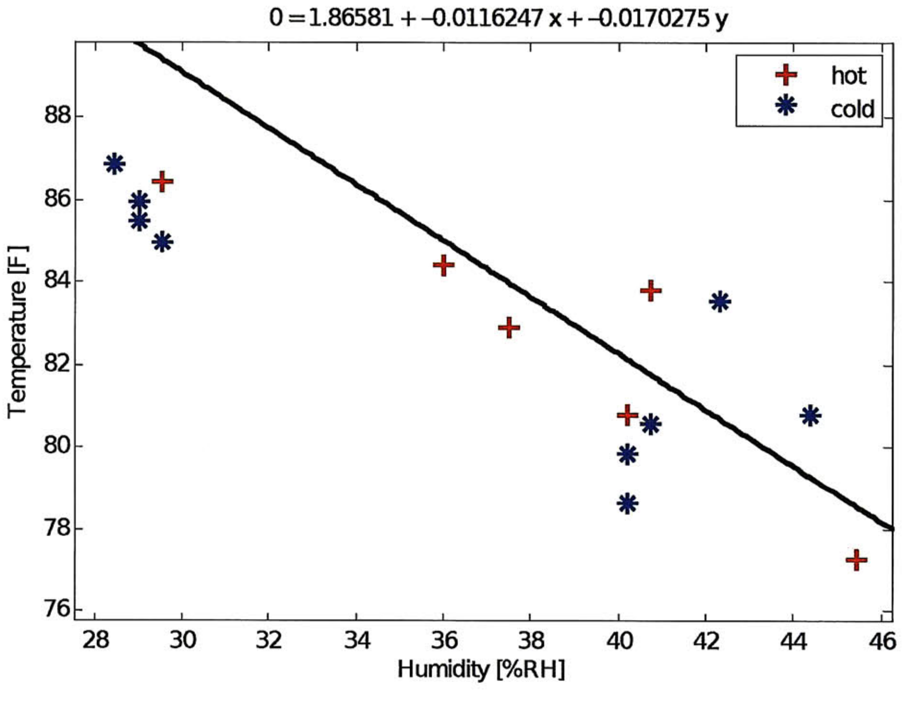


Figure 5.3.1: Fisher Discriminant Boundary: Node 8

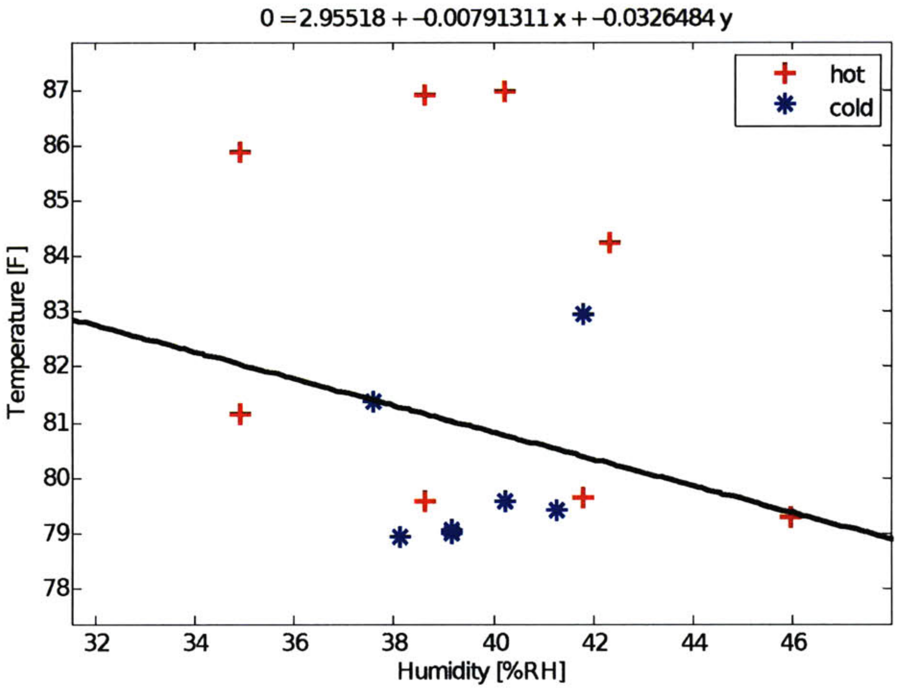


Figure 5.3.2: Fisher Discriminant Boundary: Node 20

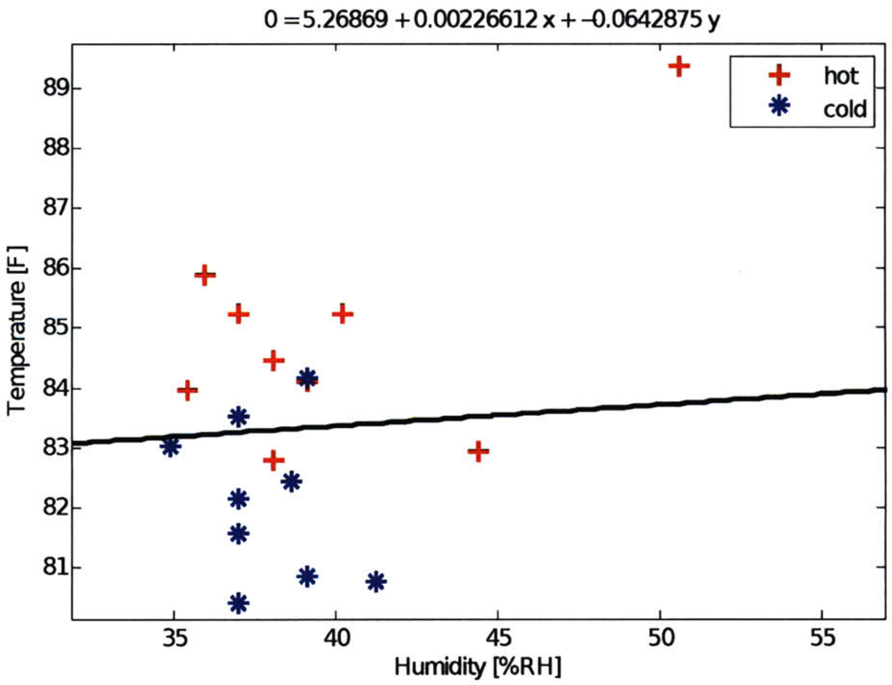


Figure 5.3.3: Fisher Discriminant Boundary: Node 24

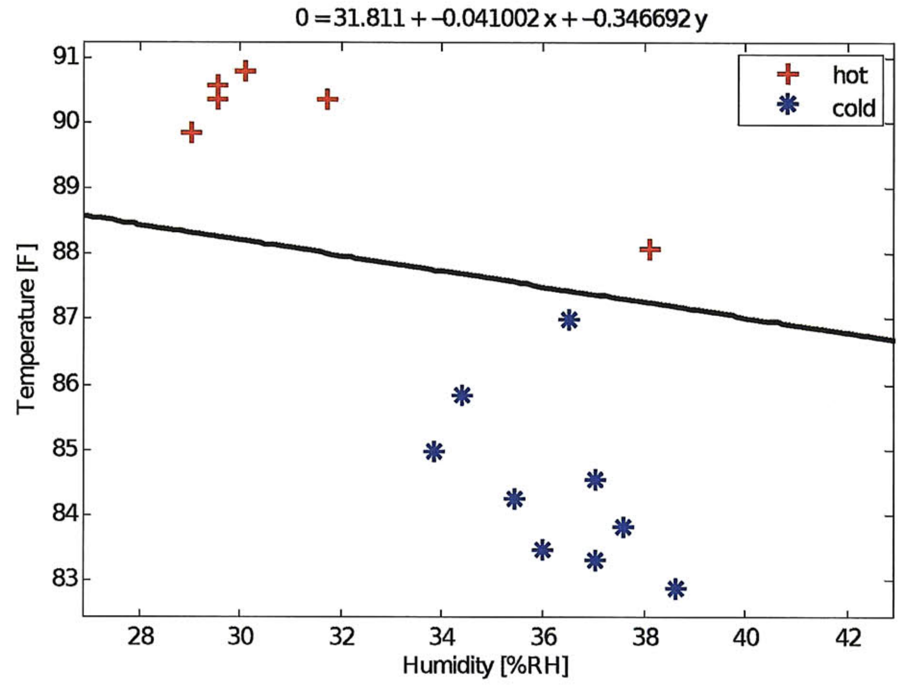


Figure 5.3.4: Fisher Discriminant Boundary: Node 28

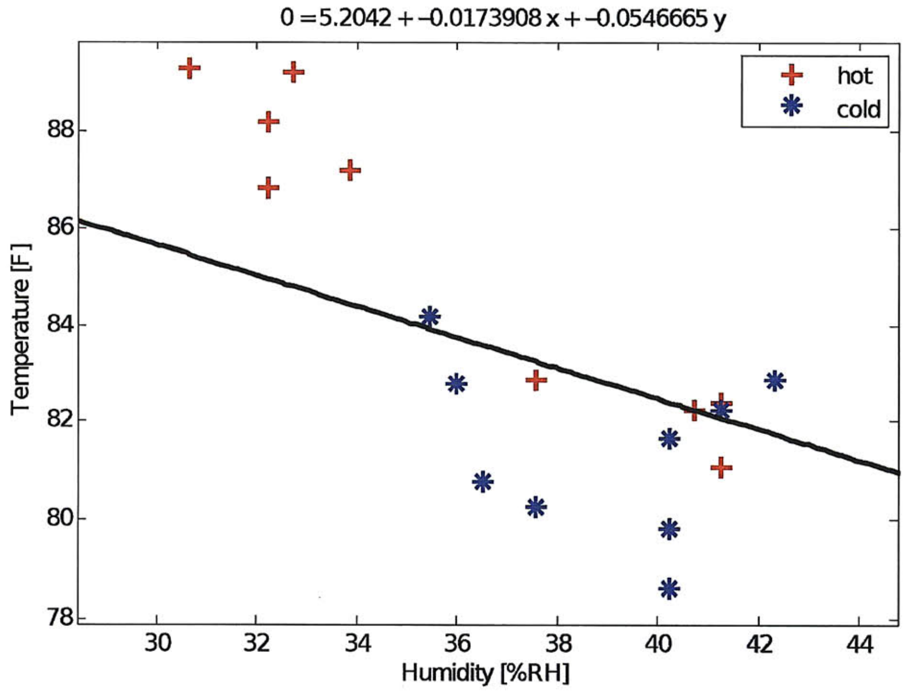


Figure 5.3.5: Fisher Discriminant Boundary: Node 84

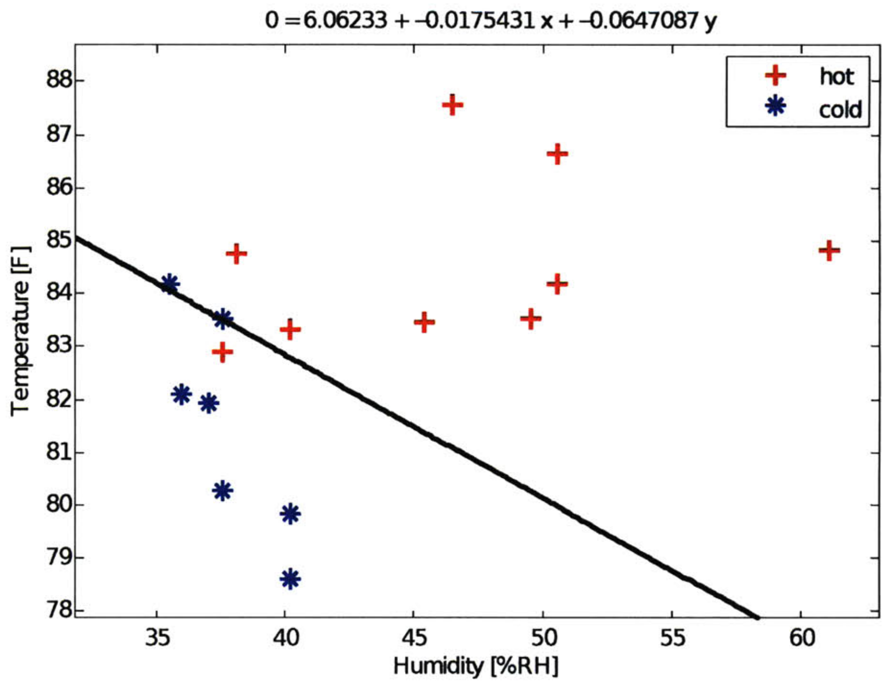


Figure 5.3.6: Fisher Discriminant Boundary: Node 88

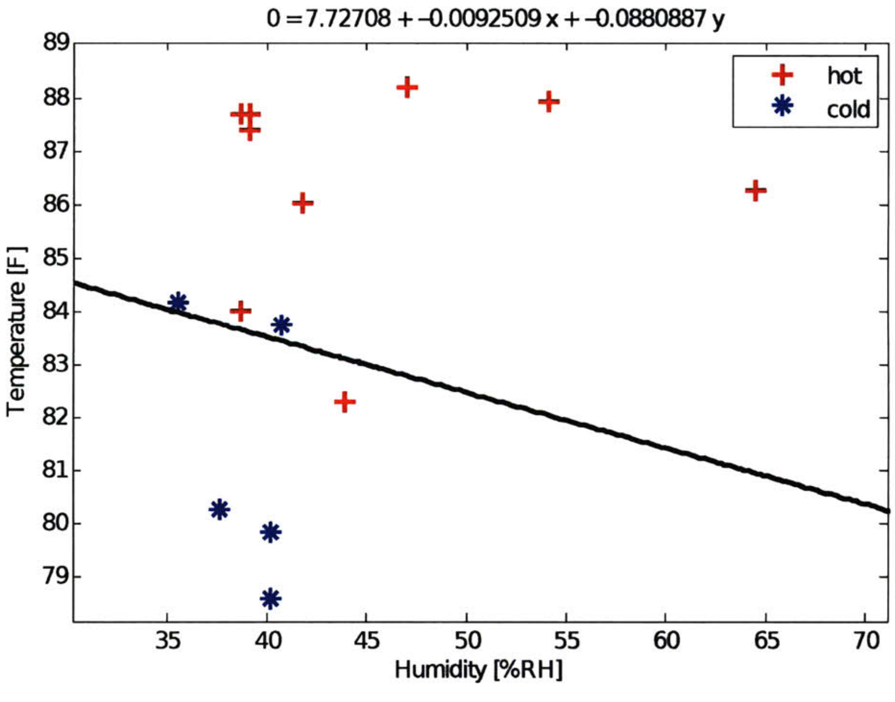


Figure 5.3.7: Fisher Discriminant Boundary: Node 96

Chapter 6

Evaluation

In order to assess the efficacy of a body-worn comfort control system, a long-term user study was performed from May 18th through August 21st of 2009. The study was carried out with a mostly graduate student population at the M.I.T. Media Laboratory. Ten people were assigned individual portable nodes, and four offices and two common areas were equipped with room nodes and control nodes. A full system overview can be found in Section 3.1.

6.1 Experimental Procedure

Phase One of the study ran from May 18th to June 21st, and was mostly a hardware evaluation stage. No actuation was performed for this period, and data were merely gathered on how users interacted with the devices, and how effective the current HVAC system was in terms of meeting their needs. The majority of technical problems were fixed during this time, and the various methods of using the portable nodes were evaluated. In order to have a fair baseline to compare to, the maintenance department made repairs to the VAV damper controllers and thermostats for all of the offices and common spaces in the experiment, before Phase Two of the study began.

During Phase One, participants were allowed to use the portable node in any way they chose. The majority of users left the sensor on their desk, and merely picked it up to press a button if they were too hot or too cold. One user carried the sensor in his pocket, and two users wore the sensor around their necks on lanyards. From analysis of the data, it was found that the inertial activity sensor on the portable node was too noisy to detect proximate motion, and therefore could not tell if its user was in the room if it was not worn. The unworn nodes would also sometimes be set near laptops, or other warm electronic devices, causing the environmental data to no longer represent the ambient conditions. With a more accurate accelerometer, these different usage conditions could be sensed [80] and accounted for, but the portable node battery life would decrease by two orders of magnitude (see Section 3.3).

The data from the user who carried the portable node in his pocket was extremely erratic, with very large temperature excursions depending upon how closely the sensor was being held to the body. To complicate matters further, the device would be removed from the pocket when the button was pressed, causing the temperature to drop very quickly and no longer indicate the pocket temperature. In general, the distinction between the hot and cold classes in all conditions except when the nodes were worn on the lanyards, was very low. For this reason, all nodes were attached to lanyards for Phase Two of the study.

Phase Two of the study ran from June 22$^{\text{nd}}$ to July 26$^{\text{th}}$. The purpose of this stage was baseline data collection. The users were first asked to complete a questionnaire regarding their comfort under the current HVAC system. They were then asked to wear the portable node, via a lanyard, around the neck (either under or over their shirt), and press a button whenever they felt inclined to do so (either hot, cold, or neutral). No actuation was performed for the majority of this period, with the only exception being sporadic control tests to verify the control system software, mostly occurring during the final week.

The portable node data was analyzed during this period to determine the feasibility of using the wearable sensor to control the HVAC system. For the majority of users, the results were still too incoherent to give a reliable indication of comfort. This is a result of the steep thermal gradient next to the human body, and the ability of the sensor to move around depending upon user position. To verify this issue, a series of portable nodes were connected together at a spacing of three centimeters in front of a test subject (see

Figure 6.1.1 for test set-up). These nodes were worn for an hour and a half, and their data compared to one another, as well as to data from nodes which were situated on the desk where the subject sat, and on a nearby wall.

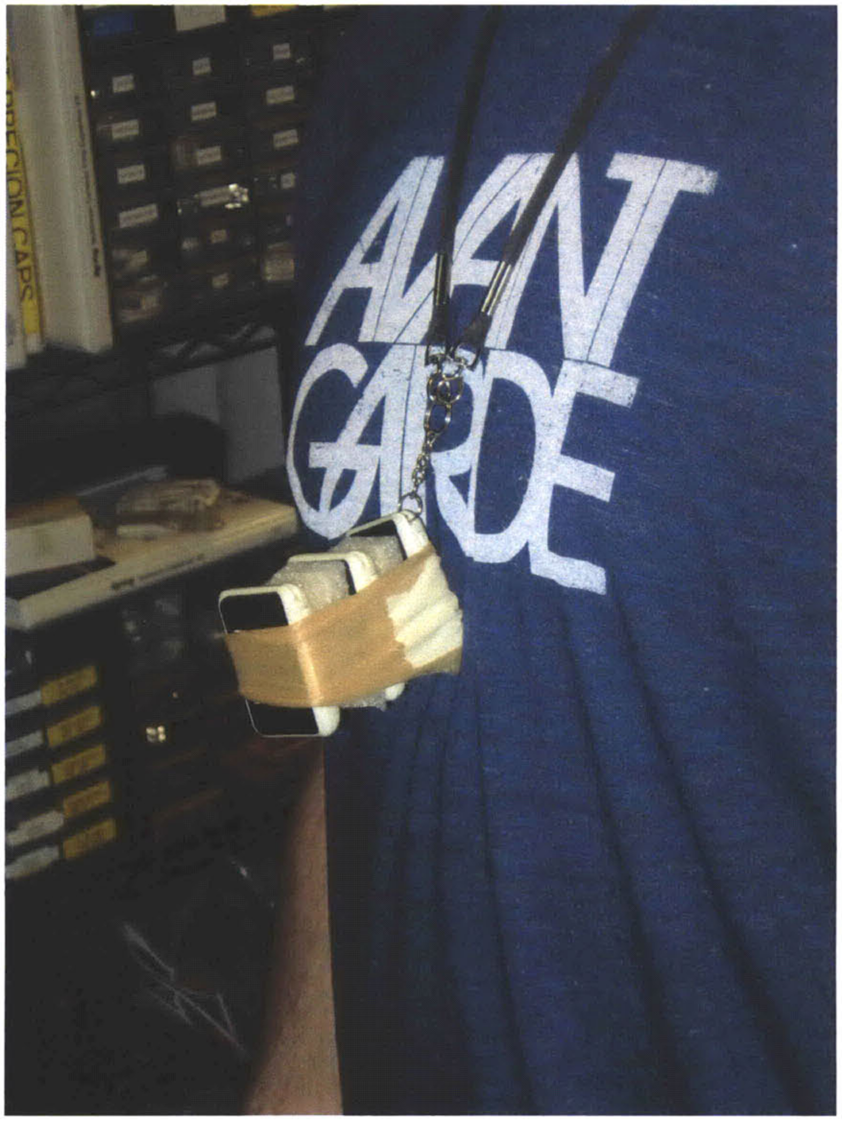


Figure 6.1.1: Body Thermal Gradient Test Set-up

A plot of the temperatures at these locations can be seen in Figure 6.1.2. At 12.5 h three of the nodes were placed next to each other on a table to show the temperature offsets of each node. One of the nodes was previously being worn, and had not fully acclimated by 13.25 h, when they were placed on the body, but the remaining nodes show very similar readings.

It can be seen that there is a 9 °F difference between the temperature measured by a sensor resting against the body, and one just 6 cm away. This is much less than the distance the sensor can move if a person is leaning over while typing, or performing a similar attentive task. The rate at which these sensors respond to ambient conditions is shown by the relative depth of the dips at 13.6 h and 14.4 h. At these points, the subject walked from a warm room to a cool room for a short period. The thermal mass of the body helps stabilize the results.

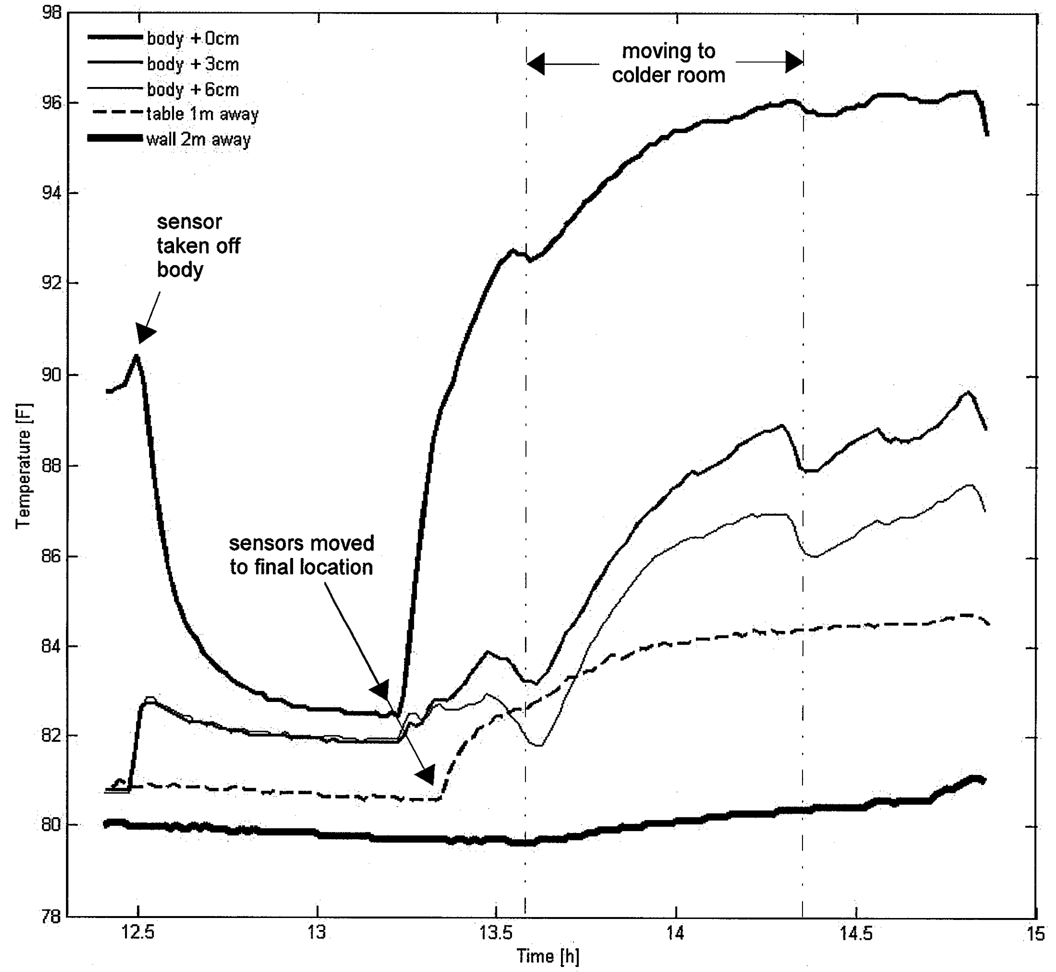


Figure 6.1.2: Body Thermal Gradient

In order to eliminate the variability in the received data that the body thermal gradient incurs, the users were asked to wear the portable nodes on their wrists, beginning July 17th.

A full week of data was collected, with a few periods of routine polling of the subjects as to their comfort condition, in order to build up a large enough data set for analysis. This polling was usually done every fifteen minutes by asking users to press one of the three comfort buttons, and would last anywhere from one to three hours. These data proved stable enough to proceed with a control system based upon the wrist-worn sensors.

Phase Three of the study commenced on July 27th, and continued until August 21st. During this period, the experimental control system was run, with the HVAC system and window motors being controlled via the various sensors in the network. The first few days of the study involved periodic adjustment of controller gain settings, but the control software remained effectively unchanged for the remainder of the study. Users were asked to press buttons on the portable nodes indicating their comfort level whenever they wanted. They were told that the climate control system would respond to their wearable sensor, and would try to mediate the comfort preferences of each occupant of the individual office or common space. Periodic surveys were administered during this period to assess the user's comfort level, and their beliefs about the system. The surveys used in this study can be found in Appendix B, and the results are shown in the next sections.

Since the time-changing aspects of the control algorithm are of interest, Phase Three is subdivided into three sections: Week Two, Week Three, and Week Four. Week Two ran from July 29th to August 7th, eliminating the first few days of the experimental control period, during which parameters were being modified to make the system stable. Week Three ran from August 8th to August 14th, and Week Four ran from August 15th to August 21st.

6.2 Energy Metrics

Any commercial HVAC installation consists of a large number of components, making it difficult to assess the effects of any single part. This work attempts to overcome this problem by averaging over a large number of days, allowing for variability in components to be averaged out as well. Unfortunately, for the majority of the baseline testing period (Phase Two), the outdoor climate was dramatically different from that of the experimental period (Phase Three). To illustrate this, the heating and cooling days for these periods are shown in Table 6.2.1. These data are taken from the National Weather Service database for Boston (measured at Logan Airport) [81], as the outdoor nodes give erroneous data due to direct sunlight effects. Heating and cooling days are an integration, over an entire day, of the temperature difference from 65 °F, with positive values being cooling days, and negative values being heating days. This is often used as a metric to determine how much energy is required to heat or cool a space, as it represents the temperature difference the HVAC system must produce, and has been shown to give linear correlations in some cases [82].

Period	Heating Days	Cooling Days	Number of Days	Average Cooling Days
Phase Two	26	55	22	2.500
Week Two	0	115	10	11.500
Week Three	0	41	7	5.857
Week Four	0	108	7	15.429

Table 6.2.1: Number of Heating and Cooling Days During Experiment

The only method the Personalized Comfort Control System has available to measure energy usage is via the air flow sensors on the VAV boxes. Although energy monitors on the chilled water lines and fan motors would give more accurate results, the area being controlled by the experimental system is a small percentage of the total space, and the effects would be unnoticeable. Despite the air flow from the VAV boxes giving an incomplete picture of the total energy used, it does show the fan usage for the space very accurately. Since fan energy can represent up to 40 percent of total HVAC power consumption [83], this is an important metric by itself. The chiller energy can be estimated by multiplying by the number of cooling days, as this represents the total energy put into the air flowing out of the VAV boxes. These numbers are divided through by the number of degree cooling days to make the results more generally applicable. Plots of these energy metrics showing each room's contribution can be seen in Figures 6.2.1 – 6.2.2.

It can be seen that the total chilled air used decreases for both metrics. The modest improvements shown for Week Two, in comparison to Week Three and Week Four, are due

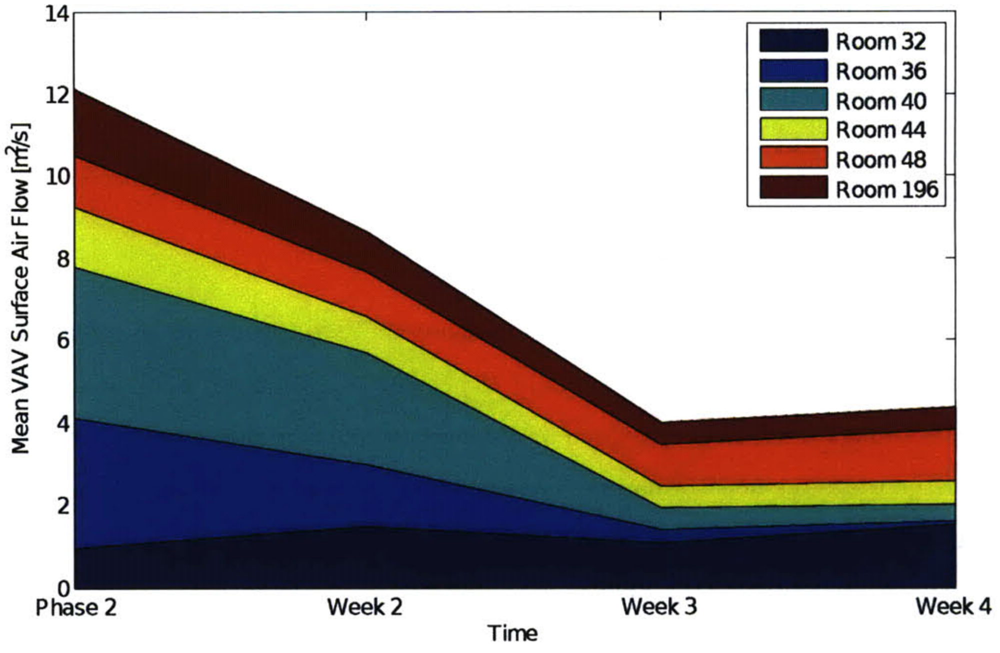


Figure 6.2.1: Chilled Air Usage by Room

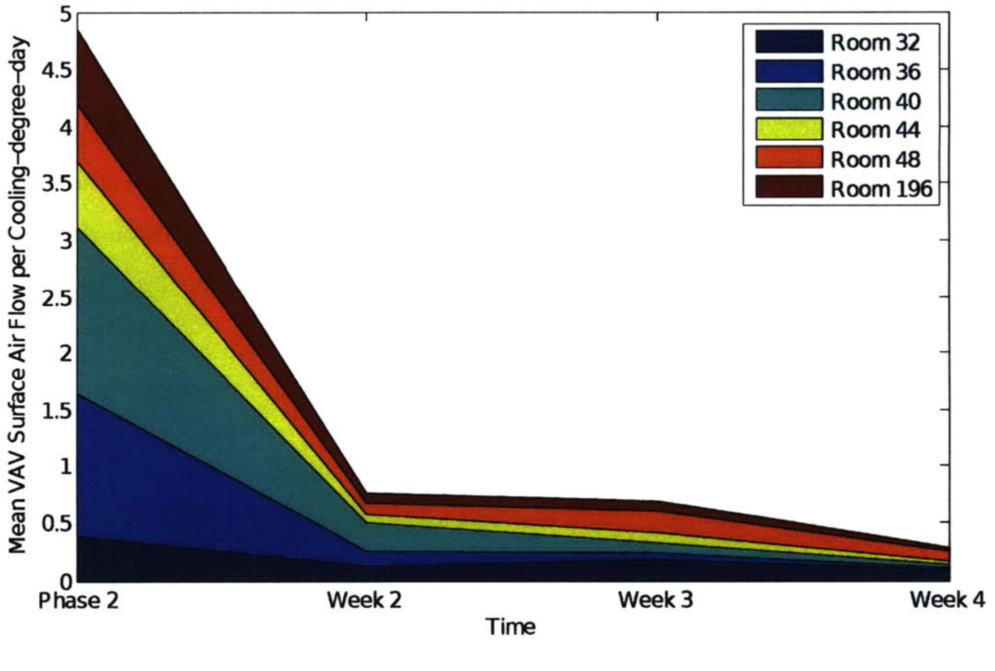


Figure 6.2.2: Chilled Air Usage by Room: Normalized by Cooling Days

to thermostat node temperature settings being changed after Week Two. It was found that the HVAC system could respond fast enough to allow a 6 °F setback, rather than the 4 °F setback which was used previously. Also, the normal control temperatures for Room 36 and Room 40 were increased from 70 °F to 72 °F, due to complaints about the space being too cold when occupants arrived for the day. This adaptation could eventually be done automatically, based upon an average of users' past preferences.

A cursory approximation of energy savings, based upon Phase Two, and an average of Week Three and Week Four, shows a reduction of 75 percent. This is based upon an estimated 40 percent fan usage times the normalized VAV air usage, plus an estimated 60 percent chilled water usage times the non-normalized VAV air usage. The actual savings are much smaller for a number of reasons. Firstly, the main savings shown are due to Room 36 and Room 40 reductions, which represent an unfair comparison, as the area they cooled is also serviced by eight other VAV boxes, none of which were under experimental control. Secondly, the HVAC system, despite having been repaired, was not running properly for the majority of Phase Two. Finally, the chiller units were most likely not running during Phase Two, as the outside temperature was low enough to not require them.

A floorplan of the public area which contained Room 36 and Room 40, showing the other VAV boxes, is depicted in Figure 6.2.3. There are two banks of control, each with a standard thermostat. Bank 2's thermostat is far away from the experimentally controlled damper, and although shutting off this damper completely would have little effect on the reading at the thermostat, cold air can still move across and make up for the cooling difference. The situation is worse for Bank 1, which has its thermostat very close to the experimentally controlled damper, such that the remainder of the bank would produce more cold air to compensate for the difference.

The dampers for the experimentally controlled VAV boxes in the public space were essentially shut for the entire duration of the experiment, as the temperature in the public space was set to 69.7 °F, much colder than was preferred by the occupants. If the space were entirely enclosed, the air usage would reduce, but only enough to bring the temperature up to the preferred level. As it occurred, the temperature only raised near the VAV box

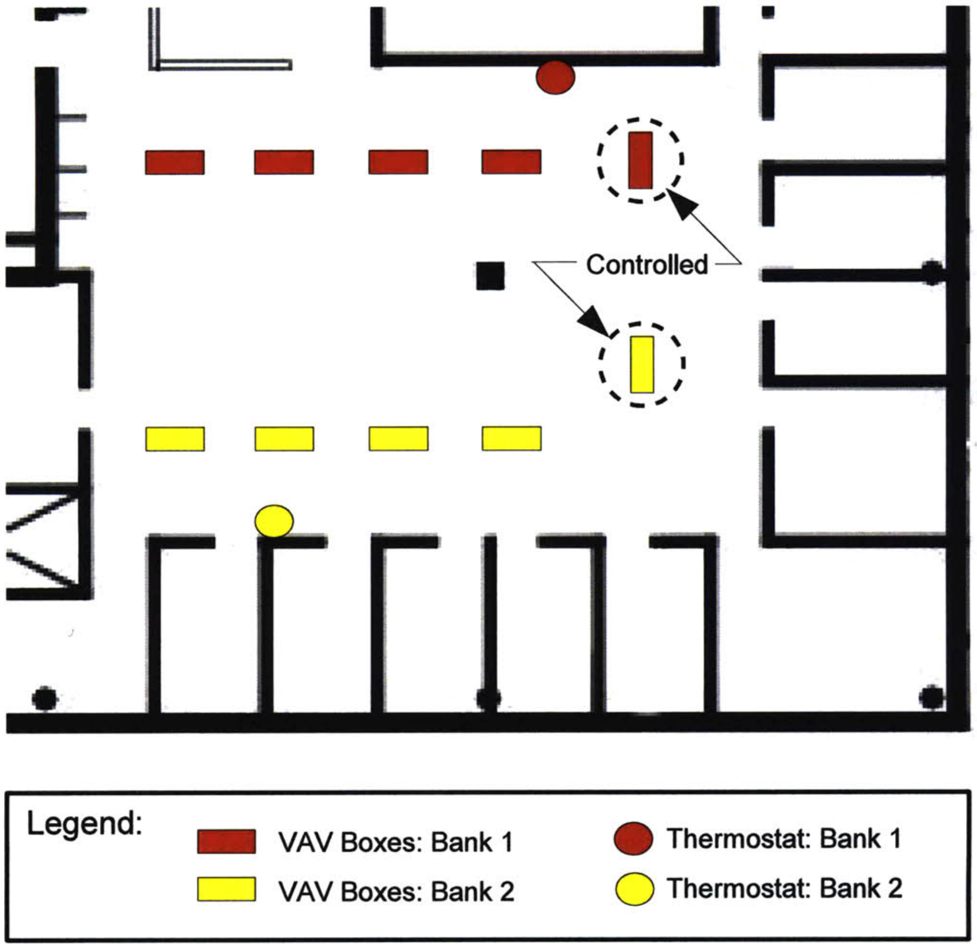


Figure 6.2.3: Floorplan of Public Space with VAV Boxes Shown

in Bank 2, and this was only 1.3 °F. The occupants of that area still stated that it was too cold, even after the dampers had fully closed. It is difficult to predict what the system would have done under different conditions, but it is shown that the temperature would have risen by at least 1.3 °F, and it can be assumed that the temperature near the other damper would have increased by at least as much, as this area was not often occupied.

To best account for Phase Two energy usage, Room 44 will be analyzed in detail. This room is selected because it was completely repaired, with both the thermostat and damper motor controller being replaced. It also has the most linear and sensitive wind speed sensor, so the data can be relied upon to give an accurate view of the room state. Unfortunately, the wind sensor was replaced half-way through Phase Two, but the latter half of this period had a larger percentage of cooling degree days, making it more useful for this analysis.

A plot of Room 44's temperature and chilled air usage can be seen in Figure 6.2.4. This period represents the last 13 days of Phase Two, which had 37 cooling degree days, and six heating degree days. This is about half of the cooling degree days seen at the lowest period for Phase Three, and as a result, the main fan speed was reduced, and the economizer was most likely run at full open. The reduced fan speed can be seen from days 7 – 13, where the damper was full open, and the maximum air flow still could not reach its previous level. The dips during this period represent reductions due to increased demand during the day, with volume increasing again at night as some of the thermostats would no longer call for more air. Periodic glitches in the wind speed readings are caused by normal control actions and a few experimental tests.

It is important to note that the temperature during this period was oscillating at least 5 °F every day, with daytime temperatures usually around 73 °F. This temperature was considered too warm by the occupants of the room, so they turned the thermostat down. Since the chillers were not running, and the fan speed was reduced, this did not have the desired effect of cooling down the room, and the thermostat was turned down even further, probably to its lowest level. This did not change the room temperature during the day, but did affect the temperature at night, as the air would become cool enough to bring the room temperature down. Unfortunately, the low setting on the thermostat also forced the damper to stay full open for the entire duration of Phase Two.

The thermostat settings can be inferred from the temperature versus air flow profile for days 0 – 7, when there was sufficient air flow to cool the room. The air flow had previously been low, and the thermostat was set low, and day 0 represents the fan speed being increased. The following evening, the temperature dropped to 65 °F, and the user most likely turned the thermostat to 70 °F, where it stayed for the next day. The reduction in air flow during these periods shows the system backing off to maintain temperature. At the end of day two, the system could not maintain temperature, and the thermostat was most likely turned back down to 66 °F, where it stayed for the remainder of Phase Two.

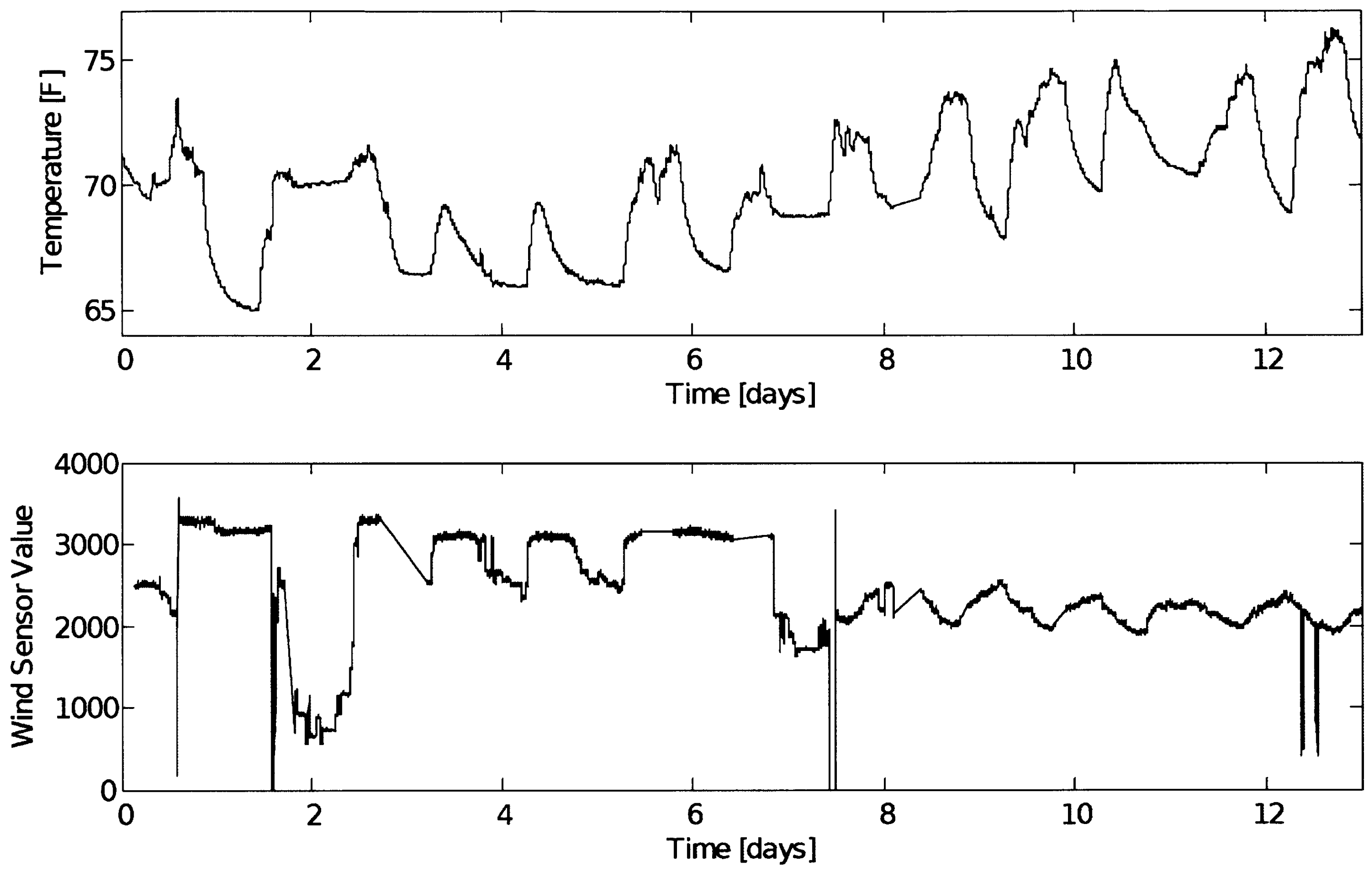


Figure 6.2.4: Temperature and Chilled Air Usage for Room 44: 7.1 – 7.13

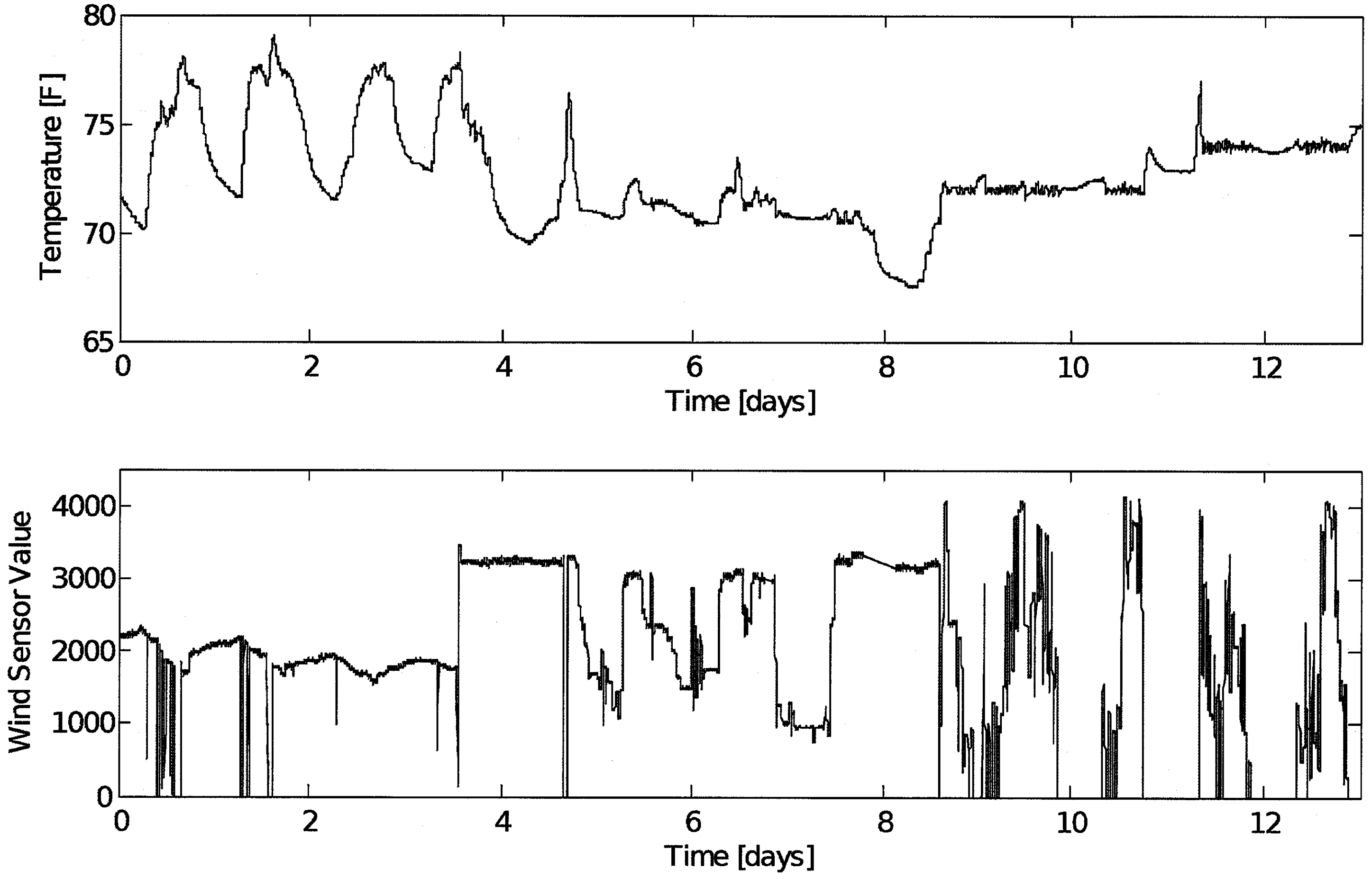


Figure 6.2.5: Temperature and Chilled Air Usage for Room 44: 7.14 – 7.26

This plot of essentially continuous chilled air usage shows that there is a high penalty to be paid for not meeting occupant expectations. It is not clear, however, what the exact energy trade-off would be for running the chillers and fan at a slightly increased level, in order to decrease occupied temperatures enough to encourage reasonable thermostat usage. Regardless, due to the nightly dips and continuous air usage, the average temperature during this period was 69.9 °F, which is much lower than would have been desired if the occupants could have accurately set their temperature. A justification for this is shown in Figure 6.2.5. During this time period, the system was being transitioned from standard to experimental control. Fan speed was also increased during day three, as it had finally become warm enough to warrant extra cooling. The system can be seen to bring itself into regulation during day four, with the slight exception of a fan malfunction during that day. On day nine, the experimental control was initiated, and an even tighter thermal regulation can be seen.

During the period when fan speed was increased, the average occupied room temperature was 71.5 °F, with excursions up to 73 °F. This suggests that even given adequate air flow, and new components, the system was incapable of maintaining a constant temperature in the room, although changes could be due to thermostat actions. After the experimental system was initiated, a fixed 72 °F was held for two days, with an increase to 74 °F after occupant complaints of being too cold. Slight deviations during this time are due to system resets, as the control structure was still being assembled. It is not exactly clear why the preferred ambient temperature increased after the tighter thermal regulation was imposed, but it is believed to be related to the increased level of discomfort which has been shown to accompany large temperature swings [84]. Alternatively, perhaps the occupants no longer believed the thermostat had any effect on the temperature in the room, and did not previously attempt to change the setting. In either case, the ability to control the temperature in the room accurately led to an increase in temperature of 4 °F, and the energy savings associated with this increase. The chilled air usage decreased by over 50 percent between the standard and experimental control systems, and they both had almost identical amounts of cooling degree days: 26 and 27, respectively.

The energy savings of the experimental control system are, therefore, mostly due to fixing a broken system. Although this is not to be discounted as a benefit of this type of wireless sensor retrofit, it is not a fair comparison to the savings which personalized control can enable. To account for this, the energy used for the four day period when the experimental control system was initiated, but the comfort control algorithms were not yet running (Figure 6.2.5: days 9 – 13), will be used as a baseline. During this period, the average temperature was 73.1 °F, average air usage was $0.6528\,\mathrm{m}^2/\mathrm{s}$, and average air usage per cooling degree day was 0.0967. Comparing this to Week Three and Week Four of the personalized control system shows an increase in temperature to 73.9 °F, a decrease in air usage by 15 percent, and a decrease in air usage per cooling day of 31 percent. Weighting these by the relative fan and chiller metrics discussed above gives a total energy saving of 21 percent. A more conservative estimate would be based solely on Week Three's data, as this week had a very similar average number of cooling degree days to the baseline (Week Three had 5.857, the baseline had 6.75, and Week Four had 15.429). This gives an increase of average temperature to 73.5 °F (very similar to the baseline), a decrease in air usage of 17 percent, and a decrease in air usage per cooling degree day of 4 percent, summing to an estimated 11.8 percent savings.

A comparison of damper usage for the remainder of the rooms between Week Three and this baseline period of proper HVAC functioning produces similar results to Room 44. With Rooms 36 and 40 removed, as they represent an unfair comparison, the remaining rooms show an average decrease of 20 percent in air usage per degree cooling day per degree change in room temperature, and a decrease of 30 percent in air usage per degree change in room temperature. Regardless of degree cooling days and damper usage, it can be generally inferred that a system which keeps the average room temperature higher will consume less energy. The 12 percent savings from the 0.4 °F increase shown for Room 44 can be seen throughout the system, as the average room temperature increased for almost all of the rooms. A plot of the temperature differential contributions of these rooms can be seen in Figure 6.2.6, with the average temperature rising 0.8 °F.

Assuming a linear correlation between temperature change and energy savings, this average

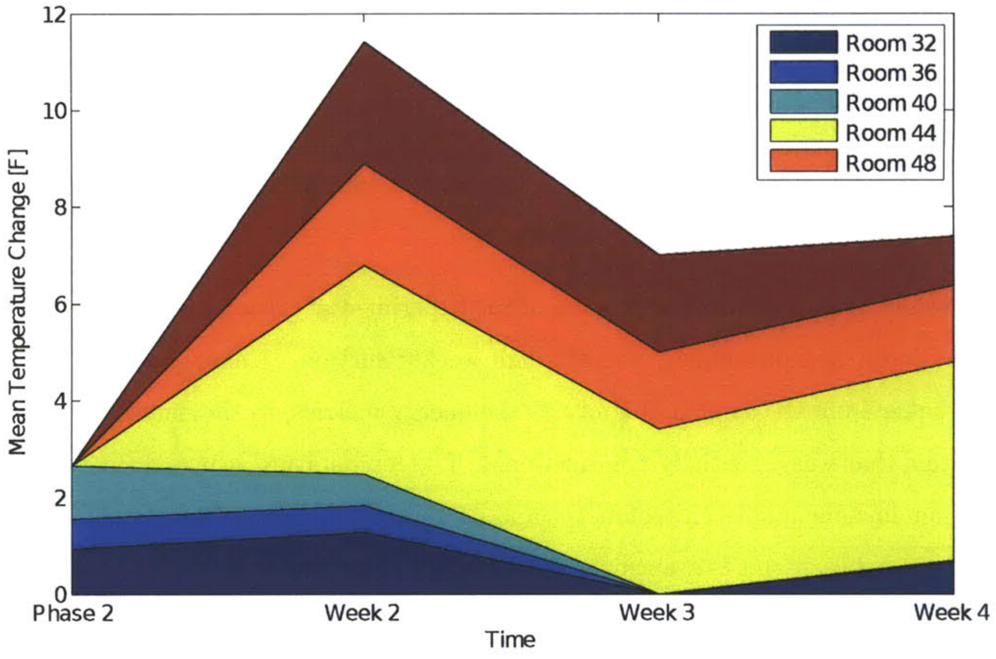


Figure 6.2.6: Change in Temperature of Rooms from Phase Two

0.8 °F temperature increase would give an upper bound of 24 percent savings. Ultimately, the decrease in chilled air usage per degree cooling day is the only metric that is certain, and this shows an eight percent decrease, which, when multiplied by the 40 percent fan usage metric, can be taken as a lower bound of 3.2 percent. This approximate 3.2 – 24 percent savings gives an indication of the Personalized Comfort Control System's ability to modify room temperature to reduce energy while increasing comfort. Many other factors, which are unavailable to this study, such as solar gains and daily temperature profile during energy draw, must also be factored in for a more accurate assessment. The Week Four data, although showing greater energy savings, also showed a decrease in the comfort of the occupants, as the control system could not respond fast enough under the much hotter conditions of that week, so these are not used. Furthermore, of the two methods employed by the experimental control system to save energy (setback and comfort control), one of these (the setback function) was rarely running for Room 44 (upon which the savings estimates are based) during Week Three. An error in the Room Module code kept old entry times from being removed, leading the system to believe it would be occupied at almost all hours of the day. This correlates the net savings almost entirely to comfort control: the micro-adjustments throughout the day that keep the temperature only as low as needed.

6.3 Comfort Metrics

There are two ways in which the comfort of the experimental subjects is measured: through 'hot' and 'cold' button presses, and through weekly surveys. Unfortunately, the comfort metrics suffer from the same ambiguity as the energy metrics, as they are being compared to a system that was essentially nonfunctional. The temperature swings during Phase Two created an uncomfortable environment, and the Entrance Survey clearly shows this. A comparison between the Entrance Surveys and Exit Surveys is shown in Figures 6.3.1 – 6.3.8, with the Entrance Survey representing user beliefs under normal control, and the Exit Survey referencing the four weeks of experimental control. These surveys had identical questions, and were taken two months apart from each other, making them a relatively unbiased indicator of user preferences. All of the surveys used, and their responses, can be found in Appendix B.

Users clearly felt uncomfortable under the standard control system, with the majority disagreeing with the statement: "The building I work in is comfortable in terms of temperature." A feeling of a lack of control over the environment, and a desire for more control, is also shown via Figures 6.3.5 and 6.3.7. The experimental system was mildly successful in granting this control, as users responded more favorably to the Exit Survey, but a desire for more control was still expressed. Ultimately, everybody's comfort can not be optimized at all times, as spaces must be shared, so a desire for more control will always exist.

To assess how well the system performed at managing these conflicting comfort needs, weekly polls of thermal comfort level were performed. These employed the seven point scale used in the PMV, and can therefore be compared to standard HVAC practices of keeping the temperature within bounds of 80 percent occupant satisfaction. An average of

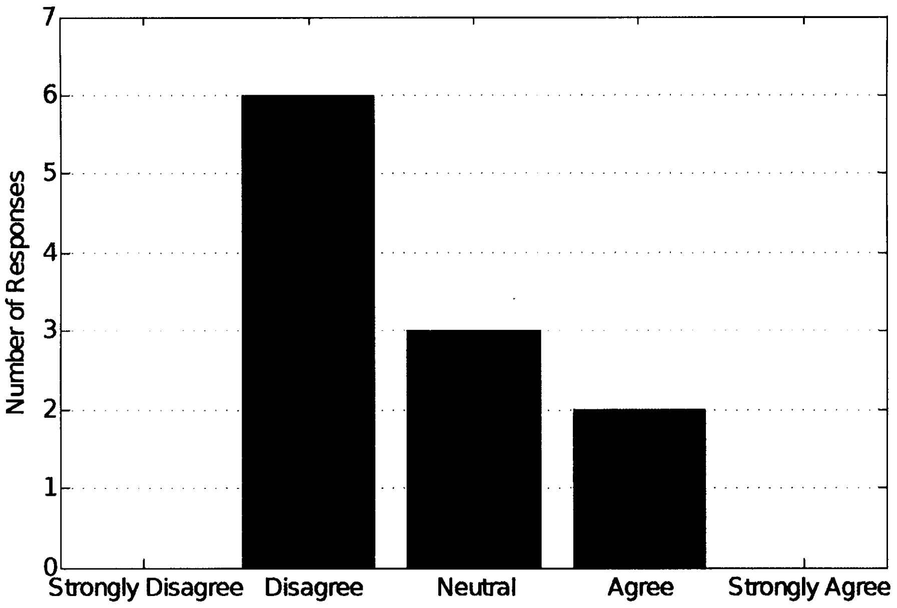


Figure 6.3.1: Responses to Entrance Survey: "The building I work in is comfortable in terms of temperature."

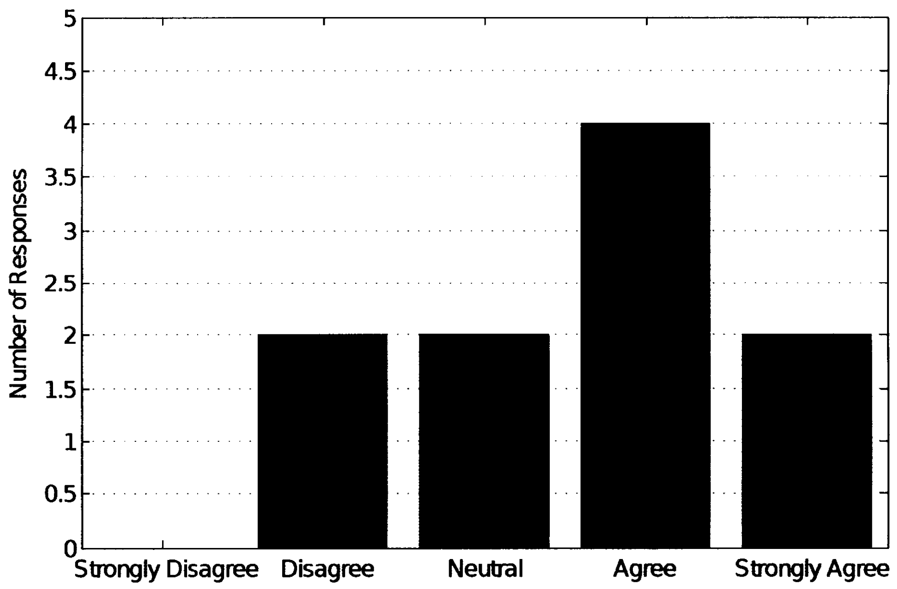


Figure 6.3.2: Responses to Exit Survey: "The building I work in is comfortable in terms of temperature."

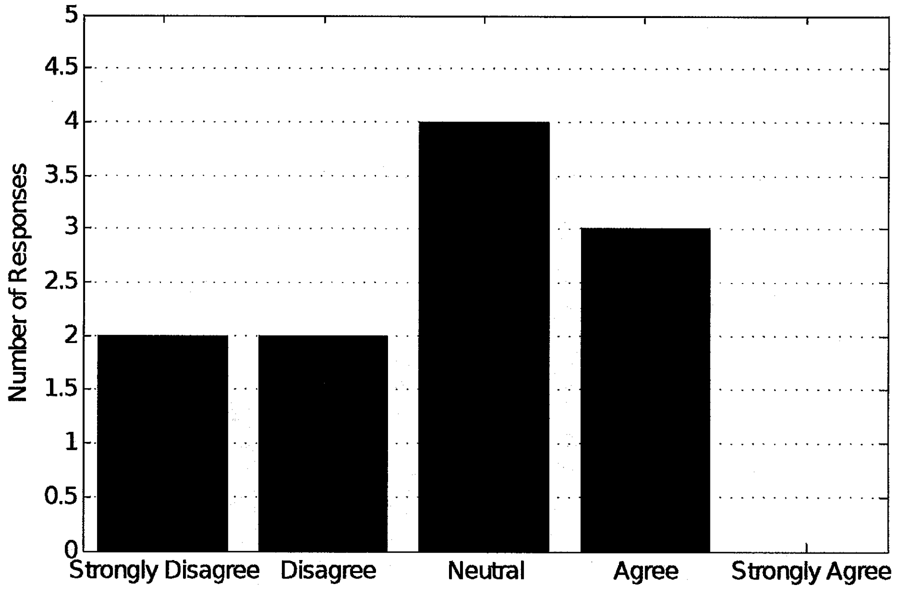


Figure 6.3.3: Responses to Entrance Survey: "Overall, I am satisfied with the comfort level in my office."

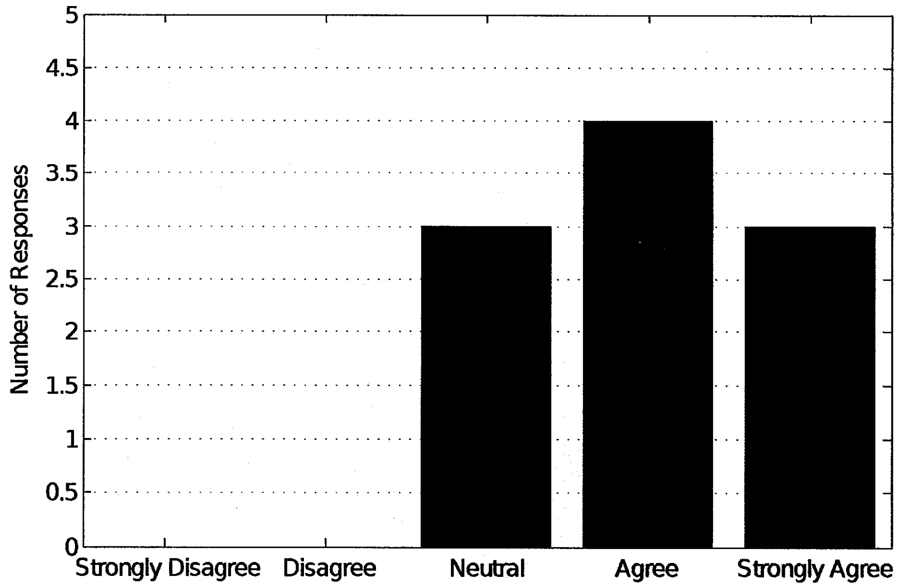


Figure 6.3.4: Responses to Exit Survey: "Overall, I am satisfied with the comfort level in my office."

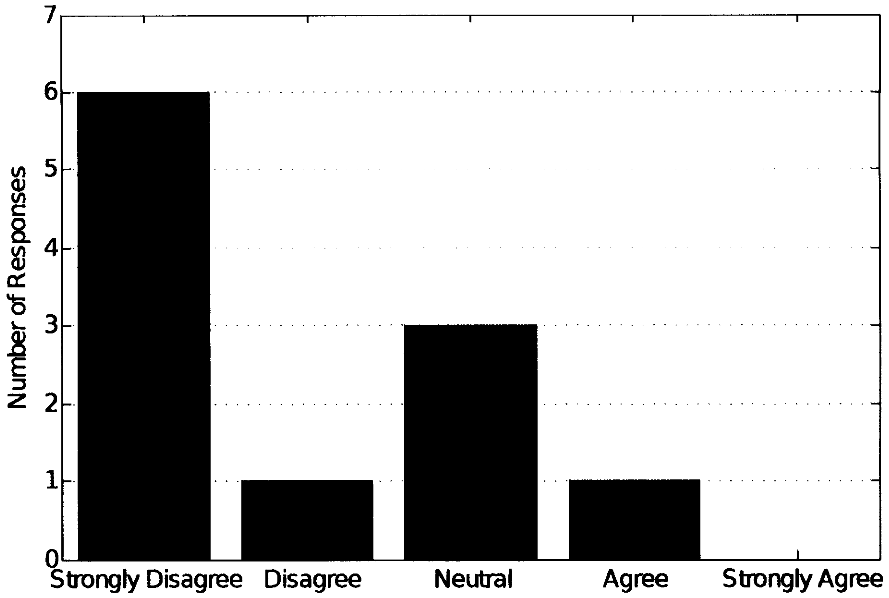


Figure 6.3.5: Responses to Entrance Survey: "I feel in control of my local comfort level in my building."

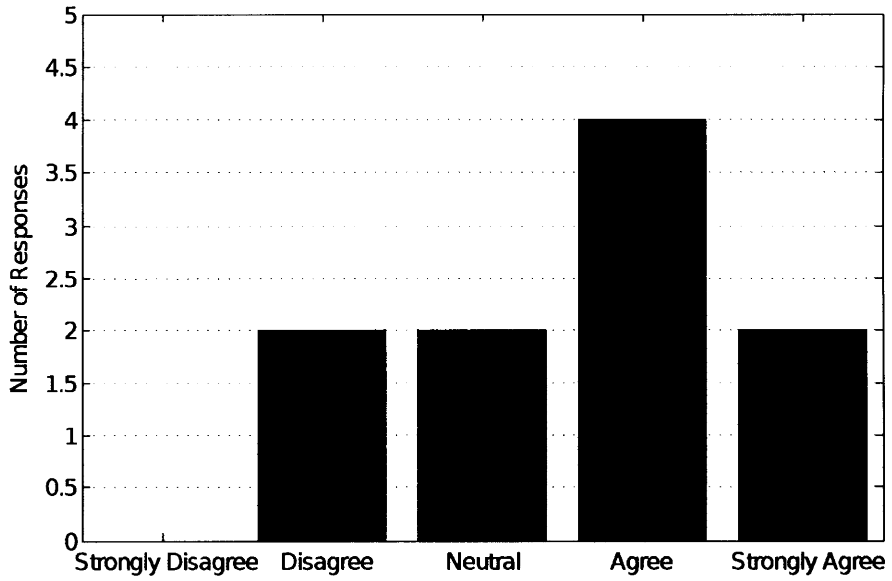


Figure 6.3.6: Responses to Exit Survey: "I feel in control of my local comfort level in my building."

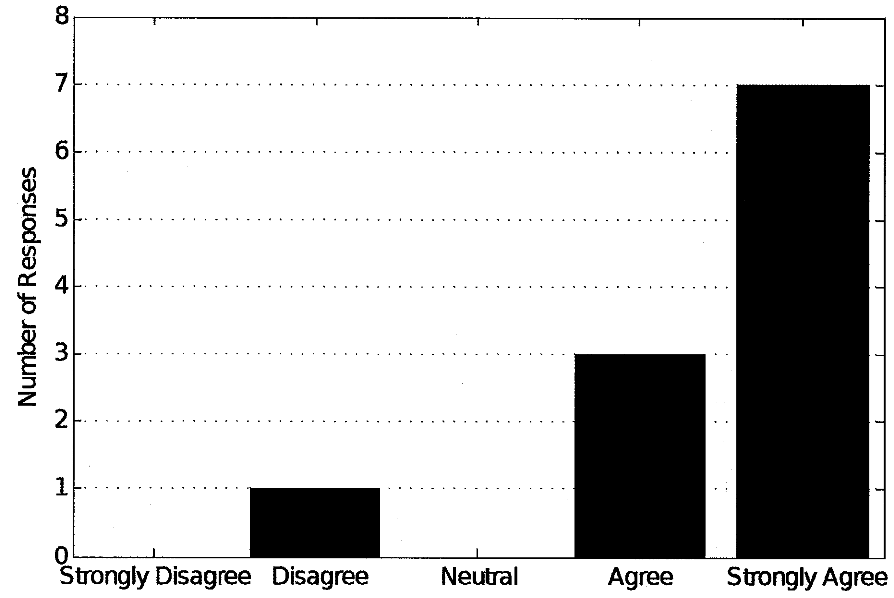


Figure 6.3.7: Responses to Entrance Survey: "I would like more control over my local temperature in my building."

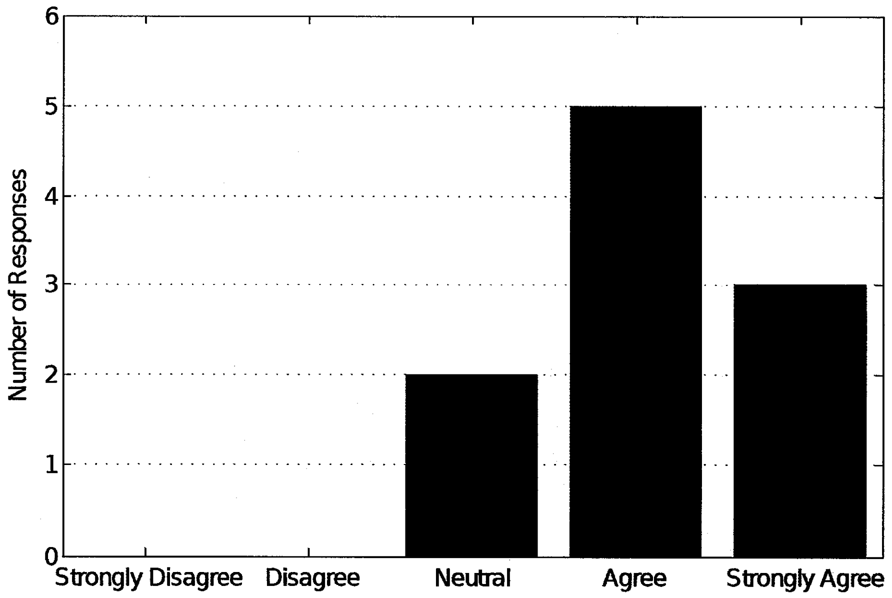


Figure 6.3.8: Responses to Exit Survey: "I would like more control over my local temperature in my building."

the users' comfort levels for the four-week experimental control period can be seen in Figure 6.3.9. The PMV counts the 'Slightly Cool' through 'Slightly Warm' categories as being comfortable, and the occupants were in this zone 81 percent of the time. This percentage increased over the study, starting at 76 percent and ending at 85 percent, most likely due to the system learning the preferences of the users.

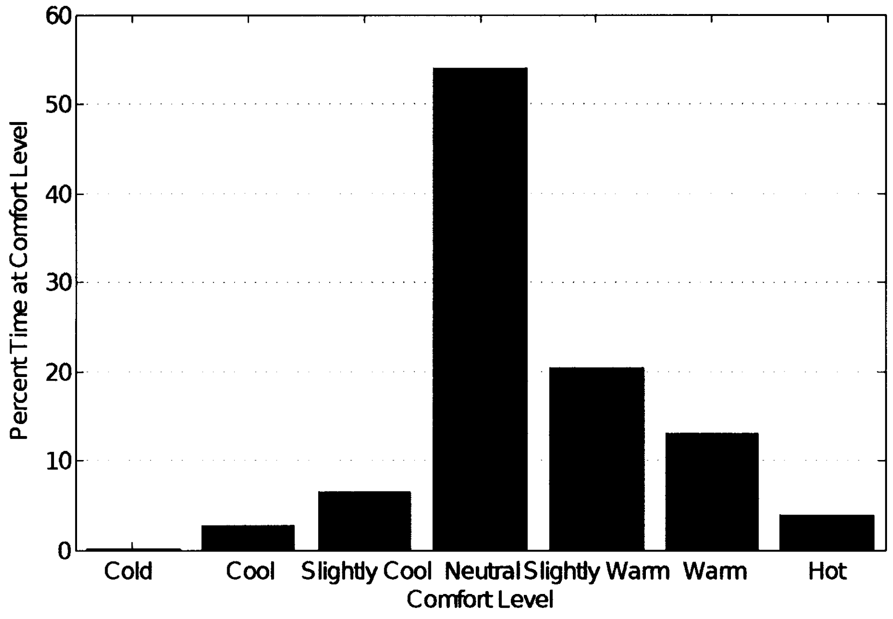


Figure 6.3.9: Average Comfort Levels for Users During Experimental Control

Although this places the experimental control system within the theoretical bounds of standard practices, it is not a good indicator of how well the system could perform. A number of problems during the experiment caused periods of uncomfortable temperatures for a few of the occupants. For example, Node 20 was blocked by an RF transmitter with the same node ID for almost the entirety of Week Two. Room nodes 44 and 32 experienced outages during Week Two and Week Three, essentially disabling the location system for the occupants of those offices. These outages occurred for approximately ten percent of the time, which represents almost an entire day of non-functionality. Furthermore, the HVAC system never went cold enough for a few of the users to hit the 'cold' button, giving no training to determine a lower bound on comfort. To make up for this, estimated cold points were used, which most likely placed the decision boundary too high for these users.

To accommodate for these malfunctions, the following section will evaluate the system for each user, revealing both the promises and shortcomings of this implementation. A tablature of user comfort is shown in Table 6.3.1 for the duration of the experiment. These data are taken from survey responses to the PMV scale, used in weekly pollings. For the majority of the users, over 95 percent comfort rates were common. In fact, the only users for which this is not the case are those for which 'cold' data points had to be manually entered, and User 20, whose data was not received during Week Two. It can be seen that User 20's comfort increased dramatically after this time, suggesting that the comfort system results are not merely placebo gains.

	Percentage of Time Comfortable		
User	Week Two	Week Three	Week Four
4	20	40	60
8	95	100	100
20	20	60	75
24	80	90	90
28	95	90	97
76	90	N/A	N/A
84	100	100	100
84	90	95	95
96	90	95	90
104	80	60	60

Table 6.3.1: User Comfort Versus Time

This dramatic difference in effectiveness between users is partially explained by estimated data points and radio interference, but there must be other issues, as User 20's comfort did not come back up the level of the others after the interference went away. Nor did the system effectively learn the zones of discomfort after repeated training by the users with estimated 'cold' data points. Fundamentally, the system either did not understand the users' discomfort, or was unable to respond to it effectively enough. To disambiguate between these conditions, a plot of users' reported comfort level versus the system's beliefs of their comfort is shown in Figure 6.3.10. The data is taken from the weekly polling of comfort, with -3 representing 'cold' and $+3$ representing 'hot'. The computed comfort uses a similar metric, with greater positive values representing greater heat discomfort, and the reverse for cold discomfort. The malfunctioning nodes are shown with asterisks in the plot, to help differentiate their performance from the functioning nodes. Lines are also placed on this plot to indicate the thresholds below which the comfort control algorithm would decide no action was required.

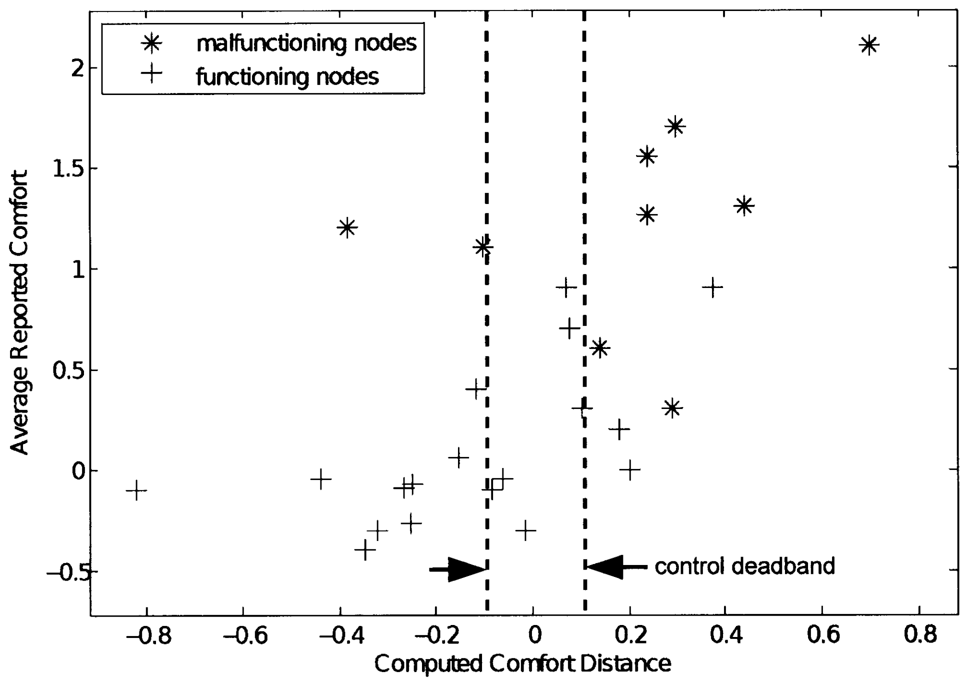


Figure 6.3.10: Reported Comfort versus Computed Comfort for All Users: Week Two – Week Four

With the exception of a few points, the computed comfort tracks the reported comfort fairly well. It is also important to note that the level of reported comfort used in the PMV to determine comfort boundaries is ± 1, which is where all the functioning nodes lie. Even for the majority of the malfunctioning nodes, the system knew those users were hot, but was unable to respond accordingly. This could be due to a number of factors. First, the deadband on the controller limited the system's ability to respond to all levels of discomfort. Second, the response time on the controller was limited in its ability to cool down the user. Typically, an hour was required for stabilization, due to the fact that not just the air, but also the thermal mass of the user was required to change temperature for the setpoint to move. For users who frequently left the office and returned, this meant that the system may have never settled upon their needs, as the room would switch to normal control when they left. Unfortunately, location system failures would have the same problem, so a six minute time-out was placed on switching back to normal control to help alleviate this problem. Third, the user discomfort could have been due to the balancing of an officemate's opposite thermal sensation. Finally, the system may have not been able to effect the user's comfort due to the limited granularity of VAV box location.

Ultimately, the main fault seems to be with a lack of system responsiveness, as compared to a lack of system understanding of comfort. This idea is further promoted by Figure 6.3.11, which shows the average reported comfort versus computed comfort as a function of time. An idealized system goal is placed on this graph, which passes through the origin, and represents system beliefs matching user experiences. Except for User 8, the system can clearly be seen moving towards the ideal system goal over the three polling periods. User 8 only had cold complaints, and it is assumed that the reinforcement in this case was to

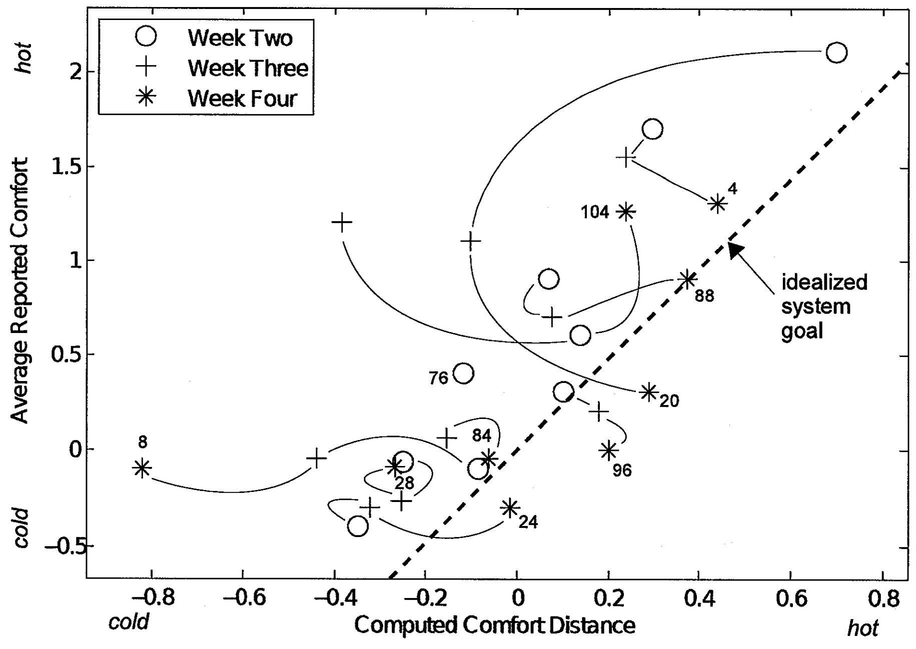


Figure 6.3.11: Reported Comfort versus Computed Comfort versus Time: Week Two – Week Four

encourage the system to believe the user was in a colder state, whereas the majority of the remaining users complained of being hot, and their general shifting to the right reinforces the system in this manner.

Another example of this lack of effective control, is the set of problems associated with frequent, or even infrequent, transitions between the offices and the public space. Since the public space's temperature was essentially not under the system's control due to the proximate VAV boxes cooling the area, the temperature in the public space was two to three degrees lower than in the offices. As shown in Figure 6.3.12, this causes the comfort distance to drop dramatically whenever a user would leave an office. This has the negative effect of setting the system's belief of the user's preference to a much lower level than is probably desired. This is not as much of an issue for leaving an office as it is for entering, as the system knows the user's location and ceases controlling when the user leaves. Upon re-entry, however, the node takes quite a few minutes to acclimate, and has an inaccurate control setpoint for this period. This could be accounted for by inhibiting control signals when a user transitions between spaces, but this would increase the lag of the system.

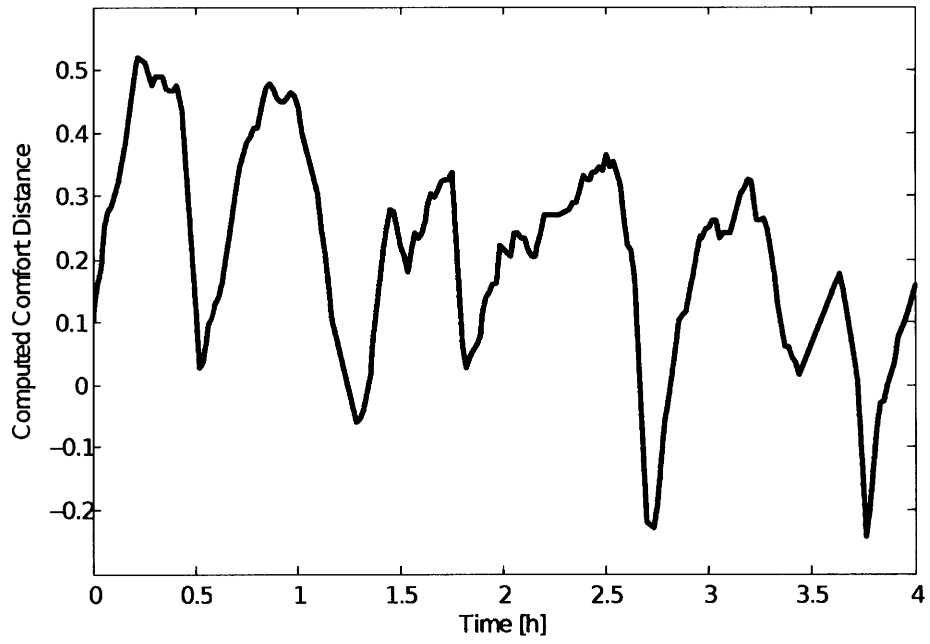


Figure 6.3.12: Example Comfort Distance Used as Control Signal

The reasons for the control system's unresponsiveness can also be seen in its attempt to arbitrate between users. A plot of two users' computed comfort distances can be seen in Figure 6.3.13, along with the associated VAV damper air flow, to indicate the control system's actions. The comfort distances are averaged over 20 samples to make the control inputs more clear. When the first user enters the room, it can be seen that the damper shuts off the airflow, as the user is cold. When the second user enters, and is hot, the system increases the air flow in an attempt to average the comfort of the two users. The damper is full open, and the comfort distances change only slightly, although in the correct directions. When each user leaves, the system again compensates correctly, but the damper opening and closing times are usually around an hour. Note that the entire time scale for this plot

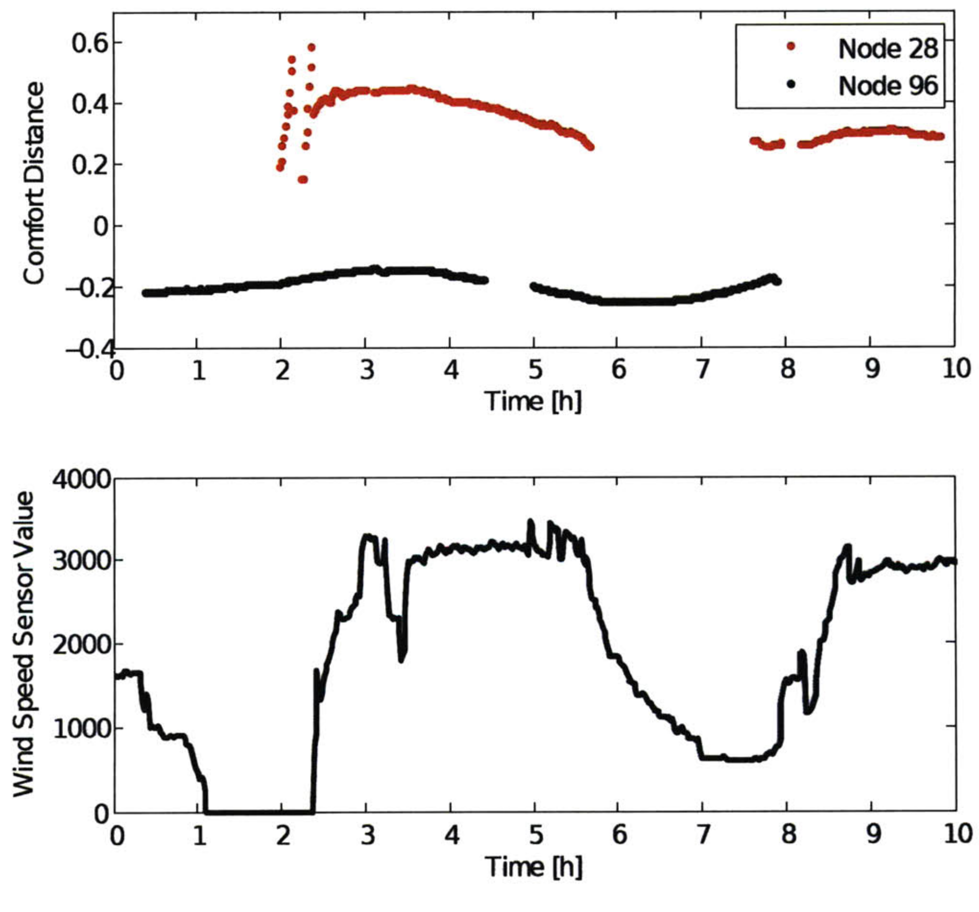


Figure 6.3.13: Example Arbitration Between Users

is ten hours, an extremely long time for occupants to stay in one place.

Despite this long response time, users still felt that the system was responding to their needs, and trying to average between all occupants. Survey Responses to the question, "I believe the personalized comfort system is doing a good job of balancing the thermal comfort needs of all the people in my workspace.", can be see in Figure 6.3.14. The majority of participants responded favorably to this question, and from personal recollection, users would often look forward to their officemates leaving, so they could have a more comfortable environment.

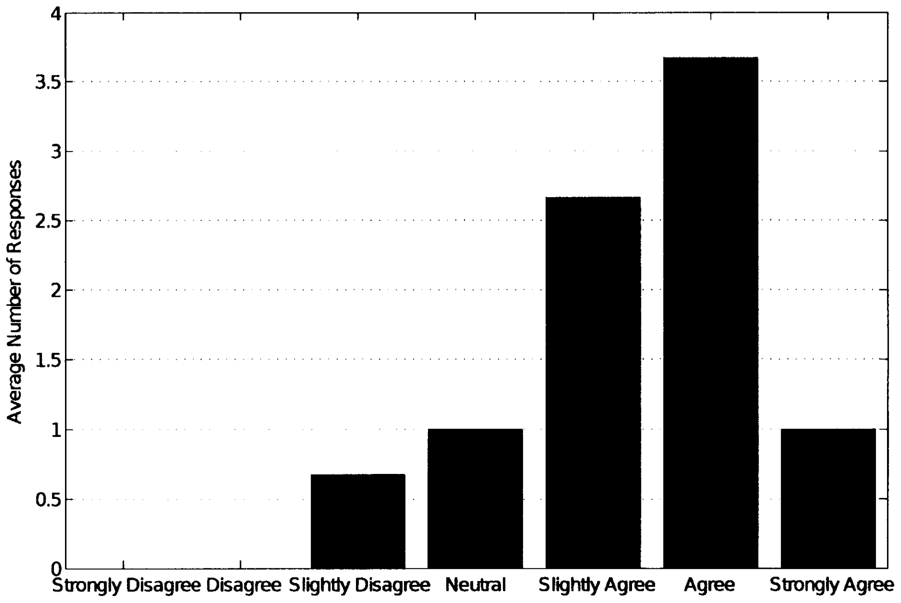


Figure 6.3.14: Average Responses to Survey Question: "I believe the personalized comfort system is doing a good job of balancing the thermal comfort needs of all the people in my workspace."

A more objective measure of this comfort balancing can be seen in the average room temperatures and air flows for each user. A tablature of these for Week Two through Week Four, grouped by room, can be found in Table 6.3.2. In general, the lower air temperatures correlate to more air usage, with exceptions for those users who worked more evening hours, such as 24, 28, and 104. The self-assessed comfort level, averaged over this period, is also given, and the colder user temperatures are consistently accompanied by the warmer occupants, except for Room 40, which was in the public space and only partially under control of the sytem. This indicates that the room is attempting to set the appropriate temperatures for each user, and is allowed to do so when others are not present.

Room	User	Average Temperature [°F]	Average Air Usage [m^2/s]	Average Discomfort
48	4	73.298	1.9735	+1.520
	20	73.668	1.8374	+1.167
44	8	73.838	1.3033	−0.083
	88	72.764	1.8579	+0.833
40	24	72.675	1.4689	−0.333
	84	73.072	1.8694	+0.003
32	28	74.580	1.0248	−0.143
	96	74.576	2.0969	+0.167
	104	73.53	2.0488	+1.020
196	76	73.026	1.8909	+0.400

Table 6.3.2: Average Room Temperature, Air Usage, and Discomfort by User: Week Two – Week Four

The final metric of personal comfort is the number of button presses during the experiment. Each user had the ability to register his or her personal comfort at any point in time via the portable nodes. They had a button for each of hot, cold, and neutral. By the beginning of Phase Two, the occupants had been using the buttons for a month, so the initial novelty factor had worn off. The number of 'hot' and 'cold' button presses should, therefore, give a good indication of how often a user was feeling uncomfortable. These button presses are averaged over the amount of time a user was wearing the portable node, as a means of normalizing the data. In general, more or less button presses between users is not a good indicator of relative comfort, as different users may have different habits. But it is a good indicator of how a single user's comfort progressed with time.

Figures 6.3.15 – 6.3.16 show the number of discomfort button presses for each user over the duration of the experiment. Figure 6.3.15 shows all button presses, whereas Figure 6.3.16 only shows button presses when the user had been active for at least 15 minutes. This is done to window out the temporary discomfort of first entering a different thermal environment. Neutral button presses are not plotted, as the users were told for Phase Three that the

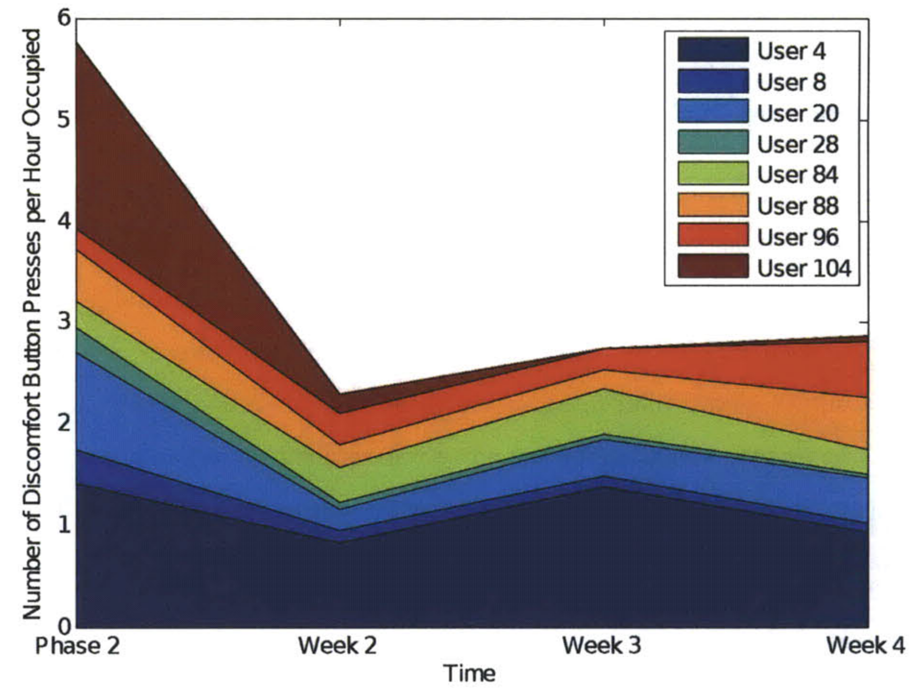


Figure 6.3.15: Discomfort Button Presses by User

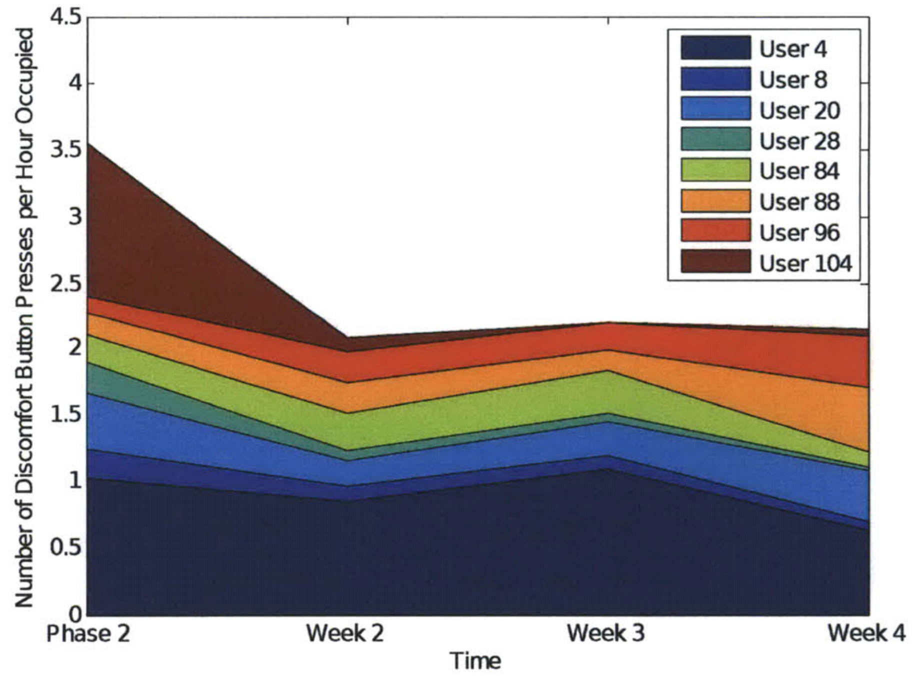


Figure 6.3.16: Discomfort Button Presses by User: Entry Presses Removed

neutral button presses would not affect the environment, and they subsequently decreased. Furthermore, Nodes 24 and 76 are excluded, as the user of Node 24 changed between Phase Two and Phase Three, and User 76 was not present for the majority of Phase Three.

It can be seen that the total number of button presses decreased in each case, but this result is mostly due to the actions of User 104, who essentially stopped pressing buttons during Phase Three. Even with this user removed, the average number of button presses decreased 14 percent between Phase Two and Phase Three, when entry presses are removed. This is a more accurate metric than total button presses, as the users were told during Phase Three that the system would ignore any entry presses. This is not a dramatic change, but it is significant, due to the fact that there was greater incentive during Phase Three to press buttons, as this would actually change environmental conditions.

Users were clearly being made more comfortable by the Personalized Comfort Control System, despite the slow control responses. This can be attributed to the building becoming aware of the desired comfort levels of its occupants, and occupants believing that the building was working to optimize their comfort. These are in direct contrast to a standard control system, and were reinforced by the actions of the building. Even though it would take hours for a user's measured comfort distance to change after a control response, the fact that the building recognized the discomfort and acted upon it was noticed by the occupant, and influenced his or her perception of comfort. Both the sound of the damper motor turning, and the associated rush of air, were strong cues that the system was working, and a welcome sign that comfort was on its way.

Chapter 7

Conclusions

This dissertation has presented a novel method of building comfort control, focused on the occupant. Special sensing, communication, and actuation hardware was developed to locate users in the building, and measure their comfort directly on the body. These comfort signals were then used to control the air-conditioning system, and direct air flow where it was needed, when it was needed. A three month study of the system was conducted, with four weeks of this experimental control strategy compared to the previous four weeks of standard control. An improvement in both comfort and energy usage is shown as a result of Personalized Comfort Control.

Although the energy savings vary depending upon assumptions, with estimates in the range of 3.2 to 24 percent, it is clear that the experimental system reduced chilled air usage. It accomplished this by only cooling areas as much as required to maintain occupant comfort, and not cooling areas when occupants were not present. It also worked to maintain room temperatures at an equitable level for all involved. It was able to do all of this as a result of a low-power, wrist worn, temperature, humidity, light, and activity sensor, which made the building aware of its occupants' state via these sensor data and simple 'hot' and 'cold' button presses.

The energy savings were the direct result of improving user comfort. In a well-functioning building, it is not the case that the temperature is either too hot or too cold, but rather that it is too hot or too cold for particular individuals: the air is not being distributed effectively. A number of personal cooling systems are commercially available, but these are expensive, and in some cases impossible to install. A solution which is retrofittable to older, existing buildings is necessary, as this is not only the largest portion of existing building stock, but also the least efficient and effective. This system accomplishes this by using wireless sensors and actuators that create their own communication network upon deployment.

In the process of creating this system, a number of results were obtained which are not strictly quantifiable. Through conversations with test subjects, observations on their behaviour, and personal experience in the controlled environment, a model of user comfort and expectations was developed, influencing a large portion of this work. The remainder of this chapter will focus on these ideas, and qualitatively evaluate how well they were implemented. These concepts will then be projected forward onto the larger issues involved in building automation.

7.1 Design Strategies for Comfort Control

An efficient, active building does not do tasks the occupants can already do themselves, but rather, it is capable of doing things the occupants can not do. There are many tasks which require an expert's knowledge to perform correctly: even thermostat setting has been shown to be too complicated in some cases. To no longer require a professional's assistance not only reduces maintenance costs, but it also increases the likelihood that the building is operating correctly, as the threshold for requesting repairs is often quite high.

The main task this system performed was determining comfort level. Although the occupant is currently capable of doing this, acting upon that comfort level is not so simple. If comfort was as easy as a fixed temperature setting, the thermostat model would work much better than it does. Instead, the comfort of an individual will vary greatly depending upon activity level, location, time of day, and a host of other factors. For this reason, a proximate sensing device is critical to understanding the user.

There are many ways of accomplishing this proximate sensing, and the wrist-worn version shown here is perhaps the least favorable. Because of its size and materials, it proved to be distracting, and most users were very happy to be free of it at the end of the experiment. Ultimately, the form factor could change, or it could be incorporated into another wearable device, like a wrist-watch, but this may not be necessary. The cost of sensing, temperature sensing in particular, has become increasingly inexpensive, and the power requirements are so low that solar powered versions are currently available. It is quite possible to distribute dozens of these per room, such that the occupant has a sensor nearby, regardless of position. A portable node of some kind will still most likely be necessary, as occupant identity, location, activity level, and comfort status will need to be assessed, but these are things that modern cellphones could be programmed to do. These are also bits of information that the user might want to maintain personally, for privacy reasons.

This dense distribution of sensors throughout a room could remove some of the problems encountered during this thesis. For example, false readings from nearby, warm electronic components could be reduced via outlier elimination and sensor data fusion. It might also liberate the user from the need for cumbersome, on-body temperature sensing. The wrist-worn device used here, which was nicknamed 'the shackle' by its users, was unobtrusive in comparison to direct thermal contact devices, which would be preferred for accurate measurement. There is also variation in how occupants' skin temperatures vary with comfort level, with the system working well for those who had large fluctuations. Ultimately, giving users a choice of how to communicate with their environment will improve perceptions of the system greatly.

Besides its lack of thermal contact with the body, the portable node failed in another important way: its inability to convey the magnitude of user discomfort. Perhaps the duration or pressure of the button press could indicate this information, which would be very helpful in developing a better comfort metric. One of the great promises of this system is the ability to make trade-offs between comfort and energy. A user could select a cost function that best represents his or her willingness to sacrifice short term comfort for long term financial gain. Without a better understanding of exactly how uncomfortable the user is, this sort of system would most likely just lead to frustration.

This is also equally important to the task of conflict resolution. Social pressures tend to dominate in determining office temperatures, with occupants having no reason to believe that the others are as uncomfortable as themselves. Arguments of an almost moral nature were common, such as "You should exercise more so you're not cold all the time." A comfort arbitration system offers the possibility of a far more democratic approach, where users believe that an unbiased critic is selecting the appropriate boundary. However, this only holds as long as these beliefs are reinforced by experience. In general, the users of the Personal Comfort System felt it was very equitable, although feedback showing the state of others' discomfort would make this situation even better.

The comfort metric employed here was the Fisher Discriminant, which worked surprisingly well for most users. In the cases where it did not work, the main source of failure was a result of the system never becoming cold enough for users to press the 'cold' button, making for an ambiguous comfort boundary. In retrospect, the KNN Distance Metric might have been a better choice, as it was far more robust to insufficient data. In fact, it has the ability to work in the complete absence of 'cold' data points. It also creates a more flexible boundary, closer to the labeled discomfort points. It is unclear, though, whether it produces a good distance metric. Perhaps a hybrid system which uses a Cartesian distance from the KNN determined decision boundary would make a more adaptable system. Ultimately, these are but a few of the many ways in which the machine learning aspect of this problem could be solved.

Work also needs to be done to determine the best method to remove outliers and data which are no longer appropriate. With a larger data set than the one used here, outlier detection would be much easier, and a larger feature space could be incorporated. A time-weighted metric, which leaves many more data points in the set, but gives preference to more recent events, might solve some of these issues. As adaptability is a prime concern, perhaps the user should be granted inputs other than just button presses, to communicate which scenarios are most relevant.

The main reason why the comfort system was not as effective as it could be, was a lack of responsiveness. Those users who had the best results were the ones who spent the most time in a single location. A more stable control input might help remedy this situation, but a larger issue is of concern: there is an energy penalty for fast response times. An example of this is the initial discomfort upon entry. It was noted that almost all occupants would be uncomfortable for approximately 15 – 30 minutes upon entering their office. After this time, they usually acclimated, and stated they were comfortable. Before the Personalized Comfort System was in place, occupants would often change the thermostat setting during this period, or open the dampers manually by disconnecting the control system. This not only wasted energy instantaneously, but also for a long time afterwards as the thermostat or damper was rarely set back to the initial setting. The experimental system saved energy by ignoring discomfort during this period, under the expectation that it would pass.

This idea is reinforced by the notion that occupants remember moments of discomfort far more greatly than the longer periods of comfort. Furthermore, once an occupant has crossed the comfort threshold, it is not merely a matter of lowering the temperature a small fraction to cross back over the boundary. Whether it is a result of psychological factors, or the thermal mass of the body, a hysteresis effect was noticed in the test subjects' impression of comfort, and a great deal more cooling was required to give them relief. It is in these periods that they will act in inefficient ways. With appropriate feedback to the user (as will be discussed in the next section), these moments can be transitioned without lowering temperature. The experimental system aimed to avoid these by using an extremely conservative setback control technique, which aimed to ensure that rooms were always within the realm of comfort before the user entered.

These energy saving techniques are further hindered by the fact that users needed greater cold discomfort before indicating discomfort to the system. From the Week Four Survey, users stated that they would require a discomfort level of -2.33, on average, to press the 'cold' button. This is a large difference from the $+1.44$ average hot discomfort level, which indicates that a slightly hot environment is far less tolerable to people than a slightly cold environment. Some users even stated that they preferred it cold, even though it was not quite as comfortable. This may be due to the novelty of a cold environment in summer, or the expectation that it will not be cold for long, but in either case, it is a strong argument for mediated comfort control, as it suggests users will set temperatures much lower than their upper comfort boundary, if they are able to do so.

Ultimately, it is the maintaining of this upper comfort boundary that saves the most energy. The system here did this unintentionally, as the response time was too slow to cool to the appropriate level. A more agile system would be able to meet this need, and reduce user frustration. Nevertheless, there will always be times of transition, or unexpected occurrences, and it is the ability of the building to communicate to the user what its objectives are that will determine the success of these systems.

7.2 Requirements of Automated Environments

As described in Section 2.4, this work is grounded in the concept that both occupant and building have much to gain by increased communication with each other. In order for this to happen effectively, the automation system must be robust to failure, and always provide basic functionality under all circumstances. Pre-programming from the user should be kept to a minimum, as these mappings will eventually become inappropriate, and the user may not have the time, knowledge, or inclination to input the correct settings. Most importantly, the control system should give appropriate feedback to the user as to its intentions. In this manner, the building and occupant will understand each other, and respond accordingly.

Ensuring basic functionality becomes a more difficult task as the complexity of a system increases. The control structure must be built upon a basic layer which only relinquishes control under very specific conditions. Constant checking for activity from sensors and actuators is needed to limit system responses to invalid data. Ultimately, the actuation hardware could contain this basic control foundation, allowing the entire system to function under the most severe network failures. Unfortunately, what is considered a basic function will most likely become more and more complex as building automation progresses, and this sort of localized control will no longer suffice.

The hardware for the Personalized Comfort System was relatively robust, with the wireless devices switching seamlessly between branch nodes when connections were lost. The only failures experienced were due to bad power supplies, and an RF unit lock-up problem. These are not fundamental limitations of the system, and merely represent common prototype issues. A larger problem is the robustness to RF interference, which could be accounted for with a channel hopping protocol and better packet identification. The control platform was very stable, with room occupancy detectors ensuring that the temperature was at a reasonable level, regardless of whether the occupant had a portable node. The window actuators had hardware over-ride buttons that allowed occupants to open and close them when desired, and these same controls could have been placed on the damper motors as well. This was not done for these experiments, in order to accurately assess the efficacy of the system. Despite this, the occupants rarely felt out of control of their environment.

This basic functionality came at the cost of a small amount of pre-programming. Mappings were made between dampers, rooms, and users, and PID control gains and temperature settings were preloaded. Although these represent a minimal set of information, the system could eventually learn all of these with time. This lack of pre-programming is critical to the acceptance of intelligent building automation systems. If a high level of expertise, information, or time is required to commission a control system, it will most likely not be done correctly, and will not remain stable after prolonged usage, as the pre-programmed conditions can not be guaranteed to persist. Ultimately, some amount of pre-assigned information will probably be desired by building owners and maintenance staff, but having this as an option, and not a requirement, will make these systems far easier to install and maintain.

Rather than fixing setpoints, the Personalized Comfort System learned user temperature preferences and room usage schedules, and adapted these with time to account for changes, or to improve comfort. The room usage schedules were automatically created by the system, but these could be augmented with personal information from online calendars or location information from cellphones. In contrast, the temperature preferences were inputted directly by the users via button presses. Although this worked well for this application, as the automated portions of our environments increase in complexity, the causality between user action and source of discomfort may no longer be clear to the system. The user will not tolerate having a button for every desire imaginable. More creative ways of communicating intent to the building will be required. Did he close the window because it was too windy, or because it was too cold? Did she shut off the light because she is going to sleep, or because she is going to watch a movie? The building will need to become more adept at making the correct guess in these conditions.

The penalty for wrong guesses is very high in building automation. If the occupant does not believe that the building is acting in a rational manner, or if the occupant can not formulate a simple model of the building's behaviour, then the entire system breaks down. The relationship becomes adversarial, as was witnessed in the thermostat usage during this experiment. Regardless of the fact that a thermostat setting of 55 °F is completely unreasonable, most of the thermostats were set at this temperature, because they were not behaving rationally: they were not lowering temperature when turned down. Out of frustration, the occupants began behaving irrationally in retaliation.

In the thermostat example, the users' behaviour had the negative effect of wasting energy in the evening hours. In a system with more user control, such as the one presented here, the lack of positive user beliefs could have more dramatic effects. If a user felt that the system was not being equitable, and was making the room too hot for their needs, they may begin hitting the 'hot' button in even the coldest conditions, in the hope that the boundary would move lower, and begin favoring their desires. This problem of 'gaming the system' is one of the many issues resulting from giving users unconstrained control. A fine balance must be found between granting enough control to convince users that the system is working to optimize their needs, and placing restrictions on temperature and energy excursions.

This ultimately becomes a problem of appropriate feedback. This is perhaps the most critical aspect of an automation system, and, unfortunately, this is where this dissertation succeeded the least. The reliance on linear control techniques led to slow response times to ensure stability over the unpredictable and wide ranging comfort inputs. Better control algorithms are needed to deal with these signals which are sometimes completely inaccurate.

The main feedback signal users received was an unintentional one: The whirring of the damper control motor. If not for this audible cue of changes occurring, users would not have been as tolerant of the hour-long lag on cool air arrival.

There is a trade-off between fast response time and energy consumption, so other methods besides a quick rush of air will most likely need to be employed to communicate to the occupants that the system is functioning. The motor sound worked in this case, but future solutions could incorporate a wider range of information, taking inspiration from the field of pursuasive computing, displaying the system's belief about the users' current comfort, and the action that is being taken. In cases where the building can not meet their needs, or comfort is intentionally being compromised for energy reasons, the users' expectations could be set appropriately, and they could plan accordingly. This would begin to open up the conversation between building and occupant, and would be a great improvement over the current deaf and mute thermostat.

Appendix A

Prototype Wearable Comfort System

Predictors for Human Temperature Comfort

Final Project for MAS.622J, completed December 15th, 2006.

A.1 Introduction

All homes and offices have environmental control systems. These usually consist of a source of hot or cold air, and a thermostat located nearby which turns the source on or off to regulate the temperature of the room. Unfortunately, the inhabitants of most spaces are not located near the thermostat, so the system does not adequately regulate the temperature for their comfort. The system also does not know whether or not a person is in the room, requiring that the ventilation run continuously, regardless of its demand. A more energy efficient and effective environmental control system can be achieved by placing the thermostat on the person, creating a more personalized and responsive space which knows when people enter a room, and what their current comfort level is.

The objective of this work is to determine the ability of a simple sensor system to predict human temperature comfort. Our perceived comfort level is a function of many factors: metabolic rate, stress, fatigue, activity level, etc. It is possible that there is no clear correlation between ambient air temperature and our perception of hot or cold. Exactly how does our body express its comfort level? Further more, for this wearable thermostat to be economically and socially acceptable, it must contain a minimum complement of sensors to keep costs down, and have these sensors located on the body in such a fashion as to not bother the user. In the ideal case, a single sensor would be worn as a button on the user's shirt, and transmit the user's preference wirelessly to the environmental control system.

A.2 Methodology

There are many possible locations and types of sensors which could be of use to the wearable thermostat. The locations chosen for this work are based upon places on the body which are currently ornamented: finger (rings), wrist (watches), neck (necklaces), chest (necklaces), shirt exterior (pendants). And, although there are many sensors which may be relevant, temperature and humidity sensors were chosen as the most likely candidates for predicting environmental comfort. To this end, a wearable temperature and humidity logging device (see Figure A.1) is developed which records the time of day, all sensor readings, and the user's comfort level. Every five minutes, a pager motor on the device vibrates, queuing the user to input his preference. One button represents 'hot' (+1), the other 'cold' (−1), and both pressed at the same time represents 'neutral' (0). 'hot' and 'cold' are determined by the user as states where, if given the option, an outer layer of clothing would be added or removed to help improve thermal comfort. A sample of the raw sensor data can be seen in Figure A.2. The system is worn and trained for a single user, as thermal comfort patterns are different for each person, and a wearable thermostat would have to learn the preferences of its owner.

A.3 Hardware

The core of the sensing system (see left image in Figure A.1) is based upon the CARGONET [85] environmental data logging board developed by Mateusz Malinowski. It has a TI MSP430 low power micro controller which communicates with the sensors, logs the data, and uploads these data to a computer via USB. It also has a real time clock for logging time and waking up the system from sleep mode every five minutes. Finally, it has an on-board temperature and humidity sensor. All sensors in the system are the Sensirion SHT15 (see right image in Figure A.1), which is extremely small, has a fast response time ($\leq$4 s), combines both humidity and temperature sensing in one package, and has humidity accuracy of $\pm$2% RH and temperature accuracy of $\pm$.3 °C. Four of these sensors are tethered via ribbon cable to be mounted to the body at the aforementioned locations with medical tape, and the fifth sensor is located on the board to gather ambient air temperature at the user's location. This gives a total of 10 sensors at five locations.

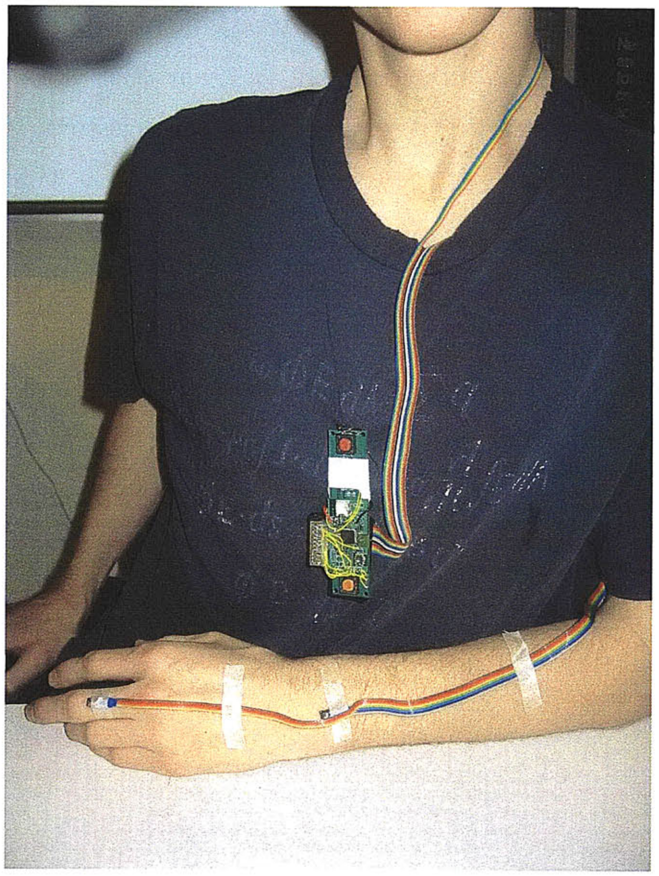


Figure A.1: Sensor System as Worn by User

A.4 Data Analysis

Data was collected for four days, giving 360 data points. Given the large dimensionality of the data set, this is a relatively small set of data points. To help reduce non-representative results given this small data set, the results are averaged over the entire data set. This is done by dividing the data into ten random sets without replacement. The algorithms are then trained on nine of these sets and tested on the remaining one. This is then repeated for all ten permutations of the sets, and the testing results are averaged over these permutations. The time data is not used, as the moving between hot and cold environments required to gather enough data points in each class produced a time series which does not relate to the current level of comfort.

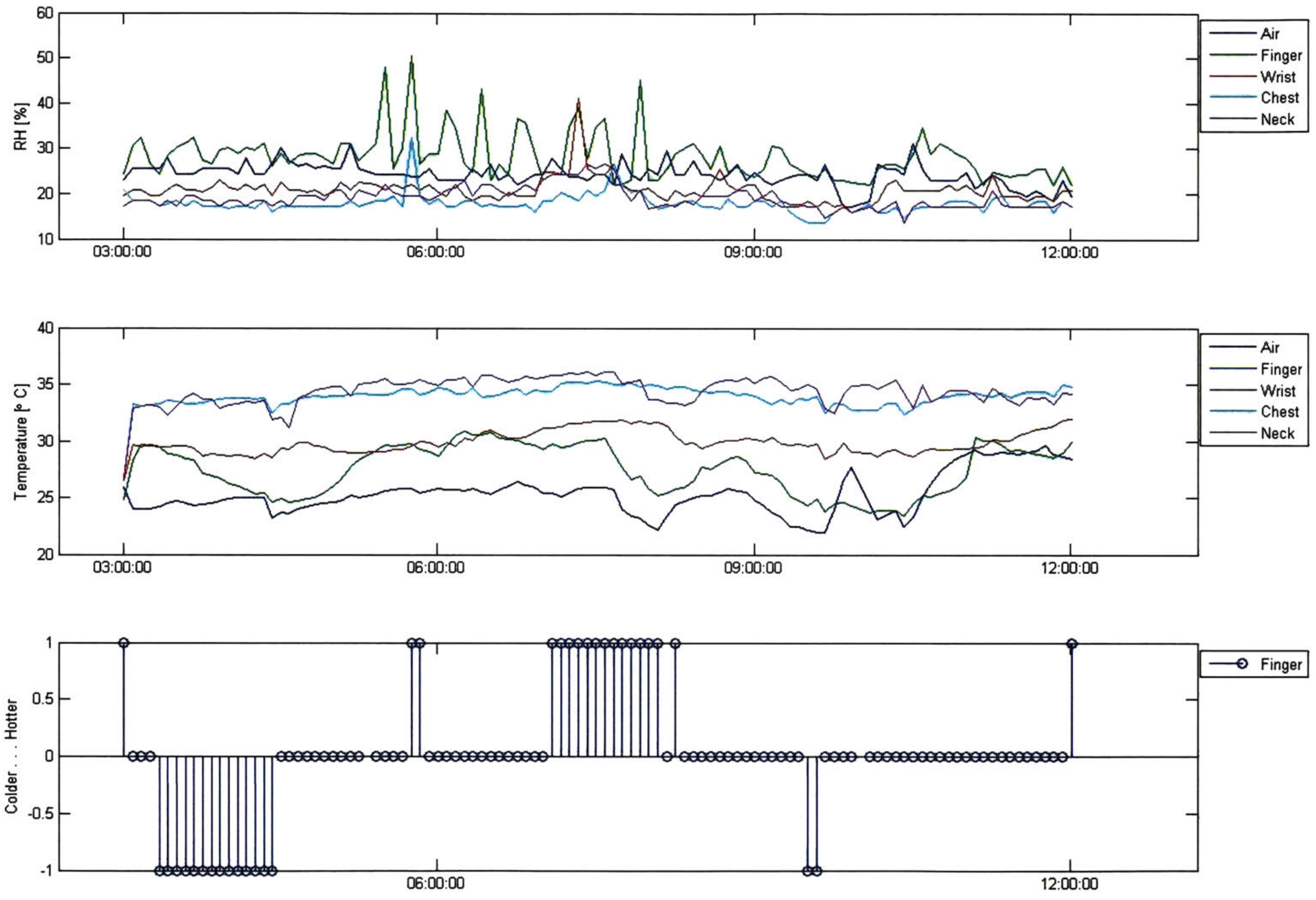


Figure A.2: Raw Data Logged by Sensors: Temperature, Humidity, Comfort

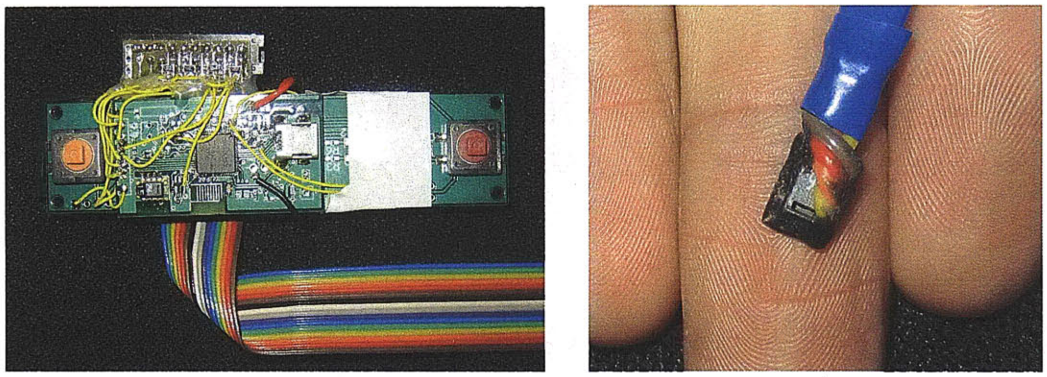


Figure A.1: Data Logging Board (left) and SHT15 Sensor (right)

Two different algorithms are chosen to test the ability of the system to predict the thermal comfort of the user. The first was a Gaussian model which simply took the mean and covariance of each of the sets of the training data marked by the user as hot, cold, and neutral. The testing data points were then input to each of the models, and each testing data point would receive the class of the model which returned the maximum value. The accuracy of the system was determined by the total number of correct labellings, divided by the total number of testing data points. The second algorithm was a K-Nearest Neighbor algorithm, which was trained with a leave one out strategy. The tested data point would assume a class label which represents the average of the K nearest neighbors' class labels. Since the system rounded .5 up to 1 and $-.5$ down to -1, this broke ties by favoring either 'hot' or 'cold' over neutral, assuming that there would generally not be a tie between 'hot' and 'cold'. The value of K was chosen as the value, less than ten, which returned the highest accuracy on the training data, when trained on all possible permutations of which data point was left out. K was chosen to be less than ten to keep processing time down, as ten was the maximum value observed for a few runs which were allowed to test values of K up to the number of data points in the training set.

A.5 Results

These two algorithms are then tested on all possible sensor combinations. The top ten sensor combinations for each quantity of sensors are returned, along with the accuracies associated with these combinations (see Table A.2 through Table A.11). A plot of the maximum accuracy versus sensor quantity can be seen in Figure A.1. The K-Nearest Neighbor algorithm is more accurate than the Gaussian model, although they both show a maximum accuracy with seven sensors. The dotted line in Figure A.1 shows the maximum accuracy achievable with each model given the number of body positions rather than the number of sensors. For example, 1 represents one sensor at one point on the body, 2 represents two sensors at the same location on the body, and 7 represents seven sensors at 4 locations on the body.

Sensor Number	Sensor Location	Sensor Type
1	hand	humidity
2	hand	temperature
3	wrist	humidity
4	wrist	temperature
5	chest	humidity
6	chest	temperature
7	kneck	humidity
8	kneck	temperature
9	air	humidity
10	air	temperature

Table A.1: Sensor Identification Number, Placement, and Type

From Table A.2 it can be seen that the wrist temperature sensor is the best single sensor, with the chest humidity sensor being the next best sensor to add. The chest temperature sensor is the third best sensor to add, with minimal improvements in accuracy from adding more sensors. Both algorithms concur on which sensors have the strongest correlation with thermal comfort.

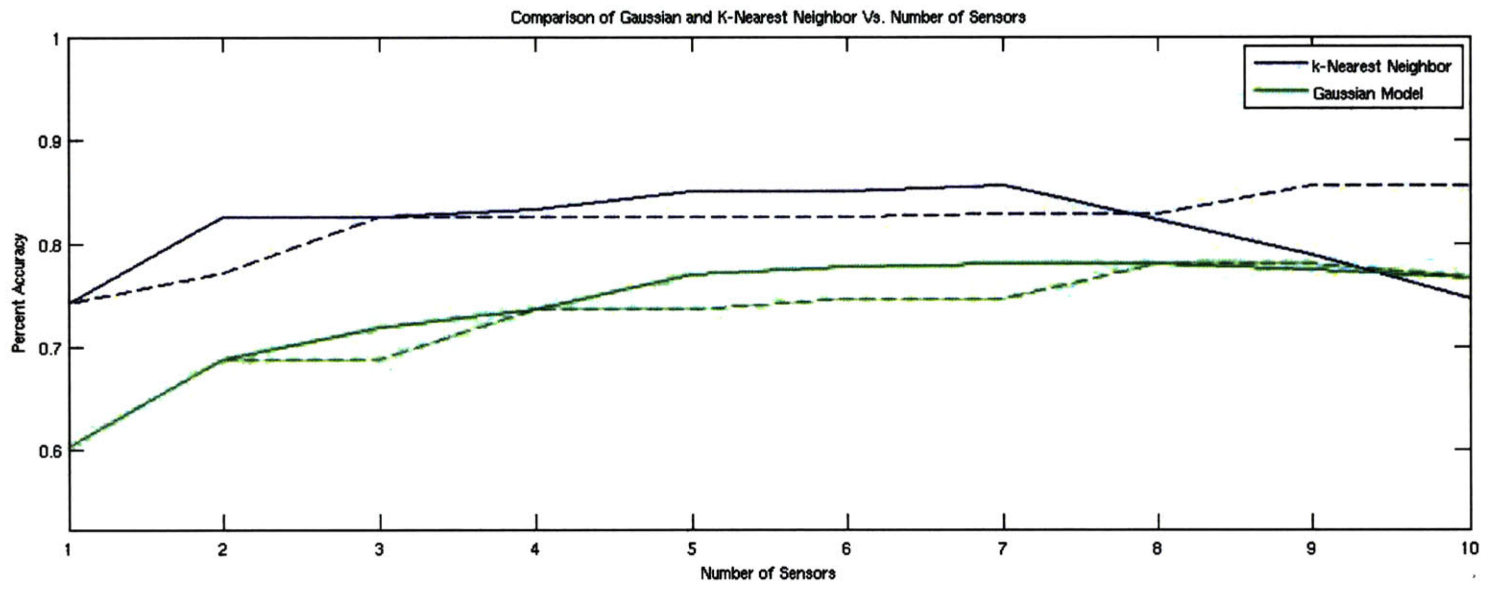


Figure A.1: KNN and Gaussian Accuracies versus Number of Sensors

Sensor Number	KNN Accuracy	Sensor Number	Gaussian Accuracy
8	0.6028	9	0.3333
7	0.6250	2	0.3417
9	0.6417	3	0.3667
1	0.6583	10	0.3972
10	0.6611	8	0.3972
6	0.6806	6	0.4361
2	0.6833	7	0.4417
3	0.6861	5	0.5250
5	0.7222	1	0.5806
4	0.7417	4	0.6028

Table A.2: KNN and Gaussian Accuracies for 1 Sensor

Sensor Numbers	KNN Accuracy	Sensor Numbers	Gaussian Accuracy
4 7	0.7556	1 9	0.6000
4 9	0.7556	4 9	0.6083
5 9	0.7583	3 4	0.6250
2 4	0.7583	4 7	0.6333
3 4	0.7611	1 4	0.6417
4 10	0.7639	5 8	0.6472
5 6	0.7722	4 5	0.6528
4 8	0.7806	4 8	0.6778
4 6	0.8194	4 6	0.6861
4 5	0.8250	5 6	0.6889

Table A.3: KNN and Gaussian Accuracies for 2 Sensors

Sensor Numbers	KNN Accuracy	Sensor Numbers	Gaussian Accuracy
2 4 5	0.7806	3 5 6	0.6889
2 3 5	0.7861	1 5 6	0.6944
4 5 9	0.7861	2 4 6	0.6944
2 4 6	0.7944	5 6 8	0.6944
3 4 5	0.7972	3 4 6	0.6944
4 5 8	0.8056	1 4 6	0.6972
4 5 10	0.8083	4 6 9	0.6972
4 6 10	0.8083	4 6 8	0.7000
4 6 8	0.8111	4 6 7	0.7167
4 5 6	0.8250	4 5 6	0.7194

Table A.4: KNN and Gaussian Accuracies for 3 Sensors

Sensor Numbers	KNN Accuracy	Sensor Numbers	Gaussian Accuracy
4 5 7 10	0.8028	1 4 6 8	0.7194
4 7 9 10	0.8083	2 4 5 6	0.7222
2 4 6 10	0.8083	1 4 6 9	0.7222
4 5 9 10	0.8083	1 4 5 6	0.7250
2 4 6 8	0.8139	1 4 6 7	0.7250
2 4 5 7	0.8139	3 4 6 8	0.7250
2 4 5 8	0.8139	3 4 5 6	0.7278
3 4 5 10	0.8194	4 6 7 8	0.7278
2 4 5 6	0.8222	4 6 8 9	0.7361
4 5 6 10	0.8333	4 5 6 8	0.7361

Table A.5: KNN and Gaussian Accuracies for 4 Sensors

Sensor Numbers	KNN Accuracy	Sensor Numbers	Gaussian Accuracy
2 4 5 6 8	0.8194	4 6 7 8 9	0.7333
2 4 5 8 10	0.8194	1 3 4 5 6	0.7361
4 5 6 7 10	0.8194	1 4 6 7 8	0.7389
4 5 6 8 9	0.8194	4 5 6 7 8	0.7389
4 5 7 8 10	0.8194	1 3 4 6 8	0.7417
2 3 5 6 10	0.8222	2 4 5 6 8	0.7444
4 5 6 8 10	0.8278	2 3 4 5 6	0.7472
2 3 4 5 8	0.8306	3 4 5 6 8	0.7472
2 4 5 7 8	0.8333	1 4 6 8 9	0.7500
2 4 5 7 10	0.8500	1 4 5 6 8	0.7694

Table A.6: KNN and Gaussian Accuracies for 5 Sensors

Sensor Numbers	KNN Accuracy	Sensor Numbers	Gaussian Accuracy
2 4 5 6 9 10	0.8167	4 5 6 7 8 10	0.7500
4 5 7 8 9 10	0.8167	1 3 4 6 8 9	0.7528
2 3 4 5 8 10	0.8194	1 4 6 7 8 9	0.7528
4 5 6 8 9 10	0.8194	3 4 5 6 8 9	0.7528
2 4 5 6 7 8	0.8222	1 2 4 5 6 8	0.7556
2 3 4 5 6 10	0.8361	1 3 4 5 6 8	0.7583
2 4 5 7 8 10	0.8361	1 4 5 6 8 9	0.7639
2 3 4 5 6 8	0.8389	1 4 5 6 7 8	0.7750
2 4 5 6 7 10	0.8500	2 4 5 6 7 8	0.7778
2 4 5 6 8 10	0.8500	1 4 5 6 8 10	0.7778

Table A.7: KNN and Gaussian Accuracies for 6 Sensors

Sensor Numbers	KNN Accuracy	Sensor Numbers	Gaussian Accuracy
2 3 4 5 7 8 10	0.8000	1 2 3 4 6 8 9	0.7611
2 3 4 5 6 8 9	0.8000	1 4 6 7 8 9 10	0.7611
4 5 6 7 8 9 10	0.8056	3 4 5 6 7 8 10	0.7611
2 3 4 6 7 9 10	0.8083	3 4 5 6 8 9 10	0.7667
2 4 5 7 8 9 10	0.8083	2 4 5 6 7 8 10	0.7694
2 3 4 5 6 8 9	0.8111	1 3 4 5 6 7 8	0.7694
3 4 5 6 7 8 10	0.8278	1 2 3 4 5 6 8	0.7722
2 3 4 5 6 7 10	0.8306	1 4 5 6 7 8 10	0.7750
2 4 5 6 7 8 10	0.8389	1 2 4 5 6 8 10	0.7778
2 3 4 5 6 8 10	0.8556	1 2 4 5 6 7 8	0.7806

Table A.8: KNN and Gaussian Accuracies for 7 Sensors

Sensor Numbers	KNN Accuracy	Sensor Numbers	Gaussian Accuracy
2 3 4 5 6 7 9 10	0.7639	1 3 4 6 7 8 9 10	0.7639
1 2 3 5 6 7 8 10	0.7694	1 2 3 4 6 8 9 10	0.7667
2 3 4 5 6 7 8 9	0.7750	2 3 4 6 7 8 9 10	0.7667
2 3 4 5 7 8 9 10	0.7778	3 4 5 6 7 8 9 10	0.7694
3 4 5 6 7 8 9 10	0.7833	1 2 4 6 7 8 9 10	0.7694
1 2 3 4 5 6 7 10	0.7889	1 3 4 5 6 7 8 9	0.7750
1 2 3 4 5 7 8 10	0.7889	2 3 4 5 6 7 8 10	0.7750
2 4 5 6 7 8 9 10	0.8028	1 2 4 5 6 7 8 10	0.7778
2 3 4 5 6 7 8 10	0.8167	1 3 4 5 6 7 8 10	0.7778
2 3 4 5 6 8 9 10	0.8222	1 2 3 4 5 6 7 8	0.7806

Table A.9: KNN and Gaussian Accuracies for 8 Sensors

Sensor Numbers	KNN Accuracy	Sensor Numbers	Gaussian Accuracy
1 2 3 4 6 7 8 9 10	0.7194	1 2 3 5 6 7 8 9 10	0.7111
1 2 3 5 6 7 8 9 10	0.7278	1 2 3 4 5 7 8 9 10	0.7333
1 2 3 4 5 6 7 9 10	0.7333	1 2 3 4 5 6 7 9 10	0.7333
1 2 3 4 5 6 7 8 9	0.7361	1 2 3 4 5 6 8 9 10	0.7583
1 2 3 4 5 7 8 9 10	0.7389	1 2 4 5 6 7 8 9 10	0.7611
1 3 4 5 6 7 8 9 10	0.7444	1 2 3 4 6 7 8 9 10	0.7694
1 2 3 4 5 6 8 9 10	0.7583	2 3 4 5 6 7 8 9 10	0.7722
1 2 4 5 6 7 8 9 10	0.7639	1 2 3 4 5 6 7 8 9	0.7722
2 3 4 5 6 7 8 9 10	0.7806	1 2 3 4 5 6 7 8 10	0.7722
1 2 3 4 5 6 7 8 10	0.7889	1 3 4 5 6 7 8 9 10	0.7750

Table A.10: KNN and Gaussian Accuracies for 9 Sensors

Sensor Numbers	KNN Accuracy	Sensor Numbers	Gaussian Accuracy
1 2 3 4 5 6 7 8 9 10	0.7472	1 2 3 4 5 6 7 8 9 10	0.7667

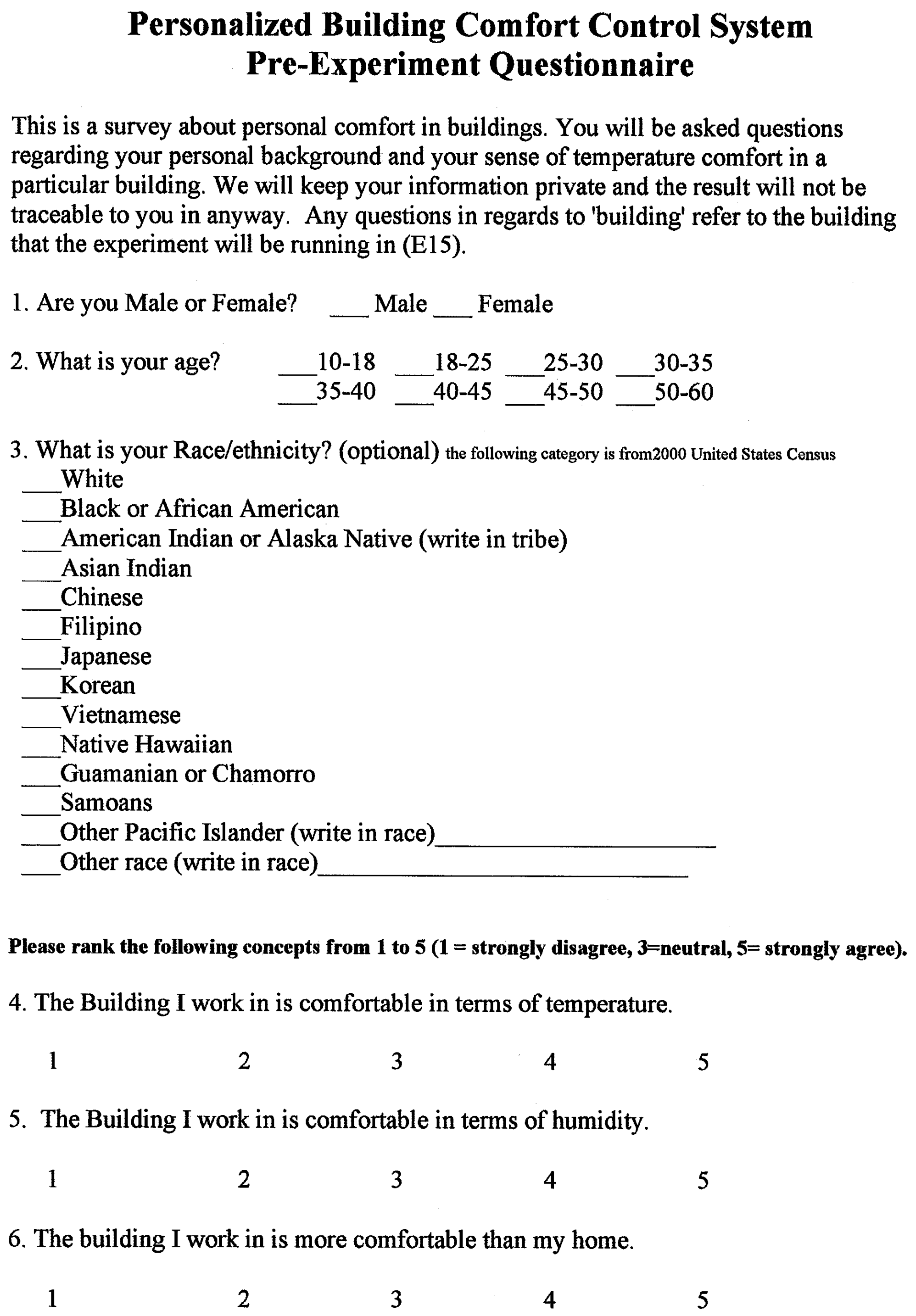
Table A.11: KNN and Gaussian Accuracies for 10 Sensors

A.6 Conclusions and Future Work

These preliminary results show that a wearable thermostat has the possibility of being accurate enough to regulate a building's ventilation system for a user's comfort. Although the system only reached a maximum accuracy of 85 percent, further work which took into account the time series data could possibly improve upon this. Also, gains could be made by changing the voting algorithm for the K-Nearest Neighbor algorithm. Since the human body works to regulate its own temperature, it is more likely that a user will be 'neutral' rather than 'hot' or 'cold', so perhaps the algorithm should favor 'neutral' classification.

Appendix B

Survey Results



Personalized Building Comfort Control System
Pre-Experiment Questionnaire

This is a survey about personal comfort in buildings. You will be asked questions regarding your personal background and your sense of temperature comfort in a particular building. We will keep your information private and the result will not be traceable to you in anyway. Any questions in regards to 'building' refer to the building that the experiment will be running in (E15).

1. Are you Male or Female? ___ Male ___ Female

2. What is your age? ___10-18 ___18-25 ___25-30 ___30-35
___35-40 ___40-45 ___45-50 ___50-60

3. What is your Race/ethnicity? (optional) the following category is from2000 United States Census

___White
___Black or African American
___American Indian or Alaska Native (write in tribe)
___Asian Indian
___Chinese
___Filipino
___Japanese
___Korean
___Vietnamese
___Native Hawaiian
___Guamanian or Chamorro
___Samoans
___Other Pacific Islander (write in race)____________________
___Other race (write in race)__________________________

Please rank the following concepts from 1 to 5 (1 = strongly disagree, 3=neutral, 5= strongly agree).

4. The Building I work in is comfortable in terms of temperature.

1 2 3 4 5

5. The Building I work in is comfortable in terms of humidity.

1 2 3 4 5

6. The building I work in is more comfortable than my home.

1 2 3 4 5

Figure B.1: Entrance Survey: Page One

7. I feel in control of my local comfort level in my building.

1 2 3 4 5

8. I believe the building I work in is efficient in terms of the energy used to control the building temperature.

1 2 3 4 5

9. The building is often too cold for me.

1 2 3 4 5

10. The building is often too hot for me.

1 2 3 4 5

11. The building is more often too hot than too cold.

1 2 3 4 5

12. The building is more often too cold than too hot.

1 2 3 4 5

13. I would like more control over my local temperature in my building.

1 2 3 4 5

14. My officemate and I often disagree over whether its too hot or too cold.

1 2 3 4 5

15. I often wear two or more layers of clothing during the summer in my building in order to keep warm.

1 2 3 4 5

16. I often open a window to try to cool down my office.

1 2 3 4 5

17. Overall, I am satisfied with the comfort level in my office.

1 2 3 4 5

Figure B.2: Entrance Survey: Page Two

Question Number	Male	Female
1	9	2

Table B.1: Number of Responses for Question 1 on Entrance Survey

Question Number	18 – 25	25 – 30	30 – 35	50 – 60
2	4	4	2	1

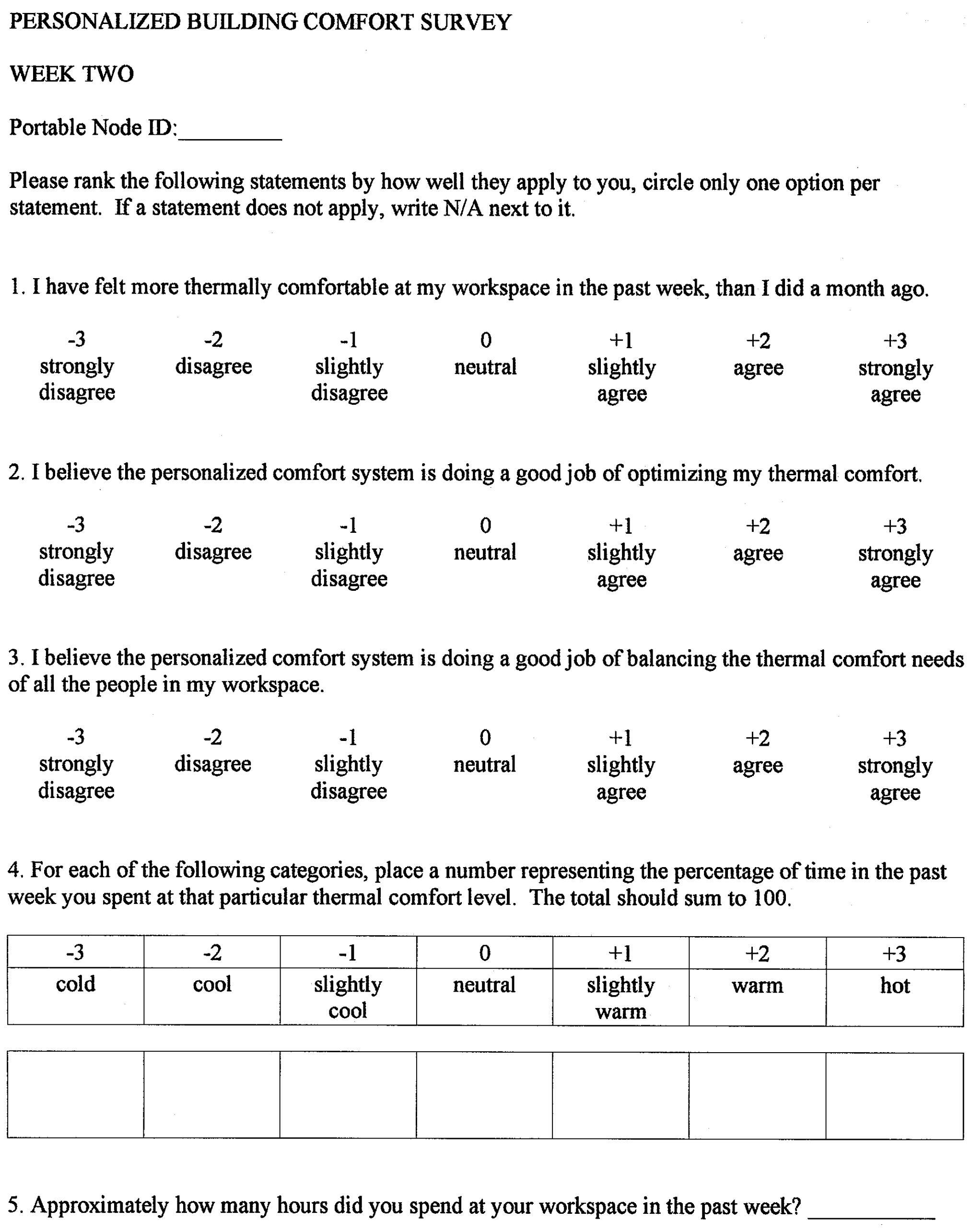
Table B.2: Number of Responses for Question 2 on Entrance Survey

Question Number	Chinese	White	Mixed
2	1	9	1

Table B.3: Number of Responses for Question 3 on Entrance Survey

Question Number	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
4	0	6	3	2	0
5	0	2	4	4	1
6	0	4	3	2	2
7	6	1	3	1	0
8	2	4	3	1	1
9	1	2	2	4	2
10	2	2	1	3	3
11	1	3	0	3	4
12	2	6	0	2	1
13	0	1	0	3	7
14	0	2	2	4	2
15	4	4	0	1	2
16	0	0	0	3	1
17	2	2	4	3	0

Table B.4: Number of Responses for Questions 4 – 17 on Entrance Survey



PERSONALIZED BUILDING COMFORT SURVEY

WEEK TWO

Portable Node ID:_________

Please rank the following statements by how well they apply to you, circle only one option per statement. If a statement does not apply, write N/A next to it.

1. I have felt more thermally comfortable at my workspace in the past week, than I did a month ago.

-3	-2	-1	0	+1	+2	+3
strongly disagree	disagree	slightly disagree	neutral	slightly agree	agree	strongly agree

2. I believe the personalized comfort system is doing a good job of optimizing my thermal comfort.

-3	-2	-1	0	+1	+2	+3
strongly disagree	disagree	slightly disagree	neutral	slightly agree	agree	strongly agree

3. I believe the personalized comfort system is doing a good job of balancing the thermal comfort needs of all the people in my workspace.

-3	-2	-1	0	+1	+2	+3
strongly disagree	disagree	slightly disagree	neutral	slightly agree	agree	strongly agree

4. For each of the following categories, place a number representing the percentage of time in the past week you spent at that particular thermal comfort level. The total should sum to 100.

-3	-2	-1	0	+1	+2	+3
cold	cool	slightly cool	neutral	slightly warm	warm	hot

5. Approximately how many hours did you spend at your workspace in the past week? __________

Figure B.3: Survey for Week 2 of Experimental Control

Node Number	Question Number			
	1	2	3	5
4	+1	+2	+1	30
8	+3	+3	+3	60
20	+3	+2	+2	35
24	+2	+2	0	50
28	0	+2	+1	100
76	+3	+3	0	12
84	0	0	+1	25
88	+2	+2	+2	25
96	+3	+2	+1	84
104	+2	+1	+2	68
Mean	+1.90	+1.90	+1.30	48.90

Table B.5: Survey Responses by Node Number For Week 2: Questions 1, 2, 3, 5

Node Number	Percentage of Time at Each Comfort Level						
	−3	−2	−1	0	+1	+2	+3
4	0	0	0	5	15	70	5
8	0	5	10	75	10	0	0
20	0	0	0	20	0	30	50
24	0	30	25	30	25	10	0
28	1	3	10	75	10	1	0
76	0	0	0	70	20	10	0
84	0	0	5	90	5	0	0
88	0	0	0	20	70	10	0
96	0	0	5	70	15	10	0
104	0	0	0	60	20	20	0
Mean	0.10	3.80	5.50	51.50	19.00	16.10	5.50

Table B.6: Survey Responses by Node Number For Week 2: Question 4

PERSONALIZED BUILDING COMFORT SURVEY

WEEK 3

Portable Node ID:_________

Please rank the following statements by how well they apply to you, circle only one option per statement. If a statement does not apply, write N/A next to it.

1. I have felt more thermally comfortable at my workspace in the past week, than I did last week.

-3	-2	-1	0	+1	+2	+3
strongly disagree	disagree	slightly disagree	neutral	slightly agree	agree	strongly agree

2. I believe the personalized comfort system is doing a good job of optimizing my thermal comfort.

-3	-2	-1	0	+1	+2	+3
strongly disagree	disagree	slightly disagree	neutral	slightly agree	agree	strongly agree

3. I believe the personalized comfort system is doing a good job of balancing the thermal comfort needs of all the people in my workspace.

-3	-2	-1	0	+1	+2	+3
strongly disagree	disagree	slightly disagree	neutral	slightly agree	agree	strongly agree

4. For each of the following categories, place a number representing the percentage of time in the past week you spent at that particular thermal comfort level. The total should sum to 100.

-3	-2	-1	0	+1	+2	+3
cold	cool	slightly cool	neutral	slightly warm	warm	hot

5. Approximately how many hours did you spend at your workspace in the past week? ___________

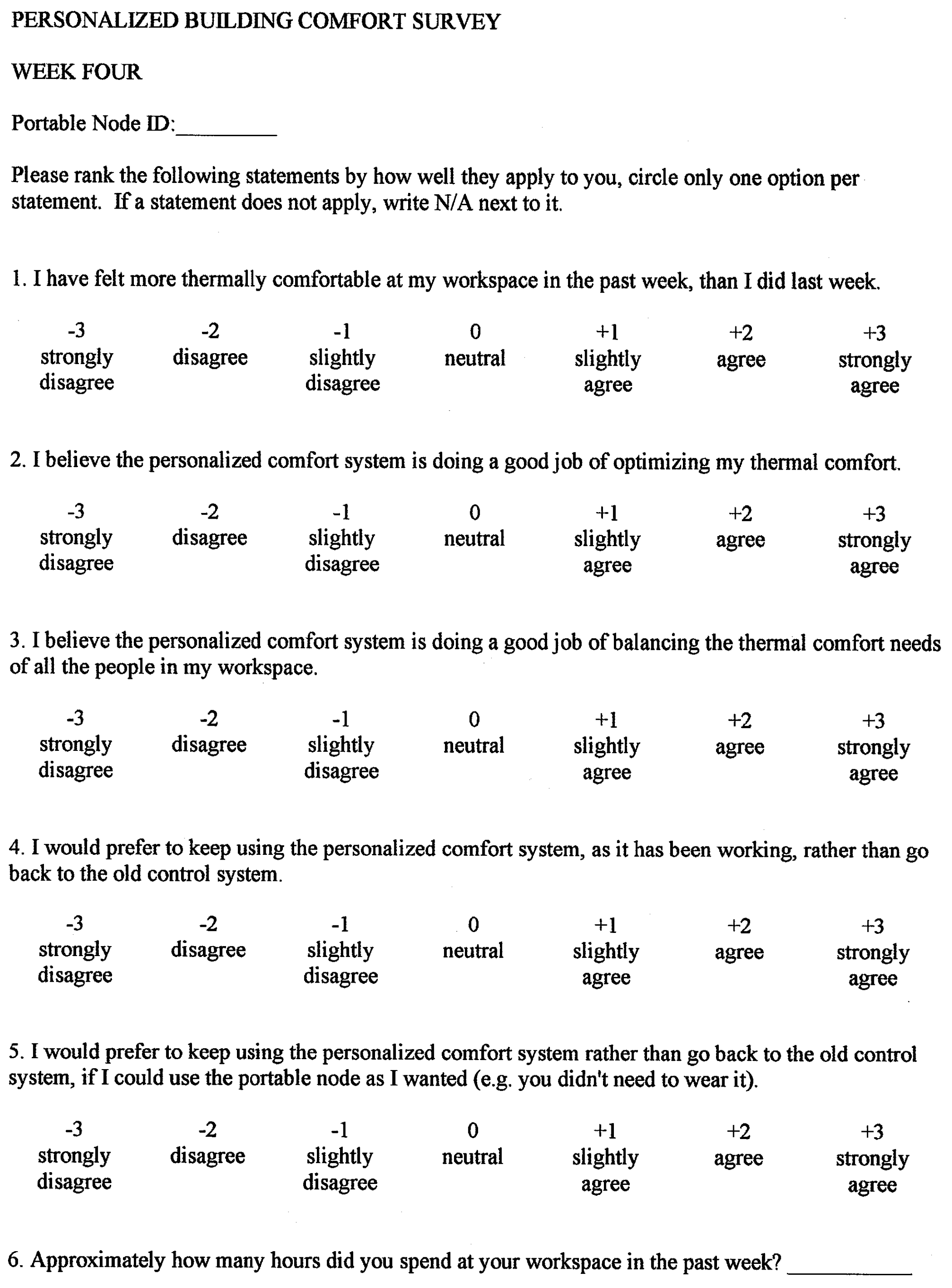
Figure B.4: Survey for Week 3 of Experimental Control

Node Number	Question Number			
	1	2	3	5
4	+1	+1	+1	30
8	+3	+3	+2	40
20	+2	+2	+2	40
24	+2	+2	0	40
28	+0	+2	+2	100
84	+2	+2	+1	25
88	+1	+2	+3	40
96	+2	+1	−1	45
104	+1	+2	+1	45
Mean	+1.56	+1.89	+1.22	45.00

Table B.7: Survey Responses by Node Number For Week 3: Questions 1, 2, 3, 5

Node Number	Percentage of Time at Each Comfort Level						
	−3	−2	−1	0	+1	+2	+3
4	0	0	0	20	20	45	15
8	0	0	5	95	0	0	0
20	0	0	10	25	25	25	15
24	0	10	20	60	10	0	0
28	1	8	15	70	5	1	0
84	0	0	2	90	8	0	0
88	0	0	0	35	60	5	0
96	0	0	5	75	15	5	0
104	0	0	0	30	30	30	10
Mean	0.11	2.00	6.33	55.56	19.22	12.33	4.44

Table B.8: Survey Responses by Node Number For Week 3: Question 4



PERSONALIZED BUILDING COMFORT SURVEY

WEEK FOUR

Portable Node ID:_________

Please rank the following statements by how well they apply to you, circle only one option per statement. If a statement does not apply, write N/A next to it.

1. I have felt more thermally comfortable at my workspace in the past week, than I did last week.

-3	-2	-1	0	+1	+2	+3
strongly disagree	disagree	slightly disagree	neutral	slightly agree	agree	strongly agree

2. I believe the personalized comfort system is doing a good job of optimizing my thermal comfort.

-3	-2	-1	0	+1	+2	+3
strongly disagree	disagree	slightly disagree	neutral	slightly agree	agree	strongly agree

3. I believe the personalized comfort system is doing a good job of balancing the thermal comfort needs of all the people in my workspace.

-3	-2	-1	0	+1	+2	+3
strongly disagree	disagree	slightly disagree	neutral	slightly agree	agree	strongly agree

4. I would prefer to keep using the personalized comfort system, as it has been working, rather than go back to the old control system.

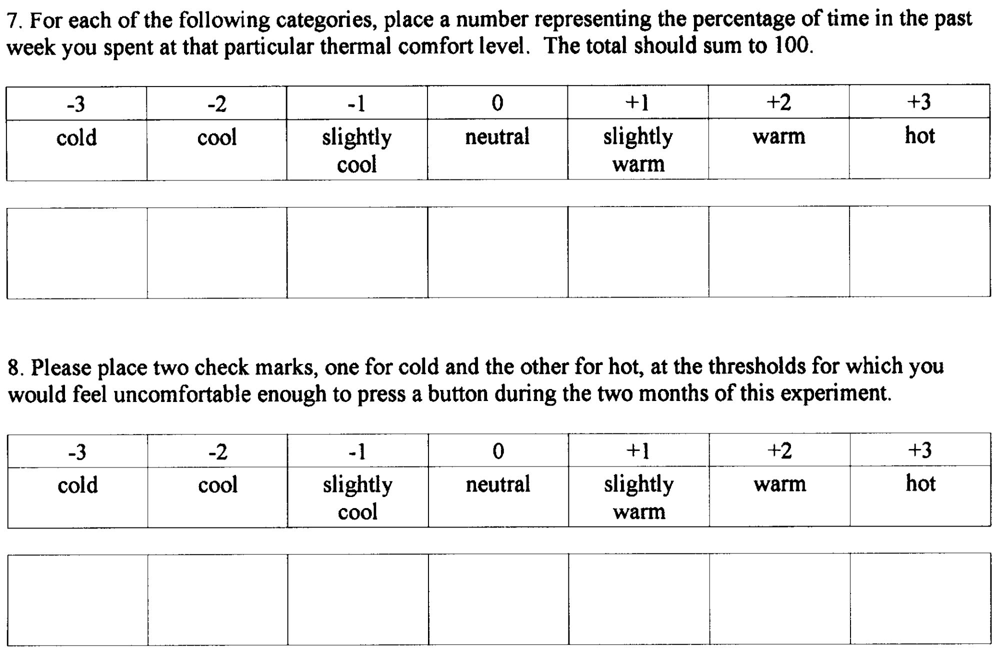
-3	-2	-1	0	+1	+2	+3
strongly disagree	disagree	slightly disagree	neutral	slightly agree	agree	strongly agree

5. I would prefer to keep using the personalized comfort system rather than go back to the old control system, if I could use the portable node as I wanted (e.g. you didn't need to wear it).

-3	-2	-1	0	+1	+2	+3
strongly disagree	disagree	slightly disagree	neutral	slightly agree	agree	strongly agree

6. Approximately how many hours did you spend at your workspace in the past week? ___________

Figure B.5: Survey for Week 4 of Experimental Control: Page One



7. For each of the following categories, place a number representing the percentage of time in the past week you spent at that particular thermal comfort level. The total should sum to 100.

-3	-2	-1	0	+1	+2	+3
cold	cool	slightly cool	neutral	slightly warm	warm	hot

8. Please place two check marks, one for cold and the other for hot, at the thresholds for which you would feel uncomfortable enough to press a button during the two months of this experiment.

-3	-2	-1	0	+1	+2	+3
cold	cool	slightly cool	neutral	slightly warm	warm	hot

Figure B.6: Survey for Week 4 of Experimental Control: Page Two

Node Number	Question Number					
	1	2	3	4	5	6
4	+2	+1	+2	+2	+3	30
8	0	+3	+3	+3	+3	50
20	+2	+2	+2	+3	+3	40
24	+3	+2	0	+3	+3	50
28	0	+2	+2	+2	+3	100
84	+1	+2	+2	+2	+3	25
88	−2	−1	−1	+1	+2	40
96	+1	+1	+2	+2	+3	40
104	+2	+1	+1	+2	+2	45
Mean	+1.00	+1.44	+1.44	+2.22	+2.78	46.67

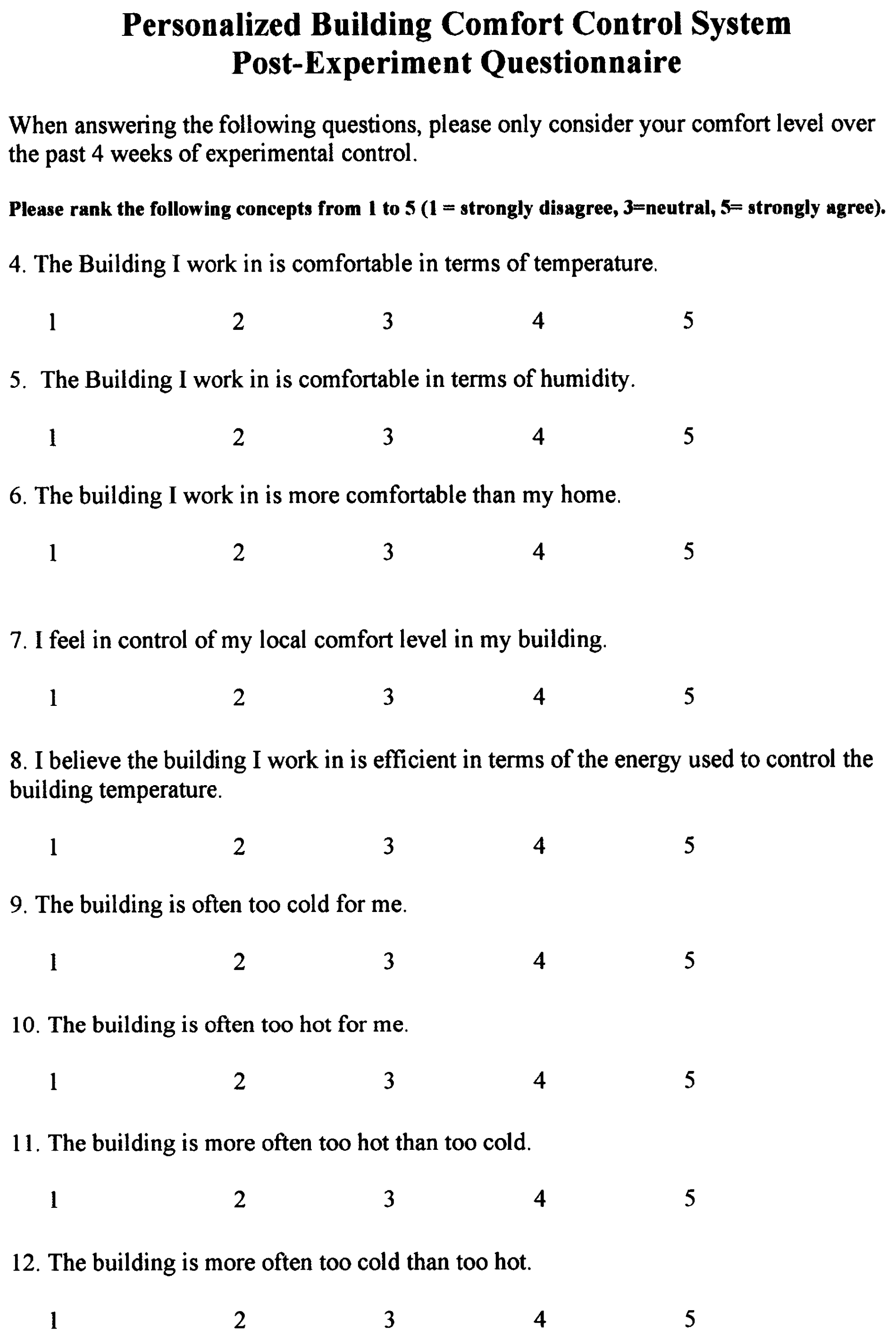
Table B.9: Survey Responses by Node Number For Week 4: Questions 1 – 6

Node Number	Percentage of Time at Each Comfort Level						
	−3	−2	−1	0	+1	+2	+3
4	0	0	0	10	50	40	0
8	0	0	10	90	0	0	0
20	0	5	5	70	0	15	5
24	0	10	20	60	10	0	0
28	1	2	10	79	8	0	0
84	0	0	10	85	5	0	0
88	0	0	0	15	80	5	0
96	0	5	10	70	10	5	0
104	0	0	4	16	40	30	10
Mean	0.11	2.44	7.67	55.00	22.56	10.56	1.67

Table B.10: Survey Responses by Node Number For Week 4: Question 7

Node	Cold	Hot
4	−3	+2
8	−2	+2
20	−3	+2
24	−2	+2
28	−1	+1
84	−2	+1
88	−3	+1
96	−3	+1
104	−2	+1
Mean	−2.33	+1.44

Table B.11: Survey Responses by Node Number For Week 4: Question 8



Personalized Building Comfort Control System
Post-Experiment Questionnaire

When answering the following questions, please only consider your comfort level over the past 4 weeks of experimental control.

Please rank the following concepts from 1 to 5 (1 = strongly disagree, 3=neutral, 5= strongly agree).

4. The Building I work in is comfortable in terms of temperature.

1 2 3 4 5

5. The Building I work in is comfortable in terms of humidity.

1 2 3 4 5

6. The building I work in is more comfortable than my home.

1 2 3 4 5

7. I feel in control of my local comfort level in my building.

1 2 3 4 5

8. I believe the building I work in is efficient in terms of the energy used to control the building temperature.

1 2 3 4 5

9. The building is often too cold for me.

1 2 3 4 5

10. The building is often too hot for me.

1 2 3 4 5

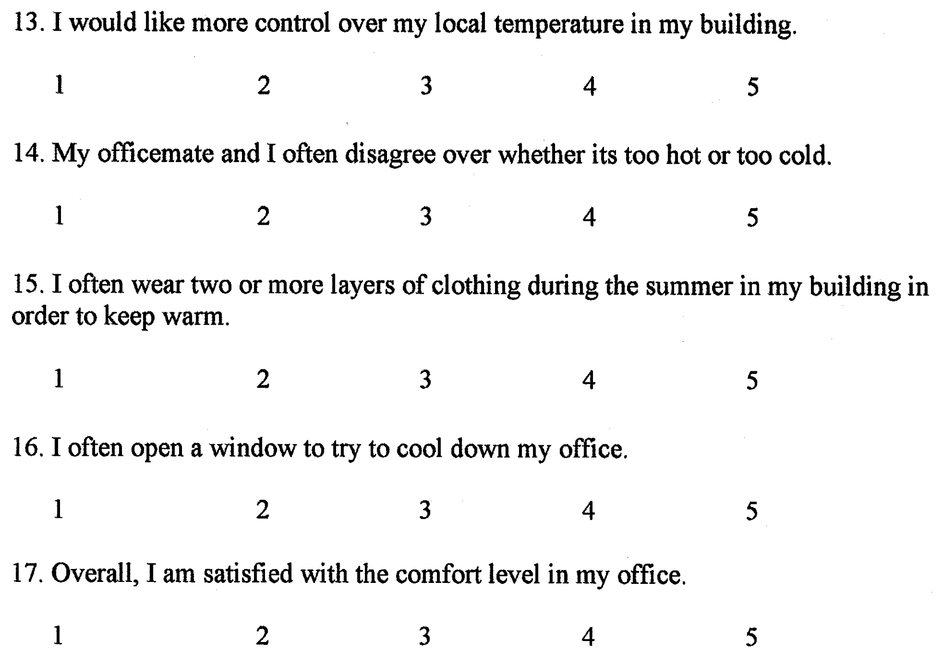
11. The building is more often too hot than too cold.

1 2 3 4 5

12. The building is more often too cold than too hot.

1 2 3 4 5

Figure B.7: Exit Survey: Page One



13. I would like more control over my local temperature in my building.

1 2 3 4 5

14. My officemate and I often disagree over whether its too hot or too cold.

1 2 3 4 5

15. I often wear two or more layers of clothing during the summer in my building in order to keep warm.

1 2 3 4 5

16. I often open a window to try to cool down my office.

1 2 3 4 5

17. Overall, I am satisfied with the comfort level in my office.

1 2 3 4 5

Figure B.8: Exit Survey: Page Two

Question Number	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
4	0	2	2	4	2
5	0	1	3	3	3
6	2	0	2	4	2
7	0	2	2	4	2
8	1	3	2	3	1
9	4	3	2	0	1
10	3	2	0	5	0
11	3	1	0	2	4
12	4	3	0	2	1
13	0	0	2	5	3
14	2	1	0	5	1
15	6	2	1	0	1
16	2	1	0	1	2
17	0	0	3	4	3

Table B.12: Number of Responses for Questions 4 – 17 on Exit Survey

Appendix C

Notes on Experimental Data

This appendix contains details on the nodes and users of the studies conducted in this dissertation. They are included here for future reference.

- All users are assumed to be present for all parts of the study, except as noted.
- Nodes 72 and 28 represent the same user. The portable node was changed from 72 to 28 on July 23rd to replace a bad activity sensor. This user is also the author of this work.
- Nodes 100 and 4 represent the same user. The portable node was changed from 100 to 4 on June 22nd to replace a bad activity sensor.
- Nodes 12 and 88 represent the same user. The portable node was changed from 12 to 88 on June 22nd to replace a bad activity sensor.
- Node 24 represents two different users, although they shared the same workspace, so the data represents the same area. The first user is from June 22nd to July 16th, and the second user is from July 17th through the end of the study.
- Node 20 has bad data for the first week of Phase Two, due to another node accidentally being on the same channel. These data were backed out later by seperating the two nodes based upon their RTC offsets.
- Node 84 represents a user that entered the study on June 22nd.
- Nodes 8 and 88 shared an office space.
- Nodes 28, 96, and 104 shared an office space.
- Nodes 4 and 20 shared an office space.
- Nodes 24 and 84 shared proximate locations in the public space.
- Node 76 had an unshared office space.
- Room nodes 44 and 48 had extended outages during Phase One and Phase Two of the study due to bad powersupplies.

- Room node 32 had extended outages during Phase Three of the study, presumably due to a network collision problem, as this was the most heavily trafficed node. This was eventually fixed with a hardware reset upon failure.
- For periods of room node outages, location information is inconclusive, and the damper control functions ceased to operate.
- Control node 184 had its wind speed sensor changed on June 30th, due to a malfunction. The data before this time is not useful.
- Control node 248 and thermostat node 240 were installed for Phase Three of the study. These were initially intended to perfrom a baseline comparison, as this office was not modified with the experimental control system. But, the office usage was very low, and the office did not have any desktop computers in it, making it a poor comparison point. The remainder of the offices had anywhere from two to nine computers in them, along with many other heat producing electronic devices.
- Control nodes would sometimes stop transmitting data during Phase One and Phase Two of the study, presumably due to bad state accounting in the wireless firmware. This was eliminated during Phase Three, as receiving a wireless transmission would wake them up, and wireless transmissions occured quite frequently during that period. The exception to this rule is control node 248, which never received control commands, as it did not have a damper motor associated with it.

Appendix D

Hardware Schematics

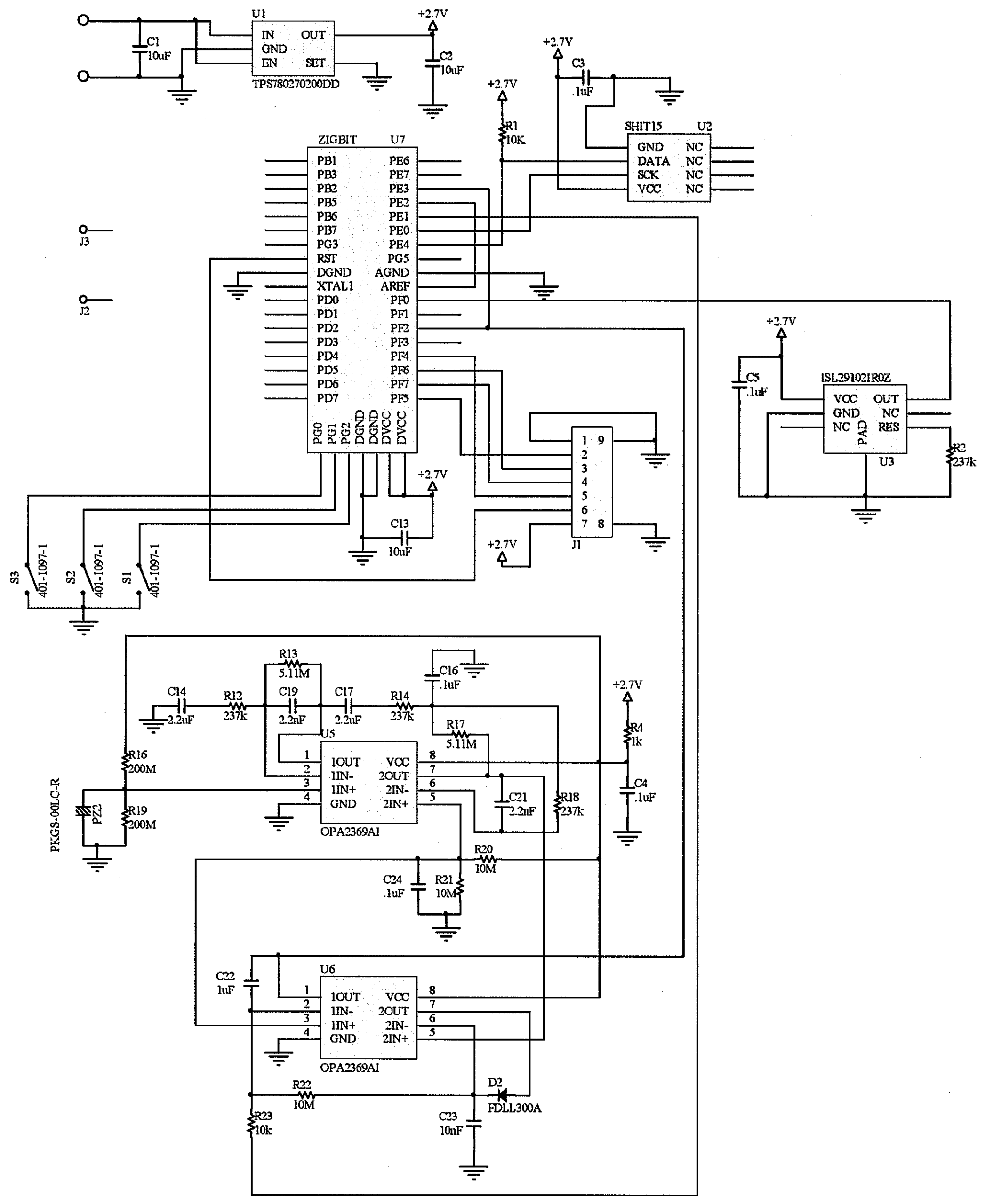


Figure D.1: Portable Node Schematic

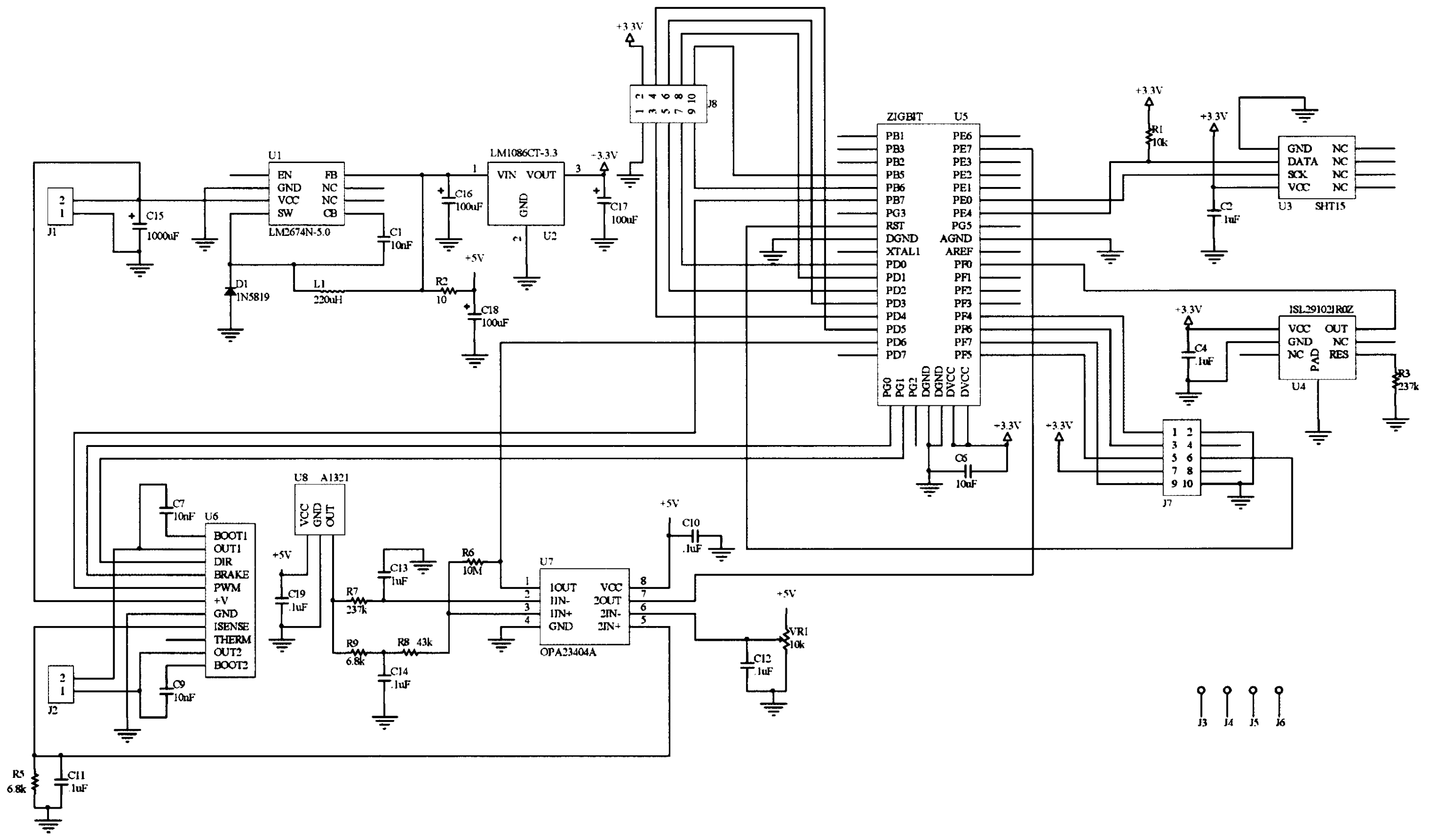
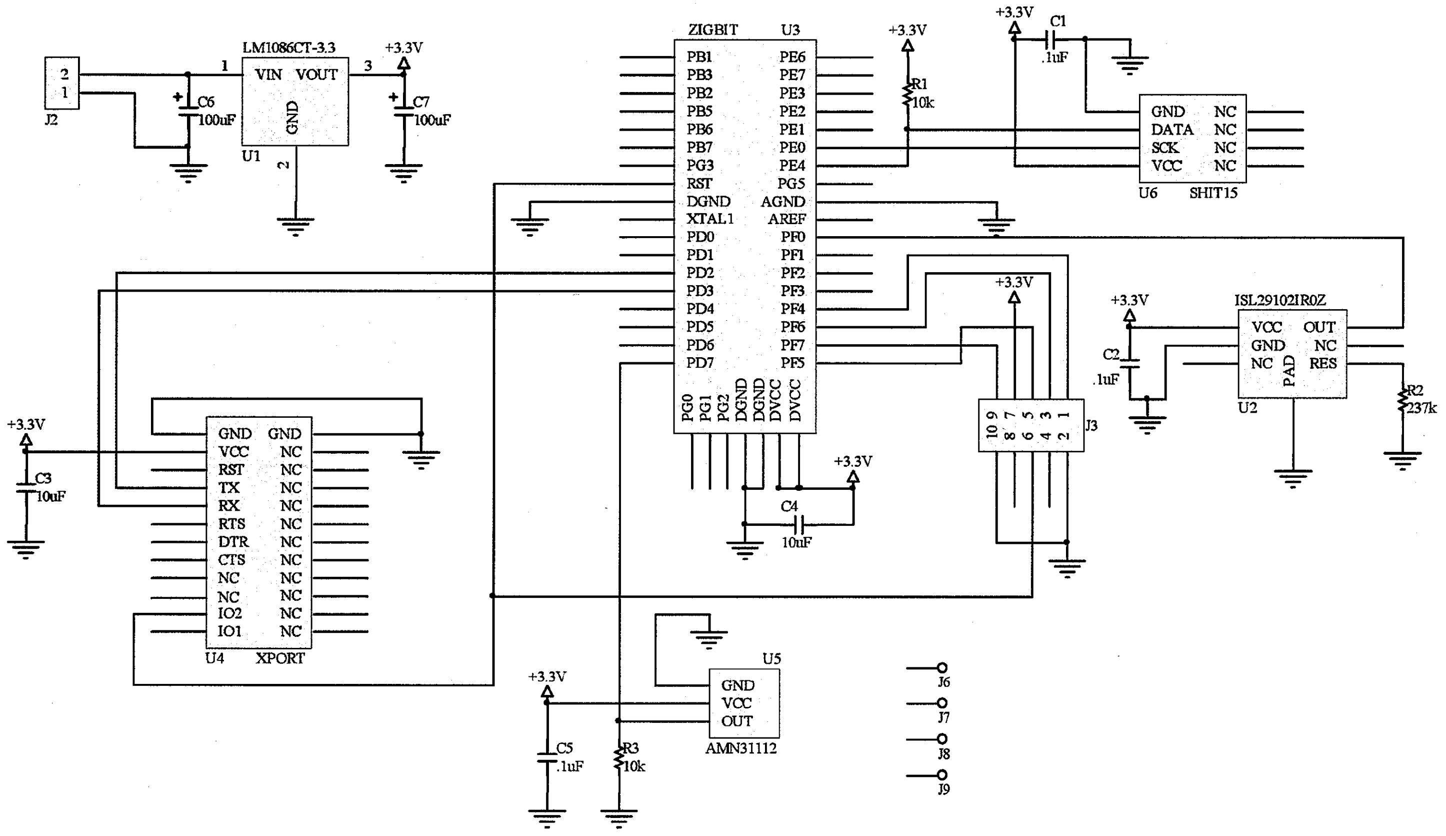


Figure D.2: Control Node Schematic



J2
LM1086CT-3.3
VIN VOUT GND
U1
C6 100uF
C7 100uF
+3.3V
ZIGBIT U3
PB1 PB3 PB2 PB5 PB6 PB7 PG3 RST DGND XTAL1 PD0 PD1 PD2 PD3 PD4 PD5 PD6 PD7
PE6 PE7 PE3 PE2 PE1 PE0 PE4 PG5 AGND AREF PF0 PF1 PF2 PF3 PF4 PF6 PF7 PF5
PG0 PG1 PG2 DGND DGND DVCC DVCC
R1 10k
C1 .1uF
GND DATA SCK VCC NC
U6 SHIT15
C4 10uF
J3
ISL29102IR0Z
VCC GND NC OUT NC RES PAD
U2
C2 .1uF
R2 237k
C3 10uF
GND VCC RST TX RX RTS DTR CTS NC IO2 IO1
U4 XPORT
U5
GND VCC OUT
AMN31112
C5 .1uF
R3 10k
J6 J7 J8 J9

Figure D.3: Room Node Schematic

Appendix E

Hardware PCB Layouts

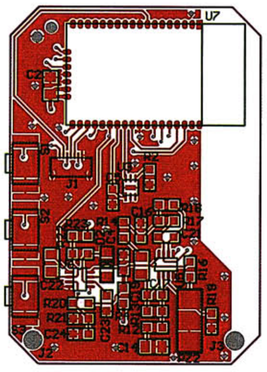


Figure E.1: Portable Node PCB Layout: Top Side (actual size)

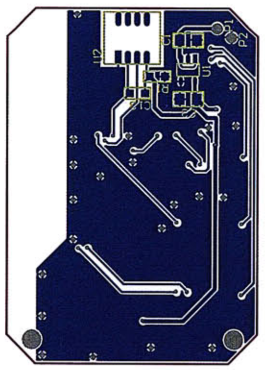


Figure E.2: Portable Node PCB Layout: Bottom Side (actual size)

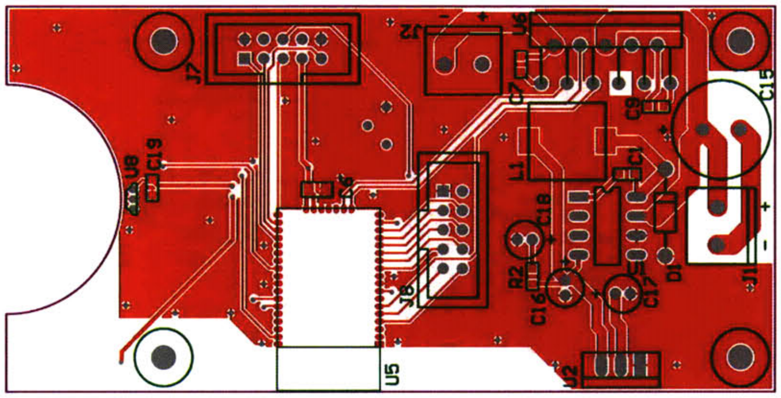


Figure E.3: Control Node PCB Layout: Top Side (actual size)

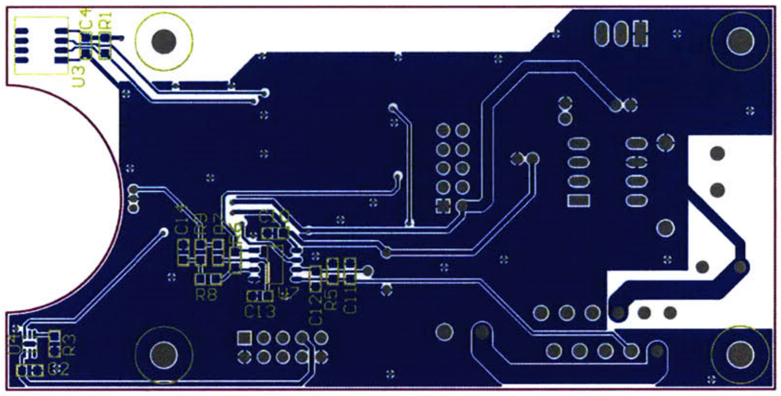


Figure E.4: Control Node PCB Layout: Bottom Side (actual size)

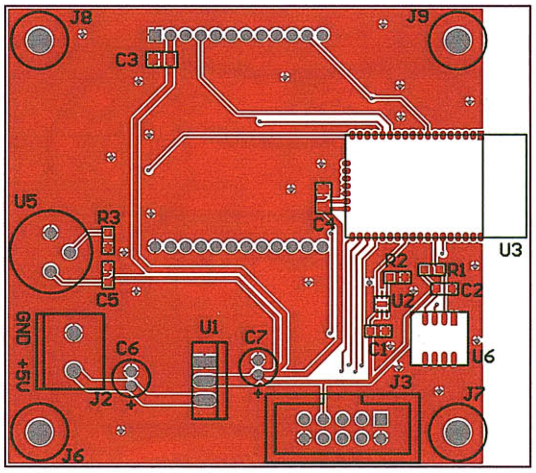


Figure E.5: Room Node PCB Layout: Top Side (actual size)

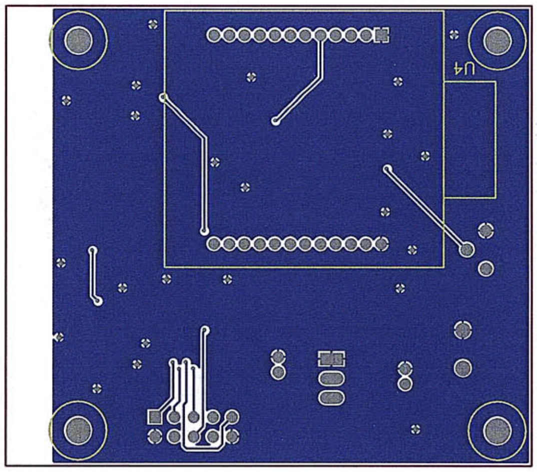


Figure E.6: Room Node PCB Layout: Bottom Side (actual size)

Appendix F

Portable Node Firmware

```
.include "m1281def.inc"

; use increments of 4 for src_addr
; csma_seed = 8 x src_addr
.equ src_addr = $aa58 ; place unique source address here
.equ csma_seed = low(src_addr)*$08 ; random backoff exponent - must be unique,<$0800
.equ home_addr = $aabb ; local node des_addr
.equ pan_id = $abcd ; system pan_id
.equ retries = $38 ; high nibble = frame retries,low nibble = 2x(csma retries)
.equ channel = $36 ; tx/rx channel, $2b -> $3a valid

.org 0
rjmp start
.org int4addr
rjmp wakeupshit15
.org int5addr
rjmp clearirq
.org OVF2addr
rjmp wakeup
.org ADCCaddr
rjmp wakeup

; accelerometer reset = pe1
; buttons = pg0,pg1,pg2
; accelerometer = pf2(adc2),pe3(ain1)
; shit15 data = pe4(int4)
; shit15 clock = pe0
; vref to pe2(ain0)
; light sensor out = pf0(adc0)
; light sensor power = pf1
; r0  shit15 temperature msb
; r1  shit15 temperature lsb
; r2  shit15 humidity
; r3  activity msb
; r4  activity lsb
; r5  buffer pointer temporary storage
; r6  light sensor msb
; r7  light sensor lsb
; r8
; r9
; r10 des_addr msb
; r11 des_addr lsb
; r12
; r13
; r14
; r15 button press temporary register
; r16 interrupt temporary register
; r17 interrupt temporary register
; r18 rf unit state register
; r19 adc delay timer
; r20 timer register 1
; r21 no_ack counter
; r22 wakeup delay timer
; r23 button wait register
; r24 button press
; r25 previous button press (sent as button data)
; r26 wait timer for joinrequest
; r27
; ypointer is buffer end
; zl
; zh
```

```
start: ; configure microcontroller registers
; using internal 8MHz RC oscillator
; system clock set to 1MHZ by /8 prescaler in fuse bits
ldi r16,high(RAMEND)
out SPH,r16 ; Set Stack Pointer to top of RAM
ldi r16,low(RAMEND)
out SPL,r16
ldi r16,$17
out ddrb,r16 ; set ss,sclk,mosi,slp_tr as output
ldi r16,8
out portb,r16 ; enable pullup for miso
ldi r16,1
out spsr,r16 ; set up spi
ldi r16,$50
out spcr,r16 ; set up spi
sbi portb,portb0 ; set ss pin high
cbi portb,portb4 ; set slp_tr low
sbi ddra,dda7 ; set reset pin as output
sbi porta,porta7 ; pull reset high on rf unit
ldi r16,$0c
sts eicrb,r16 ; configure int5 for rising edge
ldi r16,$20
out eimsk,r16 ; enable int5
sbi portg,portg2 ; turn on pullup for button1
sbi portg,portg1 ; turn on pullup for button2
sbi portg,portg0 ; turn on pullup for button3
ldi r16,high(home_addr) ; set up des_addr to something in system for start
mov r10,r16
ldi r16,low(home_addr) ; set up des_addr
mov r11,r16
ldi yh,$02 ; initialize buffer pointer
ldi yl,$00
clr r18 ; initialize rf state register
clr r5 ; initialize pointer buffer
clr r25 ; initialize registers
clr r24
clr r23
sbi portf,portf1 ; set pf1 to high to activate light sensor
sbi ddrf,ddf1 ; set pf1 to output

;turn off watchdog timer
wdr ; reset wdt
ldi r16,$00 ; clear all resets
out mcusr,r16 ; turn off wdrf
ldi r16,$1e
sts wdtcsr,r16 ; enable writing of wdt
ldi r16,$06
sts wdtcsr,r16 ; turn off wdt, 1s timer

;setup power reduction registers
ldi r16,$ab
sts prr0,r16 ; turn off adc,usart0,t1,t0,twi
ldi r16,$3f
sts prr1,r16 ; turn off t5,t4,t3,usart3,usart2,usart1
ldi r16,$90
out acsr,r16 ; turn off comparator

;setup analog inputs
ldi r16,$05
sts didr0,r16 ; turn off input stage for adc0,adc2 pins
```

```
ldi r16,$03
sts didr1,r16 ; turn off input stage for ain0,ain1 pins

;setup t2 as rtc at 1s interval wakeup
ldi r16,$20
sts assr,r16 ; set t2 to assynchronous mode
ldi r16,$05
sts tccr2b,r16 ; set t2 prescaler to /128 - 1s wakeup period

;make sure t2 is done rewriting itself
checkassr:

lds r16,assr
andi r16,$1f ; check assr(4:0) clear
brne checkassr

clr r16
sts tifr2,r16 ; clear all pending interrupts
ldi r16,$01
sts timsk2,r16 ; enable t2 overflow interrupt

;setup shit15 - pe0 is data, pe4 is clock
sbi ddre,dde0 ; set clock as output
cbi porte,porte4 ; make sure pullups are off for data
rcall shit15setup

;configure rf unit registers

;wait
ldi r20,$80 ; load the wait timer to 128 cycles (x10/ck)
rcall wait2 ; wait ~(1280/1MHz = 1.2ms) for rf unit to stabilize

;go to trx_off state
rcall txoff ; go to trx_off state on rf unit

;setup clkm
ldi r16,$c3 ; load first data byte - write trx_ctrl_0 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
clr r16 ; load second data byte - turn off clkm
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup short_addr_0
ldi r16,$e0 ; load first data byte - write short_addr_0 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,low(src_addr) ; load second data byte - src_addr lsb
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup short_addr_1
ldi r16,$e1 ; load first data byte - write short_addr_1 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,high(src_addr) ; load second data byte - src_addr msb
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup pan_id_0
ldi r16,$e2 ; load first data byte - write pan_id_0 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,low(pan_id) ; load second data byte - pan_id lsb
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup pan_id_1
ldi r16,$e3 ; load first data byte - write pan_id_1 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,high(pan_id) ; load second data byte - pan_id msb
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup phy_cc_ca - set channel id
ldi r16,$c8 ; load byte - write phy_cc_ca register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,channel ; load data byte -
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup phy_tx_pwr - turn crc on
ldi r16,$c5 ; load byte - write phy_tx_pwr register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,$80 ; load data byte - auto_crc=on,pwr=+3dbm
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup xah_ctrl - set frame and csma retries
ldi r16,$ec ; load byte - write xah_ctrl register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,retries ; load data byte - frame and csma retries
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup csma_seed_1
ldi r16,$ee ; load byte - write csma_seed_1 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,high(csma_seed) ; load data byte - min_be=0,aack_set=0,i_am_coord=0,csma(10:8)
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup csma_seed_0
ldi r16,$ed ; load first data byte - write csma_seed_1 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,low(csma_seed) ; load data byte - csma(7:0)
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;clear pending irqs on rf unit
ldi r16,$8f ; load first data byte - read register irq_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte and clear pending irqs
sbi portb,portb0 ; pull ss high

;enable irqs on rf unit
```

```
ldi r16,$ce ; load first data byte - write register irq_mask command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$08 ; load second data byte - trx_end only
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

;clear pending irqs on micro
clr r16
out eifr,r16 ; clear any interrupt flags that are set

;enable interrupts on micro
sei ; turn on interrupts

;enter sleep mode on rf unit
sbi portb,portb4 ; pull slp_tr high to enter sleep mode
ldi r20,$80 ; load the wait timer to 128 cycles (x10/ck)
rcall wait2 ; wait ~(1280/1MHz = 1.2ms) to make sure were in sleep

repeat: ; transmit data from sleep mode
; use watchdog timer to eliminate lockups while not sleeping
cli ; turn off interrupts while enabling watchdog timer
wdr ; reset wdt
ldi r16,$1e
sts wdtcsr,r16 ; enable writing of wdt
ldi r16,$0e
sts wdtcsr,r16 ; turn on wdt, 1s timer
sei ; turn interrupts back on

ldi r19,$0f ; sample n times - 4ms

adcsample: ; take measurement from adc0 - light sensor
ldi r16,$aa
sts prr0,r16 ; turn on adc, leave off usart0,t1,t0,twi
ldi r16,$c0
sts admux,r16 ; use internal 2.56v as voltage reference,sample adc0
ldi r16,$9c
sts adcsra,r16 ; turn on adc,clear interrupt flag,enable interrupt,set to /16
ldi r16,$03
out smcr,r16 ; setup adc noise reduction mode
nop
nop
nop
sleep ; wait till converesion is done
nop
nop
nop
clr r16
out smcr,r16 ; disable sleep modes
lds r7,adcl ; load low bit to r7
lds r6,adch ; load high bit to r6
dec r19
brne adcsample ; take n samples to let vref settle

; turn off light sensor
cbi ddrf,ddf1 ; set pf1 to input to turn off light sensor
cbi portf,portf1 ; turn pf1 pullups off

; sample adc2 - accelerometer
ldi r16,$c2
sts admux,r16 ; use internal 2.56v as voltage reference,sample adc2
ldi r16,$9c
sts adcsra,r16 ; turn on adc,clear interrupt flag,enable interrupt,set to /16
ldi r16,$03
out smcr,r16 ; setup adc noise reduction mode
nop
nop
nop
sleep ; wait till converesion is done
nop
nop
nop
clr r16
out smcr,r16 ; disable sleep modes
lds r4,adcl ; load low bit to r4
lds r3,adch ; load high bit to r3

;reset accelerometer
ldi r16,$10
out acsr,r16 ; turn on comparator
nop ; delay to let things settle a bit
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
in r16,acsr ; check comparator output
sbrs r16,5 ; dont bother resetting if already high
rjmp compdone
cbi porte,porte1 ; set reset pin to low
sbi ddre,dde1 ; set reset pin to output

compwait: ; idle until ain1 > ain0

in r16,acsr
sbrc r16,5
rjmp compwait ; keep checking comparator output for low output
cbi ddre,dde1 ; turn off the resetting function

compdone: ; resetting done

ldi r16,$90
out acsr,r16 ; turn off comparator
ldi r16,$00
sts adcsra,r16 ; turn off adc
ldi r16,$ab
sts prr0,r16 ; turn off adc,usart0,t1,t0,twi

rcall shit15temp ; read shit15 temperature

rcall shit15humidity ; read shit15 humidity

cbi portb,portb4 ; pull slp_tr low to exit sleep mode
ldi r20,$28 ; load wait timer to 40 cycles (x10/ck)
rcall wait2 ; wait ~(400/1MHZ = 400us) for rf unit clock to stabilize

rcall checktxoff ; check to see if rf unit is in tx_off state
rcall txonaret ; put the rf unit in tx_aret_on state

cpi r21,$0c ; check how long its been since ack (12min)
brlo shortout ; dont bother with data packets if longout
ldi yl,$00 ; clear old data
ldi r21,$0f ; keep ack counter from incrementing back to $00
rcall transmitrequest ; transmit join request
ldi r18,$02 ; set rf state to joinrequest sent
rjmp rfwait ; wait for transmit done

shortout: ; possibly still in system - keep full data transmits

rcall transmitbeacon ; transmit beacon packet
ldi r18,$01 ; set rf state register to transmitbeacon

rfwait: ; wait till transmission is complete - could be replaced with a sleep
;int5 can only be used as level interrupt to wake from most sleep modes
;int5 must be configured as rising edge as it is active high
;could use idle mode to sleep during this operation
;idle mode would save 0.7mA max - probably not worth it
nop
nop
nop
cpi r18,$40 ; check if all transmissions are done
breq sleepstate ; shut down if all transmissions done
cpi r18,$80 ; check if all transmission are done with backlog
breq backlog ; backlog data
cpi r18,$04 ; check if waiting for joinack
breq rfwait1 ; go to timeout sequence if waiting for joinack
rjmp rfwait ; else keep waiting


rfwait1: ; wait for joinack

ldi r26,$ff ; wait some sweet ass long time for joinack

wait3: ; 255x16/1MHz = 4ms waitloop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
cpi r18,$04 ; check if we have a new network
brne rfwait ; go back to waiting if new network
dec r26 ; check if done
brne wait3 ; keep going if not done

backlog: ; store backlogged data

cpi r21,$0c ; check if in long_out state
brsh sleepstate ; dont bother backlogging if long_out
st y+,r6 ; load data to buffer if failed
st y+,r7
st y+,r3
st y+,r4
st y+,r0
st y+,r1
st y+,r2
st y+,r25
inc r21 ; increment no_ack counter
cpi yh,$03 ; check if counter has overflowed
brlo sleepstate ; skip if not
ldi yh,$02 ; reset pointer if overflow
ldi yl,$00 ; reset pointer if overflow

sleepstate: ; finish off and go to sleep

rcall forcetxoff ; go to tx_off state so you can sleep
sbi portb,portb4 ; pull slp_tr high to enter sleep mode
ldi r16,$07
out smcr,r16 ; set to power-save mode and enable sleep mode
ldi r22,$3c ; set up repeat timer to 60cycles
; turn watchdog timer off when going back to sleep
cli ;  turn off interrupts while disabling watchdog timer
wdr ; reset wdt
in r16,mcusr ; get reset flag status
andi r16,$f7 ; mask off wdrf to 0
out mcusr,r16 ; turn off wdrf
ldi r16,$1e
sts wdtcsr,r16 ; enable writing of wdt
ldi r16,$06
sts wdtcsr,r16 ; turn off wdt, 1s timer
sei ; turn interrupts back on

sleepstate1:
cpi r22,$01
brne sleepstate3
sbi portf,portf1 ; turn on light sensor via pf1
sbi ddrf,ddf1 ; set pf1 to output

sleepstate3:
sleep
nop
nop
nop
mov r25,r15 ; move last current button press to previous button press
clr r15 ; reset button state register
in r24,ping ; read button state
ser r16
eor r24,r16 ; flip state of buttons because they are currently pulled up
andi r24,$07 ; mask off the three buttons of interest
breq sleepstate2 ; do not turn on lightsensor if no buttons pressed
sbi portf,portf1 ; turn on light sensor via pf1 if button pressed
sbi ddrf,ddf1 ; set pf1 to output
sec ; set carry bit

buttonshift: ; convert to bitwise representation
rol r15 ; increment button press number
dec r24 ; check if done
brne buttonshift ; keep incrementing if not done
and r25,r15 ; check if button pressed last time
breq sleepstate2 ; do nothing if button was not pressed last time and this time
mov r16,r23 ; get set of buttons pressed in past minute
and r16,r15 ; check if already sent that button press in past minute
brne sleepstate2 ; dont send data if already sent it
```

```
or r23,r15 ; mask off button wait register
rjmp repeat ; take measurements and send them off - r25

sleepstate2:
dec r22
brne sleepstate1 ; return to sleep if not yet 60s
clr r16
out smcr,r16 ; disable sleep mode
and r25,r15 ; check if both last and current are the same
mov r23,r25 ; reset button wait register
rjmp repeat ; take measurements and send them off

txoff: ; send trx_off command to rf unit and wait till stable

ldi r16,$c2 ; load first data byte - write register trx_state command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$08 ; load second data byte - data trx_off bit
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

checktxoff: ; check current state to see if in tx_off state

ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$08
brne checktxoff
ret ; return to previous duty

forcetxoff: ; send trx_off command to rf unit and wait till stable

ldi r16,$c2 ; load first data byte - write register trx_state command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$03 ; load second data byte - data force_trx_off bit
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high


checkforcetxoff: ; check current state to see if in tx_off state

nop ; delay for a bit to make sure the device is off
nop
nop
nop
ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$08
brne forcetxoff ; redo if not off - only takes 1us to turn off
;(changed due to lock up bug of getting stuck in check loop)
ret ; return to previous duty

pllon: ; send pll_on command to rf unit

ldi r16,$c2 ; load first data byte - write register trx_state command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$09 ; load second data byte - data pll_on bit
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

checkpllon: ; check current state to see if in pll_on state

ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$09
brne checkpllon
ret ; return to previous duty

txonaret: ; send trx_aret_on command to rf unit and wait till stable

ldi r16,$c2 ; load first data byte - write register trx_state command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$19 ; load second data byte - data trx_aret_on bit
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

checktxonaret: ; check current state to see if in tx_aret_on state

ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$19
brne checktxonaret
ret ; return to previous duty

rxonaack: ; send rx_aack_on command to rf unit and wait till stable

ldi r16,$c2 ; load first data byte - write register trx_state command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$16 ; load second data byte - data rx_aack_on bit
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

checkrxonaack: ; check current state to see if in rx_aack_on state

ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$16
brne checkrxonaack
ret ; return to previous duty
```

```
spiwrite: ; read and write data over spi

out spdr,r16 ; write transmitted byte to spi

wait: ; read spi register to check when done

in r16,spsr
sbrs r16,spif
rjmp wait
in r16,spdr ; write recieved byte to spi register
ret ; return to previous duty

wait2: ; wait timer (~12 cylces per value in r20)

nop
nop
nop
nop
nop
nop
nop
nop
dec r20
brne wait2
ret ; return to previous duty

joinrecieve: ; get new des_addr

cpi r18,$04 ; check if waiting for joinack
brne done9 ; skip if not
ldi r16,$20 ; load data byte - read buffer command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - frame length byte
cpi r16,$0b ; check if its a join frame length
brne donejoinfail ; finish if not
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcf(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcf(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - seq (used as command byte)
cpi r16,$06 ; check if joinpacket
brne donejoinfail ; finish if not
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_panid(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_panid(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_addr(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_addr(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - src_addr(7:0)
mov r11,r16
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - src_addr(15:8)
mov r10,r16
; dont bother with the rest of the data - it doesnt matter
```

```
;ldi r16,$00 ; load data byte - blank for reading
;rcall spiwrite ; send byte - do not save fcs(7:0)
;ldi r16,$00 ; load data byte - blank for reading
;rcall spiwrite ; send byte - do not save fcs(15:8)
;ldi r16,$00 ; load data byte - blank for reading
;rcall spiwrite ; send byte - lqi
sbi portb,portb0 ; pull ss high
rcall checkrxonaack ; make sure we are no longer in rx_busy
rcall pllon ; go to pll_on state
rcall txonaret ; go to tx_on state
rcall transmitpacket ; send out data packet
ldi r18,$10 ; set state register to retry transmit
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; clear interrupt and return to previous duty

donejoinfail: ; finish and wait for next packet

sbi portb,portb0 ; pull ss high
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; clear interrupt and return to previous duty

done9: ; rx bad packet

ldi r18,$40 ; set rf state to done
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; clear interrupt and return to previous duty

clearirq: ; clear irq register and irq line

in r16,sreg ; get sreg
push r16 ; push sreg on stack
ldi r16,$8f ; load first data byte - read register irq_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

checkrf: ; check rf unit state

ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$16 ; check if in rx_on
breq joinrecieve ; go to get data
cpi r16,$11 ; check if in busy_rx
breq joinrecieve ; go to get data
cpi r16,$1f ; check if in state transition
breq checkrf ; keep checking till in a new state
cpi r16,$12 ; check if in busy_tx state
breq checkrf ; keep checking till in a new state
;else check successful transmit before switching back to sleep
ldi r16,$82 ; load data byte - read trx_state to get trac_status
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load data byte - blank byte for reading
rcall spiwrite ; send byte
```

```
sbi portb,portb0 ; pull ss high
andi r16,$e0 ; mask off trac_status
cpi r16,$60 ; check if success or data_pending success
brlo success ; skip if success
cpi r18,$00 ; check if last transmission was a data packet
breq retry ; get new des_addr
cpi r18,$10 ; check if last transmission was a retry packet
breq retryfail ; backlog and end
cpi r18,$20 ; check if last transmission was a backlog packet
breq backlogfail ; backlog and end
ldi r18,$40 ; else set to shutdown
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; return from interrupt

retry: ; get new des_addr and retransmit

rcall transmitrequest
ldi r18,$02 ; set joinrequest state bit
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; clear interrupt and return to previous duty

backlogfail: ; reset backlog pointer

mov yl,r5 ; reset data pointer because data didnt go out

retryfail: ; backlog and end

ldi r18,$80 ; set backlog flag
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; clear interrupt and return to previous duty

success: ; tx success

cpi r18,$01 ; check if last packet was beacon
breq txdata ; transmit data off
cpi r18,$00 ; check if last packet was data
breq checkback ; see if there is backlogged data
cpi r18,$02 ; check if last packet was join request
breq gotorx ; set to rx and wait for join ack
cpi r18,$20 ; check if last packet was backlog data
breq irqdone ; finish off
cpi r18,$10 ; check if last packet was retry
breq checkback ; check for backlogged data
rjmp done9 ; finish off if bad state

checkback: ; check for backlogged data

cpi yl,$00 ; check for backlogged data
breq irqdone ; finish if none
mov r5,yl ; store data pointer in case of failure
rcall transmitold ; transmit backlogged data
ldi r18,$20 ; set state to backlog-aack
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; clear interrupt and return to previous duty

irqdone: ; finish interrupt and return
```

```
clr r21 ; reset no_ack register
ldi r18,$40 ; set rf state to done
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; clear interrupt and return to previous duty

txdata: ; transmit a data packet

rcall transmitpacket ; send data
ldi r18,$00 ; set rf state to data sent
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; clear interrupt and return to previous duty

gotorx: ; recieve joinconfirm

rcall pllon ; go to pll_on state
rcall rxonaack ; go to rx_on_aack state
ldi r18,$04 ; set rf state to rx-waiting
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; return from interrupt

transmitbeacon: ; transmit a beacon frame

sbi portb,portb4 ; pull slp_tr high to begin transmit
nop ; delay to make sure the pin is high long enough
cbi portb,portb4 ; pull slp_tr low

ldi r16,$60 ; load data byte - write buffer command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$0b ; load data byte - frame length byte
rcall spiwrite ; send byte
ldi r16,$40 ; load data byte - fcf(7:0) - broadcast,no ack
rcall spiwrite ; send byte
ldi r16,$88 ; load data byte - fcf(15:8) - short address,pan_ids equal
rcall spiwrite ; send byte
ldi r16,$00 ; load data byte - seq(7:0) - command(beacon)
rcall spiwrite ; send byte
ldi r16,low(pan_id) ; load data byte - pan_id(7:0)
rcall spiwrite ; send byte
ldi r16,high(pan_id) ; load data byte - pan_id(15:8)
rcall spiwrite ; send byte
ldi r16,$ff ; load data byte - destination address (7:0) - broadcast
rcall spiwrite ; send byte
ldi r16,$ff ; load data byte - destination address (15:0) - broadcast
rcall spiwrite ; send byte
ldi r16,low(src_addr) ; load data byte - source address (7:0)
rcall spiwrite ; send byte
ldi r16,high(src_addr) ; load data byte - source address (15:8)
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
ret ; return to previous duty

wakeup: ; clear interrupts and return to previous task
nop
nop
nop
reti
```

```
;talk to shit15, only works for cpu clock <=2MHZ
;data pin can never be output high - leave pullup off
;needs external pullup resistor for proper operation
wakeupshit15: ; clear interrupts and return
nop
nop
nop
cbi eimsk,4 ; disable int4 as pe4 will still be low when returning
nop
reti


shit15setup: ; set the device to low res mode

rcall shit15reset ; reset the device in case its fuxxored
rcall startshit15 ; send start sequence

;data sequence - register write command $06
sbi porte,porte0 ; clock high - data bit one
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit two
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit three
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit four
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit five
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; data high
sbi porte,porte0 ; clock high - data bit six
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit seven
cbi porte,porte0 ; clock low
sbi ddre,dde4 ; data low
sbi porte,porte0 ; clock high - data bit eight
cbi porte,porte0 ; clock low

rcall shit15ack0 ; wait for ack from shit15

;data sequence - register write data $01
sbi ddre,dde4 ; data low
sbi porte,porte0 ; clock high - data bit one
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit two
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit three
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit four
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit five
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit six
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit seven
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; data high
sbi porte,porte0 ; clock high - data bit eight
cbi porte,porte0 ; clock low

rcall shit15ack0 ; wait for ack from shit15
ret ;done setting up the device
```

```
shit15temp: ; read temperature data from shit15
;send read temp commmand
rcall startshit15 ; send start sequence

;data sequence - read temperature command $03
sbi porte,porte0 ; clock high - data bit one
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit two
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit three
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit four
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit five
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit six
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; data high
sbi porte,porte0 ; clock high - data bit seven
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit eight
cbi porte,porte0 ; clock low
;sbi ddre,dde4 ; data low - if last databyte is 1 - leave it high for ack

rcall shit15ack0 ; wait for ack from shit15

sbi eimsk,4 ; enable int4
ldi r16,$07
out smcr,r16 ; set to power-save mode and enable sleep mode
nop
nop
nop
sleep ; sleep while waiting for low level on pe4
nop
nop
nop
clr r16
out smcr,r16 ; disable sleep mode

rcall shit15read ; read out temperature data from shit15 (msb)
rcall shit15ack3 ; acknowledge reciept of bit
mov r0,r16 ; move msb to another register
rcall shit15read ; read out temperature data from shit15 (lsb)
rcall shit15ack4 ; end transmission ack
mov r1,r16 ;  move lsb to another register
ret ; continue with previous task

shit15humidity: ; read humidity data from shit15
;send read humidity command
rcall startshit15 ; send start sequence

;data sequence - read humidity command $05
sbi porte,porte0 ; clock high - data bit one
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit two
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit three
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit four
cbi porte,porte0 ; clock low
```

```
sbi porte,porte0 ; clock high - data bit five
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; data high
sbi porte,porte0 ; clock high - data bit six
cbi porte,porte0 ; clock low
sbi ddre,dde4 ; data low
sbi porte,porte0 ; clock high - data bit seven
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; data high
sbi porte,porte0 ; clock high - data bit eight
cbi porte,porte0 ; clock low
;sbi ddre,dde4 ; data low - if last databyte is 1 - leave it high for ack

rcall shit15ack0 ; wait for acknowledge from shit15

sbi eimsk,4 ; enable int4
ldi r16,$07
out smcr,r16 ; set to power-save mode and enable sleep mode
nop
nop
nop
sleep ; sleep while waiting for low level on pe4
nop
nop
nop
clr r16
out smcr,r16 ; disable sleep mode

rcall shit15read ; read out humidity data from shit15 (msb)
rcall shit15ack3 ; acknowledge reciept of bit
rcall shit15read ; read out humidity data from shit15 (lsb)
rcall shit15ack4 ; end transmission ack
mov r2,r16 ;  move lsb to another register
ret ; return with previous task

shit15ack4: ; end transmission ack

sbi porte,porte0 ; clock high - ack
cbi porte,porte0 ; clock low
ret ; return to previous task

shit15ack3: ; data recipet acknowledge

sbi ddre,dde4 ; pull data low
sbi porte,porte0 ; clock high - ack
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; release data line
ret ; return to previous task

shit15read: ; read data from shit15

clr r16 ; clear the register where incoming data will be written
sbi porte,porte0 ; clock high - data bit one
sbic pine,pine4 ; check if data is low
sbr r16,$80 ; write data bit one to register
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit two
sbic pine,pine4 ; check if data is low
sbr r16,$40 ; write data bit two to register
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit three
sbic pine,pine4 ; check if data is low
sbr r16,$20 ; write data bit three to register
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit four
sbic pine,pine4 ; check if data is low
sbr r16,$10 ; write data bit four to register
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit five
sbic pine,pine4 ; check if data is low
sbr r16,$08 ; write data bit five to register
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit six
sbic pine,pine4 ; check if data is low
sbr r16,$04 ; write data bit six to register
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit seven
sbic pine,pine4 ; check if data is low
sbr r16,$02 ; write data bit seven to register
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high - data bit eight
sbic pine,pine4 ; check if data is low
sbr r16,$01 ; write data bit eight to register
cbi porte,porte0 ; clock low
ret ; return to previous activity

startshit15: ; start sequence

cbi ddre,dde4 ; set data high
cbi porte,porte0 ; make sure clock is low
sbi porte,porte0 ; pull clock high
sbi ddre,dde4 ; set data low
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high
cbi ddre,dde4 ; data high
cbi porte,porte0 ; clock low
sbi ddre,dde4 ; data low
ret ; return to previous task

shit15ack0: ; ack sequence for sending data

cbi ddre,dde4 ; release data line

shit15ack2: ; check that data line is low

sbic pine,pine4
rjmp shit15ack2
sbi porte,porte0 ; clock high - ack
cbi porte,porte0 ; clock low

shit15ack1: ; check that line has been released

sbis pine,pine4 ; check that line has been released
rjmp shit15ack1
ret ; return to previous task

shit15reset: ; reset sequence if it gets out of phase
;status register preserved
;must be followed by data start sequence
cbi ddre,dde4 ; data high
sbi porte,porte0 ; clock high
cbi porte,porte0 ; clock low
```

```
sbi porte,porte0 ; clock high
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high
cbi porte,porte0 ; clock low
sbi porte,porte0 ; clock high
cbi porte,porte0 ; clock low
ret ; return to previous task

transmitrequest: ; transmit a request frame

sbi portb,portb4 ; pull slp_tr high to begin transmit
nop ; delay to make sure the pin is high long enough
cbi portb,portb4 ; pull slp_tr low

ldi r16,$60 ; load data byte - write buffer command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$0b ; load data byte - frame length byte
rcall spiwrite ; send byte
ldi r16,$40 ; load data byte - fcf(7:0) - broadcast,no ack
rcall spiwrite ; send byte
ldi r16,$88 ; load data byte - fcf(15:8) - short address,pan_ids equal
rcall spiwrite ; send byte
ldi r16,$01 ; load data byte - seq(7:0) - command(request)
rcall spiwrite ; send byte
ldi r16,low(pan_id) ; load data byte - pan_id(7:0)
rcall spiwrite ; send byte
ldi r16,high(pan_id) ; load data byte - pan_id(15:8)
rcall spiwrite ; send byte
ldi r16,$ff ; load data byte - destination address (7:0) - broadcast
rcall spiwrite ; send byte
ldi r16,$ff ; load data byte - destination address (15:0) - broadcast
rcall spiwrite ; send byte
ldi r16,low(src_addr) ; load data byte - source address (7:0)
rcall spiwrite ; send byte
ldi r16,high(src_addr) ; load data byte - source address (15:8)
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
ret ; return to previous duty

transmitold: ; transmit backlogged data

sbi portb,portb4 ; pull slp_tr high to begin transmit
nop ; delay to make sure the pin is high long enough
cbi portb,portb4 ; pull slp_tr low

ldi r16,$60 ; load data byte - write buffer command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
```

```
mov r17,yl ; load data byte - frame length byte
ldi r16,$0b ; increase frame length for header and crc
add r16,r17 ; increase frame length
rcall spiwrite ; send byte
ldi r16,$61 ; load data byte - fcf(data)
rcall spiwrite ; send byte
ldi r16,$88 ; load data byte - fcf
rcall spiwrite ; send byte
ldi r16,$04 ; load data byte - seq(backlogged data command)
rcall spiwrite ; send byte
ldi r16,low(pan_id) ; load data byte - des_panid
rcall spiwrite ; send byte
ldi r16,high(pan_id) ; load data byte - des_panid
rcall spiwrite ; send byte
mov r16,r11 ; load data byte - des_addr
rcall spiwrite ; send byte
mov r16,r10 ; load data byte - des_addr
rcall spiwrite ; send byte
ldi r16,low(src_addr) ; load data byte - src_addr
rcall spiwrite ; send byte
ldi r16,high(src_addr) ; load data byte - src_addr
rcall spiwrite ; send byte

dataload: ; load data to rf unit

ld r16,-y ; load data byte - data byte
rcall spiwrite ; send byte
cpi yl,$00 ; check if last byte
brne dataload ; branch if not last byte
sbi portb,portb0 ; pull ss high
ret ; return to previous duty


transmitpacket: ; transmit a packet

sbi portb,portb4 ; pull slp_tr high to begin transmit
nop ; delay to make sure the pin is high long enough
cbi portb,portb4 ; pull slp_tr low

ldi r16,$60 ; load data byte - write buffer command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$13 ; load frame length byte - total bytes excluding this one, +2
rcall spiwrite ; send byte
ldi r16,$61 ; load fcf(7:0) byte - $61 for data packet
rcall spiwrite ; send byte
ldi r16,$88 ; load fcf(15:8) byte - $88 for data packet
rcall spiwrite ; send byte
ldi r16,$02 ; load sequence byte - command(data)
rcall spiwrite ; send byte
ldi r16,low(pan_id) ; load destination panid(7:0) byte
rcall spiwrite ; send byte
ldi r16,high(pan_id) ; load destination panid(15:8) byte
rcall spiwrite ; send byte
mov r16,r11 ; load destination address(7:0) byte
rcall spiwrite ; send byte
mov r16,r10 ; load destination address(15:8) byte
rcall spiwrite ; send byte
ldi r16,low(src_addr) ; load source address(7:0) byte
rcall spiwrite ; send byte
ldi r16,high(src_addr) ; load source address(15:8) byte
```

```
rcall spiwrite ; send byte
mov r16,r25 ; load data byte - button press
rcall spiwrite ; send byte
mov r16,r2 ; load data byte - humidity
rcall spiwrite ; send byte
mov r16,r1 ; load data byte - temperature0
rcall spiwrite ; send byte
mov r16,r0 ; load data byte - temperature1
rcall spiwrite ; send byte
mov r16,r4 ; load data byte - accelerometer lsb
rcall spiwrite ; send byte
mov r16,r3 ; load data byte - accelerometer msb
rcall spiwrite ; send byte
mov r16,r7 ; load data byte - light lsb
rcall spiwrite ; send byte
mov r16,r6 ; load data byte - light msb
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
ret ; return to previous duty
```

Appendix G

Control Node Firmware: Damper Motor

```
.include "m1281def.inc"

; increment of 4 for src_addr
; csma_seed = 8 x src_addr
.equ src_addr = $aab8 ; place unique source address here
.equ csma_seed = low(src_addr)*$08 ; random backoff exponent - must be unique,<$0800
.equ home_addr = $aab1 ; local node des_addr
.equ pan_id = $abcd ; system pan_id
.equ retries = $38 ; high nibble = frame retries,low nibble = 2x(csma retries)
.equ channel = $36 ; tx/rx channel, $2b -> $3a valid
.equ stepsize = $08 ; control stepsize
.equ hysteresis = $01 ; hysteresis on control setpoint
.equ delay = $02 ; motor turn on delay msb (32.768ms per bit)
.equ motor_time = $05 ; number of 128us intervals in motor on-time increments ($05 = 164ms)

.org 0
rjmp start
.org int4addr
rjmp wakeupshit15
.org int5addr
rjmp clearirq
.org OVF2addr
rjmp wakeup
.org ADCCaddr
rjmp wakeupadc
.org OC5Aaddr
rjmp timer5int

; lightsensor = pf0(adc0)
; shit15 data = pe4(int4)
; shit15 clock = pe0
; wind sensor = pd6(t1)
; current threshold = pe7(int7)
; direction = pg1
; brake = pg0
; pwm = pb7(oc0a)
; r0  frame length temporary register - write
; r1  frame length temporary register - read
; r2  adc msb temporary register
; r3  shit15 msb temporary register
; r4  shit15 lsb temporary register
; r5  adc lsb temporary register
; r6  control setpoint bottom
; r7  control setpoin top
; r8  wind sensor lsb
; r9  wind sensor msb
; r10 des_addr msb
; r11 des_addr lsb
; r12 shit15 humidity temporary register
; r13 ed_level buffer
; r14 motor state register for transmit
; r15
; r16 temporary swap register for interrupts
; r17 temporary swap register for main
; r18 rf state register
; r19 motor state register
; r20 wait loop register
; r21 wait loop register
; r22 motor timer advance register
; r23
; r24 temporary swap register for main
; r25 wakeup counter
; xregister is recieve buffer (r27(xh),r26(xl))
; yregister is buffer start (r29(yh),r28(yl))
; zregister is buffer end (r31(zh),r30(zl))
; data buffer set at 1k
; tregister is shit15 humidity/temperature irq differentiator
; joinrequest buffer is 256byte
; recieve buffer is 256byte

start: ; configure microcontroller registers
; start up time 6CK + 65ms
; set to 8MHz by internal rc oscillator
;turn off watchdog timer in case of watchdog reset
wdr ; reset wdt
ldi r16,high(RAMEND)
out SPH,r16 ; Set Stack Pointer to top of RAM
ldi r16,low(RAMEND)
out SPL,r16
ldi r16,$17
out ddrb,r16 ; set ss,sclk,mosi,slp_tr as output
ldi r16,$68
out portb,r16 ; enable pullup for miso and pb5,pb6 for buttons
ldi r16,$01
out spsr,r16 ; set up spi - mode0, ck/2
ldi r16,$50
out spcr,r16 ; set up spi - mode0, ck/2 (4MHz)
sbi portb,portb0 ; set ss pin high
cbi portb,portb4 ; set slp_tr low
sbi porta,porta7 ; pull reset high on rf unit
sbi ddra,dda7 ; set reset pin as output
ldi r16,$0c
sts eicrb,r16 ; configure int5 for rising edge
ldi r16,$20
out eimsk,r16 ; enable int5
ldi r16,high(home_addr) ; set up des_addr to something in system for start
mov r10,r16
ldi r16,low(home_addr) ; set up des_addr
mov r11,r16
clr zl ; set buffer end to beginning of sram
clr yl ; set buffer start to beginning of sram
ldi zh,$02 ; set buffer end to beginning of sram
ldi yh,$02 ; set buffer start to beginning of sram
ldi xh,$05 ; set recieve buffer to middle of sram
clr xl ; set recieve buffer to middle of sram
ldi r23,$ff ; initialize receieve buffer count
ldi r19,$80 ; set motor state register to damper-motor
clr r18 ; reset rf state register
clr r14 ; reset tx motor state register
ldi r25,$1e ; set wakeup counter to 60s
ldi r16,$3f
out portd,r16 ; turn on pullups for unused pins on portd since they might have long cables attached

;turn off watchdog timer in case of watchdog reset
wdr ; reset wdt
ldi r16,$00 ; clear all resets
out mcusr,r16 ; turn off wdrf
ldi r16,$1e
sts wdtcsr,r16 ; enable writing of wdt
ldi r16,$06
sts wdtcsr,r16 ; turn off wdt, 1s timer
```

```
;wait for rf unit to stabilize
ldi r21,$04 ; load the wait timer to 3 cycles (x3084/ck)
rcall waitloop1 ; wait ~(9252/8MHz = 1.2ms) for rf unit to stabilize

cbi porta,porta7 ; pull reset low to reset rf unit if watchdog reset
nop ; delay so reset has time to activate
nop
nop
nop
nop
nop
nop
nop
nop
nop
sbi porta,porta7 ; pull reset high to start

;wait for rf unit to stabilize
ldi r21,$04 ; load the wait timer to 3 cycles (x3084/ck)
rcall waitloop1 ; wait ~(9252/8MHz = 1.2ms) for rf unit to stabilize

;setup analog inputs
ldi r16,$01
sts didr0,r16 ; turn off input stage for adc0 pin
ldi r16,$c0
sts admux,r16 ; use internal 2.56v as voltage reference,sample adc0

;setup motor control pins as output (no pwm at the moment)
cbi portg,portg1 ; make sure pins are off
cbi portb,portb7 ; turn pwm off
sbi portg,portg0 ; turn brake on
sbi ddrg,ddg1 ; direction
sbi ddrg,ddg0 ; brake
sbi ddrb,ddb7 ; pwm (on/off for now)
sbi portg,portg0 ; turn on brake pin to shut off transistors

;setup t1 as windspeed counter
ldi r16,$06
sts tccr1b,r16 ; set t1 to external clock on t0 falling edge

;setup t2 as rtc at 4s interval wakeup
ldi r16,$20
sts assr,r16 ; set t2 to assynchronous mode
ldi r16,$06
sts tccr2b,r16 ; set t2 prescaler to /256 - 2s wakeup period

;make sure t2 is done rewriting itself
checkassr:

lds r16,assr
andi r16,$1f ; check assr(4:0) clear
brne checkassr

ldi r16,$07
out tifr2,r16 ; clear all pending interrupts
ldi r16,$01
sts timsk2,r16 ; enable t2 overflow interrupt

;setup shit15 - pe0 is data, pe4 is clock
sbi ddre,dde0 ; set clock as output
cbi porte,porte4 ; make sure pullups are off for data
rcall shit15setup

;configure rf unit registers

;go to trx_off state
rcall txoff ; go to trx_off state on rf unit

;setup clkm
ldi r16,$c3 ; load first data byte - write trx_ctrl_0 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
clr r16 ; load second data byte - turn off clkm
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup short_addr_0
ldi r16,$e0 ; load first data byte - write short_addr_0 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,low(src_addr) ; load second data byte - src_addr lsb
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup short_addr_1
ldi r16,$e1 ; load first data byte - write short_addr_1 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,high(src_addr) ; load second data byte - src_addr msb
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup pan_id_0
ldi r16,$e2 ; load first data byte - write pan_id_0 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,low(pan_id) ; load second data byte - pan_id lsb
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup pan_id_1
ldi r16,$e3 ; load first data byte - write pan_id_1 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,high(pan_id) ; load second data byte - pan_id msb
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup phy_cc_ca - set channel id
ldi r16,$c8 ; load byte - write phy_cc_ca register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,channel ; load data byte -
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup phy_tx_pwr - turn crc on
ldi r16,$c5 ; load byte - write phy_tx_pwr register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
```

```
ldi r16,$80 ; load data byte - auto_crc=on,pwr=+3dbm
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup xah_ctrl - set frame and csma retries
ldi r16,$ec ; load byte - write xah_ctrl register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,retries ; load data byte - frame and csma retries
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup csma_seed_1
ldi r16,$ee ; load byte - write csma_seed_1 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,high(csma_seed) ; load data byte - min_be=0,aack_set=0,i_am_coord=0,csma(10:8)
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup csma_seed_0
ldi r16,$ed ; load first data byte - write csma_seed_1 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,low(csma_seed) ; load data byte - csma(7:0)
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;enable irqs
ldi r16,$ce ; load first data byte - write register irq_mask command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$08 ; load second data byte - trx_end only
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

;clear pending irqs on rf unit
ldi r16,$8f ; load first data byte - read register irq_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte and clear pending irqs
sbi portb,portb0 ; pull ss high

;clear pending irqs on micro
ldi r16,$ff
out eifr,r16 ; clear any interrupt flags that are set

;enable interrupts on micro
sei ; turn on interrupts

rcall rxonaack ; put the rf unit in rx_aack_on state

repeat: ; handle backlogged data

cpse zh,yh ; check if buffer has data
rjmp handle ; go to data handler
cpse zl,yl ; check if buffer has data
rjmp handle ; go to data handler
in r17,pinb ; get button press data
andi r17,$60 ; mask off button bits
ldi r24,$60 ; invert button bits
eor r17,r24 ; invert button bits
mov r24,r19 ; move current motor state to temp register
andi r24,$60 ; mask off button state
cp r24,r17 ; check if button state has changed
breq repeat ; continue checking if no change
andi r19,$9b ; turn off control bit and clear button state in motor state register
ori r19,$08 ; set manual bit in motor state register
or r19,r17 ; set current button state in motor state register
or r14,r17 ; store button presses to tx motor state register
cpi r17,$60 ; check if both buttons are pressed
breq buttonstop
cpi r17,$40 ; check if open button is pressed
breq openbutton
cpi r17,$20 ; check if close button is pressed
breq closebutton
andi r19,$97 ; else shut off manual bit and button bits in motor state register

buttonstop: ; turn off motor via buttonpress

ldi r17,$08 ; turn off timer
sts tccr5b,r17 ; turn off timer
ldi r17,$00
sts timsk5,r17 ; disable t5 interrupts
cbi portb,portb7 ; stop motor
sbi portg,portg0 ; turn on brake
rjmp repeat ; return to state checking

openbutton: ; open motor via buttonpress

ldi r17,$08 ; turn off timer
sts tccr5b,r17 ; turn off timer
ldi r17,$00
sts timsk5,r17 ; disable t5 interrupts
cbi portg,portg0 ; turn brake off
sbi portg,portg1 ; set direction to open
sbi portb,portb7 ; turn on motor
ori r19,$01 ; set direction open
rjmp repeat ; return to state checking

closebutton: ; close motor via buttonpress

ldi r17,$08 ; turn off timer
sts tccr5b,r17 ; turn off timer
ldi r17,$00
sts timsk5,r17 ; disable t5 interrupts
cbi portg,portg0 ; turn brake off
cbi portg,portg1 ; set direction to closed
sbi portb,portb7 ; turn on motor
andi r19,$fe ; set direction closed
rjmp repeat ; return to state checking

handle: ; respond to recieved data packets

ld r17,z+ ; get data frame length from buffer
mov r1,r17 ; move frame length to counter
cpi r17,$08 ; check if packet is the right length
brne resetbuffer ; reset buffer if not
ld r17,z+ ; get header byte from buffer
dec r1
cpi r17,$08 ; check if right packet type
brne resetbuffer
ld r17,z+ ; lsb src_addr - ignore
```

```
dec r1
ld r17,z+ ; msb src_addr - ignore
dec r1
ld r17,z+ ; get command byte
dec r1
cpi r17,$08 ; check if motor control command
breq motor ; control motor
rjmp resetbuffer ; else finish off

motor: ; control motor

ld r17,z+ ; get motor command
dec r1
sbrc r19,3 ; do not modify if under manual control
rjmp resetbuffer ; commands past this point are overriden by manual control
cpi r17,$08 ; check if damper full open
breq open
cpi r17,$06 ; check if damper full close
breq close
cpi r17,$0a ; check if damper stop
breq stop
cpi r17,$0c ; check if damper set by speed
brne resetbuffer ; else end
ld r17,z+ ; get control setpoint
dec r1
cpi r17,(hysteresis + $01) ; check if setpoint is too low
brlo setpointclose ; completely shut
cpi r17,($ff - hysteresis) ; check if setpoint is too high
brsh setpointopen ; completely open
subi r17,hysteresis ; put hysterisis in
mov r6,r17 ; move setpoint to control bottom
subi r17,($00 - 2*(hysteresis)) ; put hysterisis in
mov r7,r17 ; move setpoint to control top
ori r19,$04 ; turn on control bit in motor state register
ldi r17,$08 ; turn off timer
sts tccr5b,r17 ; turn off timer
ldi r17,$00
sts timsk5,r17 ; disable t5 interrupts
cbi portb,portb7 ; stop motor
sbi portg,portg0 ; turn on brake
rjmp resetbuffer

resetbuffer: ; reset buffer pointer

ldi r17,$00
add zl,r1 ; add remaining packet count
adc zh,r17 ; increment zh if zl overflow
sbrc zh,2 ; check if buffer at 1k
ldi zh,$02 ; reset buffer pointer to bottom of buffer
rjmp repeat ; get next packet from buffer

open: ; open damper command

andi r19,$fb ; turn off control bit
ld r17,z+ ; get step size packet
dec r1
rcall motortime ; start motor timer
cbi portg,portg0 ; turn brake off
sbi portg,portg1 ; set direction to open
sbi portb,portb7 ; turn on motor
ori r19,$01 ; set direction open
rjmp resetbuffer

stop: ; stop motor

ldi r17,$08 ; turn off timer
sts tccr5b,r17 ; turn off timer
ldi r17,$00
sts timsk5,r17 ; disable t5 interrupts
andi r19,$fb ; turn off control bit
cbi portb,portb7 ; stop motor
sbi portg,portg0 ; turn on brake
rjmp resetbuffer

close: ; close damper command

andi r19,$fa ; turn off control bit and set direction to closed
ld r17,z+ ; get step size packet
dec r1
rcall motortime ; start motor timer
cbi portg,portg0 ; turn brake off
cbi portg,portg1 ; set direction to closed
sbi portb,portb7 ; turn on motor
rjmp resetbuffer

setpointclose: ; go to full closed because setpoint is too low for control

andi r19,$fa ; turn off control bit and set direction closed
ldi r17,$ff
rcall motortime
cbi portg,portg0 ; turn brake off
cbi portg,portg1 ; set direction to closed
sbi portb,portb7 ; turn on motor
rjmp resetbuffer

setpointopen: ; go to full open because setpoint is too high for control

andi r19,$fb ; turn off control bit
ldi r17,$ff
rcall motortime
cbi portg,portg0 ; turn brake off
sbi portg,portg1 ; set direction to open
sbi portb,portb7 ; turn on motor
ori r19,$01 ; set direction open
rjmp resetbuffer

motortime: ; have motor on for a certain amount of time

mov r22,r17 ; move motortime to counter
ldi r17,(motor_time)
sts ocr5ah,r17 ; set counter top to packet time
ldi r17,$00
sts ocr5al,r17 ; set counter top to packet time
sts tcnt5h,r17 ; set counter to zero
sts tcnt5l,r17 ; set counter to zero
ldi r17,$2f
out tifr5,r17 ; clear all t5 interrupt flags
ldi r17,$02
sts timsk5,r17 ; turn on t5 interrupt
ldi r17,$0d
sts tccr5b,r17 ; turn on counter, set to 128us period
ret
```

```
timer5int: ; turn off motor

in r16,sreg ; get sreg
push r16 ; push sreg on stack
dec r22 ; decrement 3rd byte counter
brne timer5intdone ; dont do anything if not done
cbi portb,portb7 ; stop motor
sbi portg,portg0 ; turn on brake
ldi r16,$08 ; turn off timer
sts tccr5b,r16 ; turn off timer
ldi r16,$00
sts timsk5,r16 ; disable t5 interrupts

timer5intdone: ; finish off

pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; return from interrupt

txoff: ; send trx_off command to rf unit and wait till stable

ldi r16,$c2 ; load first data byte - write register trx_state command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$08 ; load second data byte - data trx_off bit
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

checktxoff: ; check current state to see if in tx_off state

ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$08
brne checktxoff
ret ; return to previous duty

pllon: ; send pll_on command to rf unit

ldi r16,$c2 ; load first data byte - write register trx_state command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$09 ; load second data byte - data pll_on bit
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

checkpllon: ; check current state to see if in pll_on state

ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$09
brne checkpllon
ret ; return to previous duty

rxonaack: ; send rx_aack_on command to rf unit and wait till stable

ldi r16,$c2 ; load first data byte - write register trx_state command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$16 ; load second data byte - data rx_aack_on bit
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

checkrxonaack: ; check current state to see if in rx_aack_on state

ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$16
brne checkrxonaack
ret ; return to previous duty

txonaret: ; send trx_aret_on command to rf unit and wait till stable

ldi r16,$c2 ; load first data byte - write register trx_state command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$19 ; load second data byte - data trx_aret_on bit
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

checktxonaret: ; check current state to see if in tx_aret_on state

ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$19
brne checktxonaret
ret ; return to previous duty

joinrecieve: ; get new des_addr

cpi r18,$04 ; check if waiting for joinack
brne rxrequest ; dont accpet joinack if not waiting for joinack
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_panid(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_panid(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_addr(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_addr(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - src_addr(7:0)
mov r11,r16
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - src_addr(15:8)
mov r10,r16
```

```
;dont bother with the rest of the data - it doesnt matter
;ldi r16,$00 ; load data byte - blank for reading
;rcall spiwrite ; send byte - do not save fcs(7:0)
;ldi r16,$00 ; load data byte - blank for reading
;rcall spiwrite ; send byte - do not save fcs(15:8)
;ldi r16,$00 ; load data byte - blank for reading
;rcall spiwrite ; send byte - lqi
sbi portb,portb0 ; pull ss high
sbiw yh:yl,$02 ; reset data buffer to begining of join request
ldi r18,$10 ; set state register to retry transmit
rcall transmit ; set rf unit to tx_on_aret state
rcall transmitpacket ; transmit data
ret ; done

rxreadinterrupt: ; empty rx buffer while in interrupt

;ed_level is taken first because it is only valid for 224us after interrupt fires
ldi r16,$87 ; load data byte - read ed_level
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load data byte - blank byte for reading
rcall spiwrite ; send byte
mov r13,r16 ; move ed_level to temporary buffer
sbi portb,portb0 ; pull ss high

;get data from recieve buffer
ldi r16,$20 ; load data byte - read buffer command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - frame length byte
subi r16,$06 ; decrease frame length for bits to be removed
st y+,r16 ; move frame length to buffer
subi r16,$03 ; decrease frame length for buffer fitting
mov r0,r16 ; move frame length to temporary register
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcf(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcf(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - seq (used as command byte)
st y+,r16
cpi r16,$06 ; check if command byte is $06 (join acknowledge)
breq joinrecieve ; get new home_addr if $06
cpi r16,$08 ; check if data to tethered node
breq notrequest1 ; load data to buffer, else discard packet

rxrequest: ; finish off

sbi portb,portb0 ; pull ss high
sbiw yh:yl,$02 ; reset data buffer to begining of join request
ret ; done

notrequest1: ; continue with data loading

ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_panid(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_panid(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_addr(7:0)
```

```
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_addr(15:8)

getdatarf1: ; keep loading data till frame compelete

ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte
st y+,r16 ; move byte to buffer
dec r0 ; decrement frame length byte
brne getdatarf1 ; keep decrementing until done
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcs(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcs(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - lqi
sbi portb,portb0 ; pull ss high
st y+,r16 ; move lqi to buffer
st y+,r13 ; move ed_level to buffer
sbrc yh,2 ; check if buffer at 1k
ldi yh,$02 ; reset buffer pointer to bottom of buffer
ret ; done writing data

transmit: ; set rf unit to tx_on_aret state

;check to see if not in rx_busy state
ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$11 ; check if in busy_rx
breq transmit ; keep checking till in a new state
cpi r16,$1f ; check if in state transition
breq transmit ; keep checking till in a new state
cpi r16,$12 ; check if in busy_tx state
breq transmit ; keep checking till in a new state
rcall pllon ; go to pll on state
;check if there is data pending in rx buffer
ldi r16,$8f ; load first data byte - read register irq_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
sbrc r16,3 ; check if trx_end interrupt flag is set
rcall rxreadinterrupt ; empty rx buffer
rcall txonaret ; put the rf unit in tx_aret_on state
ret ; return to previous duty

waitloop1: ; wait timer (~3084 cycles per value in r21)

ser r20
rcall wait2
nop
nop
nop
nop
nop
nop
```

```
nop
nop
dec r21
brne waitloop1
ret ; return to previous duty

spiwrite: ; read and write data over spi

out spdr,r16 ; write transmitted byte to spi

wait: ; read spi register to check when done

in r16,spsr
sbrs r16,spif
rjmp wait
in r16,spdr ; write recieved byte to spi register
ret ; return to previous duty

wait2: ; wait timer (~12 cylces per value in r20)

nop
nop
nop
nop
nop
nop
nop
nop
dec r20
brne wait2
ret ; return to previous duty

gotorx: ; return to rx_on_aack state

;check successful transmit before switching back to rx
ldi r16,$82 ; load data byte - read trx_state to get trac_status
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load data byte - blank byte for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
andi r16,$e0 ; mask off trac_status
cpi r16,$60 ; check if success or data_pending success
brlo gotorx1 ; skip if success
cpi r18,$01 ; check if this is the first failure
brne gotorx3 ; if not - finish off
rcall transmitrequest ; get new des_addr and retransmit
ldi r18,$02 ; set joinrequest state bit
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; clear interrupt and return to previous duty

gotorx1: ; turn on rx_on_aack state

rcall pllon ; go to pll_on state
rcall rxonaack ; go to rx_on_aack state
cpi r18,$02 ; check if last state was a joinrequest
brne gotorx2 ; finish off if not waiting for joinrequest ack
ldi r18,$04 ; set rf state to joinrequest ack waiting
;turn off watchdog timer
wdr ; reset wdt
```

```
in r16,mcusr ; get reset flag status
andi r16,$f7 ; mask off wdrf to 0
out mcusr,r16 ; turn off wdrf
ldi r16,$1e
sts wdtcsr,r16 ; enable writing of wdt
ldi r16,$06
sts wdtcsr,r16 ; turn off wdt, 1s timer
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; return to previous task

gotorx3: ; set to rx_an_aack state

rcall pllon ; go to pll_on state
rcall rxonaack ; go to rx_on_aack state

gotorx2: ; finish off

clr r14 ; reset tx motor state register
ldi r18,$00 ; reset rf state to complete
;turn off watchdog timer
wdr ; reset wdt
in r16,mcusr ; get reset flag status
andi r16,$f7 ; mask off wdrf to 0
out mcusr,r16 ; turn off wdrf
ldi r16,$1e
sts wdtcsr,r16 ; enable writing of wdt
ldi r16,$06
sts wdtcsr,r16 ; turn off wdt, 1s timer
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; return to previous task

clearirq: ; clear irq register and irq line

in r16,sreg ; get sreg
push r16 ; push sreg on stack
ldi r16,$8f ; load first data byte - read register irq_status command to clear interrupt
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

checkrf: ; check rf unit state

ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$16 ; check if in rx_on
breq datacollect ; get data
cpi r16,$11 ; check if in busy_rx
breq datacollect ; get data
cpi r16,$1f ; check if in state transition
breq checkrf ; keep checking till in a new state
cpi r16,$12 ; check if in busy_tx state
breq checkrf ; keep checking till in a new state
rjmp gotorx ; else change state back to rx

;removed checkcrc because crc is automatically checked with rx_aack
```

```
datacollect: ; get data from rf unit

rcall rxreadinterrupt ; get data from rf unit
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; done writing data

wakeup: ; check if time to take data

in r16,sreg ; get sreg
push r16 ; push sreg on stack
dec r25 ; decrement wakeup counter
breq wakeup1 ; take data if 60s
pop r16 ; get sreg off stack
out sreg,r16 ; return sreg
reti

wakeup1: ; take data and return to previous task

;enable watchdog timer to eliminate lock up bug
wdr ; reset wdt
ldi r16,$1e
sts wdtcsr,r16 ; enable writing of wdt
ldi r16,$0e
sts wdtcsr,r16 ; turn on wdt, 1s timer

push r17 ; push r17 on stack
lds r8,tcnt1l ; move windspeed lsb to temporary register
lds r9,tcnt1h ; move windspeed msb to temporary register
clr r16 ; reset activity level counter
sts tcnt1h,r16
sts tcnt1l,r16
;motor control code
sbrs r19,2 ; check if in control mode
rjmp wakeupend ; finish off if not
cp r9,r6 ; check if speed too low
brlo incrementup ; move motor open
cp r9,r7 ; check if speed to high
brsh incrementdown ; move motor closed

wakeupend: ; else finish off

or r14,r19 ; move motor state register to tx motor state register
rcall shit15temp ; initiate shit15 read - humidity included
ldi r16,$df ; take measurement from adc0
sts adcsra,r16 ; turn on adc,clear interrupt flag,enable interrupt,set to /128 (62.5kHz)
ldi r25,$1e ; reset wakeup counter to 60s
pop r17 ; get r17 off stack
pop r16 ; get sreg off stack
out sreg,r16 ; return sreg
reti

incrementup: ; open with motor a few steps

ldi r17,stepsize ; set increment length
rcall motortime ; move packet to motor timer
cbi portg,portg0 ; turn brake off
sbi portg,portg1 ; set direction to open
sbi portb,portb7 ; turn on motor
ori r19,$01 ; set direction open
rjmp wakeupend ; finish off

incrementdown: ; close with motor a few steps

ldi r17,stepsize ; set increment length
rcall motortime ; move packet to motor timer
cbi portg,portg0 ; turn brake off
cbi portg,portg1 ; set direction to closed
sbi portb,portb7 ; turn on motor
andi r19,$fe ; set direction closed
rjmp wakeupend ; finish off

wakeupadc: ; wakeup and return to previous task

in r16,sreg ; get sreg
push r16 ;  push sreg on stack
lds r5,adcl ; load lsb to r5
lds r2,adch ; load msb to r2
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti

shit15setup: ; set the device to low res mode
;talk to shit15, only works for cpu clock <=2MHZ
;code modded with nops to operate at 8MHz
;data pin can never be output high - leave pullup off
;needs external pullup resistor for proper operation
;sleep operation disabled for this application
;always read temperature before humidity

rcall shit15reset ; reset the device in case its fuxxored
rcall startshit15 ; send start sequence

;data sequence - register write command $06
;nops added to keep clock at 1MHz
sbi porte,porte0 ; clock high - data bit one
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit two
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit three
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit four
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit five
nop
```

```
nop
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; data high
nop ; data takes time to rise and needs setup time
nop
nop
nop
nop
nop
sbi porte,porte0 ; clock high - data bit six
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit seven
nop
nop
cbi porte,porte0 ; clock low
sbi ddre,dde4 ; data low
sbi porte,porte0 ; clock high - data bit eight
nop
nop
cbi porte,porte0 ; clock low

rcall shit15ack0 ; wait for ack from shit15

;data sequence - register write data $01
;nops added to keep clock at 1MHz
sbi ddre,dde4 ; data low
sbi porte,porte0 ; clock high - data bit one
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit two
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit three
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit four
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit five
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit six
```

```
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit seven
nop
nop
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; data high
nop ; data takes time to rise and needs setup time
nop
nop
nop
nop
nop
sbi porte,porte0 ; clock high - data bit eight
nop
nop
cbi porte,porte0 ; clock low

rcall shit15ack0 ; wait for ack from shit15
ret ;done setting up the device

shit15temp: ; read temperature data from shit15
;send read temp commmand
rcall startshit15 ; send start sequence

;data sequence - read temperature command $03
sbi porte,porte0 ; clock high - data bit one
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit two
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit three
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit four
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit five
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit six
nop
```

```
nop
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; data high
nop ; data takes time to rise and needs setup time
nop
nop
nop
nop
nop
sbi porte,porte0 ; clock high - data bit seven
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit eight
nop
nop
cbi porte,porte0 ; clock low
;sbi ddre,dde4 ; data low - if last databyte is 1 - leave it high for ack

rcall shit15ack0 ; wait for ack from shit15

sbi eimsk,4 ; enable int4
ret ; return to previous task and wait for shit15 to be done

wakeupshit15: ; collect data from shit15
in r16,sreg ; get sreg
push r16 ; push sreg on stack
brts shit15humread ; skip if its a humidity read
cbi eimsk,4 ; disable int4 as pe4 will still be low when returning
rcall shit15read ; read out temperature data from shit15 (msb)
rcall shit15ack3 ; acknowledge reciept of bit
mov r3,r16 ; move msb to another register
rcall shit15read ; read out temperature data from shit15 (lsb)
rcall shit15ack4 ; end transmission ack
mov r4,r16 ;  move lsb to another register
rcall shit15humidity
pop r16 ; get off stack
out sreg,r16 ; return sreg
set ; set tregister to indicate humidity request in progress
reti ; continue with previous task

shit15humread: ; read out humidity data from shit 15
rcall shit15read ; read out humidity data from shit15 (msb)
rcall shit15ack3 ; acknowledge reciept of bit
rcall shit15read ; read out humidity data from shit15 (lsb)
rcall shit15ack4 ; end transmission ack
mov r12,r16 ; move humidity data to temporary register
ldi r18,$01 ; set rf state to first transmit
rcall transmit ; set rf unit to tx state
rcall transmitpacket ; send data off
pop r16 ; get off stack
out sreg,r16 ; return sreg
clt ; reset to temperature read
reti ; return with previous task

shit15humidity: ; read humidity data from shit15
;send read humidity command
rcall startshit15 ; send start sequence
```

```
;data sequence - read humidity command $05
;nops added to keep clock at 1MHz
sbi porte,porte0 ; clock high - data bit one
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit two
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit three
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit four
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit five
nop
nop
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; data high
nop ; data takes time to rise and needs setup time
nop
nop
nop
nop
nop
sbi porte,porte0 ; clock high - data bit six
nop
nop
cbi porte,porte0 ; clock low
sbi ddre,dde4 ; data low
sbi porte,porte0 ; clock high - data bit seven
nop
nop
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; data high
nop ; data takes time to rise and needs setup time
nop
nop
nop
nop
nop
sbi porte,porte0 ; clock high - data bit eight
nop
nop
cbi porte,porte0 ; clock low
;sbi ddre,dde4 ; data low - if last databyte is 1 - leave it high for ack

rcall shit15ack0 ; wait for acknowledge from shit15
sbi eimsk,4 ; enable int4
```

```
ret ; return to previous task and wait for shit15 to take data

shit15ack4: ; end transmission ack
;nops added to keep clock at 1MHz
sbi porte,porte0 ; clock high - ack
nop
nop
cbi porte,porte0 ; clock low
ret ; return to previous task

shit15ack3: ; data recipet acknowledge
;nops added to keep clock at 1MHz
sbi ddre,dde4 ; pull data low
sbi porte,porte0 ; clock high - ack
nop
nop
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; release data line
ret ; return to previous task

shit15read: ; read data from shit15
;nops added to keep clock at 1MHz and meet data valid time of 250ns
clr r16 ; clear the register where incoming data will be written
sbi porte,porte0 ; clock high - data bit one
sbic pine,pine4 ; check if data is low
sbr r16,$80 ; write data bit one to register
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit two
sbic pine,pine4 ; check if data is low
sbr r16,$40 ; write data bit two to register
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit three
sbic pine,pine4 ; check if data is low
sbr r16,$20 ; write data bit three to register
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit four
sbic pine,pine4 ; check if data is low
sbr r16,$10 ; write data bit four to register
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit five
sbic pine,pine4 ; check if data is low
sbr r16,$08 ; write data bit five to register
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit six
sbic pine,pine4 ; check if data is low
sbr r16,$04 ; write data bit six to register
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit seven
sbic pine,pine4 ; check if data is low
sbr r16,$02 ; write data bit seven to register
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit eight
sbic pine,pine4 ; check if data is low
sbr r16,$01 ; write data bit eight to register
cbi porte,porte0 ; clock low
ret ; return to previous activity

startshit15: ; start sequence
;nops added to keep clock at 1MHz
cbi ddre,dde4 ; set data high
cbi porte,porte0 ; make sure clock is low
nop
nop
sbi porte,porte0 ; pull clock high
sbi ddre,dde4 ; set data low
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
cbi ddre,dde4 ; data high
nop ; data takes time to rise and needs setup time
nop
nop
nop
nop
nop
cbi porte,porte0 ; clock low
sbi ddre,dde4 ; data low
ret ; return to previous task

shit15ack0: ; ack sequence for sending data

cbi ddre,dde4 ; release data line

shit15ack2: ; check that data line is low
;nops added to keep clock at 1MHz
sbic pine,pine4
rjmp shit15ack2
sbi porte,porte0 ; clock high - ack
nop
nop
cbi porte,porte0 ; clock low

shit15ack1: ; check that line has been released

sbis pine,pine4 ; check that line has been released
rjmp shit15ack1
ret ; return to previous task

shit15reset: ; reset sequence if it gets out of phase
;status register preserved
;must be followed by data start sequence
;nops added to slow it down to 1MHz
cbi ddre,dde4 ; data high
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
```

```
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
ret ; return to previous task

transmitrequest: ; transmit a request frame

sbi portb,portb4 ; pull slp_tr high to begin transmit
nop ; delay to make sure the pin is high long enough
cbi portb,portb4 ; pull slp_tr low
```

```
ldi r16,$60 ; load data byte - write buffer command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$0b ; load data byte - frame length byte
rcall spiwrite ; send byte
ldi r16,$40 ; load data byte - fcf(7:0) - broadcast,no ack
rcall spiwrite ; send byte
ldi r16,$88 ; load data byte - fcf(15:8) - short address,pan_ids equal
rcall spiwrite ; send byte
ldi r16,$01 ; load data byte - seq(7:0) - command(request)
rcall spiwrite ; send byte
ldi r16,low(pan_id) ; load data byte - pan_id(7:0)
rcall spiwrite ; send byte
ldi r16,high(pan_id) ; load data byte - pan_id(15:8)
rcall spiwrite ; send byte
ldi r16,$ff ; load data byte - destination address (7:0) - broadcast
rcall spiwrite ; send byte
ldi r16,$ff ; load data byte - destination address (15:0) - broadcast
rcall spiwrite ; send byte
ldi r16,low(src_addr) ; load data byte - source address (7:0)
rcall spiwrite ; send byte
ldi r16,high(src_addr) ; load data byte - source address (15:8)
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
ret ; return to previous duty

transmitpacket: ; transmit a packet

sbi portb,portb4 ; pull slp_tr high to begin transmit
nop ; delay to make sure the pin is high long enough
cbi portb,portb4 ; pull slp_tr low

ldi r16,$60 ; load data byte - write buffer command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$13 ; load frame length byte - total bytes excluding this one, +2
rcall spiwrite ; send byte
ldi r16,$61 ; load fcf(7:0) byte - $61 for data packet
rcall spiwrite ; send byte
ldi r16,$88 ; load fcf(15:8) byte - $88 for data packet
rcall spiwrite ; send byte
ldi r16,$07 ; load sequence byte - command (data from tethered node)
rcall spiwrite ; send byte
ldi r16,low(pan_id) ; load destination panid(7:0) byte
rcall spiwrite ; send byte
ldi r16,high(pan_id) ; load destination panid(15:8) byte
rcall spiwrite ; send byte
mov r16,r11 ; load destination address(7:0) byte
rcall spiwrite ; send byte
mov r16,r10 ; load destination address(15:8) byte
rcall spiwrite ; send byte
ldi r16,low(src_addr) ; load source address(7:0) byte
rcall spiwrite ; send byte
ldi r16,high(src_addr) ; load source address(15:8) byte
rcall spiwrite ; send byte
mov r16,r14 ; load data byte - motor state register
rcall spiwrite ; send byte
mov r16,r12 ; load data byte - humidity
rcall spiwrite ; send byte
mov r16,r4 ; load data byte - temperature0
```

```
rcall spiwrite ; send byte
mov r16,r3 ; load data byte - temperature1
rcall spiwrite ; send byte
mov r16,r8 ; load data byte - windspeed lsb
rcall spiwrite ; send byte
mov r16,r9 ; load data byte - windspeed msb
rcall spiwrite ; send byte
mov r16,r5 ; load data byte - light lsb
rcall spiwrite ; send byte
mov r16,r2 ; load data byte - light msb
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
ret ; return to previous duty
```

Appendix H

Control Node Firmware: Window Motor

```
.include "m1281def.inc"

; increment of 4 for src_addr
; csma_seed = 8 x src_addr
.equ src_addr = $aad4 ; place unique source address here
.equ csma_seed = low(src_addr)*$08 ; random backoff exponent - must be unique,<$0800
.equ home_addr = $aab1 ; local node des_addr
.equ pan_id = $abcd ; system pan_id
.equ retries = $38 ; high nibble = frame retries,low nibble = 2x(csma retries)
.equ channel = $36 ; tx/rx channel, $2b -> $3a valid
.equ stepsize = $0a ; control stepsize
.equ hysteresis = $08 ; hysteresis on control setpoint
.equ delay = $02 ; motor turn on delay msb (32.768ms per bit)
.equ motor_time = $0f ; number of 128us intervals in motor on-time increments ($0f = 492ms)

.org 0
rjmp start
.org int4addr
rjmp wakeupshit15
.org int5addr
rjmp clearirq
.org int7addr
rjmp currentstop
.org OVF2addr
rjmp wakeup
.org ADCCaddr
rjmp wakeupadc
.org OC4Aaddr
rjmp timer4int
.org OC5Aaddr
rjmp timer5int

; lightsensor = pf0(adc0)
; shit15 data = pe4(int4)
; shit15 clock = pe0
; wind sensor = pd6(t1)
; current threshold = pe7(int7)
; direction = pg1
; brake = pg0
; pwm = pb7(oc0a)
; r0  frame length temporary register - write
; r1  frame length temporary register - read
; r2  adc msb temporary register
; r3  shit15 msb temporary register
; r4  shit15 lsb temporary register
; r5  adc lsb temporary register
; r6  control setpoint bottom
; r7  control setpoin top
; r8  wind sensor lsb
; r9  wind sensor msb
; r10 des_addr msb
; r11 des_addr lsb
; r12 shit15 humidity temporary register
; r13 ed_level buffer
; r14 motor state register for transmit
; r15
; r16 temporary swap register for interrupts
; r17 temporary swap register for main
; r18 rf state register
; r19 motor state register
; r20 wait loop register
; r21 wait loop register
; r22 motor time counter
; r23
; r24 temporary swap register for main
; r25 wakeup counter
; xregister is recieve buffer (r27(xh),r26(xl))
; yregister is buffer start (r29(yh),r28(yl))
; zregister is buffer end (r31(zh),r30(zl))
; data buffer set at 1k
; tregister is shit15 humidity/temperature irq differentiator
; joinrequest buffer is 256byte
; recieve buffer is 256byte

start: ; configure microcontroller registers
; start up time 6CK + 65ms
; set to 8MHz by internal rc oscillator
ldi r16,high(RAMEND)
out SPH,r16 ; Set Stack Pointer to top of RAM
ldi r16,low(RAMEND)
out SPL,r16
ldi r16,$17
out ddrb,r16 ; set ss,sclk,mosi,slp_tr as output
ldi r16,$68
out portb,r16 ; enable pullup for miso and pb5,pb6 for buttons
ldi r16,$01
out spsr,r16 ; set up spi - mode0, ck/2
ldi r16,$50
out spcr,r16 ; set up spi - mode0, ck/2 (4MHz)
sbi portb,portb0 ; set ss pin high
cbi portb,portb4 ; set slp_tr low
sbi porta,porta7 ; pull reset high on rf unit
sbi ddra,dda7 ; set reset pin as output
ldi r16,$cc
sts eicrb,r16 ; configure int5 and int7 for rising edge
ldi r16,$a0
out eimsk,r16 ; enable int5 and int7
ldi r16,high(home_addr) ; set up des_addr to something in system for start
mov r10,r16
ldi r16,low(home_addr) ; set up des_addr
mov r11,r16
clr zl ; set buffer end to beginning of sram
clr yl ; set buffer start to beginning of sram
ldi zh,$02 ; set buffer end to beginning of sram
ldi yh,$02 ; set buffer start to beginning of sram
ldi xh,$05 ; set recieve buffer to middle of sram
clr xl ; set recieve buffer to middle of sram
ldi r23,$ff ; initialize receieve buffer count
clr r19 ; reset motor state register
clr r18 ; reset rf state register
clr r14 ; reset tx motor state register
ldi r25,$08 ; set wakeup counter to 64s
ldi r16,$3f
out portd,r16 ; turn on pullups for unused pins on portd since they might have long cables attached

;wait for rf unit to stabilize
ldi r21,$04 ; load the wait timer to 3 cycles (x3084/ck)
rcall waitloop1 ; wait ~(9252/8MHz = 1.2ms) for rf unit to stabilize

;setup analog inputs
ldi r16,$01
```

```
sts didr0,r16 ; turn off input stage for adc0 pin
ldi r16,$c0
sts admux,r16 ; use internal 2.56v as voltage reference,sample adc0

;setup motor control pins as output (no pwm at the moment)
cbi portg,portg1 ; make sure pins are off
cbi portb,portb7 ; turn pwm off
sbi portg,portg0 ; turn brake on
sbi ddrg,ddg1 ; direction
sbi ddrg,ddg0 ; brake
sbi ddrb,ddb7 ; pwm (on/off for now)
sbi portg,portg0 ; turn on brake pin to shut off transistors

;setup t1 as windspeed counter
ldi r16,$06
sts tccr1b,r16 ; set t1 to external clock on t0 falling edge

;setup t2 as rtc at 8s interval wakeup
ldi r16,$20
sts assr,r16 ; set t2 to assynchronous mode
ldi r16,$07
sts tccr2b,r16 ; set t2 prescaler to /1024 - 8s wakeup period

;make sure t2 is done rewriting itself
checkassr:

lds r16,assr
andi r16,$1f ; check assr(4:0) clear
brne checkassr

ldi r16,$07
out tifr2,r16 ; clear all pending interrupts
ldi r16,$01
sts timsk2,r16 ; enable t2 overflow interrupt

;setup shit15 - pe0 is data, pe4 is clock
sbi ddre,dde0 ; set clock as output
cbi porte,porte4 ; make sure pullups are off for data
rcall shit15setup

;configure rf unit registers

;go to trx_off state
rcall txoff ; go to trx_off state on rf unit

;setup clkm
ldi r16,$c3 ; load first data byte - write trx_ctrl_0 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
clr r16 ; load second data byte - turn off clkm
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup short_addr_0
ldi r16,$e0 ; load first data byte - write short_addr_0 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,low(src_addr) ; load second data byte - src_addr lsb
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high
```

```
;setup short_addr_1
ldi r16,$e1 ; load first data byte - write short_addr_1 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,high(src_addr) ; load second data byte - src_addr msb
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup pan_id_0
ldi r16,$e2 ; load first data byte - write pan_id_0 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,low(pan_id) ; load second data byte - pan_id lsb
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup pan_id_1
ldi r16,$e3 ; load first data byte - write pan_id_1 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,high(pan_id) ; load second data byte - pan_id msb
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup phy_cc_ca - set channel id
ldi r16,$c8 ; load byte - write phy_cc_ca register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,channel ; load data byte -
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup phy_tx_pwr - turn crc on
ldi r16,$c5 ; load byte - write phy_tx_pwr register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,$80 ; load data byte - auto_crc=on,pwr=+3dbm
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup xah_ctrl - set frame and csma retries
ldi r16,$ec ; load byte - write xah_ctrl register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,retries ; load data byte - frame and csma retries
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup csma_seed_1
ldi r16,$ee ; load byte - write csma_seed_1 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,high(csma_seed) ; load data byte - min_be=0,aack_set=0,i_am_coord=0,csma(10:8)
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup csma_seed_0
ldi r16,$ed ; load first data byte - write csma_seed_1 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,low(csma_seed) ; load data byte - csma(7:0)
```

```
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;enable irqs
ldi r16,$ce ; load first data byte - write register irq_mask command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$08 ; load second data byte - trx_end only
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

;clear pending irqs on rf unit
ldi r16,$8f ; load first data byte - read register irq_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte and clear pending irqs
sbi portb,portb0 ; pull ss high

;clear pending irqs on micro
ldi r16,$ff
out eifr,r16 ; clear any interrupt flags that are set

;enable interrupts on micro
sei ; turn on interrupts

rcall rxonaack ; put the rf unit in rx_aack_on state

repeat: ; handle backlogged data

cpse zh,yh ; check if buffer has data
rjmp handle ; go to data handler
cpse zl,yl ; check if buffer has data
rjmp handle ; go to data handler
in r17,pinb ; get button press data
andi r17,$60 ; mask off button bits
ldi r24,$60 ; invert button bits
eor r17,r24 ; invert button bits
mov r24,r19 ; move current motor state to temp register
andi r24,$60 ; mask off button state
cp r24,r17 ; check if button state has changed
breq repeat ; continue checking if no change
andi r19,$9b ; turn off control bit and reset button state in motor state register
ori r19,$08 ; set manual bit in motor state register
or r19,r17 ; set current button state in motor state register
or r14,r17 ; store button presses to tx motor state register
cpi r17,$60 ; check if both buttons are pressed
breq buttonstop
cpi r17,$40 ; check if open button is pressed
breq openbutton
cpi r17,$20 ; check if close button is pressed
breq closebutton
andi r19,$97 ; else shut off manual bit and button bits in motor state register

buttonstop: ; turn off motor via buttonpress

rcall motordelay
cbi portb,portb7 ; stop motor
cbi portg,portg0 ; turn on brake
rjmp repeat ; return to state checking

openbutton: ; open motor via buttonpress

cpi r19,$4b ; check if already jammed open
breq repeat ; do not open if jammed
rcall motordelay
cbi portg,portg0 ; turn brake off
sbi portg,portg1 ; set direction to open
sbi portb,portb7 ; turn on motor
andi r19,$fd ; clear jam bit
ori r19,$01 ; set direction open
rjmp repeat ; return to state checking

closebutton: ; close motor via buttonpress

cpi r19,$2a ; check if already jammed closed
breq repeat ; do not open if jammed
rcall motordelay
cbi portg,portg0 ; turn brake off
cbi portg,portg1 ; set direction to closed
sbi portb,portb7 ; turn on motor
andi r19,$fc ; clear jam bit and set direction closed
rjmp repeat ; return to state checking

handle: ; respond to recieved data packets

ld r17,z+ ; get data frame length from buffer
mov r1,r17 ; move frame length to counter
cpi r17,$08 ; check if packet is the right length
brne resetbuffer ; reset buffer if not
ld r17,z+ ; get header byte from buffer
dec r1
cpi r17,$08 ; check if right packet type
brne resetbuffer
ld r17,z+ ; lsb src_addr - ignore
dec r1
ld r17,z+ ; msb src_addr - ignore
dec r1
ld r17,z+ ; get command byte
dec r1
cpi r17,$08 ; check if motor control command
breq motor ; control motor
rjmp resetbuffer ; else finish off

motor: ; control motor

ld r17,z+ ; get motor command
dec r1
sbrc r19,3 ; do not modify if under manual control
rjmp resetbuffer ; commands past this point are overriden by manual control
cpi r17,$08 ; check if window full open
breq open
cpi r17,$06 ; check if window full close
breq close
cpi r17,$0a ; check if window stop
breq stop
cpi r17,$0c ; check if window set by speed
brne resetbuffer ; else end
ld r17,z+ ; get control setpoint
dec r1
cpi r17,(hysteresis + $01) ; check if setpoint is too low
brlo setpointclose ; completely shut
cpi r17,($ff - hysteresis) ; check if setpoint is too high
brsh setpointopen ; completely open
```

```
subi r17,hysteresis ; put hysterisis in
mov r6,r17 ; move setpoint to control bottom
subi r17,($00 - 2*(hysteresis)) ; put hysterisis in
mov r7,r17 ; move setpoint to control top
ori r19,$04 ; turn on control bit in motor state register
rjmp resetbuffer ; finish off

resetbuffer: ; reset buffer pointer

ldi r17,$00
add zl,r1 ; add remaining packet count
adc zh,r17 ; increment zh if zl overflow
sbrc zh,2 ; check if buffer at 1k
ldi zh,$02 ; reset buffer pointer to bottom of buffer
rjmp repeat ; get next packet from buffer

open: ; open window command

andi r19,$fb ; turn off control bit
cpi r19,$03 ; check if already jammed open
breq resetbuffer ; do not open if jammed
ld r17,z+ ; get step size packet
dec r1
cpi r17,$ff ; check if full open command
breq openfull
cpi r17,(delay + $01) ; check if step size is bigger than motordelay
brlo resetbuffer ; end if not big enough
rcall motortime ; move packet to motor timer

openfull: ; turn on motor full open

rcall motordelay
cbi portg,portg0 ; turn brake off
sbi portg,portg1 ; set direction to open
sbi portb,portb7 ; turn on motor
andi r19,$fd ; clear jam bit
ori r19,$01 ; set direction open
rjmp resetbuffer

stop: ; stop motor

andi r19,$fb ; turn off control bit
rcall motordelay
cbi portb,portb7 ; stop motor
cbi portg,portg0 ; turn on brake
rjmp resetbuffer

close: ; close window command

andi r19,$fb ; turn off control bit
cpi r19,$02 ; check if already jammed closed
breq resetbuffer ; do not close if jammed
ld r17,z+ ; get step size packet
dec r1
cpi r17,$ff ; check if full open command
breq closefull
cpi r17,(delay + $01) ; check if step size is bigger than motordelay
brlo resetbuffer ; end if not big enough
rcall motortime ; move packet to motor timer

closefull: ; turn on motor full closed

rcall motordelay
cbi portg,portg0 ; turn brake off
cbi portg,portg1 ; set direction to closed
sbi portb,portb7 ; turn on motor
andi r19,$fc ; clear jam bit and set direction closed
rjmp resetbuffer

setpointclose: ; go to full closed because setpoint is too low for control

andi r19,$fb ; turn off control bit if already on
cpi r19,$02 ; check if already jammed closed
breq resetbuffer ; do not close if jammed
ldi r17,$08 ; turn off timer
sts tccr5b,r17 ; turn off timer
ldi r17,$00
sts timsk5,r17 ; disable t5 interrupts
rjmp closefull ; close

setpointopen: ; go to full open because setpoint is too high for control

andi r19,$fb ; turn off control bit if already on
cpi r19,$03 ; check if already jammed open
breq resetbuffer ; do not open if jammed
ldi r17,$08 ; turn off timer
sts tccr5b,r17 ; turn off timer
ldi r17,$00
sts timsk5,r17 ; disable t5 interrupts
rjmp openfull ; close

motordelay: ; delay the current limit on motor at start

ldi r17,$00
sts tcnt4h,r17 ; set counter to zero
sts tcnt4l,r17 ; set counter to zero
ldi r17,delay
sts ocr4ah,r17 ; set counter top to 100ms
ldi r17,$00
sts ocr4al,r17 ; set counter top to 100ms
ldi r17,$2f
out tifr4,r17 ; clear all t4 interrupt flags
ldi r17,$02
sts timsk4,r17 ; turn on t4 interrupt
ldi r17,$0d
sts tccr4b,r17 ; turn on counter, set to 128us period
cbi eimsk,7 ; turn off currentlimit interrupt
ret

motortime: ; have motor on for a certain amount of time
mov r22,r17 ; move motortime to counter
ldi r17,(motor_time)
sts ocr5ah,r17 ; set counter top to packet time
ldi r17,$00
sts ocr5al,r17 ; set counter top to packet time
sts tcnt5h,r17 ; set counter to zero
sts tcnt5l,r17 ; set counter to zero
ldi r17,$2f
out tifr5,r17 ; clear all t5 interrupt flags
ldi r17,$02
sts timsk5,r17 ; turn on t5 interrupt
ldi r17,$0d
```

```
sts tccr5b,r17 ; turn on counter, set to 128us period
ret

currentstop: ; brake motor if current is too high

in r16,sreg ; get sreg
push r16 ; push sreg on stack
cbi portb,portb7 ; stop motor
cbi portg,portg0 ; turn on brake
ldi r16,$08 ; turn off timer
sts tccr5b,r16 ; turn off timer if still on
ldi r16,$00
sts timsk5,r16 ; disable t5 interrupts
ori r19,$02 ; set jam bit
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; return from interrupt

timer4int: ; reset timer and do whatever task is required

ldi r16,$08 ; turn off timer
sts tccr4b,r16 ; turn off timer
ldi r16,$00
sts timsk4,r16 ; disable t4 interrupts
sbi eifr,7 ; clear currentlimit interrupt
sbi eimsk,7 ; turn currentlimit interrupt back on
sbic pine,7 ; check if current at limit
rjmp currentstop
reti ; return from interrupt

timer5int: ; turn off motor

push r17
in r16,sreg ; get sreg
push r16 ; push sreg on stack
dec r22 ; decrement 3rd byte counter
brne timer5intdone ; dont do anything if not done
rcall motordelay
cbi portb,portb7 ; stop motor
cbi portg,portg0 ; turn on brake
ldi r16,$08 ; turn off timer
sts tccr5b,r16 ; turn off timer
ldi r16,$00
sts timsk5,r16 ; disable t5 interrupts

timer5intdone: ; finish off

pop r16 ; get off stack
out sreg,r16 ; return sreg
pop r17
reti ; return from interrupt

txoff: ; send trx_off command to rf unit and wait till stable

ldi r16,$c2 ; load first data byte - write register trx_state command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$08 ; load second data byte - data trx_off bit
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

checktxoff: ; check current state to see if in tx_off state

ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$08
brne checktxoff
ret ; return to previous duty

pllon: ; send pll_on command to rf unit

ldi r16,$c2 ; load first data byte - write register trx_state command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$09 ; load second data byte - data pll_on bit
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

checkpllon: ; check current state to see if in pll_on state

ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$09
brne checkpllon
ret ; return to previous duty

rxonaack: ; send rx_aack_on command to rf unit and wait till stable

ldi r16,$c2 ; load first data byte - write register trx_state command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$16 ; load second data byte - data rx_aack_on bit
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

checkrxonaack: ; check current state to see if in rx_aack_on state

ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$16
brne checkrxonaack
ret ; return to previous duty

txonaret: ; send trx_aret_on command to rf unit and wait till stable

ldi r16,$c2 ; load first data byte - write register trx_state command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$19 ; load second data byte - data trx_aret_on bit
rcall spiwrite ; send byte
```

```
sbi portb,portb0 ; pull ss high

checktxonaret: ; check current state to see if in tx_aret_on state

ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$19
brne checktxonaret
ret ; return to previous duty

joinrecieve: ; get new des_addr

cpi r18,$04 ; check if waiting for joinack
brne rxrequest ; dont accpet joinack if not waiting for joinack
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_panid(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_panid(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_addr(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_addr(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - src_addr(7:0)
mov r11,r16
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - src_addr(15:8)
mov r10,r16
;dont bother with the rest of the data - it doesnt matter
;ldi r16,$00 ; load data byte - blank for reading
;rcall spiwrite ; send byte - do not save fcs(7:0)
;ldi r16,$00 ; load data byte - blank for reading
;rcall spiwrite ; send byte - do not save fcs(15:8)
;ldi r16,$00 ; load data byte - blank for reading
;rcall spiwrite ; send byte - lqi
sbi portb,portb0 ; pull ss high
sbiw yh:yl,$02 ; reset data buffer to begining of join request
ldi r18,$10 ; set state register to retry transmit
rcall transmit ; set rf unit to tx_on_aret state
rcall transmitpacket ; transmit data
ret ; done

rxreadinterrupt: ; empty rx buffer while in interrupt

;ed_level is taken first because it is only valid for 224us after interrupt fires
ldi r16,$87 ; load data byte - read ed_level
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load data byte - blank byte for reading
rcall spiwrite ; send byte
mov r13,r16 ; move ed_level to temporary buffer
sbi portb,portb0 ; pull ss high

;get data from recieve buffer
ldi r16,$20 ; load data byte - read buffer command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
```

```
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - frame length byte
subi r16,$06 ; decrease frame length for bits to be removed
st y+,r16 ; move frame length to buffer
subi r16,$03 ; decrease frame length for buffer fitting
mov r0,r16 ; move frame length to temporary register
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcf(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcf(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - seq (used as command byte)
st y+,r16
cpi r16,$06 ; check if command byte is $06 (join acknowledge)
breq joinrecieve ; get new home_addr if $06
cpi r16,$08 ; check if data to tethered node
breq notrequest1 ; load data to buffer, else discard packet

rxrequest: ; finish off

sbi portb,portb0 ; pull ss high
sbiw yh:yl,$02 ; reset data buffer to begining of join request
ret ; done

notrequest1: ; continue with data loading

ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_panid(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_panid(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_addr(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_addr(15:8)

getdatarf1: ; keep loading data till frame compelete

ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte
st y+,r16 ; move byte to buffer
dec r0 ; decrement frame length byte
brne getdatarf1 ; keep decrementing until done
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcs(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcs(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - lqi
sbi portb,portb0 ; pull ss high
st y+,r16 ; move lqi to buffer
st y+,r13 ; move ed_level to buffer
sbrc yh,2 ; check if buffer at 1k
ldi yh,$02 ; reset buffer pointer to bottom of buffer
ret ; done writing data

transmit: ; set rf unit to tx_on_aret state

;check to see if not in rx_busy state
ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
```

```
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$11 ; check if in busy_rx
breq transmit ; keep checking till in a new state
cpi r16,$1f ; check if in state transition
breq transmit ; keep checking till in a new state
cpi r16,$12 ; check if in busy_tx state
breq transmit ; keep checking till in a new state
rcall pllon ; go to pll on state
;check if there is data pending in rx buffer
ldi r16,$8f ; load first data byte - read register irq_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
sbrc r16,3 ; check if trx_end interrupt flag is set
rcall rxreadinterrupt ; empty rx buffer
rcall txonaret ; put the rf unit in tx_aret_on state
ret ; return to previous duty

waitloop1: ; wait timer (~3084 cycles per value in r21)

ser r20
rcall wait2
nop
nop
nop
nop
nop
nop
nop
nop
dec r21
brne waitloop1
ret ; return to previous duty

spiwrite: ; read and write data over spi

out spdr,r16 ; write transmitted byte to spi

wait: ; read spi register to check when done

in r16,spsr
sbrs r16,spif
rjmp wait
in r16,spdr ; write recieved byte to spi register
ret ; return to previous duty

wait2: ; wait timer (~12 cylces per value in r20)

nop
nop
nop
nop
nop
nop
nop
nop
dec r20
brne wait2
ret ; return to previous duty

gotorx: ; return to rx_on_aack state

;check successful transmit before switching back to rx
ldi r16,$82 ; load data byte - read trx_state to get trac_status
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load data byte - blank byte for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
andi r16,$e0 ; mask off trac_status
cpi r16,$60 ; check if success or data_pending success
brlo gotorx1 ; skip if success
cpi r18,$01 ; check if this is the first failure
brne gotorx3 ; if not - finish off
rcall transmitrequest ; get new des_addr and retransmit
ldi r18,$02 ; set joinrequest state bit
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; clear interrupt and return to previous duty

gotorx1: ; turn on rx_on_aack state

rcall pllon ; go to pll_on state
rcall rxonaack ; go to rx_on_aack state
cpi r18,$02 ; check if last state was a joinrequest
brne gotorx2 ; finish off if not waiting for joinrequest ack
ldi r18,$04 ; set rf state to joinrequest ack waiting
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; return to previous task

gotorx3: ; set to rx_an_aack state

rcall pllon ; go to pll_on state
rcall rxonaack ; go to rx_on_aack state

gotorx2: ; finish off

clr r14 ; reset tx motor state registser
ldi r18,$00 ; reset rf state to complete
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; return to previous task

clearirq: ; clear irq register and irq line

in r16,sreg ; get sreg
push r16 ; push sreg on stack
ldi r16,$8f ; load first data byte - read register irq_status command to clear interrupt
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

checkrf: ; check rf unit state

ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
```

```
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$16 ; check if in rx_on
breq datacollect ; get data
cpi r16,$11 ; check if in busy_rx
breq datacollect ; get data
cpi r16,$1f ; check if in state transition
breq checkrf ; keep checking till in a new state
cpi r16,$12 ; check if in busy_tx state
breq checkrf ; keep checking till in a new state
rjmp gotorx ; else change state back to rx

;removed checkcrc because crc is automatically checked with rx_aack

datacollect: ; get data from rf unit

rcall rxreadinterrupt ; get data from rf unit
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; done writing data

wakeup: ; check if time to take data

in r16,sreg ; get sreg
push r16 ; push sreg on stack
dec r25 ; decrement wakeup counter
breq wakeup1 ; take data if 64s
pop r16 ; get sreg off stack
out sreg,r16 ; return sreg
reti

wakeup1: ; take data and return to previous task

push r17 ; push r17 on stack
lds r8,tcnt1l ; move windspeed lsb to temporary register
lds r9,tcnt1h ; move windspeed msb to temporary register
clr r16 ; reset activity level counter
sts tcnt1h,r16
sts tcnt1l,r16
;motor control code
sbrs r19,2 ; check if in control mode
rjmp wakeupend ; finish off if not
cp r8,r6 ; check if speed too low
brlo incrementup ; move motor open
cp r8,r7 ; check if speed to high
brsh incrementdown ; move motor closed

wakeupend: ; else finish off

or r14,r19 ; move motor state register to tx motor state register
rcall shit15temp ; initiate shit15 read - humidity included
ldi r16,$df ; take measurement from adc0
sts adcsra,r16 ; turn on adc,clear interrupt flag,enable interrupt,set to /128 (62.5kHz)
ldi r25,$08 ; reset wakeup counter to 64s
pop r17 ; get r17 off stack
pop r16 ; get sreg off stack
out sreg,r16 ; return sreg
reti

incrementup: ; open with motor a few steps

cpi r19,$07 ; check if already jammed open
breq wakeupend ; do nothing if jammed open
ldi r17,stepsize ; set increment length
rcall motortime ; move packet to motor timer
rcall motordelay ; turn on current limit timer
cbi portg,portg0 ; turn brake off
sbi portg,portg1 ; set direction to open
sbi portb,portb7 ; turn on motor
andi r19,$fd ; clear jam bit
ori r19,$01 ; set direction open
rjmp wakeupend ; finish off

incrementdown: ; close with motor a few steps

cpi r19,$06 ; check if already jammed closed
breq wakeupend ; do nothing if jammed closed
ldi r17,stepsize ; set increment length
rcall motortime ; move packet to motor timer
rcall motordelay ; turn on current limit timer
cbi portg,portg0 ; turn brake off
cbi portg,portg1 ; set direction to closed
sbi portb,portb7 ; turn on motor
andi r19,$fc ; clear jam bit and set direction closed
rjmp wakeupend ; finish off

wakeupadc: ; wakeup and return to previous task

in r16,sreg ; get sreg
push r16 ;  push sreg on stack
lds r5,adcl ; load lsb to r5
lds r2,adch ; load msb to r2
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti

shit15setup: ; set the device to low res mode
;talk to shit15, only works for cpu clock <=2MHZ
;code modded with nops to operate at 8MHz
;data pin can never be output high - leave pullup off
;needs external pullup resistor for proper operation
;sleep operation disabled for this application
;always read temperature before humidity

rcall shit15reset ; reset the device in case its fuxxored
rcall startshit15 ; send start sequence

;data sequence - register write command $06
;nops added to keep clock at 1MHz
sbi porte,porte0 ; clock high - data bit one
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit two
nop
nop
cbi porte,porte0 ; clock low
nop
nop
```

```
sbi porte,porte0 ; clock high - data bit three
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit four
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit five
nop
nop
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; data high
nop ; data takes time to rise and needs setup time
nop
nop
nop
nop
nop
sbi porte,porte0 ; clock high - data bit six
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit seven
nop
nop
cbi porte,porte0 ; clock low
sbi ddre,dde4 ; data low
sbi porte,porte0 ; clock high - data bit eight
nop
nop
cbi porte,porte0 ; clock low

rcall shit15ack0 ; wait for ack from shit15

;data sequence - register write data $01
;nops added to keep clock at 1MHz
sbi ddre,dde4 ; data low
sbi porte,porte0 ; clock high - data bit one
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit two
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit three
nop
nop
cbi porte,porte0 ; clock low
nop
```

```
nop
sbi porte,porte0 ; clock high - data bit four
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit five
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit six
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit seven
nop
nop
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; data high
nop ; data takes time to rise and needs setup time
nop
nop
nop
nop
nop
sbi porte,porte0 ; clock high - data bit eight
nop
nop
cbi porte,porte0 ; clock low

rcall shit15ack0 ; wait for ack from shit15
ret ;done setting up the device

shit15temp: ; read temperature data from shit15
;send read temp commmand
rcall startshit15 ; send start sequence

;data sequence - read temperature command $03
sbi porte,porte0 ; clock high - data bit one
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit two
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit three
nop
nop
cbi porte,porte0 ; clock low
nop
nop
```

```
sbi porte,porte0 ; clock high - data bit four
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit five
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit six
nop
nop
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; data high
nop ; data takes time to rise and needs setup time
nop
nop
nop
nop
nop
sbi porte,porte0 ; clock high - data bit seven
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit eight
nop
nop
cbi porte,porte0 ; clock low
;sbi ddre,dde4 ; data low - if last databyte is 1 - leave it high for ack

rcall shit15ack0 ; wait for ack from shit15

sbi eimsk,4 ; enable int4
ret ; return to previous task and wait for shit15 to be done

wakeupshit15: ; collect data from shit15
in r16,sreg ; get sreg
push r16 ; push sreg on stack
brts shit15humread ; skip if its a humidity read
cbi eimsk,4 ; disable int4 as pe4 will still be low when returning
rcall shit15read ; read out temperature data from shit15 (msb)
rcall shit15ack3 ; acknowledge reciept of bit
mov r3,r16 ; move msb to another register
rcall shit15read ; read out temperature data from shit15 (lsb)
rcall shit15ack4 ; end transmission ack
mov r4,r16 ;  move lsb to another register
rcall shit15humidity
pop r16 ; get off stack
out sreg,r16 ; return sreg
set ; set tregister to indicate humidity request in progress
reti ; continue with previous task

shit15humread: ; read out humidity data from shit 15
rcall shit15read ; read out humidity data from shit15 (msb)
rcall shit15ack3 ; acknowledge reciept of bit
rcall shit15read ; read out humidity data from shit15 (lsb)
```

```
rcall shit15ack4 ; end transmission ack
mov r12,r16 ; move humidity data to temporary register
ldi r18,$01 ; set rf state to first transmit
rcall transmit ; set rf unit to tx state
rcall transmitpacket ; send data off
pop r16 ; get off stack
out sreg,r16 ; return sreg
clt ; reset to temperature read
reti ; return with previous task

shit15humidity: ; read humidity data from shit15
;send read humidity command
rcall startshit15 ; send start sequence

;data sequence - read humidity command $05
;nops added to keep clock at 1MHz
sbi porte,porte0 ; clock high - data bit one
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit two
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit three
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit four
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit five
nop
nop
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; data high
nop ; data takes time to rise and needs setup time
nop
nop
nop
nop
nop
sbi porte,porte0 ; clock high - data bit six
nop
nop
cbi porte,porte0 ; clock low
sbi ddre,dde4 ; data low
sbi porte,porte0 ; clock high - data bit seven
nop
nop
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; data high
```

```
nop ; data takes time to rise and needs setup time
nop
nop
nop
nop
nop
sbi porte,porte0 ; clock high - data bit eight
nop
nop
cbi porte,porte0 ; clock low
;sbi ddre,dde4 ; data low - if last databyte is 1 - leave it high for ack

rcall shit15ack0 ; wait for acknowledge from shit15
sbi eimsk,4 ; enable int4
ret ; return to previous task and wait for shit15 to take data

shit15ack4: ; end transmission ack
;nops added to keep clock at 1MHz
sbi porte,porte0 ; clock high - ack
nop
nop
cbi porte,porte0 ; clock low
ret ; return to previous task

shit15ack3: ; data recipet acknowledge
;nops added to keep clock at 1MHz
sbi ddre,dde4 ; pull data low
sbi porte,porte0 ; clock high - ack
nop
nop
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; release data line
ret ; return to previous task

shit15read: ; read data from shit15
;nops added to keep clock at 1MHz and meet data valid time of 250ns
clr r16 ; clear the register where incoming data will be written
sbi porte,porte0 ; clock high - data bit one
sbic pine,pine4 ; check if data is low
sbr r16,$80 ; write data bit one to register
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit two
sbic pine,pine4 ; check if data is low
sbr r16,$40 ; write data bit two to register
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit three
sbic pine,pine4 ; check if data is low
sbr r16,$20 ; write data bit three to register
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit four
sbic pine,pine4 ; check if data is low
sbr r16,$10 ; write data bit four to register
cbi porte,porte0 ; clock low
nop
nop
```

```
sbi porte,porte0 ; clock high - data bit five
sbic pine,pine4 ; check if data is low
sbr r16,$08 ; write data bit five to register
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit six
sbic pine,pine4 ; check if data is low
sbr r16,$04 ; write data bit six to register
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit seven
sbic pine,pine4 ; check if data is low
sbr r16,$02 ; write data bit seven to register
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit eight
sbic pine,pine4 ; check if data is low
sbr r16,$01 ; write data bit eight to register
cbi porte,porte0 ; clock low
ret ; return to previous activity

startshit15: ; start sequence
;nops added to keep clock at 1MHz
cbi ddre,dde4 ; set data high
cbi porte,porte0 ; make sure clock is low
nop
nop
sbi porte,porte0 ; pull clock high
sbi ddre,dde4 ; set data low
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
cbi ddre,dde4 ; data high
nop ; data takes time to rise and needs setup time
nop
nop
nop
nop
nop
cbi porte,porte0 ; clock low
sbi ddre,dde4 ; data low
ret ; return to previous task

shit15ack0: ; ack sequence for sending data

cbi ddre,dde4 ; release data line

shit15ack2: ; check that data line is low
;nops added to keep clock at 1MHz
sbic pine,pine4
rjmp shit15ack2
sbi porte,porte0 ; clock high - ack
nop
nop
cbi porte,porte0 ; clock low

shit15ack1: ; check that line has been released
```

```
sbis pine,pine4 ; check that line has been released
rjmp shit15ack1
ret ; return to previous task

shit15reset: ; reset sequence if it gets out of phase
;status register preserved
;must be followed by data start sequence
;nops added to slow it down to 1MHz
cbi ddre,dde4 ; data high
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
ret ; return to previous task

transmitrequest: ; transmit a request frame

sbi portb,portb4 ; pull slp_tr high to begin transmit
nop ; delay to make sure the pin is high long enough
cbi portb,portb4 ; pull slp_tr low

ldi r16,$60 ; load data byte - write buffer command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$0b ; load data byte - frame length byte
rcall spiwrite ; send byte
ldi r16,$40 ; load data byte - fcf(7:0) - broadcast,no ack
rcall spiwrite ; send byte
ldi r16,$88 ; load data byte - fcf(15:8) - short address,pan_ids equal
rcall spiwrite ; send byte
ldi r16,$01 ; load data byte - seq(7:0) - command(request)
rcall spiwrite ; send byte
ldi r16,low(pan_id) ; load data byte - pan_id(7:0)
rcall spiwrite ; send byte
ldi r16,high(pan_id) ; load data byte - pan_id(15:8)
rcall spiwrite ; send byte
ldi r16,$ff ; load data byte - destination address (7:0) - broadcast
rcall spiwrite ; send byte
ldi r16,$ff ; load data byte - destination address (15:0) - broadcast
rcall spiwrite ; send byte
ldi r16,low(src_addr) ; load data byte - source address (7:0)
rcall spiwrite ; send byte
ldi r16,high(src_addr) ; load data byte - source address (15:8)
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
ret ; return to previous duty

transmitpacket: ; transmit a packet

sbi portb,portb4 ; pull slp_tr high to begin transmit
nop ; delay to make sure the pin is high long enough
cbi portb,portb4 ; pull slp_tr low

ldi r16,$60 ; load data byte - write buffer command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$13 ; load frame length byte - total bytes excluding this one, +2
rcall spiwrite ; send byte
ldi r16,$61 ; load fcf(7:0) byte - $61 for data packet
rcall spiwrite ; send byte
ldi r16,$88 ; load fcf(15:8) byte - $88 for data packet
rcall spiwrite ; send byte
ldi r16,$07 ; load sequence byte - command (data from tethered node)
rcall spiwrite ; send byte
ldi r16,low(pan_id) ; load destination panid(7:0) byte
rcall spiwrite ; send byte
ldi r16,high(pan_id) ; load destination panid(15:8) byte
```

```
rcall spiwrite ; send byte
mov r16,r11 ; load destination address(7:0) byte
rcall spiwrite ; send byte
mov r16,r10 ; load destination address(15:8) byte
rcall spiwrite ; send byte
ldi r16,low(src_addr) ; load source address(7:0) byte
rcall spiwrite ; send byte
ldi r16,high(src_addr) ; load source address(15:8) byte
rcall spiwrite ; send byte
mov r16,r14 ; load data byte - motor state register
rcall spiwrite ; send byte
mov r16,r12 ; load data byte - humidity
rcall spiwrite ; send byte
mov r16,r4 ; load data byte - temperature0
rcall spiwrite ; send byte
mov r16,r3 ; load data byte - temperature1
rcall spiwrite ; send byte
mov r16,r8 ; load data byte - windspeed lsb
rcall spiwrite ; send byte
mov r16,r9 ; load data byte - windspeed msb
rcall spiwrite ; send byte
mov r16,r5 ; load data byte - light lsb
rcall spiwrite ; send byte
mov r16,r2 ; load data byte - light msb
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
ret ; return to previous duty
```

Appendix I

Room Node Firmware

```
.include "m1281def.inc"

; increment of 4 for src_addr
; csma_seed = 8 x src_addr
.equ src_addr = $aac4 ; place unique source address here
.equ csma_seed = low(src_addr)*$08 ; random backoff exponent - must be unique,<$0800
.equ home_addr = $aabb ; local node des_addr
.equ pan_id = $abcd ; system pan_id
.equ retries = $38 ; high nibble = frame retries,low nibble = 2x(csma retries)
.equ channel = $36 ; tx/rx channel, $2b -> $3a valid

.org 0
rjmp start
.org int4addr
rjmp wakeupshit15
.org int5addr
rjmp clearirq
.org OVF2addr
rjmp wakeup
.org ADCCaddr
rjmp wakeupadc
.org URXC1addr ; USART1 - Rx Complete
rjmp wakeuprx
.org OC4Aaddr
rjmp timer4int

; lightsensor = pf0(adc0)
; shit15 data = pe4(int4)
; shit15 clock = pe0
; activity sensor = pd7(t0)
; xport = pd2,pd3(usart1),
; r0  frame length temporary register - write
; r1  frame length temporary register - read
; r2  adc msb temporary register
; r3  shit15 msb temporary register
; r4  shit15 lsb temporary register
; r5  adc lsb temporary register
; r6  joinrequest buffer index lsb
; r7  joinrequest buffer index msb
; r8  activity level temporary register
; r9  ed_level temporary register
; r10 usart temporary register
; r11 last rf tx packet length temporary register
; r12 shit15 humidity temporary register
; r13
; r14
; r15 usart tx temporary register
; r16 temporary swap register for interrupts
; r17 temporary swap register for interrupts
; r18 temporary swap register for main
; r19 temporary swap register for main
; r20 wait loop register
; r21 wait loop register
; r22 hextoascii temporary register
; r23 usart rx temporary register
; r24 wakeup counter
; r25 transmit temporary register
; xregister is recieve buffer (r27(xh),r26(xl))
; yregister is buffer start (r29(yh),r28(yl))
; zregister is buffer end (r31(zh),r30(zl))
```

```
; data buffer set at 1k
; tregister is shit15 humidity/temperature irq differentiator
; joinrequest buffer is 256byte
; recieve buffer is 256byte

start: ; configure microcontroller registers
; using external oscillator (from RF unit)
; start up time 6CK + 65ms
; set to 1MHz initially, changed to 8MHz by clkm setup
ldi r16,high(RAMEND)
out SPH,r16 ; Set Stack Pointer to top of RAM
ldi r16,low(RAMEND)
out SPL,r16
ldi r16,$17
out ddrb,r16 ; set ss,sclk,mosi,slp_tr as output
ldi r16,8
out portb,r16 ; enable pullup for miso
ldi r16,1
out spsr,r16 ; set up spi - mode0, ck/2
ldi r16,$50
out spcr,r16 ; set up spi - mode0, ck/2 (4MHz)
sbi portb,portb0 ; set ss pin high
cbi portb,portb4 ; set slp_tr low
sbi porta,porta7 ; pull reset high on rf unit
sbi ddra,dda7 ; set reset pin as output
ldi r16,$0c
sts eicrb,r16 ; configure int5 for rising edge
ldi r16,$20
out eimsk,r16 ; enable int5
clr zl ; set buffer end to beginning of sram
clr yl ; set buffer start to beginning of sram
ldi zh,$02 ; set buffer end to beginning of sram
ldi yh,$02 ; set buffer start to beginning of sram
ldi xh,$05 ; set recieve buffer to middle of sram
clr xl ; set recieve buffer to middle of sram
ldi r23,$ff ; initialize receieve buffer count
ldi r24,$04 ; set wakeup counter to 32s

;wait for rf unit to stabilize
ldi r21,$04 ; load the wait timer to 3 cycles (x3084/ck)
rcall waitloop1 ; wait ~(9252/8MHz = 1.2ms) for rf unit to stabilize

;setup clkm
ldi r16,$c3 ; load first data byte - write trx_ctrl_0 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,$04 ; load second data byte - lowest driver strength, clkm = 8MHz
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high
nop ; idle for a bit as clock changes
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
```

```
nop
nop
nop
ldi r21,$ff ; load wait timer to 255 cycles (x3084/ck)
rcall waitloop1 ; wait ~100ms to make sure clock is stable

;setup analog inputs
ldi r16,$01
sts didr0,r16 ; turn off input stage for adc0 pin
ldi r16,$c0
sts admux,r16 ; use internal 2.56v as voltage reference,sample adc0

;setup t0 as activity counter
ldi r16,$06
out tccr0b,r16 ; set t0 to external clock on t0 falling edge

;setup t2 as rtc at 8s interval wakeup
ldi r16,$20
sts assr,r16 ; set t2 to assynchronous mode
ldi r16,$07
sts tccr2b,r16 ; set t2 prescaler to /1024 - 8s wakeup period

;make sure t2 is done rewriting itself
checkassr:

lds r16,assr
andi r16,$1f ; check assr(4:0) clear
brne checkassr

clr r16
out tifr2,r16 ; clear all pending interrupts
ldi r16,$01
sts timsk2,r16 ; enable t2 overflow interrupt

;setup t4 as usart recieve counter
ldi r16,$00
sts ocr4ah,r16 ; set counter top to 2ms
ldi r16,$10
sts ocr4al,r16 ; set counter top to 2ms

;setup shit15 - pe0 is data, pe4 is clock
sbi ddre,dde0 ; set clock as output
cbi porte,porte4 ; make sure pullups are off for data
rcall shit15setup

;setup usart1
ldi r16,$98
sts ucsr1b,r16 ; enable rx,tx,rx interrupt
ldi r16,$0c
sts ubrr1l,r16 ; set baud rate to 38.4k

;configure rf unit registers

;go to trx_off state
rcall txoff ; go to trx_off state on rf unit

;setup short_addr_0
ldi r16,$e0 ; load first data byte - write short_addr_0 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,low(src_addr) ; load second data byte - src_addr lsb
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup short_addr_1
ldi r16,$e1 ; load first data byte - write short_addr_1 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,high(src_addr) ; load second data byte - src_addr msb
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup pan_id_0
ldi r16,$e2 ; load first data byte - write pan_id_0 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,low(pan_id) ; load second data byte - pan_id lsb
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup pan_id_1
ldi r16,$e3 ; load first data byte - write pan_id_1 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,high(pan_id) ; load second data byte - pan_id msb
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup phy_cc_ca - set channel id
ldi r16,$c8 ; load byte - write phy_cc_ca register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,channel ; load data byte -
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup phy_tx_pwr - turn crc on
ldi r16,$c5 ; load byte - write phy_tx_pwr register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,$80 ; load data byte - auto_crc=on,pwr=+3dbm
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup xah_ctrl - set frame and csma retries
ldi r16,$ec ; load byte - write xah_ctrl register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,retries ; load data byte - frame and csma retries
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup csma_seed_1
ldi r16,$ee ; load byte - write csma_seed_1 register
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,high(csma_seed) ; load data byte - min_be=0,aack_set=0,i_am_coord=0,csma(10:8)
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;setup csma_seed_0
ldi r16,$ed ; load first data byte - write csma_seed_1 register
```

```
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send data
ldi r16,low(csma_seed) ; load data byte - csma(7:0)
rcall spiwrite ; send data
sbi portb,portb0 ; pull ss high

;enable irqs
ldi r16,$ce ; load first data byte - write register irq_mask command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$08 ; load second data byte - trx_end only
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

;clear pending irqs on rf unit
ldi r16,$8f ; load first data byte - read register irq_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte and clear pending irqs
sbi portb,portb0 ; pull ss high

;clear pending irqs on micro
clr r16
out eifr,r16 ; clear any interrupt flags that are set

;enable interrupts on micro
sei ; turn on interrupts

rcall rxonaack ; put the rf unit in rx_aack_on state

repeat: ; handle backlogged data

cpse zh,yh ; check if buffer has data
rjmp handle ; go to data handler
cpse zl,yl ; check if buffer has data
rjmp handle ; go to data handler
rjmp repeat ; keep checking for data

handle: ; deal with data
; needs fixing to deal with buffer overruns

senddatausart: ; send data over usart from buffer

ldi r18,$ff ; move header byte to transmit buffer
rcall txusart
ld r18,z+ ; get data frame length from buffer
mov r1,r18 ; move frame lenghth to temporary register
rcall txusart

senddatausart1: ; send data over usart from buffer

ld r18,z+ ; get data from buffer
rcall txusart
dec r1
brne senddatausart1
ldi r18,low(src_addr)
rcall txusart
ldi r18,high(src_addr)
rcall txusart
ldi r18,$fe
rcall txusart
ldi r18,$fd
rcall txusart
sbrc zh,2 ; check if buffer at 1k
ldi zh,$02 ; reset buffer pointer to bottom of buffer
rjmp repeat ; get next byte from buffer

txoff: ; send trx_off command to rf unit and wait till stable

ldi r16,$c2 ; load first data byte - write register trx_state command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$08 ; load second data byte - data trx_off bit
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

checktxoff: ; check current state to see if in tx_off state

ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$08
brne checktxoff
ret ; return to previous duty

pllon: ; send pll_on command to rf unit

ldi r16,$c2 ; load first data byte - write register trx_state command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$09 ; load second data byte - data pll_on bit
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

checkpllon: ; check current state to see if in pll_on state

ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$09
brne checkpllon
ret ; return to previous duty

rxonaack: ; send rx_aack_on command to rf unit and wait till stable

ldi r16,$c2 ; load first data byte - write register trx_state command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$16 ; load second data byte - data rx_aack_on bit
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

checkrxonaack: ; check current state to see if in rx_aack_on state

ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
```

```
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$16
brne checkrxonaack
ret ; return to previous duty

txonaret: ; send trx_aret_on command to rf unit and wait till stable

ldi r16,$c2 ; load first data byte - write register trx_state command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$19 ; load second data byte - data trx_aret_on bit
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

checktxonaret: ; check current state to see if in tx_aret_on state

ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$19
brne checktxonaret
ret ; return to previous duty

rxreadinterrupt: ; empty rx buffer while in interrupt

;ed_level is taken first because it is only valid for 224us after interrupt fires
ldi r16,$87 ; load data byte - read ed_level
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load data byte - blank byte for reading
rcall spiwrite ; send byte
mov r9,r16 ; move ed_level to temporary buffer
sbi portb,portb0 ; pull ss high

;get data from recieve buffer
ldi r16,$20 ; load data byte - read buffer command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - frame length byte
subi r16,$06 ; decrease frame length for bits to be removed
st y+,r16 ; move frame length to buffer
subi r16,$03 ; decrease frame length for buffer fitting
mov r0,r16 ; move frame length to temporary register
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcf(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcf(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - seq (used as command byte)
st y+,r16
dec r16 ; check if command byte is $01
brne notrequest1 ; skip if not $01

notrequest1: ; continue with data loading
```

```
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_panid(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_panid(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_addr(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_addr(15:8)

getdatarf1: ; keep loading data till frame compelete

ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte
st y+,r16 ; move byte to buffer
dec r0 ; decrement frame length byte
brne getdatarf1 ; keep decrementing until done
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcs(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcs(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - lqi
sbi portb,portb0 ; pull ss high
st y+,r16 ; move lqi to buffer
st y+,r9 ; move ed_level to buffer
sbrc yh,2 ; check if buffer at 1k
ldi yh,$02 ; reset buffer pointer to bottom of buffer
ret ; done writing data

transmit: ; transmit a packet

rcall checkrxonaack ; check to see if not in rx_busy state
rcall pllon ; go to pll on state
;check if there is data pending in rx buffer
ldi r16,$8f ; load first data byte - read register irq_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
sbrc r16,3 ; check if trx_end interrupt flag is set
rcall rxreadinterrupt ; empty rx buffer
rcall txonaret ; put the rf unit in tx_aret_on state

sbi portb,portb4 ; pull slp_tr high to begin transmit
nop ; delay to make sure the pin is high long enough
cbi portb,portb4 ; pull slp_tr low

ldi r16,$60 ; load data byte - write buffer command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ld r16,x+ ; load data byte - frame length byte
mov r25,r16 ; load frame length byte to byte counter
ldi r17,$08 ; increase frame length for crc,panid,fcf,src_addr
add r16,r17 ; increase frame length
rcall spiwrite ; send byte
ldi r16,$61 ; load data byte - fcf(data)
rcall spiwrite ; send byte
ldi r16,$88 ; load data byte - fcf
rcall spiwrite ; send byte
ld r16,x+ ; load data byte - seq(command)
```

```
rcall spiwrite ; send byte
ldi r16,low(pan_id) ; load data byte - des_panid
rcall spiwrite ; send byte
ldi r16,high(pan_id) ; load data byte - des_panid
rcall spiwrite ; send byte
ld r16,x+ ; load data byte - des_addr
rcall spiwrite ; send byte
ld r16,x+ ; load data byte - des_addr
rcall spiwrite ; send byte
ldi r16,low(src_addr) ; load data byte - src_addr
rcall spiwrite ; send byte
ldi r16,high(src_addr) ; load data byte - src_addr
rcall spiwrite ; send byte
subi r25,$03 ; decrease byte counter for fixed transmit bytes
breq loaddone ; finish if no more bytes

dataload: ; load data to rf unit

ld r16,x+ ; load data byte - data byte
rcall spiwrite ; send byte
dec r25 ; check byte in sequence
brne dataload ; branch if not last byte

loaddone: ; finish data transmission and return

sbi portb,portb0 ; pull ss high
ret ; return to previous duty

waitloop1: ; wait timer (~3084 cycles per value in r21)

ser r20
rcall wait2
nop
nop
nop
nop
nop
nop
nop
nop
dec r21
brne waitloop1
ret ; return to previous duty

spiwrite: ; read and write data over spi

out spdr,r16 ; write transmitted byte to spi

wait: ; read spi register to check when done

in r16,spsr
sbrs r16,spif
rjmp wait
in r16,spdr ; write recieved byte to spi register
ret ; return to previous duty

wait2: ; wait timer (~12 cylces per value in r20)

nop
nop
nop
nop
nop
nop
nop
nop
dec r20
brne wait2
ret ; return to previous duty

gotorx: ; return to rx_on_aack state

;check successful transmit before switching back to rx
ldi r16,$82 ; load data byte - read trx_state to get trac_status
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load data byte - blank byte for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
andi r16,$e0 ; mask off trac_status
cpi r16,$60 ; check if success or data_pending success
brlo gotorx1 ; skip if success

;get data from rf unit tx buffer
ldi r16,$20 ; load data byte - read buffer command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte -  old frame length byte - do not save
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte -  new frame length byte
subi r16,$08 ; decrease frame length for bits to be removed
st y+,r16 ; move frame length to buffer
mov r17,r16 ; move frame length to temporary register
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcf(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcf(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - seq (used as command byte)
st y+,r16
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_panid(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_panid(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - save des_addr(7:0)
st y+,r16
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - save des_addr(15:8)
st y+,r16
subi r17,$03 ; decrease frame length for buffer fitting
breq done7 ; finish if last byte
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save src_addr(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save src_addr(15:8)

getdatarf2: ; keep loading data till frame compelete

ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte
```

```
st y+,r16 ; move byte to buffer
dec r17 ; decrement frame length byte
brne getdatarf2 ; keep decrementing until done

done7: ; finish data storage

sbi portb,portb0 ; pull ss high
sbrc yh,2 ; check if buffer at 1k
ldi yh,$02 ; reset buffer pointer to bottom of buffer

gotorx1: ; turn on rx_on_aack state

rcall pllon ; go to pll_on state
rcall rxonaack ; go to rx_on_aack state
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; return to previous task

clearirq: ; clear irq register and irq line

in r16,sreg ; get sreg
push r16 ; push sreg on stack
ldi r16,$8f ; load first data byte - read register irq_status command to clear interrupt
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high

checkrf: ; check rf unit state

ldi r16,$81 ; load first data byte - read register trx_status command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load second data byte - blank for reading
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
cpi r16,$16 ; check if in rx_on
breq datacollect ; get data
cpi r16,$11 ; check if in busy_rx
breq datacollect ; get data
cpi r16,$1f ; check if in state transition
breq checkrf ; keep checking till in a new state
cpi r16,$12 ; check if in busy_tx state
breq checkrf ; keep checking till in a new state
rjmp gotorx ; else put rf unit in rx_on_aack state

;removed checkcrc because crc is automatically checked with rx_aack

datacollect: ; get data from rf unit
;ed_level is taken first because it is only valid for 224us after interrupt fires
ldi r16,$87 ; load data byte - read ed_level
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$00 ; load data byte - blank byte for reading
rcall spiwrite ; send byte
mov r9,r16 ; move ed_level to temprorary buffer
sbi portb,portb0 ; pull ss high

;get data from recieve buffer
ldi r16,$20 ; load data byte - read buffer command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
```

```
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - frame length byte
subi r16,$06 ; decrease frame length for bits to be removed
st y+,r16 ; move frame length to buffer
mov r17,r16 ; move frame length to temporary register for joinrequest check
subi r16,$03 ; decrease frame length for buffer fitting
mov r0,r16 ; load frame length to byte counter
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcf(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcf(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - seq (used as command byte)
st y+,r16
cpi r16,$01 ; check if command byte is $01 (joinrequest)
breq rxrequest ; go to joinrequest if $01

notrequest: ; continue with normal operation

ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_panid(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_panid(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_addr(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_addr(15:8)

getdatarf: ; keep loading data till frame compelete

ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte
st y+,r16 ; move byte to buffer
dec r0 ; decrement frame length byte
brne getdatarf ; keep decrementing until done
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcs(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcs(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - lqi
sbi portb,portb0 ; pull ss high
st y+,r16 ; move lqi to buffer
st y+,r9 ; move ed_level to buffer
sbrc yh,2 ; check if buffer at 1k
ldi yh,$02 ; reset buffer pointer to bottom of buffer
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; done writing data

rxrequest: ; perform joinrequest storage

ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_panid(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_panid(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_addr(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save des_addr(15:8)
```

```
getdatarf3: ; keep loading data till frame compelete

ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte
st y+,r16 ; move byte to buffer
dec r0 ; decrement frame length byte
brne getdatarf3 ; keep decrementing until done
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcs(7:0)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - do not save fcs(15:8)
ldi r16,$00 ; load data byte - blank for reading
rcall spiwrite ; send byte - lqi
sbi portb,portb0 ; pull ss high
st y+,r16 ; move lqi to buffer
st y+,r9 ; move ed_level to buffer
; transmit data out
rcall checkrxonaack ; check to see if not in rx_busy state
rcall pllon ; go to pll on state
rcall txonaret ; put the rf unit in tx_aret_on state

sbi portb,portb4 ; pull slp_tr high to begin transmit
nop ; delay to make sure the pin is high long enough
cbi portb,portb4 ; pull slp_tr low

sbiw yh:yl,$04 ; reset data buffer to begining of join request
ldi r16,$60 ; load data byte - write buffer command
cbi portb,portb0 ; pull ss low
rcall spiwrite ; send byte
ldi r16,$0b ; load data byte - frame length byte
rcall spiwrite ; send byte
ldi r16,$61 ; load data byte - fcf(data)
rcall spiwrite ; send byte
ldi r16,$88 ; load data byte - fcf
rcall spiwrite ; send byte
ldi r16,$06 ; load join confirm command
rcall spiwrite ; send byte
ldi r16,low(pan_id) ; load data byte - des_panid
rcall spiwrite ; send byte
ldi r16,high(pan_id) ; load data byte - des_panid
rcall spiwrite ; send byte
ld r16,y+ ; load data byte - des_addr
rcall spiwrite ; send byte
ld r16,y+ ; load data byte - des_addr
rcall spiwrite ; send byte
ldi r16,low(src_addr) ; load data byte - src_addr
rcall spiwrite ; send byte
ldi r16,high(src_addr) ; load data byte - src_addr
rcall spiwrite ; send byte
sbi portb,portb0 ; pull ss high
adiw yh:yl,$02 ; reset pointer
sbrc yh,2 ; check if buffer at 1k
ldi yh,$02 ; reset buffer pointer to bottom of buffer
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; done writing data

txusart: ; transmit a byte via usart1 for main

lds r10,ucsr1a
sbrs r10,UDRE1
rjmp txusart
sts udr1,r18 ; send out data
ret ; go back to previous task

timer4int: ; reset usart timer due to bad packet

ldi r16,$08 ; turn off timer
sts tccr4b,r16 ; turn off timer
ldi r16,$00
sts timsk4,r16 ; disable t4 interrupts
ldi r23,$ff ; reset usart byte counter
ldi xl,$00 ;  reset buffer to bottom
reti ; return from interrupt

wakeup: ; take data every 32s

in r16,sreg ; get sreg
push r16 ;  push sreg on stack
dec r24 ; decrement wakeup counter
breq t0read ; take data if 32s
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti

t0read: ; get activity level from t0

in r8,tcnt0 ; move activity to temporary register
clr r16 ; reset activity level counter
out tcnt0,r16
rcall shit15temp ; initiate shit15 read - humidity included
ldi r16,$df ; take measurement from adc0
sts adcsra,r16 ; turn on adc,clear interrupt flag,enable interrupt,set to /128 (62.5kHz)
ldi r24,$04 ; reset wakeup counter to 32s
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti

wakeupadc: ; wakeup and return to previous task

in r16,sreg ; get sreg
push r16 ;  push sreg on stack
lds r5,adcl ; load lsb to r5
lds r2,adch ; load msb to r2
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti

shit15setup: ; set the device to low res mode
;talk to shit15, only works for cpu clock <=2MHZ
;code modded with nops to operate at 8MHz
;data pin can never be output high - leave pullup off
;needs external pullup resistor for proper operation
;sleep operation disabled for this application
;always read temperature before humidity

rcall shit15reset ; reset the device in case its fuxxored
rcall startshit15 ; send start sequence

;data sequence - register write command $06
;nops added to keep clock at 1MHz
sbi porte,porte0 ; clock high - data bit one
```

```
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit two
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit three
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit four
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit five
nop
nop
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; data high
nop ; data takes time to rise and needs setup time
nop
nop
nop
nop
nop
sbi porte,porte0 ; clock high - data bit six
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit seven
nop
nop
cbi porte,porte0 ; clock low
sbi ddre,dde4 ; data low
sbi porte,porte0 ; clock high - data bit eight
nop
nop
cbi porte,porte0 ; clock low

rcall shit15ack0 ; wait for ack from shit15

;data sequence - register write data $01
;nops added to keep clock at 1MHz
sbi ddre,dde4 ; data low
sbi porte,porte0 ; clock high - data bit one
nop
nop
cbi porte,porte0 ; clock low
nop
nop
```

```
sbi porte,porte0 ; clock high - data bit two
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit three
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit four
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit five
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit six
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit seven
nop
nop
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; data high
nop ; data takes time to rise and needs setup time
nop
nop
nop
nop
nop
sbi porte,porte0 ; clock high - data bit eight
nop
nop
cbi porte,porte0 ; clock low

rcall shit15ack0 ; wait for ack from shit15
ret ;done setting up the device

shit15temp: ; read temperature data from shit15
;send read temp commmand
rcall startshit15 ; send start sequence

;data sequence - read temperature command $03
sbi porte,porte0 ; clock high - data bit one
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit two
```

```
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit three
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit four
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit five
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit six
nop
nop
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; data high
nop ; data takes time to rise and needs setup time
nop
nop
nop
nop
nop
sbi porte,porte0 ; clock high - data bit seven
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit eight
nop
nop
cbi porte,porte0 ; clock low
;sbi ddre,dde4 ; data low - if last databyte is 1 - leave it high for ack

rcall shit15ack0 ; wait for ack from shit15

sbi eimsk,4 ; enable int4
ret ; return to previous task and wait for shit15 to be done

wakeupshit15: ; collect data from shit15
in r17,sreg ; get sreg
push r17 ; push sreg on stack
brts shit15humread ; skip if its a humidity read
cbi eimsk,4 ; disable int4 as pe4 will still be low when returning
rcall shit15read ; read out temperature data from shit15 (msb)
rcall shit15ack3 ; acknowledge reciept of bit
mov r3,r16 ; move msb to another register
rcall shit15read ; read out temperature data from shit15 (lsb)
rcall shit15ack4 ; end transmission ack
```

```
mov r4,r16 ;  move lsb to another register
rcall shit15humidity
pop r17 ; get off stack
out sreg,r17 ; return sreg
set ; set tregister to indicate humidity request in progress
reti ; continue with previous task

shit15humread: ; read out humidity data from shit 15
rcall shit15read ; read out humidity data from shit15 (msb)
rcall shit15ack3 ; acknowledge reciept of bit
rcall shit15read ; read out humidity data from shit15 (lsb)
rcall shit15ack4 ; end transmission ack
mov r12,r16 ; move humidity data to temporary register
ldi r16,$07 ; load frame length
st y+,r16 ; move frame length to buffer
ldi r16,$03 ; set command byte to data from stationary nodes
st y+,r16 ; move frame type to buffer
st y+,r5 ; move adc lsb to buffer
st y+,r2 ; move adc msb to buffer
st y+,r4 ; move shit15 temperature lsb to buffer
st y+,r3 ; move shit15 temperature msb to buffer
st y+,r12 ; move shit15 humidity to buffer
st y+,r8 ; move activity level to buffer
sbrc yh,2 ; check if buffer at 1k
ldi yh,$02 ; reset buffer pointer to bottom of buffer
pop r17 ; get off stack
out sreg,r17 ; return sreg
clt ; reset to temperature read
reti ; return with previous task

shit15humidity: ; read humidity data from shit15
;send read humidity command
rcall startshit15 ; send start sequence

;data sequence - read humidity command $05
;nops added to keep clock at 1MHz
sbi porte,porte0 ; clock high - data bit one
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit two
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit three
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit four
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit five
```

```
nop
nop
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; data high
nop ; data takes time to rise and needs setup time
nop
nop
nop
nop
nop
sbi porte,porte0 ; clock high - data bit six
nop
nop
cbi porte,porte0 ; clock low
sbi ddre,dde4 ; data low
sbi porte,porte0 ; clock high - data bit seven
nop
nop
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; data high
nop ; data takes time to rise and needs setup time
nop
nop
nop
nop
nop
sbi porte,porte0 ; clock high - data bit eight
nop
nop
cbi porte,porte0 ; clock low
;sbi ddre,dde4 ; data low - if last databyte is 1 - leave it high for ack

rcall shit15ack0 ; wait for acknowledge from shit15
sbi eimsk,4 ; enable int4
ret ; return to previous task and wait for shit15 to take data

shit15ack4: ; end transmission ack
;nops added to keep clock at 1MHz
sbi porte,porte0 ; clock high - ack
nop
nop
cbi porte,porte0 ; clock low
ret ; return to previous task

shit15ack3: ; data recipet acknowledge
;nops added to keep clock at 1MHz
sbi ddre,dde4 ; pull data low
sbi porte,porte0 ; clock high - ack
nop
nop
cbi porte,porte0 ; clock low
cbi ddre,dde4 ; release data line
ret ; return to previous task

shit15read: ; read data from shit15
;nops added to keep clock at 1MHz and meet data valid time of 250ns
clr r16 ; clear the register where incoming data will be written
sbi porte,porte0 ; clock high - data bit one
sbic pine,pine4 ; check if data is low
sbr r16,$80 ; write data bit one to register
cbi porte,porte0 ; clock low
```

```
nop
nop
sbi porte,porte0 ; clock high - data bit two
sbic pine,pine4 ; check if data is low
sbr r16,$40 ; write data bit two to register
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit three
sbic pine,pine4 ; check if data is low
sbr r16,$20 ; write data bit three to register
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit four
sbic pine,pine4 ; check if data is low
sbr r16,$10 ; write data bit four to register
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit five
sbic pine,pine4 ; check if data is low
sbr r16,$08 ; write data bit five to register
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit six
sbic pine,pine4 ; check if data is low
sbr r16,$04 ; write data bit six to register
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit seven
sbic pine,pine4 ; check if data is low
sbr r16,$02 ; write data bit seven to register
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high - data bit eight
sbic pine,pine4 ; check if data is low
sbr r16,$01 ; write data bit eight to register
cbi porte,porte0 ; clock low
ret ; return to previous activity

startshit15: ; start sequence
;nops added to keep clock at 1MHz
cbi ddre,dde4 ; set data high
cbi porte,porte0 ; make sure clock is low
nop
nop
sbi porte,porte0 ; pull clock high
sbi ddre,dde4 ; set data low
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
cbi ddre,dde4 ; data high
nop ; data takes time to rise and needs setup time
nop
nop
nop
```

```
nop
nop
cbi porte,porte0 ; clock low
sbi ddre,dde4 ; data low
ret ; return to previous task

shit15ack0: ; ack sequence for sending data

cbi ddre,dde4 ; release data line

shit15ack2: ; check that data line is low
;nops added to keep clock at 1MHz
sbic pine,pine4
rjmp shit15ack2
sbi porte,porte0 ; clock high - ack
nop
nop
cbi porte,porte0 ; clock low

shit15ack1: ; check that line has been released

sbis pine,pine4 ; check that line has been released
rjmp shit15ack1
ret ; return to previous task

shit15reset: ; reset sequence if it gets out of phase
;status register preserved
;must be followed by data start sequence
;nops added to slow it down to 1MHz
cbi ddre,dde4 ; data high
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
nop
nop
sbi porte,porte0 ; clock high
nop
nop
cbi porte,porte0 ; clock low
ret ; return to previous task

wakeuprx: ; process incoming bytes on usart1

in r16,sreg ; get sreg
push r16 ;  push sreg on stack
ldi r16,$00
sts tcnt4h,r16 ; set counter to zero
sts tcnt4l,r16 ; set counter to zero
lds r16,udr1 ; load recieved byte
;subi r16,$20 ; test code to make ascii work
;lsl r16 ; test code to make ascii work
cpi r23,$ff ; check to see if you already have the header byte
brne usartrx ; go to data recieve

rxheadercheck: ; check for header byte

;cpi r16,$a0 ; check if header byte - testcode to make ascii work
cpi r16,$ff ; check if header byte - $ff
brne done5 ; skip if not
ldi r16,$2f ; set up byte timeout counter
out tifr4,r16 ; clear all t4 interrupt flags
ldi r16,$02
sts timsk4,r16 ; turn on t4 interrupt
ldi r16,$0d
sts tccr4b,r16 ; turn on counter, set to 128us period
clr r23 ; set byte number register to 0

done5:
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; return to previous task

usartrx: ; recieve data
```

```
tst r23 ; check for first byte
brne usartrxdata; skip if not first byte
clr xl ; reset buffer to bottom
st x+,r16 ; store recieved byte
mov r23,r16 ; move frame length to byte number
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; return to previous task

usartrxdata: ; load data bytes

st x+,r16 ; store recieved byte
dec r23 ; check for last byte
brne done5 ; continue
ldi xl,$00 ; go to bottom of buffer
ld r16,x+ ; load frame length byte
cpi r16,$03 ; check if there are at least three bytes
brlo done6 ; throw away the packet if its invalid
cpi r16,$40 ; check if there are more than 64 bytes
brsh done6 ; dont transmit if too big
ld r16,x ; load command byte
cpi r16,$08 ; check if a transmit packet
brne done6 ; skip if not a transmit packet
ldi xl,$00 ; reset to packet length byte
rcall transmit ; transmit from buffer if valid

done6: ; reset counter and return to previous task

ldi r16,$08 ; turn off timeout counter
sts tccr4b,r16 ; turn off timeout counter
ldi r16,$00
sts timsk4,r16 ; disable t4 interrupts
ldi r23,$ff ; set byte number to $ff to reset count
pop r16 ; get off stack
out sreg,r16 ; return sreg
reti ; return to previous task
```

Appendix J

System Control Code

```
import socket
import glob
import logging
import logging.handlers
import thread
import time
import ConfigParser
import numpy
import scipy
import scipy.linalg


hvacconfig = ConfigParser.RawConfigParser() # create configuration object
hvacconfig.read('hvacconfig1.cfg')

LOG_FILENAME = 'data.txt'

logger = logging.getLogger('BarnMonitor')
hdlr = logging.handlers.TimedRotatingFileHandler(
              LOG_FILENAME, when='midnight', backupCount=100)
formatter = logging.Formatter('%(message)s')
hdlr.setFormatter(formatter)
logger.addHandler(hdlr)
logger.setLevel(logging.INFO)


def temperature(t):
    return (-39.1 + .072 * t)

def temperature1(t):
    return (-39.4 + .072 * t)

def humidity(h):
    return (-2.0468 + 0.5872 * h + -.00040845 * h * h)

UDP_IP="18.85.45.140"
UDP_PORT=10001

UDP_IP_SEND="18.85.45.185"
UDP_PORT_SEND=10001

def send_command(dst, parm1, parm2): # map destinations to room nodes

    if dst == 156:
        send_add = "18.85.45.185"
    elif dst == 160:
        send_add = "18.85.45.131"
    elif dst == 176:
        send_add = "18.85.45.145"
    elif dst == 180:
        send_add = "18.85.45.170"
    elif dst == 184:
        send_add = "18.85.45.142"
    elif dst == 188:
        send_add = "18.85.45.185"
    elif dst == 192:
        send_add = "18.85.16.117"
    elif dst == 200:
        send_add = "18.85.45.185"
    elif dst == 204:
        send_add = "18.85.16.117"
    message = chr(0xff) + chr(0x06) + chr(0x08) + chr(dst) + chr(0xaa) + chr(0x08) + chr(parm1) + chr(parm2)
    sock = socket.socket(socket.AF_INET, # Internet
                         socket.SOCK_DGRAM) # UDP
    sock.sendto(message, (send_add, UDP_PORT_SEND))
    sock.close()
    print "SENT: 0x%x 0x%x 0x%x 0x%x 0x%x 0x%x 0x%x 0x%x IP: %s PORT: %s" % (ord(message[0]),ord(message[1]),ord(message[2]),
                        ord(message[3]),ord(message[4]),ord(message[5]),ord(message[6]),ord(message[7]), send_add, UDP_PORT)


class Location:

    # create location object
    def __init__(self, data_id): # takes a packet which is [rssi, dst] and id of the portable node being tracked
        self.__lock = thread.allocate_lock() # get a thread lock in case the timeout happens when data is being transferred
        self.__first = True # set start flag to first time
        self.__packet = [0, 0] # initialize packet
        self.__location_out = 0 # clear the output buffer
        self.__location_array = [] # clear the location array
        self.__location_out1 = 1 # last sample - for lowpassing
        self.__location_out2 = 2 # two samples ago - for lowpassing
        self.__location_out3 = 0 # temp register for lowpassing
        self.__data_id = data_id # id of portable being tracked

    # run location object
    def run(self, packet):
        self.__packet = packet
        self.__lock.acquire() # get a lock in case the data is being modified
        if self.__first: # check if first packet
            self.__location_array = [self.__packet] # initialize packet array with first rssi value
            self.__first = False # unset start flag
            thread.start_new_thread(self.__sort, ()) # start timeout timer for packet aggregation
        else:
            self.__location_array.append(packet) # put next rssi value on to array
        self.__lock.release() # release lock

    # allow for passing of location to other programs
    def where(self):
        return self.__location_out3

    # sort locations after timeout
    def __sort(self):
        time.sleep(.05) # set timeout for packet waiting
        self.__lock.acquire() # get a lock in case the data is being modified
        self.__location_array.sort() # sort the array to find highest rssi
        self.__location_out = self.__location_array[-1][1] # get location for highest rssi
        self.__location_out3 = self.__location_out # make a temporary copy
        if self.__location_out1 == self.__location_out2: # check if the location changed
            self.__location_out3 = self.__location_out2 # keep it the same if it changed
        self.__location_out2 = self.__location_out1 # move past sample to two samples ago
        self.__location_out1 = self.__location_out # move sample to past sample
        self.__first = True # reset start flag
        self.__location_array = [] # clear the location array
        logger.info("%f %d %d %d" % (time.time(), 13, self.__data_id, self.__location_out3)) # log the data to a file for testing
        self.__lock.release() # release lock


class Room:

    # create room object
    def __init__(self, data_id, buffsize, levelin, levelout, control_id): # takes in room motion sensor and room id
        self.__occupied = True # variable for determining if room is occupied
        self.__motion = 0 # motion sensor variable
        self.__timer = 0 # manual control timeout
        self.__timer1 = time.time() # entry control timeout
        self.__timer2 = time.time() # comfort control timeout
        self.__time = time.time() # current time
        self.__data_id = data_id # id of room node
        self.__temp = 0 # room temperature buffer
        self.__hum = 0 # room humidity buffer
        self.__control = [] # control vector
        self.__normal = True # initialize auto/normal control flag
        self.__setback = False # initialize setback control flag
        self.__buff = [0 for i in range(70)] # create data buffer for first entry
        self.__levelin = levelin # threshold for detecting person in room
        self.__levelout = levelout # threshold for detecting no one in room
        self.__daytime = 0 # buffer for time of day
        self.__entry = True # in system flag for entry
        self.__entry_buff = eval(hvacconfig.get('entrybuffer', str(self.__data_id))) # create entry buffer
        self.__buffsize = buffsize # setup buffer size for occupancy calculation
        self.__control_id = control_id
        self.__flip = False
        self.__range = False

    # check to see if someone is in the room
    def run(self, motion):
        self.__motion = motion >= 1 # clip off all motion above one
        del self.__buff[-1] # do buffer rotate
        self.__buff.insert(0,self.__motion) # do buffer rotate
        self.__time = time.time()
        self.__daytime = int(divmod(self.__time, 86400)[1])
        if (self.__occupied == True) and (self.__time - self.__timer < 10800): # if occupied - check to see if still occupied
            if (numpy.sum(self.__buff[:self.__buffsize]) >= self.__levelout): # see if there is motion
                self.__timer = self.__time # reset counter if occupied
        elif (self.__occupied == True) and (self.__time - self.__timer >= 10800): # if timer expires - set to not occupied
            self.__occupied = False # set occupancy flag
            logger.info("%f %d %d" % (self.__time, 15, self.__data_id)) # log the data to a file for testing
        else:
            if (numpy.sum(self.__buff[:self.__buffsize]) >= self.__levelin): # see if there is motion
                self.__occupied = True # set occupancy flag
                self.__timer = self.__time # reset timer
                logger.info("%f %d %d" % (self.__time, 14, self.__data_id)) # log the data to a file for testing
            else:
                for x in self.__entry_buff[:]:
                    if self.__time - x[0] > 604800: # check if a week old
                        self.__entry_buff.remove(x) # eliminate the element
                for i in range(len(self.__entry_buff)):
                    # the next line checks if an entry time is within n hours
                    # first check is time till entry, second check is time since entry
                    if (divmod(self.__entry_buff[i][1] - self.__daytime, 86400)[1] < 7500) or
                       (divmod(self.__daytime - self.__entry_buff[i][1], 86400)[1] < 6000):
                        self.__range = True
                        break
            else:
                self.__range = False
if (self.__entry == True) and (self.__time - self.__timer1 < 10800): # if occupied - check to see if still occupied
    if (numpy.sum(self.__buff) >= 10): # see if there is motion
        self.__timer1 = self.__time # reset counter if occupied
elif (self.__entry == True) and (self.__time - self.__timer1 >= 10800): # if timer expires - set to not occupied
    self.__entry = False # set occupancy flag
    logger.info("%f %d %d" % (self.__time, 21, self.__data_id)) # log the data to a file for testing
else:
    if (numpy.sum(self.__buff) >= 40): # see if there is motion
        self.__entry = True # set occupancy flag
        self.__timer1 = self.__time # reset timer
        logger.info("%f %d %d %d" % (self.__time, 20, self.__data_id, self.__daytime)) # log
        self.__entry_buff.append([self.__time, self.__daytime]) # store time for predictions
        hvacconfig.set('entrybuffer',str(self.__data_id),str(self.__entry_buff)) # store buffer to configuration file
        f=open('hvacconfig1.cfg','wb') # send data to file
        hvacconfig.write(f)
        f.close()
if self.__flip == False: # do an alternating bit to limit control cycle to once a minute
    self.__flip = True
else:
    self.__flip = False
    if (self.__occupied == True) and (self.__time - self.__timer < 400): # find out who it is
        self.__control = [] # reset control vector
        if self.__data_id == 32: # only allow specific users control
            if (location_96.where() == self.__data_id) and (self.__time - activity_96.active() < 90):
                self.__control = self.__control + [activity_96.comfort()]
            if (location_104.where() == self.__data_id) and (self.__time - activity_104.active() < 90):
                self.__control = self.__control + [activity_104.comfort()]
            if (location_28.where() == self.__data_id) and (self.__time - activity_28.active() < 90):
                self.__control = self.__control + [activity_28.comfort()]
        elif self.__data_id == 36:
            if (location_4.where() == self.__data_id) and (self.__time - activity_4.active() < 90):
                self.__control = self.__control + [activity_4.comfort()]
            if (location_8.where() == self.__data_id) and (self.__time - activity_8.active() < 90):
                self.__control = self.__control + [activity_8.comfort()]
            if (location_20.where() == self.__data_id) and (self.__time - activity_20.active() < 90):
                self.__control = self.__control + [activity_20.comfort()]
            if (location_24.where() == self.__data_id) and (self.__time - activity_24.active() < 90):
                self.__control = self.__control + [activity_24.comfort()]
            if (location_28.where() == self.__data_id) and (self.__time - activity_28.active() < 90):
                self.__control = self.__control + [activity_28.comfort()]
            if (location_76.where() == self.__data_id) and (self.__time - activity_76.active() < 90):
                self.__control = self.__control + [activity_76.comfort()]
            if (location_84.where() == self.__data_id) and (self.__time - activity_84.active() < 90):
                self.__control = self.__control + [activity_84.comfort()]
            if (location_88.where() == self.__data_id) and (self.__time - activity_88.active() < 90):
                self.__control = self.__control + [activity_88.comfort()]
            if (location_96.where() == self.__data_id) and (self.__time - activity_96.active() < 90):
                self.__control = self.__control + [activity_96.comfort()]
            if (location_104.where() == self.__data_id) and (self.__time - activity_104.active() < 90):
                self.__control = self.__control + [activity_104.comfort()]
        elif self.__data_id == 40:
            if (location_4.where() == self.__data_id) and (self.__time - activity_4.active() < 90):
                self.__control = self.__control + [activity_4.comfort()]
            if (location_8.where() == self.__data_id) and (self.__time - activity_8.active() < 90):
                self.__control = self.__control + [activity_8.comfort()]
            if (location_20.where() == self.__data_id) and (self.__time - activity_20.active() < 90):
                self.__control = self.__control + [activity_20.comfort()]
            if (location_24.where() == self.__data_id) and (self.__time - activity_24.active() < 90):
                self.__control = self.__control + [activity_24.comfort()]
            if (location_28.where() == self.__data_id) and (self.__time - activity_28.active() < 90):
                self.__control = self.__control + [activity_28.comfort()]
            if (location_76.where() == self.__data_id) and (self.__time - activity_76.active() < 90):
                self.__control = self.__control + [activity_76.comfort()]
            if (location_84.where() == self.__data_id) and (self.__time - activity_84.active() < 90):
                self.__control = self.__control + [activity_84.comfort()]
            if (location_88.where() == self.__data_id) and (self.__time - activity_88.active() < 90):
                self.__control = self.__control + [activity_88.comfort()]
            if (location_96.where() == self.__data_id) and (self.__time - activity_96.active() < 90):
                self.__control = self.__control + [activity_96.comfort()]
            if (location_104.where() == self.__data_id) and (self.__time - activity_104.active() < 90):
                self.__control = self.__control + [activity_104.comfort()]
        elif self.__data_id == 44:
            if (location_8.where() == self.__data_id) and (self.__time - activity_8.active() < 90):
                self.__control = self.__control + [activity_8.comfort()]
            if (location_88.where() == self.__data_id) and (self.__time - activity_88.active() < 90):
                self.__control = self.__control + [activity_88.comfort()]
        elif self.__data_id == 48:
            if (location_4.where() == self.__data_id) and (self.__time - activity_4.active() < 90):
                self.__control = self.__control + [activity_4.comfort()]
            if (location_20.where() == self.__data_id) and (self.__time - activity_20.active() < 90):
                self.__control = self.__control + [activity_20.comfort()]
        elif self.__data_id == 196:
            if (location_76.where() == self.__data_id) and (self.__time - activity_76.active() < 90):
                self.__control = self.__control + [activity_76.comfort()]
        if len(self.__control) > 0: # if someone was in the room
```

```
                self.__normal = False # turn off normal control
                self.__setback = False # turn off setback control
                self.__control = numpy.mean(self.__control) # arbitrate between users
                self.__control_id.run(self.__control) # control damper
                self.__timer2 = self.__time
            elif self.__time - self.__timer2 > 400: # check if comfort control in past 6 minutes
                self.__normal = True
                self.__setback = False
        elif (self.__occupied == True) or (self.__range == True):
            self.__normal = True # set to normal control
            self.__setback = False # turn off setback control
        else:
            self.__normal = False
            self.__setback = True

    # output occupied global variable
    def occupied(self):
        return self.__occupied

    def normal(self):
        return self.__normal

    def setback(self):
        return self.__setback


class Window:

    # create window object
    def __init__(self, data_id): # takes in data motor state, wind speed
        self.__manual = False # variable for determining if window is under manual control
        self.__closed = False # variable for determining if window is closed
        self.__motor = 0 # motor state variable
        self.__wind = 0 # wind speed variable
        self.__timer = 0 # manual control timeout
        self.__data_id = data_id # id of window under control

    # do over speed control and variable determination
    def run(self, motor, wind):
        self.__motor = motor
        self.__wind = wind
        self.__closed = (((self.__motor) & 0x03) == 0x02) # test for jam closed
        if ((self.__motor) & 0x68) > 1: # test if buttons pressed or manual bit set
            self.__timer = time.time() # reset timeout counter
            self.__manual = True # set manual flag
        else: # if no buttons, test for 3 hour timeout
            self.__manual = ((time.time() - self.__timer) < 10800) # see if its been 3 hours since last manual control
        if (self.__manual == False) and (self.__closed == False): # if not under manual control and open
            if self.__wind > 1000: # check if wind speed too high
                send_command(self.__data_id, 0x06, 0x20) # close window

    # output manual global variable
    def man(self):
        return self.__manual

    # output closed global variable
    def shut(self):
        return self.__closed


class Activity:

    # initialize activity buffers
    def __init__(self, start_act, start_var, reset_act, reset_var, cont_temp, expire, data_id):
        self.__start_act = start_act
        self.__start_var = start_var
        self.__reset_act = reset_act
        self.__reset_var = reset_var
        self.__cont_temp = cont_temp
        self.__expire = expire
        self.__timeout = 0
        self.__last_time = 0
        self.__time = 0
        self.__buff1 = 0
        self.__buff2 = 0
        self.__buff = [0, 0, 0]
        self.__data_id = data_id
        self.__data_id_cold = data_id + 1
        self.__data_id_hot = data_id + 2
        self.__data_id_tran = data_id + 3
        self.__gain_id = data_id + 200
        self.__time_active = 0 # buffer for last time active
        self.__hum = 0
        self.__button = 0
        self.__butt_time = 0
        self.__activity = False # activity flag
        self.__active2 = False # activity timeout flag
        self.__timer2 = time.time()
        self.__buff_cold = eval(hvacconfig.get('comfortbuffer', str(self.__data_id_cold))) # create cold buffer
        self.__buff_hot = eval(hvacconfig.get('comfortbuffer', str(self.__data_id_hot))) # create hot buffer
        self.__mean_cold = []
        self.__mean_hot = []
        self.__scat_cold = []
        self.__scat_hot = []
        self.__tran_hot = []
        self.__tran_cold = []
        self.__transform = numpy.matrix(eval(hvacconfig.get('comfortbuffer', str(self.__data_id_tran)))).T
        self.__boundary = hvacconfig.getfloat('comfortbuffer', str(self.__data_id))
        self.__comf_dis = 0
        self.__time_active2 = 0
        self.__temp = 80
        self.__hum = 35
        self.__gain = hvacconfig.getfloat('comfortbuffer', str(self.__gain_id))

    def run(self, act, temp, hum, button):
        self.__act = act
        self.__temp = temp
        self.__hum = hum
        self.__button = button
        self.__time = time.time()
        if (self.__time - self.__last_time > 55) and (self.__button == 0): # check if not repeat packet or button press
            self.__buff = [self.__buff2, self.__buff1, self.__act] # create buffer
            if (self.__time - self.__timeout < self.__expire): # check if activity timeout has expired
                if (numpy.var(self.__buff) > self.__reset_var) or (self.__act > self.__reset_act): # check for restart conditions
                    self.__activity = True
                    logger.info("%f %d %d %d" % (self.__time, 18, self.__data_id, 100))
                    self.__timeout = self.__time
                    self.__time_active = self.__time
                elif (self.__act > self.__cont_temp): # check for continue conditions
                    self.__activity = True
                    self.__time_active = self.__time
                    logger.info("%f %d %d %d" % (self.__time, 18, self.__data_id, 100))
                else:
                    self.__activity = False
                    logger.info("%f %d %d %d" % (self.__time, 18, self.__data_id, 0))
            else:
                if (numpy.var(self.__buff) > self.__start_var) or (self.__act > self.__start_act): # check for start conditions
                    self.__timeout = self.__time
                    self.__activity = True
                    self.__time_active = self.__time
                    logger.info("%f %d %d %d" % (self.__time, 18, self.__data_id, 100))
                else:
                    self.__activity = False
                    logger.info("%f %d %d %d" % (self.__time, 18, self.__data_id, 0))
            self.__last_time = self.__time # reset timer with last packet time
            self.__buff2 = self.__buff1 # shift buffer
            self.__buff1 = self.__act # shift buffer
            if (self.__active2 == True) and (self.__time - self.__time_active > 900): # check if its been 20 minutes
                self.__active2 = False # set 15min activity flag to false
            elif (self.__active2 == False) and (self.__activity == True):
                self.__active2 = True
                self.__timer2 = self.__time
                logger.info("%f %d %d %d" % (self.__time, 22, self.__data_id, 100))
        if (self.__activity == True) and (self.__time - self.__timer2 > 900):
            self.__time_active2 = self.__time
        if (self.__button > 0) and (self.__time - self.__butt_time > 1800) and (self.__activity == True) and
                                                              (self.__time - self.__timer2 > 900):
            self.__butt_time = self.__time
            if self.__button == 8:
                if len(self.__buff_cold) >= 9:
                    del self.__buff_cold[-1]
                self.__buff_cold.insert(0,[self.__temp, self.__hum])
                hvacconfig.set('comfortbuffer', str(self.__data_id_cold), str(self.__buff_cold)) # store buffer to configuration
                f=open('hvacconfig1.cfg','wb') # send data to file
                hvacconfig.write(f)
                f.close()
                self.__fisher()
            elif self.__button == 1:
                if len(self.__buff_hot) >= 9:
                    del self.__buff_hot[-1]
                self.__buff_hot.insert(0,[self.__temp, self.__hum])
                hvacconfig.set('comfortbuffer', str(self.__data_id_hot), str(self.__buff_hot)) # store buffer to configuration
                f=open('hvacconfig1.cfg','wb') # send data to file
                hvacconfig.write(f)
                f.close()
                self.__fisher()

    def __fisher(self):
        self.__scat_cold = numpy.matrix([[0, 0], [0, 0]])
        self.__scat_hot = numpy.matrix([[0, 0], [0, 0]])
        self.__mean_hot = numpy.matrix(numpy.mean(self.__buff_hot,0))
        self.__mean_cold = numpy.matrix(numpy.mean(self.__buff_cold,0))
        for i in range(len(self.__buff_cold)):
            k = numpy.subtract(self.__buff_cold[i], self.__mean_cold)
            self.__scat_cold = numpy.add(self.__scat_cold, (k.T * k))
        for i in range(len(self.__buff_hot)):
            k = numpy.subtract(self.__buff_hot[i], self.__mean_hot)
            self.__scat_hot = numpy.add(self.__scat_hot, (k.T * k))
        self.__transform = numpy.subtract(self.__mean_cold, self.__mean_hot)
        self.__transform = self.__transform.T
        self.__transform = (scipy.linalg.inv(numpy.add(self.__scat_cold, self.__scat_hot))) * self.__transform
        self.__tran_cold = self.__buff_cold * self.__transform
        self.__tran_hot = self.__buff_hot * self.__transform
        self.__boundary = numpy.mean(numpy.vstack((numpy.msort(self.__tran_cold)[:2], numpy.msort(self.__tran_hot)[-2:])))
        self.__gain = -1.5 / (abs((self.__mean_hot * self.__transform) - self.__boundary) +
                             abs((self.__mean_cold * self.__transform) - self.__boundary))
        self.__gain = self.__gain[0,0]
        hvacconfig.set('comfortbuffer', str(self.__gain_id), str(self.__gain))
        hvacconfig.set('comfortbuffer', str(self.__data_id), str(self.__boundary))
        g=numpy.array(self.__transform)
        hvacconfig.set('comfortbuffer', str(self.__data_id_tran), str([g[0,0], g[1,0]]))
        f=open('hvacconfig1.cfg','wb') # send data to file
        hvacconfig.write(f)
        f.close()

    def active(self):
        return self.__time_active2

    def comfort(self):
        self.__comf_dis = ((numpy.matrix([self.__temp, self.__hum]) * self.__transform)[0,0] - self.__boundary) *
                                                                                         (self.__gain) * 0.4
        logger.info("%f %d %d %f %f %f %f %f %f %f" % (self.__time, 23, self.__data_id, self.__comf_dis, self.__gain,
                              self.__boundary, self.__transform[0,0], self.__transform[1,0], self.__temp, self.__hum))
        if self.__comf_dis > 1:
            self.__comf_dis = 1
        elif self.__comf_dis < -1:
            self.__comf_dis = -1
        return self.__comf_dis


class Thermostat:

    def __init__(self, data_id, room_id, control_id, setback_temp, normal_temp):
        self.__data_id = data_id
        self.__control_id = control_id
        self.__temp = 72
        self.__hum = 40
        self.__room_id = room_id
        self.__cold = False
        self.__setb = setback_temp
        self.__norm = normal_temp

    def run(self, temp, hum):
        self.__hum = hum
        self.__temp = temp
        if self.__room_id.normal() == True:
            self.__control_id.run(self.__temp - self.__norm) # normal temperature
        elif self.__room_id.setback() == True:
            self.__control_id.run(self.__temp - self.__setb) # setback temperature

    def temp(self):
        return self.__temp

    def hum(self):
        return self.__hum

    def enth(self):
        return ((.007468 * self.__temp * self.__temp) - (.4344 * self.__temp) + 11.1769) * self.__hum / 100 +
                                                                              (.2372 * self.__temp) + .1230

    def cold(self):
        if self.__room_id.normal() == True:
            return self.__temp < self.__norm - 1 # normal lower limit
        elif self.__room_id.setback() == True:
            return self.__temp < 65 # setback lower limit
        else:
            return self.__temp < 70 # auto lower limit


class Outdoor:

    def __init__(self, data_id, therm_id, window_id, command_id):
        self.__data_id = data_id
        self.__therm_id = therm_id
        self.__control_id = window_id
        self.__command_id = command_id
        self.__temp = 72
        self.__hum = 60
        self.__enth = 30

    def run(self, temp, hum):
        self.__temp = temp
        self.__hum = hum
        if (self.__therm_id.cold() == True) and (self.__control_id.man() == False) and (self.__control_id.shut() == False):
            send_command(self.__command_id, 0x06, 0xff) # shut window full
        elif (self.__control_id.man() == False) and (self.__therm_id.cold() == False): # is it under manual or is it too cold
            self.__enth = ((.007468 * self.__temp * self.__temp) - (.4344 * self.__temp) + 11.1769) * self.__hum / 100 +
                                                                                     (.2372 * self.__temp) + .1230
            logger.info("%f %d %d %f %f" % (time.time(), 19, self.__data_id, self.__enth, self.__therm_id.enth()))
            if (self.__control_id.shut() == True) and (self.__temp < 78) and (self.__enth < (self.__therm_id.enth() - 1)):
                send_command(self.__command_id, 0x08, 0x80) # open window a bit
                logger.info("%f %d %d %d" % (time.time(), 16, self.__data_id, 100))
            elif (self.__control_id.shut() == False) and (self.__enth > (self.__therm_id.enth())):
                send_command(self.__command_id, 0x06, 0xff) # full close window
                logger.info("%f %d %d %d" % (time.time(), 16, self.__data_id, 0))

    def temp(self):
        return self.__temp

    def hum(self):
        return self.__hum


class Control:

    def __init__(self, dst, offset, pgain, igain):
        self.__dst = dst
        self.__pgain = pgain
        self.__igain = igain
        self.__time = 0
        self.__last_time = time.time()
        self.__offset = offset
        self.__position = 0
        self.__last_error = 0

    def run(self, error):
        self.__error = error
        self.__time = time.time()
        if self.__time - self.__last_time > 50: # remove repeat packets
            self.__pos = (self.__error - self.__last_error) / (self.__time - self.__last_time) # get positional data
            if ((self.__position > 255) and (self.__error > 0)) or ((self.__position < -255) and (self.__error < 0)):
                self.__signal = int(self.__pos * self.__pgain)
            else:
                self.__signal = int((self.__pos * self.__pgain) + (self.__igain * self.__error)) # make control signal
            self.__last_error = self.__error
            self.__last_time = self.__time
            if self.__signal > 253:
                self.__signal = 254
            elif self.__signal < -253:
                self.__signal = -254
            if (abs(self.__error) > self.__offset) and (abs(self.__position) < 255):
                if (self.__signal > 0):
                    send_command(self.__dst, 0x08, self.__signal)
                    if (self.__dst == 188):
                        send_command(0xc8, 0x08, self.__signal) # repeat signal for second unit
                elif (self.__signal < 0):
                    send_command(self.__dst, 0x06, abs(self.__signal))
                    if (self.__dst == 188):
                        send_command(0xc8, 0x06, abs(self.__signal)) # repeat signal for second unit
            self.__position = self.__position + self.__signal


location_4 = Location(4)
location_8 = Location(8)
location_20 = Location(20)
location_24 = Location(24)
location_28 = Location(28)
location_76 = Location(76)
location_84 = Location(84)
location_88 = Location(88)
location_96 = Location(96)
location_104 = Location(104)

control_160 = Control(160, .1, 8000, 10)
control_176 = Control(176, .1, 8000, 10)
control_180 = Control(180, .1, 8000, 10)
control_184 = Control(184, .1, 8000, 10)
control_188 = Control(188, .1, 4000, 5)
control_204 = Control(204, .1, 8000, 10)

room_32 = Room(32, 12, 7, 4, control_188)
room_36 = Room(36, 12, 7, 4, control_180)
room_40 = Room(40, 12, 7, 4, control_176)
room_44 = Room(44, 12, 7, 4, control_184)
room_48 = Room(48, 12, 7, 4, control_160)
room_196 = Room(196, 12, 7, 4, control_204)

activity_4 = Activity(70, 200, 45, 15, 80, 360, 4)
```

```
activity_8 = Activity(120, 200, 70, 25, 80, 360, 8)
activity_20 = Activity(70, 200, 50, 25, 80, 360, 20)
activity_24 = Activity(70, 200, 45, 15, 80, 360, 24)
activity_28 = Activity(50, 200, 20, 15, 80, 360, 28)
activity_76 = Activity(70, 200, 30, 15, 80, 360, 76)
activity_84 = Activity(100, 200, 65, 25, 80, 360, 84)
activity_88 = Activity(70, 200, 30, 15, 80, 360, 88)
activity_96 = Activity(70, 200, 50, 20, 80, 360, 96)
activity_104 = Activity(70, 200, 30, 15, 80, 360, 104)

window_156 = Window(156)
window_192 = Window(192)

thermo_216 = Thermostat(216, room_196, control_204, 78, 72)
thermo_220 = Thermostat(220, room_44, control_184, 80, 74)
thermo_224 = Thermostat(224, room_48, control_160, 78, 72)
thermo_228 = Thermostat(228, room_32, control_188, 80, 74)
thermo_232 = Thermostat(232, room_36, control_180, 76, 72)
thermo_236 = Thermostat(236, room_40, control_176, 76, 72)

outdoor_148 = Outdoor(148, thermo_228, window_156, 156)
outdoor_172 = Outdoor(172, thermo_216, window_192, 192)


sock = socket.socket( socket.AF_INET, # Internet
                      socket.SOCK_DGRAM ) # UDP
sock.bind((UDP_IP,UDP_PORT))


while True:
    data, addr = sock.recvfrom( 1024 ) # buffer size is 1024 bytes
    if (len(data)==ord(data[1])+6) and (ord(data[0]) == 0xff) and (ord(data[ord(data[1])+4]) == 0xfe) and
                                                                  (ord(data[ord(data[1])+5]) == 0xfd):
        type1 = ord(data[2])
        if type1 == 0x03: # room data
            src = ord(data[9])
            light = ord(data[4]) * 256 + ord(data[3])
            temp = temperature1(ord(data[6]) * 256 + ord(data[5]))
            hum = humidity(ord(data[7]))
            act = ord(data[8])
            logger.info("%f %d %d %f %f %d %d" % (time.time(), type1, light, temp, hum, act, src))
            if src == 32:
                room_32.run(act)
            elif src == 36:
                room_36.run(act)
            elif src == 40:
                room_40.run(act)
            elif src == 44:
                room_44.run(act)
            elif src == 48:
                room_48.run(act)
            elif src == 196:
                room_196.run(act)
        elif type1 == 0x00: # location
            src = ord(data[3])
            dst = ord(data[7])
            lqi = ord(data[5])
            rssi = ord(data[6])
            logger.info("%f %d %d %d %d %d" % (time.time(), type1, src, lqi, rssi, dst))
            if src == 4:
                location_4.run([rssi, dst])
            elif src == 8:
                location_8.run([rssi, dst])
            elif src == 20:
                location_20.run([rssi, dst])
            elif src == 24:
                location_24.run([rssi, dst])
            elif src == 28:
                location_28.run([rssi, dst])
            elif src == 76:
                location_76.run([rssi, dst])
            elif src == 84:
                location_84.run([rssi, dst])
            elif src == 88:
                location_88.run([rssi, dst])
            elif src == 96:
                location_96.run([rssi, dst])
            elif src == 104:
                location_104.run([rssi, dst])
        elif type1 == 0x02: # data from portable
            src = ord(data[3])
            dst = ord(data[15])
            button = ord(data[5])
            light = ord(data[12]) * 256 + ord(data[11])
            temp = temperature(ord(data[8]) * 256 + ord(data[7]))
            hum = humidity(ord(data[6]))
            act = 0x3ff - (ord(data[10]) * 256 + ord(data[9]))
        lqi = ord(data[13])
        rssi = ord(data[14])
        logger.info("%f %d %d %d %f %f %d %d %d %d %d" % (time.time(), type1, src, button, hum, temp, act, light, lqi,
                                                                                                          rssi, dst))
        if src == 4:
            activity_4.run(act, temp, hum, button)
        elif src == 8:
            activity_8.run(act, temp, hum, button)
        elif src == 20:
            activity_20.run(act, temp, hum, button)
        if src == 24:
            activity_24.run(act, temp, hum, button)
        elif src == 28:
            activity_28.run(act, temp, hum, button)
        elif src == 76:
            activity_76.run(act, temp, hum, button)
        elif src == 84:
            activity_84.run(act, temp, hum, button)
        elif src == 88:
            activity_88.run(act, temp, hum, button)
        elif src == 96:
            activity_96.run(act, temp, hum, button)
        elif src == 104:
            activity_104.run(act, temp, hum, button)
        elif src == 148:
            outdoor_148.run(temp, hum)
        elif src == 172:
            outdoor_172.run(temp, hum)
        elif src == 216:
            thermo_216.run(temp, hum)
        elif src == 220:
            thermo_220.run(temp, hum)
        elif src == 224:
            thermo_224.run(temp, hum)
        elif src == 228:
            thermo_228.run(temp, hum)
        elif src == 232:
            thermo_232.run(temp, hum)
        elif src == 236:
            thermo_236.run(temp, hum)
    elif type1 == 0x07: # data from control
        src = ord(data[3])
        dst = ord(data[15])
        motor = ord(data[5])
        light = ord(data[12]) * 256 + ord(data[11])
        temp = temperature1(ord(data[8]) * 256 + ord(data[7]))
        hum = humidity(ord(data[6]))
        wind = ord(data[10]) * 256 + ord(data[9])
        lqi = ord(data[13])
        rssi = ord(data[14])
        logger.info("%f %d %d %d %f %f %d %d %d %d %d" % (time.time(), type1, src, motor, hum, temp, wind, light, lqi,
                                                                                                          rssi, dst))
        if src == 156:
            window_156.run(motor, wind)
        elif src == 192:
            window_192.run(motor, wind)
    #elif type1 == 0x09: # data from power
        #src = ord(data[3])
        #dst = ord(data[9])
        #power = ord(data[6])*256 + ord(data[5])
        #lqi = ord(data[7])
        #rssi = ord(data[8])
        #logger.info("%f %d %d %d %d %d" % (time.time(), type1, src, power, lqi, rssi, dst))
    elif type1 == 0x01: # join request
        src = ord(data[3])
        dst = ord(data[7])
        lqi = ord(data[5])
        rssi = ord(data[6])
        logger.info("%f %d %d %d %d %d" % (time.time(), type1, src, lqi, rssi, dst))
    else:
        print("UNKNOWN PACKET TYPE length: %d type: %d src: 0x%x" % (ord(data[1]), ord(data[2]), ord(data[3])))
else:
    print("BAD PACKET TYPE length: %d type: %d src: 0x%x" % (ord(data[1]), ord(data[2]), ord(data[3])))
```

Bibliography

[1] US Bureau of Labor Statistics. Employee tenure in the mid-1990s. 1997. http://www.bls.census.gov/cps/pub/tenure_0296.htm.

[2] Smart Home Systems, Inc. http://www.smarthomeusa.com/.

[3] Xanboo, Inc. http://www.xanboo.com/.

[4] X10, Ltd. http://www.x10.com/homepage.htm.

[5] Carrier Corporation: Infinity System. http://www.residential.carrier.com/products/controls/wireless.shtml.

[6] Eaton Corporation: HOMEheartbeat. http://www.homeheartbeat.com/HomeHeartBeat/index.htm.

[7] Home Comfort Zones, Inc.: MyTemp. http://www.homecomfortzones.com.

[8] Federspiel Controls. http://www.federspielcontrols.com.

[9] W.J. Fisk and A.H. Rosenfeld. Estimates of improved productivity and health from better indoor environments. *Indoor Air*, 7:158–172, 1997.

[10] J. Kreider, P. Curtiss, and A. Rabi. *Heating and Cooling of Buildings: Design for Efficiency*. McGraw-Hill, 2002.

[11] U.S. Energy Information Administration. Petroleum navigator. 2007.

[12] U.S. Energy Information Administration. Energy consumption by sector. *2006 Annual Energy Review*, 2007.

[13] U.S. Energy Information Agency. Residential end use electricity consumption. *Residential End Use Consumption Survey, 2001*, 2007.

[14] U.S. Energy Information Administration. Preliminary end use consumptin estimates. *Commercial Building End Use Consumption Survey, 1999*, 2003.

[15] R.C. Diamond. An overview of the US Building Stock. *Indoor air quality handbook*, 2001.

[16] M.C. Mozer. The neural network house: An environment that adapts to its inhabitants. In *Proceedings of the American Association for Artificial Intelligence Spring Symposium on Intelligent Environments*, pages 110–114, Menlo Park, CA, 1998. AAAI Press.

[17] M. Hordeski. *HVAC Control in the New Millenium.* Marcel Dekker, 2001.

[18] M. Arima, E.H. Hara, and J.D. Katsberg. A fuzzy logic and rough sets controller for hvac systems. In *Proceedings of the IEEE WESCANEX*, pages 133–138, 1995.

[19] P.Y. Glorennec. Application of fuzzy control for building energy management. *Building Simulation: International Building Performance Simulation Association*, pages 197–201, 1991.

[20] S. Huang and R.M. Nelson. Rule development and adjustment strategies of a fuzzy logic controller for an hvac system – parts i and ii (analysis and experiment). *ASHRAE Transacations*, 100(1):841–856, 1994.

[21] S.J. Hepworth and A.L. Dexter. Neural control of non-lindear hvac plants. In *Proceedings of the Third IEEE Conference on Control Applications, Vol. 3*, pages 1849–1854, 1994.

[22] K.J. Astrom, T. Hagglund, and A. Wallenborg. Automatic tuning of digital controllers with applications to hvac plants. *Automatica*, 29:1333–1343, 1993.

[23] P. Curtiss, J. Kreider, and M. Bralnclemuehl. Local and global control of commercial building hvac systems using artificial neural networks. In *Proceedings of the American Control Conference, Vol. 3*, pages 3029–3044, 1994.

[24] A. Kelly. Understanding hvac economics. *Building Operation Management*, Oct. 2002.

[25] I.H. Lin and H.L. Broberg. Internet based monitoring and controls for hvac applications. *IEEE Industry Applications Magazine*, 8(1):49–54, 2002.

[26] E. Arens and et al. How ambient intelligence will improve habitability and energy efficiency in buildings. *Ambient Intelligence*, pages 63–80, 2005.

[27] C. Lin, D. Auslander, and C. Federspiel. Multi-sensor single-acutuator control of hvac systems. *Energy Systems Laboratory*, 2002.

[28] M. Kintner-Meyer and R. Conant. Opportunities of wireless sensors and controls for building operation. *Energy Engineering Journal*, 102(5):27–48, 2005.

[29] Alcala R. and et al. Fuzzy control of hvac systems optimized by genetic algorithms. *Applied Intelligence*, 18(2):155–177, 2003.

[30] N. Nassif, S. Kajl, and R. Sabourin. Optimization of hvac control system strategy using two-objective genetic algorithm. *HVAC&R Research*, 11(3), 2005.

[31] M Anderson and et al. Mimo robust control for heating, ventilating and air conditioning (hvac) systems. In *Proceedings of the 41st IEEE Conference on Decision and Control*, pages 167–172, 2002.

[32] A. Yahiaoui and et al. Specificatin for real-time control using hybrid systems in building automation systems. In *Proceedings of the 17th International Air-conditioning and Ventilation Conference*, 2006.

[33] G. Guarracino and et al. Advanced control systems for energy and environmental performance of buildings. In *Solar Thermal Technologies for Buildings*, pages 65–89, London, UK, 2003. James and James Ltd.

[34] Q.G. Wang and et al. Specification for real-time control using hybrid systems in building automation systems. In *Proceedings of the American Control Conference, Vol. 6*, pages 4353–4357, 1999.

[35] S. Sharples, V. Callaghan, and G. Clarke. A multi-agent architecture for intelligent building sensing and control. *International Sensor Review Journal*, 1999.

[36] M. Liu, D.E. Claridge, and W.D. Turner. Continuous commissioning of building energy systems. *Journal of Solar Energy Engineering*, 125(3):275–281, 2003.

[37] Cimetrics, Inc.: *info*metrics. http://www.cimetrics.com/home/infometrics/.

[38] P.O. Fanger. *Thermal Comfort*. Danish Technical Press, Copenhagen, 1970.

[39] R. De Dear. Thermal comfort in practice. *Indoor Air*, 14(s7):32–39, 2004.

[40] N.A. Taylor, N.K. Allsopp, and D.G. Parkes. Preferred room temperature of young vs aged males: the influence of thermal sensation, thermal comfort, and affect. *Journals of Gerontology Series A: Biological and Medical Sciences*, 50(4):216–221, 1995.

[41] J.F. Nicol and M.A. Humphreys. Adaptive thermal comfort and sustainable thermal standards for buildings. *Energy & Buildings*, 34(6):563–572, 2002.

[42] B. Ivanov, O. Zhelondz, L. Borodulkin, and H. Ruser. Distributed smart sensor system for indoor climate monitoring. In *KONNEX Scientific Conference, München*, 2002.

[43] W.L. Tse and W.L. Chan. A distributed sensor network for measurement of human thermal comfort feelings. *Sensors & Actuators: A. Physical*, 144(2):394–402, 2008.

[44] C.C. Federspiel and H. Asada. User-adaptable comfort control for HVAC systems. In *Journal of Dynamic Systems, Measurement, and Control*, volume 116, pages 474–487, Sep 1994.

[45] A.C. Megri, I.E. Naqa, and F. Haghighat. A Learning Machine Approach for Predicting Thermal Comfort Indices. *International journal of ventilation*, 3(4):363–376, 2005.

[46] RR Gonzalez, Y. Nishi, and AP Gagge. Experimental evaluation of standard effective temperature a new biometeorological index of man's thermal discomfort. *International Journal of Biometeorology*, 18(1):1–15, 1974.

[47] F. Bauman, A. Baughman, G. Carter, and E. Arens. A field study of PEM (Personal Environmental Module) performance in Bank of America's San Francisco office buildings. *Berkeley, Center for Environmental Design Research, University of California. Final report submitted to Johnson Controls World Services, Inc*, 1997.

[48] D. Wang, E. Arens, T. Webster, and M. Shi. How the number and placement of sensors controlling room air distribution systems affect energy use and comfort. In *International Conference for Enhanced Building Operations*, 2002.

[49] G.S. Brager, G. Paliaga, and R. de Dear. Operable windows, personal control, and occupant comfort. *Ashrae Transactions*, 110(2):17–35, 2004.

[50] M.J. O'Neill. Effects of workspace design and environmental control on office workers' perceptions of air quality. In *Human Factors and Ergonomics Society Annual Meeting Proceedings*, pages 890–894. Human Factors and Ergonomics Society, 1992.

[51] "The Jetsons". Hanna-Barbera Productions, 1962–1988. http://www.imdb.com/title/tt0055683/.

[52] M. Weiser, R. Gold, and J.S. Brown. The origins of ubiquitous computing research at PARC in the late 1980s. *IBM Systems Journal*, 38(4):693–696, 1999.

[53] S. Elrod, G. Hall, R. Costanza, M. Dixon, and J. Des Rivieres. The responsive environment: Using ubiquitous computing for office comfort and energy management. Technical report, Technical Report CSL-93-5, Xerox Palo Alto Research Center, 1993.

[54] H. Lee, J.S. Choi, and R. Elmasri. A Conflict Resolution Architecture for the Comfort of Occupants in Intelligent Office. In *Intelligent Environments, 2008 IET 4th International Conference on*, pages 1–8, 2008.

[55] U. Rutishauser, J. Joller, and R. Douglas. Control and learning of ambience by an intelligent building. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 35(1):121–132, 2005.

[56] T.E. Sterk. Building upon Negroponte: a hybridized model of control suitable for responsive architecture. *Automation in Construction*, 14(2):225–232, 2005.

[57] R. Vastamäki, I. Sinkkonen, and C. Leinonen. A behavioural model of temperature controller usage and energy saving. *Personal and Ubiquitous Computing*, 9(4):250–259, 2005.

[58] W.M. Kroner. An intelligent and responsive architecture. *Automation in construction*, 6(5–6):381–393, 1997.

[59] Arch-Os. http://www.arch-os.com/.

[60] P. Anders and M. Phillips. Arch-os: An operating system for buildings. http://www.chrisoshea.org/files/cybrid/rp150.pdf.

[61] LuxLabs, Ltd. dba MeshNetics. ZigBit OEM Modules. Datasheet, Apr 2008. http://www.meshnetics.com/netcat_files/Image/M-251$\tilde{0}$1-(ZigBit OEM Module Product Datasheet).pdf.

[62] Atmel Corporation. ATmega1281 8-bit Microcontroller. Datasheet, Aug 1997. http://www.atmel.com/dyn/resources/prod_documents/doc2549.pdf.

[63] Atmel Corporation. AT86RF230. Datasheet, Feb 2009. http://www.atmel.com/dyn/resources/prod_documents/doc5131.pdf.

[64] Texas Instruments, Inc. TPS780 Series. Datasheet, May 2008. http://focus.ti.com/lit/ds/symlink/tps780270200.pdf.

[65] STMicroelectronics, Inc. M41t series. Datasheet, Aug 2006. http://www.st.com/stonline/products/literature/ds/10397.pdf.

[66] SENSIRION AG. SHT15. Datasheet, Apr 2009. http://www.sensirion.com/en/pdf/product_information/Datasheet-humidity-sensor-SHT1x.pdf.

[67] Intersil Americas Inc. ISL29102. Datasheet, Jul 2008. http://www.intersil.com/data/fn/FN6483.pdf.

[68] Analog Devices, Inc. ADXL330. Datasheet, Sep 2006. http://www.analog.com/static/imported-files/data_sheets/ADXL330.pdf.

[69] Murata Manufacturing Company, Ltd. Piezoelectric Ceramic Sensors (PIEZOTITE). Datasheet, Jun 2002.

[70] Texas Instruments, Inc. OPA2369. Datasheet, Dec 2008. http://www.analog.com/static/imported-files/data_sheets/ADXL330.pdf.

[71] Fairchild Semiconductor Corporation. FDLL300. Datasheet, Mar 2000. http://www.fairchildsemi.com/ds/FD/FDLL300A.pdf.

[72] National Semiconductor Corporation. LMD18200. Datasheet, Apr 2005. http://www.national.com/ds/LM/LMD18200.pdf.

[73] Wintrol Window Motor. http://www.wintrol.com/standardmotorkit.htm.

[74] Belimo Aircontrols, Inc. CM Series Actuators. Datasheet, Jan 2009. http://www.belimo.us/bellib/Damper_Actuators/CM_Actuators.pdf.

[75] Kestrel 1000. http://www.kestrelmeters.com/Kestrel-1000-wind-meter.pro.

[76] Lantronix, Inc. XPort Direct. Datasheet, Dec 2007. http://www.lantronix.com/pdf/XPort-Direct_IG.pdf.

[77] Panasonic Electric Works Co., Ltd. MP Motion Sensor. Datasheet. http://pewa.panasonic.com/pcsd/product/sens/pdf/amn.pdf.

[78] J. G. Ziegler and N. E. Nichols. Optimum settings for automatic controllers. *ASME Transactions*, 64:759–768, 1942.

[79] Calculating the Enthalpy of Air. http://cr4.globalspec.com/thread/3873/Calculating-the-Enthalpy-of-Air.

[80] Y. Kawahara, H. Kurasawa, and H. Morikawa. Recognizing user context using mobile handsets with acceleration sensors. In *IEEE International Conference on Portable Information Devices, 2007. PORTABLE07*, pages 1–5, 2007.

[81] National Weather Service. http://www.weather.gov/climate/index.php?wfo=box.

[82] D.E. Claridge, J.S. Haberl, S. Katipamula, D.L. O'Neal, D. Ruch, L. Chen, T. Heneghan, S. Hinchey, JK Kissock, and J. Wang. Analysis of Texas LoanSTAR Data. In *7th Annual Symposium on Improving Building Systems in Hot and Humid Climates*, pages 9–10, 1990.

[83] D. Westphalen and S. Koszalinski. Energy Consumption Characteristics of Commercial Building HVAC Systems. Volume II: Thermal Distribution, Auxiliary Equipment, and Ventilation, 1999.

[84] JLM Hensen. Literature review on thermal comfort in transient conditions. *Environment*, 25(4):309–316.

[85] M. Malinowski, M. Moskwa, M. Feldmeier, M. Laibowitz, and J.A. Paradiso. CargoNet: a low-cost micropower sensor node exploiting quasi-passive wakeup for adaptive asychronous monitoring of exceptional events. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 145–159. ACM New York, NY, USA, 2007.